Pretvorba XML i JSON formata

Boban, Sten

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

PRETVORBA XML I JSON FORMATA

Diplomski rad

Sten Boban

Osijek, 2020.

SADRŽAJ

1. UVOD

Potreba za spremanjem i prenošenjem podataka, u današnje vrijeme sve je veća. Svakog dana se stvaraju neke nove aplikacije, pružaju nove usluge korisnicima (poput *cloud-a*), nastaju novi softveri; tehnologija napreduje. S obzirom na mnoštvo različitih tehnoloških riješenja, nije niti čudno što je moguće susretanje korisnika s različitim oblicima spremanja podataka. Tako razlikujemo dva oblika prikaza podataka, XML (*Extensible Markup Language*) i JSON (*JavaScript Object Notation*) oblik. Kako je već ranije navedeno, stvaranjem novih sustava i nastankom novih usluga, stvara se ogromna potreba za međusobnu konverziju podataka iz jednog oblika u drugi (XML u JSON i JSON u XML). U ovom radu ćete se moći pobliže upoznati s osnovama XML-a i JSON-a te će Vam biti objašnjeno i prezentirano na koji način je moguće ostvariti konverziju između ova dva formata i zašto je ona toliko bitna.

1.1. Opis projektnog zadatka

Potrebno je napraviti aplikaciju koja će pretvoriti JSON zapis podataka u odgovarajuće elemente i podatke u XML formatu. U ovom smjeru ne koristiti uopće XML atribute. Također napraviti pretvorbu u obrnutom smjeru gdje će se i elementi i atributi pretvoriti u odgovarajući JSON format.

2. PODACI

Svima je poznato koliko podaci znaju biti osjetljivi i koliko je bitno pravilno rukovanje istima, pa je tako potrebno imati određenu ideju o tome na koji način, ovisno o potrebi, ćete spremiti svoje podatke. Zamislite da na raspolaganju imate ogromnu količinu podataka, na primjer popis proizvoda koji se nalaze unutar Vašeg skladišta, prvo što biste željeli napraviti je na neki način strukturirati te podatake, zar ne? Upravo tu na snagu stupaju XML i JSON, koji će Vam pružiti mogućnost strukturiranog spremanja podataka i samim time laganog pristupa pojedinim dijelovima spremljenih elemenata. Za obradu podataka postoje različiti programski jezici (poput c, c++, c#, java...), ali kako poslati podatke s jednog mjesta na drugo? Recimo, kupac poželi sa svojeg računala izlistati sve proizvode koji se nalaze u Vašem skladištu, a da imaju cijenu manju od 500kn. To bi bilo vrlo teško ostvariti kod nestrukturiranih podataka, gotovo nemoguće. Dakle, možemo reći da nam XML i JSON omogućuju spremanje podataka u obliku koji je čovjeku prihvatljiv, odnosno lako shvatljiv i razumljiv te ih koristimo za transport između servera i klijenta (i obrnuto).

3. XML (Extensible Markup Language)

Kao što i samo ime kaže, XML je jezik dizajniran za pohranu, prijenos i razmjenu podataka. XML je sličan HTML-u (*HyperText Markup Language*), što čini poznavanje HTML-a korisno za njegovo shvaćanje i bolje razumijevanje. Za razliku od HTML-a koji se fokusira na manipulaciju tekstualnim atributima (izmjena fonta, boja, stvaranje različitih stranica...), odnosno prikazuje podatke, dok je XML usmjeren prema podacima (nije stvoren za prikaz podataka). Također još jedna bitna razlika je u tome što XML pruža mogućnost stvaranja vlastitih oznaka unutar dokumenta, za opis podataka (HTML koristi predefinirane oznake). Da bismo dokument smatrali valjanim XML dokumentom, on mora zadovoljavati određena pravila, pa tako ispravan dokument mora posjedovati isključivo jedan „*root*“ element (korijenski element) unutar kojega će se nalaziti svi ostali elementi. Svaki element koji je deklariran unutar samog dokumenta, mora biti zatvoren, odnosno mora se jasno dati do znanja u kojem trenutku prestaje njegovo djelovanje. [1]

Recimo da nekome želimo prezentirati koje smo sve životinje danas vidjeli u zoološkom vrtu. To bi izgledalo kao na Sl. 3.1.

```
There is a dog with name Jake.
There is an elephant with name Eli.
There is a cat with name Fluffy.
There is a hamster with name Ham.
```

Sl. 3.1. *Obični prikaz podataka.*

S obzirom da se radi o podacima, morali bismo moći podatke sa Sl. 3.1. prikazati u XML formatu. Valjana transformacija predstavljenih podataka, u XML format, je vidljiva na Sl. 3.2.

```
<?xml version="1.0" ?>
<zoo>
    <dog>Jake</dog>
    <elephant>Eli</elephant>
    <cat>Fluffy</cat>
    <hamster>Ham</hamster>
</zoo>
```

Sl. 3.2. *XML prikaz podataka.*

Možete uočiti kako se poštuju pravila XML-a, definirana je verzija koja se koristi te je svaka deklaracija pojedinog elementa zatvorena (završena) – na primjer: „</dog>“ predstavlja završetak

elementa „<dog>“. U prezentiranom primjeru elementi bi bili: *zoo, dog, elephant, cat, hamster*; vrijednosti elemenata su: Jake, Eli, Fluffy, Ham; a „*root*“ element dokumenta je „*zoo*“. To znači da su elementi deklarirani na sljedeći način:

<element_name>value_of_element</element_name>

Ukoliko umjesto Sl. 3.1. svojem prijatelju pošaljete Sl. 3.2. on će lako moći zaključiti kako se u zoološkom nalaze pas, slon, mačka i hrčak, te će isto tako znati koja su njihova imena. Dakle može se zaključiti kako u transformaciji izjavnih rečenica sa Sl. 3.1. u XML oblik (Sl. 3.2.) nema gubitka informacija.

Važno je napomenuti kako XML poznaje razliku između velikih i malih slova (engl. *case sensitive*), zbog čega bi dokument na Sl. 3.3. bio nevažeći.

```
<?xml version="1.0" ?>
<zoo>
    <Dog>Jake</DOG>
    <elephant>Eli</Elephant>
    <cat>Fluffy</cat>
    <hamster>Ham</hamster>
</zoo>
```

Sl. 3.3. *Primjer nepoštivanja pravila (case sensitive).*

Uočite (Sl. 3.3.) da „*Dog*“ nije isto što i „*DOG*“, prema tome, ne poštuje se pravilo koje kaže da je potrebno svaki deklarirani element zatvoriti (deklriran je „*Dog*“, a zatvoren je „*DOG*“ element). Isto je slučaj i sa „*elephant*“-„*Elephant*“ elementima, oni nisu isti!

```
<?xml version="1.0" ?>
<root> <!--ROOT NODE and parent to dog, elephant, cat, hamster-->
    <dog>Jake</dog> <!--child of root-->
    <elephant>Eli</elephant> <!--sibling of dog, cat and hamster-->
    <cat>Fluffy</cat>
    <hamster>Ham</hamster>
</root>
```

Sl. 3.4. *Međusobne relacije elemenata.*

Na Sl. 3.4. je objašnjen međusobni odnos elemenata, dakle „*root*“ je *parent* (roditelj) elementima: *dog, elephant, cat, hamster* koji su *child* elementi (njegova djeca). Također u prikazanom primjeru

sva djeca „*root*“ elementa (*dog*, *elephant*, *cat*, *hamster*) su međusobno braće i sestre. Za komentiranje XML dokumenta koriste se „<!---->“ oznake (<!--ovo je komentar-->). Djeca roditelja također mogu imati svoju djecu, pa se tako stvara ugnježđeno stablo elemenata. XML dokumenti formiraju podatke u strukturi stabla (djed/baka, roditelji, djeca...). Pogledate li Sl. 3.1.1. možete uočiti kako se stablo grana, sada je „*relatives*“ element *parent* (roditelj elementima „*mother*“ i „*father*“) i *grandparent* (djed/baka elementima „*first_name*“, „*last_name*“, „*age*“).

3.1. Deklaracija elemenata

Imena elemenata mogu sadržavati slova, brojeve i ostale znakove, ali nikako ne smiju počimati s brojem ili nekim od ostalih znakova, već isključivo sa slovima (osim kombinacije slova xml, XML, Xml i slično). U imenu elementa se ne smije koristiti razmak, niti oznake koje su rezervirane (poput „:“, „-“). Poželjno bi bilo koristiti kratke i jasne nazive elemenata.

```
<?xml version="1.0" ?>

<relatives><!--parent to "mother" and grandparent to "first_name"-->
    <mother>
        <first_name>Margot</first_name> <!--grandchild of "relatives"-->
        <last_name>Marckson</last_name>
        <age>23</age>
    </mother>

    <father>
        <first_name>Jake</first_name>
        <last_name>Jackson</last_name>
        <age>28</age>
    </father>
</relatives>
```

Sl. 3.1.1. *Primjer pojavljivanja većeg broja istih elemenata s različitim roditeljima (parent).*

U slučaju koji se pojavljuje na Sl. 3.1.1. kad bismo zatražili računalo da nam prikaže vrijednost elementa „*first_name*“ kao rezultat bi dobili vrijednosti Margot i Jake, odnosno pronašao bi sve elemente tog imena i vratio njihove vrijednosti korisniku.

Elementi mogu biti zapisani i na drugačiji način, sadržavajući atribute (vidi Sl. 3.1.2.). Atribute uvijek moramo navesti unutar navodnika (možemo koristiti dvostruke „“ ili jednostruke ' ' navodnike). Primijetite da je kod prve osobe („*person1*“) „*gender*“ atribut, a kod druge osobe („*person2*“) je element, oba načina osiguravaju jednaku informaciju. Atribute možete koristiti za

specificiranje jednog svojstva nekog elementa. Atributima nije lako pristupiti, moguće je, ali nije toliko jednostavno, zato je dobro ne koristiti ih ukoliko to stvarno nije potrebno.[2]

```
<?xml version="1.0" ?>

<relatives>
    <person1 gender="female"> <!--using attribute-->
        <first_name>Margot</first_name>
        <last_name>Marckson</last_name>
        <age>23</age>
    </person1>

    <person2>
        <gender>male</gender>
        <first_name>Jake</first_name>
        <last_name>Jackson</last_name>
        <age>28</age>
    </person2>
</relatives>
```

Sl. 3.1.2. *Deklaracija atributa.*

Proučite sljedeći element:

<person1 gender="female">

person1 – predstavlja ime elementa

gender – predstavlja atribut

„female" – predstavlja vrijednost atributa

Prilikom zatvaranja elementa koji je dodatno opisan atributom, nije potrebno koristiti nikakve dodatne oznake za njegovo zatvaranje, prema tome iz prethodnog primjera, ispravno zatvoreni element bi izgledao ovako:

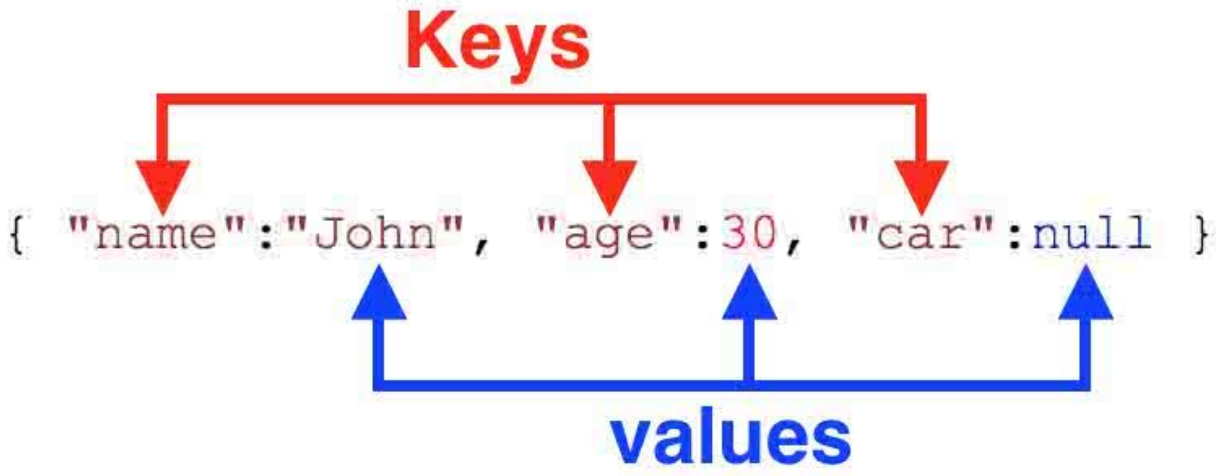
</person1>

Pojedini element može imati isključivo jednu vrijednost atributa. [3]

4. JSON (JavaScript Object Notation)

JSON koristimo za spremanje podataka na organizirani i jednostavno čitljiv način (samim time je jednostavan i pristup podacima). JSON se smatra predefiniranim formatom za prijenos podataka kod modernih aplikacija. Primarno se koristi za prijenos podataka između web aplikacija i srevera. Uglavnom se koristi za spremanje i reprezentaciju kompleksnih podataka. JSON dokumenti su bazirani na tekstu, čovjeku su lako čitljivi (lakše nego XML dokumenti) i malih su veličina. [4]

Podaci su predstavljeni u obliku ključ/vrijednost parova, gdje ključevi predstavljaju *string*-ove, a vrijednost predstavlja JSON oblike podataka. Uređeni parovi su međusobno odvojeni zarezima (,), dok su sam ključ i vrijednost međusobno odvojeni dvotočkom (:), kao što je prikazano na Sl. 4.1. [5]



Sl. 4.1. *Definiranje JSON elemenata.*[1]

U predstavljenom primjeru na Sl.4.1. je već na prvi pogled lako uočljivo o čemu se radi i jasno je što prikazani podaci predstavljaju. Uređeni par, odnosno ključ/vrijednost element bi bio “name“:“John“. Svi podaci se nalaze unutar dvostrukih navodnika („“).

[1] Preuzeto s https://www.shapediver.com/blog/json-objects-explained/

5. XML vs. JSON

Većina računala rukuje s podacima koji nemaju kompatibilne formate. Prema tome, razmjena podataka između takvih nekompatibilnih sustava je vremenski vrlo zahtjevan proces s kojim se susreću svi web developeri. Velike količine podataka je potrebno konvertirati, pri čemu dolazi do gubitka podataka koji nisu komaptibilni.

XML pohranjuje podatke u tekstualnom formatu čime omogućuje softverski i hardverski neovisan način spremanja i prijenosa podataka. XML također omogućuje lakše proširivanje ili nadogradnju za novije operacijske sustave, aplikacije ili web pretraživače bez gubitka podataka. XML je sastavni dio svake web aplikacije, pa je tako korišten za konfiguraciju datoteka, definiranje shema ili mapiranje dokumenata. Svojom formom i jednostavnom strukturom je značajno olakšao prijenos podataka . [6]

JSON sve podatke pohranjuje u mapiranom formatu (ključ/vrijedenost parovi) što ga čini lakšim i jednostavnijim za shvatiti. Format podataka kojim se polako sve češće zamjenjuje XML, jer pruža mogućnost laganog oblikovanja podataka ili direktnog mapiranja u objekte, te mu je struktura puno jednostavnija za razumjeti (odnosno, lakše je uočiti značenje podataka). [7]

Za JSON se može reći kako je jednostavan format kojim se služimo za predstavljanje podataka, dok XML uz to pruža još i mogućnost označavanja unutar dokumenta. U XML-u je moguće postaviti pitanje i dobiti odgovor, odnosno korisnik može zahtijevati od dokumenta da mu odgovori na neko pitanje i XML će mu biti u mogućnosti vratiti odgovor. Za ovakav oblik komunikacije, gdje korisnik može zahtijevati određene informacije o elementima, XML jezik koristi XPath (XML Path Language). Xpath predstavlja jezik koji ima mogućnost pristupa i odabiranja (korištenja) elemenata, odnosno čvorova iz nekog XML dokumenta. Dakle može se reći kako XML još uvijek posjeduje neke posebne značajke koje ga čine interesantnim i efikasnijim od JSON-a.

JSON može predstavljati podatke u obliku stringova, brojeva, nizova, objekata te boolean i null oznaka, dok XML svoje podatke predstavlja kao čvorove i elemente koje je, prije samog korištenja unutar aplikacije, potrebno parsirati u stringove, brojeve, nizove, boolean ili null vrijednost.

Ukoliko radite na nekom projektu koji zahtjeva stvaranje dokumenata s mogućnošću označavanja podataka i stvaranja opisanih podataka s njihovim vrijednostima, odabrali biste XML. No, ukoliko se susrećete sa stvaranjem dokumenta u kojemu je bitno održati organiziranu strukturu podataka vjerojatno biste trebali odabrati JSON.

JSON se smatra odličnim, jer je potrebno manje kodiranja od strane korisnika, i zauzima malo prostora, čime se ostvaruje veća brzina prilikom prijenosa podataka. Ali to je uglavnom sve za što ga je dobro koristiti, zašto je onda toliko popularan? Podaci se spremaju u tekstualnom formatu i lagano ih je parsirati (vidi poglavlje 6. Parser) što je vrlo korisno kod web aplikacija i usluga, jer se na taj način ostvaruje brža razmjena rezultata web usluga. Samim time što su dokumenti manjih veličina od recimo XML dokumenata, lakše ih je i parsirati.

Postoje situacije u kojima će se pojaviti potreba za obradu i XML-a i JSON-a unutar iste aplikacije. Na primjer, ukoliko određena aplikacija prima XML podatke, koje potom mora proslijediti nekom sustavu koji očekuje JSON podatke. Ovo je čest slučaj kod integracije osnovih i starih softverskih sustava. Za riješavanje ovog problema potrebno je izvršiti konverziju XML-a u JSON ili obrnuto (vidi poglavlje 7. Ostvarivanje konverzije).

Razlike u strukturama:

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

Sl. 5.1. *JSON primjer.*[2]

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

Sl. 5.2. *XML primjer.*[3]

[2] Preuzeto s https://www.w3schools.com/js/js_json_xml.asp
[3] Preuzeto s https://www.w3schools.com/js/js_json_xml.asp

Doista je JSON reprezentacija učinkovitija u dočaravanju značenja podataka u odnosu na XML (Sl. 5.1. i Sl. 5.2.). Čovjeku je lakše pročitati i razumijeti podatke na Sl. 5.1. nego što je to slučaj na Sl. 5.2., ali to računalu ne igra veliku ulogu, pa samim time mu niti ne predstavlja neki problem.

XML prednosti:

- Prijenos podataka između međusobno različitih platformi (aplikacija i sustava) je brz i jednostavan
- Odvaja podatke od HTML-a
- Mogućnost korištenja XPath-a

XML nedostatci:

- Zahtijeva aplikaciju za procesuiranje
- Sintaksa može biti zbunjujuća, jer može postati vrlo kompleksna

JSON prednosti:

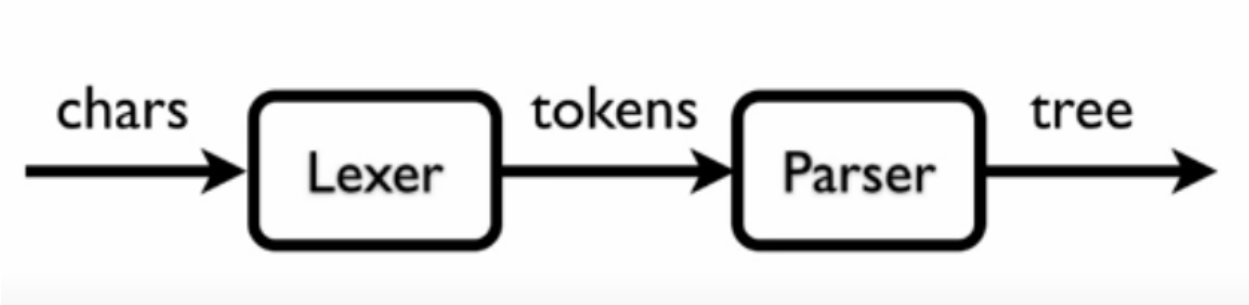
- Podržan u svim pretraživačima
- Jednostavan za razumjeti, lako shvatljiv
- Jednostavna sintaksa
- Može se parsirati u JavaScript
- Stvaranje i manipulacija podacima je jednostavna
- Većina *backend* tehnologija podržava JSON

JSON nedostatci:

- Loša proširivost
- Ogranični alati za razvoj

6. PARSER

Zamislite da imate jako veliki dokument nestrukturiranih podataka te da ih kao takve, nestrukturirane, morate koristiti za nešto. Već u startu ćete se susresti s problemima, jer je gotovo nemoguće korištenje podataka u takvom formatu. Za transformaciju ovakvih nestrukturiranih dokumenata u njihov strukturirani ekvivalent koriste se parseri. Parsiranjem podataka stvara se njihov strukturirani oblik koji korisniku omogućuje lagano korištenje podataka. Parsirani podaci se pretvaraju u oblik stabla, čime se korisniku pružaju razne mogućnosti kao što su jednostavan pristup podacima, iteracija kroz sve podatke, međusobni odnos podataka (roditelj/dijete) i slično.



Sl. 6.1. *Parsiranje podataka.* [4]

Na Sl. 6.1. je prikazan proces parsiranja. Lexer prima podatke u char obliku te ih pretvara u niz tokena koje parser potom pretvara u stablo podataka. Parsiranje se može definirati kao rastavljanje određenog inputa (ulaznog niza podataka) na dijelove koji će imati neko značenje (Vidi Sl. 6.2.). Za stvaranje takvih dijelova koriste se tokeni kojima se opisuju pojedini elementi ulaznog inputa čijim se daljnjim parsiranjem dobivaju samo ključni elementi predanog inputa.

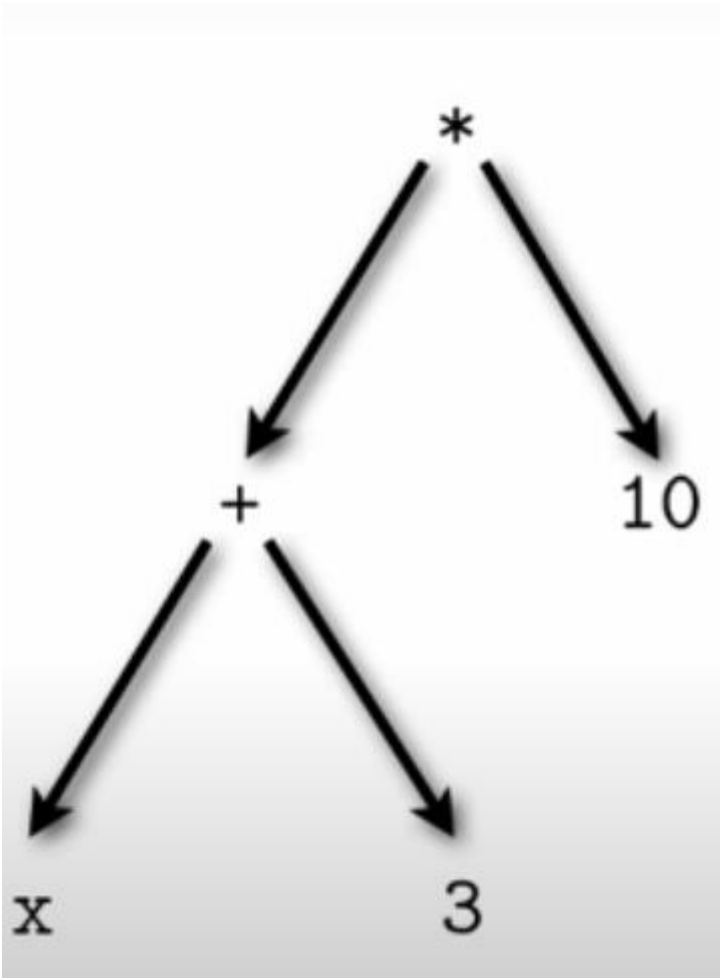
Neka je ulazni string, koji korisnik predaje parseru, sljedećeg oblika:

(x + 3) * 10

Cijeli string prolaskom kroz lexer se rastavlja na dijelove koje nazivamo tokenima. U proučavanom primjeru stvaraju se tokeni koji sadrže slijedeće elemente: (, x, +, 3, *, 10,). Dakle nakon razlaganja ulaznog inputa u manje dijelove dobili ste 7 tokena koji se potom predaju u parser. Parser od predanih podataka stvara strukturu stabla kao na Sl. 6.2. U stvorenom stablu

[4] Preuzeto s https://www.youtube.com/watch?v=V6LiAtG_0QI

možete vidjeti kako su zagrade zanemarene, jer je poredak obavljanja računskih operacija ugrađen u ugnježđenoj strukturi stabla.



Sl. 6.2. *Stvaranje stabla parsiranjem.*[5]

[5] Preuzeto s https://www.youtube.com/watch?v=V6LiAtG_0QI

7. OSTVARIVANJE KONVERZIJE

Konverter je oblik aplikacije ili usluga koja pretvara jednu formu podataka u neku drugu formu. Na primjer, možete izvršiti konverziju određenog videa u zvučni zapis. U navedenom primjeru izgubili biste sve osim zvučnog zapisa, čime bi se smanjila veličina datoteke i izvukli samo potrebni podaci. Jednako tako je moguće izvršiti konverziju između XML-a u JSON i obrnuto. Konverziji podataka je moguće pristupiti na dva različita načina:

1. Tekstualni oblik se pretvara u tekstualni oblik
2. Tekstualni oblik se pretvara u objekt, potom u tekstualni oblik

U prvom slučaju je moguće izvršiti direktnu konverziju XML podataka u njihovu odgovarajuću JSON verziju. Takvim pristupom nije potrebno stvarati objekte kojima bi se mapirali podaci. Dok je u drugom slučaju potrebno prvo rekonstruirati podatke u objekte te ih potom pretvoriti u JSON.

Problemi koji se pojavljuju prilikom konverzije podataka nastaju u slučajevima gdje pojedini podaci, koje je potrebno pretvoriti, nisu podržani u jednom od oblika u koji se pretvaraju. Ako se sjetite primjera konverzije videa u audio zapis, možete uočiti ovaj problem. Prilikom pretvaranja video zapisa u audio zapis izgubili smo sliku, odnosno sam video, a dobili smo samo podatke u obliku zvuka. No, kad bismo htjeli iz tog dobivenog zvukovnog zapisa ponovno dobiti video koji smo imali na početku konverzije, to ne bi bilo moguće. Zbog ovakvih situacija je vrlo teško, odnosno gotovo nemoguće ostvariti savršenu konverziju, pogotovo u kompleksnim strukturama podataka.

Pretvorba se mora temeljiti na nekim pravilima čijom primjenom težimo dobivanju što točnije reprezentacije predanih podataka u novonastalom obliku. Jednako tako će se za pretvorbu XML-a u JSON koristiti pravila prikazana na Sl. 7.1. XML elementi iz tablice imaju svoje ekvivalente u JSON obliku, prema čemu je potrebno stvoriti alat koji će omogućiti ovu konverziju.

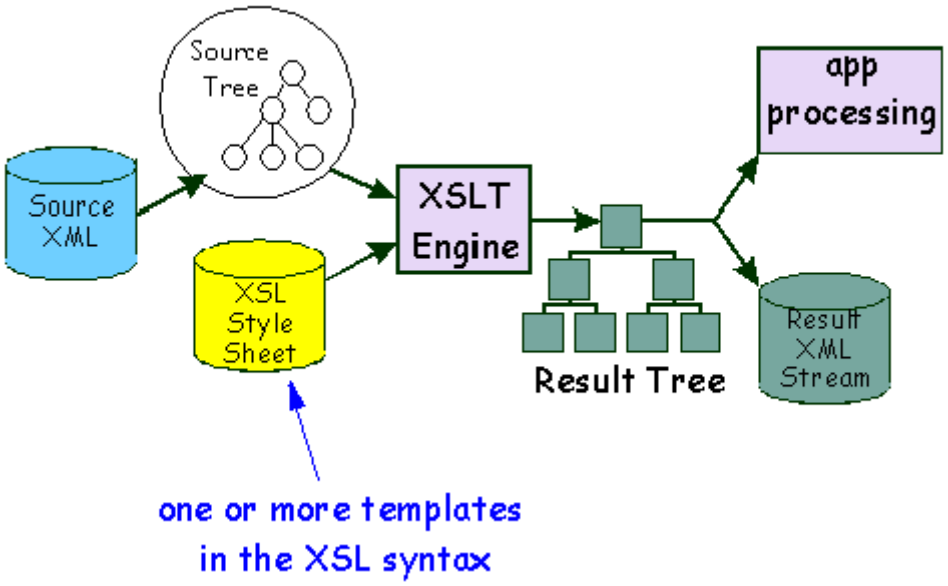
Pattern	XML	JSON
1	`<e/>`	`"e": null`
2	`<e>text</e>`	`"e": "text"`
3	`<e name="value" />`	`"e":{"@name": "value"}`
4	`<e name="value">text</e>`	`"e": { "@name": "value", "#text": "text" }`
5	`<e> <a>text</a> <b>text</b> </e>`	`"e": { "a": "text", "b": "text" }`
6	`<e> <a>text</a> <a>text</a> </e>`	`"e": { "a": ["text", "text"] }`

Sl. 7.1. *Pravila za pretvorbu XML-a u JSON i obrnuto.*[6]

[6] Preuzeto s https://www.xml.com/pub/a/2006/05/31/converting-between-xml-and-json.html

8. XSLT (EXtensible Stylesheet Language Transformation)

XSLT predstavlja jezik koji je temeljen na korištenju XML-a i koji pruža mogućnost transformacije XML dokumenta (vidi Sl. 8.1..) u neki drugi format (HTML, HML, JSON i slično). Za jedostavnu navigaciju kroz cijeli XML dokument te pronalazak i odabir čvorova koristi se XPath. Čvorovi predstavljaju dijelove dokumenta kojima se može pristupiti kao što su: atributi, elementi, tekst ili bilo koji drugi objekti unutar XML-a. Prilikom korištenja XPath-a korisno je XML dokument gledati kao stablo podataka, jer je na taj način jednostavnije prolaziti kroz elemente i dohvaćati ono što je korisniku potrebno. XSLT također pruža mogućnost iteracije i postavljanja uvjeta čime se prolazi kroz stablo podataka.



Sl. 8.1. *Arhitektura XSLT-a.*[7]

Arhitektura XSLT-a (Sl. 8.1.) se temelji na korištenju XML-a i XSL-a (*Extensible Style Sheet*). XML podatke iz predanog dokumenta prezentira u obliku stabla te na njih možete djelovati korištenjem XSL-a. Nakon što postavite pravila i formu (na Sl. 8.1. „*Result Tree*“) koju želite primijeniti nad ulaznim podacima, XSLT stvara rezultat koji je također u obliku stabla . Proces transformacije podataka iz XML-a u stablo podataka se odvija uz pomoć XPath-a kojim se definiraju dijelovi ulaznog dokumenta koji se moraju podudarati s barem jednim definiranim predloškom (*template*) kako bi se transformacija uspješno izvršila. U trenutku kada se neki dio

[7] Preuzeto s https://xml.apache.org/xalan-j/design/design2_0_0.html

ulaznih podataka podudara s postojećim predloškom tada se isti transformira na osnovu pravila definiranih u samom predlošku i zapisuje u krajnji rezultat (*Result Tree*). Dobiveni rezultat se dalje može koristiti za prikaz podataka ili obradu aplikacija.

XSLT pruža mogućnost korištenja *namespace*-a (klasa elemenata) čijim korištenjem možemo razlikovati elemente koji su naizgled jednaki (Vidi sl. 8.2.). Zašto je bitno imati mogućnost razlikovanja elemenata? Jednostavno zato što se može dogoditi da postoji više dokumenata s istim oznakama, pa bi se prilikom njihovog dvostrukog pojavljivanja dogodila greška. Spojimo li na primjer, dokumente koji sadrže informacije o proizvodima dvije firme, možemo imati dvije jednake oznake „cijena“, što stvara problem, jer oznake nisu jedinstvene. Zbog toga se umjesto stvaranja različitih oznaka (poput „cijena1“, „cijena2“) koriste namespace-ovi kojima se omogućuje stvaranje oznaka s istim nazivom („cijena“, „cijena“), ali s drugačijom identifikacijom. Korištenjem *namespace*-a stvara se jedinstvena identifikacija za svaku oznaku, čime se otklanja mogućnost pojavljivanja grešaka (vidi Sl. 8.2.). *Namespace* se obično definira na samom početku dokumenta.

```
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:json="http://www.ibm.com/xmlns/prod/2009/jsonx"
```

Sl. 8.2. *Definiranje namespace-a*

Kako bi se ispravno deklarirao *namespace*, potrebno je navesti prefiks „xmlns:“ oznaci koju želimo jedinstveno implementirati u dokument. Zatim se navodi ime oznake i daje joj se jedinstveni niz znakova kojim će se razlikovati od identičnih oznaka. Za vrijednost oznake može se koristiti URL (*Uniform Resource Locator*), URI (*Uniform Resource Identifier*) ili URN (*Universal Resource Name*). Ako na primjer koristite URL, predani link ne mora postojati, bitno je samo da je jedinstven i da se po njemu vaša oznaka može razlikovati od bilo koje druge iste takve oznake.

Prilikom stvaranja XSLT dokumenta potrebno je definirati *root* oznaku naziva „*stylesheet*“, koja je oblika kao na Sl. 8.3. Deklariranjem *root* oznake dobivate, kao korisnik, mogućnost korištenja svih svojstava, atributa, značajki vezanih za XSL (EXtensible Stylesheet Language). Ukoliko želite iskoristiti nešto od mogućnosti koje Vam pruža XSL, morate koristiti „xsl:“ prefiks prije svake oznake (<xsl:appy-templates/>).

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Sl. 8.3. *Definiranje XSLT dokumenta*

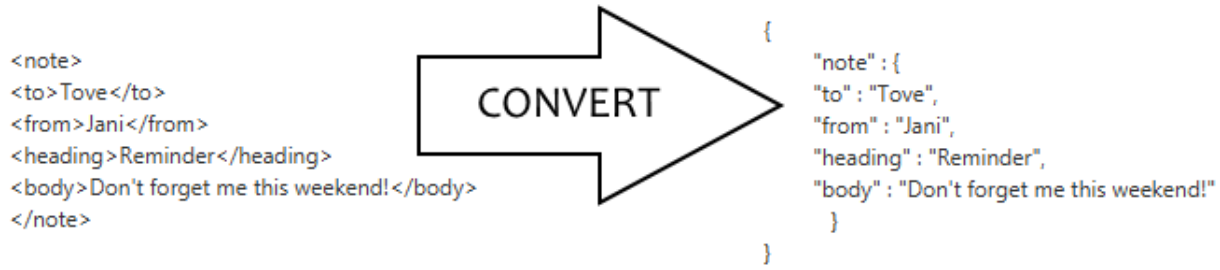
Stvaranjem *template*-a (predložaka) možete postaviti određena pravila koja želite primijeniti na pojedinim čvorovima, odnosno ukoliko pronađete željeni čvor. Pa tako možete definirati predložak kojim želite ispisati text „name_of_element found!“, onda kada pronađete čvor „name_of_element“ kao na Sl. 8.4.

```
<xsl:template match="name_of_element">
    <xsl:text>name_of_element found!</xsl:text>
</xsl:template>
```

Sl. 8.4. *Deklaracija predloška*

9. PRETVORBA XML-A U JSON

Za ovu pretvorbu su korištena već ranije spomenuta pravila (vidi Sl. 7.1.) prema kojima će se izvršavati konverzija. S obzirom da XSLT pruža mogućnost pretvaranja XML dokumenata u bilo koji drugi oblik, tako ćemo ga i koristiti za potrebe ove pretvorbe. Cilj pretvorbe je iz XML dokumenta stvoriti ekvivalentni JSON, a da pri tome ne postoji gubitak pojedinih informacija i elemenata iz XML dokumenta, odnosno da se isti smanje koliko god je to moguće.



. **Sl. 9.1.** *Primjer pretvorbe XML-a u JSON*

Na Sl. 9.1. možete vidjeti jednostavan primjer na osnovu kojega možete stvoriti sliku kako bi pretvorba trebala izgledati. S obzirom da se podaci u XML-u nalaze u strukturi stabla, korištenjem XPath-a možemo pristupiti svim elementima i odraditi potrebnu pretvorbu postavljanjem određenih pravila na osnovu kojih bismo željeli odraditi konverziju.

9.1. Sintaksa XSLT-a

Prije samog koda, bilo bi dobro upoznati se s nekim od osnovnih elemenata XSL-a. Već smo spomenuli *namespace* i *template* elemente, ali kako se pristupa pojedinim elementima, na osnovu kojih oznaka? Kao što je slučaj u svim programskim jezicima, tako i u XSL-u imamo mogućnost iteracije i prolaska kroz veliki broj elemenata, što je omogućeno korištenjem sljedećih logičkih elemenata:

- Xsl:for-each select = - omogućuje nam prolazak kroz određenu grupu podataka koji se ponavljaju, a vrijednost koju predajemo „*select*“-u je XPath.
- Xsl:value-of select = - omogućuje prikaz odabranog podatka.
- Xsl:sort select = - omogućuje sortiranje podataka, ovisno o odabranom atributu.
- Xsl:if test = - postavljanje uvjeta kojeg je potrebno zadovoljiti da bi se određeni dio koda izveo.

- Xsl:choose - omogućuje biranje pojedinih podataka; koristi se zajedno s xsl:when i xsl:otherwise, gdje se postavljaju određeni testovi pomoću kojih se može određivati način na koji će se podaci obraditi.
- Xsl:template - koristimo za postavljanje određenih pravila na osnovu kojih se obavljaju potrebne mjere nad podacima.; postavljenjem dodatnog „name“ atributa omogućuje se deklariranje pojedinog tameplate-a; postavljanjem dodatnog „match“ atributa omogućuje se pozivanje definiranog tameplate-a u trenutku kad se pronađe željena vrijednost.
- Xsl:apply-templates - primjenjuje pravila koja su definirana u tameplate-ima na trenutnom elementu ili njegovoj djeci. Ukoliko predamo „select“ atribut možemo specificirati koji template želimo pozvati.
- Xsl:call-template - omogućuje poziv predanog tamplate-a

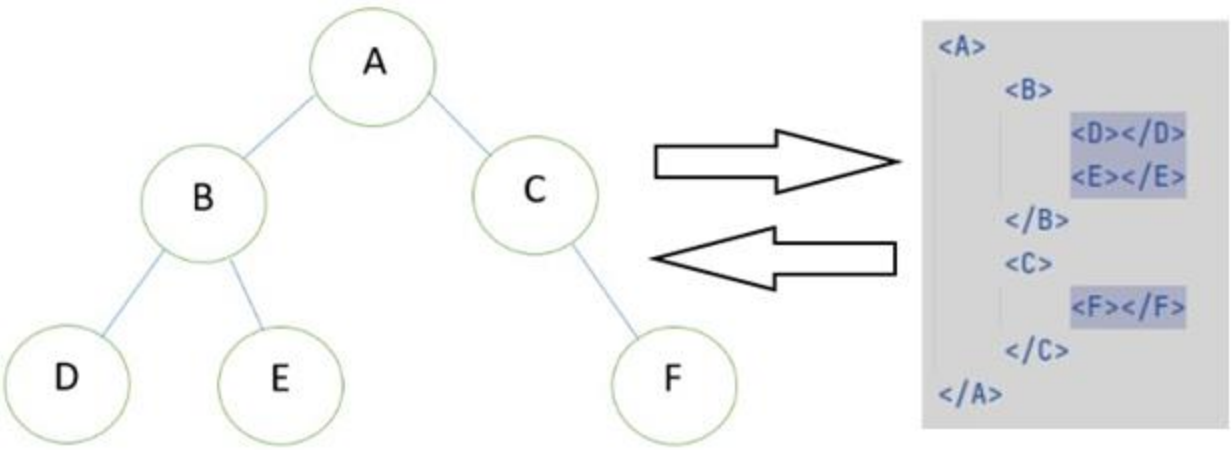
U kombinaciji s prethodno navedenim logičkim elementima, mogu se koristiti i semantičke oznake čiji odabir ovisi o tome na kakav način želite djelovati na podatke.

Semantičke oznake:

- | - Operator pridruživanja, koristi se za grupiranje čvorova u XPath izrazima
- & - and operator
- <!-- - početak komentara
- --> - kraj komentara
- $ - označava početak imena varijable
- Name() – označava ime oznake koja je trenutno u fazi obrade
- @ - predstavlja atribut u XML dokumentu, što nam omogućuje njegov odabir
- . – točka, omogućuje odabir trenutnog čvora
- .. – omogućuje odabir roditelja trenutnog čvora
- / - omogućuje odabir *root* elementa
- // - omogućuje pronalazak elementa gdje god da se on nalazio u dokumentu (na primjer //pronađi_ovaj_element će naći traženi element bez obzira gdje se nalazi u dokumentu)
- * - koristi se za odabir bilo kojeg elementa [8]

9.2. Programsko rješenje pretvorbe XML-a u JSON

Potrebno je imati na umu kako su podaci prikazani u obliku stabla te se tome mišlju voditi prilikom pristupanja podatcima, jer je bitno poznavati strukturu samog stabla i međusobni odnos njegovih čvorova. Na Sl. 9.2.1. možete vidjeti prikaz podataka u stablu i njihov ekvivalent u XML-u.



Sl. 9.2.1. *Stablo podataka prikazano u XML obliku*

Također je bitno poznavati međusobne poveznice između čvorova. Korijenski čvor (*root*) je „A“, njegova djeca su elementi „B“ i „C“, a unuci su mu „D“, „E“ i „F“. Pogledamo li čvor „B“ njegov roditelj (*parent*) je „A“, dok su mu djeca „D“ i „E“, a važno je uočiti i to da su čvorovi „B“ i „C“ u odnosu braće i sestara.

Prije nego krenemo pisati pravila na osnovu kojih će se izvršavati pretvorba, na početku dokumenta (vidi Sl. 9.2.2.) moramo definirati da se radi o XML dokumentu te koju verziju ćemo koristiti (*namespace*). Također je definirana i metoda kojom postavljamo koji tip podataka želimo koristiti kao rezultat (u ovom slučaju je to text).

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text"/>
```

Sl. 9.2.2. *Definiranje verzije XMLS-a i metode za prikaz rezultata*

Kroz podatke unutar stabla ćemo prolaziti od korijenskog čvora, pa ćemo s njegove pozicije pristupati ostalim čvorovima. Napravimo template kojim ćemo biti u mogućnosti odabrati korijenski (*root*) čvor (Sl. 9.2.3.). Nakon što uspješno odaberemo korijenski čvor, postavljamo vitičaste zagrade „{}“ unutar kojih pozivamo izraz „<xsl:apply-templates slect=“*“/>“ čime programu dajemo do znanja da želimo primijeniti sve predloške (*template*), koje ćemo naknadno definirati, nad korijenskim čvorom.

```
<xsl:template match="/">{
    <xsl:apply-templates select="*"/>}
</xsl:template>
```

Sl. 9.2.3. *Template za odabir root čvora*

Trenutno se prilikom poziva ovog predloška nad nekim ulaznim dokumentom s podacima neće dogoditi ništa osim ispisa vitičastih zagrada „{}“, jer nismo odredili što želimo napraviti s podacima, ne postoje predlošci koji se mogu pozvati za obradu podataka.

Sljedeći predložak koji ćemo definirati je podudaranje bilo kojeg čvora u predanom dokumentu te ćemo njime zadati što i kako želimo napraviti nad čvorovima (vidi Sl. 9.2.4.). Nakon što naiđemo na bilo koji čvor, provjerit ćemo ima li on braću/sestre te ukoliko je to istina, da proučavani čvor ima barem jednog brata ili sestru, onda uzimamo ime trenutno proučavanog čvora – „name()“ i zapisujemo ga u sljedećem obliku: “ime_trenutnog_čvora“ : (vidi Sl. 9.2.4.) te pozivamo predložak („*template*“) imena „*properties*“. Usput deklariramo parametar imena „*parent*“ i postavljamo mu vrijednost na 'Yes' čime automatski pretpostavljamo da je proučavani čvor roditelj. Također je nakon poziva predloška postavljen zarez, jer elementi moraju biti odvojeni zarezom, ako ih ima više od jednog. S obzirom da provjeravamo slučaj u kojemu čvor ima barem jednog brata ili sestru, onda možemo zaključiti da je potrebno dodati zarez, dok u slučaju kada čvor nema niti jednog brata ili sestru ne želimo stavljati zarez (vidi Sl. 9.2.5.), jer taj element potom smatramo zadnjim.

```
<xsl:template match="*">
    <xsl:choose>
        <xsl:when test="following-sibling::*">
            "<xsl:value-of select="name()"/>" :
            <xsl:call-template name="Properties">
            <xsl:with-param name="parent" select="'Yes'"> </xsl:with-param>
            </xsl:call-template>,
        </xsl:when>
```

Sl. 9.2.4. *Definiranje predloška za bilo koji element koji ima brata/sestru.*

```
<xsl:otherwise>
    "<xsl:value-of select="name()"/>" : <xsl:call-template name="Properties">
    <xsl:with-param name="parent" select="'Yes'"> </xsl:with-param>
</xsl:call-template>
</xsl:otherwise>
```

Sl. 9.2.5. *Definiranje predloška za bilo koji element koji ne prethodi bratu/sestri (zadnji je).*

U „*properties*“ predlošku (vidi Sl. 9.2.6.) dodatno definiramo varijablu „*child*“ te njome odabiremo prvi čvor parametra „*name*“, odnosno za roditeljski čvor koji smo posjetili, ako ima djece, uzet ćemo prvo dijete i postaviti ga kao vrijednost za „*child*“ varijablu. Za trenutni čvor je potrebno testirati ima li atribute, što postižemo izjavom test = “not(*|@*)“. Testiramo ima li trenutno proučavani čvor djecu ili atribute bilo kojeg imena. Test prolazi ukoliko proučavani čvor nema niti djecu, niti atribute bilo kojeg imena, čime se pozivaju „*when*“ – „*otherwise*“ oznake (vidi Sl. 9.2.6.). Još je potrebno testirati je li proučavani čvor roditelj, ako je onda ćemo uzeti njegovu vrijednost i staviti je unutar navodnika (“ “), no ukoliko proučavani čvor nije roditelj onda ćemo uzeti njegovo ime i vrijednost te ih ispisati u sljedećem formatu: “ime“ : “vrijednost“. Možda se pitate zašto smo za slučaj kada je čvor roditelj stavili samo njegovu vrijednost bez navođenja imena čvora čiju vrijednost prezentiramo. Pogledate li predložak na Sl. 9.2.4. bit će Vam jasno da je njegovo ime već postavljeno prije samog ulaska u „*properties*“ predložak.

```
<xsl:template name="Properties">
    <xsl:param name="parent"></xsl:param>
    <xsl:variable name="child" select="name(*[1])"/>
    <xsl:choose>
        <xsl:when test="not(*|@*)">
            <xsl:choose>
                <xsl:when test="$parent='Yes'">
                    <xsl:text>&quot;</xsl:text><xsl:value-of select="."/><xsl:text>&quot;</xsl:text>
                </xsl:when>
                <xsl:otherwise>
                    "<xsl:value-of select="name()"/>":"<xsl:value-of  select="."/>"
                </xsl:otherwise>
            </xsl:choose>
        </xsl:when>
```

Sl. 9.2.6. *Definiranje predloška „properties“*

U slučaju da prvi test ne prolazi zadani uvjet, odnosno ako za trenutno proučavani čvor ne vrijedi da nema djecu ili atribute, onda želimo provjeriti koliko djece ima (vidi Sl. 9.2.7.). Testiramo ima

li trenutno proučavani čvor više od jednog djeteta istog imena, ako je to slučaj onda želimo grupirati te podatke u niz. Dakle, za čvor koji ima dva ili više djeteta istog imena uzimamo vrijednost svakog „*child*“ čvora te proučavani čvor zapisujemo u formatu koji je definiran uvjetima postavljenim kao na Sl. 9.2.7. Za svako dijete trenutno proučavanog čvora, provjeravamo je li to njegovo zadnje dijete. Ukoliko se radi o zadnjem djetetu te on ima više od jednog vlastitog djeteta, pozivamo „*Array*“ predložak (vidi Sl. 9.2.8.) na tom djetetu. Također smo postavili uvjete za slučajeve kada dijete ima samo jedno dijete i da to dijete nije tekstualnog tipa. Ako niti jedan od postavljenih uvjeta nije zadovoljen izvršit će se postavljanje vrijednosti trenutnog čvora.

```
<xsl:when test="count(*[name()=$child]) > 1">
    { "<xsl:value-of  select="$child"/>" : [
    <xsl:for-each select="*[name()=$child]">
        <xsl:choose>
            <xsl:when test="position() = last()">
                <xsl:choose>
                    <xsl:when test="count(node()) &gt; 1">
                        <xsl:apply-templates  select="." mode="Array"/>
                    </xsl:when>
                    <xsl:when test="(count(node()) = 1) and (not(text()))">
                        <xsl:apply-templates  select="." mode="Array"/>
                    </xsl:when>
                    <xsl:otherwise>
                        "<xsl:value-of  select="."/>"
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:when>
```

Sl. 9.2.7. *Kreiranje niza podataka uz definiranje zadnjih čvorova.*

U slučaju da se proučavani čvor ne nalazi na posljednjem mjestu, postavljamo identične uvjete kao na Sl. 9.2.7. uz dodavanje zareza nakon dohvaćanja tražene vrijednosti (vidi Sl. 9.2.9.).

```
<xsl:template match="*" mode="Array">
    <xsl:call-template name="Properties"/>
</xsl:template>
```

Sl. 9.2.8. *Kreiranje niza elemenata rekurzijom.*

Na Sl. 9.2.8. definiran je predložak kojim ćemo djelovati na čvorove s više od jednog djeteta, odnosno na čvorove koji će biti predstavljeni nizom. Nakon što otkrijemo takav čvor (da ima više od jednog djeteta) pozivamo predložak koji će nam reći što treba napraviti s njim (vidi Sl. 9.2.6.), a to je da se nad njim primjeni „*properties*“ predložak (Sl. 9.2.8.).

```
<xsl:otherwise>
    <xsl:choose>
        <xsl:when test="count(node()) &gt; 1">
            <xsl:apply-templates  select="." mode="Array"/>,
        </xsl:when>
        <xsl:when test="(count(node()) = 1) and (not(text()))">
            <xsl:apply-templates  select="." mode="Array"/>,
        </xsl:when>
        <xsl:otherwise>
            "<xsl:value-of  select="."/>",
        </xsl:otherwise>
    </xsl:choose>
</xsl:otherwise>
```

Sl. 9.2.9. *Kreiranje niza podataka uz definiranje čvorova koji se ne nalaze na poslijednjem mjestu.*

U slučaju da proučavani čvor nema više djece istog imena, za svaki od njih moramo provjeriti na kojoj se poziciji nalazi (vidi Sl. 9.2.10.) te u skladu s tom informacijom postaviti/ne postaviti zarez i pozvati predložak za stvaranje niza (Sl. 9.2.8.).

```
<xsl:if test="*[not(name()=$child)]">
    ,
    <xsl:for-each select="*[not(name()=$child)]">
        <xsl:choose>
            <xsl:when test="position() = last()">
                <xsl:apply-templates select="." mode="Array"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:apply-templates select="." mode="Array"/>,
            </xsl:otherwise>
        </xsl:choose>
    </xsl:for-each>
</xsl:if>
```

Sl. 9.2.10. *Definiranje djelovanja na čvor s atributom, bez djece.*

Kako bismo ispravno definirali atribute, potrebno je provjeriti na kojoj poziciji se nalazi pojedini atribut te u skladu s tim postaviti njegovo ime i vrijednost (vidi Sl. 9.2.11.), uzimajući u obzir slučaj u kojemu je potrebno staviti zarez. Za ostvarivanje ispravnog JSON formata je vrlo bitno paziti na postojanje i položaj zareza, tako moramo uzeti u obzir da se u slučaju gdje proučavani čvor ima tekstualnu vrijednost atribut mora definirati u formatu sa zarezom (“@ime_atributa“ : “vrijednost_atributa“,).

```
<xsl:when test="./text()">
    <xsl:for-each select="@*">
        "@<xsl:value-of select="name()"/>" : "<xsl:value-of select="."/>",
    </xsl:for-each>
</xsl:when>
<xsl:otherwise>
    <xsl:for-each select="@*">
        <xsl:choose>
            <xsl:when test="position() = last()">
                "@<xsl:value-of select="name()"/>" : "<xsl:value-of select="."/>"
            </xsl:when>
            <xsl:otherwise>
                "@<xsl:value-of select="name()"/>" : "<xsl:value-of select="."/>",
            </xsl:otherwise>
        </xsl:choose>
    </xsl:for-each>
</xsl:otherwise>
```

Sl. 9.2.11. *Postavljanje uvjeta za obradu čvorova s atributima.*

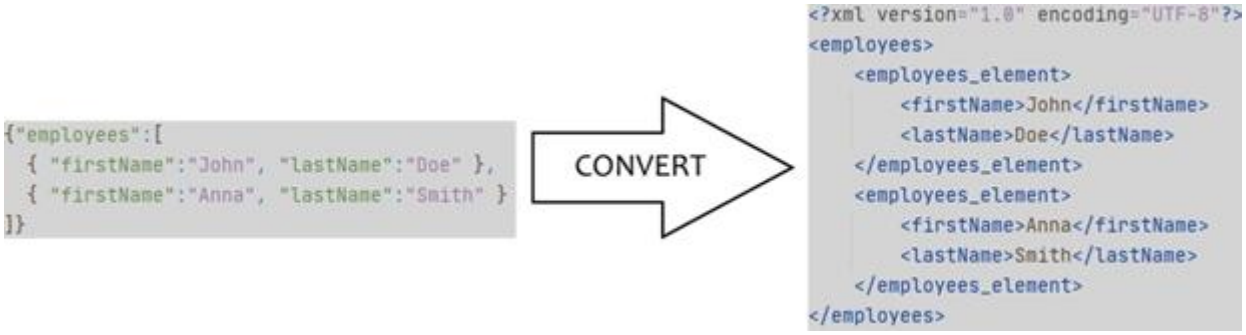
Ako postoji čvor koji nema niti djecu niti atribute, a ima neki tekst ili je jednostavno bez predane vrijednosti, možemo ga obraditi prema predlošku definiranom na Sl. 9.2.12. Predložak će se odraditi onda kad unutar XML dokumenta naiđe na neki tekst. Prvo se provjerava ima li proučavani tekst braću ili sestre, ukoliko je to istina bit će predstavljen u zadanoj formi s korištenjem zareza na kraju. Dok za slučaj kada nema brata ili sestru, jednostavno možemo zaključiti da se radi o posljednjem elementu, pa prema tome i postavljamo format bez korištenja zareza na kraju.

```
<xsl:template match="text()">
    <xsl:choose>
        <xsl:when test="following-sibling::*">
            "#text" : "<xsl:value-of select="."/>",
        </xsl:when>
        <xsl:otherwise>"#text" : "<xsl:value-of select="."/>"
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Sl. 9.2.12. *Način obrade teksta.*

10. PRETVORBA JSON-A U XML

JSON podaci su specifični po tome što se prezentiraju uređenim parovima ključ-vrijednost. Potrebno je naglasiti kako ključevi moraju biti tipa string, dok vrijednosti moraju poštivati pravila korištenja ispravnog JSON tipa podataka (poput objekta, niza, boolean, null vrijednosti, broj ili string). Prilikom pridruživanja pojedine vrijednosti odgovarajućem ključu, potrebno je koristiti se dvotočkom („:“), a za međusobno odvajanje uređenih parova koriste se zarezi. Svi JSON objekti se nalaze unutar vitičastih zagrada („{}“). (9)



Sl. 10.1. *Primjer pretvorbe JSON-a u XML.*

Kako bi pretvorba iz JSON-a u XML bila ostvariva, potrebno je dodati *root* element JSON podacima (vidi Sl. 10.2.). Nakon što je osiguran *root* element bilo kojim podacima koje korisnik predaje, možemo započeti s prolaskom kroz JSON format podataka (vidi Sl. 10.3.).

```
String str = "<template>" + usersInput.getText() +"</template>";
```

Sl. 10.2. *Dodavanje root elementa JSON podacima.*

Na Sl. 10.3. je kreiran predložak kojim se definira način djelovanja u trenutku kada se pronađe bilo koji *root* element. Pozivanjem „object“ predloška odrađuje se parsiranje JSON podataka u elemente. Kreira se nova varijabla imena „t1“ te se poziva novi predložak „*object*“ kojemu kao parametar predajemo podatke koje želimo pretvoriti u XML.

```
<xsl:template match="/*">
    <xsl:variable name="t1">
        <xsl:call-template name="object">
            <xsl:with-param name="json-in" select="." />
        </xsl:call-template>
    </xsl:variable>
```

Sl. 10.3. *Pronalazak root elementa u JSON formatu.*

```
<xsl:template name="object">
    <xsl:param name="json-in" />
    <xsl:param name="parent-ele" select="''" />
    <xsl:variable name="t1" select="normalize-space(substring-after($json-in,'{'))" />
    <xsl:variable name="t2">
        <xsl:call-template name="fields">
            <xsl:with-param name="json-in" select="$t1" />
        </xsl:call-template>
    </xsl:variable>
    <xsl:variable name="t3" select="normalize-space(substring-after( exsl:node-set($t2)/so:extra, '}'))" />
    <so:output>
        <xsl:choose>
            <xsl:when test="$parent-ele">
                <xsl:element name="{$parent-ele}">
                    <xsl:copy-of select="exsl:node-set($t2)/so:output/node()" />
                </xsl:element>
            </xsl:when>
            <xsl:otherwise>
                <xsl:copy-of select="exsl:node-set($t2)/so:output/node()" />
            </xsl:otherwise>
        </xsl:choose>
    </so:output>
    <xsl:if test="$t3">
        <so:extra><xsl:value-of select="$t3" /></so:extra>
    </xsl:if>
</xsl:template>
```

Sl. 10.4. *Predložak za objekte.*

Svi podaci koji su predani u JSON formatu parsiraju se nakon poziva root predloška. Unutar samog predloška za objekte, stvaraju se dva parametra, „json-in“ koji predstavlja podatke koje je predao korisnik i „parent-ele“ koji će predstavljati roditelja te se inicijalizira na null vrijednost. U „t1“ varijablu se sprema vrijednost ulaznih podataka izuzevši prvu vitičastu zagradu „{“ predanog stringa. Zatim se poziva predložak „fields“ uz predavanje vrijednosti varijable „t1“ te se prije nastavka „object“ predloška, do kraja mora izvršiti pozvani predložak.

```
<xsl:template name="fields">
    <xsl:param name="json-in" />
    <xsl:variable name="n" select="normalize-space($json-in)" />
    <xsl:choose>
        <xsl:when test="substring($n,1,1) = $quot">
            <xsl:variable name="t1">
                <xsl:call-template name="field">
                    <xsl:with-param name="json-in" select="$n" />
                </xsl:call-template>
            </xsl:variable>
            <xsl:variable name="t2" select="normalize-space( exsl:node-set($t1)/so:extra) " />
            <xsl:variable name="t3">
                <xsl:choose>
```

Sl. 10.5. *Fields predložak.*

Podaci nad kojima je potrebno izvršiti pretvorbu se sastoje od uređenih parova, ključ-vrijednost koji se zapisuju unutar navodnih znakova, pa je prema tome potrebno utvrditi radi li se o nekoj vrijednosti koja će odgovarati formi uređenog para, ili je program naišao na neki drugi tip podatka (poput teksta, null vrijednosti, atributa). Ova provjera se realizira korištenjem „substring($n,1,1)=$quot“ funkcije, kojom uspoređujemo prvi znak proučavanih podataka sa znakom navodnika (Sl. 10. 5.). U slučaju da funkcija vrati istinitu vrijednost, došlo bi do pozivanja „field“ predloška (Sl. 10. 6.). Za više JSON podatka koji imaju formu kao na Sl. 10.7. koristimo „fields“ predložak kojim se vrši parsiranje ovakvih podataka.

```
<xsl:template name="field">
    <xsl:param name="json-in" />
    <xsl:variable name="field-name" select="substring-before(substring-after($json-in,$quot),$quot)" />
    <xsl:variable name="remainder" select="substring-after($json-in,':')" />
    <xsl:call-template name="value">
        <xsl:with-param name="json-in" select="$remainder" />
        <xsl:with-param name="parent-ele" select="$field-name" />
    </xsl:call-template>
</xsl:template>
```

Sl. 10.6. *Field predložak.*

```
"id": "file",
"value": "File"
```

Sl. 10.7. *Primjer Json-a nad kojim se koristi „fields“ predložak.*

Predloškom na Sl. 10.6. se parsira jedan JSON unos oblika: „Price“:“20“. Definirano je dohvaćanje vrijednosti koja se nalazi unutar navodnih znakova, te ostatak podataka koji još nisu obrađeni, a nalaze se iza znaka dvotočke („:“). Potom se poziva „value“ predložak koji definira način na koji će se prezentirati podaci ovisno o znaku koji se pročita (vidi Sl. 10.7. i 10.8.).

```
<xsl:when test="$first-letter='{'">
    <xsl:call-template name="object">
        <xsl:with-param name="json-in" select="$json-in" />
        <xsl:with-param name="parent-ele" select="$parent-ele" />
    </xsl:call-template>
</xsl:when>
<xsl:when test="$first-letter='['">
    <xsl:call-template name="array">
        <xsl:with-param name="json-in" select="$json-in" />
        <xsl:with-param name="parent-ele" select="$parent-ele" />
    </xsl:call-template>
</xsl:when>
:xsl:when test="$first-letter=$quot">
    <xsl:call-template name="string">
        <xsl:with-param name="json-in" select="$json-in" />
        <xsl:with-param name="parent-ele" select="$parent-ele" />
    </xsl:call-template>
</xsl:when>
```

Sl. 10.7. *Postavljanje pravila za obradu u slučaju pojave objekta, niza ili stringa.*

```
<xsl:when test="contains($numbers,$first-letter)">
    <xsl:call-template name="number">
        <xsl:with-param name="json-in" select="$json-in" />
        <xsl:with-param name="parent-ele" select="$parent-ele"/>
    </xsl:call-template>
</xsl:when>
<xsl:when test="contains($booleans,$first-letter)">
    <xsl:call-template name="boolean">
        <xsl:with-param name="json-in" select="$json-in" />
        <xsl:with-param name="parent-ele" select="$parent-ele"/>
    </xsl:call-template>
</xsl:when>
<xsl:otherwise>
    <so:output>ERROR</so:output>
</xsl:otherwise>
```

Sl. 10.8. *Pravila za obradu u slučaju pojave boolean vrijednosti ili brojeva.*

```
<xsl:template name="string">
    <xsl:param name="json-in" />
    <xsl:param name="parent-ele" />
    <xsl:variable name="value" select="substring-before(substring-after($json-in,$quot),$quot)" />
    <xsl:variable name="remainder" select="normalize-space(substring-after(substring-after($json-in,$quot),$quot))" />
    <so:output>
                <xsl:element name="{$parent-ele}">
                    <xsl:value-of select="$value" />
                </xsl:element>
    </so:output>
    <xsl:if test="$remainder">
        <so:extra><xsl:value-of select="$remainder" /></so:extra>
    </xsl:if>
</xsl:template>
```

Sl. 10.9. *String predložak.*

Za „string“ predložak (vidi Sl.10.9.) su definirane varijable koje će uzeti vrijednost unutar navodnih znakova te će ostatak podataka predviđenih za pretvorbu spremiti u varijablu „remainder“. Ispisuje se vrijednost roditelja te ukoliko nije odrađena pretvorba svih podataka, odnosno, ako postoji vrijednost unutar „remainder“ varijable (true) nastavljamo s kodom.

„Number“ predložak radi na sličnom principu kao i „string“ predložak, samo što se poziva u slučaju kada je pronađen broj u JSON dokumentu, jer brojevi neće imati navodnike. Boolean je isto tako sličnih karakteristika, osim što se poziva ukoliko se poklope slova „t“ ili „f“.

Korištenjem „value“ predloška se vrši provjera jesu li predani JSON podaci niz, objekt, string, broj ili boolean. U skladu sa zadanim uvjetom pozvat će se odgovarajući predložak kojim je zatim definiran način stvaranja ispravnog XML formata, ovisno o tome je li program naišao na broj, boolean, string, niz ili objekt.

```
<xsl:template name="array">
    <xsl:param name="json-in" />
    <xsl:param name="parent-ele" />
    <xsl:variable name="t1" select="normalize-space(substring-after($json-in,'['))" />
    <xsl:variable name="t2">
        <xsl:call-template name="objects">
            <xsl:with-param name="json-in" select="$t1" />
            <xsl:with-param name="parent-ele" select="$parent-ele" />
        </xsl:call-template>
    </xsl:variable>
    <xsl:variable name="t3">
        <xsl:choose>
            <xsl:when test="contains(substring-before(exsl:node-set($t2)/so:extra,']'),',')">
                <xsl:value-of select="normalize-space(substring-after(exsl:node-set($t2)/so:extra,','))"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:value-of select="normalize-space(substring-after(exsl:node-set($t2)/so:extra,']'))"/>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:variable>
```

Sl. 10.10. *Array predložak.*

U slučaju da se radi o nizu podataka, „array“ predložak (vidi Sl. 10.10.) postavlja vrijednost varijable „t1“ na sve što se nalazi iza otvorenog znaka uglate zagrade („[“) zatim te podatke predaje varijabli „t2“ koja potom pozivom predloška „objects“ provjerava postoje li objekti unutar samog niza. Za uklanjanje nepotrebnih dijelova (zatvaranja *root* elementa) koristimo varijablu „t3“.

```
<xsl:template name="objects">
    <xsl:param name="json-in" />
    <xsl:param name="parent-ele" />
    <xsl:variable name="n" select="normalize-space($json-in)" />
    <xsl:choose>
        <xsl:when test="substring($n,1,1) = '{'">
            <xsl:variable name="t1">
                <xsl:call-template name="object">
                    <xsl:with-param name="json-in" select="$n" />
                    <xsl:with-param name="parent-ele" select="$parent-ele" />
                </xsl:call-template>
            </xsl:variable>
            <xsl:variable name="t2" select="normalize-space( exsl:node-set($t1)/so:extra) " />
            <xsl:variable name="t3">
```

Sl. 10.11. *Objects predložak.*

Za postavljanje glavne logike koristi se „object“ predložak kojim se provjerava prvi znak u JSON podacima i ovisno o tome radi li se o objektu ili nizu pozivaju se odgovarajući predlošci.

11. APLIKACIJA

Grafičko sučelje je ostvareno korištenjem JavaFX-a (softverska platforma za stvaranje desktop aplikacija). JavaFX se standardno koristi umjesto *Swing*-a (pruža elemente za stvaranje grafičkog sučelja u Javi) kao standardna biblioteka grafičkog sučelja u Javi.

JavaFX pruža mogućnost korištenja predefiniranih predložaka grafičkih sučelja, kao i mogućnost stvaranja vlastitog izgleda aplikacije. Aplikacija za pretvorbu podataka je ostvarena korištenjem prilagođenog grafičkog sučelja, pa je tako potrebno definirati cijeli prozor („*window*“) unutar kojega će se postavljati svi potrebni elementi čijom implementacijom će aplikacija postati kompletirana.

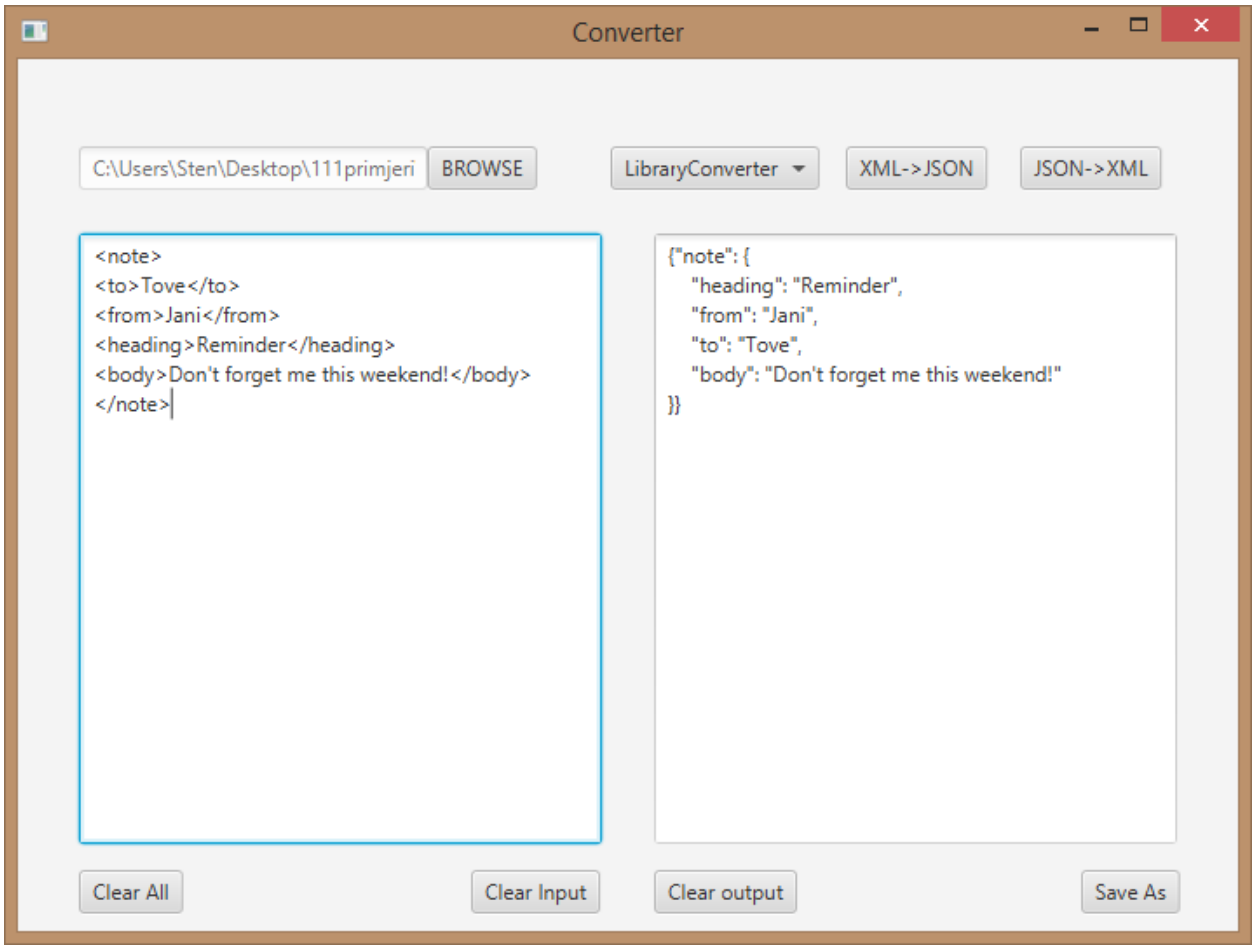
Za efikasno postavljanje grafičkog sučelja potrebno je stvoriti tekstualna polja (*TextArea* – vidi Sl. 11.1.) gdje će korisnik imati mogućnost unosa podataka nad kojima želi ostvariti pretvorbu te polje gdje će biti ispisan rezultat obrade podataka. Također je potrebno uzeti u obzir i slučajeve u kojima bi korisnik želio pretvoriti određeni dokument, pa mu je tako pružena mogućnost učitavanja pojedinog dokumenta (korištenjem „*button*-a“ „*BROWSE*“) čiji se sadržaj automatski upisuje u polje predviđeno za unos korisnikovih podataka. Ovisno o potrebi postavljena su dva „*buttona*“ (dugmeta) koji omogućuju pretvaranje XML-a u JSON i obrnuto (vidi Sl. 11.2.).

```
//creating TextBox for users input
usersInput = new TextArea();
usersInput.setLayoutX(35);
usersInput.setLayoutY(100);
usersInput.setPrefSize( prefWidth: 300, prefHeight: 350);
```

Sl. 11.1. *Stvaranje tekstualnog polja za unos podataka.*

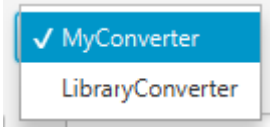
```
//naming button v2:
json_xml = new Button( text: "JSON->XML");
json_xml.setLayoutX(575);
json_xml.setLayoutY(50);
```

Sl. 11.2. *Primjer postavljanja buttona – „JSON->XML“.*



Sl. 11.3. *Aplikacija koja ostvaruje pretvorbu iz XML-a u JSON i obrnuto.*

Za potrebe uspoređivanja dobivenih rezultata pružena je mogućnost korištenja već gotovih funkcija koje dolaze zajedno s programskim jezikom (Java) ili funkcija implementiranih od strane urednika ovog rada. Prema tome padajućim izbornikom (Sl. 11.4.) su Vam na raspolaganju dvije opcije „MyConverter“ i „LibraryConverter“, pa možete birati koje biste funckije htjeli koristiti.



Sl. 11.4. *Padajući izbornik za odabir konvertera.*

Iako ne postoji mogućnost istovremenog učitavanja većeg broja dokumenata, korisnik može učitati dokument po dokument, prilikom čega će podaci unutar lijevog *TextArea* (vidi sl. 11.3.) biti sačuvani. To znači da korisnik može učitati neki „dokument1“ te nakon što se njegovi podaci upišu

u polje predviđeno za unos, može ponovno učitati novi „dokument2“ čiji podaci će se samo dodati (zajedno s podacima prethodno učitanog dokumenta – „dokument1“) u polje za upis podataka. Potrebno je paziti na oblik podataka koji predajete te na oblik u koji želite pretvoriti Vaše podatke. Ukoliko želite pretvoriti podatke koji nemaju valjanu formu XML-a ili JSON-a, nećete biti u mogućnosti odraditi pretvorbu.

Kako bi se prilikom odabira na primjer „XML->JSON“ *button*-a zaista i odradila željena akcija, moramo unutar koda definirati u kojem trenutku će se željena pretvorba dogoditi. S obzirom da se koriste *button-i* ispravno je odraditi pretvorbu, onda kada je korisnik kliknuo na pojedini *button* (deklaraciju pogledajte na Sl.11.5.). Provjerava se koji konverter je odabran („MyConverter“ ili „LibraryConverter“) te koji je *button* pritisnut, odnosno koji *button* je korisnik odabrao. U slučaju kao na Sl. 11.5. je definirano što i kako će se odraditi ukoliko je korisnik zatražio pretvorbu XML-a u JSON uz odabranu opciju „MyConverter“. Prilikom zadovoljavanja prethodno navedenih uvjeta, stvara se novi objekt klase „XSLTransformation“ imena „transform“ pomoću kojega ćemo dohvatiti funkcije za pretvorbu. Prvo uzimamo tekst koji je korisnik predao u lijevo *TextArea* polje (vidi sl. 11.3.) i spremamo ga u string „str“. Pretvoreni podaci se predaju desnom *TextArea* polju (vidi Sl. 11.3.), a prije nego što se ispišu dohvaćamo funkciju „myXmlToJson“ iz klase „XSLTransformation“ koja će kao rezultat vratiti odgovarajući JSON oblik predanih XML podataka. Fukncija „myXmlToJson“ prima dva parametra, a to su string – podaci koje korisnik predaje funkciji te boolean varijabla, koja u slučaju „*true*“ vrijednosti zna da je potrebno odraditi XML u JSON transformaciju, dok za vrijednost „*false*“ obavalja JSON u XML transfomraciju.

```
public void handle(ActionEvent event){
    if(chooseConverter.getValue() == "MyConverter" && event.getSource()==xml_json) {
        XSLTransformation transform = new XSLTransformation();
        String str = usersInput.getText();

        usersOutput.setText(transform.myXmlToJson(str, chooseConverter: true));
    }
```

Sl. 11.5. *Deklaracija onoga što će se dogoditi kada korisnik klikne na „XML->JSON“ button.*

S obzirom da se trenutni sadržaj, prilikom učitanjava novog dokumenta, ne briše iz polja za unos podataka, napravljen je *button – „Clear Input“* koji pruža tu mogućnost. Također su dodani *button*-i za brisanje podataka unutar polja gdje se prikazuju pretvoreni podaci (desni *TextArea* – vidi Sl. 11.3.) te za brisanje podataka unutar oba *TextArea* polja.

Nakon što korisnik uspješno odradi pretvorbu podataka iz XML-a u JSON ili iz JSON-a u XML, može spremiti dobiveni rezultat, kao dokument, na svoje računalo. Spremljene rezultate korisnik potom može jednostavnije koristiti za vlastite potrebe.

12. ZAKLJUČAK

U radu je prezentirana važnost omogućivanja pretvorbe XML formata u JSON i obrnuto. Podaci su općenito vrlo osjetljivi i potrebno je ispravno rukovati njima, prema tome treba paziti i na način na koji ih pohranjujemo i prenosimo. Zbog ogromnog broja web usluga i aplikacija općenito, čiji broj svakog dana sve više raste, može doći do nekompatibilnosti prilikom prijenosa podataka. U svrhu čuvanja originalne forme podataka, moramo imati pristup kvalitetnom pretvorniku koji će izvršiti pretvorbu XML-a u JSON bez gubitka značenja originalnog dokumenta, jer u slučaju gubitka i najmanje inačice može doći do velikih grešaka i odstupanja od prvobitnog značenja. Ostvareno aplikacijsko rješenje za pojedine primjere zahtijeva više vremena za obradu, nego što je to slučaj kod korištenja gotovih funkcija. Brzina prijenosa podataka je također vrlo bitan faktor, pa bi prilikom primjene rješenja iz ovog rada, nad podacima većih veličina bilo potrebno pričekati određeno vrijeme za dobivanje rezultata, što ostavlja prostora za poboljšanje konvertera.

Literatura

[1] XML i razlika u odnosu na HTML, https://www.youtube.com/watch?v=yUw-aTOwAw8 [30.08.2020.]

[2] Način korištenja atributa, https://www.w3schools.com/xml/xml_attributes.asp [30.08.2020.]

[3] Broj atributa koje pojedini element može imati, https://www.youtube.com/watch?v=nyk8QO08grM&t [30.08.2020.]

[4] Korištenje JSON-a, https://www.c-sharpcorner.com/article/is-json-overridden-xml/, [31.08.2020.]

[5] JSON ključ/vrijednost prezentacija, https://www.geeksforgeeks.org/python-xml-to-json/, [30.08.2020.]

[6] Podaci kod nekopatibilnih sustava, https://www.w3schools.com/xml/xml_whatis.asp, [30.08.2020.]

[7] Zastupljenost XML-a i JSON-a, https://hackr.io/blog/json-vs-xml, [30.08.2020.]

[8] XSLT sintaksa, https://www.w3.org/TR/1999/REC-xslt-19991116, [09.09.2020.]

[9] Json forma, https://www.w3schools.com/js/js_json_objects.asp, [09.09.2020.]

SAŽETAK

Naslov: Pretvorba XML i JSON formata

U ovom diplomskom radu su opisani osnovni dijelovi XML (*Extensible Markap Language*) i JSON (*JavaScript Object Notation*) formata, u svrhu razumijevanja potrebe za njihovom pretvorbom. Također ćete biti upoznati s njihovim prednostima i nedostatcima, što će Vam omogućiti ispravno donošenje odluke prilikom odabira reprezentacije Vaših podataka (XML ili JSON). Za programsko rješenje je korišten XSLT (*Extensible Stylesheet Language Transformations*), kojim se postavljaju pravila za pretvorbu iz jednog formata u drugi. Kroz rad su detaljno opisani načini pretvorbe podataka iz XML-a u JSON i obrnuto.

Ključne riječi: XML, JSON, XSLT, XPath, pretvorba podataka, konverter, Java, JavaFX

ABSTRACT

Title: XML and JSON format conversion

This thesis describes the basic parts of XML (*Extensible Markup Language*) and JSON (*JavaScript Object Notation*) formats, in order to understand the need for their conversion. You will also be introduced to their adventages and disadvantages, which will allow you to make the right decision when choosing the representation of your data (XML or JSON). XSLT (*Extensible Stylesheet Langauge Transformations*) was used for the software solution, which sets the rules for conversion from one format to another. The paper describes in detail how to convert data from XML to JSON and vice versa.

Keywords: XML, JSON, XSLT, XPath, data conversion, converter, Java, JavaFX

ŽIVOTOPIS

Sten Boban rođen je u Novoj Gradiški dana 6.6.1996. Cijeli život živi u rodnom mjestu, gdje je pohađao Osnovnu školu, OŠ „Mato Lovrak“ Nova Gradiška te srednju školu Elektrotehnička i ekonomska škola Nova Gradiška. Završetkom srednje škole stječe status „Tehničar za računarstvo“. Nakon ostvarenog odličnog uspjeha u srednjoj školi, 2015. godine upisuje sveučilišni preddiplomski studij na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija (u trenutku upisa Elektrotehnički fakultet, Osijek) u Osijeku. Uspješnim završetkom preddiplomskog studija, 2018. godine upisuje studij računarstva te je trenutno druga godina diplomskog studija računarstva smjer Informacijske i podatkovne znanosti.

U Osijeku, dana _____

Sten Boban

_____