SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET U ZAGREBU
Odsjek za informacijske i komunikacijske znanosti

Katedra za društveno-humanističu informatiku
Katedra za arhivistiku i dokumentalistiku
Ak. god. 2016./2017.

Mislav Sever

WEB APLIKACIJA ZA VIZUALIZACIJU I UPRAVLJANJE ARHIVSKIM GRADIVOM

Diplomski rad

Mentor: izv. prof. dr. sc. Hrvoje Stančić
Mentor: dr. sc. Vedran Juričić

Zagreb, rujan 2017.

Sadržaj

1. Uvod ... 6
2. Problematika tradicionalnog korištenja usluga informacijskih ustanova ... 7
2.1. Što su arhivi i arhivsko gradivo ... 7
2.2. Fizička dostupnost arhivskog gradiva i način njegovog korištenja ... 7
3. Arhivske web aplikacije kao alat informacijskih ustanova ... 10
3.1. Korištenje ispravnih medija za pohranu digitalnog gradiva ... 10
3.2. Pitanje pohrane gradiva u oblaku ... 11
3.3. Približavanje gradiva javnosti ... 12
3.4. Web aplikacija kao medij za informiranje o arhivskom gradivu ... 13
4. Django radni okvir i programski jezik Python ... 16
4.1. Python ... 16
4.2. Što je radni okvir ... 18
4.3. Django radni okvir ... 19
4.4. ORM i CMS ... 21
4.4.1. ORM ... 22
4.4.2. CMS – Django Admin ... 22
4.4.3. Sigurnost ... 23
4.5. Linux ... 23
4.5.1. Što je Linux ... 23
4.5.2. Komponente Linux distribucije ... 25
4.5.3. Instalacija softverskih paketa ... 27
5. Pregled aplikacije ArchiPoint ... 29
5.1. Uvod u ArchiPoint i korištena okruženja ... 29
5.2. Upraviteljski servis iza aplikacije ... 31
5.3. Pozadina aplikacije ... 34
5.3.1. Postavke projekta ... 34
5.3.2. Programska pozadina – URL-i, prikazi i iznimke ... 37
5.3.3. Klase i modeli ... 42
5.3.4. Prilagođena polja u modelima ... 44
5.4. Programske knjižnice i druga proširenja radnog okvira ... 49

5.4.1. GeoPy 50
5.4.2. Beautiful Soup 51
5.4.3. Bootstrap 54
5.5. Pristupni dio aplikacije 56
5.5.1. Što je pristupni dio 56
5.5.2. Korištenje Django predložaka 57
5.5.3. Google JavaScript API i generiranje karata 58
5.6. Web poslužitelji i okruženje za objavljivanje aplikacije 61
5.6.1. Što je web poslužitelj 61
5.6.2. Nginx i Gunicorn 62
5.7. Korisničko sučelje aplikacije ArchiPoint 63
5.8. Daljnji razvoj aplikacije 68
6. Zaključak 69
7. Literatura 71
8. Popisi pomoćnih elemenata 73
8.1. Popis slika 73
8.2. Popis kodnih isječaka 74
8.3. Popis tablica 74
9. Prilog – Lokacija objavljene aplikacije ArchiPoint 75

Web aplikacija za upravljanje i vizualizaciju arhivskog gradiva

Sažetak

Cilj ovog rada je prikazati proces nastajanja web aplikacije za upravljanje arhivskim gradivom i njegovu vizualizaciju te objasniti sve njene ključne komponente. Uz to, bit će opisane i popratne tehnologije korištene u stvaranju aplikacije, bez kojih to ne bi bilo moguće. Kao neke od svojih osnovnih funkcija, aplikacija ima mogućnost interaktivnog kronološkog prikaza informacija o lokaciji arhivskog gradiva pomoću vizualnih formi i upravljanje bazom arhivskih metapodataka. Glavna karakteristika ove aplikacije jest njena proširivost i korištenje arhivskih standarda pri upravljanju gradivom, što je čini spremnom za produkcijsku implementaciju uz minimalne izmjene.

Uz detaljan pregled same aplikacije, rad iznosi problematiku tradicionalne uloge arhiva, kao informacijskih institucija, u modernom društvu te mogućnosti za uvođenje noviteta putem implementacije suvremenih tehnoloških rješenja. Kao glavni prijedlog prilagodbe tradicionalnog poslovanja arhiva modernom društvu uzeta je web aplikacija koja je ovdje upisana.

Ključne riječi: web aplikacija, geolociranje, radni okviri, arhivskog gradivo, metapodaci

Web application for management and visualisation of archival materials

Abstract

The goal of this thesis is to visualise the process of developing a web application for managing and visualisation of archival materials and also to bring out its key components. What will also be described are the underlying technologies whose absence would make creating such an application impossible. As one of its basic functions, the application holds the option of creating an interactive chronological view of the archival materials' geographical information and a capability of managing a database of archival metadata. The main characteristic of this application is its expandability and the use of archival standards within the field of material management, which makes it production-ready by adding only little changes to it.

Alongside a detailed description of the application itself, the thesis brings out the question of the traditional role of archives in modern society and the possibilities of introducing novelties through the implementation of contemporary technological solutions. The web application is used as the main suggestion in the adaptation of traditional archival processes to the modern society.

Keywords: web application, geolocation, frameworks, archival materials, metadata

1. Uvod

Eksponencijalnim rastom internetskih tehnologija, smanjuje se potreba za čestim korištenjem fizičkih informacijskih institucija koje pružaju usluge korištenja fizičkog gradiva. Također nedovoljna eksponiranost informacijskih institucija doprinosi svojevrsnoj nepopularnosti istraživanja fizičkog gradiva. Budući da je razvoj tehnologije neizbježan, a također i pozitivan u više aspekata, za očekivati je da je široj javnosti jednostavnije potražiti pristupačne, no nerijetko i nepotvrđene i netočne informacije, na raznim mrežnim mjestima.

Ako se za primjer uzme istraživački seminarski rad jednog prosječnog hrvatskog studenta, često je vidljiv mnogo veći omjer korištenih nepotvrđenih i neslužbenih internetskih izvora naspram legitimnih izvora informacija, kao što su službeni dokumenti, knjižnična građa ili raznoliko arhivsko gradivo koje je dostupno na korištenje.

Uvijek je jednostavnije otvoriti web preglednik, posjetiti Google tražilicu i jednostavno upisati relevantne pojmove kako bi se na licu mjesta dobili željeni odgovori i rezultati. Sa svakodnevnim rastom sadržaja na Internetu, neminovno raste i statistička mogućnost korištenja netočnih informacija u istraživanjima.

Stoga je potrebno dovesti informacijske institucije na prikladna mrežna mjesta gdje bi one bile uočljivije i pristupačnije široj javnosti, što bi omogućilo i njihovo bolje upoznavanje. Prisutnošću jedinstvenog mjesta na Internetu koje bi služilo kao interaktivni katalog, jednako prilagođen za sve uključene informacijske institucije, bilo bi moguće približiti suvremeni princip potrage za informacijama i korištenje vjerodostojnog gradiva.

2. Problematika tradicionalnog korištenja usluga informacijskih ustanova

2.1. Što su arhivi i arhivsko gradivo

Arhiv kao pojam u najmanju je ruku dvojak, iz razloga što on može predstavljati fizički objekt arhivske zgrade i lokaciju, kao što je to neki određeni repozitorij, ili pak ustanovu nadležnu za arhivsku službu na određenom području. Ovaj rad bavit će se razradom i analizom prototipa rješenja za učinkovitije i jednostavnije korištenje usluga oba poimanja arhiva.

Prema opisu ponuđenom od strane Hrvatskog državnog arhiva, „arhivsko su gradivo zapisi i dokumenti nastali djelovanjem pravnih ili fizičkih osoba u obavljanju njihove djelatnosti i od trajne su važnosti za kulturu, povijest i druge znanosti, bez obzira na mjesto i vrijeme njihova nastanka te neovisno o obliku i mediju na kojemu su sačuvani. Zapisi i dokumenti su spisi, isprave, pomoćne uredske i poslovne knjige, kartoteke, karte, nacrti, crteži, plakati, fotografije, slikopisi, pokretne slike (filmovi i videozapisi), zvučni zapisi, mikrooblici te digitalni zapisi i dokumenti.“[1]

2.2. Fizička dostupnost arhivskog gradiva i način njegovog korištenja

Mnogo arhivskog gradiva dostupno je na korištenje široj javnosti. Informacija je to koja je, nažalost, poznata nedovoljno velikom broju ljudi. Arhivi, kao ustanove, često pružaju korisnicima na uvid gradivo koje čuvaju, bilo uz određenu novčanu naknadu ili bez nje.

Zbog vrijednosti određenog gradiva, sam postupak odobravanja gradiva na korištenje i posudbu ponekad je dovoljno zamršen i opsežan da bi određeni broj korisnika odustao od istraživanja informacija u arhivima i odlučio potražiti svoje odgovore na Internetu.

U prilog činjenici da pod arhivsko gradivo spada mnogo vrsta dokumenata, predmeta, biljega i raznih drugih artefakata vrijednih dugotrajnog čuvanja zbog svoje povijesne ili dokazne informacijske vrijednosti, ide i pravilo da je arhivsko gradivo, primjerice, Hrvatskog državnog arhiva moguće koristiti samo u čitaonicama Arhiva.[2] Posudbu gradiva za korištenje izvan prostora Arhiva moguće je dogovoriti samo za određen tip stranaka, o čemu će više riječi biti dalje u radu.

[1]Istražite gradivo. *Hrvatski državni arhiv*. URL: http://www.arhiv.hr/Istrazite-gradivo. (10.9.2017.)
[2] Istražite gradivo. *Hrvatski državni arhiv*. n. dj.

Hrvatski državni arhiv također napominje kako je gradivo obično sređeno prema institucijama čijim je djelovanjem ono i nastalo. Iz tog razloga struktura i struktura samog gradiva odražava način na koji je pojedina institucija organizirala svoje poslovanje i dokumentaciju. Moguće je da zato korisnici arhivskih usluga ne nađu uvijek na jednom mjestu svo gradivo koje im je relevantno za određeno istraživanje.[3]

Prije posudbe i početka korištenja željenog gradiva u istraživanju, korisno je poznavati gdje se točno željeni dokumenti nalaze. Kao što je već poznato, u arhivskoj se struci gradivo može nalaziti u cjelinama kao što su fondovi, serije, zbirke i pojedinačni komadi. Ukoliko određena arhivska ustanova nema razvijen mrežni servis za pronalazak željenog gradiva unutar različitih cjelina, odlazak u različite ustanove i ručna potraga za gradivom pomoću obavijesnih pomagala može biti nepraktična osobama koje na to nisu navikle.

Osim standardnih ograničenja i utvrđenih pravila korištenja regularno dostupnog gradiva, arhivi mogu čuvati gradivo s ograničenom razinom dostupnosti. Takvo gradivo zahtjeva posebne osiguravajuće mjere te uvjete korištenja, koje se odnose na upotrebu u strogo kontroliranim uvjetima ili pak na korištenje od strane samo određenih osoba ili sloja korisnika. Obično to gradivo sačinjavaju razni stari i oštećeni spisi, dokumenti osjetljivih fizičkih karakteristika ili gradivo od izrazite važnosti i vrijednosti.

Sve okolnosti vezane u korištenje javno dostupnog arhivskog gradiva definirane su i navedene u Pravilniku o korištenju arhivskoga gradiva. Pravilnik u svojoj cijelosti dijeli se na sljedeće cjeline:

1. Opće odredbe
2. Dostupnost
3. Postupak
4. Način i uvjeti korištenja arhivskog gradiva u čitaonici
5. Posudba arhivskog gradiva
6. Izrada preslika
7. Obveze korisnika
8. Ovjerovljeni prijepisi i preslici

[3] Istražite gradivo. *Hrvatski državni arhiv*. n. dj.

9. Naknade

Posudbu gradiva za korištenje izvan čitaonica arhiva, moguće je ostvariti u strogo kontroliranim uvjetima i ukoliko samo posudbu zatraži državno tijelo. Također, prije same posudbe, nužno je izraditi presliku posuđenog gradiva na trošak stranke koja posuđuje gradivo. Građani korisnici, kao privatne osobe, nisu u mogućnosti posuditi arhivsko gradivo za korištenje i istraživanje izvan prostora arhiva.[4]

Ukoliko pojedini arhiv ne posjeduje mrežni katalog putem kojeg bi se korisnici arhiva mogli informirati o dostupnosti i detaljima određenog gradiva te ukoliko gradivo arhiva nije uključeno u popis nekog drugog objedinjujućeg mrežnog mjesta, potencijalni korisnici usluga tog arhiva primorani su doći osobno u arhiv kako bi se uopće informirali o gradivu putem obavijesnih arhivskih pomagala.

Razlozi za manjak, takozvanog, web kataloga nekog arhiva mogu biti mnogi, no većinski se svode na manjak resursa ili mogućnosti za razvoj takvog rješenja ili pak njegove kupovine. Kada bi, u teoriji, svaki pojedini arhiv imao svoj zaseban web katalog, korisnici, ukoliko pretražuju informacije o nekim dokumentima kroz više arhiva, morali bi prilagođeno koristiti svaki katalog zasebno. Prisutnost jedne objedinjujuće web lokacije koja bi bila u mogućnosti pretraživati javno dostupno gradivo svih arhiva, ili k tome bar konstantno težiti, značila bi olakšanu pretragu i informiranje o gradivu prije negoli se korisnik fizički uputi u željenu ustanovu kako bi to gradivo i podigao. Pravilnim adresiranjem ranije navedene problematike neobjedinjenosti informacija o gradivu svih pojedinih arhiva u Hrvatskoj i visokom razinom standardizacije, moguće je razviti rješenje koje bi nudilo sve potrebne informacije prikazane na jednak način i na jednom mjestu.

[4] Narodne Novine, Pravilnik o korištenju arhivskoga gradiva. (1999.)

3. Arhivske web aplikacije kao alat informacijskih ustanova

3.1. Korištenje ispravnih medija za pohranu digitalnog gradiva

Uz modernu informacijsku tehnologiju i njenu eksponencijalno rastuću ulogu u svim aspektima ljudskih života, racionalno je za očekivati kako je potrebno ulagati napor i upornost u stalno pronalaženje boljih i održivih načina pohrane digitalnog gradiva. Bitno je razlikovati dvije vrste digitalnog gradiva:

- izvorno nastalo u digitalnome obliku (engl. born digital) – gradivo koje je nastalo kao digitalni dokument i kao takvo postoji od svog samog početka,
- digitalizirano – gradivo koje je nastalo kao fizičko, ali je tijekom svog životnog vijeka digitalizirano na jedan od mnogobrojnih načina.

Usprkos značajne razlike u nastanku ova dva tipa digitalnog arhivskog gradiva, briga oko pronalaska kvalitetnih medija za pohranu i ispravnih principa pohrane u posljetku su vrlo slični. Budući da se većina suvremenog arhivskog gradiva digitalizira, važno je obratiti izrazitu pozornost na medije na kojima će se to gradivo čuvati. Nisu svi digitalni mediji jednako prikladni za pohranu svakog tipa arhivskog gradiva. U interesu je svakog arhiva omogućiti što dugotrajnije očuvanje uz brigu o sljedećim pitanjima:

- standardiziranost medija za pohranu – ukoliko se velika količina arhivskog gradiva pohranjuje na jednom tipu digitalnog medija, bitno je odabrati medij koji je dovoljno standardiziran i prihvaćen za pohranu gradiva,
- mogućnost ispravne obrade i uređivanja digitalnog gradiva – odabrani digitalni medij mora pružati dovoljne mogućnosti kvalitetne obrade pohranjenog digitalnog arhivskog gradiva,
- trajnost medija za pohranu – bitno je da odabrani medij ima što dulji vijek trajanja, bilo u smislu standarda ili fizičke iskoristivosti,
- cijena očuvanja digitalnog gradiva – na kraju svega, bitna je i sama cijene pri odabiru digitalnog medija.

Velik broj digitalnih medija kroz godine je doživio iznenađujuće kratak rok trajanja i predstavljanja standarda u domeni pohrane podataka. Ukoliko se kao referenca uzmu floppy magnetski diskovi, magnetske vrpce, CD-i pa postupno čak i tvrdi diskovi, moguće je vidjeti

kolikom brzinom tehnologija može zastarjeti i izaći iz standarda korištenja. Arhivsko gradivo potrebno je održavati u najboljoj formi po svim standardima arhiviranja i ne dozvoliti da medij na kojem se ono nalazi dotraje. Upravo zato brza mijenjanje tehnologije i ostalih standarda vezanih uz nju predstavlja veliki izazov za arhivsku struku.

3.2. Pitanje pohrane gradiva u oblaku

Jedna od sve češće razmatranih solucija za pohranu digitalnog arhivskog gradiva jest pohrana u oblaku. Oblak je tehnologija korištenja udaljenih mrežnih resursa za pohranu podataka i pristup pohranjenim podacima bez brige od stvarnoj fizičkoj lokaciji medija za pohranu pa tako i lokacije samih podataka. Unazad nekoliko godina, oblak je postao sveprisutan pojam i nezaobilazna tema gotovo svakog razgovora o informacijskoj tehnologiji i pohrani digitalnih podataka.

No, je li pohrana digitalnog arhivskog gradiva u oblaku doista rješenje većine ranije navedenih problema i svojevrsna budućnost digitalne pohrane s arhivskim standardima? Od pojave usluga pohrane podataka u oblaku, arhivska struka postavlja si pitanje jesi li te usluge pravo rješenje za što dugotrajniju pohranu. Kao jedan od glavnih problema lokalne pohrane na odvojenim medijima, navedena je konstantna potreba za migracijom podataka na druge medije, bili oni istog tipa zbog fizičke dotrajalosti prošlog ili potpuno drugačija tehnologija zbog standardne zastarjelosti. Korištenjem pohrane arhivskog gradiva u oblaku, riješio bi se problem migracije s medija na medij zato što bi se o tome brinuo sam pružatelj usluge pohrane. Također, fizička lokacija i manjak prostora postali bi znatno manji problem.

Arhivska pohrana ne uključuje samo pohranu kao takvu, već mnoštvo drugih radnji i postupaka. Arhivske norme jedan su od pratećih elemenata pohrane gradiva zato što neke, kao primjerice ISO 15489, određuju cijeli spektar standardiziranih i obaveznih pravila za upravljanje zapisima. Ukoliko se pri pohrana gradiva kao „medij“ odabere oblak, od izuzetne je važnosti osigurati da svi standardi arhivske pohrane i uređivanja gradiva budu zadovoljeni, zato što je arhivska pohrana mnogo više od samog spremanja digitalnih podataka na medij.[5]

[5] Stančić, H., Je li pohrana podataka u oblaku pravo rješenje za arhivsku struku?. *Netokracija*. URL: http://www.netokracija.com/arhivska-pohrana-oblak-sector-118085. (5.5.2016.).

3.3. Približavanje gradiva javnosti

Jedan od ciljeva arhivske struke također mora biti i približavanje raspoloživog i iskoristivog arhivskog gradiva javnosti. Potrebno je ukazati na postojanje gradiva kao relevantnog izvora informacija u zajednici i ukloniti ponekad smatranu karakteristiku svrsishodnosti s arhivskog čuvanja i uređivanja gradiva.

Arhivsko gradivo nije svrha samom sebi već se u njegovu dugotrajno očuvanje i uređenje ulažu napor i vrijeme kako bi ostalo postojano kroz vrijeme s razlogom. Pri svakoj ideji čuvanja nekih resursa, nameće se pitanje hoće li se ti resursi koristiti u budućnosti te da li je njihovo čuvanje uistinu potrebno. Upravo iz tog razloga postoji arhivska struka i sam čin očuvanja relevantnog gradiva. Nije svaki nastali dokument potencijalni arhivski spis već postoji područje djelovanja arhivske struke koje brine i odlučuje o relevantnosti određenog gradiva i njegovoj vrijednosti očuvanja. Ukoliko se neko gradivo odluči očuvati, razlog je njegova potencijalna važnost i uloga informiranja u budućem vremenu.

Nije sve arhivsko gradivo dostupno javnosti na korištenje te je važno naglasiti kako se neka dokumentacija nastala u javnim institucijama odlučuje čuvati zbog internih procesa te institucije ili kao dokazni materijal u poslovanju. Takvo gradivo ne mora biti široko dostupno, već služi kao formalno sredstvo informiranja i potvrde u određenim administrativnim razinama.

Gradivo koje jest dostupno javnosti na korištenje, bilo to svojim općim povijesnim značajem ili jednostavno zbog već dovoljno minulog vremena, potrebno je eksponirati zajednici na ispravan način. Vijeće australazijskih arhiva i dokumentacijskih tijela navodi kako kulturne institucije nisu odgovorne samo za akumuliranje i održavanje svojih jedinstvenih kolekcija, već im je dužnost i povećanje dostupnosti tih kolekcija prema javnosti. Kako bi se to postiglo, potrebna je odgovarajuća infrastruktura pristupa gradivu te stoga potencijalni korisnici iziskuju:

- mogućnost pristupa i interpretacije digitalnih kolekcija,
- mrežna korisnička sučelja prema digitalnim kolekcijama koja su jednostavna za korištenje,
- sustave brzog i učinkovitog pristupa željenom digitalnom gradivu,

- kontrolirane mehanizme pristupa, kako bi se očuvala privatnost, sigurnost i razina prava na korištenje gradiva.[6]

3.4. Web aplikacija kao medij za informiranje o arhivskom gradivu

Kao jedan od najučinkovitijih načina za prikaz arhivskog gradiva raspoloživog za korištenje široj javnosti, u ovom radu uzeto je korištenje web aplikacija. Pojam web aplikacije obuhvaća širok spektar softverskih rješenja dostupnih na mreži ili Internetu koji se koriste u određenu svrhu. Kako bi se smanjilo područje obuhvaćanja značenja web servisa, naglasit će se kako se u ovom radu referira na web aplikacije s grafičkim korisničkim sučeljima koje služe kao katalog dostupnih resursa.

Vijeće australazijskih arhiva i dokumentacijskih tijela navodi problem vidljivosti i eksponiranosti digitalnih kolekcija isključivo putem web stranica individualnih ustanova koje to gradivo čuvaju te navodi kako je bitno omogućiti pronalazak tog gradiva putem internetskih pretraživača. U takvom okruženju, korisnici bi, pretražujući neki pojam putem generalnog internetskog pretraživača, kao relevantne rezultate također dobili i arhivsko gradivo različitih informacijskih ustanova[7].

Jedan od načina za postizanje navedenih rezultata upravo je korištenje javno dostupne web aplikacije čiji bi sadržaj bio dostupan na indeksiranje često korištenim pretraživačima. Takva aplikacija bila bi u mogućnosti posluživati korisnike detaljnim informacijama o svakom željenom i dostupnom arhivskom gradivu, bilo ono digitalno ili fizičko. U slučaju digitalnog gradiva, raspoložive zbirke i dokumenti mogu se u svojoj cijelosti prenijeti putem Interneta te preuzeti na korisničko računalo na korištenje, dok bi fizičko gradivo bi potrebno preuzeti u njegovoj matičnoj ustanovi ili na određenoj centralnoj lokaciji preuzimanja svog arhivskog gradiva uključenog u projekt.

Bez obzira na način preuzimanja i korištenja određenog tipa gradiva, ključan faktor u postizanju željenog cilja jest prikaz detaljnih informacija svih tipova gradiva na jednom mjestu i

[6] Council of Australasian Archives and Records Authorities. *Digital archiving in the 21st century*. URL: http://www.caara.org.au/wp-content/uploads/2010/03/DigitalArchiving21C.pdf. (9.2006.)

[7] Council of Australasian Archives and Records Authorities. *Digital archiving in the 21st century*. n. dj.

na jedan unificiran način. U toj namjeri, glavnu ulogu imaju digitalni metapodaci o gradivu. Kao što je već široko poznato, metapodaci su detaljni opisni podaci o nekom drugom objektu, bio on dokument, muzejski eksponat ili čak nešto nevezano uz informacijske institucije kao takve. Popularna definicija metapodataka govori kako su metapodaci podaci o podacima. Takva definicija je, bar donekle, površna, jednostavna i nedorečena, tako da će se u ovom radu pokušati na primjerima iz prakse objasniti moguća forma i uloga metapodataka u populariziranju korištenja arhivskog gradiva.

Ideja ovog rada jest osmisliti i stvoriti web aplikaciju koja bi služila kao svojevrsna dodirna točka svih informacijskih ustanova, sadržavajući detaljne informacije o svakom gradivu dostupnom na korištenje do određene mjere. Tako nešto postiglo bi se korištenjem jedinstvene sheme i formata navođenja i označavanja digitalnih metapodataka o svim mogućim tipovima gradiva. Tada bi svaka institucija koja se želi uključiti u projekt, mogla bez pretjerane prilagodbe predati detaljne i standardizirane metapodatke o svom gradivu navedenoj web aplikaciji. Zbog korištenja neke određene sheme, kao što je to EAD arhivski standard u XML označavanju, web aplikacija bila bi već spremna i prilagođena za prihvat informacija od strane svake prilagođene ustanove.

Najučinkovitiji i najiskoristiviji izbor formata digitalnih metapodataka u procesu standardizacije jedne takve internetske usluge, jest jedan od popularnih jezika za označavanje i notaciju podataka, kao što su to XML ili JSON. Korištenjem jednog jezika za označavanje te jedinstvenog spektra elemenata oznake, navedena web aplikacija uvijek bi bila u mogućnosti ispravno prikazati informacije o gradivu bilo koje institucije koja informacije o svom gradivu predaje na prikaz.

Odabrani format prijenosa i pohrane metapodataka za aplikaciju ArchiPoint jest XML kod pisan u EAD standardu. U službi XML-a, EAD predstavlja specifični imenski prostor (engl. namespace) koji određuje cijeli spektar prilagođenih XML elemenata i njihove strukture bilježenja. Ono što je svakoj informacijskoj ustanovi, zainteresiranoj za prikaz njenog gradiva na jednoj takvoj centralnoj web platformi, nužno osigurati jest da njeni metapodaci za sadržano gradivo prate EAD standard označavanja u XML-u. Ako je navedeni uvjet zadovoljen, prijenos i integracija u aplikaciju kao što je ArchiPoint postaju vrlo jednostavni procesi koji zahtijevaju minimalno prilagodbe i dorade.

XML, kao metoda označavanja podataka, široko je priznat i korišten standard. Pri izboru jedne takve metode za označavanje te bilo koje druge tehnologije i paradigme za uspostavu web aplikacije koja za svoj cilj ima služiti kao centralna lokacija prikaza informacija o gradivu, izrazito je bitno voditi računa o svjetskim tehnološkim standardima. Sustavi za upravljanje i korištenje digitalnih informacija trebali bi koristiti otvorene standarde kako bi osigurali svoju održivost kroz vrijeme. Standardizacija se ne odnosi samo na korištenje određenog sustava označavanja podataka, već i na formate digitalnog gradiva i sve ostale potencijalne točke razilaženja s drugim sustavima.

U konačnici, jedan takav sustav mogao bi doprinijeti, osim većoj popularizaciji korištenja arhivskog gradiva u javnosti, i povećanju kvalitete pisanja znanstvenih radova kroz vrijeme. Jasno je kako se najrelevantnije i najvjerodostojnije informacije za korištenje u znanstvenim istraživanjima nalaze u repozitorijima i spremištima informacijskih ustanova, kao što su to arhivi. No, usprkos tome, u današnje vrijeme sveprisutne dostupnosti internetskih pretraživača i raznolikih portala, često je mnogo jednostavnije potražiti odgovore na Internetu, koliko god oni možda biti netočni ili nedosljedni. Ideja iza aplikacije ArchiPoint, koja će biti detaljno opisana u daljnjem tekstu, jest upravo pružanje potrebnih informacija o vjerodostojnom gradivu na jednako jednostavan i izravan način.

4. Django radni okvir i programski jezik Python

4.1. Python

Python je programski jezik visoke razine (engl. high-level) kojeg je 1991. godine razvio nizozemski razvojni programer Guido van Rossum. Svojom popularnošću, mogućnostima implementacije, prilagodljivošću te intuitivnošću, Python je kroz godine od svog nastanka zauzeo ulogu jednog od najkorištenijih i najsvestranijih programskih jezika u svijetu informacijske tehnologije.

Svojom sintaksom i stilom pisanja, Python se djelomično razlikuje od velikog dijela postojećih modernih programskih jezika. Navedeno svakako ne znači da se Python više ne tretira kao jedan suvremeni programski jezik, već da mu se stil pisanja razlikuje od onog drugih objektno orijentiranih jezika, kao što su Java, PHP ili JavaScript. Sam po sebi, Python pripada skupini programskih jezika treće generacije, no svojom iznimnom proširivošću i mogućnošću korištenja mnogobrojnih knjižnica (engl. libraries) i modula koji mu mogu pridodati brojne različite uloge i područja primjene, on također dijelom ulazi u skupinu jezika četvrte generacije. Upravo zbog tih mogućnosti proširenja, Python može imati mnogo različitih funkcionalnosti i služiti kao vrlo funkcionalan alat.

Primarno osmišljen kao nasljednik jezika ABC, Python je stvoren za korištenje na Unix platformama kao skriptni jezik za automatizaciju procesa i jednostavniju administraciju. Dobivanjem na popularnosti, Python je s vremenom prerastao u višenamjenski jezik (engl. multi-purpose language) te zajednica korisnika za njega konstantno stvara nove module i proširenja, koja mu omogućavaju poprimanje novih svojstava. Jednako tako je, s pojavom web radnih okvira, Python također postao i objektno orijentirani jezik za razvoj web servisa.

Jedna od specifičnosti Pythona je njegova već spomenuta sintaksa, koja se, za razliku od jezika kao što su Java i C#, ne oslanja na korištenje vitičastih zagrada i oznaka za kraj izraza, već uvelike koristi vizualno uvlačenje dijelova koda kako bi određeni izrazi činili cjelinu. To znači da je u Pythonu obično od iznimne važnosti vizualna uvučenost dijelova koda, to jest koliko je puta korišten tabulator, dok Java odjeljuje izraze zagradama, a sami po sebi mogu biti napisani u istom redu. Još jedna posebnost Pythona naspram većine suvremenih objektno orijentiranih jezika je njegova fleksibilnost pri definiranju varijabli i funkcija, pri čemu ne zahtijeva korištenje

oznaka tipa i pristupa (*Integer*, *public*, *static*, *var*, …) koje unaprijed definiraju korišteni tip podataka, vrijednosti koju neka funkcija / metoda vraća ili razinu pristupa nekoj metodi / funkciji. Primjerice, ukoliko u Pythonu neki, nama idejno, nevezani izraz ostavimo uvučen na razini *for* petlje, on će se izvršavati unutar te petlje kao jedna od njenih akcija.

```python
from random import randint
from datetime import datetime


def generate_unique_id (count):
    identifier = str(randint(0, 1000)) + str(count) + str(datetime.now())


    return indentifier
```

Kod 1. Python funkcija za generiranje jedinstvenog identifikatora

```java
public static String generateUniqueId (Integer count){
    Random random = new Random();
    String identifier = random.nextInt(1000).toString() + count.toString + System.currentTimeMillis().toString;


    return (identifier);
}
```

Kod 2. Java metoda za generiranje jedinstvenog identifikatora

Usprkos Pythonovoj mogućnosti da bude korišten kao jezik za logičku pozadinu web aplikacija i servisa, on i dalje slovi kao jedan od najkorištenijih jezika za skriptiranje i automatizaciju poslužiteljskih procesa, stoga mu je ta uloga i dalje najznačajnija. Mnogi poznati i često korišteni web servisi navode upravo Python kao svoj alat za upravljanje zahtjevnim zadaćama na svojim poslužiteljima. Tako Cuong Do, softverski arhitekt iz YouTubea, govori kako je „Python dovoljno brz za njihovu stranicu (YouTube) i dozvoljava im stvaranje održivih

svojstava u rekordnom vremenu i uz minimalno programera.“[8] Još jedan od zanimljivih citata koji pobliže dočarava Pythonovu implementaciju na području skriptiranja jest izjava Hilmara Veigara Peturssona, direktora tvrtke CCP Games, u kojoj kaže da im je Python omogućio da stvore igru EVE Online, masivnu igru za mnogo igrača, u rekordnom vremenu. Naveo je kako EVE Online poslužiteljski klaster istovremeno upravlja s preko 50.000 igrača u dijeljenoj igri koja je simulacija svemira. Većina navedenog stvorena je pomoću Pythona upravo zbog njegove fleksibilnosti i skriptnih mogućnosti.[9]

4.2. Što je radni okvir

Kako bi uopće bilo moguće pričati o Djangu i Pythonu, potrebno je prvenstveno objasniti što je točno radni okvir za neki programski jezik. Radni okvir označava svojevrsno proširenje neke tehnologije u svrhu učinkovitijeg obavljanja neke postojeće funkcije i zadaće ili pak stvaranje nove primjene za tu istu tehnologiju. U kontekstu programskih jezika, radni okviri su najčešće neizmjenjiva proširenja koda koja uz sebe često donose i razne druge alate koji programskom jeziku omogućavaju obavljanje različitih novih funkcija i rad u novim okruženjima. Oni su najčešće nalik ekstenzivnim programskim knjižnicama (engl. library) – proširivi no ne izmjenjivi.

Web radni okviri programskim jezicima najčešće, uz proširenja funkcionalnosti samog koda, donose i druge alate koji olakšavaju razvoj web i lokalnih poslužiteljskih aplikacija, kao što su podesivi razvojni web poslužitelji (engl. development web server), poseban prevodioc (engl. compiler) i tumač (engl. interpreter) programskog koda, poseban jezik za web predloške (engl. templating language), prilagođen odnos logičke pozadine i vidljivog dijela svake aplikacije te mogućnost korištenja raznih programibilnih sučelja (engl. application programming interface, API).

Budući da je razlika između radnih okvira i API-a jasna većini razvojnih programera no ujedno i čest „kamen spoticanja“ za većinu ostalih ljudi, korisno je napomenuti kako API-ji najčešće sadržavaju samo određene prilagodljive akcije koje je moguće izvršiti nad njima kako bismo dobili željene rezultate na licu mjesta i uz što manje napora i razvoja. Takvi su i neki od

[8] Do, C., Quotes about Python, *Python.org*, URL: https://www.python.org. (2.09.2017.)
[9] Prema: Petursson, H. V., Quotes about Python, n. dj.

poznatiji API-ja, kao što su Instagram API koji omogućava pozivanje različitih automatiziranih akcija nad Instagram servisom i Googleov Maps API koji vraća određene vrijednost inicirane pozivima od strane naše aplikacije. O samom Googleovom Maps API-u će biti riječi kasnije u radu.

Neki od popularnijih web radnih okvira su:

- Ruby on Rails – radni okvir za programski jezik Ruby,
- Laravel – radni okvir za programski jezik PHP,
- ASP.NET – radni okvir za programski jezik C# i Microsoft tehnologije,
- Django – radni okvir za programski jezik Python,
- Grails – radni okvir za programski jezik Groovy i tehnologije JVM (engl. Java virtual machine) platforme,
- Bootstrap – radni okvir za jezike pristupnog dijela weba: HTML, CSS (engl. cascading style sheets) i JavaScript te također i omogućuje korištenje drugih JavaScript radnih okvira.

4.3. Django radni okvir

Django je najpopularniji i najeksponiraniji web radni okvir za programski jezik Python. Pritom, valja napomenuti da, kako za ostale programske jezike i tehnologije tako i za Python, postoji više radnih okvira koje je moguće slobodno koristiti. Kao samo neke od njih, moguće je navesti također često korištene Flask ili TurboGears te neke manje popularne kao što su Pyramid ili web2py[10]. Pod izrazom „slobodni za koristiti“ referira se na alate otvorenog koda ili, u najmanju ruku, alate koji su za korištenje dostupni besplatno ili bar uz opcionalne donacije.

U opisanoj skupini alata, Django kao radni okvir prednjači svojom jednostavnošću za korištenje i implementaciju, no ujedno i izrazitom fleksibilnošću i prilagodljivošću. Kao takav, Django podržava jednostavno i intuitivno pokretanje web projekta, integraciju s često korištenim i suvremenim web tehnologijama, kao što su Bootstrap, jQuery i AJAX, te podršku za više različitih vrsta baza podataka kao okosnicu servisa. Uzimajući navedeno u obzir, bitno je navesti

[10] Web Frameworks for Python, 11.08.2017., *Python.org*, URL: https://wiki.python.org/moin/WebFrameworks. (4.09.2017.)

kako je krivulja učenja korištenja Djanga specifična iz razloga što je potrebno razmjerno malo vremena za napraviti vrlo bazičnu web aplikaciju naspram vremena koje je potrebno uložiti kako bi se ovladalo mnogim naprednim mogućnostima koje on u konačnici pruža.

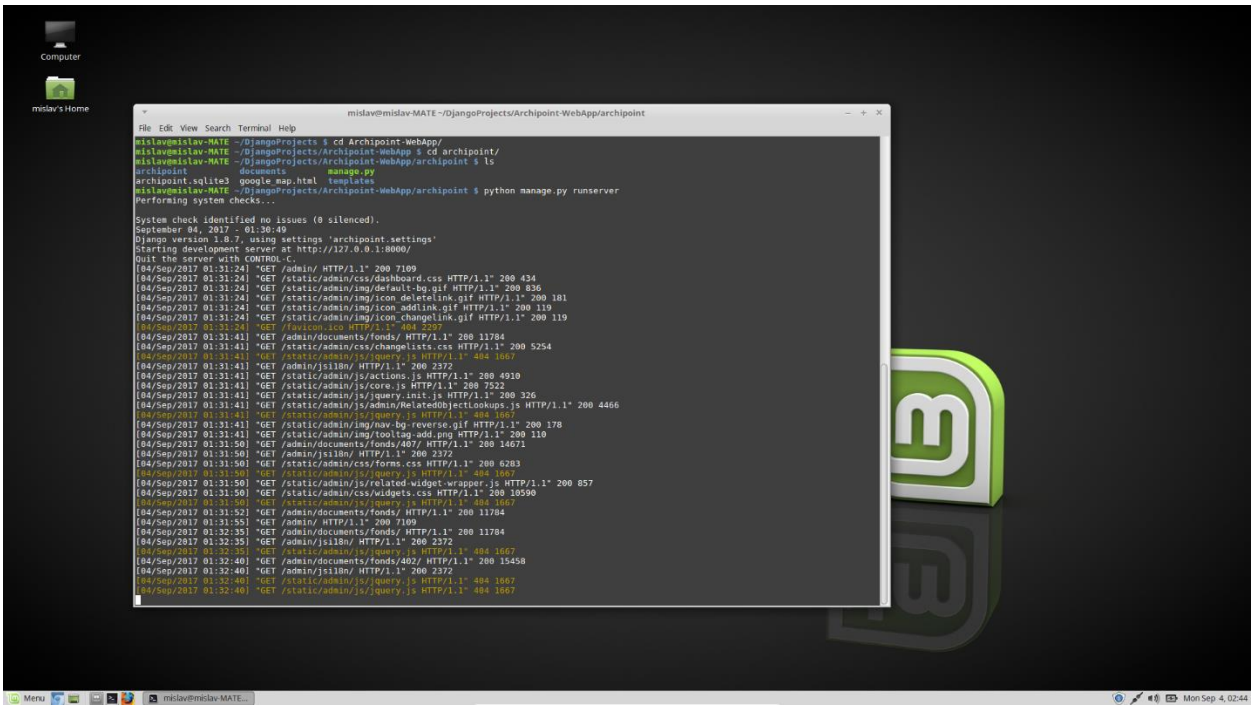
Tipovi standardno podržanih baza podataka i sustava za upravljanje bazama podataka u Djangu su:

- SQLite,
- PostgreSQL,
- MySQL,
- Oracle

Ostali tipovi baza podataka i API-a za pristup bazama podataka koji su u Django podržani djelomično ili uz potrebne značajnije modifikacije su:

- Microsoft SQL Server
- IBM DB2
- SAP SQL Anywhere
- Firebird
- ODBC (engl. Open database connectivity)

Ukoliko se koristi neka od ovdje navedenih djelomično podržanih baza podataka, nije moguće u potpunosti računati na sve Djangove ORM funkcionalnosti, budući da radni okvir nije napravljen s podrškom za te baze na umu, te je moguće da neki od Djangovih transakcijskih metoda na bazi neće biti izvršene u potpunosti ili ispravno.



Slika 1. Pokretanje Django razvojnog poslužitelja putem Terminala

Gledano iz aspekta programskog jezika, Django je moguće tretirati kao svojevrsnu knjižnicu (engl. library) ili modul za programski jezik Python, koji mu omogućava mnoge funkcionalnosti za razvoj web baziranih aplikacija i servisa. Dakle, Django radni okvir pretvara Python u punopravni objektno orijentirani programski jezik koji tada može predstavljati logičku pozadinu jednog web ili poslužiteljskog servisa. Ukoliko gledamo na Django kao na sam programski kod, to je proširivi paket Python koda koji sadržava različite iskoristive modele koje je moguće pozvati kroz Python i raditi u njima. U konačnici, Django i jest web radni okvir napisan u jeziku Python upravo za jezik Python.

4.4. ORM i CMS

Posebnost Djanga kao web radnog okvira nad određenim brojem drugih radnih okvira za ostale objektno orijentirane programske jezike jesu njegova mogućnost efektivnog i intuitivnog objektno relacijskog mapiranja (engl. object-relational mapping, ORM), funkcionalno prevođenje Python modela / klasa u tablice unutar relacijskih baza podataka, jednostavnu i uključenu integraciju SQLite baze podataka, integrirani i programibilni sustav za upravljanje

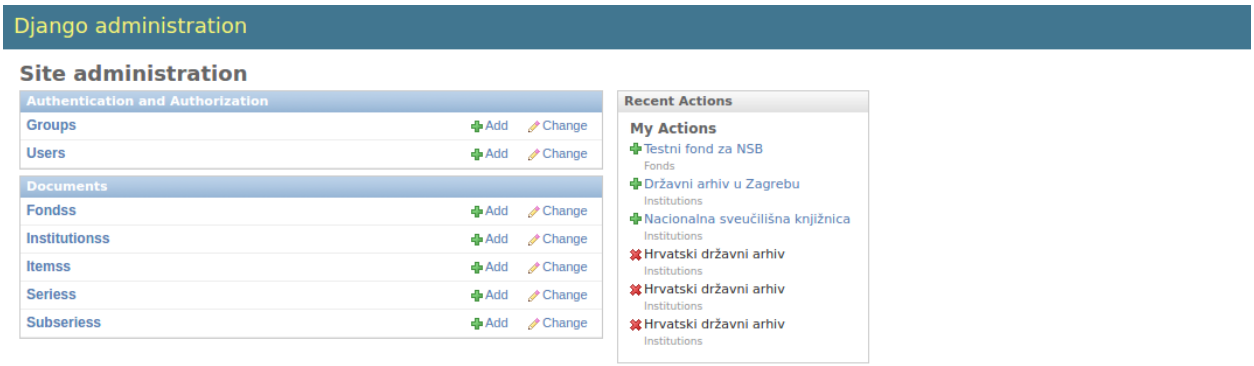
sadržajem (engl. content management system, CMS) pod imenom Django Admin, uključen razvojni web poslužitelj koji je jednostavan za korištenje te gotovu podršku za više jezika za stvaranje predložaka (engl. templating language).

4.4.1. ORM

Za bolje razumijevanje web razvoja i objektno orijentirane paradigme programiranja, potrebno je razumjeti koncept klasa. U navedenom kontekstu, klasa predstavlja svojevrsni kalup ili predložak za kreiranje i korištenje objekata. Klase definiraju svojstva i mogućnosti svih objekata napravljenih iz određene klase. Imajući na umu navedeno lakše je shvatiti korištenje klasa i u Djangu, koje se apstraktno nazivaju modelima. Svaki model može sadržavati varijable određenog podatkovnog tipa te definirane funkcije koje njegovi objekti mogu pozivati i izvršavati. S obzirom da Django svaki definirani model tretira kao jednu tablicu u relacijskoj bazi podataka, definirane varijable se analogno manifestiraju kao stupci u tablici podatkovnog tip jednakog podatkovnom tipu varijable. Navedene odnose Django uspostavlja takozvanim automatskim SQL (engl. structured query language) migracijama i ranije spomenutim objektno relacijskim mapiranjem. Bez potrebe za ručnim pisanjem SQL upita prema bazama podataka, Django omogućuje izvršavanje upita nad bazom podataka pomoću specifičnih Python funkcija i argumenata.

4.4.2. CMS – Django Admin

Django Admin jedan je od značajnih modula cjelokupnog Django radnog okvira. To je web sučelje koje djeluje kao sustav za upravljanje sadržajem (engl. content management system, CMS) u aplikacijama te, kao takvo, ima mogućnost upravljanja bazom podataka. S obzirom da relacijska baza podataka jedne aplikacije sadrži sve ključne informacija o njoj, preko Django Admina moguće je upravljati sadržajem tablica (koje predstavljaju modele) i dodavati i uređivati korisnike aplikacije. Prikaz pojedinih modela u Adminu prilagodiv je i kroz sam kod, tako da je ovo sučelje i fleksibilno u podešavanju prema potrebama administratora aplikacije.



Slika 2. Django Admin

4.4.3. Sigurnost

Upravo takva navedena paradigma (ORM) korištenja baza podataka i automatske migracije omogućuju bolju zaštitu web aplikaciju od napada kao što je ubrizgavanje SQL koda (engl. SQL injection). Iako tehnički podržani, u Djangu se standardno ne pišu SQL izrazi unutar koda aplikacije, pa stoga ne postoji direktna mogućnost za izvršavanje jednog takvog malicioznog napada.

4.5. Linux

4.5.1. Što je Linux

Linux, kao pojam, označava porodicu operacijskih sustava (engl. operation system family, OS family) koja, među svojim pripadnicima, kao poveznicu sadrži Linux jezgru (engl. kernel). Jezgra sustava njegova je najniža softverska ustrojstvena jedinica te, kao takva, odrađuje komunikaciju sa računalnim komponentama na niskoj programskoj razini (engl. low-level) i predstavlja logički komunikacijski sloj između računalnih komponenata i softverskih aplikacija koje se na tom računalu pokreću i koriste.

Linux jezgra napravljena je od strane Linusa Torvaldsa 1991. godine djelomično po uzoru na Unix operacijski sustav u sklopu studentskog projekta. Uz samu sličnost jezgre,

Torvalds je Linux jezgri prilagodio i već postojeći Bash (engl. Bourne again shell) i GNU (engl. Gnu's not Unix) prevodioc te ih postavio kao integrirani dio Linuxa. Upravo zbog navedenih sličnosti, Linux distribucije nazivaju se sustavima sličnim Unixu (engl. Unix-like systems) ili se pak navode pod kraticom *NIX sustava.

Takozvane Linux distribucije međusobno se mogu, naoko, znatno razlikovati, no sve ih povezuje Linux jezgra. Prve kreirane Linux distribucije, početkom devedesetih godina prošlog stoljeća, bile su Fedora, Debian i Slackware. Uz rijetka odstupanja od povezanog grananja kroz vrijeme, gotovo sve trenutno prisutne distribucije svojevrsni su derivat (engl. fork) jedne od te tri inicijalne.

Prema Linux portalu DistroWatch.com, sljedećih deset distribucija su u protekloj godini bile najpopularnije:[11]

- Mint – baziran na Ubuntu / Debian,
- Debian,
- Manjaro – baziran na Arch,
- Ubuntu – baziran na Debian,
- openSUSE – nezavisna distribucija,
- Antergos – baziran na Arch,
- Fedora,
- Zorin – baziran na Ubuntu / Debian,
- elementary – baziran na Ubuntu / Debian,
- deepin – baziran na Debian.

Proučavanjem ove liste deset najpopularnijih i najkorištenijih Linux distribucija u zadnjih dvanaest mjeseci, može se zaključiti kako među njima prednjače derivati Debian originalne distribucije. Fedora i njene izvedenice, kao što su RHEL (engl. Red Hat Enterprise Linux) i CentOS, uglavnom se koriste kao platforma za poslužiteljska računala. Uzimajući u obzir karakteristike Debian sustava i njegova područja primjene, a posebice njegovih derivata, vidljivo je kako Linux postaje sve popularniji izbor operacijskog sustava za osobna računala, čemu doprinosi i popularizacija desktop okruženja suvremenog izgleda i funkcionalnosti.

[11] Bodnar, L.; et al, *DistroWatch.com,* URL: https://distrowatch.com/index.php. (5.09.2017.)

4.5.2. Komponente Linux distribucije

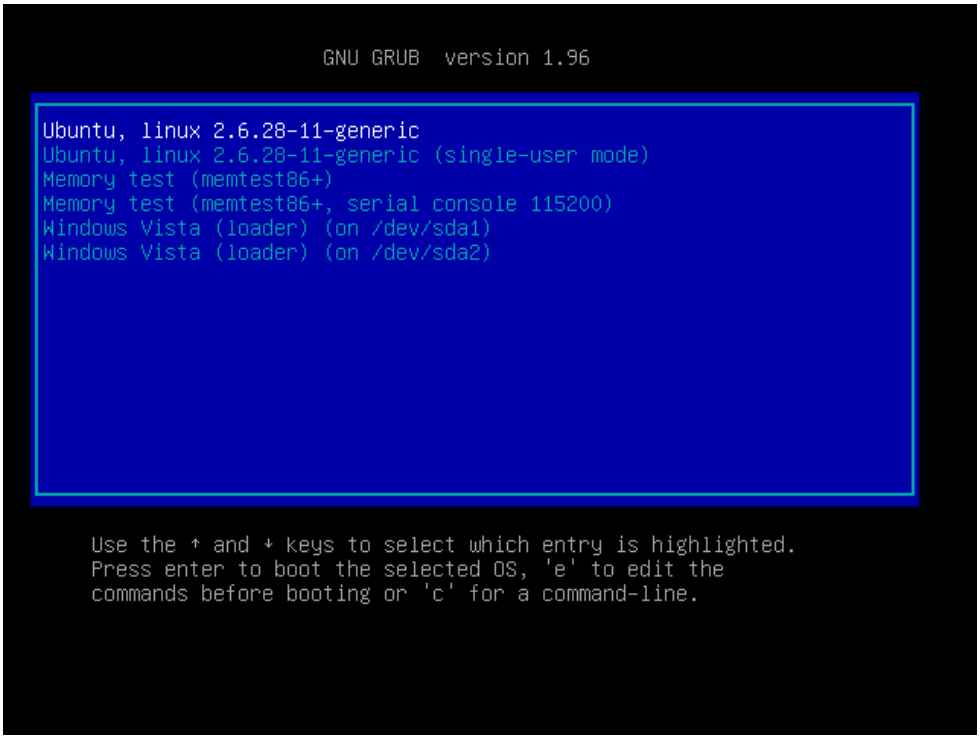
Linux.com, web portal The Linux Foundationa, navodi sljedeće softverske komponente kao sastavne dijelove jedne Linux distribucije:[12]

- Pokretač (engl. bootloader) – softver koji upravlja procesom pokretanja operacijskih sustava prisutnih na računalu. Ukoliko na određenom računalu postoji više operacijskih sustava, pokretač će, kao jednu od svojih mogućnosti, ponuditi izbor operacijskog sustava za pokretanje. Ukoliko je odabran operacijski sustav koji nije Linux, Linux pokretač prosljeđuje daljnje iniciranje pokretaču odabranog sustava, primjerice Windows pokretaču – NTLDR (engl. NT loader). Primjeri najkorištenijih Linux pokretača su GRUB (engl. Grand Unified Bootloader) i LILO (engl. Linux Loader), od kojih je GRUB noviji i trenutno najkorišteniji.
- Jezgra (engl. kernel) - već spomenuti softverski sloj koji upravlja hardverskim komponentama i njihovom komunikacijom sa softverskim aplikacijama. Jezgra je glavna poveznica među svim Linux distribucijama te je, u srži, upravo ono što se naziva Linuxom.
- Daemoni (engl. daemons) – programi koji se sami pokreću i izvršavaju u pozadini bez potrebe ljudske interakcije i djelovanja. Donekle ih se može povezati sa pojmom servisa, no u mnogo slučajeva daemoni pokreću i pružaju servise, tako da takva poveznica nije uvijek ispravna.
- Ljuska (engl. shell) – sučelje koje radi kao tekstualni procesor naredaba. Linux ljuska (Bash, Ksh, Fish,…) ono je što korisnicima, ukoliko su dovoljno upoznati, kroz neku od aplikacija za kontrolu (Terminal, Konsole, XTerm, Sakura, …) omogućuje izrazito visoku razinu kontrole nad sustavom i njegovim operacijama putem skriptnih jezika. Moderne inačice Linux sustava, zajedno s njihovim grafičkim sučeljima, od prosječnog korisnika rijetko zahtijevaju interakciju s jezgrom.
- Grafički poslužitelj (engl. graphical server) – podsustav zadužen za prikaz grafike na monitoru računala. S obzirom da su prozori (engl. windows) također i na Linux

[12] The Linux Foundation, What is Linux?, *Linux.com*, URL: https://www.linux.com/what-is-linux. (3.09.2017.)

sustavima glavni oblik prikaza lokacija i zadaća u računalu, sustav se naziva X Window System ili jednostavno X.

- Desktop okruženje (engl. desktop environment) – naziv za objedinjeno radno grafičko sučelje koje se sastoji od više softverskih komponenata (upravljači datoteka, konfiguracijski alati, alatne trake, web preglednici, …). Neki od poznatijih desktop okruženja su Cinnamon, Unity, KDE, GNOME i MATE.
- Aplikacije – svaka Linux distribucija posjeduje mogućnost instalacije mnoštva specifičnih ili aplikacija dijeljenih s drugim distribucijama. Putem grafičkog desktop sučelja softverski upravljači i repozitoriji pojedinih distribucija najjednostavniji su način za instalaciju željenih aplikacija.



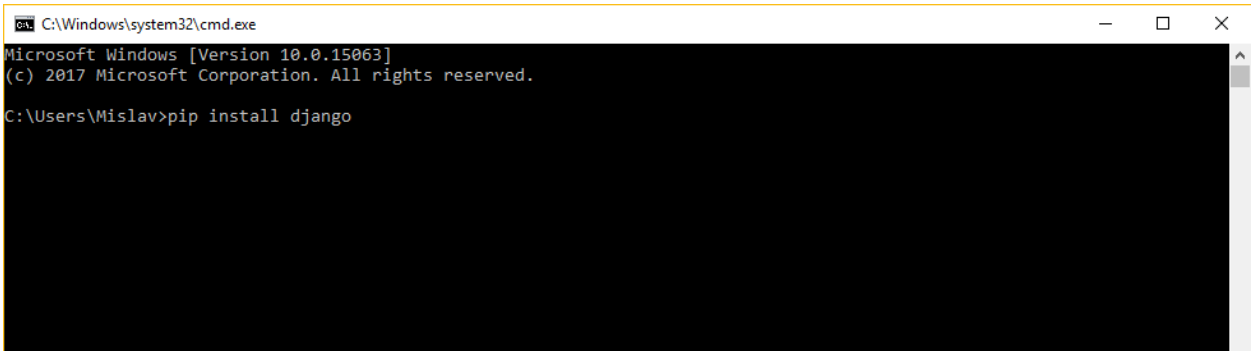
Slika 3. GRUB pokretač

Izvor: Ljubunčić, I. GRUB 2 bootloader – full tutorial, *Dedoimedo*, URL: https://www.dedoimedo.com/computers/grub-2.html. (3.09.2017.)

4.5.3. Instalacija softverskih paketa

Uz sve dosad navedeno, još jedan od potencijalnih razloga u korist Djanga i Pythona kao alata za kreiranje web aplikacija i servisa jest njihova dostupnost bez ikakve obavezne novčane naknade. Predviđeno, to jest, „prirodno“ (engl. native) okruženje za korištenje jezika Python upravo su Linux operacijski sustavi, u kojima on dolazi kao već inicijalno instalirani podatkovni paket i u kojima ga je moguće koristiti jednostavnije i integriranije nego na operacijskim sustavima Windows. S obzirom da je Django proširenje jezika Python, Linux je također „prirodno“ okruženje i za Django aplikacije. Naravno, moguće je instalirati i postaviti Python i Django i na Windows sustavima no to zahtijeva dodatno konfiguriranje određenih postavki sustava za jednostavnije i logičnije upravljanje i korištenje.

Instalacija Python softverskih paketa i knjižnica na Linuxu je obično vrlo izravna i jednostavna. Većina web izvora s uputama navest će kako određena Python knjižnica postoji na odabranom udaljenom Linux repozitoriju, ta da je za instalaciju i integraciju u Python i sam sustav dovoljno dohvatiti je i instalirati kroz konzolu i jezgru putem naredbe ili kroz grafičko sučelje putem upravljača softverom. Tako instaliran Python paket, dostupan je odmah globalno iz razloga što je Python standardan dio Linux sustava. Naravno, moguće je posjetiti web lokaciju željenog softverskog paketa, preuzeti ga, raspakirati i izgraditi ručno putem jezgre, no to je često poprilično kompliciran proces za kojim, uz upravljače softverom, nema pretjerane potrebe. Kako na Linux, tako i na drugim operacijskim sustavima (Windows, OSX, …) moguće je instalirati i koristiti pip softverski paket (engl. „Pip installs packages“) koji se koristi za jednostavnu daljnju instalaciju Python paketa.



Slika 4. Instacija Djanga u Command Promptu putem pip-a (Windows)

Slika 5. Instalacija Djanga u Terminalu putem apt-a (Linux)

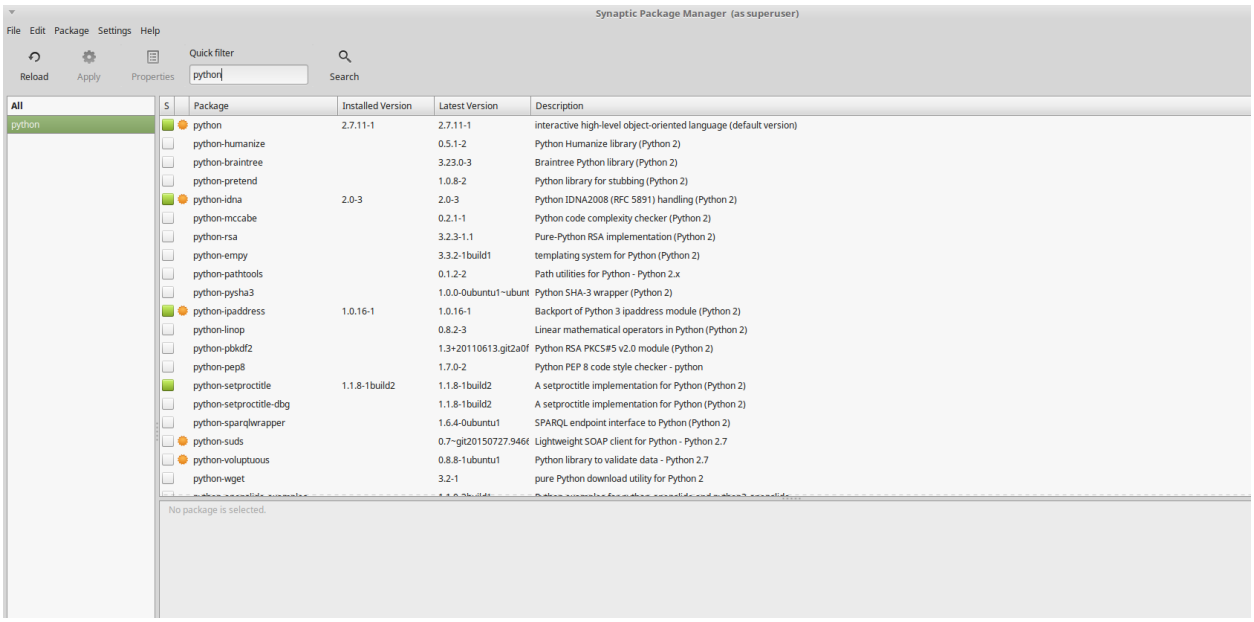
5. Pregled aplikacije ArchiPoint

5.1. Uvod u ArchiPoint i korištena okruženja

Budući da je Linux „prirodno“ okruženje za razvoj Django aplikacija, izbor upravo tog sustava za izradu aplikacije ArchiPoint bio je logičan i izvjestan. Distribucija korištena pri razvoju ove aplikacije, Linux Mint, jedna je od poznatih i često korištenih inačica sustava Linux bazirana na distribucijama Debian ili Ubuntu. Usprkos činjenici da je i sama distribucija Ubuntu derivat Debiana, te je tako i Mint dalji derivat Debiana, postoje i verzije Linux Minta direktno izvedene iz Debian distribucije te nose neke operativne razlike naspram češće korištene Ubuntu verzije. Debian i Ubuntu distribucije su koje se, uglavnom, razlikuju po pristupačnosti i lakoći korištenja, pri čemu Debian zahtijeva nešto više administrativnog poznavanja Linux sustava kako bi se koristile neke njegove naprednije postavke i mogućnosti.

Linux Mint, pri instalaciji, dolazi s već postavljene dvije verzije (točnije, tumača i prevodioca) jezika Python: 2.7 i 3.4. S obzirom da je pri izradi aplikacije ArchiPoint konkretno korišten Python 2.7, potrebno je napomenuti kako se dvije navedene serije, 2.x i 3.x, međusobno blago razlikuju na razini jezične sintakse i određenih funkcionalnosti te kako su obje i dalje podržane od strane većine poznatih servisa i u paralelnom razvoju. Izbor jedne od dvije navedene serije na posljetku je pitanje osobne preferencije. Linux distribucija korištena kao podloga za izradu ArchiPoint aplikacije jest Mint 18.2 (kodnog naziva „Sonya“) u kombinaciji s Cinnamon desktop okruženjem. Linux Mint izravan je derivat Ubuntu Linuxa pa tako pripada grani Debian distribucija, što znači da koristi i popularne Aptitude i Synaptic upravitelje softverskih paketa. Putem tih upravitelja, instalirati Django izrazito je jednostavno iz razloga što ga se odmah po instalaciji tretira kao integriranim u Python jezik. S obzirom da je u Mintovom repozitoriju kao standardno dostupna verzija prisutan Django 1.8.7, on je korišten i u izradi ArchiPointa kako bi se izbjegle potencijalne kolizije verzija ostalih korištenih Python i Linux paketa s nekom novijom i s Interneta preuzetom verzijom Djanga. Usprkos tome, može se napomenuti kako je, u trenutku pisanja ovog rada, najnovija verzija Djanga bila 1.11.4.

Na priloženoj slici 6., uhvaćenoj na ranije navedenom Linux Mint sustavu, vidljiv je Synaptic grafički upravitelj paketa s prikazanim rezultatima pretrage za softverskim paketima koji sadržavaju izraz „python“ u sebi. Kroz daljnjih nekoliko „klikova“, moguće je preuzeti i instalirati željeni paket. Na jednak način moguće je na spomenutom sustavu instalirati i Django, kao alternativa postupku koji koristi Terminal.



Slika 6. Pregled Linux paketa kroz Synaptic

Od mnogih mogućih načina i okruženja za razvoj Django aplikacija, u ovom slučaju korišteno je integrirano razvojno okruženje (engl. integrated development environment, IDE) PyCharm. To razvojno okruženje jedan je od mnogih JetBrains alata te je jedinstveno u tome što predstavlja cjelovitu soluciju prilagođenu upravo razvoju Python i, povrh svega, Django aplikacija. Samo okruženje napisano je u Javi i Pythonu te vrlo učinkovito upravlja unošenjem i korištenjem vanjskih Python knjižnica i procesiranjem Python koda. Štoviše, sama Django dokumentacija navodi PyCharm kao preporučenu soluciju u razvoju web aplikacija. Ostale moguće IDE solucije se uglavnom svode na popularno Java razvojno okruženje Eclipse i Aptana Studio. No, uzimajući u obzir činjenice kako ova dva navedena okruženja mogu djelomično učinkovito raditi s Python kodom tek nakon upotrebljavanja PyDev proširenja te kako su sva

dosad navedena razvojna okruženja besplatna za korištenje, PyCharm se može smatrati logičnim i profesionalnim izborom za razvoj Django aplikacija.

Osim korištenja cjelovite razvojne solucije kao što su integrirana razvojna okruženja, Django aplikacije, kao i većinu drugih aplikacija, moguće je razvijati i u jednostavnim uređivačima teksta (engl. text editors). Usprkos tome što takav pristup ne nudi mnogobrojne korisne alate i funkcije koje pogoduju učinkovitijem razvoju, sam kod je svakako moguće pisati, te neki uređivači nude i neke proširene funkcionalnosti, kao što su automatsko završavanje izraza (engl. autocompletion), ispravljači pogrešaka (engl. debuggers), označivači jezične sintakse (engl. syntax highlighting) te sučelja za interakciju s jezgrom ili konzole. Takav pristup razvoju kompleksnih aplikacija često znači korištenje više različitih alata koje IDE-i uglavnom imaju sadržane u sebi te više truda u upravljanju mnogobrojnim kodnim skriptama.

Neki od poznatijih i suvremenih uređivača teksta su:

- Visual Studio Code – iznimno opremljen uređivač teksta tvrtke Microsoft, koja također razvija i C# razvojno okruženje Visual Studio,
- Notepad++ - vrlo popularan, jednostavan no učinkovit uređivač teksta,
- Sublime Text,
- Brackets,
- Atom,
- Komodo Edit.

Dok integrirana razvojna okruženja najčešće predstavljaju cjelovita rješenja za razvoj aplikacija, određeni uređivači teksta također mogu predstavljati značajan alat svojom opremljenošću i proširivošću.

5.2. Upraviteljski servis iza aplikacije

Prije nego se krene u opis vidljivih i „opipljivih“ dijelova ArchiPoint aplikacije, opisat će se kod, alati i akcije iza same aplikacije, koji djeluju kao svojevrsni pozadinski servis za automatizirano održavanje ArchiPoint baze podataka.

Navedeni pozadinski servis osmišljen je kao način upravljanja bazom podataka koji ne zahtijeva konstantnu ljudsku interakciju i održavanje. Takav pristup omogućavao bi konstantan

priljev novih podataka u određenu lokalnu ili udaljenu lokaciju na nekom poslužitelju te obradu tih podataka i spremanje u bazu podataka koju aplikacija ArchiPoint koristi za vizualizaciju.

Za početak, sastavni dijelovi, bili oni apstraktni ili konkretni alati, ovog servisa su:

- Mrežna i poslužiteljska infrastruktura ustanove koja pruža podatke,
- Dijeljeni direktorij s datotekama smješten lokalno ili udaljeno na poslužitelju,
- Python programske skripte,
- Django manage.py datoteka za upravljanje i automatizaciju procesa,
- Cron daemon,
- Linux Bash jezgra.

Prvi uvjet za korištenje ovakve solucije jest mogućnost i način da određena informacijska ustanova pruža meta podatke o svom gradivu u formatu XML (engl. extensible markup language) koda s unificiranim i standardiziranim imenskim prostorom i strukturom. Daljnji proces treba bit postavljen kako slijedi.

1. Navedeni kod se, u obliku pojedinačnih datoteka ili jedne koja sadrži podatke o više jedinica gradiva, dostavlja u za to predodređenu lokaciju na poslužitelju. Jedan od mogućih i funkcionalnih načina za prijenos može biti vremenski određeni FTP servis.
2. Na poslužitelju je postavljen Cron daemon koji izvršava napisane Python programske skripte koje odrađuju glavninu posla. Cron je pozadinski servis koji pokreće takozvane Crontab datoteke u vremenski preddefiniranim terminima.
3. Crontab datoteka sadrži Bash skriptne naredbe koje koje pozivaju Python nad Djangovom *manage.py* upravljačkom datotekom i kao argument koriste ime jedne od napisanih Python programskih skripti.
4. Python skripte sadrže kod koji odrađuje otvaranje i raščlanjivanje (engl. parsing) dostavljenih XML datoteka po setu definiranih parametara. Tako ekstrahirani željeni podaci privremeno se spremaju u varijable te upisuju u ArchiPoint bazu podataka kroz Djangovo svojstvo objektno relacijskog mapiranja.

Budući da postoji mogućnost ponovljene dostave već obrađenih datoteka, u skripte je moguće implementirati i provjeru duplikata datoteka te, ukoliko se zaprimljene XML datoteke ne

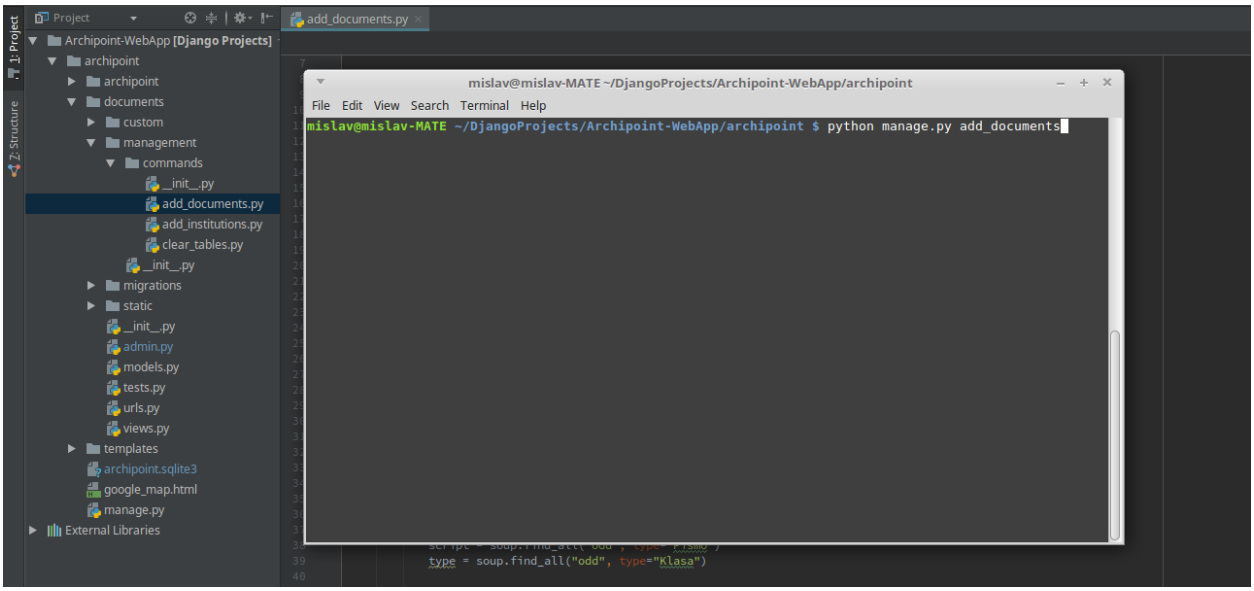
odluče čuvati u dijeljenom direktoriju, i brisanje tih datoteka, također uz provjeru duplikata te označavanje obrađenih datoteka.

ArchiPoint je aplikacija stvorena kao funkcionalni prototip i demonstracija ideje za potencijalnu integraciju jednog takvog sustava u mrežu arhivskih ustanova u Hrvatskoj. Iz tog razloga, trenutno sadrži tri različite pozadinske skripte:

- *add_institutions.py* – upravlja dodavanjem novih ustanova u tablicu *Institutions*,
- *add_documents.py* – upravlja dodavanjem fondova, serija i podserija u njihove odgovarajuće tablice,
- *clear_tables.py* – briše sve postojeće zapise iz tablica.

Svaku od navedenih programskih skripti moguće je proširiti i modificirati po potrebi, te napisati i dodati nove, ovisno o razmjerima i prirodi implementacije ArchiPoint aplikacije.

Python vrši interakciju s operacijskim sustavom na kojem se nalazi i njegovim datotečnim sustavom (engl. file system) pomoću integriranog programskog modula os. Taj modul omogućava Pythonu pozivanje i izvršavanje raznih funkcija koje koriste pozadinski operacijski sustav i njegove datoteke. Na taj način Python je u mogućnosti otvarati željene XML datoteke unutar dijeljenog direktorija i raditi s podacima koji se u njima nalaze.



Slika 7. Izvršavanje upravljačke Python skripte u Djangu

Nakon što se napisane programske skripte implementiraju u hijerarhiju Django projekta na ispravan način i inicijaliziraju tako da budu iskoristive, putem datoteke manage.py moguće ih je izvršavati kao dodatne upravljačke naredbe. Na priloženoj slici vidljiv je položaj tih datoteka u projektu te njihovo pozivanje kroz Bash jezgru.

U razdoblju inicijalnog razvoja ArchiPoint aplikacije i pisanja ovog rada, ArchiPointov pozadinski servis imao je razrađene sve predispozicije za potpunu implementaciju i automatska ažuriranja baze podataka, no nije integriran u potpunosti zbog nedostatka potrebe. Takav servis može pokazati svoju korisnost ukoliko više informacijskih ustanova ažurno dijeli meta podatke svog raspoloživog gradiva. No, uz minimalnu doradu, moguće je i s jednim izvorom meta podataka prikazati potpunu navedenu funkcionalnost, ukoliko je to potrebno.

5.3. Pozadina aplikacije

5.3.1. Postavke projekta

Svaki novo započeti razvoj neke određene aplikacije u Djangu se naziva projektom te će u ovom radu aplikacija, u trenucima kada će to biti potrebno, biti tako i referencirana. Ključan segment razvoja svake Django aplikacije jesu postavke cjelokupnog projekta. One se definiraju unutar posebne Python datoteke imenom *settings.py* koja se nalazi u direktoriju istog imena kao i cijeli projekt.

Datoteka *settings.py* sadrži već upisane varijable sa izmjenjivim vrijednostima te mogućnost dodavanja novih. Django se pri izvođenu raznih operacija referencira na postavke upisane u navedenoj datoteci, tako da je potrebno znati koje je varijable moguće dodati i kako izmijeniti postojeće da one budu važeće.

Neke od ključnih varijabli su:

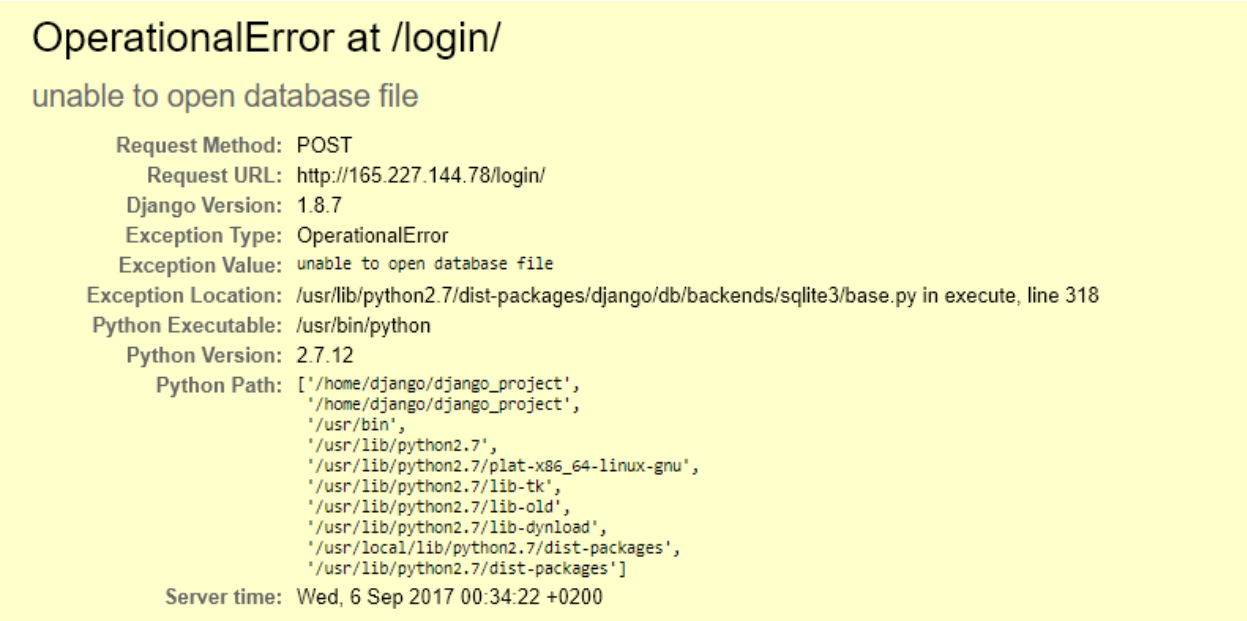
- *BASE_DIR* – određuje korijenski direktorij projekta,
- *DEBUG* – varijabla tipa boolean koja prima vrijednosti True ili False te određuje da li će Django vršiti analizu grešaka u produkciji,

- *ALLOWED_HOSTS* – lista adresa na kojima Django može poslužiti aplikaciju preko HTTP-a (engl. Hypertext transfer protocol),
- *INSTALLED_APPS* – moduli („aplikacije“) koji su integrirani u projekt,
- *ROOT_URLCONF* – određuje korijensku datoteku koja mapira HTTP upite na Python prikaze,
- *WSGI_APPLICATION* – određuje WSGI (engl. Web server gateway inteface) poslužitelja koji će vršiti komunikaciju između web poslužitelja i aplikacije,
- *DATABASES* – određuje bazu podataka koja će se koristiti te njene postavke.

```
DEBUG = True

ALLOWED_HOSTS = ['192.168.1.3', 'localhost']

INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'documents',
)

LOGIN_REDIRECT_URL = '/documents'
ROOT_URLCONF = 'archipoint.urls'
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, "templates")],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Kod 3. Datoteka settings.py

Na priloženom kodu 3. vidljiv je mali dio mogućeg sadržaja datoteke *settings.py*, u kojem su neke od ključnih ranije navedenih postavki. Varijablu DEBUG uvijek je potrebno postaviti na „False“ u produkcijskom korištenju web aplikacije zato što Django pri analizi pogrešaka prikazuje mnogo informacija o cjelokupnom servisu kritičnih za potencijalnu eksploataciju te predstavlja sigurnosni rizik. Takvi prikazi nazivaju se iskrcajima podataka (engl. data dumps). Popis *INSTALLED_APPS* sadrži popis dijelova cjelokupnog projekta koji se u Djangu nazivaju aplikacijama (engl. apps) te mogu biti prenošeni i korišteni u više projekata. Iz tog razloga nužno je definirati aktivne „aplikacije“ unutar projekta. U slučaju ovoj projekta, glavna „aplikacija“ nazvana je „documents“ i dodana na popis. O utjecaju postavki kao što su *TEMPLATES*, *WSGI_APPLICATION* i *ROOT_URLCONF* na web aplikaciju bit će više riječi u daljnjim poglavljima.



Slika 8. Djangova analiza pogrešaka

Standardni sustav baza podataka za novokreirani Django projekt jest SQLite. To je jednostavan i lako upravljiv tip baza podataka koji zahtijeva minimalno podešavanja te radi izrazito dobro u kombinaciji s Django aplikacijama. Drugi najkorišteniji tip baza podataka u

Django aplikacijama je MySQL, koji je, naspram SQLite-a, nešto robusniji, a također i najkorišteniji tip baza podataka u sustavima otvorenog koda (engl. open-source).

5.3.2. Programska pozadina – URL-i, prikazi i iznimke

Gotovo svaka web aplikacija ugrubo se dijeli na pozadinske i pristupne dijelove. Dok pristupni dio označava tehnologije korištene na vidljivom dijelu aplikacije te njen vizualni dizajn, pozadinski dio objedinjuje cjelokupnu funkcijsku logiku, baze podataka, programski kod i transakcijsku komunikaciju iza vidljivog dijela.

Tako pozadinski dio web aplikacije može čini komunikacija nekog objektno orijentiranog radnog okvira i njegove baze podataka putem SQL jezika ili nekog drugo tipa upita. Poznato je da baze podataka sadrže mapirane informacije pa je stoga potrebno stvoriti programsku logiku koja će vršiti upite na baze podataka u određenim situacijama i na određene načine, primjerice inicirane određenom akcijom na pristupnom dijelu aplikacije.

Još jedan aspekt programske pozadine web aplikacije jest upravljanje HTTP upitima. Svaka web aplikacija, zajedno s web poslužiteljem, mora biti u mogućnosti ispravno povezivati HTTP upite sa svojim prikazima. Na isti takav način radi i Django, određujući početnu Python datoteku u svojim postavkama koja prepoznaje URL-e (engl. uniform resource locator) u HTTP upitu te ih prosljeđuje na sljedeću datoteku ili, ukoliko terminira upit, inicijalizira povezane prikaze (engl. views).

U aplikaciji ArchiPoint, datoteka *urls.py*, unutar osnovnog direktorija aplikacije, vrši već navedenu primarnu provjeru URL-a u HTTP upitu, te prosljeđuje daljnje akcije na sljedeću *urls.py* datoteku u documents direktoriju ili terminira upit na Djangovim funkcijama za prijavu korisnika, ukoliko upit doista jest upućen na prijavu. Djangova knjižnica za autentifikaciju korisnika i trenutne sesije zove se *django.contrib.auth* te ju je moguće pozvati za jednostavnije i prilagođeno upravljanje prijavama. Ona sadrži funkcije imenom *login* i *logout* koje odrađuju prijavu i odjavu korisnika s aplikacije ArchiPoint onda kada je prema njima upućen ispravan HTTP upit.

Ukoliko URL u HTTP upitu sadrži parametre za dohvaćanje daljnjih resursa, *urls.py* prosljeđuje taj upit na sljedeću takvu datoteku. Druga datoteka za upravljanje URL-ima terminira

upite na nekom od napisanih Django prikaza. Primjerice, ukoliko navedena datoteka zaprimi URL koji glasi http://165.227.144.78/documents/fonds/, Django će provjeriti koji prikaz je potrebno pokrenuti za URL „fonds/“. Svi prikazi nalaze se u datoteci *views.py* i, u suštini, oni su Python funkcije koje odrađuju skup akcija i vraćaju vrijednosti kroz varijable u kontekstu za HTML prikaz i ime HTML predloška kojeg je potrebno inicijalizirati uz prikaz. Pravilo pri korištenju prikaza jest da se jedan URL poziv veže na jedan prikaz, koji zatim inicijalizira jedan HTML predložak. Drugim riječima, pozadinske operacije prema jednoj HTML datoteci potrebno je odraditi unutar jedne funkcije. Koliko god takvo pravilo zvučalo ograničavajuće, moguće je vrlo učinkovito upravljati različitim tijekom operacija i ishodima unutar jedne funkcije kroz programske petlje, uvjete i upravljanje iznimkama (engl. exception handling).

Iznimke (engl. exceptions) naziv je za programske greške koje se mogu javiti ukoliko aplikacija zaprimi neočekivanu vrijednost u ključnim trenucima koji ovise upravo o ispravnom tijeku programa ili tipu podataka i rezultata. Primjer koji jednostavno može objasniti iznimke i upravljanje njima nalazi se upravo na više mjesta u ArchiPointovim skriptama za popunjavanje tablica. U njima se nalaze funkcije koje odrađuju ekstrahiranje određenih vrijednosti iz XML koda te ih spremaju u varijable i upisuju u bazu podataka. Ukoliko skripta pokuša ekstrahirati vrijednost iz praznog XML elementa, vrijednost varijable koja ju prima bit će „bez vrijednosti“ (engl. null). Kako Python prevodilac prilikom takvog nevažećeg dodjeljivanja vrijednosti i upisa u ćeliju tablice ne bi naletio na neočekivanu grešku i prekinuo izvođenje aplikacije, potrebno je tu operaciju staviti unutar *try / except* bloka i definirati što će se dogoditi ukoliko program naiđe upravo na takvu situaciju. Dok se takav blok koda u Pythonu naziva *try / except*, u većini drugih programskih jezika, kao što su Java i C#, takav blok zove se *try / catch*. Obje sintakse zvuče logično i razumljivo ako uzmemo u obzir da je engleski naziv za takve neočekivane greške „exceptions“, a upravljanje njima naziva se hvatanjem (engl. catching).

```
xmlfile = open(directory + filename, "r")
soup = BeautifulSoup(xmlfile, "lxml")
value_note = soup.find_all("appraisal")

def try_value_note():
    try:
        value = value_note[0].text
    except (IndexError, AttributeError):
        value = ""
    return value
```

Kod 4. Korištenje *try / except* bloka u Pythonu

U priloženom isječku koda 4. vidljivo je upravljanje izlaznom vrijednošću varijable *value*. Budući da je varijabla *value_note* lista, varijabla *value* uzima vrijednost njenog prvog člana. Python, kao i većina programskih jezika, koristi princip nultog položaja pri označavanju. Ukoliko je vrijednost dohvaćenog prvog člana liste jednaka „bez vrijednosti“ i ova akcija vrati grešku indeksa ili pogrešnog atributa, varijabli *value* će biti dodijeljena vrijednost „prazno“. U području informacijske tehnologije, a posebice baza podataka i programskih jezika, od izrazite je važnosti razlikovati praznu vrijednost i „bez vrijednosti“ (null) iz razloga što su to dva potpuno drugačija koncepta, od kojih „null“ najčešće uzrokuje programske greške zbog pokušaja dodjeljivanja „ničega“ varijabli.

Na ranije opisani način vrši se komunikacija između programskog jezika pozadine, koji je u ovom slučaju Python, i vidljivog dijela web aplikacije, koji je HTML kod. Python funkcije odrađuju upite i izvršavaju potrebne akcije te prosljeđuju svoje varijable kroz kontekst u povezanu HTML datoteku. Na taj način podaci dohvaćeni iz baze podataka kroz Python mogu biti vidljivi i dostupni na korištenje u pristupnom sloju aplikacije u HTML-u i JavaScriptu. Kontekst kao apstraktni pojam u Djangu jest Python rječnik koji za ključeve ima nazive pod kojima će varijable biti dostupne u HTML-u, a za vrijednosti odgovarajuće Python varijable.

Proslijeđeni kontekst manifestira se u HTML-u web aplikacije kroz takozvani jezik za predloške. Django ima već ugrađenu podršku za dva takva jezika, Django Templating Language i Jinja, koji se međusobno minimalno razlikuju. Na taj način korištenje jednog od dva navedena jezika u projektu ponovno pada na osobnu preferenciju pa je potrebno definirati izbor u Django postavkama.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.jinja2.Jinja2',
        'DIRS': [os.path.join(BASE_DIR, "templates")],
        'APP_DIRS': True,
        'OPTIONS': {
            # ... some options here ...
        },
    },
]
```

Kod 5. Postavke za Jinja2 jezik za predloške

Razlika između Django Templating Language i Jinja2 svodi se na određeni broj sintaktičkih varijacija u izrazima. Oba jezika vizualno okvirno prate Python sintaksu, što djeluje vrlo logično ako se uzme u obzir da su napravljeni upravo za Python. Iako većinski obje tehnologije mogu gotovo jednako odraditi neku zadaću, web stranica Jinja2 jezika navodi kako, osim omanjih vizualnih razlika, Jinja2 ima neke proširene funkcionalnosti. Primjerice, pozivanje rezultata neke funkcije u for petlji kroz Jinja2 jezik, za raziku od Django Templating Language, omogućuje prosljeđivanje i proizvoljnih argumenata. Takav primjer usporedbe s Jinja2 web stranice priložen je niže kao kodni isječak 6. [13]

[13] Ronacher, A. Switching from other Template Engines, *Jinja2 Documentation*, URL: http://jinja.pocoo.org/docs/2.9/switching/. (7.09.2017.)

In Django method calls work implicitly, while Jinja requires the explicit Python syntax. Thus this Django code:

```
{% for page in user.get_created_pages %}

...

{% endfor %}
```

...looks like this in Jinja:

```
{% for page in user.get_created_pages() %}

...

{% endfor %}
```

Kod 6. Razlika između DTL i Jinja2

Izvor: Ronacher, A. Switching from other Template Engines, *Jinja2 Documentation*, URL: http://jinja.pocoo.org/docs/2.9/switching/. (7.09.2017.)

Unutar jedne Python funkcije prikaza mogu se odviti mnogobrojne operacije, no u kontekst za HTML prosljeđuje se određeni odabrani broj vrijednosti koje su, kao rezultat, bitne za korištenje u HTML-u. S obzirom da je već navedeno kako funkcije prikaza zaprimaju HTTP upite, u definiciji same funkcije kao argument se obavezno navodi upit, te uz njega proizvoljan broj drugih argumenata koje je funkciji potrebno proslijediti. Python izraz *return* na kraju funkcije prikaza određuje kako će vraćen biti HTTP upit, kontekst te ime HTML datoteke (predloška) na kojem će sve biti iskorišteno, objedinjeni pod integriranom funkcijom *render*. Na niže prikazanom isječku koda 7. moguće je vidjeti jednu cjelovitu funkciju prikaza, zajedno s hvatanjem i obradom HTTP zahtjeva, definiranjem konteksta te vraćanjem ranije navedenog skupa parametara.

```
def index_page(request):

    if request.GET:
        search_term = request.GET['term']
        results_fonds = Fonds.objects.filter(title__icontains=search_term)
        results_series = Series.objects.filter(unit_title__icontains=search_term)
        context = {"results_fonds":results_fonds, "results_series":results_series, "search_term":search_term}
        return render(request, "documents/search_results.html", context)
    else:
        return render(request, "documents/index.html")
```

Kod 7. Primjer funkcije prikaza

Konkretno gledano, priloženi isječak koda 7. jedan je od ArchiPoint prikaza koji upravlja pretragom baze podataka iz korisničkog sučelja. Funkcija provjerava da li je izvršen GET upit prema bazi podataka te kroz kontekst iz nje vraća željene vrijednosti. Ukoliko na početnoj stranici aplikacije nije izvršen upit za pretragu, funkcija jednostavno vraća početnu stranicu.

5.3.3. Klase i modeli

Osnovica svakog objektno orijentiranog jezika su klase. Klase su naziv za apstraktne definicije kalupa koji imaju mogućnost generiranja objekata, odakle i potječe atribut „objektno orijentirano“ u nazivu.

U većini programskih jezika, definiciju klase čine njeno ime, izrazi njenih karakteristika te izrazi proširenja ili implementacije neke postojeće klase. Tako definicija jedne Java klase može sadržavati izraze prisutnosti kao što su *public* i *private*, ime klase te izraz nasljeđivanja (engl. inheritance) kao što je *extends*. Nasljeđivanje omogućuje klasi preuzimanje svih karakteristika neke već definirane klase te na taj način svojom definicijom dodaje nove karakteristike. Python pri definiciji klase koristi ime klase, dok ime klase koju proširuje može izraziti kao argument unutar zagrada.

```java
//Java
public class Animal {

}

public class Porcupine extends Animal {

}


#Python
class Animal():

class Porcupine(Animal):
```

Kod 8. Usporedba Java i Python nasljeđivanja u klasama

U priloženom kodnom isječku 8. vidljivi su primjeri nasljeđivanja u Java i Python klasama. U primjerima za oba jezika, klasa Porcupine naslijedit će sve karakteristike klase Animal te svojom definicijom pridodati nove.

Modeli u Djangu su Python klase koje predstavljaju tablice u bazi podataka. Službena dokumentacija Django radnog okvira objašnjava modele na sljedeći način:

- „Svaki model je Python klasa koja proširuje *django.db.models.Model*.
- Svaki atribut modela predstavlja kolonu u tablici.
- Pomoću takvog ustroja, Django omogućuje automatski generiran API za pristup bazi podataka.“[14]

Ukoliko se uzme u obzir da su Django modeli klase, analogno se može zaključiti kako i oni kao atribut mogu imati neku funkciju. U slučaju korištenja neke funkcije kao atributa modela, ta funkcija se pri migracijama ne prenosi u tablicu kao kolona, za razliku od drugih varijabli. Na taj način dolazi se do djelomičnog pobijanja službene tvrdnje o atributima sa stranica Django dokumentacije. Aplikacija ArchiPoint koristi nekoliko napisanih funkcija kao

[14] Django Software Foundation, Models, *Django documentation*, URL: https://docs.djangoproject.com/en/1.11/topics/db/models/. (7.09.2017.)

atribute unutar modela, tako da je navedena iznimka i dokazana u praksi. Primjerice, model / klasa *Institutions* ima definiranu funkciju *count_fonds* unutar sebe i ta funkcija nije nikako prenesena u bazu podataka. No, funkcija intuitivnog naziva *count_fonds* svakako je iskoristiva na jedan drugi način. Ona se može pozvati i izvršiti preko objekta koji se generira iz klase *Institutions* te se koristi kao izvor podataka o broju fondova koji su povezani uz određenu informacijsku ustanovu. Taj podatak dobiva se na sljedeći način:

1. Klasa *Fonds* kao strani ključ ima klasu *Institutions*.
2. Klasa *Institutions* tada ima mogućnost korištenja varijable *fonds_set* koja predstavlja set svih fondova asociranih s određenom institucijom putem stranog ključa.
3. Funkcija *count_fonds*, kao atribut klase *Institutions*, vraća broj fondova u varijabli *fonds_set*.
4. Ako se generira objekt iz klase *Institutions*, on je u mogućnosti pozvati funkciju *count_fonds* kao svoj atribut.

5.3.4. Prilagođena polja u modelima

Python klase, kao i one drugih programskih jezika, imaju mogućnost nasljeđivanja karakteristika. Te karakteristike općenito se nazivaju atributima. Svi napisani modeli aplikacije ArchiPoint nasljeđuju karakteristike klase *Model* pa su tako i njihovi atributi objekti generirani iz klasa koje predstavljaju stupce tablice u bazi podataka. Te klase sve u svom nazivu imaju riječ „field“ upravo iz razloga što su namijenjene da budu tablična polja te ih postoji određen broj već integriranih i definiranih u Django radnom okviru koji su spremni za pozivanje i korištenje. Svi integrirani i raspoloživi tipovi polja mogu se pronaći na stranicama službene Django dokumentacije, zajedno sa njihovim opisima i primjerima implementacije.

Tipovi polja u Django modelima ugrubo se dijele na polja koja definiraju tip podataka i polja koja definiraju odnose, a abecednim redom to su:

Polja tipova podataka	
AutoField	Cjelobrojno polje koje se automatski redom popunjava.
BigAutoField	AutoField polje koje podržava vrijednosti od 1 do 9223372036854775807.
BigIntegerField	Cjelobrojno polje koje podržava vrijednosti od -9223372036854775808 do 9223372036854775807
BinaryField	Polje za spremanje binarnih zapisa.
BooleanField	Polje koje prima logički „TRUE“ ili „FALSE“.
CharField	Tekstualno polje za tekst srednje duljine.
CommaSeparatedIntegerField	Polje za cjelobrojne vrijednosti odvojene zarezima.
DateField	Polje za format nadnevka.
DateTimeField	Polje za format nadnevka s vremenom.
DecimalField	Decimalno polje koje koristi Decimal tip podataka.
DurationField	Polje za spremanje formata vremenskih raspona.
EmailField	Polje za spremanje formata email adresa.
FileField	Polje za prijenos podataka.
FilePathField	Tekstualno polje ograničeno na imena datoteka koje se nalaze na određenoj lokaciji.
FloatField	Decimalno polje koje koristi Float tip podataka.
ImageField	Polje koje nasljeđuje atribute FileFielda no također i vrši provjeru da li je datoteka grafika.
IntegerField	Cjelobrojno polje.
GenericIPAddressField	Tekstualno polje koje prihvaća format IP adrese.
NullBooleanField	BooleanField koje prihvaća i „bez vrijednosti“ kao unos.
PositiveIntegerField	Cjelobrojno polje koje prihvaća vrijednosti jednake nuli i veće od nje.
SlugField	Kratko tekstualno polje koje se najčešće koristi za spremanje URL-a.

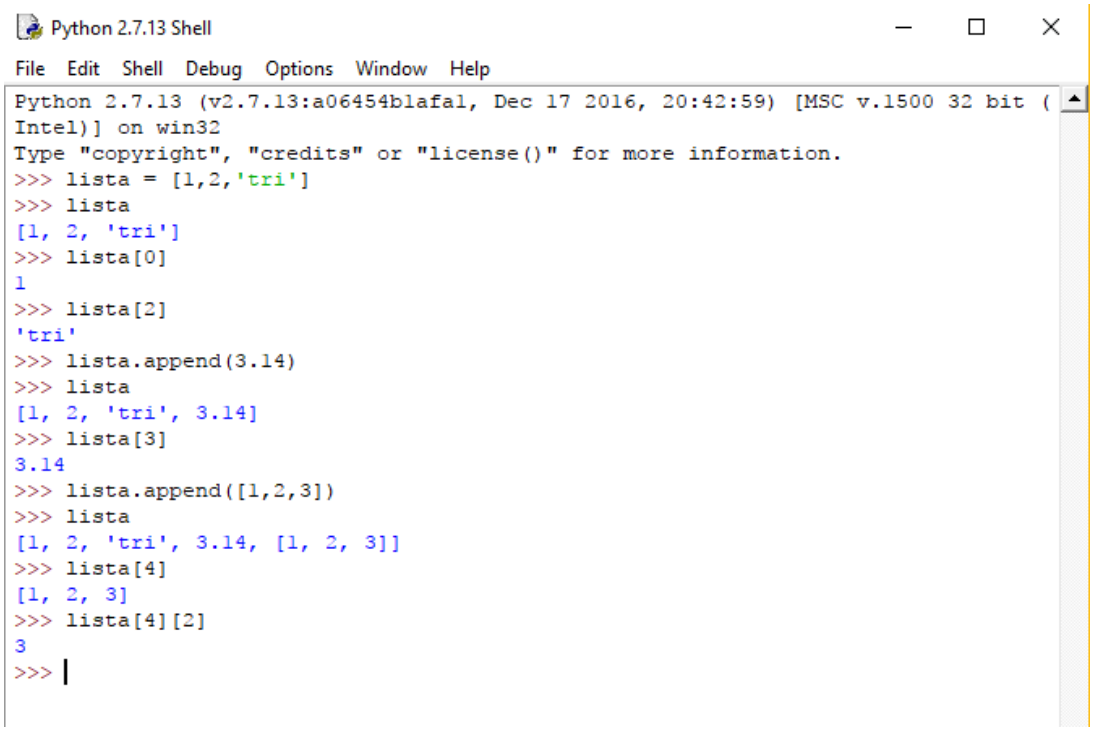
SmallIntegerField	Polje za male cjelobrojne tipove.
TextField	Tekstualno polje koje prihvaća velik raspon znakova.
TimeField	Polje za format vremena bez nadnevka.
URLField	Tekstualno polje za spremanje URL-a.
UUIDField	Polje za spremanje jedinstvenih identifikatora koje korist Pythonovu UUUID klasu.
Polja odnosa	
ForeignKey	Poveznica više objekata prema jednom. Standardna forma stranog ključa.
ManyToManyField	Poveznica više objekata prema više objekata.
OneToOneField	Poveznica jednog objekta na samo jedan objekt.

Tablica 1. Tipovi Django polja u modelima

Usprkos Djangovom širokom spektru raspoloživih integriranih klasa za različite tipove tabličnih polja, neke okolnost mogu stvoriti potrebu za nekom nekonvencionalnom formom podataka ili, u krajnjem slučaju, onom koja nije dostupna kroz već definirane tipove polja. U takvim prigodama do izražaja dolazi Djangova mogućnost stvaranja i deriviranja novih i prilagođenih tipova polja. Kao što su i standardna polja predstavljena Python klasama, tako se i prilagođena polja definiraju zasebnom klasom. S obzirom da je Django ograničen tipovima podataka koje klase polja mogu predstavljati, pri definiranju vlastite prilagođene klase polja, potrebno ju je derivirati iz već postojeće koristeći sposobnost nasljeđivanja.

Na navedeni način za aplikaciju ArchiPoint stvorena je prilagođena klasa Django polja imenom *SeparatedValuesField*, što u doslovnom prijevodu s engleskog jezika znači „polje odvojenih vrijednosti“. Ako se promotre integrirani tipovi Django polja, moguće je primijetiti kako značajan broj polja predstavlja određeni ekvivalent Python standardnog tipa podataka (engl. standard data types). Neki od standardnih tipova podataka u jeziku Python su cjelobrojni tip (engl. integer), znakovni tip (engl. integer), decimalni tip (engl. float), lista (engl. list) i rječnik (engl. dictionary). U većini slučajeva, ostali programski jezici također imaju iste ili izrazito slične tipove podataka na raspolaganju, uz česte razlike u nazivu, kao što je razlika između Python liste i Java arraya te Python rječnika i Java HashMapa.

Kroz razvoj ArchiPointa javila se potreba za implementacijom polja koje prihvaća Python listu kao podatkovni tip. Lista je serija jednakih ili različitih tipova podataka koja se može koristiti u cijelosti kao svojevrsni objekt ili kroz dohvaćanje pojedinih članova. Na slici 9. prikazana je demonstracija upravljanja Python listom.



Slika 9. Upravljanje Python listom

Na slici 9. je vidljivo da lista može sadržavati različite tipove podataka, pa čak i ugniježđenu drugu listu, čiji se pojedini članovi mogu dohvatiti drugom razinom uglatih zagrada. Kao i ostalom Python numeriranju, potrebno je voditi računa o korištenju nultog položaja.

Prepreka u razvoju ArchiPointa bilo je upravo nepostojanje Django polja koje je u mogućnosti ispravno prihvatiti i obrađivati Python listu kao tip podataka. Iz tog je razloga napravljena ranije navedena *SeparatedValuesField* klasa koja je proširenje već postojeće *TextField* klase. No, budući da je ta klasa poprima karakteristike klase tekstualnog polja, potrebno je bilo simulirati polje koje će moći prihvaćati i vraćati Python listu koristeći postojeće metode. Ono što jedinstveno odjeljuje članove u listi jest zarez. Vodeći se time, klasa *SeparatedValuesField* koristi funkcije koje listu svedu na obični znakovni tip (engl. string)

podataka koji u sebi sadrži zareze. Kao standardan tekst lista je pohranjena i u tablicu baze podataka. Pri dohvaćanja vrijednosti iz takve ćelije, određena funkcija vrši dijeljenje, to jest „tokenizaciju“, teksta po zarezima koji se u njemu nalaze. Ukoliko uneseni tekst sadrži zareze samo na mjestima odvajanja članova, pri dohvaćanju podataka dijeljenje će vratiti listu u obliku u kojem je ona u tablicu i unesena.

Problem se nameće u situacijama kada je član liste tekst koji u sebi sadrži zarez. Jasna je pretpostavka da će funkcija u klasi *SeparatedValuesField* pri dohvaćanju podataka podijeliti tekst na previše dijelova te tako vratiti pogrešnu listu. No, i takve je okolnosti moguće ispraviti privremenim dorađivanjem unesenog teksta na način da se željeni zarezi, samo za potrebe pohranjivanja u bazu podataka, zamijene nekim drugim znakom, čije su vjerojatnosti za nasumično pojavljivanje u nekom tekstu izrazito male. Tada se funkcija koja vrši dijeljenje teksta izmijeni tako da dijeli tekst po željenom znaku i slaže listu za vraćanje. Na taj će način dohvaćena lista biti istog oblika kao i ona koja je pohranjena u tablicu. Naravno, kako bi se vjerojatnosti za slučajnim pojavljivanjem spornog znaka za dijeljenje teksta još drastičnije smanjile, kao parametar moguće je postaviti nekoliko znakova, umjesto samo jednog. Primjerice, teško je da će neki prirodni tekst sam po sebi sadržavati slijed znakova kao što su „///“ ili „##“.

```
from django.db import models

class SeparatedValuesField(models.TextField):
    __metaclass__ = models.SubfieldBase

    def __init__(self, *args, **kwargs):
        self.token = kwargs.pop('token', ',')
        super(SeparatedValuesField, self).__init__(*args, **kwargs)

    def to_python(self, value):
        if not value: return
        if isinstance(value, list):
            return value
        return value.split(self.token)

    def get_db_prep_value(self, value, connection, prepared=False):
        if not value: return
        assert (isinstance(value, list) or isinstance(value, tuple))
        return self.token.join([unicode(s) for s in value])

    def value_to_string(self, obj):
        value = self._get_val_from_obj(obj)
        return self.get_db_prep_value(value)
```

Kod 9. Klasa *SeparatedValuesField*

5.4. Programske knjižnice i druga proširenja radnog okvira

Programske knjižnice često su jedan od ključnih dijelova aplikacije, bila ona web ili desktop orijentirana. U mnogo situacija, korištenje knjižnica značajno olakšava proces razvoja aplikacije ili ga čak čini općenito izvedivim. Svaki programski jezi i tehnologija imaju svoj način uključivanja vanjskih knjižnica (engl. external libraries) u svoju strukturu ili projekt. Kako je već objašnjeno na primjeru Djanga, implementacija vanjskih knjižnica i ostalih modula i paketa u Pythonu je izrazito jednostavna. Ukoliko se koristi knjižnica koja je pristupna na Python repozitoriju s paketima ili ukoliko se željena knjižnica dohvaća preko Linux distribucije sa udaljenog repozitorija, instalacija postaje vrlo jednostavna i svodi se na svega jednu ili nekoliko potrebnih akcija. No, ako željena Python knjižnica ne postoji u pip repozitoriju ili na repozitoriju

korištene Linux distribucije, potrebno ju je preuzeti s određene web lokacije preko putem web preglednika te ju ručno integrirati u Python izvorni kod (engl. source code).

5.4.1. GeoPy

Tvrdnju da je bez programskih knjižnica u nekim situacijama gotovo i nemoguće ostvariti neki projekt razvoja moguće je vrlo jasno potkrijepiti na primjeru Python knjižnice GeoPy (u daljnjem tekstu geopy, što joj je drugi službeni oblik pisanja naziva). Geopy se koristi kao programibilno Python sučelje (engl. API) za interakciju s više različitih web servisa za geokodiranje. Nakon što je ispravno implementirana u Python, geopy knjižnica ima mogućnost primanja alfanumeričkih oblika lokacija i adresa, te vraćanja njihovih zemljopisnih dužina i širina u obliku decimalnih vrijednosti.

Ključni sastavni dijelovi cjelokupne geopy knjižnice su zasebne Python klase napravljene za svaki od poznatih web servisa za geolociranje i geokodiranje. Samo neke od raspoloživih klasa za interakciju s odgovarajućim web servisima su ArcGIS, Bing, Baidu, GeoNames, GoogleV3, IGNFrance, YahooPlaceFinder i Yandex. Neke od navedenih klasa odgovaraju vrlo popularnim i često korištenim servisima na webu, kao što su to upravo Google Maps ili Bing Maps. Svaka klasa, nakon što se pozove kroz Python kod, zaprima određene prilagođene argumente te izvršava komunikaciju prema odgovarajućem web servisu. Nakon toga vraća geokodirane vrijednosti adrese ili neki drugi rezultat, ovisno o funkciji koja se koristi i argumentima koji se prosljeđuju.

Ako se za primjer uzme klasa GoogleV3, njen objekt koristi funkciju geocode koja, kao jedan od argumenata prihvaća alfanumerički format neke adrese kao tekstualnu varijablu. Funkcija geocode sastavlja ispravan upit koji prosljeđuje na Googleovo sučelje (API) za geokodiranje te, također, s njega u aplikaciju vraća rezultat u obliku zemljopisne dužine i širine.

U aplikaciji ArchiPoint knjižnica geopy korištena je za slanje upita upravo prema Google web servisu za geokodiranje iz razloga što je taj servis izrazito pouzdan, precizan i besplatan za korištenje do određenog broja iskorištenih upita. Također, Googleov web servis posjeduje izrazito visok prag tolerancije prema formatu proslijeđene alfanumeričke lokacije. To znači da obrada različitih nepotpunih i nestandardnih naziva lokacija u mnogo slučajeva uspijeva vratiti

željene rezultate. Takva fleksibilnost može reći da je Googleov web servis za geokodiranje prilično otporan na pogreške (engl. failproof).

Geokodirane adrese u ArchiPointu spremaju se u odgovarajuće stupce u bazi podataka, zajedno s odgovarajućim alfanumeričkim formatom adrese. Njihovo korištenje u svrhu vizualizacije podataka bit će opisano kasnije u ovom radu.

Ako se u obzir uzme kompleksnost jednog web servisa i njegova uloga u razvoju ArchiPoint aplikacije, postaje jasno koliko su programske knjižnice olakšavajući faktor u cijelom procesu. Kada programibilna sučelja web servisa ne bi bila omogućena na korištenje knjižnicama kao što je geopy, najmanje što bi bilo potrebno jest razviti cjeloviti algoritam koji obrađuje takve podatke i radi gotovo identičan posao koji, u slučaju ArchiPointa i mnoštva drugih web aplikacija, odrađuju servisi kao što je Google. Podnošenje određenih upita za obradu podataka bilo bi moguće i kroz samo web sučelje servisa (API), no korištenje jedne ovakve knjižnice znatno skraćuje posao i dodaje nove funkcionalnosti. Svaka tehnologija koja nema knjižnice prilagođene izvršavanju upravo takvih upita i dalje može koristiti Googleov servis za geokodiranje sastavljanjem i slanjem HTTP upita prema samom API-u, no tada je potrebno uložiti mnogo više u razvoj programskih metoda koje će takav upit slati, zaprimati i obrađivati.

5.4.2. Beautiful Soup

Još jedna izrazito korisna Python knjižnica korištena u izradi aplikacije ArchiPoint jest Beautiful Soup. To je knjižnica koja Pythonu omogućuje čitanje i raščlanjivanje HTML i XML koda po preciziranim parametrima.

Raščlanjivanje je jedan od prijevoda engleskog termina „parsing“. Ostali mogući prijevodi uključuju nazive parsiranje i sintaksna analiza. Željko Agić u svom radu „Pristupi ovisnosnom parsanju hrvatskih tekstova“ pojam parsanje objašnjava kao sintaksnu analizu, no također i navodi kako je to formalniji naziv, dok se u računalnoj znanosti uobičajeno koristi kolokvijalni naziv parsanje.[15] Ako se u kontekstu korištenja jedne programske knjižnice za takvu obradu teksta i koda pogleda što stoji iza riječi parsanje ili parsiranje, vidljivo je kako ta praksa

[15] Agić, Ž. Pristupi ovisnosnom parsanju hrvatskih tekstova. Doktorski rad. Filozofski fakultet Sveučilišta u Zagrebu. URL: http://darhiv.ffzg.unizg.hr/2337/1/587199.zagic_phd.pdf. (Zagreb, 2012.)

donekle odskače od opisa sintaksne analize u području lingvistike. Parsiranje je u slučaju korištenja Beautiful Soupa raščlanjivanje koda po zadanim parametrima i ekstrahiranje podataka. Riječi parsanje i parsiranje obje je moguće pronaći korištene u jednakoj mjeri zato što su obje tuđice porijeklom iz engleskog jezika.

Ono što Beautiful Suop odrađuje izrazito dobro jest dohvaćanje različitih informacija iz XML koda koristeći širok spektar mogućih parametara i uvjeta. Primjerice, iteracijom kroz XML elemente, moguće je dohvatiti tekst unutar svih *title* elemenata, čime bi se dobio popis svih naslova iz nekog dokumenta. Osim samog dohvaćanja teksta unutar elemenata, moguće je dohvatiti i sam naziv XML elementa kao izlaznu vrijednost i dodatne atribute, ukoliko postoji više atributima označenih elemenata istoga naziva.

```
<c level="item">
    <did>
        <unitid repositorycode="HR-HDA" label="Signatura">2</unitid>
        <unittitle label="Naslov" type="glavni">Agonija</unittitle>
        <unitdate label="Datum" normal="1998/1998">1998-1998</unitdate>
    </did>
</c>
<c level="item">
    <did>
        <unitid repositorycode="HR-HDA" label="Signatura">3</unitid>
        <unittitle label="Naslov" type="glavni">Ajde, dan...prođi...</unittitle>
        <unittitle label="Naslov" type="Skraćeni naziv">Slow days,-2006</unittitle>
        <unitdate label="Datum" normal="2006/2006">2006-2006</unitdate>
    </did>
</c>
```

Kod 10. Primjer hijerarhije XML elemenata s dodatnim atributima

```
from bs4 import BeautifulSoup

xmlfile = """<c level="item">
    <did>
        <unitid repositorycode="HR-HDA" label="Signatura">2</unitid>
        <unittitle label="Naslov" type="glavni">Agonija</unittitle>
        <unitdate label="Datum" normal="1998/1998">1998-1998</unitdate>
    </did>
</c>
<c level="item">
    <did>
        <unitid repositorycode="HR-HDA" label="Signatura">3</unitid>
        <unittitle label="Naslov" type="glavni">Ajde, dan...prođi...</unittitle>
        <unittitle label="Naslov" type="Skraćeni naziv">Slow days,-2006</unittitle>
        <unitdate label="Datum" normal="2006/2006">2006-2006</unitdate>
    </did>
</c>"""

soup = BeautifulSoup(xmlfile, "lxml")

naslovi = soup.find_all("unittitle", label="Naslov")
print naslovi

for naslov in naslovi:
    print naslov["type"]
```

Kod 11. Primjer dohvaćanja vrijednosti pomoću Beautiful Soupa

U priloženim kodnim isječcima 10. i 11., moguće je vidjeti primjer dohvaćanja vrijednosti iz XML koda čiji elementi sadrže i dodatne atribute. Kao i svaku Python programsku knjižnicu i bilo koji vanjski kod, Beautiful Suop se u trenutnu skriptu uključuje pomoću izraza *from / import* koji u ovom slučaju glasi „from bs4 import BeautifulSoup“. Algoritam napisan u priloženom kodu iz XML koda dohvaća sve elemente naslova koji su imaju oznaku *Naslov* i sprema ih u varijablu naslovi, te iz nje, za svaki od vraćenih članova, ispisuje vrijednost dodatnog atributa *type*. Na taj način dobije se tekst „glavni“ i „Skraćeni naziv“ pri ispisu.

XML kod korišten kao izvor meta podataka za arhivsko gradivu u ArchiPointu kompleksan je u svojim atributima te dovoljno opsežan da bi nedovoljno precizirano ekstrahiranje vrijednosti iz njega bilo izrazito nepraktično. Stoga je u ranije spomenutim Python skriptama *add_institutions* i *add_documents* korišten širok spektar Beautiful Soup definicija, u kojima se katkad kao izlazna vrijednost uzima vrijednost atributa iz razloga što postoji mnogo XML elemenata istog imena ili čak zbog toga što je za potrebe aplikacije u nekim trenucima bila bitnija upravo ta vrijednost od samog teksta elementa. S obzirom da cijeli kod kroz više datoteka ima istu hijerarhiju i koristi iste nazive elemenata, u trenucima kada neko arhivsko gradivo nema neku moguću osobinu koju ostatak gradiva ima, pojedini standardizirani XML elementi mogu ostati prazni. Kako bi se izbjegle programske pogreške u aplikaciji, *try / except* blokovi bili su uvelike korišteni.

5.4.3. Bootstrap

Kao jedan od najkorištenijih web radnih okvira za pristupne slojeve (engl. front-end web framework) aplikacija, Bootstrap se primjenjuje u trenucima kada je za projekt potreban praktičan i izravan način kreiranja vidljivog pristupnog dijela aplikacije, ali svakako i u situacijama koje iziskuju proširivanje, obradu i stvaranje novih vizualnih komponenata web aplikacija kroz moderan radni okvir orijentiran na, trenutno, vrlo popularan materijalni dizajn. Bootstrap službena web stranica navodi ga kao najpopularniji radni okvir ranije navedenog tipa te kao drugi najpopularniji projekt na GitHubu.[16]

Bootstrap, kao primarno HTML i CSS radni okvir, nudi mnogo preddefiniranih klasa HTML objekata i CSS stilova. Klase HTML objekata označavaju se i navode na jednak način kao i ranije opisani XML atributi te se na taj način mogu iz Bootstrapa koristiti razni integrirani modeli. Svaka HTML klasa uključena u Bootstrapu ima svoje definirane parametre i karakteristike koje se, također, mogu mijenjati i proširivati korištenjem vlastitih vrijednosti za CSS varijable. Na taj način predviđene vrijednosti se zamijene vlastitima definiranim unutar CSS oznaka stilova.

[16] Bootstrap. URL: https://getbootstrap.com/docs/3.3/. (9.09.2017.)

Budući da je ArchiPoint prototipni projekt samo jedne osobe, Bootstrap kao radni okvir za dizajn i izgled web aplikacije bio je logičan put. Glavnina vizualnog prikaza aplikacije ArchiPoint dobivena je korištenjem, mijenjanjem i proširivanjem Bootstrap klasa. Primjerice, glavna ArchiPoint navigacijska traka koja sadrži izbornike za različite funkcionalnosti i prikaze aplikacije je modificirana i prilagođena *navbar* Bootstrap klasa HTML objekta *nav*.

Implementacija Bootstrapa u web projekt vrši se sličan način kao i ostale programske knjižnice i moduli. Doslovna instalacija Bootstrapa ne postoji nego se njegov izvorni kod preuzme sa službene web lokacije te se smjesti u određene direktorije projekta. Način referenciranja projektnog koda na Bootstrapove HTML, CSS i JavaScript knjižnice ovisi o web radnom okviru u kojem se one koriste. Pri razvoju ArchiPointa Bootstrap je implementiran postavljanjem njegovih resursa u Django projektni direktorij zvan static, koji se koristi za sve resurse koji utječu na pristupni i vizualni dio aplikacije, kao što su slike i CSS stilovi. Lokaciju tog direktorija potrebno je bilo definirati u postavkama Django projekta, kako bi se njegov sadržaj mogao referencirati u izvornom kodu aplikacije.

Drugi način na koji je moguće implementirati Bootstrapove knjižnice i referencirati se na njih jest putem poveznice na mrežu za dostavu sadržaja (engl. content delivery network, CDN). Na taj način zaobilazi se cijeli postupak preuzimanja izvornog koda željene knjižnice (u ovom slučaju osnovnog Bootstrap koda), spremanja, te određivanja putanje na direktorij unutar koda. Za korištenje programskih knjižnica sa udaljene online lokacije putem CDN-a, dovoljno je referencirati udaljeni izvor u *head* dijelu HTML koda projekta na način na koji bismo definirali svaku poveznicu u HTML-u. Prednost takvog principa korištenja vanjskih programskih knjižnica je jednostavnost implementacije putem jedne reference u izvornom kodu, dok je nedostatak potreba za stalnom povezanošću na Internet. Činjenica jest da se ovdje radi o web aplikaciji, no postoje situacije u kojima se aplikaciju može pokrenuti, a računalo ne mora biti spojeno na

```
<!-- Dropdowns -->

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-
select/1.12.2/css/bootstrap-select.min.css">
<script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-select/1.12.2/js/bootstrap-
select.min.js"></script>
```

Kod 12. Dohvaćanje programskih knjižnica putem CDN-a

Internet. Najbolji primjer takve situacije jest faza lokalnog razvoja ArchiPoint aplikacije, u kojoj ne bi bilo moguće dohvatiti referencirane izvore sa lokalnog računala, ukoliko ono nije bilo povezano na Internet. Kodni isječak 12. prikazuje primjer udaljenog referenciranja na CSS i JavaScript knjižnice preko CDN-a.

5.5. Pristupni dio aplikacije

5.5.1. Što je pristupni dio

Pristupni dio (engl. front-end) web aplikacije često se opisuje kao sve što je korisniku aplikacije pri korištenju vidljivo, uključujući sam dizajn aplikacije, korisničko sučelje i neke jezike, kao što su HTML, CSS i JavaScript.[17] Razvoj pristupnog dijela neke aplikacije može uključivati čisto dizajniranje grafike i samog prikaza koristeći softver za rad s grafikom, rad na površinskim funkcionalnostima aplikacije koristeći HTML, CSS i različite JavaScript radne okvire ili jednostavno oboje, budući da je ponekad čak teško povući crtu i između naoko prepoznatljivo definiranih područja razvoja web aplikacija, kao što su sam pozadinski i pristupni dio.

Ukoliko se kao glavni orijentir uzima odnos web preglednik – poslužitelj, vrlo precizna definicija pojma pristupnog dijela jest onaj dio aplikacije (koda) koji se izvršava u web pregledniku, dok pozadinski dio, svojim izvršavanjem na poslužitelju s razlogom nosi i naziv poslužiteljska strana (engl. server-side).

Web preglednici odavno su nadišli svoju prvobitnu definiciju i funkciju, te već značajno dugo predstavljaju cijeli zatvoreni sustav. Kod pristupanja nekoj web aplikaciji, web preglednik pomoću HTTP upita preuzima HTML i CSS kod, to jest prikaz, te aplikacije, dok se metode definirane u programskom jeziku pozadine te aplikacije odviju na poslužitelju i jednostavno vrate svoj rezultat ukomponiran u preuzeti HTML kod.

[17] What's the difference Between the Front-End and Back-End?. *Pluralsight*. URL: https://www.pluralsight.com/blog/film-games/whats-difference-front-end-back-end. (28.01.2015.)

5.5.2. Korištenje Django predložaka

Aplikacija ArchiPoint svoj prikaz web stranica temelji na korištenju HTML datoteka zvanih Django predlošci. Takve datoteke ekstenzijom su označene kao regularne HTML datoteke, no njihova struktura koda djelomično se razlikuje od uobičajene HTML strukture. Kao prva razlika, može se uočiti kako predlošci ne moraju posjedovati HTML oznake za početak i kraj stranice, a također niti za ostale sastavne dijelove, kao što je to zaglavlje (engl. head) dokumenta. Umjesto navedenog, Django predlošci koriste izraze Django jezika za predloške (u daljnjem tekstu DTL) kako bi označili početak i kraj relevantnog HTML koda.

Funkcionalnost Django predložaka u kombinaciji s DTL-om moguće je proučiti na primjeru ArchiPointove stranice za prikaz popisa svih informacijskih ustanova uključenih u

```
{% extends "documents/headerfooter.html" %}

{% block title %}Archipoint{% endblock %}

{% block content %}
  <h3>Institucije</h3>
  <div title="institutions_list">
      <ul class="list-group">
        {% for institution in all_institutions %}
          <a href="{{institution.id}}"><li class="list-group-item"><span class="glyphicon glyphicon-
chevron-right"></span> {{institution.name}}<span
class="badge">{{institution.count_fonds}}</span></li></a>
        {% endfor %}
      </ul>
  </div>

<div class="well">
<p>Popis svih raspoloživih institucija na web servisu.</p>
<p>Brojčana oznaka na kraju imena pojedine institucije označava broj registriranih fondova u njoj.</p>
</div>

<a href="/documents/fonds"><button type="button" class="btn btn-success"><span class="glyphicon
glyphicon-th-list"></span> Popis svih fondova</button></a>
<br><br>
<p><a href="/documents"><button type="button" class="btn btn-success"><span class="glyphicon
glyphicon-arrow-left"></span> Povratak</button></a></p>
{% endblock %}
```

Kod 13. Predložak *institutions.html* i primjer sadržaja za proširenje

aplikaciju te broja fondova koje uključuju. Datoteka *institutions.html* predložak je koji sadržava HTML kod, no njegov sadržaj obuhvaćen je DTL izrazima *block content* i *endblock* koji intuitivno predstavljaju početak i kraj koda koji će biti uključen u drugu HTML datoteku.

Datoteka *headerfooter.html* sadrži HTML kod koji čini osnovu svake ArchiPoint stranice. Dizajnom ArchiPoint je dosljedan kroz sve svoje stranice, tako da je jedino što se mijenja sadržaj središnjeg dijela svake od njih. Osnovni i ponavljajući vizualni elementi, kao što je traka s izbornicima i tekstualno polje za pretragu baze podataka nalaze se definirani u datoteci *headerfooter.html*, koja sadrži i osnovne HTML oznake za početak i kraj koda te područje koje označava gdje će sadržaj predložaka biti uključen. DTL izrazi na području uključivanja jednaki su kao i na početku svakog predloška, uz razliku što svaki predložak dodatno sadrži izraz koji definira koju HTML datoteku on proširuje. Tako svaki predložak u ArchiPointu sadrži DTL izraz *extends /documents/headerfooter.html* pri samom vrhu.

Ono što web preglednik putem URL-a u HTTP upitu dohvaća jest upravo Django predložak koji dodatno definira koju HTML datoteku on proširuje. Provjerom ArchiPointovog izvornog koda dohvaćene web stranice vidljivo je kako web preglednik navedeno proširivanje prikazuje kao jedinstveni kod jedne stranice koja sadrži sve elemente osnovnog HTML koda i predloška koji ga proširuje. Također, sve DTL varijable i funkcije koje se nalaze u HTML kodu stranice, provjerom koda u web pregledniku manifestiraju se isključivo putem svojih vrijednosti prikazanih u HTML-u. Kroz takvu provjeru moguće je jasno vidjeti ranije opisanu primjenu DTL-a u prenošenju vrijednosti iz programskog jezika u kontekst HTML koda.

Još jedna vrlo korisna mogućnost Django predložaka jest uključivanje cijelog predloška ili samo njegovog fragmenta u više različitih HTML stranica pomoću izraza *includes*. Na taj način Django posjeduje mehanizam za pametno upravljanje HTML sadržajem.

5.5.3. Google JavaScript API i generiranje karata

JavaScript jedna je od najkorištenijih tehnologija pristupnog dijela weba. To je objektno bazirani programski jezik originalno stvoren kao metoda za manipulaciju HTML kodom na web stranicama. Od svoje izvorne primjene do danas, JavaScript se razvio u potpuni programski jezik

kojeg je moguće koristiti na širokom spektru platforma. Kada se JavaScript opisuje kao objektno baziran jezik, referira se na njegovu nemogućnost korištenja nasljeđivanja karakteristika klasa, za razliku od objektno orijentiranih jezika, kao što su to Java, C# ili Python. Unatoč tome, JavaScriptovo korištenje objekata kao načina enkapsulacije vrijednosti, omogućilo je, onom što je izvorno bio skriptni jezik za web, postupno korištenje na sve većem broju tehnologija. Uz korištenje mnogobrojnih popularnih radnih okvira i knjižnica te sveopću prisutnost na webu, JavaScript trenutno drži titulu jednog od najkorištenijih programskih jezika.

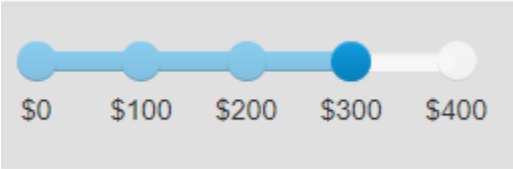
Googleov Maps JavaScript API za prikaz karata u ArchiPointu je korišten kako bi se željene vrijednosti vizualizirale kao lokacija. Jedna od glavnih funkcionalnosti ArchiPointa jest upravo prototipni prikaz lokacije nekog dijela arhivskog gradiva u danom trenutku kroz povijest. Kako bi se to ostvarilo, korišten je navedeni Googleov servis koji nosi naziv API zato što je moguće koristiti mnoštvo preddefiniranih JavaScript varijabli i funkcija koje se, kao resursi, nalaze na Internetu te se dohvaćaju širokim spektrom mogućih JavaScript izraza.

Glavna JavaScript funkcija za prikaz osnovnog tipa Google karte zove se *intiMap* te, uz pozivanje novog *Map* objekta, stvara kartu koju je moguće nadograditi dodatnim karakteristikama, kao što je centriranje prikaza, udaljenost i pokazivač lokacije. Za razliku od Googleovog servisa za geokodiranje, Maps JavaScript API, kroz funkciju za iniciranje karte, prihvaća lokacijske vrijednosti u obliku zemljopisne širine i dužine. Tako varijabla *center* za definiranje centra mape, kroz objekt *LatLng*, prihvaća već geokodirane iznose zemljopisne širine i dužine željene lokacije.

Ekstrahirane adrese u ArchiPointu geokodiraju se upravo u standardizirane decimalne iznose zemljopisne širine i dužine putem Pythonove geopy knjižnice i pohranjuju u bazu podataka. Pomoću Python prikaza za određenu web stranicu i DTL-a, željene lokacijske vrijednosti dohvaćaju se iz baze podataka i prosljeđuju u JavaScript kod pristupnog dijela aplikacije. S obzirom da se JavaScript kod može pisati unutar same HTML datoteke označen jednostavno oznakama za početak i kraj *script* bloka, vrijednosti DTL varijabli mogu se prenositi u JavaScript jednako kao i u HTML. Navedeni princip prenošenja vrijednosti uspješno prikazuje odnos JavaScripta i HTML-a kao tehnologija pristupnog dijela aplikacije te DTL-a kao tehnologije koja ih povezuje sa programskom pozadinom.

Google karta koja na ArchiPointovoj stranici za pojedini fond prikazuje lokacije tog fonda kroz dosadašnje vrijeme, koristi pokazivač koji je u osnovni prikaz karte moguće dodati kao Googleov preddefinirani JavaScript objekt imenom *Marker*. Pokazivač ne mora obavezno biti pozicioniran u samom središtu trenutnog prikaza karte, već može primati i svoje vlastite vrijednosti za zemljopisnu lokaciju. Te vrijednosti, kao i za pozicioniranje same karte, prosljeđuju se iz DTL varijabli u JavaScript varijable putem petlji u JavaScript kodu. Od izrazite je važnosti razumjeti kako se DTL vrijednosti standardno dohvaćaju i prenose samo pri učitavanju web stranice, dok se vrijednosti JavaScript varijabli mogu mijenjati dinamički bez preuzimanja novog sadržaja s poslužitelja. Iz tog razloga potrebno je ispravno koristiti prenošenje vrijednosti između tih dviju tehnologija.

Povrh svake karte za prikaz lokacije fonda nalazi se vremenska traka s označenim godinama i kliznim pokazivačem kojeg je moguće postaviti na svaku od prikazanih godina kako bi se na karti prikazala lokacija fonda u danoj godini. Vremenska traka s pokazivačem konstruirana je pomoću Bootstrap knjižnice imenom bootstrap-slider i sastoji se od HTML koda za prikaz trake te pripadajućeg JavaScript koda koji manipulira HTML-om. Lokacijske vrijednosti dinamički se dohvaćaju iz varijabli te pokazivač na isti način mijenja svoj položaj na karti, bez potrebe za ponovnim učitavanjem cijele karte. Bez asinkronih tehnologija, kao što je AJAX (engl. asynchronous JavaScript and XML), takve se tranzicije moguće ukoliko se vrijednosti iz jezika za predloške pri učitavanju stranice spreme u JavaScript varijable, te se kasnije dinamički koriste u web pregledniku.



Slika 10. Primjer bootstrap-slider trake

Izvor: Slider for Bootstrap. URL: http://seiyria.com/bootstrap-slider/. (8.09.2017.)

```
<input id="ex13" type="text" data-slider-ticks="[0, 100, 200, 300, 400]"
 data-slider-ticks-snap-bounds="30" data-slider-ticks-labels='["$0", "$100", "$200", "$300",
"$400"]'/>

var slider = new Slider("#ex13", {
    ticks: [0, 100, 200, 300, 400],
    ticks_labels: ['$0', '$100', '$200', '$300', '$400'],
    ticks_snap_bounds: 30
});
```

Kod 14. HTML i JavaScript kod primjera bootstrap-slider trake

Izvor: Slider for Bootstrap. URL: http://seiyria.com/bootstrap-slider/. (8.09.2017.)

5.6. Web poslužitelji i okruženje za objavljivanje aplikacije

5.6.1. Što je web poslužitelj

Kako bi se jasno definirao pojam web poslužitelja i njegova uloga u okruženju web aplikacije, potrebno je navesti kako web poslužitelj u kontekstu web aplikacija ne označava obavezno cijelo poslužiteljsko računalo. Web poslužitelj može se promatrati kao aplikacijski mrežni servis konfiguriran na poslužiteljskom računalu. Takav servis zadužen je za posluživanje web aplikacija klijentima na mreži otvarajući potrebne mrežne ulaze i upravljajući HTTP upitima prema samoj aplikaciji.

Mozilla Development Network objašnjava pojam web poslužitelja dvojako i navodi kako se on može odnositi na hardver, softver ili oboje. S hardverske strane, na web poslužitelj može se gledati kao na računalo spojeno na mrežu koje sadrži resurse neke web aplikacije. Kao softver, web poslužitelj je aplikacijski servis koji omogućuje i kontrolira pristup korisnika određenim resursima i dijelovima web aplikacije putem upravljanja HTTP upitima. Web poslužitelj, ako servis, može tumačiti URL u HTTP upitima te dohvaćati potrebne dijelove web aplikacije, ukoliko je to korisniku omogućeno.[18]Za potrebe daljnjeg teksta, a i kako bi se izbjeglo nejasno

[18] What is a web server?, *MDN Web Docs*. 13.04.2017. URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server. (11.09.2017.)

referenciranje, web poslužitelj kao servis bit će navođen kao HTTP poslužitelj, iz razloga što mu je to alternativni naziv i njegova osnovna funkcija.

5.6.2. Nginx i Gunicorn

Za potrebe razvoja ArchiPointa, na lokalnom računalu korišten je Django razvojni poslužitelj, kojeg je moguće izrazito konfigurirati i pokrenuti. Ukoliko se sve postavke ostave nepromijenjene, Django razvojni poslužitelj pokreće se naredbom *runserver* putem Python upraviteljske datoteke na adresi lokalnog računala (localhost ili 127.0.0.1) i mrežnom ulazu 8000. Usprkos tome što je Django razvojni poslužitelj moguće konfigurirati prema vlastitim potrebama, on je kao HTTP poslužitelj ograničen u svojim funkcijama te ne bi trebao biti korišten u produkcijske svrhe.

Aplikacija ArchiPoint objavljena je i omogućena za pristup putem Interneta pomoću kombinacije dvaju HTTP poslužitelja: Nginx i Gunicorn. Operacijski sustav na kojem su oba poslužitelja konfigurirana i pokrenuta jest Ubuntu 16.04, što čini navedeno konfiguriranje vrlo izravnim i jednostavnim. Ubuntu je operacijski sustav vrlo blizak Mintu, te je stoga prelazak iz razvojnog okruženja u produkcijsko izvršen uz samo osnovne izmjene postavki same aplikacije.

Nginx (engl. Engine X) je HTTP poslužitelj koji prihvaća HTTP upite prema aplikaciji ArchiPoint te upravlja njima kako bi bili ispravno procesirani. Nginx je ključna komponenta cjelokupnog projekta koja omogućuje ArchiPointu dostupnost putem Interneta, otvarajući mrežne ulaze na poslužitelju i povezujući određene URL-e u HTTP upitima s odgovarajućim dijelovima aplikacije.

ArchiPoint je Django web aplikacija, te stoga zahtjeva određenu poveznicu na Nginx poslužitelj. Python, u službi jezika za razvoj web aplikacija i servisa, uz standardne HTTP poslužitelje za obradu upita i posluživanje sadržaja same aplikacije, koristi i određeni tip HTTP poslužitelja koji se skraćenim imenom nazivaju WSGI (engl. web server gateway interface), što označava sučelje prema web poslužitelju. U slučaju ArchiPointa, web poslužitelj (HTTP poslužitelj) je Nginx, a WSGI poslužitelj je Gunicorn. Kao servis u potpunosti kompatibilan s web radnim okvirima kao što su Django i web2py, Gunicorn je najvećim dijelom napisan

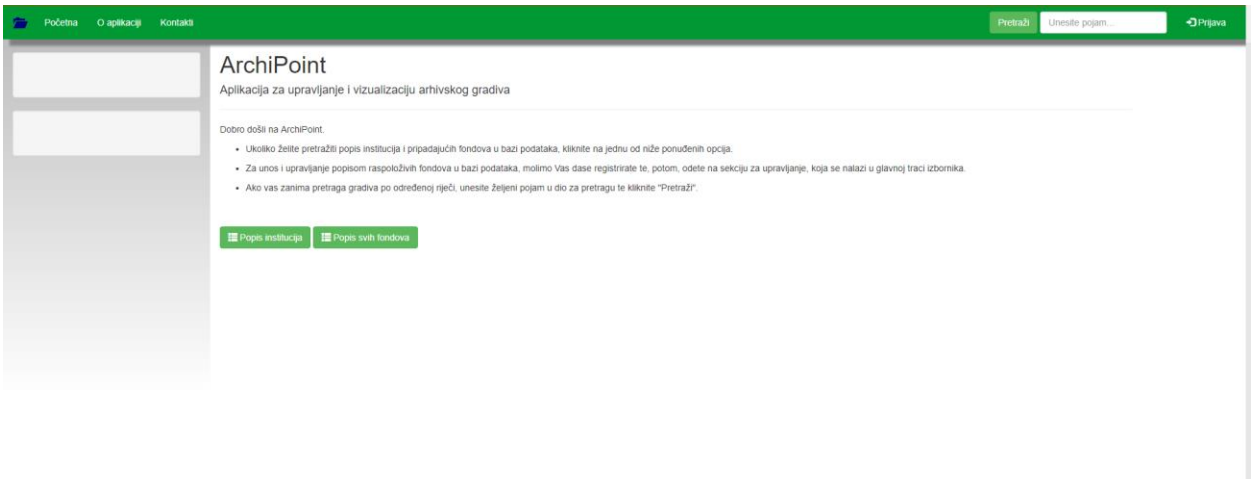
koristeći sam jezik Python i, kao takav, omogućava jednostavnu i cjelovitu komunikaciju Nginx poslužitelja s ArchiPointom kao Django aplikacijom.

Osim Nginx i Gunicorn poslužitelja, Django aplikacije mogu koristiti i neku drugu kombinaciju standardnog HTTP poslužitelja i WSGI, kao što su to Apache HTTP poslužitelj i mod_wsgi. Nginx i Gunicorn često se koriste pri posluživanju Django aplikacija iz razloga što su izuzetno dobro prilagođeni za rad s web aplikacijama napisanim u jeziku Python.

5.7. Korisničko sučelje aplikacije ArchiPoint

Vizualna manifestacija pristupnog dijela aplikacije ArchiPoint u web pregledniku jest njegovo korisničko sučelje. Ono je, svojim najvećim dijelom, namijenjeno korištenju od strane svih potencijalnih korisnika te je dostupno bez posebnih registracijskih metoda. Određeni dijelovi korisničkog sučelja dostupni su samo registriranim i prijavljenim korisnicima te, kao takvi, nisu prikazani u sučelju bez registracije.

Početnoj stranici ArchiPointa pristupa se putem adrese http://165.227.144.78/documents/, koja otvara naslovnicu s istaknutim imenom aplikacije te podnaslovom i uputama za navigaciju kroz korisničko sučelje. Navedeni URL sadrži adresu poslužitelja na kojem se cijela aplikacija nalazi te razinu */documents*, koja označava glavnu i trenutno jedinu projektnu podaplikaciju cijelog ArchiPoint projekta. Cijela aplikacija koja je u ovom radu prikazana spada pod navedenu podaplikaciju cjelokupnog ArchiPoint projekta. Takva hijerarhija ostavlja prostora i mogućnosti za daljnju nadogradnju ArchiPoint projekta novim funkcionalnostima, koje tada mogu imati drugačiju URL razinu. Dva glavna dugmeta raspoloživa za odabir na početnoj stranici daju korisniku na izbor pregledati popis uključenih institucija ili popis svih raspoloživih fondova u bazi, neovisno o pripadnosti određenoj instituciji.

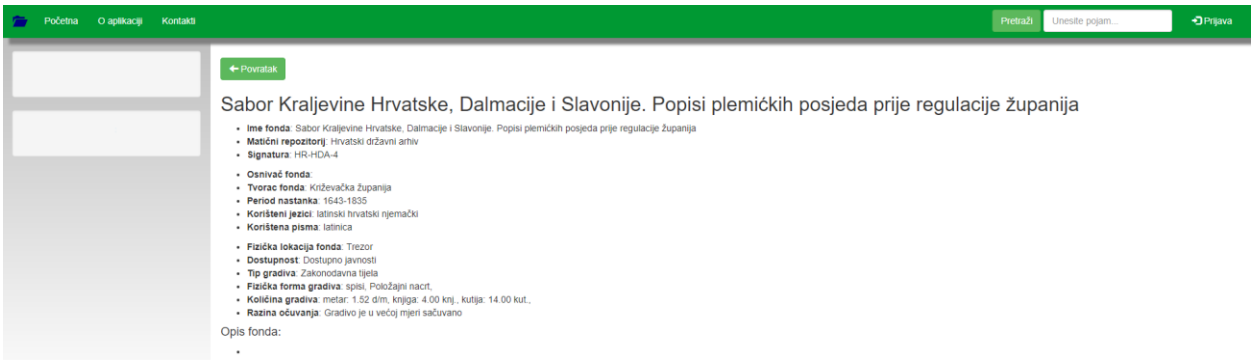


Slika 11. Početna stranica ArchiPointa

Stranica s popisom informacijskih institucija uključenih u ArchiPoint projekt sadrži brojčane oznake na kraju izbornika za svaku instituciju koje označavaju trenutni broj fondova raspoloživih za pregled unutar te institucije. Brojčani iznos fondova mijenja se dinamički, ovisno o broju dodanih ili uklonjenih fondova iz baze za određenu instituciju. Stranica svake zasebne institucije generira se također dinamički i njen sadržaj ovisi o trenutno raspoloživim podacima. Primjerice, stranica s informacijama o Hrvatskom državnom arhivu u Zagrebu trenutno sadrži adresu ustanove, kontaktne podatke, klasifikaciju ustanove, kratak opisni tekst o samom Arhivu te lokaciju Arhiva prikazanu na interaktivnoj karti. Ispod karte nalazi se popis svih trenutno raspoloživih fondova u bazi koji spadaju u nadležnost Hrvatskog državnog arhiva u Zagrebu, kao svog matičnog repozitorija.

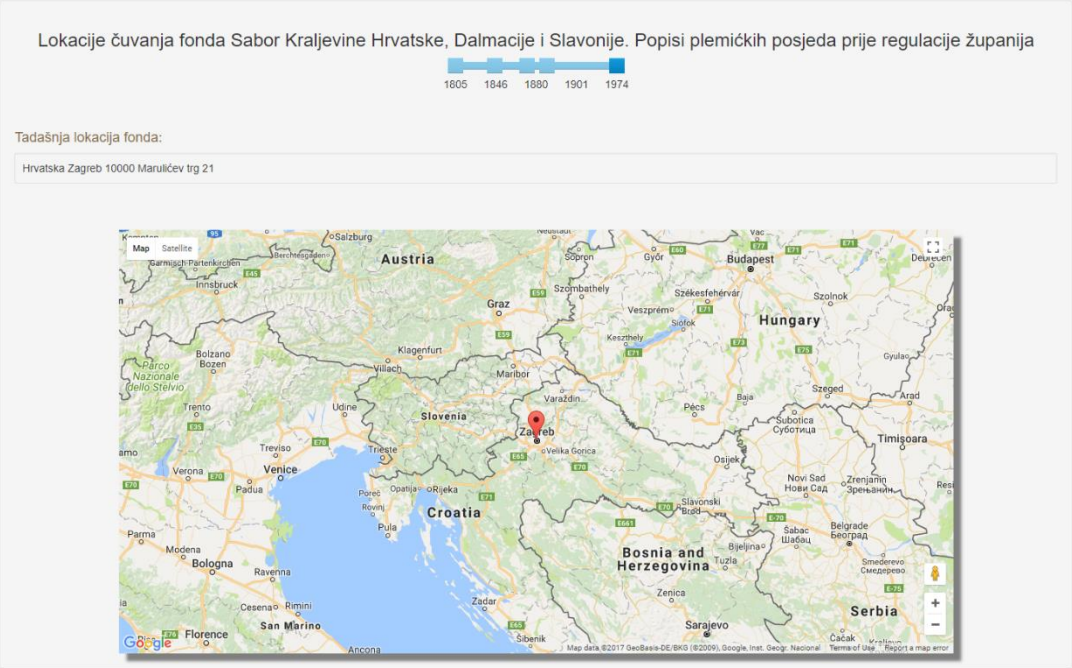
Odabirom bilo kojeg od ponuđenih fondova u danoj instituciji, otvara se stranica za prikaz interaktivnih informacija o pojedinom fondu. U trenutnoj verziji ArchiPointa, stranice fondova sadrže više detaljnih informacija o svakom pojedinom fondu negoli to rade stranice institucija o pojedinoj ustanovi. Budući da je glavnina podataka o fondovima dobivena iz XML meta podataka, svaka stranica fonda nosi otprilike isti set informacija: matični repozitorij, signaturu, tvorca fonda, period nastanka, korištena pisma i jezike u fondu te razne informacije o fizičkom stanju arhivskog gradiva unutar fonda, njegovom tipu, fizičkoj lokaciji, količini te, možda i najbitnije, njegovoj razini dostupnosti na korištenje. ArchiPoint je prototip mogućeg projekta za informacijske ustanove kojem bi jedan od glavni ciljeva bio približiti informacije od

dostupnom gradivu zainteresiranoj javnosti pa bi tako podatak o razini dostupnosti određenog gradiva na korištenje bila jedna od ključnih informacija sadržanih u aplikaciji.



Slika 12. Informacije o pojedinom fondu

Ispod detaljnih informacija o prikazanom fondu nalazi se interaktivna karta na kojoj je moguće prikazati označenu lokaciju čuvanja trenutnog fonda kroz povijest. Arhivsko gradivo može mijenjati svoju nadležnu ustanovu kroz period svog postojanja pa tako i fizičku lokaciju čuvanja. Za razliku od karte prikazane na stranici institucija, ova karta ima mogućnost dinamičkog mijenjanja položaja pokazivača lokacije, ovisno o odabranoj godini na vremenskoj traci koja se nalazi iznad nje. Povlačenjem pokazivača godine na traci, pokazivač lokacije na karti mijenja svoj položaj ovisno o adresi čuvanja arhivskog fonda u danom vremenu. Uz sam grafički prikaz, adresa čuvanja fonda ispisuje se i iznad samo karte.



Slika 13. Interaktivna karta lokacija s vremenskom crtom

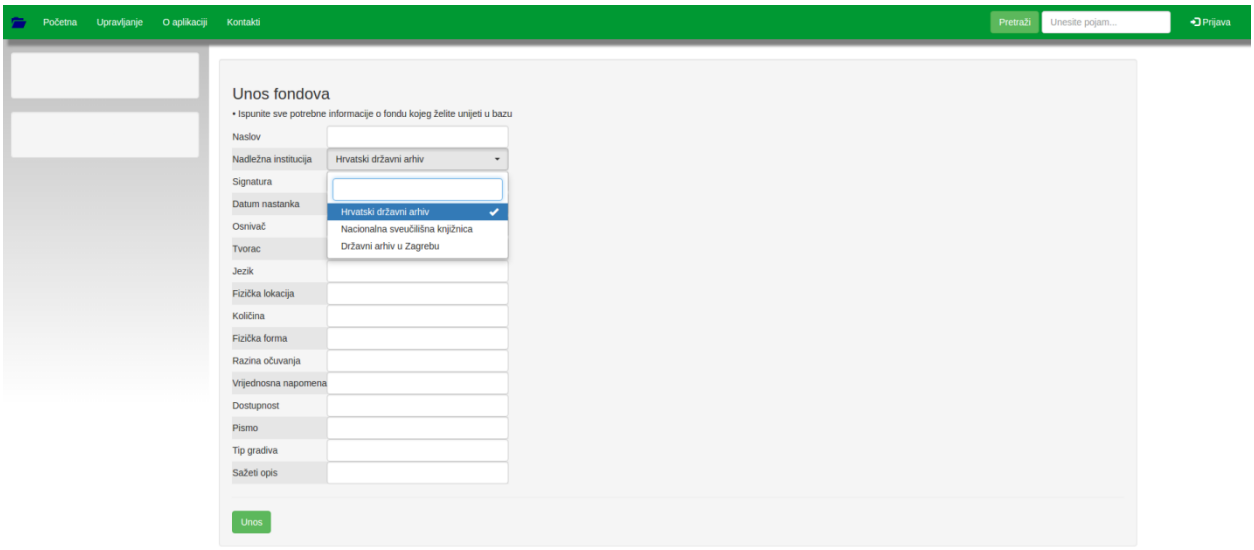
Ispod lokacijske karte nalazi se popis serija gradiva sadržanih u odabranom fondu. Stranica s prikazom detalja o svakoj pojedinoj seriji nosi nešto manje informacija naspram stranice pojedinog fonda, no to je upravo iz razloga što je većina relevantnih informacija, koje se odnose i na serije, sadržana na stranici fonda.

Navigacija kroz ostatak ArchiPointovog sadržaja i funkcija vršu se putem trake izbornika koja se nalazi pri vrhu prikaza na svakoj ArchiPoint stranici. Pomoću nje moguće je u svakom trenutku vratiti se na naslovnu stranicu ArchiPointa ili otvoriti stranice s detaljnijim opisom aplikacije i kontaktima autora aplikacije.

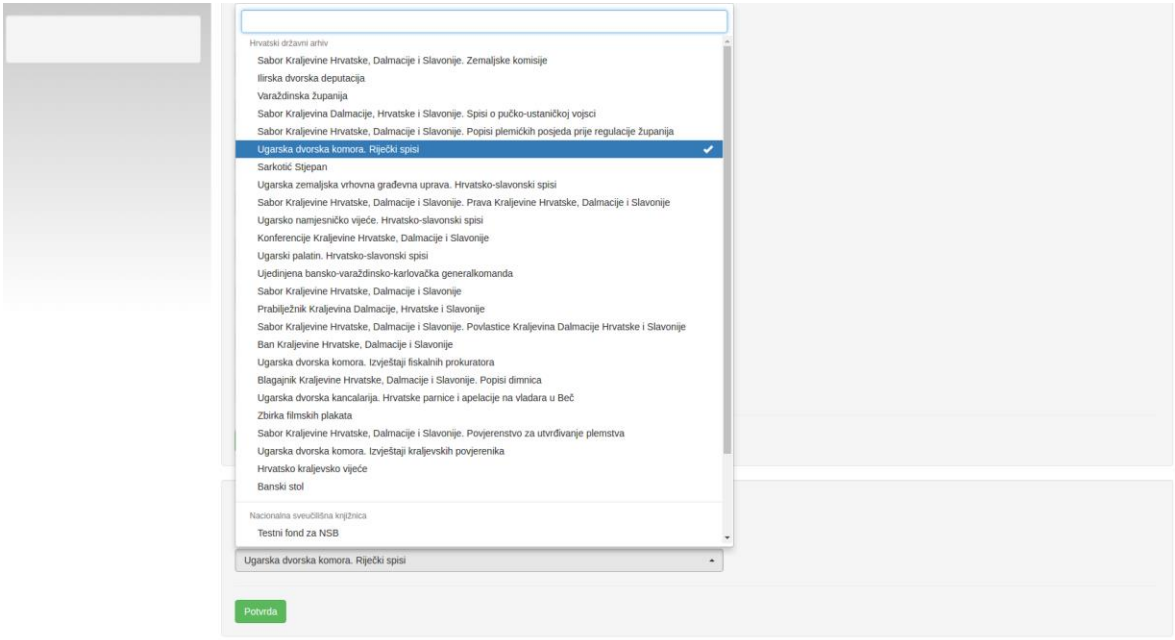
Traka izbornika također sadrži i formu za pretragu baze podataka. Ukoliko se upiše cijeli naziv fonda ili serije, jedna riječ iz naziva fonda ili serije ili samo dio sadržane riječi, ArchiPoint će vratiti popis svih arhivskih fondova i serija koje odgovaraju postavljenom upitu.

Dio trake izbornika koje skriven vodi na stranicu koja sadrži forme za unos i brisanje podataka o arhivskim fondovima iz baze podataka. Ukoliko korisnik nije registriran putem računa koji ima dodijeljena administratorska prava za vršenje brisanja i unosa podataka u bazu, taj korisnik će vidjeti traku prikazanu bez mogućnosti otvaranja stranice za upravljanje

fondovima. Sama stranica za upravljanje fondovima sadrži formu putem koje je moguće unijeti sve potrebne podatke o novom fondu u bazi podataka te formu putem koje je moguće odabrati fond za brisanje iz baze podataka. Pri brisanju, fondovi su prikazani u padajućem izborniku i grupirani po matičnim ustanovama.



Slika 15. Forma za unos meta podataka o fondu u bazu s vidljivim izbornikom "Upravljanje"



Slika 14. Forma za brisanje postojećih fondova iz baze

5.8. Daljnji razvoj aplikacije

Trenutna verzija aplikacije ArchiPoint predstavlja funkcionalan prototip koji služi kao svojevrsna potvrda koncepta. Daljnji razvoj aplikacije moguć je u više specifičnih smjerova iz razloga što trenutna verzija sadrži nekoliko različitih primjena od kojih svaka, ukoliko bi bila razvijana u svom smjeru, može rafinirati primjenu aplikacije u svom određenom smjeru. Druga opcija jest daljnji uravnotežen razvoj svih funkcionalnosti i stvaranje kvalitetne sveobuhvatne arhivske web platforme.

Korištenje nekog drugog konsenzusnog XML imenskog prostora omogućilo bi uključivanje više različitih tipova informacijskih ustanova u projekt, od kojih bi svaka mogla imati svoje informacije prikazane na način specifičan toj ustanovi, no i dalje standardiziran na razini cijele platforme. Jadan od mogućih načina za postizanje takvog okruženja jest stvaranje i korištenje XML standarda sa širokim spektrom mogućih elemenata, dok je drugi mogući način korištenje jednog specifičnog i prilagođenog imenskog prostora za svaki pojedini tip ustanove, te implementacija provjere kategorije ustanove kako bi aplikacija znala dalje procesirati dobivene metapodatke na ispravan način.

Vizualizaciju zemljopisne lokacije određenog gradiva kroz povijest moguće je nadopuniti dinamičkim prikazom informacija o okolnostima u kojima se gradivo u određenom trenutku nalazilo te informacijama o uzrocima promjene lokacije. Takva bi nadogradnja iziskivala dostavu informativnijeg seta metapodataka ili korištenje podataka iz više odvojenih izvora, te povezivanje i prikaz po zajedničkim oznakama.

Aktualna sekcija za upravljanjem metapodacima unutar baze podataka može biti proširena grafičkim prikazom stanja u bazi te kontroliranim dodavanjem novih metapodataka, koje bi prilagođavalo formu za unos ovisno o tipu ustanove, tipu unosa ili pravima pojedinog korisničkog računa.

Arhivi, kao informacijske ustanove, nisu jedini vrijedni veće eksponiranosti u zajednici i porasta korištenja, ali su uzeti kao najbolji radni primjer zbog svog opsežnog korištenja definiranih standarda pri označavanju metapodataka. Ukoliko bi se potencijalni projekt dovođenja svih hrvatskih arhiva na jednu standardiziranu web lokaciju pokazao uspješnim, logičan sljedeći korak bio bi rad na proširenju aplikacije, kako bi bila u mogućnosti obrađivati gradivo i poslovanje drugih tipova informacijskih ustanova.

6. Zaključak

Web aplikacija ArchiPoint za cilj ima objediniti relevantne informacije o svom arhivskom gradivu dostupnom javnosti kroz što više ustanova, postavljajući ga na jedno mjesto i koristeći jedinstveni prikaz informacija. Jedan takav uspjeli projekt omogućio bi veću eksponiranost, te u konačnici i veću popularnost, arhivskih ustanova i korištenje njihovog gradiva u svrhu znanstvenih istraživanja.

Korištenjem unificiranog zapisivanja metapodataka u EAD arhivskom standardu i primjenom ArchiPoint web aplikacije, javnost bi na jednom mjestu imala cjelokupan katalog svih arhivskih ustanova uključenih u projekt te detaljne informacije o njihovom gradivu i dostupnosti, što bi moglo uvelike doprinijeti porastu korištenja tog gradiva u kvalitetnijim znanstvenim istraživanjima.

Konačni cilj ArchiPointa bio bi uključivanje i drugih informacijskih institucija u svoj katalog, kao što su, primjerice, knjižnice. Arhivi su korišteni kao uspješan radni primjer u prototipu aplikacije zbog dostupnosti EAD-a, koji je izrazito iskoristiv i funkcionalan arhivski standard za označavanje matapodataka.

S obzirom da je EAD originalno arhivski standard, potencijalnim korištenjem nekog drugog srodnog i standardiziranog načina zapisivanja informacija, moguće je dalje proširiti aplikaciju da radi s drugačijim tipovima gradiva, a također i s više vrsta informacijskih ustanova.

Kao već postojeći primjer mrežnog kataloga arhivskog gradiva moguće je promatrati i NAIS, koji služi kao registar arhivskih fondova i zbirki Republike Hrvatske te središnja nacionalna evidencija arhivskoga gradiva koju vodi Hrvatski državni arhiv[19]. Na području zagrebačkih gradskih knjižnica, moguće je promatrati također ekstenzivan katalog Knjižnica grada Zagreba, koji pretražuje dostupnu građu u svim knjižnicama uključenima u projekt. Dovođenjem već takva dva sustava na jedno centralno mrežno mjesto koristeći neki ekstenzivan način opisivanja gradiva koji sadrži dovoljno mogućnosti, značilo bi već znatan korak prema onome k čemu ArchiPoint teži.

[19] Hrvatski državni arhiv, Nacionalni arhivski informacijski sustav. URL: http://arhinet.arhiv.hr/default.aspx. (13.09.2017.)

Moderno društvo provodi velik dio svog vremena na Internetu. Spajanjem suvremenog načina potrage za informacijama i usluga pregleda i korištenja gradiva tradicionalnih informacijskih institucija, kao rezultat je moguće dobiti kvalitetnije rezultate znanstvenih istraživanja te veću prisutnost informacijskih institucija u zajednici.

7. Literatura

1. Agić, Ž. Pristupi ovisnosnom parsanju hrvatskih tekstova. Doktorski rad. Filozofski fakultet Sveučilišta u Zagrebu. URL: http://darhiv.ffzg.unizg.hr/2337/1/587199.zagic_phd.pdf. (Zagreb, 2012.)
2. Council of Australasian Archives and Records Authorities. Digital archiving in the 21st century. URL: http://www.caara.org.au/wp-content/uploads/2010/03/DigitalArchiving21C.pdf. (9.2006.)
3. Django overview. *Django.* URL: https://www.djangoproject.com/start/overview/. (16.09.2017.)
4. Hrvatski državni arhiv, Istražite gradivo – Rad u čitaonicama. *Hrvatski državni arhiv.* URL: http://www.arhiv.hr/Istrazite-gradivo. (10.09.2017.)
5. Hrvatski državni arhiv, Istražite gradivo. *Hrvatski državni arhiv.* URL: http://www.arhiv.hr/Istrazite-gradivo. (10.09.2017.)
6. Linux Mint, URL: https://www.linuxmint.com/about.php. (16.09.2017.)
7. Minarik, T. (2004). Linux. *Ekscentar, (6),* 43-44. URL: http://hrcak.srce.hr/11247.
8. Narodne Novine. Pravilnik o korištenju arhivskoga gradiva. (1999.)
9. os - Miscellaneous operating system interfaces. *Python documentation.* URL: https://docs.python.org/2/library/os.html. (16.09.2017.)
10. Python.org. URL: https://www.python.org/about/. (2.09.2017.)
11. Python's OS Module. *Python For Beginners.* URL: http://www.pythonforbeginners.com/os/pythons-os-module. (16.09.2017.)
12. Richardson, L. Beautiful Soup. URL: https://www.crummy.com/software/BeautifulSoup/. (16.09.2017.)
13. Ronacher, A. Switching from other Template Engines, *Jinja2 Documentation,* URL: http://jinja.pocoo.org/docs/2.9/switching/. (7.09.2017.)
14. Slider for Bootstrap. URL: http://seiyria.com/bootstrap-slider/. (8.09.2017.)
15. Stančić, H., Je li pohrana podataka u oblaku pravo rješenje za arhivsku struku?. *Netokracija*. URL: http://www.netokracija.com/arhivska-pohrana-oblak-sector-118085. (5.5.2016.).
16. The Linux Foundation, What is Linux?, *Linux.com,* URL: https://www.linux.com/what-is-linux. (3.09.2017.)

17. Web Frameworks for Python, 11.08.2017., *Python.org*, URL: https://wiki.python.org/moin/WebFrameworks. (4.09.2017.)
18. What is a web server?, *MDN Web Docs*. 13.04.2017. URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server. (11.09.2017.)
19. What's the difference Between the Front-End and Back-End?. *Pluralsight*. URL: https://www.pluralsight.com/blog/film-games/whats-difference-front-end-back-end. (28.01.2015.)

8. Popisi pomoćnih elemenata

8.1. Popis slika

Slika 1. Pokretanje Django razvojnog poslužitelja putem Terminala 21
Slika 2. Django Admin 23
Slika 3. GRUB pokretač 26
Slika 4. Instacija Djanga u Command Promptu putem pip-a (Windows) 28
Slika 5. Instalacija Djanga u Terminalu putem apt-a (Linux) 28
Slika 6. Pregled Linux paketa kroz Synaptic 30
Slika 7. Izvršavanje upravljačke Python skripte u Djangu 33
Slika 8. Djangova analiza pogrešaka 36
Slika 9. Upravljanje Python listom 47
Slika 10. Primjer bootstrap-slider trake 60
Slika 11. Početna stranica ArchiPointa 64
Slika 12. Informacije o pojedinom fondu 65
Slika 13. Interaktivna karta lokacija s vremenskom crtom 66
Slika 14. Forma za unos meta podataka o fondu u bazu s vidljivim izbornikom "Upravljanje" .. 67
Slika 15. Forma za brisanje postojećih fondova iz baze 67

8.2. Popis kodnih isječaka

Kod 1. Python funkcija za generiranje jedinstvenog identifikatora 17
Kod 2. Java metoda za generiranje jedinstvenog identifikatora 17
Kod 3. Datoteka settings.py 35
Kod 4. Korištenje try / except bloka u Pythonu 39
Kod 5. Postavke za Jinja2 jezik za predloške 40
Kod 6. Razlika između DTL i Jinja2 41
Kod 7. Primjer funkcije prikaza 42
Kod 8. Uspodedba Java i Python nasljeđivanja u klasama 43
Kod 9. Klasa SeparatedValuesField 49
Kod 10. Primjer hijerarhije XML elemenata s dodatnim atributima 52
Kod 11. Primjer dohvaćanja vrijednosti pomoću Beautiful Soupa 53
Kod 12. Dohvaćanje programskih knjižnica putem CDN-a 55
Kod 13. Predložak institutions.html i primjer sadržaja za proširenje 57
Kod 14. HTML i JavaScript kod primjera bootstrap-slider trake 61

8.3. Popis tablica

Tablica 1. Tipovi Django polja u modelima 46

9. Prilog – Lokacija objavljene aplikacije ArchiPoint

URL: http://165.227.144.78/documents/