Razvoj programskog rješenja za mapiranje shema metapodataka

Matanović, Antun

Master's thesis / Diplomski rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Humanities and Social Sciences / Sveučilište Josipa Jurja Strossmayera u Osijeku, Filozofski fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:142:121755

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

FILOZOFSKI FAKULTET
SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

Repository / Repozitorij:

FFOS-repository - Repository of the Faculty of Humanities and Social Sciences Osijek

zir.nsk.hr

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište J.J. Strossmayera u Osijeku

Filozofski fakultet u Osijeku

Diplomski studij informatologije

Antun Matanović

Razvoj programskog rješenja za mapiranje shema metapodataka

Diplomski rad

Mentor: doc. dr. sc. Boris Bosančić

Osijek, 2016.

Sveučilište J.J. Strossmayera u Osijeku

Filozofski fakultet

Odsjek za informacijske znanosti

Diplomski studij informatologije

Antun Matanović

Razvoj programskog rješenja za mapiranje shema metapodataka

Diplomski rad

Društvene znanosti, Informacijske i komunikacijske znanosti, Knjižničarstvo

Mentor: doc. dr. sc. Boris Bosančić

Osijek, 2016.

Sažetak

Rad se bavi opisom izrade programskog rješenja za mapiranje shema metapodataka. Na početku rada izložen je pregled teorijskih postavki metapodataka uz poseban osvrt na Dublin Core i MODS sheme metapodataka koje se u okviru programskog rješenja mapiraju. Nakon toga dan je pregled korištenih programskih jezika za razvoj programskog rješenja, od kojih je PHP najvažniji i pomoću kojega je razvijena glavna logika mapiranja. Uz PHP korišten je i JavaScript, HTML te CSS. U drugom dijelu rada opisana je izrada programskog rješenja za mapiranje shema metapodataka koje od korisnika zahtjeva prilaganje XML dokumenata s podržanim shemama metapodataka za mapiranje. Tijekom procesa mapiranja uzeti su u obzir ISO standardi za kôdove jezika i država kao i formatirani podaci iz normativnih datoteka Kongresne knjižnice za autorska imena. Nakon završetka postupka mapiranja krajnji korisnik ima mogućnost uvida u sve greške koje su se eventualno nalazile u XML dokumentima i zbog kojih mapiranje nije moglo biti provedeno. Ako je postupak mapiranja uspješno dovršen, ostavljena je mogućnost pregleda provedenog postupka mapiranja u pregledniku uz mogućnost preuzimanja izlaznih datoteka. Razvijeno programsko rješenje je nadogradivo, a s obzirom da je programski kôd odlukom njegova autora javno dostupan – omogućeno je nastaviti ga razvijati i u praksi.

Ključne riječi: Dublin Core, MODS, PHP, mapiranje shema metapodataka

Sadržaj

1. Uvod 1
2. Metapodaci 2
2.1. Interoperabilnost 5
2.2. Sheme metapodataka 7
2.2.1. Dublin Core 7
2.2.2. MODS 9
2.3. Mapiranje shema metapodataka 10
4. Programski jezici 13
4.1. PHP 13
4.2. JavaScript 15
5. Izrada programskog rješenja za mapiranje shema metapodataka 17
5.1. Svrha i ciljevi izrade programskog rješenja za mapiranje shema metapodataka 17
5.2. Pregled 18
5.3. Prilaganje datoteka 22
5.4. Prije procesa mapiranja 24
5.5. Status mapiranja 25
5.6. Početak procesa mapiranja 27
5.6.1. Kontrole programskog kôda 29
5.6.2. Mapiranje Dublin Core sheme u MODS 32
5.6.3. Mapiranje MODS-a u Dublin Core 40
5.7. Završetak procesa mapiranja 43
6. Zaključak 44
Literatura 46
Prilozi 50

1. Uvod

Suvremeni život bio bi u potpunosti nezamisliv bez metapodataka. Iako ih prosječna osoba, koja nije informacijski stručnjak nije niti svjesna, upravo ona generira izuzetno veliki broj informacija, a samim time i metapodataka. S tim u vezi, može se reći kako zapravo metapodaci postoje otkad je čovjek počeo organizirati informacije, no tek u nedavnoj povijesti oni se počinju teorijski razmatrati te postaju fokus profesionalnog istraživanja. Shodno tomu, može se reći kako svaka grupa istraživača ima neku svoju definiciju metapodataka i način na koji promatra ovo područje. Knjižnična i informacijska znanost metapodatke poznaje otkad i sama postoji. Njezine glavne djelatnosti nabave, organizacije, obrade, diseminacije, korištenja i vrednovanja knjižnične građe usko su vezane uz ove "podatke o podacima". No kako je danas informacijska tehnologija duboko ukorijenjena u naše društvo, poželjno je da različiti dijelovi ovog sustava mogu učinkovito komunicirati, što bi bilo vrlo teško bez postavljenih standarda. Upravo navedene standarde predstavljaju sheme metapodataka koje definiraju konkretne načine opisa nekog informacijskog objekta. Kako standard nije samo jedan, mapiranje shema metapodataka iz jedne u drugu odnosno pronalaženje međusobno semantički ekvivalentnih elemenata različitih shema metapodataka predstavlja izazov za knjižnice u budućnosti. Izazov je još veći, ako želimo postaviti automatizirani sustav koji će odrađivati ovaj proces bez ili sa što manje ljudske intervencije, jer generalno gledano, radi se o dugotrajnom, repetitivnom poslu za što su računala sposobna, bez prevelikog trošenja vremena i sredstava, dok po pitanju ljudskih resursa to nije slučaj. Ovaj rad odgovara na pitanje zašto su programska rješenja za mapiranje shema metapodataka danas imperativ, koje su njihove implikacije na društvo te što je potrebno za razvoj jednog takvog sustava. Na postavljena pitanja odgovara se konkretnim razvojem programskog rješenja za mapiranje shema metapodataka koje se nalazi u domeni slobodnog korištenja i modificiranja programskog kôda.

Svrha rada je razvoj programskog rješenja koje će biti jedinstveno u svijetu, s obzirom da trenutno ne postoje slični alati na mreži. Ciljevi rada su izrada prednjeg (engl. *front-end*) i pozadinskog (engl. *back-end*) sustava koji će biti u potpunosti funkcionalan prilikom mapiranja, izrada kontrola u pozadinskom sustavu koje će dopustiti mapiranje samo validnih zapisa shema metapodataka (prema pripadnim XML Schemama) te krajnji prikaz statusa mapiranja uz mogućnost preuzimanja izlaznih datoteka provedenog mapiranja. Kao inicijalne sheme metapodataka koje će se mapirati odabrana je Dublin Core i MODS (*Metadata Object Description Schema*) shema metapodataka.

U skladu s tim, rad se u prvom poglavlju bavi teorijskim postavkama metapodataka, raspravlja o interoperabilnosti različitih sustava i postupcima mapiranja uz povijesni i teorijski pregled shema metapodataka s posebnim naglaskom na Dublin Core i MODS sheme metapodataka koje su služile kao temelj mapiranja u razvijenom programskom rješenju. Sljedeće poglavlje donosi pregled korištenih programskih i označiteljskih jezika za razvoj potrebitog programskog rješenja. Objašnjavaju se neki od temeljnih pojmova koji su se u praktičnom razvoju koristili. Nadalje, izložen je opis izrade razvoja programskog rješenja. Samo programsko rješenje podijeljeno je na dijelove koje sâm korisnik vidi prilikom pristupa programskom rješenju. U ovom dijelu rada objašnjeno je kako ostvariti mapiranje metapodataka koristeći programsko rješenje uz osvrt na sve njegove sastavne dijelove. Isti dio rada donosi i pregled svega onoga što krajnji korisnik programskog rješenja ne vidi: programski kôd i logiku koja se izvršava u pozadini. On je tehnički orijentiran te uz konkretan prikaz programskog kôda, najveći dio odnosi se na samu logiku koja je napravljena kako bi se omogućilo mapiranje shema metapodataka. Pritom valja naglasiti da je specifikaciju mapiranja (engl. *crosswalks*) shema metapodataka (iz Dublin Corea u MODS i obratno), prema kojoj je kreirano programsko rješenje, izradila Kongresna knjižnica u Washingtonu.

2. Metapodaci

Samo korištenje termina metapodaci potječe iz računalne znanosti gdje termin meta označava "o (nečemu)", kao u primjeru metajezik što bi značilo jezik kojim se opisuje drugi jezik tj. jezik o jeziku, samim time bi doslovno značenje metapodataka značilo podatak o podacima.[1] Iako gotovo svaka grupa stručnjaka ima svoju definiciju metapodataka i naprosto se može zaključiti da ne postoji pogrešno viđenje njih samih, u ovom kontekstu metapodaci se promatraju kao strukturirane informacije koje opisuju, objašnjavaju, lociraju ili na bilo koji drugi način čine jednostavnijim pretraživanje, korištenje ili upravljanje informacijskim objektima.[2] Informacijski objekt je sve ono stvoreno od strane čovjeka ili informacijskog sustava, a može se sastojati i od više dijelova koji čine neku cjelinu a kao tipičan primjer može se navesti baza podataka. Kao takav, neovisno o njegovoj fizičkoj ili intelektualnoj formi, informacijski objekt sadrži tri najbitnija svojstva. To su:

- sadržaj- ono što informacijski objekt sadrži ili što sâm jest; u ovom smislu, sadržaj je

[1] Usp. Caplan, Priscilla. Metadata fundamentals for all librarians. Chicago: American Library Association, 2003. Str. 1.
[2] Usp. Undestanding metadata. Bethesda: NISO Press, 2004. Str. 1. URL: www.niso.org/publications/press/UnderstandingMetadata.pdf (2016-10-10)

intrizično svojstvo informacijskog objekta;

- kontekst -tko, što, zašto, gdje i kako je povezan s informacijskim objektom; u ovom smislu, kontekst je ekstrizično svojstvo informacijskog objekta;
- struktura - odnosi se na formalni niz asocijacija unutar ili među pojedinim informacijskim objektima i stoga može biti intrizična, ekstrizična ili oboje.[3]

S obzirom na iznimnu raznolikost informacijskih objekata, postoje i velika razilaženja po nekim pitanjima. Primjerice, prožimaju li se metapodaci samo kroz digitalne informacijske objekte ili se oni mogu naći i u fizičkom okruženju. Primjerice, navod da je tradicionalni knjižnični katalog forma metapodataka jer se za njegovu izradu koriste metapodatkovni standardi[4] je neosporan, dok krovna organizacija knjižničarstva IFLA (*International Federation of Library Associations and Institutions*) smatra da se metapodaci ipak odnose samo na mrežne izvore.[5] S obzirom na ovu različitost u perspektivama promatranja metapodataka, kategorizacija prema vrstama i ključnim funkcionalnostima ipak donosi njihovo bolje razumijevanje. Prema jednoj podjeli postoje:

- administrativni metapodaci, koji se koriste kod upravljanja i administriranja informacijskih izvora i zbirki;
- opisni metapodaci, koji se koriste za identifikaciju i opis resursa / izvora; kao tipičan primjer opisnih metapodataka može se navesti kataložni zapis;
- metapodaci za zaštitu, koji se odnose na sve one aspekte arhiviranja i provođenja plana zaštite informacijskog objekta kao što je njegovo fizičko stanje, mjere poduzete za očuvanje fizičke i digitalne manifestacije objekta te promjene koje su se dogodile uslijed tih procesa;
- tehnički metapodaci, kao što je programska ili hardverska dokumentacija;
- uporabni metapodaci, koji se odnose na način i razinu korištenja izvora od strane korisnika.[6]

Za sve metapodatke karakteristična je određena sintaksa. Svakako jedna od najpoznatijih metapodatkovnih sintaksi u okviru knjižnične i informacijske znanosti je MARC (engl. *Machine-Readable Cataloging*) koji je nastao u Kongresnoj knjižnici sredinom 60-ih godina prošlog stoljeća. Iako je MARC izrazito složen strojno čitljivi format pohrane kataložnih zapisa i ima jako puno utjecaja, važno je napomenuti kako on, sam po sebi, ipak nije shema

[3] Usp. Gilliland, Anne J. Setting the stage. // Introduction to metadata / urednica Murtha Baca. 3. izd. Los Angeles: Getty Publications, 2016. URL: http://www.getty.edu/publications/intrometadata/setting-the-stage/ (2016-10-10)
[4] Usp. Understanding metadata. Nav. dj., str. 1.
[5] Usp. IFLA. Digital libraries: Metadata resources, 2005. URL: http://www.ifla.org/node/9337 (2016-09-20)
[6] Usp. Gilliland, Anne J. Nav. dj.

metapodataka u pravom smislu riječi (jer semantičko značenje elemenata preuzima od ISBD-a) i zato ga se ovdje promatra samo kao sintaksu za strojno čitljive kataložne kartice.[7] Sljedeća sintaksa s kojom je, barem na razini prepoznavanja, upoznata većina informacijskih stručnjaka je HTML, o kojem će kao označiteljskom jeziku biti riječi u nastavku rada, dok je na ovom mjestu bitno napomenuti kako imetapodaci mogu biti ugrađeni u sâm HTML, a tipičan primjer izgleda ovako:

```
<meta name="author" content="Antun Matanović">
```

I njihovo definiranje u HTML dokumentu nije beznačajno jer ih internetski servisi i tražilice prepoznaju, a pored toga, čak je moguće i koristiti sheme metapodataka, primjerice Dublin Core na sljedeći način:

```
<meta name="DC.Creator"content="Antun Matanović">
<link rel="schema.DC" href="http://purl.org/DC/elements/1.1/">
```

Uobičajena je konvencija da se iza naziva metapodatka koji ima indikaciju sheme, navede tag *link* koji će definirati prefiks sheme i putanju do njezine definicije.[8] Najjednostavniji način provjere ima li neka mrežna stranica uključene metapodatke je pogledati u njezin izvorni kôd. U velikoj većini slučajeva ne postoji previše razlike s obzirom na preglednik ili samu mrežnu stranicu tako da je moguće pozicionirati miš bilo gdje na mrežnoj stranici i desnim klikom otići na opciju *Pogledaj izvor stranice* gdje se na početku unutar elementa <*head*> nalaze metapodaci, kao u primjeru s napravljenog programskog rješenja (Slika 1). S obzirom na statičan položaj i jedinstveno mjesto gdje se nalaze u okviru stranice, za računalne programe je jednostavno njihovo preuzimanje.

Uz RDF (engl. *Resource Description Framework*) kao sljedeći primjer sintakse metapodataka, čija je primarna svrha ne biti samo strojno čitljiv nego i strojno razumljiv, postoji i SGML koji je zapravo metajezik. Njegova je uloga definirati generička sintaktička pravila, primjerice, da se nazivi tagova pišu unutar izlomljenih zagrada, no SGML ne definira niti jedan poseban tag. To postiže DTD (engl. *Document Type Definition*) kojim se zapravo kreiraju aplikacije SGML-a, a jedna od njih je upravo HTML sa svojim pravilima i specifičnim tagovima. No za ovaj rad najvažniji primjer sintakse je XML, koji je usko vezan uz SGML, no s puno više pravila i manje svojstava što ga čini striktnijim i zapravo jednostavnijim označiteljskim jezikom za računalno procesuiranje.[9] Iako može izgledati kontradiktorno, ali što je nešto strože postavljeno,

[7] Usp. Caplan, Priscilla. Nav. dj., str. 12.
[8] Usp. Isto. Str. 16.
[9] Usp. Isto. Str. 17-20.

zapravo je jednostavnije za procesuiranje. Razlog tomu su konkretno definirana pravila koja računalni program mora pratiti, dok manje pravila znači i više izuzetaka koje računalni program mora pokušati riješiti, a tu može doći do pogrešaka. Primjerice, pravila vezana uz pisanje atributa u nekom elementu su uvijek ista, način na koji se otvaraju i zatvaraju tagovi, ili kako se hijerarhijski tagovi ugnježđuju među sobom, sve to omogućava standardnoj logici ugrađenoj u XML Parser (softverski program zadužen za procesuiranje XML dokumenata) da uspješno procesuira XML dokument. Ako se tijekom procesuiranja dogodi greška, vrlo je jednostavno zaključiti da sintaksa XML-a nije bila dobra te da stoga dokument nije validan. U suprotnom bilo bi potrebno prolaziti kroz nebrojene uvjete koji provjeravaju radi li se o nekom izuzetku, što samu obradu čini sporijom. To, danas kada se želi dobiti informacija što prije, nije poželjno. U skladu s tim, gotovo svaki programski jezik ima već unaprijed definirane klase koje programer može koristiti za procesuiranje XML dokumenata ili za njihovo kreiranje što će se pokazati izuzetno korisnim u ovom radu. Kako je prethodno bilo govora o zasebnim XML datotekama koje se procesuiraju, bitno je napomenuti na koje načine se metapodaci vežu uz određeni dokument. Postoje tri mogućnosti; prema prvoj, metapodaci su dio dokumenta kao što su to unutar HTML dokumenta; sljedeći način je da se metapodaci nalaze u zasebnoj datoteci što je za ovaj rad važno naglasiti; posljednji način odnosi se na automatsko ekstrahiranje metapodataka iz sadržaja samog dokumenta. Prva dva načina postavljaju eksplicitno definirane metapodatke dok je u posljednjem slučaju velika mogućnost greške i teško je za sigurnošću tvrditi da je provedeni postupak u potpunosti točan. Jedan od postupaka koji provodi automatsko ekstrahiranje metapodataka je „rudarenje teksta“ (engl. *data mining*) koje je trenutno izuzetno popularno i tema je mnogih istraživačkih radova.[10] S obzirom na različitost sintaksi i formata metapodataka u kojemu se oni nalaze, bitno je da su različiti sustavi u mogućnosti komunicirati, odnosno da su interoperabilni.

2.1. Interoperabilnost

Interoperabilnost je mogućnost da se informacije, nastale u jednom kontekstu, mogu koristiti u drugome na što je moguće više automatiziran način.[11] Interoperabilnost tako predstavlja jedan od ključnih koncepata u digitalnom okruženju. Metapodaci nastali u jednom kontekstu, posebice

[10] Usp. Witten, Ian H. … [et al.]. Importing documents and metadata into digital libraries: Requirements analysis and an extensible architecture. // Research and Advanced Technology for Digital Libraries: 6th European Conference, ECDL 2002, Rome, Italy, September 16-18, 2002. / Thanos, Costantino (Ed.). Berlin: Springer, 2002. Str. 391-392. URL: http://www.cs.waikato.ac.nz/~ihw/papers/02-IHW-etal-Importingdocuments.pdf (2016-09-20)
[11] Usp. Rust, Godfrey; Bide, Mark. The <indecs> metadata framework: Principles, model and data dictionary, 2000. URL: http://www.doi.org/topics/indecs/indecs_framework_2000.pdf. Citirano prema: Duval, Eric. Metadata standards: What, who & why. // Journal of Universal Computer Science 7, 7(2001), str. 592. URL: http://www.jucs.org/jucs_7_7/metadata_standards_what_who/Duval_E.pdf (2016-09-20)

onom tehničkom, u kojemu jedan server mora poslati svoje podatke klijentu, tek je jedan od aspekata konteksta o kojem je ovdje riječ. Za metapodatke je tehnički kontekst jedan od najjednostavnijih koji može biti interoperabilan, a puno je teže uspostaviti komunikaciju i dijeliti informacije u okviru lingvističkog, socijalnog ili kulturološkog konteksta u kojem su metapodaci nastali. Zato je bitno napomenuti: kada se u ovom radu govori o interoperabilnosti, pritom se misli „na što je moguće više automatiziran način“, a što znači da se interoperabilnost uspostavlja stupnjevito i da može biti uspostavljena na većem ili manjem stupnju.[12] Dva sustava mogu komunicirati više ili manje uspješno uz napor i jedne i druge strane da se što bolje razumiju, a što je taj napor manji govorimo o većoj razini interoperabilnosti.

Kada se govori o interoperabilnosti, prije svega, ona se promatra na četiri razine. Prva razina je mrežni protokol čiji su standardi TCP/IP i HTTP koji omogućuje komunikaciju web preglednika i poslužitelja.[13] Govoreći o protokolima Mirna Willer se posebno osvrće na ISO 23950 odnosno Z39.50 koji omogućuje pretraživanje heterogenih baza podataka putem jednog korisničkog sučelja, a najčešće se koristi za pronalaženje bibliografskih podataka.[14] Druga razina je razina strukture zapisa odnosno sintaksa o kojoj je već bilo govora što je za interoperabilnost izuzetno bitno, primjerice, da zapisi rabe jednaku sintaksu poput XML-a. Sljedeća razina je semantika odnosno semantički označitelji sadržaja zapisa. To je sustav tagova koji se odnose na određene elemente podataka. Ako dva sustava ne koriste istu semantiku, dolazi do procesa konverzije koji će se u kontekstu shema metapodataka nazivati mapiranjem, i gdje se primjerice zapis u Dublin Core-u prevodi u MODS. Naposljetku, tu je i četvrta razina interoperabilnosti koja se odnosi na sam element metapodataka, odnosno sadržaj podatka zapisa. To je njegova vrijednost koja se identificira zasebno uključujući i dodatne elemente koji se u ovom slučaju nazivaju atributi.[15]

Vođenje računa o svim razinama interoperabilnosti govori o uspješnosti sustava u komunikaciji s drugim informacijskim sustavima. U razvoju programskog rješenja o svim razinama interoperabilnosti je vođena posebna briga. Tako se uspostavilo interoperabilno rješenje između DC i MODS sheme metapodataka u oba smjera, no isto tako, i sam sustav nije izoliran od drugih. Njegova prednost je u komunikaciji s drugima, primjerice s Kongresnom knjižnicom u kojoj pretražuje normativne datoteke ili ISO standardima za kodove država i jezika. No prije nego

[12] Usp. Duval, Eric. Nav. dj. Str. 592-593.
[13] Poslužitelj je isto što i server.
[14] Usp. Willer, Mirna. Interoperabilnost sadržaja metapodataka. // 4. seminar Arhivi, knjižnice, muzeji: mogućnosti suradnje u okruženju globalne informacijske infrastrukture / uredile Mirna Willer i Tinka Katić. Zagreb : Hrvatsko knjižničarsko društvo, 2001. Str. 60.
[15] Usp. Isto. Str. 63-68.

bude više riječi o samom razvoju programskog rješenja, dat će se osvrt na pojam shema metapodataka.

2.2. Sheme metapodataka

Želeći uspostaviti red u opis velike količine informacija kako bi ih se moglo pretraživati potrebno je definirati način na koji će se to učiniti. Shema metapodataka kao: „ ...skup elemenata metapodataka izrađen za određenu svrhu, kao što je opisivanje određene vrste informacijskog izvora“[16] želi postići upravo to. No ne postoji univerzalno prihvaćena shema metapodataka. Iako postoje one koje se eventualno češće koriste od drugih, kao što je primjerice Dublin Core, koja predstavlja trenutno najpoznatiju i najkorišteniju shemu metapodataka u mrežnom okruženju[17], gotovo svaka skupina stručnjaka ima neku metapodatkovnu shemu razvijenu upravo za njihove potrebe. Unatoč raznolikosti sve ih vežu iste osobine, od kojih se mogu izdvojiti sljedeće:

1. Skup elemenata metapodataka okupljenih za ispunjenje neke funkcije ili grupe funkcija, poput, primjerice, korištenja, identificiranja i sl., informacijskog objekta
2. Skup elemenata metapodataka koji tvori strukturirani okvir kojemu se pridodaju vrijednosti koje mogu biti kontrolirane ili nekontrolirane (kao primjer kontroliranih vrijednosti mogu se uzeti predmetne odrednice Kongresne knjižnice)
3. Skup elemenata metapodataka sa svojim atributima koji je dokumentiran.[18] (Kada se kaže da je skup elemenata u nekoj shemi metapodataka dokumentiran prije svega se misli na službenu dokumentaciju koja donosi nazive elemenata, njegov položaj u shemi, obaveznost, moguće vrijednosti, atribute i slično.)

2.2.1. Dublin Core

Dublin Core shema metapodataka je stvorena 1995. godine te ju održava *Dublin Core Metadata Initiative* (DCMI) organizacija. Primarna svrha ove sheme je olakšavanje pronalaženja elektroničkih izvora na mreži. Način opisa izvora putem ove sheme metapodataka, kojim bi se to

[16] Šarić, Ivana; Magdić, Antonio; Essert, Mario. Sheme metapodataka značajne za knjižničarstvo s primjerom implementacije OpenURL standarda. // Vijesnik bibliotekara Hrvatske 54, 1/2(2011), str. 137. URL: http://www.hkdrustvo.hr/datoteke/1150/vbh/God.54(2011),br.1-2 (2016-09-20)
[17] Usp. Bosančić, Boris; Tormaš, Tena. Istraživanje o zastupljenosti shema metapodataka u repozitorijima ustanova. // 14. seminar Arhivi, knjižnice, muzeji: mogućnosti suradnje u okruženju globalne informacijske infrastrukture / Faletar Tanacković, Sanjica ; Hasenay, Damir (ur). Zagreb: Hrvatsko knjižničarsko društvo, 2011. Str. 101.
[18] Usp. Greenberg, Jane. Understanding metadata and metadata schemes. // Cataloging & Classification Quarterly 40, 3/4(2005), str. 24. URL: http://www.columbia.edu/cu/libraries/inside/units/bibcontrol/osmc/greenberg.pdf (2016-09-20)

postiglo, dovoljno je jednostavan da bi ga i sami autori mogli rabiti, kao i zajednice koje se bave formalnim opisom izvora.[19] Sekundarni, jednako važni ciljevi su: ohrabrivanje autora u kreiranju metapodataka kako bi ih računalni programi mogli preuzimati i uključivati u baze podataka za pretraživanje, povezivanje s drugim sustavima za označavanje, stvaranje programskih rješenja za jednostavnije unošenje metapodataka, prezentacija Dublin Corea kao podloge za stvaranje nekih novih složenijih shema za opis elektroničkih izvora te standardizacija s ciljem postizanja veće razumljivosti.[20]

Dublin Core shema ima ukupno 15 elemenata koji se mogu razvrstati u određene kategorije. Prema autorici Juha Hakala te kategorije su: sadržaj, intelektualno vlasništvo i pojavnost izvora, a elementi razvrstani unutar njih su:

Tablica 1. Elementi metapodataka Dublin Core sheme metapodataka.[21]

Sadržaj	**Intelektualno vlasništvo**	**Pojavnost**
Naslov	Stvaratelj	Datum
Tema	Nakladnik	Tip
Opis	Suradnik	Format
Izvor	Vlasnička prava	Identifikator
Jezik		
Odnos		
Obuhvat		

S ukupno 15 elemenata Dublin Core shema metapodataka se smatra relativno jednostavnom. Glavne značajke su da niti jedan od navedenih elemenata nije obavezan te je svaki ponovljiv onoliko puta koliko je potrebno. Uz navedeno je bitno napomenuti kako niti redoslijed elemenata nije važan te da u najvećoj mjeri ovisi o autoru. Isto tako, jezik pisanja vrijednosti određenih elemenata je proizvoljan, tako da dolazimo do situacije gdje je isti dokument moguće opisati na više jezika. Primjerice, tako je moguće da neki informacijski stručnjak opiše neki dokument putem Dublin Core sheme metapodataka na engleskom jeziku, jer je to jezik djela, ali

[19] Usp. Dublin Core Metadata. URL: purl.oclc.org/metadata/dublin_core. Citirano prema: Dizdar, S. Od podatka do metapodatka. Sarajevo: Nacionalna i univerzitetska biblioteka Bosne i Hercegovine, 2011. Str. 301-302.
[20] Usp. Dizdar, S. Nav. dj., str. 302.
[21] Usp. Hakala, Juha. Dublin Core metadata initiative, 2000. Str. 5. URL: http://cordis.europa.eu/pub/cris2000/docs/hakala1_fulltext.pdf (2016-09-20)

isto tako, da sve elemente ponovi i u istom dokumentu i isti dokument opiše i na hrvatskom jeziku. Opis može pohraniti u više formata, primjerice, HTML ili XML, s obzirom da je Dublin Core shema metapodataka neovisna o formatu. Sve ove značajke čine Dublin Core izuzetno fleksibilnom shemom i jednostavnom za korištenje za krajnje korisnike, no ipak su uvedeni dodatni kvalifikatori putem kojih shema metapodataka može postati onoliko složena koliko to autor želi.

Postoje tri vrste kvalifikatora: za jezik, za shemu i za tip. Kvalifikator za jezik definira jezik vrijednosti pojedinog elementa te bi se mogao koristiti u prethodnom primjeru o višejezičnosti, dok kvalifikator za shemu definira da se radi o nekom formalnom standardu. Jedan od tipičnih primjera bile bi predmetne odrednice Kongresne knjižnice ili neki od tradicionalnih knjižničnih klasifikacijskih sustava; ovaj kvalifikator se tako može iskoristiti za element *tema*. Upotreba kvalifikatora u okviru sheme metapodataka je izrazito važna, ne samo za potrebe predmetnog označavanja, nego i za upute vanjskim sustavima koji žele ostvariti komunikaciju, jer s ovim podatkom imaju veću mogućnost razumijevanja promatranog sustava. Kvalifikator za tip sužava značenje elementa, tako da se oni nazivaju i podelementima.[22] Kako za programsko rješenje može biti problematično pokrivanje svakog slučaja za sve kvalifikatore primjenjuje se princip *„dumb down"* tako da ako dođe do neprepoznavanja određenog kvalifikatora, on se jednostavno ignorira, a značenje elementa ostaje nepromijenjeno.[23] Fleksibilnost Dublin Corea dodatno se povećava dodavanjem vlastitih lokalnih elemenata, ako se u osnovnom setu ne pronađe odgovarajući element za potrebe opisa dokumenta. Lokalno dodani elementi započinju sa „X-„ pa tako novi element može biti *X-Cijena.* Automatsko indeksiranje ili mapiranje ovih elemenata se u velikoj većini ignorira ili se obavlja ako postoji jasno definirana dokumentacija s uputama.[24] S obzirom na navedeno, jasno je da je Dublin Core izuzetno fleksibilan te ima izuzetno široku upotrebu u praksi. Pa iako ima izuzetno mali broj elemenata, uz sva dodatna proširenja te mogućnosti on postaje kompleksan. Time automatsko indeksiranje i mapiranje postaje složenije ako se teži pokriti sve izuzetke.

2.2.2. MODS

Razvojem informacijske tehnologije mijenja se i katalogizacijska praksa. Uvidom u prednosti XML-a Kongresna knjižnica[25] razvija, baziran na MARC-u 21, MODS koji se može smatrati nešto

[22] Usp. Dizdar, S. Nav. dj., str. 304-305.
[23] Usp. Caplan, Priscilla. Nav. dj., str. 78.
[24] Usp. Hakala, Juha. Nav. dj., str. 6.
[25] Preciznije, MODS razvijaju Library of Congress' Network Development i MARC Standards Office.

jednostavnijom MARC (odnosno ISBD) verzijom sheme metapodataka, jer koristi tek jedan dio elemenata; osim toga, nazivi elemenata nisu iskazani numerički nego su bazirani na jeziku.[26] Za učenje MARC-a 21 potrebno je iskustvo i stručnost zbog opširnosti i kompleksnosti, sama sintaksa numeričkih elemenata je izuzetno složena, tako da je smjer prema XML-u i razvoju MODS-a zapravo logičan. Potreba za XML-om izražena je od strane mnogih stručnjaka koji se bave opisom kompleksnih digitalnih informacijskih objekata gdje Dublin Core jednostavno nije dovoljan. Prevelika fleksibilnost Dublin Corea dovodi do nekonzistentnosti u njegovom korištenju. Za složenije zahtjeve knjižnica i drugih informacijskih ustanova njegovih 15 elemenata postaju ograničavajući faktor. U obzir se može uzeti i rasprava stručnjaka koja još uvijek traje oko korištenja elemenata iz grupe intelektualnog vlasništva kao što su stvaratelj, nakladnik i suradnik[27]. Iako su glavni ciljevi Dublin Corea usvajanje i od strane osoba koje nisu informacijski stručnjaci, ipak upravo zbog njih i postoji potreba za složenijom shemom što je dovelo do razvoja MODS-a kao sheme metapodataka s više elemenata od Dublin Core-a. Na prvoj razini, postoji ukupno 20 MODS elemenata: *titleInfo*, *name*, *typeOfResource*, *genre*, *originInfo*, *language*, *physicalDescription*, *abstract*, *tableOfContent*, *targetAudience*, *note*, *subject*, *classification*, *relatedItem*, *identifier*, *location*, *accessCondition*, *part*, *extension* te *recordInfo*, a jedinstvenih, podređenih elemenata je 47. S obzirom da MODS nasljeđuje semantiku MARC-a, između ove dvije sheme veća je razina interoperabilnosti nego između MODS-a i Dublin Corea, stoga je za potrebe mapiranja potrebno pratiti službenu dokumentaciju koja donosi pregled semantički ekvivalentnih elemenata između ovih dviju shema metapodataka. Prije pregleda razvijenog automatskog procesa mapiranja uz praćenje dokumentacije dat će se još i pregled teorijskih osnova mapiranja.

2.3. Mapiranje shema metapodataka

Mapiranje metapodataka se definira kao intelektualan proces uspoređivanja i analiziranja dvije ili više shema čiji je rezultat vizualan i tekstualan proizvod nazvan *crosswalks*. *Crosswalks* je već bio spomenut u radu kao dokumentacija mapiranja što je i najtočnija percepcija ovog neprevodivog pojma. Krajnji proizvod je obično tablica ili grafikon koji pokazuje veze i semantički ekvivalentne odnose među elementima u dvije ili više shema metapodataka.[28] Za stvaranje ovakve

[26] Usp. Guenther, Rebecca S. MODS: The metadata object description schema. // Libraries and the academy 3, 1(2003), str. 138. URL: http://www.loc.gov/standards/mods/3.1guenther.pdf (2016-09-20)
[27] Usp. Isto. Str. 137.
[28] Usp. Woodley, Mary S. Metadata matters: Connecting people and information. // Introduction to metadata / urednica Murtha Baca. 3. izd. Los Angeles: Getty Publications, 2016. URL: http://www.getty.edu/publications/intrometadata/metadata-matters/ (2016-09-20)

dokumentacije potrebna je izrazita stručnost u području informacijskih znanosti i teorijskih postavki metapodataka uz dubinsko razumijevanje samih shema koje se želi mapirati. No stvaranje dokumentacije za mapiranje elemenata je tek jedna razina problema. Ona druga, mnogo kompleksnija je mapiranje vrijednosti elemenata i praćenje standarda u kojemu određena shema metapodataka zahtijeva formatiranje vrijednosti. Primjerice, formatiranje datuma ili osobnih imena. Tako MODS zahtijeva strukturirani format vrijednosti elementa *name*.[29] Ekvivalentni element za MODS-ov *name* je u Dublin Coreu *Creator* koji nema standard za formatiranje vrijednosti ovog elementa. Informacijski stručnjak prilikom procesa mapiranja tako, treba obratiti pažnju hoće li mapiranje biti prema standardu ili ne. Ljudskim resursima je jednostavnije voditi računa o ovom problemu nego računalnom programu koji će ovaj proces raditi automatski. Ako je primjerice vrijednost elementa *Creator* u Dublin Coreu Miroslav Krleža, njegova formatirana vrijednost bi u MODS-u bila Krleža, Miroslav. Kada se osvrnemo na ovaj konkretan slučaj, računalni program nije moguće programirati da iz vrijednosti u Dublin Coreu uzme drugu riječ, potom stavi zarez i nakon toga prvu riječ te tu vrijednost mapira u MODS. Naime, moguće je da već imamo i u Dublin Coreu formatiranu vrijednost, jer pravilo za strukturiranje vrijednosti ne postoji. Uvjet koji provjerava postoji li zarez, pa pretpostavlja da bi to mogla biti strukturirana vrijednost i potom ju samo prepisuje isto tako nije dostatna zbog izuzetaka. Uzmimo primjer imena glazbenika Johanna Sebastiana Bacha za koje računalni program s prethodnom logikom ne bi mogao razaznati što je prezime, a što ime. Čak i kada bi se logika postavila tako da ako postoje tri riječi, ona posljednja je prezime, a ispred nje su dva imena, ponovno postoje izuzeci gdje osoba može imati dva prezimena i tada ni ta logika nije dovoljno dobra. Ovo je samo jedan od primjera koliko je teško razviti programsko rješenje koje će automatiziranim procesima vršiti tijek mapiranja, a da bi se pritom zadržao kurs propisanog standarda. Ove probleme potrebno je svakako riješiti i oni nisu beznačajni. Ako se oni ne riješe možemo samo zamisliti do kakvih bi problema došlo u, primjerice, mrežnim tražilicama ili čak metatražilicama više baza podataka. Valja napomenuti kako knjižnična zajednica ima izrazito dugu i relativno uspješnu tradiciju razmjene podataka stvorenih od strane više knjižnica s obzirom na brojne standarde koji su za ovu zajednicu specifični, dok je s druge strane situacija izvan knjižnične zajednice u mrežnom okruženju najčešće nestandardizirana i gdje autori vrlo često sami generiraju metapodatke.[30]

Postoji više vrsta mapiranja od kojih je mapiranje "jedan naprema jedan" potpuno

[29] Usp. Library of Congress. Dublin Core metadata element set mapping to MODS: Version 3, 2012.URL: http://www.loc.gov/standards/mods/dcsimple-mods.html (2016-09-20)
[30] Usp. Dushay, Naomi; Hillmann, Diane I. Analyzing metadata for effective use and re-use. // International conference on Dublin Core and metadata applications 0, 0(2003), str. 161. URL: http://dcpapers.dublincore.org/pubs/article/view/744/740 (2016-09-20). Citirano prema: Woodley, Mary S. Nav. dj.

mapiranje: element iz odredišne sheme metapodataka u potpunosti je semantički ekvivalentan s elementom iz polazne sheme metapodataka. Nadalje, postoje i mapiranja jedan prema više i više prema jedan. Iako i u prethodnim slučajevima treba donijeti odluke vezane uz mogućnost odabira s kojim elementom iz odredišne sheme metapodataka treba mapirati element iz polazne sheme metapodataka, nešto su kompliciraniji slučajevi kada postoje elementi koje nije uopće moguće mapirati. U nekim slučajevima moguće je da polazna shema metapodataka ima obavezne elemente, a da odredišna shema metapodataka ne sadrži takve semantički ekvivalentne elemente. Ako se pojavi ovakav slučaj dokumentacija mora definirati kako će se ovo razriješiti.[31] Mapiranjem iz složenije sheme u jednostavniju će zasigurno doći do gubitka informacija. Jednostavno, ne može se očekivati jednaka razina granularnosti i kompleksnosti koju složena shema metapodataka može pružiti, no tada situaciju treba promotriti u njezinom kontekstu. Jednostavnija shema je i napravljena s određenom svrhom, možda i da bude što jednostavnija, te prilikom mapiranja treba prosuditi jesu li dosegnuti ciljevi koje ona želi ostvariti, bez obzira na gubitak nekih informacija.

Mapiranje i generiranje metapodataka treba promatrati u kontekstu suvremene situacije u svijetu. Poznato je kako je informacija svakim danom sve više i da informacijski stručnjaci sve teže mogu pratiti svoje poslanje osiguravanja pristupa informacijama svojim krajnjim korisnicima. Da bi to bilo moguće, potrebno je svaki dokument opisati metapodacima kako bi se on mogao pretraživati, a za što su potrebni ljudski resursi, vrijeme i novac. Rješenje se možda nalazi u programskim rješenjima koji bi ove procese odrađivali automatizmom sa što manjom ljudskom intervencijom ili u potpunosti bez nje. Takva programska rješenja imaju mnoge prednosti. Prva je skalabilnost koja se odnosi na funkcioniranje programskih rješenja u kontekstu velike količine informacijskih izvora, a druga su ljudski resursi kojih nema dovoljno da zadovolje sve potrebe zajednice. Programska rješenja s obzirom na skalabilnost omogućuju procesuiranje velike količine informacijskih izvora u kratkom vremenu i time znatno štede financijske izdatke i vrijeme. Dobivanjem slobodnog vremena, kada bi se ovo dogodilo, informacijski stručnjaci bi se mogli posvetiti drugim poslovima koje računalo automatiziranim procesima ne može odrađivati. Oslobađanjem stručnjaka od repetitivnih zadataka bi se eventualno povećala i njihova kreativnost što bi dovelo u konačnici i do bolje vidljivosti informacijske ustanove u zajednici i većeg zadovoljstva krajnjih korisnika.[32] Iako je ovdje riječ o relativno idealnoj situaciji, već jedan korak, kreiranje polu-automatiziranih računalnih programa donijelo bi veliku promjenu. Problem ostaje,

[31] Usp. Pierre, Margaret St.; Laplant, William P. Issues in crosswalking content metadata standards. Bethesda: NISO, 1998. URL: http://www.niso.org/publications/white_papers/crosswalk/ (2016-09-20)
[32] Usp. Parl, Jung-ran; Brenza, Andrew. Evalution of semi-automatic metadata generation tools: A survey of the current state of the art. // Information technology and libraries 34, 3(2015), str. 23. URL: https://ejournals.bc.edu/ojs/index.php/ital/article/view/5889/pdf (2016-09-20)

njihovo stvaranje je specifično i teško univerzalno primjenjivo. Potrebna su velika početna ulaganja kao i ulaganja u sredstva za održavanje. Potrebni su kompetentni stručnjaci sa znanjem programiranja koji bi ovakve alate napravili i održavali. S obzirom na dobitak, u ovakvu budućnost treba uložiti i jedan od glavnih razloga zašto je u sklopu diplomskog rada kreirano ovakvo programsko rješenje je da bi se dokazalo da je ovo moguće. U nastavku slijedi kratak pregled programskih jezika i nekih osnovnih pojmova u programiranju prije nego što se izloži pregled razvoja programskog rješenja.

4. Programski jezici

Razvijeno programsko rješenje je mrežna aplikacija. Njezini dijelovi su prednji sustav, odnosno dio programskog kôda koji se izvršava na strani klijenta te pozadinski sustav koji se izvršava na strani poslužitelja, odnosno na strani servera. Pozadinski sustav je programiran u PHP programskom jeziku, dok je prednji sustav programiran u JavaScript programskom jeziku. U prednjem sustavu se još nalaze i HTML oznake te CSS za koji se pojednostavljeno može reći da je stilski jezik. Za potpuno razumijevanje programskog rješenja bit će potrebno uz navedene jezike, objasniti i osnovne pojmove u programiranju kao što su klase, objekti, petlje, nizovi, tipovi podataka itd. te još neki dodatni pojmovi kao što su Apache poslužitelj, JSON, AJAX... Pregled će biti jezgrovit s izloženim najvažnijim funkcionalnostima bez ulaženja u prevelike detalje s obzirom na opseg sadržaja koji je moguće pokriti. A kada bude riječi o razvijenom rješenju, temeljem konkretnih primjera, objasnit će se još poneki pojmovi.

4.1. PHP

PHP (engl. *Hypertext PreProcessor*) je nastao 1994. i od tada je doživio mnoge izmjene. Trenutna verzija je PHP 7 i trenutno je jedan od najpopularnijih jezika na strani poslužitelja. To je skriptni programski jezik koji se izvršava na strani poslužitelja i služi za dinamičko generiranje HTML kôda.[33] Kôd se može skriptno izvršavati na strani poslužitelja i tu se radi o njegovoj najčešćoj upotrebi. To je slučaj kada se kreiraju dinamički sadržaji na nekoj mrežnoj stranici. Isto tako se kôd može izvršavati u naredbenom retku (engl. *command-line*); jedan tipičan primjer bio bi izvršavanje skripte kroz naredbeni redak koja će napraviti sigurnosnu kopiju baze podataka koja se koristi u nekom programskom rješenju. Posljednji primjer izvršavanja kôda su klijentske GUI

[33] Usp. Havas, Ladislav; Lesar, Matija. Primjena SQL-a u programiranju otvorenog koda. // Tehnički glasnik 6, 2(2012), str. 168. URL: http://hrcak.srce.hr/file/139594 (2016-09-20)

(engl. *Graphical User Interface*) aplikacije.[34] Glavna odlika GUI-ja je ta što omogućuje i proceduralno i objektno orijentirano programiranje, a za pokretanje aplikacije je uvijek potreban poslužitelj. Najčešći poslužitelj je Apache koji može doći s već unaprijed instaliranim PHP-om te je neovisan o operativnom sustavu. Sve PHP datoteke će se vrlo jednostavno prepoznati po ekstenziji „.php“. Varijable se definiraju sa znakom dolar: „$“ i njezine vrijednosti mogu biti različiti tipovi podataka koji se rabe u PHP-u: tekstualan podatak (engl. *string*), cijeli broj, decimalni broj, logička vrijednost (engl. *boolean*), niz i objekt.

Primjer tekstualnog podatka:

```
$ime = 'Miroslav Krleža';
```

Primjer cijelog broja:

```
$broj = 42;
```

Primjer decimalnog broja:

```
$broj = 42.22;
```

Primjer logičke vrijednosti koja može biti samo točna ili netočna:

```
$logicka  = false;
```

Jedan od složenijih tipova podataka su nizovi. Oni mogu biti asocijativni i indeksni.

Primjer indeksnog niza bi bio:

```
$sheme = array('DC', 'MODS', 'METS', 'ONIX');
```

Indeksni niz se upravo tako naziva iz razloga što je indeks cjelobrojna vrijednost položaja u nizu. Indeks uvijek kreće od nule, pa je tako vrijednost na prvom položaju u ovom slučaju *DC*. Kada bi tu vrijednost htjeli ispisati na samoj stranici napisali bi:

```
echo $sheme[0];
```

Ako bi željeli ispisati sve vrijednosti iz niza, može se koristiti petlja. Petlja je dio programskog kôda koji se izvršava onoliko puta koliko je definirana, iako se može "vrtjeti" i beskonačno. Razlika između indeksnog i asocijativnog niza je što asocijativni umjesto indeksa koristi parove ključ vrijednost.

Primjer:

```
$mods = array(
    "kreator" => "Kongresna knjižnica",
    "brojElemenata" => 20
);
```

Naravno, i nizovi mogu biti višedimenzionalni, odnosno unutar niza se može nalaziti novi niz:

```
$mods = array(
    "kreator" => "Kongresna knjižnica",
```

[34] Usp. Lerdorf, R.; Tatroe, K.; Macintyre, P. Programming PHP. O'Reilly Media, 2006. Str. 1. URL: https://books.google.hr/books?id=h-E1lVko-skC&lpg=PT5&ots=ypL-cQOrSi&dq=php%20programming&lr&hl=hr&pg=PT5#v=onepage&q&f=false (2016-09-20)

```
"brojElemenata" => 20,
    "elementi" =>array("titleInfo", "name", "typeOfResource", "genre", "originInfo",
"language", "physicalDescription", "abstract", "tableOfContents", "targetAudiance",
"note", "subject", "classification", "relatedItem", "identifier", "location",
"accessCondition'", "part", "extension", "recordInfo")
);
```

Ako iz prethodnog primjera želimo dohvatiti naziv prvog elementa u nizu i ispisati ga na stranici, u ovom slučaju to je *titleInfo* to ćemo učiniti:

```
echo $mods['elementi'][0];
```

Posljednji tip podatka su objekti. Da bi se razumjeli objekti treba razumjeti klase. Klase se mogu predstaviti kao predlošci za objekte koji sadrže određena svojstva. Ako razmišljamo o shemama metapodataka oni imaju elemente, instituciju koja se brine za shemu, određenu sintaksu, format i slično, a sve to bi pripadalo klasi sheme metapodataka. Tada bi primjerice MODS bio objekt, odnosno instanca te klase. Pozitivna stvar je ta što PHP već dolazi s predefiniranim setom klasa od kojih je za ovaj rad najvažnija *DomDocument*.[35] Ona ima mogućnost manipulacije nad XML i HTML dokumentima učitavajući ih kao DOM (engl. *Document Object Model*). DOM je aplikacijsko programsko sučelje (engl. *application programming interface*, *API*) koje definira logičku strukturu dokumenta na način da se tom dokumentu može pristupiti i manipulirati njime. Na ovaj način, unutar DOM-a programeri imaju mogućnost kreiranja XML i HTML dokumenata, navigacije kroz njihovu hijerarhijsku strukturu, dodavanja i brisanja elemenata, manipuliranja vrijednostima elemenata, te mogućnost dodavanja atributa i mijenjanje već postojećih. Jednostavno rečeno, svemu što se nalazi unutar XML ili HTML dokumenta moguće je pristupiti i sve je moguće mijenjati.[36]

4.2. JavaScript

JavaScript je programski jezik mreže. Gotovo svi moderni preglednici na osobnim računalima, tabletima ili mobitelima imaju uključene JavaScript interpretere. Gotovo su svi programeri mrežnih aplikacija upoznati s tri tehnologije koje vladaju mrežom. To su: HTML koji definira sadržaj stranice, CSS koji je zadužen za izgled i JavaScript koji je zadužen za ponašanje mrežne aplikacije.[37] U HTML dokumentima JavaScript možemo pronaći u elementu *script*. Kao i svaki programski jezik, JavaScript ima varijable te one mogu imati vrijednosti različitih tipova podataka.

[35] Usp. The DomDocument class. http://php.net/manual/en/class.domdocument.php (2016-09-20)
[36] Usp. W3C. What is the Document Object Model?. URL: https://www.w3.org/TR/WD-DOM/introduction.html (2016-09-20)
[37] Usp. Flanagan, David. JavaScript: The definitive Guide: Activate you web pages. O'Reilly Media, 2011. Str. 1. URL: https://books.google.hr/books?id=j2htBAAAQBAJ&lpg=PR1&pg=PR1#v=onepage&q=ajax&f=false (2016-09-20)

Svaka varijabla se definira riječju *var*, a u nastavku se navode primjeri varijabli koje ujedno predstavljaju i različite tipove podataka:

```
var x; //deklaracija varijable[38]
x = 'Dublin Core'; //niz znakova
x = 1; //cijeli broj
x = 1.2; //decimalni broj
x = false; //logička vrijednost koja može biti true ili false
x = null; //poseban tip podaka koji se odnosi na to da nema vrijednost
x = undefined; //isto kao i null
```

Kao i PHP, JavaScript ima posebne „tipove podataka“. Oni su objekti i nizovi. Primjer deklariranja nizova i objekata u JavaScriptu izgleda ovako:

```
var mods = {        //primjer objekta
kreator: "Kongresna Knjižnica",
    elemenata: 25
};
var sheme = ["MODS", "Dublin Core", "ONIX"]; // indeksni niz
```

Nakon pregleda objekata i nizova vrlo je bitno spomenuti serijalizaciju objekata i JSON (*JavaScript Object Notation*). Serijalizacija objekata je proces konvertiranja stanja objekta u niz znakova iz kojega se on kasnije može i vratiti.[39] Dvije funkcije koje će napraviti serijalizaciju i vraćanje u objekt su *JSON.stringify()* i *JSON.parse()*. Serijalizacijom će se objekt prebaciti u format pod nazivom JSON i njegova sintaksa je gotovo identična sintaksi objekata i nizova, ali ipak JSON sintaksa je tek dio JavaScript sintakse tako da to nisu identične stvari. JSON je vrlo bitan u komunikaciji klijenta i poslužitelja, jer se radi o tipičnom formatu za razmjenu podataka. Kao primjer se može uzeti situacija u kojoj korisnik u tražilici na mrežnoj stranici unese tekst i pritisne gumb za traženje. JavaScript će uzeti vrijednost polja koje je korisnik upisao, i taj podatak poslati poslužitelju na kojemu će, primjerice, PHP poslati upit bazi podataka i vratiti sve rezultate koji sadrže upisani pojam. Rezultate će serijalizirati u JSON format i vratiti klijentu, gdje će JavaScript vraćene rezultate deserijalizirati u sebi čitljive objekte i pokazati ih korisniku na stranici. Ako pogledamo ovaj primjer, kada se klikne gumb za traženje gotovo nikada se ne osvježava cijela stranica, nego samo dio s rezultatima. To se postiže AJAX-om.

AJAX (*Asynchronus JavaScript and XML*) koristi *XMLHttpRequest* objekt za komunikaciju s poslužiteljem asinkrono bez potrebe da se osvježava cijela stranica nego samo jedan njezin dio kada poslužitelj procesuira poslani upit sa strane klijenta. Vrlo često se u programiranju u JavaScriptu ne piše sav kôd iz početka nego se koriste biblioteke koje ovaj proces znatno ubrzavaju.

[38] Dvije kose crte označavaju komentar.
[39] Usp. Isto. Str. 138.

Biblioteka koja je korištena u razvoju je *jQuery*. *jQuery* je potpuno besplatna biblioteka[40] koja znatno olakšava: pronalaženje elemenata na mrežnoj stranici, dodavanje novih elemenata i njihovo brisanje, mijenjanje CSS svojstava, pravljenje animacija te slanje AJAX upita poslužitelju. *jQuery* biblioteka koristi globalnu metodu *jQuery()* koja se izrazito često koristi unutar nje same te je jednostavno skraćena globalnim simbolom znaka dolar: $.[41] Tako da, ako se u ovom programskom rješenju unutar JavaScripta pronađe znak dolara, zna se da se radi o *jQueryu*. Kada se u programiranju koristi neka biblioteka vrlo je važno znati što se događa u pozadini tog kôda. Vrlo je česta pogreška učenje biblioteke umjesto samog jezika i time je otežana tranzicija na neku novu biblioteku u nekom projektu ako za to dođe potreba. Svaki dio kôda koji neka biblioteka za nas skraćeno radi, trebao bi se znati napisati i bez nje. Kada se poznaje programski jezik, svaka nova biblioteka je tek čitanje dokumentacije i vježba sintakse, jer je poznato što se u pozadini napisanog kôda događa.

5. Izrada programskog rješenja za mapiranje shema metapodataka

5.1. Svrha i ciljevi izrade programskog rješenja za mapiranje shema metapodataka

Danas na mreži ne postoji programsko rješenje čija je svrha mapiranje elemenata iz različitih shema metapodataka. Primjerice, alati, kao što je *The Simple Dublin Core generator*[42] koji služi za generiranje shema metapodataka, postoje[43], ali alati za mapiranje shema metapodataka, kao što je to, primjerice, bio UKOLN-ov *DC-assist*[44] više ne postoje; a ugašeni su najčešće zbog prestanka financiranja organizacije koja ga je održavala. Danas kada je potreba za ovakvim programskim rješenjima veća no ikad, čini se pomalo nevjerojatnim da takva programska rješenja nisu prisutna na mreži. Odatle, svrha rada je izrada programskog rješenja za mapiranje shema metapodatka koje će u prvoj inačici omogućavati automatsko mapiranje vrijednosti elemenata DC i MODS sheme metapodataka uz njihovu prvotnu validaciju prema pripadnim XML Schemama.

U skladu s tim, ciljevi rada su:

- izrada prednjeg sustava programskog rješenja koji je krajnjem korisniku vizualno atraktivan, jednostavan za korištenje te mu pruža osnovne informacije o programskom rješenju;

[40] JQuery može se preuzeti na poveznici: https://jquery.com/
[41] Usp. Isto. Str. 523-524.
[42] Usp. Steffel, Nick. The simple Dublin Core generator, 2010. URL: http://www.dublincoregenerator.com/index.html (2016-10-15)
[43] Za dodatan popis alata za generiranje i harvestiranje usp. Parl, Jung-ran; Brenza, Andrew. Nav.dj.
[44] DC-assist. URL: http://www.ukoln.ac.uk/metadata/dcassist/index.html (2016-10-15)

- izrada pozadinskog sustava koji će biti u potpunosti funkcionalan prilikom mapiranja te će slijediti službenu dokumentaciju specifikacije mapiranja DC u MODS sheme metapodataka i obrnuto Kongresne knjižnice u Washingtonu;
- izrada kontrola u pozadinskom sustavu koje će dopustiti mapiranje samo validnih Dublin Core i MODS XML datoteka;
- povezivanje pozadinskog i prednjeg sustava kako bi krajnji korisnik imao uvid u status mapiranja svake priložene datoteke uz mogućnost njihovog preuzimanja;

5.2. Pregled

Razvoj programskog rješenja odvijao se na Linux Ubuntu operativnom sustavu. Kôd je pisan u PhpStorm-u,[45] profesionalnoj integriranoj razvojnoj okolini. Profesionalnoj iz razloga što se licenca za korištenje plaća, dok je kao poticaj programiranju i učenju za studentsku populaciju moguće dobiti besplatnu licencu. Razvijeno programsko rješenje[46] ima krajnju strukturu od 6 direktorija i 35 datoteka (Slika 2. u Prilogu) uz nekoliko skrivenih datoteka o kojima će biti riječi u nastavku rada. Prilikom otvaranja početne stranice učitava datoteku *index.php*. Ta datoteka u sebe uključuje naredbom *include* druge datoteke koje zajedno čine cjelinu koju krajnji korisnik vidi:

```
<?php
include "head.php";
include "navigation.php";
include "firstSection.php";
include "about.php";
include "examples.php";
include "mapping.php";
include "footer.php";
include "script.php"
```

Ono što krajnji korisnik vidi je jedna mrežna stranica koja se sastoji od dijelova kroz koje se može kretati pomicanjem prema dolje ili gore ili klikom na nazive dijelova u fiksnoj navigaciji na vrhu koji će animirati put do odabrane sekcije (Slika 3. u Prilogu). Jedna stranica u mrežnom okruženju zasniva se na filozofiji minimalizma, s prikazom samo najbitnijeg sadržaja, a za krajnjeg korisnika je jednostavna i privlačna te predstavlja svojevrstan trend u posljednjih nekoliko godina razvoja

[45] PhpStorm je moguće preuzeti na adresi: https://www.jetbrains.com/phpstorm/
[46] Završna verzija programskog rješenja se nalazi na poveznici https://amatanovic.com/diplomski/index.php, dok je cjelokupan programski kôd moguće preuzeti s GitHub repozitorija na poveznici https://github.com/amatanovic/mapiranje-metapodataka

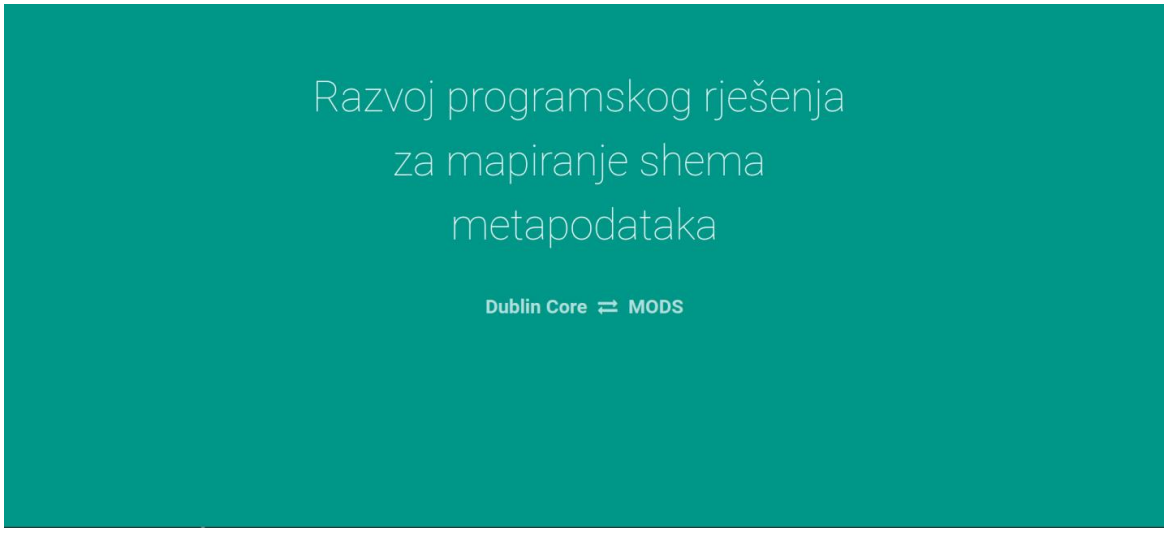
programskih rješenja ovog tipa.[47] Boje, fontovi i ikone prate suvremeni trend *Material Designa* čiji je autor Google.[48] U prednjem sustavu kao radna okolina korišten je *Foundation* koji donosi predefinirane CSS klase te tako čini oblikovanje mrežne stranice jednostavnijim i bržim.[49] Njegov CSS se učitava u *head* elementu u HTML dokumentu:

```
<link rel="stylesheet" href="css/foundation.min.css" />
```

i nalazi se na relativnoj putanji u direktoriju pod nazivom „css“. U *head* elementu se još učitava i ikona mrežne stranice koja se vidi u otvorenoj kartici u pregledniku, korišteni font, CSS datoteka za ikone, CSS za oblikovanje prikaza XML primjera, vlastiti CSS te dvije JavaScript datoteke: za saznavanje mogućnosti korisničkog preglednika koji dolazi s *Foudantionom* te se mora koristiti kako bi radna okolina bila stabilna i druga za učitavanje XML primjera:

```
<link rel="icon" href="img/favicon.ico" type="image/x-icon"/>
<link
href='https://fonts.googleapis.com/css?family=Roboto:400,100,700&subset=latin,latin-
ext' rel='stylesheet' type='text/css'>
<link href="css/font-awesome.min.css" rel="stylesheet">
<link rel="stylesheet" href="css/foundation.min.css" />
<link rel="stylesheet" href="css/monokai_sublime.css" />
<link rel="stylesheet" href="css/style.css" />
<script src="js/vendor/modernizr.js"></script>
<script src="js/highlight.pack.js"></script>
```

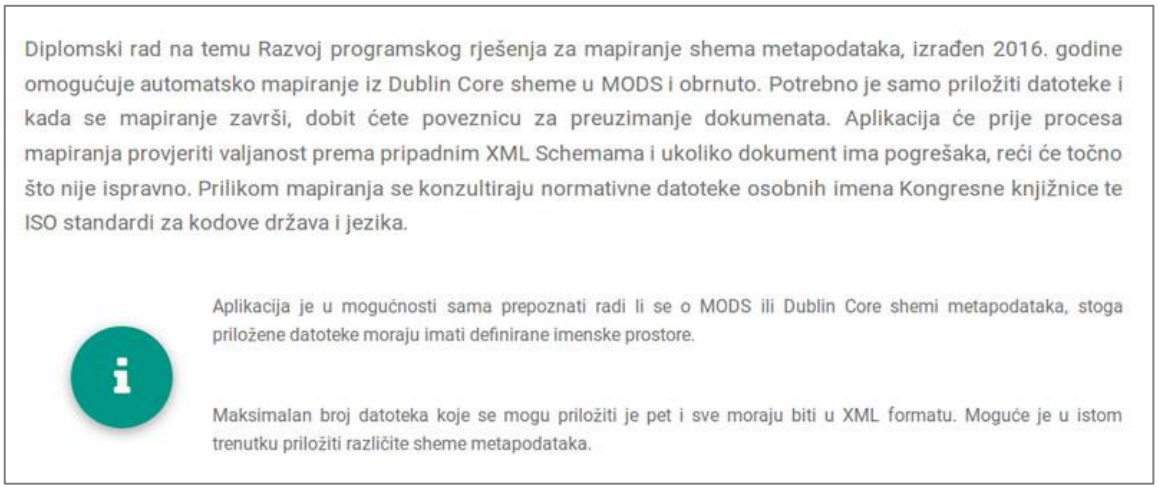
Prvi dio koji krajnji korisnik vidi je naslov i podnaslov i ta sekcija je nazvana „Početak“.



Slika 4. Naslov i podnaslov; prvi dio programskog rješenja

[47] Usp. Weller, Nathan B. 8 reasons why pageless design is the future of the web. URL: http://www.dtelepathy.com/blog/design/8-reasons-why-pageless-design-is-the-future-of-the-web (2016-09-20)
[48] Usp. Google. Material design. URL: https://material.google.com/ (2016-09-20)
[49] Foundation: the most advanced responsive front-end framework in the world. URL: http://foundation.zurb.com (2016-09-20)

Sljedeći dio donosi kratak opis programskog rješenja i kratak pregled nekih od funkcionalnosti.



Slika 5. Kratak opis programskog rješenja kao uputa krajnjem korisniku

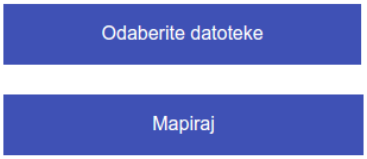
Kratak pregled govori o tome da je riječ o programskom rješenju koje služi za mapiranje shema metapodataka, konkretno Dublin Core u MODS i obrnuto te da je potrebno priložiti do pet datoteka koje će se mapirati, a njih je moguće kombinirati tako da je u jednom procesu mapiranja moguće priložiti i Dublin Core i MODS, a kao rezultat će biti ona suprotna shema od priložene. Sve priložene datoteke moraju biti valjane prema službenim, pripadnim XML Schemama, inače se proces mapiranja neće moći izvršiti, a prilikom mapiranja će se konzultirati normativne datoteke Kongresne knjižnice osobnih imena te ISO standardi za kodove država i jezika. Treći dio mrežne stranice je nazvan „Primjeri“ gdje su dana dva primjera XML datoteke s Dublin Core i MODS shemom[50] koje predstavljaju međusobne rezultate mapiranja korištenjem ovog programskog rješenja.

[50]Za pregled primjera Usp. Glas, Božidar. Generiranje i mapiranje metapodataka u digitalnoj knjižnici hrvatske tiskane baštine Edicija. Osijek: Filozofski fakultet: 2013.



Slika 6. Primjeri rezultata mapiranja

Sljedeći dio mrežne stranice je nazvan „Mapiranje“ koji s korisničke strane bez započetog procesa mapiranja ima dva gumba: Odaberite datoteke i Mapiraj.



Slika 7. Dio „Mapiranje“ bez započetog procesa mapiranja

Posljednji dio stranice je podnožje koje sadrži informacije o autoru, godini nastanka programskog rješenja te poveznice na instituciju nastanka rada i GitHub repozitorij s cjelokupnim programskim kôdom.



Slika 8. Podnožje

Dio koji sadrži gotovo cjelokupnu programsku logiku se nalazi, s korisničke strane u dijelu

stranice nazvan „Mapiranje“. Klikom na gumb „Odaberite datoteke“ otvara se dijaloški okvir za prijenos datoteka s lokalnog računala. Moguće je odabrati više datoteka te se nakon potvrde dijaloški okvir zatvara, a na gumbu se ispisuje broj datoteka koje je korisnik priložio. Nakon toga klikom na gumb „Mapiraj“ prikazuje se dio koji je do sada bio skriven, a koji će prvotno prikazivati okretnu ikonu s porukom „Molimo pričekajte“ nakon čega će se ispisati status mapiranja. Ako je sve prošlo u redu, korisnik će dobiti poveznicu za preuzimanje datoteka. Kada je mapirana samo jedna datoteka, rezultat mapiranja će se moći pregledati i preuzeti, a ako je bilo više mapiranih datoteka moći će se preuzeti sve zajedno u *zip* formatu. Kako je ovo bio vrlo kratak pregled funkcionalnosti razvijenog programskog rješenja, u nastavku rada će se, uz primjere programskog kôda, ovaj dio detaljno proći.

5.3. Prilaganje datoteka

Ova sekcija na mrežnoj stranici nalazi se u datoteci *mapping.php* i upravo je ta datoteka uključena na početku datoteke *index.php.* Dva navedena gumba su okružena HTML formom:

```
<form method="post" action="upload.php" enctype="multipart/form-data" target="log">[51]
...
</form>
```

Navedena forma koristi metodu *POST* koja, pojednostavljeno rečeno, označava da će se podaci poslati poslužitelju, nasuprot metodi *GET* koja podatke prima. Podaci će se poslati u skriptu upload.php što se saznaje preko atributa *action*. Istom formom želimo poslati datoteke poslužitelju, što znači da će se unutar nje negdje nalaziti gumb za prilaganje datoteka tako da je te podatke potrebno enkodirati kako bismo ih mogli poslati; za to je pak zaslužan atribut *enctype.* Atribut *target* definira gdje želimo da se vrati odgovor poslužitelja. U ovom slučaju, odgovor poslužitelja vraća se u element s imenom *log*. Taj element je upravo onaj element koji je zadužen za ispis poruka kada završi proces mapiranja, i na njega će se posebno obratiti pažnja. Gumb koji korisnik vidi za prilaganje datoteka (Slika 9. u Prilogu) su zapravo dva gumba, koja krajnji korisnik ne vidi, no u programskom kôdu to izgleda ovako:

```
<div class="file-input-wrapper">
<button id="odaberite" class="button expand">Odaberite datoteke</button>
<input name="upload[]" type="file" multiple="multiple" accept=".xml" id="upload"
title=""onchange="promijeniTekstUpload()"/>
</div>
```

[51] Tri točkice unutar programskog kôda označavaju da se na tom mjestu nalazi još programskog kôda koji za potrebe prikaza u tom trenutku nije prikazan.

Dva gumba su tu zbog korisničkog iskustva s obzirom da je standardni gumb za prilaganje datoteka (Slika 10. u Prilogu) izrazito neatraktivan te ga je izuzetno teško mijenjati. Razlog tomu je što je njegova funkcija specifična, i promjenom njegova izgleda se gube funkcionalnosti te se gumb počinje ponašati nestandardno. Gumb za *prilaganje* datoteka je element *input* čiji je naziv *upload[]*. Uglate zagrade u imenu definiraju varijablu koja je tipa niz, a to znači da može primiti više od jedne priložene datoteke, što se može provjeriti preko vrijednosti atributa *multiple.* Korisnik ovaj gumb ne vidi na ekranu, jer ima zadanu 100% prozirnost, a što se vidi u CSS svojstvu *opacity: 0.*

```
.file-input-wrapper > input[type="file"] {
	font-size: 1.5em;
	position: absolute;
	top: 0;
	right: 0;
	opacity: 0;
	cursor: pointer;
}
```

Isto tako, njegova širina i dužina odgovaraju širini i dužini plavog gumba koji se vidi na ekranu i na kojem piše: „Odaberite datoteke“. Ovaj gumb se, dakle, ovdje nalazi iz stilskih razloga te kako bi uputio korisnika na akciju, no konkretnim klikom na njega ne događa se ništa. Akcija se događa klikom na gumb koji ga prekriva i koji 'na sebi ima' JavaScript funkciju koja će se pokrenuti ako dođe do promjene vrijednosti elementa *input,* a to bi se trebalo dogoditi kada dođe do odabira datoteke. Tada će gumb s tekstom „Odaberite datoteke“ promijeniti poruku. JavaScript funkcija se nalazi u datoteci *script.php* i izgleda ovako:

```
function promijeniTekstUpload() {
          var uploaded = $('#upload').prop('files').length;
          if (uploaded === 0) {
              $("#odaberite").html("Odaberite datoteke");
          }
          else {
              $("#odaberite").html("Odabrano datoteka: " + uploaded);
          }
      }
```

Element *input* 'ima na sebi' atribut *accept* čija je vrijednost „.xml“ a što predstavlja prvotni pokušaj kontrole da se prilažu samo XML datoteke. Otvoreni dijaloški okvir za odabir datoteka će automatski s obzirom na ovu vrijednost filtrirati datoteke po ovoj vrsti (Slika 11. u Prilogu). Kaže se 'prvotna kontrola' s obzirom da se ovo ograničenje može vrlo lako zaobići s korisničke strane. Jednostavno se može odabrati opcija koja će prikazati sve datoteke u direktoriju, no tada će postojati kontrola na strani poslužitelja koja će to onemogućiti.

5.4. Prije procesa mapiranja

Gumb s tekstom „Mapiraj“ (Slika 12. u Prilogu) je isto tako *input* element:

```
<input type="submit" value="Mapiraj" class="button expand" name="mapping" id="mapiraj"
/>
```

Njegov tip je *submit* što znači da će se klikom na ovaj gumb forma poslati poslužitelju. No, prije nego što se forma pošalje poslužitelju odradit će se još nekoliko stvari. Najprije, JavaScript će također očekivati klik na ovaj gumb te će ona odraditi sljedeće:

```
$("#mapiraj").click(function() {
            $("#log").css("visibility", "visible");
            getProgress();
   });
```

Postavit će elementu sa selektorom identifikatora *log* CSS svojstvo vidljivosti i počet će pratiti napredak procesa mapiranja pozivajući funkciju *getProgress()*.Navedeni element se nalazi ispod forme u HTML dokumentu i on je prilično specifičan:

```
<iframe class="log" id="log" name="log" src="upload.php">
</iframe>
```

To je element koji postaje vidljiv na klik gumba za mapiranje (Slika 13. u Prilogu) i u njemu se ispisuje status mapiranja, eventualne greške te se daju poveznice za mapiranje. Iako njega korisnik ne vidi, on je uvijek učitan na samoj stranici te je reprezentiran rotirajućom ikonom s tekstom: „Molimo pričekajte ...“ koja postaje vidljiva tek kad se pokrene akcija mapiranja i koja će ostati vidljiva dok ne završi proces mapiranja. Iznad ikone, u posebnoj traci, prikazuje se napredak statusa mapiranja. Status mapiranja ima raspon od nula do sto posto i kada se akcija mapiranja dovrši obojat će se u drugačiju boju ovisno o uspješnosti završenog procesa (Slika 14. i Slika 15. u Prilogu). Popunjavanje ove vrijednosti pokreće se navedenom funkcijom *getProgress()*. Navedeni *iframe* element posjeduje nekoliko atributa. Osim toga, sadrži klasu koja čini element na početku nevidljivim:

```
.log {
      ...
      visibility: hidden;
}
```

Nadalje, postoji selektor identifikatora kojim smo dohvatili prethodni element na klik i promijenili mu CSS svojstvo, a koji ujedno posjeduje i atribute *name* i *src*. Vrijednost atributa *name* se povezuje s vrijednošću atributa *target* u formi o kojoj je bilo riječi, što znači da je ovo odredište u kojemu se želi dobiti povratna informacija poslužitelja nakon što završi procesuiranje forme. Atribut *src* označava putanju do dokumenta. *Iframe* je izrazito specifičan element s obzirom na to da on sadrži druge dokumente, u ovom slučaju dokument naziva *upload.php*. Dokument koji je

učitan u ovaj element je u stanju komunicirati s nadređenim elementom po principu: roditelj-dijete, s obzirom da jedan sadržava drugi. Datoteka *upload.php* ima početni dio koji predstavlja čisti PHP kôd nakon kojeg slijedi novi HTML dokument i u kojemu se nalazi već spomenuta rotirajuća ikona koja moli korisnika da pričeka kao i dio s prikazom napretka statusa mapiranja, no i još poneki dijelovi o kojima zasada nije bilo riječi. S obzirom da se izrazito puno događaja odvija upravo unutar *iframe* elementa, potrebno je naglasiti slijedeće: *iframe* element pozna putanju do dokumenta *upload.php* te se u njemu učitava HTML kôd iz te datoteke. HTML kôd je uvijek učitan, ali se vidi tek na klik gumba za mapiranje. Istim klikom se forma šalje poslužitelju. Nakon što procesuiranje završi, korisnik dobiva poruku, primjerice, da je neka datoteka mapirana i da je kreirana poveznica za njeno preuzimanje i sl., dok je u formi 'rečeno' da se taj status želi dobiti u elementu *log*. Element *log* je upravo spomenuti *iframe.* Dok poslužitelj nije završio s procesuiranjem datoteka, prikazivat će se rotirajuća ikona, a kada procesuiranje bude okončano, samo dokument unutar *iframea* će se osvježiti i prikazati. Situacija i nije toliko komplicirana, s obzirom na činjenicu da se jednostavno može razumjeti da će se klikom na gumb mapiranja, cjelokupna logika odviti u novoprikazanom elementu na stranici o čemu će biti riječi u narednom poglavlju.

5.5. Status mapiranja

Unutar *iframe* elementa nalazi se novi HTML dokument koji ima uobičajene sastavne dijelove kao što su *head* i *body.* Prvi dio u HTML dokumentu tiče se prikazivanja napretka u procesu mapiranja:

```
<div class="progress
<?php
    if ($_SESSION['progress'] == 100) {
        echo 'success';
    }
    else if (!empty($poruke) && $_SESSION['progress'] < 100) {
        echo "alert";
    }
?>
">
<span class="meter"></span>
</div>
```

Radi se o klasičnom *div* elementu koji ima unaprijed definirane klase koje služe za ovu svrhu a koje su definirane u *Foundationu.* Isto tako, *div* element ima početnu klasu naziva *progress* i ona je plave boje (Slika 13. u Prilogu). Ovdje postoje i uvjeti koji se tiču globalne varijable *$_SESSION*.

Ova globalna varijabla ima mogućnost pamćenja informacija koje se mogu koristiti iz bilo kojeg dijela PHP kôda na bilo kojoj stranici unutar programskog rješenja. Tipičan primjer primjene ove varijable je bilježenje informacija o autoriziranom korisniku nakon prijave koje omogućuju nesmetano pregledavanje zaštićenih mrežnih stranice programskog rješenja. Kada je prijava uspješno napravljena u *$_SESSION* se zapisuje tko je autoriziran korisnik. Nakon toga može se pregledavati programsko rješenje i na jednoj stranici, primjerice, ostaviti komentar. Na osnovi *$_SESSION* varijable može se provjeriti tko je autorizirani korisnik da bi se komentar mogao uz naziv korisnika spremiti u bazu podataka. Na ovom mjestu globalna varijabla *$_SESSION* služi tomu da se dobije uvid u fazu procesuiranja zahtijeva poslužitelju, a što znači da će se tijekom tog procesa, u nekom trenutku, zapisivati vrijednosti u ovu globalnu varijablu, a što će biti pokazano u nastavku rada. Ako vrijednost *$_SESSION* varijable dosegne vrijednost sto, tada će se nadodati nova klasa imena *success* koja će obojati cijeli *div* element u zeleno; u suprotnom, kada je vrijednost manja od sto, ali varijabla *$poruke* nije prazna, tada će *div* element biti crvene boje. S obzirom da se napredak prikazuje plavom bojom sve dok procesuiranje ne završi, logično je da se u navedenu varijablu spremaju određene vrijednosti koje postaju dostupne tek kada ovaj proces završi.

Sljedeći dio ovog HTML dokumenta je tablica:

```
<div class="row tablica">
	<div class="large-12 columns">
	<?php if(!empty($poruke)): ?>
		<table class="sirina">
		<thead>
			<th> Status mapiranja </th>
		</thead>
		<tbody>
		<?php foreach ($poruke as $p): ?>
			<tr><td><?php echo $p; ?></td></tr>
		<?php endforeach;
			endif;
		?>
		</tbody>
	</table>
	</div>
</div>
```

koja će se prikazati samo ako varijabla *$poruke* nije prazna. Ova varijabla će se 'puniti' za vrijeme procesa mapiranja i imat će vrijednosti s obzirom na uspješnost mapiranja pojedinog zapisa odnosno datoteke. Tablica će imati svoje zaglavlje s tekstom: „Status mapiranja“ (Slika 16. u Prilogu) i tijelom u kojemu će se u petlji ispisivati poruke po redovima; po jedna poruka u jednom

redu. Redovi koji će se ovdje ispisivati punit će se u PHP-u kada započne proces mapiranja. Ispod tablice će se nalaziti poveznice za pregledavanje i preuzimanje mapiranih datoteka (Slika 17. i Slika 18. u Prilogu), ako ih ima:

```
<?php if(!empty($downloadPoruka)): ?>
...
<?php foreach ($downloadPoruka as $poruka):
echo $poruka;
endforeach;
...
endif;?>
```

Ako je mapirana samo jedna datoteka, pojavit će se dvije poveznice. Jedna će biti za usporedno pregledavanje s početnom datotekom i završnom datotekom nakon procesa mapiranja. Druga poveznica je vezana uz preuzimanje datoteka s uspješno provedenim mapiranjem; ova poveznica ujedno je i jedina poveznica u slučajevima u kojima je mapirano više od jedne datoteke. Na koji način će se odvijati preuzimanje i na kojoj poveznici će se moći usporedno pregledavati priložena datoteka, kao i ona generirana u procesu mapiranja, bit će objašnjeno u dijelu u kojem će se dodavati vrijednosti ovoj varijabli.

Posljednji dio HTML dokumenta odnosi se na rotirajuću ikonu s pripadajućom porukom:

```
if (empty($poruke)): ?>
<div class="row spinner">
	<div class="large-12 columns">
		<p><i class="fa fa-spinner fa-spin"></i></p>
		<p>Molimo pričekajte...</p>
	</div>
</div>
<?php endif; ?>
```

koja će se prikazivati samo ako je varijabla *$poruke* prazna.

Dosad opisani dio odnosi se na sve ono što krajnji korisnik vidi kad pristupi programskom rješenju. U narednom poglavlju, bit će govora o PHP programskom kôdu koji se nalazi na strani poslužitelja koji je zadužen za kontroliranje priloženih datoteka te provedbu postupka njihovog mapiranja.

5.6. Početak procesa mapiranja

Klikom na gumb za početak mapiranja, forma se šalje poslužitelju metodom *POST* u datoteku *upload.php*. Ovaj gumb dobio je ime po formi koja se šalje: radi se o vrijednosti atributa *name* - *"mapping"*. Procesuiranje forme tako počinje s upitom:

```
if(isset($_POST['mapping']))
```

Prije ulaska u ovaj uvjet definiraju se funkcije i varijable koje će se koristiti. Kao primjerice:

```
$poruke = array();
$downloadPoruka = array();
session_start();
$_SESSION["progress"] = 0;
```

Varijable naziva *$poruke* i *$downloadPoruka* su tipa niz te započinje razmjena podataka odnosno sesija (engl. *session*) u kojoj globalna varijabla *$_SESSION["progress"]* na početku ima vrijednost nula, tipa cijeli broj. Dvije varijable tipa niz punit će se podacima te će upućivati na status mapiranja, i kasnije ispisati unutar tablice. Globalna varijabla uvijek će biti informirana o tome gdje je trenutno programski kôd došao sa svojom obradom. Uz navedeno, bitno je napomenuti kako će sve greške vezane uz procesuiranje XML-a kao datoteke također 'puniti' varijablu s porukama. Zahvaljujući tomu, program niti u jednom trenutku neće prestati s radom, ako dođe do eventualnih pogrešaka. Kako bi se pucanje programa zaustavilo koristi se funkcija:

```
libxml_use_internal_errors(true);
```

Ovim načinom spriječeno je da programski jezik sam ispisuje greške. Po potrebi, greške će se stavljati u varijablu za poruke i njihov prikaz bit će uređen. Funkcija koja je zaslužna za ovo je:

```
function libxml_display_errors(&$poruke) {
    $errors = libxml_get_errors();
    foreach ($errors as $error) {
     $poruke[] .= libxml_display_error($error);

    }
    libxml_clear_errors();
}
```

Kao ulazni parametar ova funkcija prima varijablu koja sadržava poruke, nakon čega se greške dohvaćaju, u petlji čitaju i stavljaju u niz. Stavljanje poruka u niz je također kontrolirano, pa se tako poziva nova funkcija:

```
function libxml_display_error($error) {
  $return = "Linija <span style='font-weight:bold'>$error->line</span>: ";
  $return .= "<span style='font-style:italic'>$error->message</span>";

    return $return;
}
```

koja će u jednom redu 'reći' na kojoj se liniji u XML-u nalazi greška i kako glasi, što se vraća prethodnoj funkciji koja će taj podatak staviti u varijablu za poruke. Nažalost, sâm tekst greške glasi na engleskom jeziku te se ne može prevesti, no u većini slučajeva i to je dovoljno kako bi se greška otklonila. U isto vrijeme, u tablici će se ispisivati crvene ikone koje će označavati da nešto nije u redu. Nakon ulaska u uvjet za početak mapiranja, programski kôd je sačinjen od brojnih kontrola.

5.6.1. Kontrole programskog kôda

Prve kontrole odnose se na broj datoteka koje je korisnik priložio za mapiranje. Sve datoteke priložene u formi nalaze se u PHP-u u globalnoj varijabli *$_FILES*. Prva kontrola se odnosi na slanje maksimalnog dozvoljenog broja datoteka (pet datoteka):

```
if (count($_FILES['upload']['name']) > 5) {
  $poruke[] = "<i class='fa fa-close'></i> Prešli ste maksimalan dozvoljeni broj
datoteka.";
 }
```

S obzirom da je omogućeno višestruko prilaganje datoteka, konzultira se veličina globalne varijable koja sadrži sve podatke, a to je moguće učiniti, jer ona predstavlja niz. Ako je njezina veličina veća od pet, u varijablu s porukama šalje se poruka da je korisnik prešao maksimalan dozvoljeni broj datoteka, dok će se ispred poruke nalaziti ikona s pogreškom (Slika 19. u Prilogu). Ako je ovaj uvjet zadovoljen, što znači da je priloženo više od pet datoteka, više se ne konzultira niti jedan uvjet te se korisniku ispisuje greška. Broj od maksimalno pet dopuštenih datoteka je odabran kao dovoljno velik broj da se dokaže da je moguće višestruko prilaganje i mapiranje, a dovoljno mali da se ne dopusti postavljanje prevelikog broja datoteka na poslužitelj koji je javno dostupan kako ne bi došlo do sigurnosnih propusta. Da se radi o komercijalnom programskom rješenju ovaj broj bi bio prilagođen potrebama; primjerice, u slučaju informacijskih ustanova koje imaju potrebu za ovakvim rješenjem taj bi broj bio sigurno puno veći, no u tom slučaju ne bi sve osobe s pristupom internetu imale pristup programskom rješenju.

Sljedeći uvjet provjerava postoje li uopće priložene datoteke:

```
else if (count($_FILES['upload']['name']) == 1 && $_FILES['upload']['name'][0] == "")
{
  $poruke[] = "<i class='fa fa-close'></i> Nemate priloženih datoteka.";
 }
```

Ako ovaj uvjet zadovoljen u varijablu s porukama se stavlja poruka da nema priloženih datoteka te se će se ta poruka ispisati u statusu mapiranja (Slika 20. u Prilogu). Ako niti jedan od prethodnih uvjeta nije zadovoljen, započinje konkretnije analiziranje priloženih datoteka. Prije svega kreira se direktorij na poslužitelju gdje će se datoteke pohranjivati.

```
$folderName = date('YmdHis', time()) . rand(0, 1000);
...
mkdir("./uploaded/" . $folderName . "/output", 0777, true);
```

Novi direktorij se stvara u direktoriju *uploaded* koji je u trenutku postavljanja projekta na poslužitelj prazan. Ime novonastalog direktorija se sastoji od brojeva trenutne godine, mjeseca, dana, i trenutnog sata, minute i sekunde uz dodatak nasumičnog broja između 0 i 1000. Nasumičan broj je postavljen s obzirom na to da se može dogoditi da dva ista korisnika u istoj sekundi započnu

mapiranje "pod istim brojem" zbog čega bi nastao problem u radu aplikacije. Dakako, teško je očekivati da bi se nešto slično moglo dogoditi, no nije i nemoguće tako da se o ovoj mogućnosti mora voditi računa. Unutar novonastalog direktorija s brojevima, a u koji će se stavljati priložene datoteke, kreira se još i direktorij naziva *output* u koji će se stavljati datoteke nakon procesa mapiranja. Kako su novonastali direktoriji javno dostupni, njihovo pregledavanje je moguće što znači i pregled svih ikad priloženih datoteka od strane korisnika koji za to ne bi trebali imati ovlasti iz sigurnosnih razloga (Slika 21. i Slika 22. u Prilogu). Naime, želi se spriječiti da bilo tko može pregledavati datoteke drugih korisnika. Iz tog razloga napravljena je datoteka *.htaccess* koja ima mogućnost konfiguriranja Apache poslužitelja. U njoj je zapisano da ako određeni direktorij nema *index* datoteku koja se uvijek prva otvara kada se pristupi cijelom direktoriju, Apache poslužitelj neće napraviti izlistanje direktorija, nego će pokazati poruku da je pristup zabranjen (Slika 23. u Prilogu).

Nakon što su direktoriji kreirani, započinje petlja koja će prolaziti kroz svaku od pojedinačnih priloženih datoteka:

```
for ($index = 0; $index < count($_FILES['upload']['name']); $index++) {
...
}
```

Svaka od priloženih datoteka će tako prolaziti kroz sljedeći niz kontrola. Prije svega želi se provjeriti je li priložena datoteka uopće XML. Iako se prilikom otvaranja dijaloškog okvira za odabir datoteka filtriraju samo XML-ovi bilo je govora kako je ovo jako jednostavno zaobići.

```
if (pathinfo($_FILES['upload']['name'][$index], PATHINFO_EXTENSION) == 'xml') {
...
}
else
$poruke[] = "<i class='fa fa-close'></i> Datoteka " .
$_FILES['upload']['name'][$index] . " nije XML.";
```

Ako datoteka nema ekstenziju *.xml* u poruke se dodaje poruka kako se ne radi o XML datoteci nakon čega petlja nastavlja s procesuiranjem eventualno drugih datoteka. Poruke se ne ispisuju odmah nego tek kada procesuiranje završi, a s obzirom da se može dogoditi da korisnik priloži jednu XML i zabunom neku drugu datoteku, u tom slučaju jedna datoteka će ući u uvjet da ima ekstenziju *.xml*, dok će procesuiranje druge prestati (Slika 24. u Prilogu). Ulaskom u uvjet da se radi o XML datoteci, ona će se pohraniti na poslužitelju u, za ovu priliku, stvorenom direktoriju. Ona mora biti pohranjena inače se neće moći programski otvoriti i procesuirati njezin sadržaj.

U ovom trenutku, započinje procesuiranje same datoteke. Prvi puta se instancira klasa *DomDocument:*

```
$dokument = new DOMDocument('1.0', 'UTF-8');
```

Nakon toga, slijedi pokušaj učitavanja datoteke u ovaj objekt.

```
if ($dokument->load($newFilePath)) {
..
}
else
	$poruke[] = "<i class='fa fa-close'></i> Datoteka " .
$_FILES['upload']['name'][$index] . " nije validan XML i ima sljedeće pogreške:";
	libxml_display_errors($poruke);
```

S obzirom da se dokument pokušava učitati kao *DOM*, sama formalna struktura XML-a mora biti ispravna. Svi elementi moraju biti pravilno zatvoreni, isto tako, jednom otvoren tag mora biti i zatvoren, moraju se slijediti pravila pisanja atributa i sl. Ako to nije zadovoljeno u varijablu s porukama će se dodati da XML nije validan s popisom svih grešaka (Slika 25. u Prilogu). Kada je učitan ispravan XML dokument, započinje logika koja pokušava raspoznati o kojoj se shemi metapodataka radi. To je moguće saznati preko URI-ja imenskog prostora koji se definira u atributu *xmlns* u korijenskom elementu. Ako shema u XML-u koristi prefiks, on će biti naveden kao dio istog atributa, prema sintaksi: *xmlns:prefiks="URI"*. Jedina stalna vrijednost je zapravo URI, dok za prefiks to nije slučaj. S tim u vezi su prije svega definirane dvije varijable čija je vrijednost URI imenskog prostora Dublin Core i MODS sheme:

```
$dcNamespaceUri = "http://purl.org/dc/elements/1.1/";
$modsNamespaceUri = "http://www.loc.gov/mods/v3";
```

Nakon toga je moguće provjeriti koristi li se u XML-u prefiks koji pripada jednom od navedenih imenskih prostora:

```
$dcPrefix = $dokument->lookupPrefix($dcNamespaceUri);
$modsPrefix = $dokument->lookupPrefix($modsNamespaceUri);
```

Ako se koristi prefiks koji pripada traženom imenskom prostoru, varijable *$dcPrefix* i *$modsPrefix* će sadržavati njegovu vrijednost, u protivnom njihova vrijednost je prazan niz znakova. U praksi je uobičajeno da Dublin Core shema metapodataka koristi prefiks *dc*, pa će varijabla *$dcPrefix* imati vrijednost *dc*. Potom slijede uvjeti koji će provjeriti vrijednost navedenih varijabli:

```
if ($dcPrefix != "" || $dokument->documentElement->getAttribute("xmlns") ==
$dcNamespaceUri) {
...
else if ($modsPrefix != "" || $dokument->documentElement->getAttribute("xmlns") ==
$modsNamespaceUri) {
...
}
```

Ako su varijable prazne, znači da se ne koristi prefiks za tražene imenske prostore te bi, ukoliko se uistinu radi o Dublin Core ili MODS shemi metapodataka, atribut *xmlns* trebao biti bez prefiksa i njegova vrijednost bi trebala biti jednaka URI-ju imenskog prostora i to je dodatna provjera u

uvjetima. Ukoliko nije pronađeno da se radi ili o MODS-u ili Dublin Coreu, odnosno uvjeti nisu zadovoljeni, ispisat će se greška kako datoteku nije moguće mapirati (Slika 26. u Prilogu):

```
else
$poruke[] = "<i class='fa fa-close'></i> Datoteku " .
$_FILES['upload']['name'][$index] . " nije moguće mapirati " .
"jer prema imenskom prostoru nije pronađeno da se radi o DC ili MODS shemi
metapodataka.";
```

U suprotnom, započinje njihova validacija prema pripadnim XML Schemama. Ovo je jedna od izuzetno bitnih funkcionalnosti samog programskog rješenja a koja bi se mogla koristiti i u druge svrhe. Vrlo često se nakon kreiranja XML dokumenta želi provjeriti je li taj dokument validan. Bez računalne obrade to je izrazito kompleksna zadaća, a uz upotrebu *DOMDocument* klase, ovaj problem može se vrlo jednostavno savladati. Samo je potrebno kao parametar proslijediti lokaciju XML Scheme, a one su ovdje proslijeđene sa svojih službenih lokacija.

Ako se zna da se radi o Dublin Core shemi, njen zapis validira se na sljedeći način:

```
  if
($dokument->schemaValidate('http://dublincore.org/schemas/xmls/qdc/2008/02/11/simpledc
.xsd')) {
       include "dc_mods.php";
} else
       $poruke[] = "<i class='fa fa-close'></i> Datoteka " .
       $_FILES['upload']['name'][$index] . " nije valjana prema DC shemi i ima sljedeće
       pogreške:";
       ibxml_display_errors($poruke);
```

MODS zapis validira se pak na način:

```
if ($dokument->schemaValidate('http://www.loc.gov/standards/mods/v3/mods-3-6.xsd')) {
       include "mods_dc.php";
} else
       $poruke[] = "<i class='fa fa-close'></i> Datoteka " .
       $_FILES['upload']['name'][$index] . " nije valjana prema MODS shemi i ima
       sljedeće pogreške:";
libxml_display_errors($poruke);
```

Ako dokument ne prođe validaciju, pogreške će se staviti u varijablu s porukama i pokazati korisniku (Slika 27. u Prilogu). Nakon što je dokument uspješno prošao sve navedene kontrole započinje njegovo mapiranje.

5.6.2. Mapiranje Dublin Core sheme u MODS

Proces mapiranja može se pratiti u datoteci *dc_mods.php* čiji je sadržaj uključen u procesuiranje ako se uspješno prođu kontrole. Jedini razlog zašto svako mapiranje proizvodi vlastitu datoteku je

zbog preglednosti i, s tim u vezi, jednostavnijeg programiranja. Na početku, otvara se prazan dokument:

```
$novi_dokument = new DOMDocument('1.0', 'UTF-8');
```

U njega se prvo zapisuje korijenski element s deklaracijom sheme metapodataka i ostalim potrebnim atributima:

```
$rootElement = $novi_dokument->createElementNS('http://www.loc.gov/mods/v3', 'mods');
$novi_dokument->appendChild($rootElement);
$rootElement->setAttributeNS('http://www.w3.org/2000/xmlns/' ,'xmlns:xsi',
'http://www.w3.org/2001/XMLSchema-instance');
$rootElement->setAttributeNS('http://www.w3.org/2001/XMLSchema-
instance' ,'schemaLocation', 'http://www.loc.gov/mods/v3
http://www.loc.gov/standards/mods/v3/mods-3-5.xsd');
$rootElement->setAttributeNS('http://www.w3.org/2000/xmlns/' ,'xmlns:xlink',
'http://www.w3.org/1999/xlink');
$rootElement->setAttribute('version', '3.5');
```

što rezultira *mods* korijenskim elementom i njegovim pripadajućim atributima:

```
<mods xmlns="http://www.loc.gov/mods/v3" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xlink="http://www.w3.org/1999/xlink"
xsi:schemaLocation="http://www.loc.gov/mods/v3
http://www.loc.gov/standards/mods/v3/mods-3-5.xsd" version="3.5">
...
</mods>
```

Nakon što je kreiran korijenski element datoteke, može se započeti čitati priložena datoteka te u novu datoteku zapisivati rezultat postupka mapiranja. Prvo se dohvate sve vrijednosti podređenih elemenata korijenskog elementa i počinje njihovo čitanje u petlji.

```
$elementi = $dokument->documentElement->childNodes;
foreach ($elementi as $element) {
...
}
```

Sva daljnja logika se događa u ovoj petlji, a s obzirom da je Dublin Core shema metapodataka s elementima na istoj razini, odmah se dolazi do elemenata koje treba mapirati. Kako bi se nazivi elemenata razlikovali, za Dublin Core će se u nastavku koristiti uobičajeni prefiks *dc*, dok MODS neće imati prefiks. Prema službenoj specifikaciji mapiranja ove dvije sheme metapodataka,[52] element *dc:title* mapira se u *<titleInfo><title>* na sljedeći način:

```
if ($element->nodeName == $dcPrefix . "title") {
 ...
}
```

Ako je uvjet zadovoljen, kreirat će se element *titleInfo* u novom dokumentu i on će se dodati

[52] Usp. Library of Congress. Dublin Core Metadata element set mapping to MODS version 3, 2012. URL: http://www.loc.gov/standards/mods/dcsimple-mods.html (2016-09-20)

korijenskom elementu. Nakon njega kreirat će se i element *title*, s vrijednošću elementa iz petlje, u ovom slučaju to je vrijednost elementa *dc:title*. Kreirani element sa svojom vrijednošću se dodaje elementu *titleInfo*, kao u primjeru:

```
$titleInfo = $novi_dokument->createElement("titleInfo");
$rootElement->appendChild($titleInfo);
$title = $novi_dokument->createElement("title", $element->nodeValue);
$titleInfo->appendChild($title);
```

Elementi *dc:creator* i *dc:contributor* mapiraju se na isti način. U MODS-u se ovi elementi mapiraju u okviru ponavljajućeg elementa *name* koji sadrži element *role*. Kao posebna napomena navodi se da MODS podrazumijeva strukturiranu vrijednost imena te dopušta korištenje svih podređenih elemenata, primjerice elementa *name* kao što su to *displayForm* ili *namePart*.[53] Iz razloga što MODS zahtijeva strukturiranu vrijednost imena, ovo programsko rješenje u ovom dijelu rješava izuzetno kompleksan slučaj. Kako je već bilo riječi da je nemoguće programski osmisliti logiku koja bi strukturirala ime prema pravilima koja se moraju zadovoljiti, dolazi se do zaključka kako niti ne treba osmišljavati neko drugo rješenje, nego se može konzultirati autoritet koji će vratiti strukturiranu vrijednost. Prilikom samog razvoja to se pokušalo riješiti na nekoliko načina. Prvi način bio je pristup OCLC-ovom aplikacijskom programskom sučelju koje programerima daje podršku za pristup normativnim datotekama Kongresne knjižnice.[54] Ako se želi pretraživati autor Juraj Mulih, programer bi poslao upit na adresu:
http://alcme.oclc.org/srw/search/lcnaf?query=local.FamilyName+%3D+%22Mulih%22+and+local.FirstName+%3D+%22Juraj%22
Prvi problem koji se ovdje uočava odnosi se na činjenicu da sam upit zahtijeva razlikovanje imena i prezimena autora, što bi eventualno i bilo rješivo logičkim operatorima unutar samog upita. Kao odgovor poslužitelja na upit dobivamo MARCXML datoteku koju treba procesuirati. Puno su veći problemi uočeni u dijelu u kojemu poslužitelj vraća svoj odgovor. Naravno, problem postaju hrvatski dijakritički znakovi. Čak niti konverzijom u UTF-8 vrijednosti unutar dokumenta nisu se mogle izmijeniti. S obzirom na dva velika problema na koja se naišlo, potraženo je drugo rješenje. Zašto bi se uopće koristio okolni pristup normativnim datotekama Kongresne knjižnice, kada je to moguće izravno napraviti. Jednostavnim pretraživanjem uočeno je kako je prvi vraćeni rezultat upravo onaj koji je potreban kako bi programsko rješenje funkcioniralo. Za autora Juraja Muliha odgovor bi bio: Mulih. Juraj, 1694-1754.[55] Prilikom razmatranja kako ovaj podatak strojno

[53]Usp. Isto.
[54]Usp. OCLC.org. LC Name Authority File (LCNAF), 2015. URL: https://www.oclc.org/developer/develop/web-services/lc-name-authority-file-lcnaf.en.html (2016-09-20)
[55]Usp. Library of Congress. Search Results. URL: http://id.loc.gov/search/?q=mulih+juraj&q=cs%3Ahttp%3A%2F%2Fid.loc.gov%2Fauthorities%2Fnames (2016-09-

obraditi, pregledavanjem izvornog kôda uočeno je da Kongresna knjižnica sve rezultate može vratiti u ATOM/XML formatu, a s obzirom da su svi podaci normativne datoteke javno dostupni,[56] zaključeno je kako će se upiti za formatirane vrijednosti u MODS-u slati upravo ovdje.

Dinamičan upit izgleda ovako:

```
http://id.loc.gov/search/?q=' . $element->nodeValue .
'&q=cs:http://id.loc.gov/authorities/names&start=1&count=1&format=atom
```

gdje je *$element->nodeValue* vrijednost iz Dublin Corea. Kako je vraćeni rezultat u XML obliku, sadržaj se učitava u objekt i iz njega se dohvaća potrebna vrijednost. Iz samog upita je vidljivo kako se vraća uvijek samo jedan rezultat, jer se pretpostavlja da će točan naziv koji se pohranjuje u Dublin Core zapis biti prvi u rezultatima prilikom pretraživanja, no to nije uvijek slučaj. Ovisno o broju rezultata, moguće je da se dogodi da se procesuira krivi podatak, no to ide u prilog tezi kako je na kraju svakog automatskog mapiranja ipak potrebna mala ljudska intervencija. S obzirom da se ovaj slučaj događa u izuzetno rijetkim slučajevima, ovaj problem u okviru diplomskog rada se smatra zanemarivim. Ako Kongresna knjižnica vrati rezultat, primjer s navedenim autorom će biti mapiran:

```
<name type="personal" authority="lcnaf">
<namePart>Mulih, Juraj</namePart>
<namePart type="date">1694-1754</namePart>
<displayForm>Mulih, Juraj, 1694-1754</displayForm>
<role>
<roleTerm type="text" authority="marcrelator">Creator</roleTerm>
</role>
</name>
```

Autoritet preuzetih podataka navodi se u elementu *name* a u elementu *role* dodana je indikacija da se radi o mapiranoj Dublin Core vrijednosti iz elementa *dc:creator*. Vrijednost elementa *roleTerm* je kontrolirana te se navodi autoritet gdje je ona definirana.[57] Tijekom postupka mapiranja ova vrijednost može biti „Creator“ ili „Contributor“. Ako Kongresna knjižnica ne vrati rezultate, mapirana vrijednost će biti:

```
<name>
<namePart>Antun Matanović</namePart>
<role>
<roleTerm type="text" authority="marcrelator">Creator</roleTerm>
</role>
</name>
```

Sljedeći element u Dublin Coreu je *dc:subject* čija vrijednost se uvijek mapira u *<subject><topic>:*

---

20)

[56] Usp. Library of Congress. Legal. URL: https://www.loc.gov/legal/ (2016-09-20)

[57] Usp. Library of Congress. Top-level element: <name>, 2016. URL: https://www.loc.gov/standards/mods/userguide/name.html (2016-09-20)

```
if ($element->nodeName == $dcPrefix . "subject") {
       $subject = $novi_dokument->createElement("subject");
       $rootElement->appendChild($subject);
       $topic = $novi_dokument->createElement("topic", $element->nodeValue);
       $subject->appendChild($topic);
    }
```

Prilikom mapiranja vrijednosti elementa *dc:description* gube se neki specifični elementi koje MODS može pokriti te se izravno mapira u element *note:*

```
if ($element->nodeName == $dcPrefix . "description") {
       $note = $novi_dokument->createElement("note", $element->nodeValue);
       $rootElement->appendChild($note);
    }
```

Vrijednosti sljedeća dva elementa, *dc:publisher* i *dc:date* će se mapirati zajedno. Mapiraju se na način da se kreira element *originInfo*, te se u njega dodaje element *publisher* koji će pak primiti vrijednost elementa *dc:publisher*, a u isti *originInfo* će se dodati element *dateOther* za vrijednost elementa *dc:date.* Dva Dublin Core elementa će tako biti mapirana u isti *originInfo* jer to dopušta MODS-ova XML Schema,[58] Ipak, prvo je potrebno pretražiti dokument i vidjeti postoji li već kreirani *originInfo*, a ako ne postoji, onda ga prvo kreirati, pa tek onda u njega dodati elemente. Ako *originInfo* već postoji, u isti se samo dodaje novi element kao u primjeru:

```
  if ($element->nodeName == $dcPrefix . "publisher" || $element->nodeName ==
$dcPrefix . "date") {
       $search_originInfo = $novi_dokument->getElementsByTagName('originInfo');
       if ($search_originInfo->length == 0) {
       $originInfo = $novi_dokument->createElement("originInfo");
       $rootElement->appendChild($originInfo);
       }
       if ($element->nodeName == $dcPrefix . "publisher") {
            $publisher = $novi_dokument->createElement("publisher",
$element->nodeValue);
            $originInfo->appendChild($publisher);
        }
        else if ($element->nodeName == $dcPrefix . "date") {
            $dateOther = $novi_dokument->createElement("dateOther",
$element->nodeValue);
            $originInfo->appendChild($dateOther);
        }
    }
```

Mapiranje vrijednosti elementa *dc:type* je nešto specifičnije s obzirom da MODS za tipove izvora

[58]Usp. Library of Congress. Top-level element: <originInfo>, 2013. URL: http://www.loc.gov/standards/mods/userguide/origininfo.html (2016-09-20)

koristi kontroliranu listu koju je potrebno pokriti u automatskom mapiranju.[59] Ako je vrijednost jednaka vrijednosti u kontroliranoj listi, dodaje se element *typeOfResource* s kontroliranom vrijednošću te element *genre* koji će imati atribut *authority="dct"*. Primjer vrijednosti iz kontrolirane liste je pokriven na način:

```
else if (strtoupper($element->nodeValue) == "DATASET") {
	$typeOfResource = $novi_dokument->createElement("typeOfResource", "software, multimedia");
	$genre = $novi_dokument->createElement("genre", "database");
	$genre->setAttribute('authority', 'dct');
	$rootElement->appendChild($typeOfResource);
	$rootElement->appendChild($genre);
	}
```

U ovom slučaju, s obzirom da se uspoređuje niz znakova, uobičajena je praksa u programiranju da se znakovi pretvore u velika slova kako bi uspoređivanje bilo točnije s obzirom na različite prakse pisanja vrijednosti. Ako se vrijednost *dc:type* ne nalazi u kontroliranoj listi, onda se samo dodaje element *genre* a vrijednost prepisuje:

```
else {
$genre = $novi_dokument->createElement("genre", $element->nodeValue);
$rootElement->appendChild($genre);
		}
```

Vrijednost elementa *dc:format* se mapira u element *physicalDescription* koji ima različite podređene elemente ovisno o vrijednosti. Logika je napravljena na sljedeći način: ako niz znakova kao svoju vrijednost sadrži znak „/“ podređeni element će biti *internetMediaType*, kao što je to u primjeru: „text/html“.:

```
$vrijednost = $element->nodeValue;
	$uvjet = "/";
	$rezultat = strpos($vrijednost, $uvjet);
	if ($rezultat !== false) {
		$internetMediaType = $novi_dokument->createElement("internetMediaType", $element->nodeValue);
		$physicalDescription->appendChild($internetMediaType);
	}
```

Ako niz znakova vrijednosti započinje s brojevima, tada će podređeni element biti *extent* jer prema dokumentaciji element *extent* uvijek započinje tako,[60] stoga, primjer njegova kreiranja izgleda ovako:

```
for ($i = 1; $i < 10; $i++) {
	if (substr($vrijednost, 0, 1) == $i) {
```

[59]Usp. Library of Congress. Nav. dj.
[60]Usp. Library of Congress. Top-level element: <physicalDescription>, 2015. URL: https://www.loc.gov/standards/mods/userguide/physicaldescription.html (2016-09-20)

```
	$extent = $novi_dokument->createElement("extent", $element->nodeValue);
$physicalDescription->appendChild($extent);
	}
}
```

Ako nije stvoren niti jedan od ova dva elementa stvara se standardni element *form:*

```
$search_internetMediaType = $novi_dokument->getElementsByTagName('internetMediaType');
	$search_extent = $novi_dokument->getElementsByTagName('extent');
	if ($search_internetMediaType->length == 0 && $search_extent->length == 0) {
		$form = $novi_dokument->createElement("form", $element->nodeValue);
		$physicalDescription->appendChild($form);
	}
```

Vrijednosti elemenata *dc:source* i *dc:relation* se mapiraju na isti način. Za oba elementa kreira se element *relatedItem* te ako vrijednost započinje s: „http://“ kreiraju se podređeni element *location*, s podređenim elementom *url*:

```
if (substr($element->nodeValue, 0, 7) == "http://") {
		$location = $novi_dokument->createElement("location");
			$relatedItem->appendChild($location);
			$url = $novi_dokument->createElement("url", $element->nodeValue);
			$location->appendChild($url);
	}
```

Ako vrijednost ne počinje s „http://“ tada se stvaraju elementi *titleInfo* i *title.* Poseban dodatak je što mapirani *relatedItem* iz *dc:source* uvijek sadrži atribut *type="original"*. Vrijednost Dublin Core elementa *dc:language* može biti napisana u obliku kôda iz ISO standarda ili kao niz znakova. U skladu s tim, vrijednost ovog elementa u MODS-u mapira se s nadređenim elementom *language* i podređenim elementom *languageTerm* koji sadrže atribut *type* a čija vrijednost može biti *text* ili *code.* Kako bi se provjerilo je li jezik napisan u obliku kôda, konzultira se izvor ISO639-2b u JSON formatu.[61] Ako se pronađe kodni termin, u elementu *languageTerm* će se dodati atribut *authority="iso639-2b"*.Tako bi mapirane vrijednosti Dublin Core elementa *dc:language* izgledale kao u primjeru:

```
<language>
<languageTerm type="text">hrvatski</languageTerm>
<languageTerm authority="iso639-2b" type="code">hrv</languageTerm>
</language>
```

Sljedeći element Dublin Corea je *dc:identifier* čije je mapiranje u MODS dosta složeno s obzirom da postoji jako puno identifikatora i da je teško pokriti sve automatskim mapiranjem. No neki imaju određeni format pa je razvijena logika koja će ovaj aspekt svaki put provjeravati.[62] Najčešći

[61] Originalna lokacija ISO639-2b jezične sheme je http://id.loc.gov/vocabulary/iso639-2.madsrdf.json
[62]Za sličnu logiku koja je pomogla da se ovaj dio razvije usp. Library of Congress. MODS XLST 1.0 Stylesheets: DC to MODS 3.5. URL: http://www.loc.gov/standards/mods/v3/DC_MODS3-5_XSLT1-0.xsl (2016-09-20)

identifikator je obična URL adresa koja će biti mapirana, ako se prepozna, u element *location* s podređenim elementom *url.* Ako se ne radi o URL adresi kreirat će se element *identifier* s atributom *type*, čija će vrijednost ovisiti o daljnjim provjerama. Ako niti jedna provjera ne prođe, njegova vrijednost će biti, jednostavno, *local.* Ako je pak pronađeno da je određeni identifikator jednak vrijednosti elementa *dc:identifier*, naziv identifikatora, napisan malim slovima, postat će vrijednost atributa *type.* Na ovaj način se provjerava radi li se o URI-ju (engl. *Uniform Resource Identifier*), HDL-u, DOI-ju (engl. *Digital Object Identifier*), ARK-u (engl. *Archival Resource Key*), PURL-u (engl. *Persistent uniform resource locator*), ISBN-u od 10 ili 13 znakova, ISRC-u (engl. *The International Standard Recording Code*), ISSN-u (engl. *International Standard Serial Numbers*), ISTC-u (engl. *The International Standard Text Code*) te SICI-u (engl. *The Serial Item and Contribution Identifier*). Većina ovih logika je zapravo vrlo jednostavna, jer se zasniva na standardnom broju znakova, dok velika većina započinje upravo nazivom identifikatora i sl. Tako bi primjerice mapirana vrijednost: „978-953-341-022-7“ bila:

```
<identifier type="isbn">978-953-341-022-7</identifier>
```

Iz navedenih vrsta identifikatora dodatnu provjeru zahtijeva ISRC, jer su njegova prva dva znaka uvijek kôd države prema ISO 3166. Iz tog razloga, u programsko rješenje uključen je JSON koji sadrži sve kodove država prema ovom standardu. Mapirani primjer ovog specifičnog tipa identifikatora bio bi:

```
<identifier type="isrc">BRBMG0300729</identifier>
```

Pretposljednji Dublin Core element je *dc:coverage* čija se vrijednost najčešće mapira u element *subject* s podređenim elementom *geographic*, iako ovo mapiranje i nije najtočnije. Kao dodatna provjera pokušava se otkriti radi li se, primjerice, o omjerima koji uvijek započinju s „1:“ ili pak koordinatama. Ako se pokaže da se radi o koordinatama, tada će se u *subject* dodati podređeni element *cartographics* koji u sebi ima podređeni element *scale* ili *coordinates.* Posljednji Dublin Core element je *dc:rights* koji je jedan od rijetkih gdje se ne vrši provjera vrijednosti te se može izravno mapirati u element *accessCondition.* Nakon završetka petlje, datoteka se pohranjuje fizički na poslužitelj u direktorij kreiran za trenutno mapiranje s nazivom izvorne datoteke, ali i dodatkom *„_output“*.

```
$novi_dokument->save('uploaded/' . $folderName . '/output/' .
$saveName .'_output.xml');
```

Nakon toga u varijablu s porukama dodaje se poruka da je datoteka uspješno mapirana (Slika 28. u Prilogu):

```
$poruke[] = "<i class='fa fa-check'></i> Datoteka " .
$_FILES['upload']['name'][$index] . " je uspješno mapirana iz DC-a u MODS.";
```

5.6.3. Mapiranje MODS-a u Dublin Core

Mapiranje MODS-a u Dublin Core je puno jednostavniji postupak. Razlog tomu je jednostavna struktura samo jedne razine elemenata te mali broj elemenata Dublin Core sheme metapodataka, tako da se iz MODS-a gubi jako puno informacija. Iz tog razloga je i programski kôd puno manji. Najveći problem je razgranata struktura MODS-a gdje je potrebno dinamički prolaziti kroz sve razine dokumenta da bi se otkrilo koji su elementi podređeni drugim elementima i sl.. Na taj način je izbjegnuto višestruko provjeravanje koje se ovim putem obavlja odjednom. Mapiranje iz MODS-a u Dublin Core se događa u datoteci *mods_dc.php*. Mapiranje započinje kreiranjem novog XML dokumenta s korijenskim elementom *simpledc* koji će u sebi sadržavati atribute vezane za deklaraciju sheme metapodataka i imenskog prostora:

```
$novi_dokument = new DOMDocument('1.0', 'UTF-8');
$rootElement = $novi_dokument->createElement('simpledc');
$novi_dokument->appendChild($rootElement);
$rootElement->setAttributeNS('http://www.w3.org/2000/xmlns/' ,'xmlns:dc',
'http://purl.org/dc/elements/1.1/');
$rootElement->setAttributeNS('http://www.w3.org/2000/xmlns/' ,'xmlns:xsi',
'http://www.w3.org/2001/XMLSchema-instance');
$rootElement->setAttributeNS('http://www.w3.org/2001/XMLSchema-
instance' ,'noNamespaceSchemaLocation',
'http://dublincore.org/schemas/xmls/qdc/2008/02/11/simpledc.xsd');
```

Nakon kreiranja korijenskog elementa, u petlji se prolazi kroz sve MODS elemente te se utvrđuje glavna logika mapiranja. Prvi element je *titleInfo* iz koje se uzima vrijednost prvog podređenog elementa, koji je uvijek *title*, te se ona mapira kao vrijednost elementa *dc:title.*

```
if ($element->nodeName == $modsPrefix . "titleInfo") {
       $title = $novi_dokument->createElement("dc:title",
$element->childNodes->item(1)->nodeValue);
       $rootElement->appendChild($title);
    }
```

Ovdje je, primjerice, specifično to da kada petlja dođe do elementa *titleInfo* uzima se vrijednost njegovog drugog „djeteta“. To je dio:

```
$element->childNodes->item(1)->nodeValue
```

Razlog tomu je što je cijeli dokument učitan kao DOM koji bi kao prvi element vratio uvlaku u XML dokumentu. To je uvlaka ili prazan prostor podređenog elementa kako bi se održala preglednost XML dokumenta. Naziv tog elementa je *„#text“*. To je razlog zašto se uzima vrijednost drugog elementa te će se na ovakve i slične slučajeve prilikom mapiranja MODS-a često nailaziti. Sljedeći element je *name* koji može sadržavati nekolicinu drugih elemenata. Prema

dokumentaciji[63] preporuča se uzimanje vrijednosti svih podređenih elementa razdvojenih nekim oblikom punktuacije. S obzirom na ovu preporuku uzima se samo vrijednost elementa *displayForm*, ako on postoji s obzirom na to da on velikom većinom sadrži cjelokupno ime, prezime i datum rođenja autora u njegovoj strukturiranoj formi. Ako ovaj element ne postoji, uzimaju se sve vrijednosti elemenata razdvojenih zarezom. S obzirom na to da se ne zna koliko će biti podređenih elemenata, to se utvrđuje sljedećom petljom:

```
for ($i = 0; $i < $childElements->length; $i++) {
if ($childElements->item($i)->nodeName !== "#text" &&
$childElements->item($i)->nodeName !== $modsPrefix . "role" &&
$childElements->item($i)->nodeName !== $modsPrefix . "displayForm") {
$values[] = $childElements->item($i)->nodeValue;
$value = implode(", ", $values);
}
else if ($childElements->item($i)->nodeName !== "#text" &&
$childElements->item($i)->nodeName !== $modsPrefix . "role" &&
$childElements->item($i)->nodeName == $modsPrefix . "displayForm") {
$value = $childElements->item($i)->nodeValue;
}
}
```

Gornja petlja će naići i na element *role* koji se zaobilazi, jer će se njega posebno dohvatiti u novoj petlji kako bi se saznalo sadrži li njegov podređeni element vrijednost koja odgovara stvaratelju ili suradniku te će se vrijednost iz prve petlje mapirati s obzirom na rezultat druge petlje.

MODS element *subject* ima mnoštvo podređenih elemenata koji ovise o tomu hoće li se mapiranje izvršiti u *dc:subject* ili *dc:coverage*. Vrijednosti podređenih elemenata *topic*, *name* i *occupation* mapiraju se u *dc:subject*, dok se vrijednosti elemenata *geographic*, *temporal*, *hierarchicalGeographic* te *cartographics* mapiraju u *dc:coverage.* Ako ima više podređenih elemenata oni se odvajaju zarezom. U *dc:subject* se uz ove elemente mapira još i vrijednost elementa *classification*.

Vrijednosti elemenata *abstract*, *note* i *tableOfContents* se uvijek mapiraju u *dc:description* kao u primjeru:

```
if ($element->nodeName == $modsPrefix . "abstract" || $element->nodeName ==
$modsPrefix . "note" || $element->nodeName == $modsPrefix . "tableOfContents") {
	$description = $novi_dokument->createElement("dc:description",
$element->nodeValue);
$rootElement->appendChild($description);
}
```

U elementu *originInfo* se nalaze podaci o datumu i nakladniku; ako ovaj element ima podređeni

[63]Usp. Library of Congress. MODS to Dublin Core metadata element set mapping: Version 3, 2012. URL: http://www.loc.gov/standards/mods/mods-dcsimple.html (2016-09-20)

element *publisher* njegova vrijednost će se mapirati u element *dc:publisher*

```
if ($element->nodeName == $modsPrefix . "originInfo") {
$childElements = $element->childNodes;
foreach ($childElements as $childElement) {
       if ($childElement->nodeName == $modsPrefix . "publisher") {
       $publisher = $novi_dokument->createElement("dc:publisher",
$childElement->nodeValue);
       $rootElement->appendChild($publisher);
       }
...
       }
     }
```

Vrijednosti podređenih elemenata vezanih uz datum se mapiraju u element *dc:date:*

```
else if ($childElement->nodeName == $modsPrefix . "dateIssued" ||
$childElement->nodeName == $modsPrefix . "dateCreated" || $childElement->nodeName ==
$modsPrefix . "dateCaptured" || $childElement->nodeName == $modsPrefix . "dateOther")
{
$date = $novi_dokument->createElement("dc:date", $childElement->nodeValue);
                    $rootElement->appendChild($date);
}
```

Sljedeća dva MODS elementa dolaze zajedno, to su *typeOfResource* i *genre* i njihove vrijednosti mapiraju se u *dc:type*, osim u slučaju kada *genre* ima atribut *authority* s vrijednošću *dct* koja označava da se radi o vrijednosti elementa iz kontrolirane liste termina. Tada se mapira samo element *genre*, a *typeOfResource* se izostavlja i njegova vrijednost se gubi. Nadalje, vrijednost podređenih elemenata *physicalDescription*-a, kao što su *form, extent* i *internetMediaType* uvijek se mapiraju u *dc:format* bez pravljenja ikakvih razlika. Vrijednost elementa *location* sa svojim podređenim elementom *url* uvijek se mapira u *dc:identifier*, jednako kao i vrijednost MODS-ovog elementa *identifier.* Slijedi element *relatedItem* koji može sadržavati bilo koji drugi MODS-ov element, a prema dokumentaciji i uputama navodi se preporuka da se mogu mapirati samo neki elementi kao što su *titleInfo, identifier* i *location*[64]. Na ovaj način, programsko rješenje mapira sve navedene primjere elemenata prema njihovim pripadajućim logikama. Za posljednja dva elementa MODS sheme metapodataka, *accessCondtion* i *recordInfo*, vrijedi da se njihove vrijednosti mapiraju u *dc:rights.*

S obzirom na navedeno, vidljivo je kako je mapiranje MODS-a u Dublin Core nešto jednostavnije. Nakon što petlja prođe kroz cijeli dokument, novi XML dokument će se spremiti sa svojim izvornim imenom i nastavkom: *„_output“*:

```
$novi_dokument->save('uploaded/' . $folderName . '/output/' .
```

[64]Usp. Library of Congress. Nav. dj.

```
$saveName .'_output.xml');
```

Nakon toga se u varijablu s porukama dodaje da je datoteka uspješno mapirana iz MODS-a u Dublin Core (Slika 29. u Prilogu).

```
$poruke[] = "<i class='fa fa-check'></i> Datoteka " .
$_FILES['upload']['name'][$index] . " je uspješno mapirana iz MODS-a u DC.";
```

5.7. Završetak procesa mapiranja

Po završetku procesa mapiranja preostaje jedino još korisniku osigurati pristup poveznicama za preuzimanje datoteka i eventualni pregled. Ako je mapirana samo jedna datoteka, u statusu mapiranja će se pokazati dvije poveznice (Slika 30. u Prilogu), a u slučajevima više mapiranih datoteka dobit će se samo jedna, ona za preuzimanje svih datoteka u komprimiranom formatu pohrane (Slika 31. u Prilogu). Kako bi se provjerilo da mapirane datoteke stvarno postoje na strani poslužitelja i da je postupak mapiranja dovršen, skenira se direktorij:

```
$mapiraneDatoteke = scandir("uploaded/". $folderName . "/output", 1);
```

Skeniranje će vratiti niz, na način da se broj mapiranih datoteka može saznati iz dužine niza:

```
$brojMapiranihDatoteka = count($mapiraneDatoteke);
```

Bitno je napomenuti da skeniranje direktorija vraća prema zadanim postavkama uvijek dva podatka, pa čak i ako nema niti jedne datoteke u njemu. To je:

- .
    - Podatak koji označava trenutni direktorij
- ..
    - Podatak koji označava direktorij iznad onoga u kojemu se trenutno nalazimo

Na ovaj način, prilikom provjere postoji li samo jedna datoteka treba postaviti:

```
if ($brojMapiranihDatoteka == 3) {
...
}
```

Svaka od poveznica ima svoj selektor identifikatora prema kojemu će se znati što je potrebno dobiti kao rezultat. Klikom na pregled datoteka otvara se nova kartica u pregledniku čiji URL sadrži određene parametre kao u primjeru:

pregled.php?id=20161013221529922&name=abecevica_mods_output

U navedenom primjeru je *id* naziv direktorija u kojemu su datoteke mapirane, dok se *name* odnosi na ime mapirane datoteke koja se želi pregledati. Otvorena stranica ima identičan izgled onomu na početnoj stranici koji sadrži primjere mapiranja (Slika 32. u Prilogu).

Klikom na preuzimanje datoteka šalje se AJAX zahtjev datoteci *download.php* koja će prema poslanim parametrima u POST-u znati što treba napraviti:

```
var folder = $("#folder").val();
        $.ajax({
        type: "POST",
        url: "download.php",
        data: "folder=" + folder + "&vrsta=" + $(".download").attr("id"),
        success: function(msg){
          window.parent.location.href= msg;
          parent.closeIFrame();
        }
      });
```

Ako POST parametar *vrsta* sadrži vrijednost *zip* znat će se da je potrebno kreirati *zip* datoteku. Ona se kreira također skeniranjem direktorija i korištenjem klase *ZipArchive* koja će stvoriti datoteku *output.*zip te će klijentu vratiti putanju do nje. Dio s kreiranjem *zip* datoteke se preskače ako je mapirana samo jedna datoteka; u tom slučaju, poslužitelj će vratiti samo putanju do mapirane XML datoteke. Većinom je XML datoteke moguće otvoriti u pregledniku, no programsko rješenje će forsirati preuzimanje. U skladu s tim, korisniku će se otvoriti dijaloški okvir za preuzimanje. Ovime je cjelokupni proces mapiranja završio, sve se vraća na početno stanje i mapiranje može opet započeti.

6. Zaključak

Razvoj programskog rješenja za mapiranje shema metapodataka je izrazito složen proces. Zahtijeva dubinsko poznavanje problematike metapodataka i mapiranih shema metapodataka, kao i poznavanje programiranja. Svrha rada je bila razvoj programskog rješenja koje će biti jedinstveno u svijetu, s obzirom da trenutno ne postoje slični alati na mreži. Ciljevi rada su bili izrada prednjeg i pozadinskog sustava programskog rješenja, izrada kontrola u pozadinskom sustavu koje će dopustiti mapiranje samo validnih Dublin Core i MODS XML datoteka te krajnji prikaz statusa mapiranja uz mogućnost preuzimanja rezultata mapiranja putem posebnih izlaznih datoteka. Sam razvoj programskog rješenja nije ograničen na jedan programski jezik, to je stvar odabira samog programera ili ustanove, ali je za ovaj rad odabran PHP kao glavni programski jezik za izradu programskog rješenja na *backend* strani. Na strani klijenta to je JavaScript uz HTML i CSS. Razvijeno programsko rješenje u ovom radu omogućuje mapiranje Dublin Core zapisa u MODS zapise te obrnuto. S tim u vezi, da bi mapiranje moglo započeti potrebno je imati definirane imenske prostore u datotekama koje se prilažu za mapiranje. Moguće je višestruko prilaganje datoteka uz kombinaciju shema, a aplikacija će sama prepoznati što je potrebno napraviti. Jednom

započeto mapiranje prolazit će brojne kontrole, kao što je to npr. provjera ispravnosti sintakse XML-a ili pak validacija shema prema pripadnim XML Schemama. Nakon prolaska kontrola započinje mapiranje čiji je razvoj napravljen prema dokumentaciji za mapiranje između ove dvije sheme metapodataka koju je izradila Kongresna knjižnica u Washingtonu. Kako bi mapiranje bilo što konzistentnije u odnosu na deklaracije odabranih shema metapodataka za mapiranje, implementirane su brojne funkcionalnosti. Primjerice za strukturu imena u MODS-u se konzultiraju normativne datoteke Kongresne knjižnice, za provjeru jezika i država se konzultiraju ISO standardi, isto tako, provjeravaju se brojni mogući identifikatori kao što su ISBN ili ISSN i sl. Završetkom postupka mapiranja krajnji korisnik ima uvid u cijeli postupak; osiguran mu je prikaz koje su datoteke uspješno mapirane a koje nisu, i prema potrebi, informira ga se o pronađenim grešakama. Sve uspješno mapirane datoteke moguće je preuzeti, nakon čega postupak mapiranja može započeti ponovno.

S obzirom na činjenicu da je cjelokupni programski kôd javno dostupan i slobodan za korištenje odlukom njegova autora, njegovo mijenjanje i prilagođavanje vlastitim potrebama nije samo dozvoljeno, nego i poželjno. Rješenje je neovisno o operativnom sustavu te ga je moguće implementirati na bilo kojem poslužitelju. Daljnji razvoj je moguć u bilo kojem smjeru. Ako se želi dodati nova shema metapodataka za mapiranje, vrlo jednostavno mogu se proširiti uvjeti unutar kontrola kako bi aplikacija prihvatila još jednu shemu metapodataka. S obzirom da rješenje implementira validaciju shema metapodataka prema pripadnim XML Schema dokumentima, bilo koja ustanova za svoje potrebe može prilagoditi rješenje i koristiti samo tu funkcionalnost.

Jedna od izuzetno bitnih stvari u suvremenom informacijskom dobu je slobodan protok informacija. Ovo rješenje je u mogućnosti komunicirati s drugim programskim rješenjima, prvenstveno Kongresnom knjižnicom. Isto tako, vrlo je jednostavno nadograditi programsko rješenje da bude u stanju komunicirati s drugim sustavima odnosno programskim rješenjima. Jedna informacijska ustanova možda ima sustav za mapiranje shema metapodataka, no ne i takav koji omogućuje mapiranje između Dublin Corea i MODS-a, u vrijeme dok je pisan ovaj rad. Proširivanje programskog rješenja u smjeru da bude u stanju primiti dokument s drugog poslužitelja te ga na isti vratiti za programera je tek mali izazov. Sve u svemu, ovo programsko rješenje može imati velike posljedice ne samo na ustanovu koja će ga primijeniti, nego i na njezine korisnike, za koje će njezini djelatnici sada imati više vremena kada posao koji je moguće automatizirati bude obavljalo računalo; u tom slučaju, oni će svoje ideje i kreativnost u potpunosti moći usmjeriti prema zajednici.

Literatura

1. Bosančić, Boris; Tormaš, Tena. Istraživanje o zastupljenosti shema metapodataka u repozitorijima ustanova. // 14. seminar Arhivi, knjižnice, muzeji: mogućnosti suradnje u okruženju globalne informacijske infrastrukture / Faletar Tanacković, Sanjica ; Hasenay, Damir (ur). Zagreb: Hrvatsko knjižničarsko društvo, 2011. Str. 91-113.
2. Caplan, Priscilla. Metadata fundamentals for all librarians. Chicago: American Library Association, 2003.
3. Dizdar, S. Od podatka do metapodatka. Sarajevo: Nacionalna i univerzitetska biblioteka Bosne i Hercegovine, 2011.
4. Dublin Core Metadata Initiative. Dublin Core Metadata. URL: purl.oclc.org/metadata/dublin_core (2016-10-10)
5. Dushay, Naomi; Hillmann, Diane I. Analyzing metadata for effective use and re-use. // International conference on Dublin Core and metadata applications 0, 0(2003), str. 161-170. URL: http://dcpapers.dublincore.org/pubs/article/view/744/740 (2016-09-20).
6. Duval, Eric. Metadata standards: What, who & why. // Journal of Universal Computer Science 7, 7(2001), str. 591-601. URL: http://www.jucs.org/jucs_7_7/metadata_standards_what_who/Duval_E.pdf (2016-09-20)
7. Flanagan, David. JavaScript: The definitive Guide: Activate you web pages. O'Reilly Media, 2011. URL: https://books.google.hr/books?id=j2htBAAAQBAJ&lpg=PR1&pg=PR1#v=onepage&q=ajax&f=false (2016-09-20)
8. Foundation: the most advanced responsive front-end framework in the world. URL: http://foundation.zurb.com (2016-09-20)
9. Gilliland, Anne J. Setting the stage. // Introduction to metadata / urednica Murtha Baca. 3. izd. Los Angeles: Getty Publications, 2016. URL: http://www.getty.edu/publications/intrometadata/setting-the-stage/ (2016-10-10)
10. Glas, Božidar. Generiranje i mapiranje metapodataka u digitalnoj knjižnici hrvatske tiskane baštine Edicija. Osijek: Filozofski fakultet: 2013.
11. Google. Material design. URL: https://material.google.com/ (2016-09-20)
12. Greenberg, Jane. Understanding metadata and metadata schemes. // Cataloging & Classification Quarterly 40, 3/4(2005), str. 17-33. URL: http://www.columbia.edu/cu/libraries/inside/units/bibcontrol/osmc/greenberg.pdf (2016-09-20)

13. Guenther, Rebecca S. MODS: The metadata object description schema. // Libraries and the academy 3, 1(2003), str. 137-150. URL: http://www.loc.gov/standards/mods/3.1guenther.pdf (2016-09-20)
14. Hakala, Juha. Dublin Core metadata initiative, 2000. Str. 1-9. URL: http://cordis.europa.eu/pub/cris2000/docs/hakala1_fulltext.pdf (2016-09-20)
15. Havas, Ladislav; Lesar, Matija. Primjena SQL-a u programiranju otvorenog koda. // Tehnički glasnik 6, 2(2012), str. 164-170. URL: http://hrcak.srce.hr/file/139594 (2016-09-20)
16. IFLA. Digital libraries: Metadata resources, 2005. URL: http://www.ifla.org/node/9337 (2016-09-20)
17. Lerdorf, R.; Tatroe, K.; Macintyre, P. Programming PHP. O'Reilly Media, 2006. URL: https://books.google.hr/books?id=h-E1lVko-skC&lpg=PT5&ots=ypL-cQOrSi&dq=php%20programming&lr&hl=hr&pg=PT5#v=onepage&q&f=false (2016-09-20)
18. Library of Cogress. Metadata object description schema (MODS): Introduction and implementation, 2013. URL: https://www.loc.gov/standards/mods/userguide/introduction.html (2016-09-20)
19. Library of Congress. Dublin Core metadata element set mapping to MODS: Version 3, 2012.URL: http://www.loc.gov/standards/mods/dcsimple-mods.html (2016-09-20)
20. Library of Congress. Dublin Core Metadata element set mapping to MODS version 3, 2012. URL: http://www.loc.gov/standards/mods/dcsimple-mods.html (2016-09-20)
21. Library of Congress. Legal. URL: https://www.loc.gov/legal/ (2016-09-20)
22. Library of Congress. MODS to Dublin Core metadata element set mapping: Version 3, 2012. URL: http://www.loc.gov/standards/mods/mods-dcsimple.html (2016-09-20)
23. Library of Congress. MODS XLST 1.0 Stylesheets: DC to MODS 3.5. URL: http://www.loc.gov/standards/mods/v3/DC_MODS3-5_XSLT1-0.xsl (2016-09-20)
24. Library of Congress. Search Results. URL: http://id.loc.gov/search/?q=mulih+juraj&q=cs%3Ahttp%3A%2F%2Fid.loc.gov%2Fauthorities%2Fnames (2016-09-20)
25. Library of Congress. Top-level element: <name>, 2016. URL: https://www.loc.gov/standards/mods/userguide/name.html (2016-09-20)
26. Library of Congress. Top-level element: <originInfo>, 2013. URL: http://www.loc.gov/standards/mods/userguide/origininfo.html (2016-09-20)
27. Library of Congress. Top-level element: <physicalDescription>, 2015. URL: https://www.loc.gov/standards/mods/userguide/physicaldescription.html (2016-09-20)

28. OCLC.org. LC Name Authority File (LCNAF), 2015. URL: https://www.oclc.org/developer/develop/web-services/lc-name-authority-file-lcnaf.en.html (2016-09-20)
29. Parl, Jung-ran; Brenza, Andrew. Evalution of semi-automatic metadata generation tools: A survey of the current state of the art. // Information technology and libraries 34, 3(2015), str. 22-42. URL: https://ejournals.bc.edu/ojs/index.php/ital/article/view/5889/pdf (2016-09-20)
30. Pierre, Margaret St.; Laplant, William P. Issues in crosswalking content metadata standards. Bethesda: NISO, 1998. URL: http://www.niso.org/publications/white_papers/crosswalk/ (2016-09-20)
31. Rust, Godfrey; Bide, Mark. The <indecs> metadata framework: Principles, model and data dictionary, 2000. URL: http://www.doi.org/topics/indecs/indecs_framework_2000.pdf(2016-10-10)
32. Steffel, Nick. The simple Dublin Core generator, 2010. URL: http://www.dublincoregenerator.com/index.html (2016-10-15)
33. Šarić, Ivana; Magdić, Antonio; Essert, Mario. Sheme metapodataka značajne za knjižničarstvo s primjerom implementacije OpenURL standarda. // Vijesnik bibliotekara Hrvatske 54, 1/2(2011), str. 136-157. URL: http://www.hkdrustvo.hr/datoteke/1150/vbh/God.54(2011),br.1-2 (2016-09-20)
34. The DomDocument class. http://php.net/manual/en/class.domdocument.php (2016-09-20)
35. Undestanding metadata. Bethesda: NISO Press, 2004. Str. 1. URL: www.niso.org/publications/press/UnderstandingMetadata.pdf (2016-09-20)
36. W3C. What is the Document Object Model?. URL: https://www.w3.org/TR/WD-DOM/introduction.html (2016-09-20)
37. Weller, Nathan B. 8 reasons why pageless design is the future of the web. URL: http://www.dtelepathy.com/blog/design/8-reasons-why-pageless-design-is-the-future-of-the-web (2016-09-20)
38. Willer, Mirna. Interoperabilnostsadržaja metapodataka. // 4. seminar Arhivi, knjižnice, muzeji: mogućnosti suradnje u okruženju globalne informacijske infrastrukture / uredile Mirna Willer i Tinka Katić. Zagreb : Hrvatsko knjižničarsko društvo, 2001. Str. 57-72.
39. Witten, Ian H. … [et al.]. Importing documents and metadata into digital libraries: Requirements analysis and an extensible architecture. // Research and Advanced Technology for Digital Libraries: 6th European Conference, ECDL 2002, Rome, Italy, September 16-18, 2002. / Thanos, Costantino (Ed.). Berlin: Springer, 2002. Str. 390-405. URL: http://www.cs.waikato.ac.nz/~ihw/papers/02-IHW-etal-Importingdocuments.pdf (2016-10-

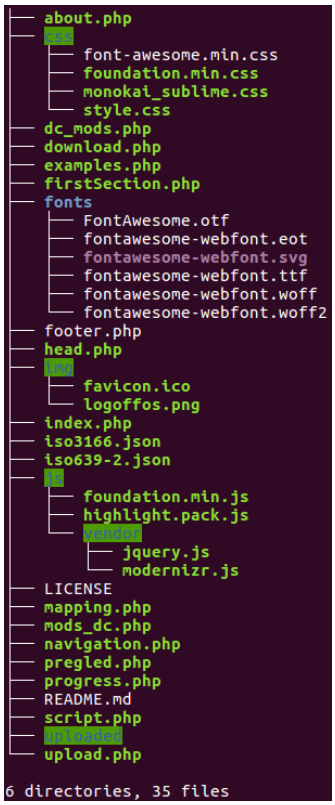
10)

40. Woodley, Mary S. Metadata matters: Connecting people and information. // Introduction to metadata / urednica Murtha Baca. 3. izd. Los Angeles: Getty Publications, 2016. URL: http://www.getty.edu/publications/intrometadata/metadata-matters/ (2016-09-20)

Prilozi

```
<meta name="description" content="Diplomski rad na temu Razvoj programskog rješenja za mapiranje shema metapodataka">
<meta name="keywords" content="diplomski rad, Dublin Core, DC, MODS, metapodaci">
<meta name="author" content="Antun Matanović">
<meta name="DC.Title" content="Razvoj programskog rješenja za mapiranje shema metapodataka">
<meta name="DC.Creator" content="Antun Matanović">
<meta name="DC.Subject" content="diplomski rad, Dublin Core, DC, MODS, metapodaci">
<meta name="DC.Description" content="Diplomski rad na temu Razvoj programskog rješenja za mapiranje shema metapodataka">
<meta name="DC.Date" content="2016">
<meta name="DC.Identifier" content="https://amatanovic.com/diplomski/index.php">
<meta name="DC.Relation" content="https://github.com/amatanovic/mapiranje-metapodataka">
<link rel="schema.DC" href="http://purl.org/DC/elements/1.1/">
```

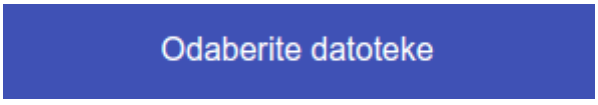
Slika 1. Metapodaci u programskom rješenju



Slika 2. Struktura datoteka uprojektu prema direktorijima



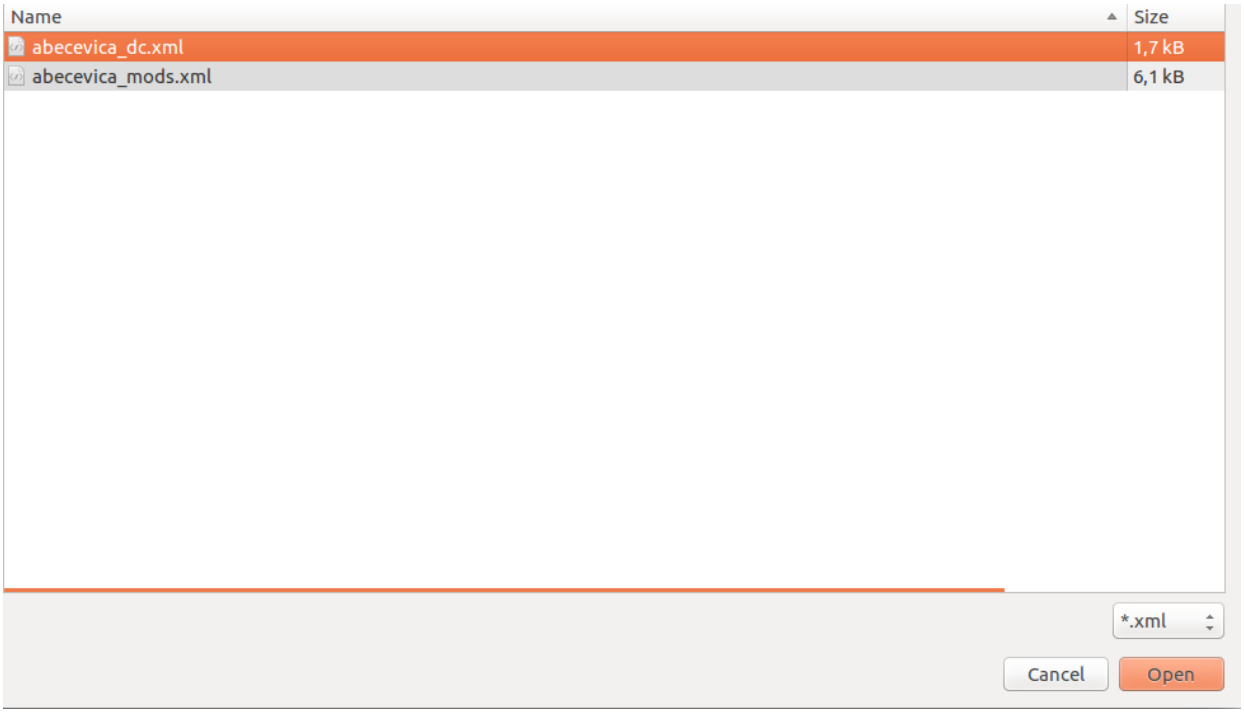
Slika 3. Fiksna navigacija na vrhu



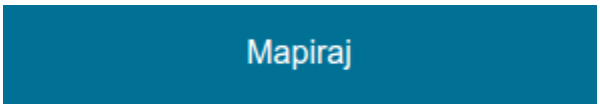
Slika 9. Gumb za prilaganje datoteka



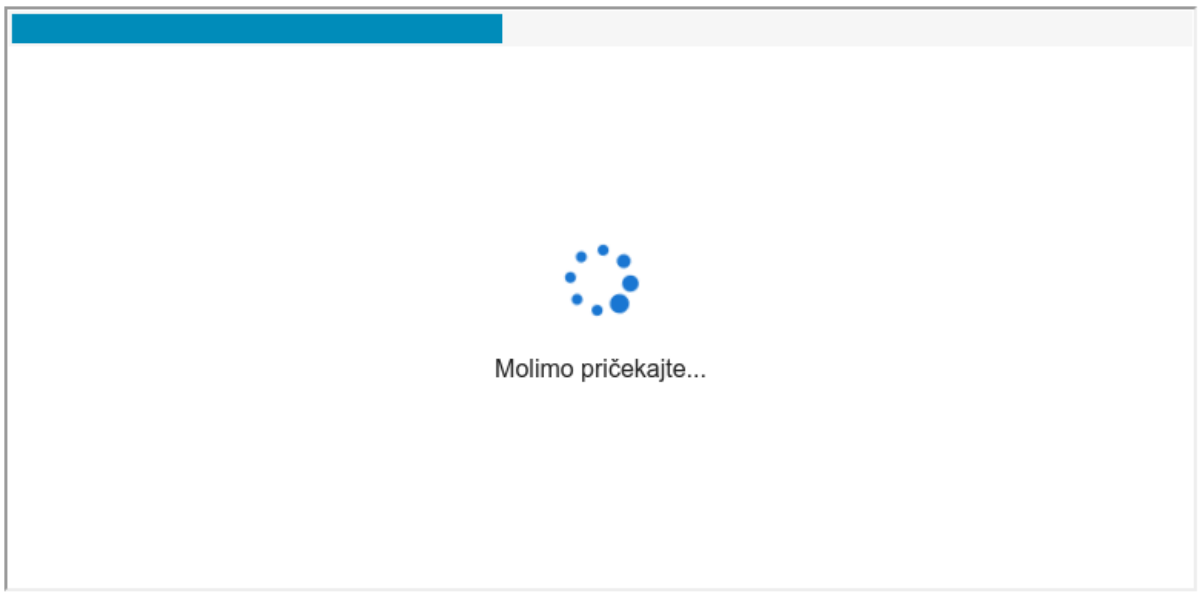
Slika 10. Standardni gumb u preglednicima



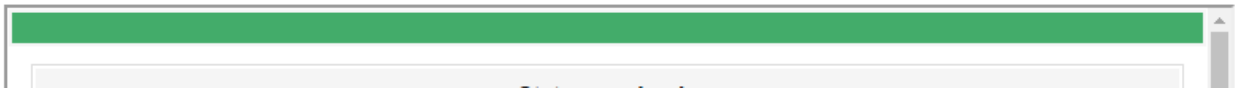
Slika 11. Dijaloški okvir za prilaganje datoteka filtrira samo XML datoteke



Slika 12. Gumb mapiraj

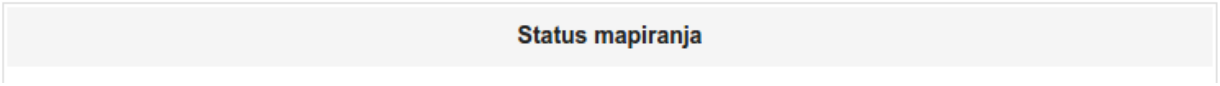


Slika 13. Iframe element u kojemu se ispisuje status mapiranja

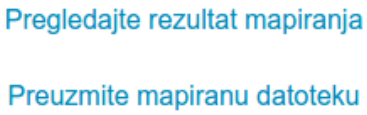

Slika 14. Uspješno završen proces mapiranja



Slika 15. Neuspješno završen proces mapiranja



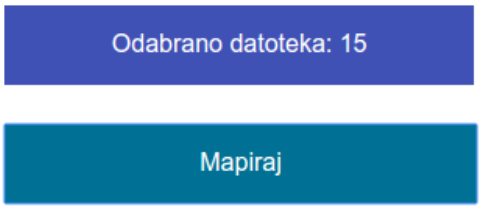
Slika 16. Zaglavlje tablice u *iframe* elementu



Slika 17. Poveznice na pregledavanje i preuzimanje jedne mapirane datoteke

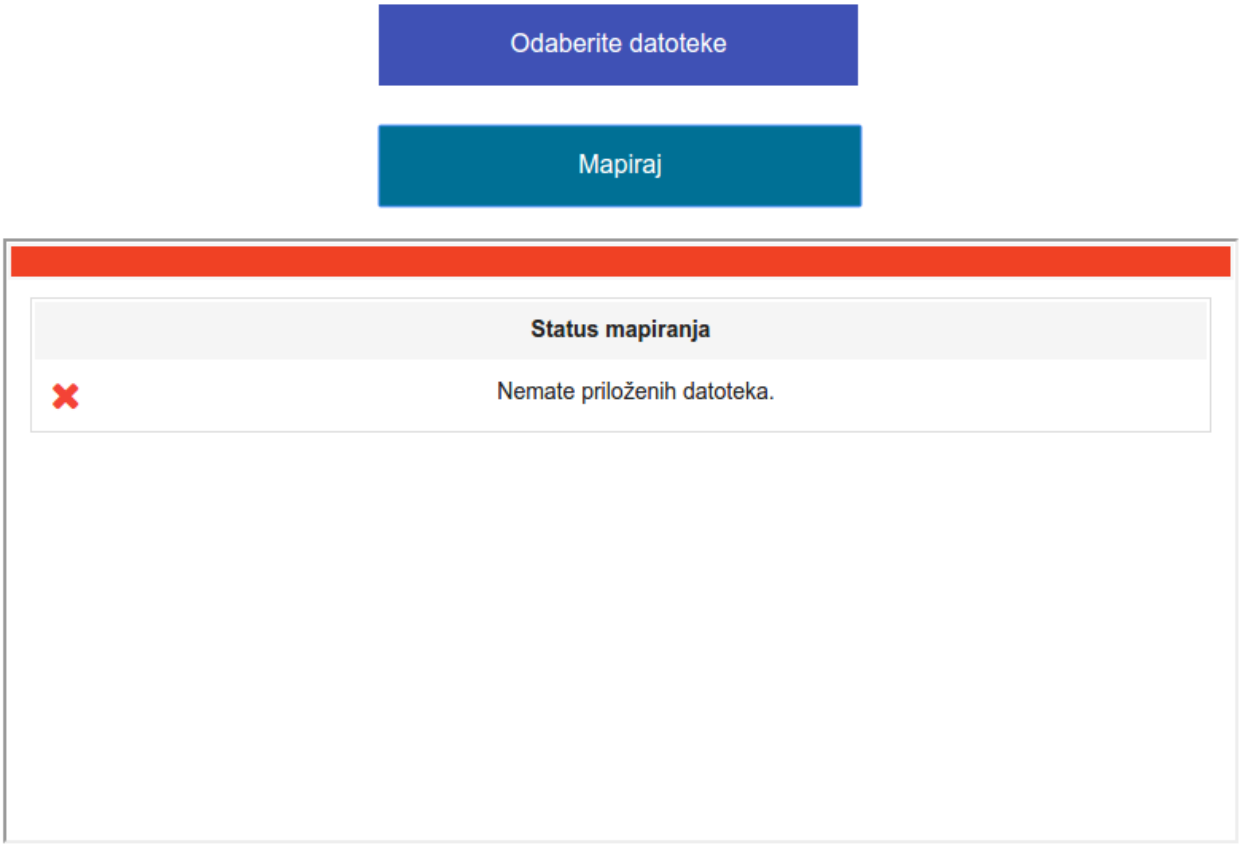


Slika 18. Poveznica za preuzimanje više mapiranih datoteka





Slika 19. Poruka pogreške o maksimalno dozvoljenom broju datoteka



Slika 20. Poruka pogreške kako niti jedna datoteka nije priložena

Index of /diplomski/uploaded

Name	Last modified	Size	Description
Parent Directory		-	
2016101017413845/	2016-10-10 17:41	-	
2016101017442918/	2016-10-10 17:44	-	
2016101019464468/	2016-10-10 19:55	-	
2016101120590219/	2016-10-11 20:59	-	
2016101121241364/	2016-10-11 21:24	-	
2016101123060480/	2016-10-11 23:06	-	
20161010171406404/	2016-10-10 17:14	-	
20161010171412746/	2016-10-10 17:14	-	
20161010172540807/	2016-10-10 17:25	-	
20161010173829641/	2016-10-10 17:38	-	
20161010173902487/	2016-10-10 17:39	-	

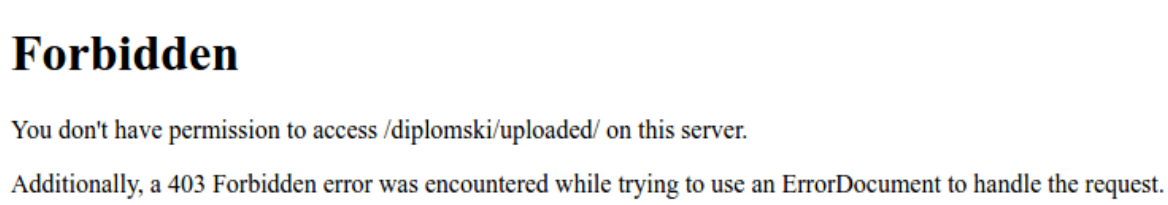
Slika 21. Sadržaj direktorija *uploaded*

Index of /diplomski/uploaded/2016101123374839 4

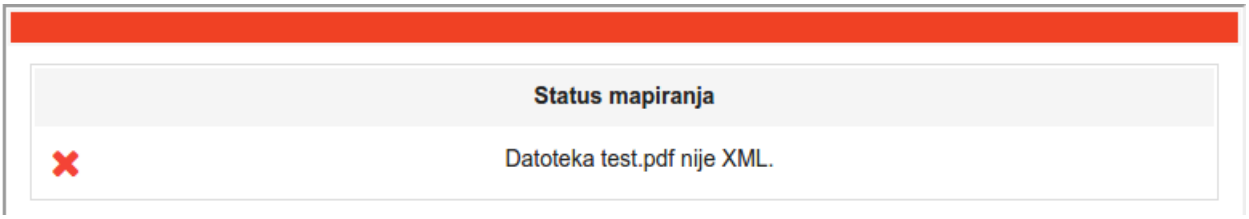
Name	Last modified	Size	Description
Parent Directory		-	
abecevica_dc.xml	2016-10-11 23:37	1.7K	
abecevica_mods.xml	2016-10-11 23:37	6.0K	
output.zip	2016-10-11 23:37	2.2K	
output/	2016-10-11 23:37	-	

Apache/2.4.18 (Ubuntu) Server at localhost Port 80

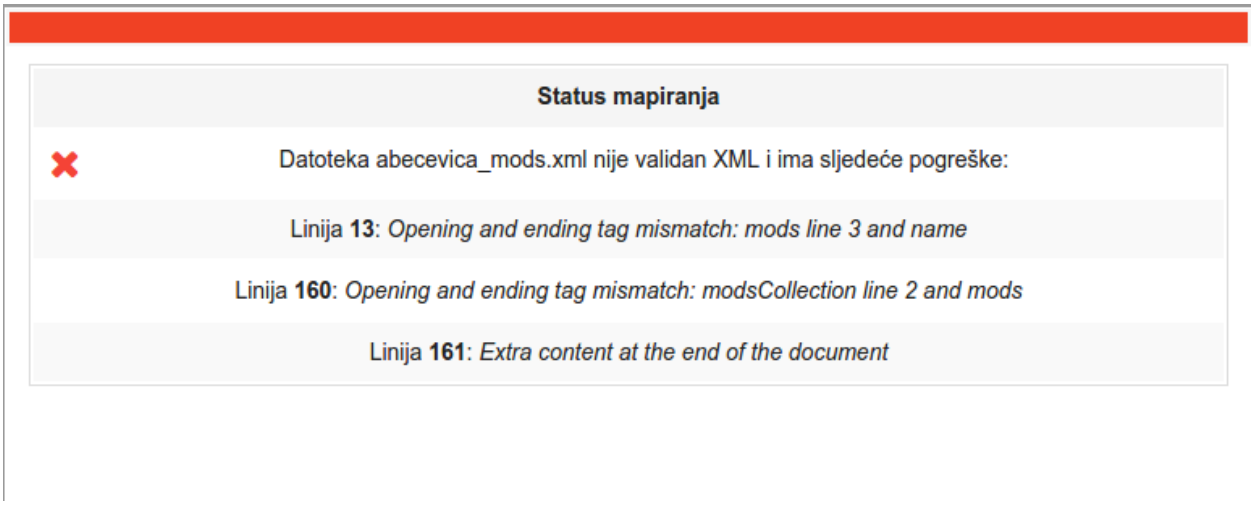
Slika 22. Sadržaj direktorija u koji se stavljaju priložene datoteke



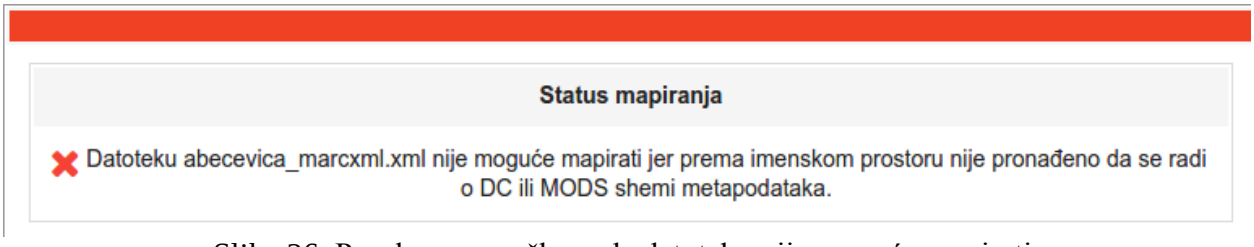
Slika 23. Zabranjeno izlistavanje sadržaja direktorija



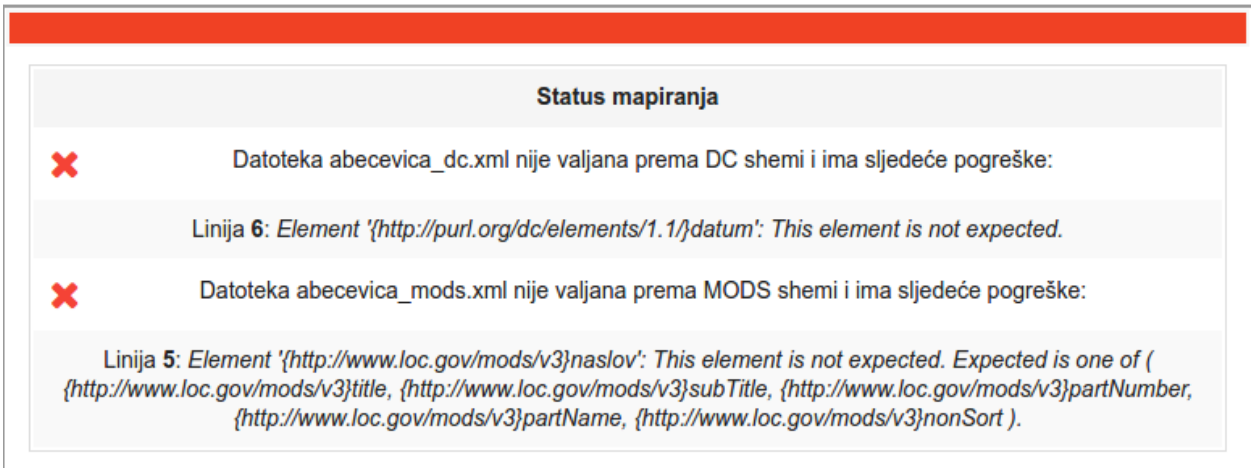
Slika 24. Poruka pogreške prilikom prilaganja datoteke koja nije XML



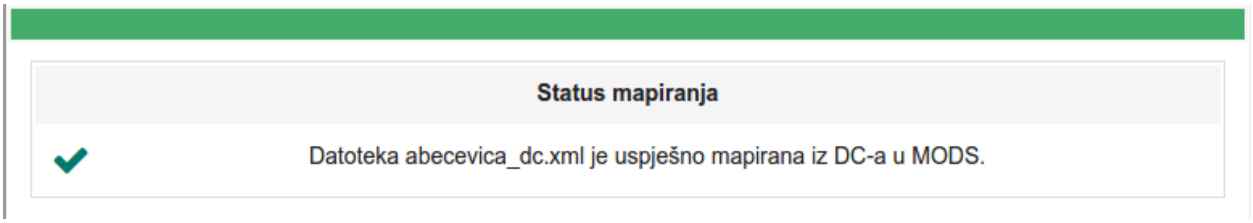
Slika 25. Poruke s pogreškama ako XML nije formalno dobro formatiran



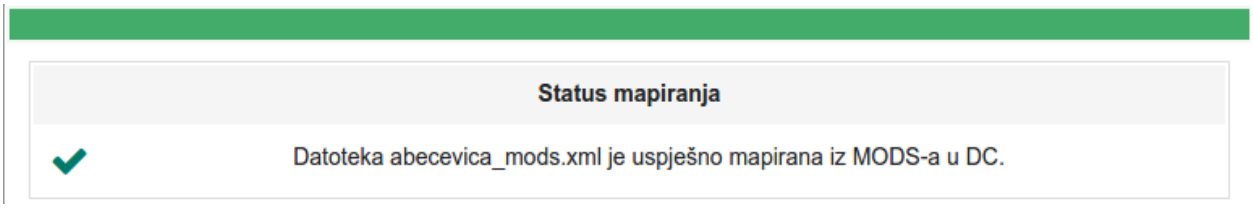
Slika 26. Poruka s pogreškom da datoteku nije moguće mapirati



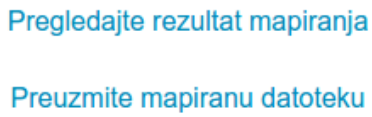
Slika 27. Poruke s pogreškama neispravnih dokumenata prema XSD shemama



Slika 28. Uspješno mapirana datoteka iz DC-a u MODS



Slika 29. Uspješno mapirana datoteka iz MODS-a u Dublin Core



Slika 30. Poveznice ako je mapirana samo jedna datoeka



Slika 31. Poveznica ako je mapirano više od jedne datoteke



Slika 32. Pregled mapirane datoteke nakon završetka procesa mapiranja