УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Марко Вуковић

Доменски језик за опис пољопривредних биљака

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, 2019.

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска документација
Тип записа, **ТЗ**:		Текстуални штампани материјал
Врста рада, **ВР**:		Завршни (Bachelor) рад
Аутор, **АУ**:		Марко Вуковић
Ментор, **МН**:		др Зорица Сувајџин Ракић, доцент
Наслов рада, **НР**:		Доменски језик за опис пољопривредних биљака
Језик публикације, **ЈП**:		Српски / ћирилица
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		Војводина
Година, **ГО**:		2019
Издавач, **ИЗ**:		Ауторски репринт
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		7/30/0/0/7/0/0
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Примењене рачунарске науке
Предметна одредница/Кључне речи, **ПО**:		Доменски језици
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		Овим радом представљено је софтверско решење за опис пољопривредних биљака. Предложено решење се састоји од доменског језика за опис биљака и клијентске веб апликације. Доменски језик имплементиран је помоћу *JFlex* генератора скенера и *Cup* генератора парсера за програмски језик *Java*, док је клијентска веб апликација имплементирана помоћу *Spring Boot* радног оквира за креирање динамичких веб апликација и *Vue.js* радног оквира за креирање корисничких интерфејса.
Датум прихватања теме, **ДП**:		
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	**<име председника комисије>**
	Члан:	**<име члана комисије>** — Потпис ментора
	Члан, ментор:	др Зорица Сувајџин Ракић, доцент

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Bachelor Thesis	
Author, **AU**:		Marko Vuković	
Mentor, **MN**:		Zorica Suvajdžin Rakić, Assistant Professor, Ph.D.	
Title, **TI**:		Domain Specific Language for Agricultural Plant description	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		2019	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		7/29/0/0/7/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Applied computer science	
Subject/Key words, **S/KW**:		Domain specific languages	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		This paper presents a software-based solution for description of agricultural plants. The solution is made of a domain specific language for plant description, as well as a client-side web application. Domain specific language was developed in *JFlex* scanner generator and *Cup* parser generator for *Java* programming language, and the client web application was implemented in *Spring Boot* framework for building dynamic web applications and *Vue.js* framework for building user interfaces.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	**<ime predsednika komisije>**	
	Member:	**<ime člana komisije>**	Menthor's sign
	Member, Mentor:	Zorica Suvajdžin Rakić, Assistant Professor,Ph.D.	

Садржај

Списак слика

1. Увод

Евидентирање пољопривредних биљака проблем је који се може решити адекватним софтверским решењем. Циљ овог рада је представљање једне од могућих имплементација решења, која поред класичне апликације за манипулацију биљкама нуди и посебан – доменски језик, који омогућује независност дела за евиденцију биљака од клијентске апликације, а самим тим нуди могућност за промену и имплементирање нових клијентских апликација, без потребе да се на било који начин мења део апликације задужен за саму евиденцију биљака. Идеја решења је имплементирати подршку за скуп једноставних команди, које би могле да користе како крајњи корисници без знања о програмским језицима, тако и инжењери који развијају нове клијентске апликације за манипулацију биљкама. Радом је детаљно описана имплементација решења употребом следећих технологија: *JFlex* генератор скенера, *Cup* генератор парсера и *MongoDB* база података за део решења који се бави манипулацијом биљкама, док је су за израду клијентске веб апликације искоришћени *Spring Boot* и *Vue.js*.

Рад је организован на следећи начин:

- *Поглавље 2. Преглед коришћених технологија* садржи кратак преглед *JFlex* генератора скенера, *Cup* генератора парсера, *Spring Boot* радног оквира за развој динамичких веб апликација, *Vue.js* радног оквира за развој корисничких интерфејса и *MongoDB* базе података
- *Поглавље 3. Опис програмског решења* садржи детаљан опис спецификација парсера и скенера, као и детаље имплементације веб апликације
- *Поглавље 4. Приказ апликације* садржи детаљан опис апликације са приказом сваког од случаја коришћења

- *Поглавље 5. Ограничења и могућа побољшања* представља анализу решења и предлоге за његово унапређење.

Наставак рада се састоји од стандардних поглавља Закључак и Литература, које није потребно посебно објашњавати.

2. Преглед коришћених технологија

2.1 *JFlex*

JFlex је генератор скенера, написан за програмски језик *Java*. Дизајниран је како би радио заједно са *Cup* генератором парсера. Користи се тако што се напише *JFlex* програм, који садржи спецификацију скенера, и затим се на основу написане спецификације генерише класа која садржи програмски код скенера.

Скенери су програми који анализирају улазни текстуални садржај, са циљем проналажења и издвајања различитих типова скупова карактера. Типови скупова карактера најчешће се описују употребом регуларних израза – механизма за претраживање који се заснива на шаблонима. Датотека у којој се пише *JFlex* код назива се спецификација. Спецификација се састоји од листе регуларних израза упарених са акцијама које је потребно извршити при свакој детекцији израза. Спецификацију чине три сегмента раздвојена карактерима *%%*. Први сегмент представља кориснички код, који се копира на почетак генерисаног скенера, пре дефиниције класе. Ова секција најчешће се попуњава само наредбама за декларисање и укључивање пакета који ће бити коришћени. Другу секцију чине операције које нуди *JFlex* и кориснички дефинисане декларације и макрои. Опције су додатне уграђене функције *JFlex*-а које се омогућују навођењем знака *%* за којим следи назив опције и евентуални параметри. Значајне опције су: *%class* којом се омогућује промена назива класе генерисаног скенера, *%cup* која омогућује компатибилност са *Cup* генератором парсера, као и *%line* и *%column* помоћу којих се укључују бројачи редова и колона, ради побољшања порука приликом пријављивања грешака. Декларације су *Java* програмски код који се наводи између знакова *{:* и *:}*, и може послужити за дефинисање променљивих и функција које се потом могу користити у

акцијама из наредне секције. Макрои се такође могу писати у овој секцији, и служе као скраћенице за регуларне изразе. Завршну секцију спецификације чине правила која се специфицирају помоћу регуларних израза, и програмског кода између знакова *{:* и *:}*, који се извршава приликом сваке детекције израза.

JFlex представља имплементацију *Flex* генератора скенера, који све регуларне изразе интерно репрезентује као детерминистички коначни аутомат, чиме се омогућава ефикасно испитивање слагања улазног садржаја са свим обрасцима, без пада перформанси приликом дефинисања већег броја правила.

2.2 *Cup*

Cup је генератор парсера, написан за програмски језик *Java*. Користи се тако што се напише *Cup* програм, односно спецификација парсера, и затим се на основу написане спецификације генерише класа која садржи програмски код парсера.

Парсери су програми који проверавају синтаксу програма, односно исправност редоследа токена које добијају од скенера. Језик који парсери користе за овакве провере назива се контекстно независна граматика, а уобичејена форма за писање контекстно независних граматика је Бакус-Наурова форма (БНФ форма).

Cup програм, односно спецификација састоји се од четири дела. Први део чини наредба за дефинисање пакета. Други део представљају наредбе за укључивање пакета чије ће класе бити коришћене у парсеру. Трећи део чине наредбе помоћу којих се у парсер директно додаје програмски код. Постоји неколико врста наредби за додавање кода: *action code {: /* ... */ :}* додаје наредбу у датотеку у којој се генерише и парсер, али у засебну, приватну класу. Наредбом *parser code {: /* ... */ :}* програмски код се додаје у класу самог парсера. Наредба *init with {: /* ... */ :}* служи за додавање програмског кода који се извршава пре него што парсер затражи први токен од скенера, док се наредбом *scan with {: /* ... */ :}* може специфицирати на који начин парсер тражи токене од скенера. Трећи део спецификације служи за специфицирање појма од ког се започиње парсирање и то наредбом *start with*. Последњи део спецификације служи за опис граматике. Граматика се састоји од скупа форми, где се појмовима именује одређени скуп симобола и/или других појмова и придружује код који се извршава приликом њиховог успешног детектовања.

Cup је имплементиран за генерисање *LALR(1)* парсера, што значи да користи један *lookahead* токен, односно да од скенера може да добије само један токен пре него што почне да га препознаје као део обрасца или правила.

2.3 *Spring Boot, Vue.js, MongoDB*

Spring Boot је радни оквир за развијање динамичких веб апликација. Неке од погодности које пружа *Spring Boot* су то да долази са уграђеним *Tomcat* веб сервером, па се покретање врши позивањем *main* методе, као код обичних *Java* апликација. Омогућава једноставно имплементирање и конфигурисање веб апликација, употребом анотација и конфигурационих класа.

Vue.js је *JavaScript* радни оквир за развијање корисничких интерфејса. Користи комбинацију технологија: *HTML, JavaScript* и *CSS* за креирање веб страница. Нуди велики број доступних компоненти за приказ и манипулацију различитим типовима садржаја.

MongoDB је *NoSQL* база података. Подаци се у оквиру базе чувају у документима која структуром подсећају на *JSON* документа. Нуди једноставно и брзо чување и претраживање података, без потребе за експлицитним дефинисањем шеме базе података.

3. Опис софтверског решења

Доменски језик у потпуности је имплементиран у оквиру скенера и парсера, док је веб апликација само један од могућих клијената за његово једноставније коришћење.

3.1 Скенер

Скенер је имплементиран помоћу претходно представљеног генератора скенера *JFlex*. Генерисање скенера врши се на основу написане спецификације. Спецификација се састоји из три претходно описана сегмента.

Први сегмент садржи само декларацију пакета и наредбе за укључивање пакета и није га потребно додатно коментарисати.

У оквиру другог сегмента искоришћене су опције *%cup*, како би било могуће позивати скенер из парсера и опције *%line* и *%column* како би се при детекцији лексичких грешака корисницима могла приказати информативна порука, што се у коду ради на следећи начин:

```
%cup
%line
%column
```

Такође у оквиру другог сегмента спецификације, креирани су и макрои зарад поједностављења правила треће секције. Макрои су дефинисани као регуларни изрази, при чему су *text* и *broj* специфицирани као скуп карактера који се могу понављати:

```
text    = [a-zA-Z_0-9]+
broj    = [1-9][0-9]*
```

Макрои *mesec*, *mesto*, *svetlo* и *voda* специфицирани експлицитним навођењем скупова вредности од којих је дозвољено појављивање само једне вредности:

```
mesec   = "JAN"|"FEB"|"MAR"|"APR"|"MAJ"|"JUN"|"JUL"|"AVG"|"SEP"|
"OKT"|"NOV"|"DEC"
mesto   = "basta"|"rasad"
svetlo  = "senka"|"polusenka"|"sunce"
voda    = "puno"|"srednje"|"malo"
```

Трећа секција састоји се од регуларних израза или претходно описаних макроа, и токена који се враћају као повратна вредност скенера, приликом њихове детекције. Посебан токен враћа се и приликом доласка до краја датотеке, како би се омогућила подршка при раду са генерисаним парсером. Детекција свих кључних речи у имплементационом делу извршена је на униформан начин, а на примеру детекције кључне речи *dodaj*, ради се на следећи начин:

```
"dodaj"             { return new_symbol(sym.DODAJ); }
```

Детекција кључних речи као кључних речи, а не као обичног текста (попут текста за име биљке, које поседује свој токен *text*), решена је тако што се регуларни изрази који описују те кључне речи наводе пре регуларних израза који описују обичан текст, што омогућава постојање кључних речи које у оквиру команди имају посебно значење, али забрањује коришћење истих речи за имена биљака. Параметар који се прослеђује функцији *new_symbol* представља енумерацију, која се генерише из спецификације парсера, навођењем назива симбола које ће генерисани парсер користити, о чему ће бити речи у Поглављу 3.2.

Детекција карактера за раздвајање сегмената наредби једнака је за све карактере, а њен пример приказан је на карактеру = :

```
"="                 { return new_symbol(sym.JEDNAKO); }
```

Детекција дозвољених скупова карактера специфицираних макроом, униформно је имплементирана за сва правила која су специфицирана макроима, а пример илуструје детекцију месеца, из претходних примера:

```
{mesec}            { return new_symbol(sym.MESEC, yytext()); }
```

Месец се детектује као токен *MESEC*, те је потребно преузети и његову конкретну вредност, што се ради помоћу уграђене функције *JFlex*-а, *yytext()*.
Правилом:

```
"//".*             { /* Ignorisi */ }
```

омогућено је игнорисање делова кода који почињу карактерима // након којих може следити било који карактер, осим карактера за почетак новог реда. Овим правилом омогућени су линијски коментари.
Правилом:

```
\t|\f|" "|\r|\n    { /* Ignorisi */}
```

омогућено је прескакање карактера који означавају празнине у коду, па је захваљујући њему команде могуће писати и у више редова.

Последњим правилом у спецификацији парсер пријављује лексичке грешке. Ово понашање имплементирано је регуларним изразом '.', којим се обухвата било који карактер, осим карактера за означавање новог реда, који није препознат помоћу претходно описаних правила:

```
.            { System.out.println("Neispravan karakter: " + yytext()); }
```

Резултат извршавања *JFlex* генератора скенера над овом спецификацијом је *YYlex.java* датотека која садржи изворни код скенера.

3.2 Парсер

Парсер је имплементиран помоћу претходно представљеног генератора парсера *Cup*. Генерисање парсера врши се на основу написане спецификације. Средишњи део спецификације представља граматика парсера. Граматика се састоји од правила, појмова и симбола. Симболи граматике парсера су токени за кључне речи: *dodaj, pretrazi, pretrazi_raspon* и *ukloni*; затим токени за раздвајање сегмената наредби – наводник, зарез, отворена и затворена заграда, токени који представљају текст (укључујући кључне речи које означавају месеце садње и бербе, места садње, количину светлости и воде, али и

слободан текст за имена биљака и напомене), токени који представљају кључне речи назива атрибута по којима је могуће вршити претрагу (име биљке, месец садње и бербе, место садње, дубина садње, потребна количина светлости и воде) и цифре које означавају дубину садње. Ово је специфицирано следећим наредбама:

```
terminal           DODAJ, PRETRAZI, PRETRAZI_RASPON, UKLONI;
termianl           MESTO_KW, DUBINA, SVETLO_KW, VODA_KW, SADNJA, BERBA;
terminal           NAVODNIK, ZAREZ, OZAGRADA, ZZAGRADA, JEDNAKO;
terminal           IME, MESEC, MESTO, SVETLO, VODA, TEXT;
terminal Integer   BROJ;
```

Појам који се налази на врху хијерархије је команда (*komanda*). Појмови који представљају конкретне команде су команда за додавање биљке (*dodavanje*), команда за претрагу биљака по параметрима (*pretraga*) или по трајању вегетације (*pretraga_raspon*), као и команда за уклањање биљака (*uklanjanje*). Команде за додавање, претрагу по трајању вегетације и уклањање чине претходно набројани симболи, док команду за претрагу по параметрима, поред симбола чини и појам који представља потпојам претраге (*pre_part*). Потпојам претраге састоји се поново од претходно набројаних симбола.
Ови појмови декларисани су следећом наредбом:

```
      non terminal komanda, dodavanje, pretraga, pretraga_raspon, pre_part,
uklanjanje;
```

Парсирање почиње од појма која представља команду. Датотека која се парсира мора садржати тачно једну команду. Поред команде, у датотеци су дозвољени и линијски коментари који почињу двоструким знаком ‘//’. Постоје четири типа дозвољених команди: команда за додавање биљке, команда за претрагу биљака по параметрима, команда за претрагу биљака по трајању вегетације и команда за брисање биљке.

Општи облик команде за додавање биљака је следећи:

```
      dodaj “ime_biljke” (mesec_sadnje, mesec_berbe) mesto_sadnje,
dubina_sadnje, količina_svetla, količina_vode [ “ napomena ” ]
```

Име биљке је низ карактера који представља назив биљке, месеци садње и бербе наводе се са три велика почетна слова. Место садње специфицира се једном од кључних речи *basta* или *rasad*, дубина садње задаје се бројем који представља дубину садње у центиметрима, количина воде наводи се једном од кључних речи *puno*, *srednje* или *malo*, док се количина

светлости дефинише навођењем једне од предефинисаних вредности: *senka*, *polusenka* или *sunce*. Пример исправне наредбе за додавање биљке са именом *европска малина*, која се сади у месецу септембру, док се берба врши у јуну, место садње је расад, дубина садње износи 30 центиметара, и захтева пуно воде и узгајање на сунцу, уз напомену да је латински назив биљке - *Rubus idaeobatus idaeus:*

```
dodaj “Evropska_malina” (SEP, JUN) rasad, 30, sunce, puno
“Rubus_idaeobatus_idaeus”
```

Парсирање наредбе за додавање врши се од правила:

```
dodavanje ::= DODAJ NAVODNIK TEXT:ime NAVODNIK OZAGRADA MESEC:sadnja
ZAREZ MESEC:berba ZZAGRADA MESTO:mesto ZAREZ BROJ:broj ZAREZ SVETLO:svetlo
ZAREZ VODA:voda NAVODNIK TEXT:napomena NAVODNIK {: /* … */ :}
```

Приликом успешног детектовања секвенце токена, вредности тих токена прослеђују се у код између знакова *{: :}*, везивањем за локалне променљиве навођењем њихових назива иза знака ‘*:*’ уз токен. Прослеђене вредности чувају се у мапу, и прослеђују као повратна вредност правила, смештањем у променљиву *RESULT*. Наставак имплементације наредбе налази се у правилу:

```
izraz ::= dodavanje:d {: /* … */ :}
```

Претходно прослеђена мапа се преузима, вредности се чувају у бази података и мења се вредност глобалне променљиве, којом се главној апликацији јавља да је парсирање наредбе успешно извршено, ради приказивања одговарајуће поруке.

Општи облик наредбе за парсирање почиње кључном речи *pretrazi*, за којом следи непразан скуп сегмената наредби за претрагу, које се састоје од кључне речи која означава параметер по којем се претрага врши, знак једнакости који повезује параметар са вредности и саме вредности по којој се претрага врши (у случају више оваквих сегмената у једној наредби, ти сегменти раздвајају се зарезом) и изгледа овако:

```
pretrazi ime=ime_biljke, sadnja=mesec_sadnje, berba=mesec_berbe,
mesto=mesto sadnje, dubina=dubina_sadnje, svetlo=kolicina_svetla,
voda=kolicina_vode
```

Исправан пример наредбе за претрагу по почетном слову имена биљке *м* и месецу садње *септембру*, изгледао би:

```
pretrazi ime=m, sadnja=SEP
```

Парсирање наредбе за претрагу по параметрима почиње на правилу које је на нижем хијерархијском степенику од правила којим почиње парсирање наредбе за додавање. Појам *pre_part* представља градивну јединицу наредбе за претрагу коју чине кључна реч која означава параметар по ком се претражује, знак једнакости и текстуалну или нумеричку вредност податка за који се претрага врши. На примеру претраге по дубини садње, правило граматике изгледа овако:

```
pre_part ::= DUBINA JEDNAKO BROJ:t {: /* … */ :}
```

Нумеричка вредност токена *BROJ* везује се за променљиву *t*, и прослеђује у правило које обухвата појам *pre_part*:

```
pretraga ::= pretraga:mapa ZAREZ pre_part:p {: /* … */ :}
             | PRETRAZI pre_part:p {: /* … */ :};
```

Препознавање правила почиње од друге опције правила, где се препознаје кључна реч и први сегмент претраге, параметар и сегмент се смештају у мапу која представља повратну вредност правила, док се до прве опције правила стиже уколико се врши претраживање по већем броју параметара, и тада се нови пар параметар-вредност додаје у постојећу лису парова.

Општи облик наредбе за претрагу по трајању вегетације једноставнији је него наредба за претрагу по параметрима. Састоји се од кључне речи *pretrazi_raspon* и нумеричке вредности броја месеци и изгледа овако:

```
pretrazi_raspon broj_meseci
```

Исправан пример наредбе за претрагу биљака по трајању вегетације у износу од *9* месеци (распон сетпембар – јун из претходних примера) изгледао би:

```
pretraga_raspon 9
```

Парсирање наредбе за претрагу по трајању вегетације почиње у правилу:

```
pretraga_raspon ::= PRETRAZI_RASPON BROJ:broj {: /* … */ :}
```

Нумеричка вредност токена број везује се за променљиву *broj*, прослеђује се коду и враћа као повратна вредност правила, да би се у правилу вишег нивоа хијерархије извршила претрага базе података по прослеђеној вредности. Правило чији код врши ову проверу је:

```
komanda ::= pretraga_raspon:p {: /* … */ :}
```

Општи облик команде за уклањање биљке почиње кључном речи *ukloni*, за којом следи идентификациони број биљке из базе података:

```
ukloni identifikacioni_broj
```

Исправан пример команде за уклањање биљке са идентификационим бројем *abc1234* изгледао би:

```
ukloni abcd1234
```

Парсирање команде за уклањање почиње у правилу:

```
uklanjanje ::=      UKLONI TEXT: id {: /* … */ :}
```

Прослеђени идентификациони број везује се за променљиву *id* и враћа као повратна вредност правила, да би се у правилу вишег нивоа хијерархије извршило уклањање биљке из базе података. Правило које врши брисање је:

```
komanda ::= uklanjanje: u {: /* … */ :}
```

Сегмент спецификације *parser code*, који садржи код који ће бити копиран директно у класу парсера која се генерише, поред метода које иницирају парсирање садржи и *report_error* методу, којом се исписују подаци о евентуалној грешци, као и две додатне методе, методу *getDistance* која као параметар прима два назива месеца, а као повратну вредност враћа трајање вегетације биљке и њој помоћну методу *getMonth* која преводи назив месеца у његов редни број.

Резултат извршавања *Cup* генератора парсера над претходно описаном спецификацијом је *parser.java* датотека која садржи изворни код парсера, као и *sym.java* датотека која садржи дефиниције токена путем које се омогућава комуникација између скенера и парсера.

3.3 Веб апликација

Функционалну суштину веб апликације чине класе *AplikacijaApplication* и *Controller*. Прва представља почетну тачку апликације, и нуди методу од које почиње извршавање. Пре дефиниције класе налази се анотација *@SpringBootApplication*, која означава да у тој класи сервер који извршава апликацију треба да тражи *main* методу.

```
@SpringBootApplication
public class AplikacijaApplication {

 public static void main(String[] args) {
       SpringApplication.run(AplikacijaApplication.class, args);
 }
}
```

Класа *Controller* почиње анотацијом *@RestController*, која омогућује корисничком интерфејсу апликације да позива методе које су дефинисане унутар класе и означене одговарајућим анотацијама, док се анотацијом *@RequestMapping("/biljke")*, означава да ће методе класе обрађивати захтеве који садрже суфикс */biljke* на основној путањи (*http://localhost:8080*, у случају да се користи стандардни порт *8080*).

```
@RestController
@RequestMapping("/biljke")
public class Controller {
 // ...
}
```

Класа садржи методе за креирање и извршавање команди доменског језика за опис биљака, које су имплементиране на сличан начин, и биће објашњене на примеру методе *add*, за додавање биљке:

```
@PostMapping("/add")
public ResponseEntity<Boolean> add(@RequestBody Biljka b) {
 // ...
}
```

Пре декларације методе, анотацијом *@PostMapping("/add")* означава се да ће метода обрађивати захтев који на основу путању додаје суфикс */bilke/add*, и који се прослеђује помоћу *POST HTTP* методе. Повратна вредност ове методе је типа

ResponseEntity<Boolean>, што омогућава враћање одговарајућих *HTTP* статусних кодова у зависности од успеха операције (код *200* за успешно додавање и код *400* за неуспешно). Анотацијом *@RequestBody* пре улазног параметра методе означава се да ће тај параметар бити прослеђен као параметар *HTTP* захтева.

Извршавање наредбе почиње креирањем саме команде, на основу прослеђених параметара захтева.

```
    String komanda = "dodaj " + "\"" + b.getIme().replaceAll(" ", "_") + "\""
+ " (" + b.getMesecSadnje() + ", " + b.getMesecBerbe() + ") " +
b.getMestoSadnje() + ", " + b.getDubinaSadnje() + ", " + b.getSvetlo() + ", "
+ b.getVoda();

    if (b.getNapomena() != null && !b.getNapomena().equals("")) {
     komanda += " \"" + b.getNapomena().replaceAll(" ", "_") + "\"";
    }
```

Из назива се уклањају размаци и мењају доњим цртама, због ограничења језика, које не дозвољава постојање размака у текстуалним сегментима команди, ради омогућавања прескакања празнина и писања команди у више редова. Напомена се у команду додаје посебно, јер је опциона и њено навођење није обавезно. Након креирање команде, врши се њено чување у привремену текстуалну датотеку, а затим се позива статичка метода парсера *par*, којој се прослеђује путања до датотеке.

```
    try {
      fileWriter = new FileWriter(System.getProperty("user.dir") +
"/temp.txt");
      PrintWriter printWriter = new PrintWriter(fileWriter);
      printWriter.print(komanda);
      printWriter.close();

      parser.par(System.getProperty("user.dir") + "/temp.txt");

    } catch (IOException e) {
      e.printStackTrace();
    } catch (Exception e) {
      e.printStackTrace();
    }
```

По извршеном парсирању, у зависности од променљиве парсера *success*, врши се враћање одговарајућег статусног кода у *HTTP* одговору.

```
if (parser.success == true) {
 return ResponseEntity.ok().build();
} else {
 return ResponseEntity.badRequest().build();
}
```

Преостале наредбе писане су на исти начин, и неће бити детаљно анализиране, већ ће бити описане само значајније разлике.

У методи *search* за претрагу по параметрима, креирање команде мало се разликује од претходне методе, у томе што је сваки сегмент наредбе опциони, те је обухваћен *if* блоком, слично као код напомене у наредби за додавање, при чему се последњим *if* блоком уклања евентуални вишак карактера на крају наредбе.

```
String komanda = "pretrazi";

if (b.getIme() != null && !b.getIme().equals("")) {
 komanda += " ime=" + b.getIme().replaceAll(" ", "_");
 komanda += ",";
}

// ...

if (komanda.lastIndexOf(',') == komanda.length() - 1) {
 komanda = komanda.substring(0, komanda.length()-1);
}
```

Такође, ова наредба, као и наредба за претрагу биљака по трајању вегетације има другачију повратну вредност – листу биљака (*List<Biljka>*). Због ограничења доменског језика, која су описана у *Поглављу 5. Ограничења и могућа побољшања решења*, потребно је мало стилских улепшавања података пре њиховог приказа (замена карактера ‘_’ размаком), што је описано наредним приказом кода:

```
List<Biljka> ret = new ArrayList<>();

for (DBObject obj : res) {
 Biljka temp = new Biljka();
 for (String key : obj.keySet()) {
       if (key.equals("ime")) {
                   temp.setIme(obj.get("ime").toString());
        } else if (key.equals("sadnja")) {
               temp.setMesecSadnje(obj.get("sadnja").toString());
        } else if (key.equals("berba")) {
               temp.setMesecBerbe(obj.get("berba").toString());
        } else if (key.equals("mesto")) {
               temp.setMestoSadnje(obj.get("mesto").toString());
        } else if (key.equals("dubina")) {
               temp.setDubinaSadnje(obj.get("dubina").toString());
        } else if (key.equals("svetlo")) {
               temp.setSvetlo(obj.get("svetlo").toString());
        } else if (key.equals("voda")) {
               temp.setVoda(obj.get("voda").toString());
        } else if (key.equals("napomena")) {
               temp.setNapomena(obj.get("napomena").toString());
        } else if (key.equals("_id")) {
               temp.setId(obj.get("_id").toString());
        }
 }
 ret.add(temp);
}
```

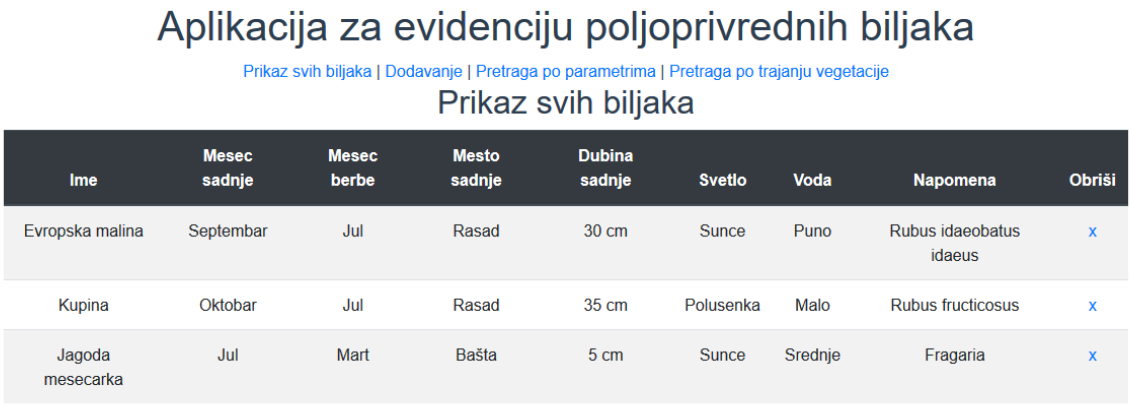
Разлика присутна у последње две методе у односу на претходне је у томе што уместо *POST HTTP* методе обрађују *GET* методу, што се специфицира анотацијом *@GetMapping*, и у томе што параметре прихватају из путање, а не из тела захтева. На примеру методе за претрагу биљака по трајању вегетације, то је урађено на следећи начин:

```
@GetMapping("/search/{range}")
public List<Biljka> searchRange(@PathVariable("range") Integer range) {
 // ...
}
```

Садржај сегмента путање *range*, везује се за улазни параметар *range* типа *Integer*, помоћу анотације *@PathVariable(“range”)*, где се параметром те анотације одређује који сегмент путање се смешта у промењљиву која следи иза те анотације.

4. Приказ апликације

Почетна страница апликације представља табеларни приказ свих биљака у систему, и приказана је на Слици 1. На врху сваке странице, испод наслова, доступан је навигациони панел са везама за навигирање.

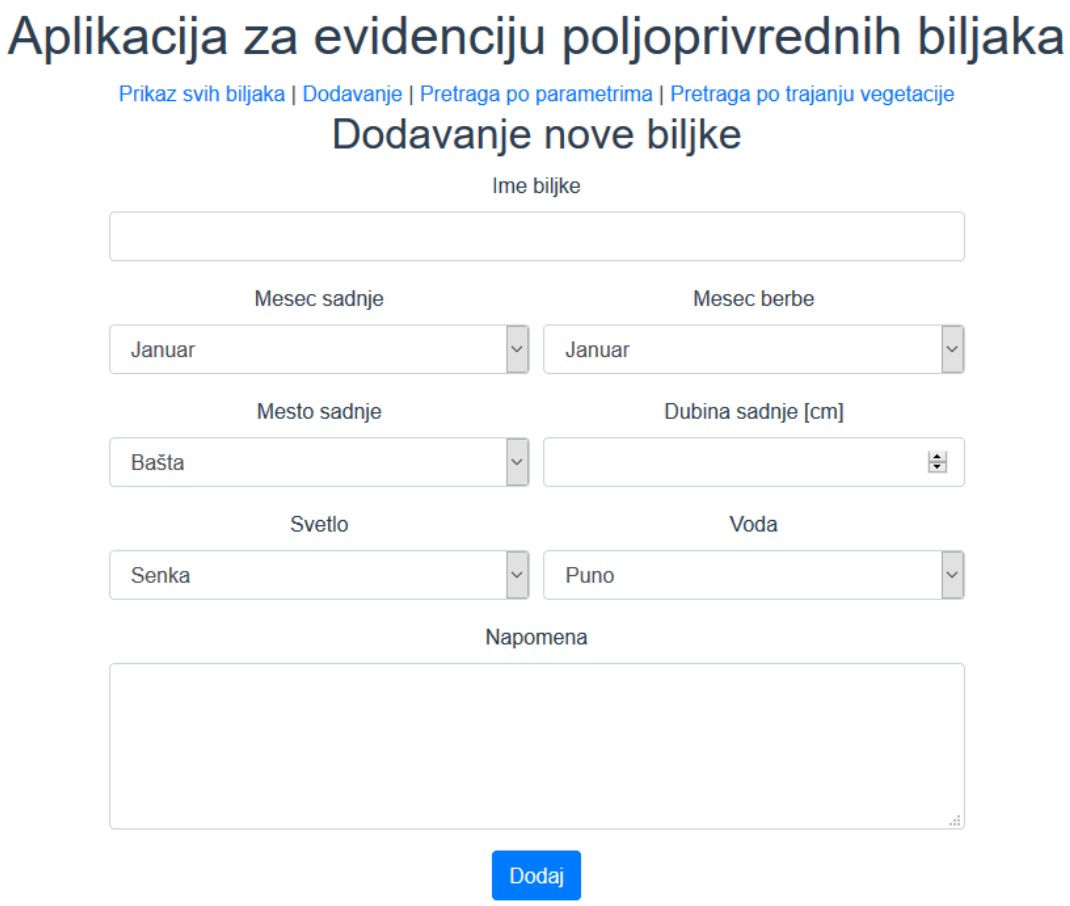


Слика 1 – Страница за приказ свих биљака

Кликом на везу *x* у постојећим редовима табеле који представљају конкретне биљке, врши се њихово уклањање.

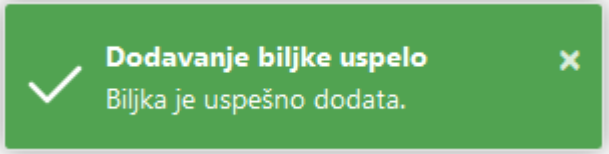
Одабиром везе *Dodavanje* прелази се на картицу намењену додавању нове биљке у систем. Ова страница састоји се од поља за унос имена биљке, поља за избор месеци садње и бербе, поља за одабир места садње, простора за унос дубине на којој се биљка сади, и поља за одабир количине светла и воде потребних за узгајање биљке. Такође,

присутно је и поље за опциону напомену, које није обавезно попунити. Изглед странице приказан је на Слици 2.

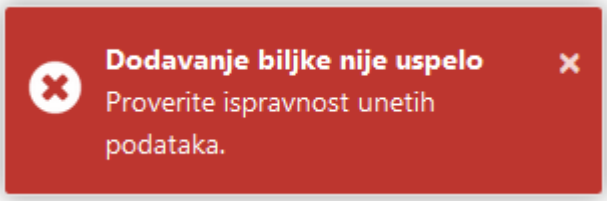


Слика 2 – Изглед странице за додавање биљке

Кликом на дугме *Dodaj* врши се креирање и извршавање наредбе за додавање и у доњем десном углу странице приказује се једна од две могуће поруке. Приликом успешног додавања биљке, кориснику се приказује порука са Слике 3, док се при неуспешном додавању (приликом изостављања неког од обавезних поља и/или коришћења неисправних карактера) приказује порука са Слике 4.

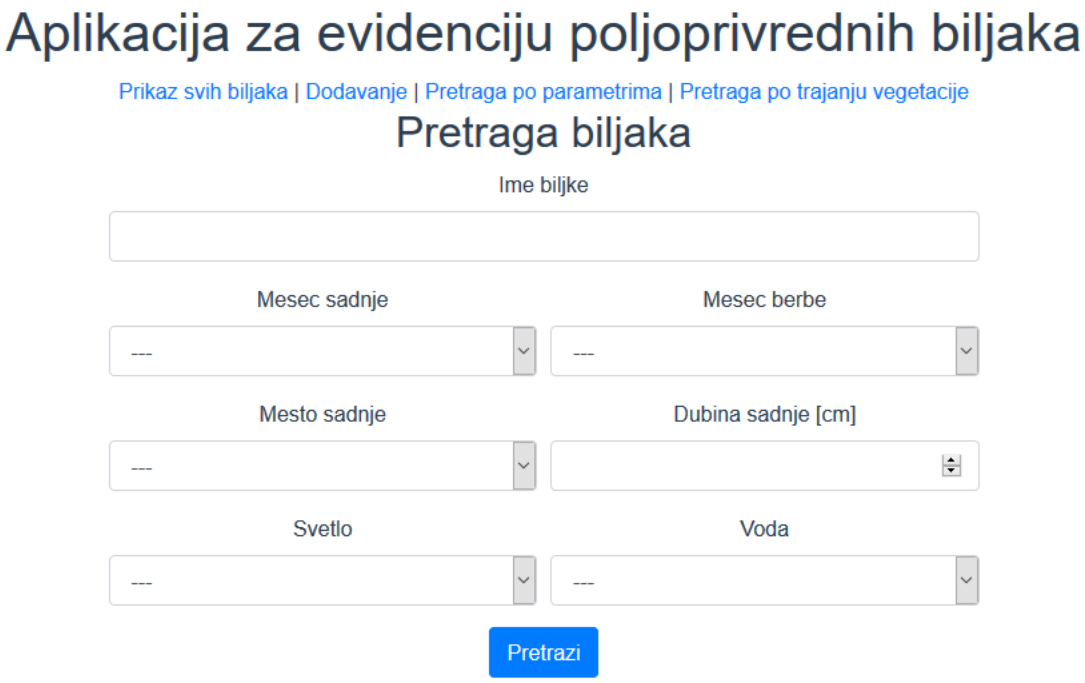


Слика 3 – Порука приликом успешног додавања биљке



Слика 4 – Порука приликом неуспешног додавања биљке

Одабиром везе *Pretraga po paramterima* прелази се на страницу за претрагу биљака, која је приказана на Слици 5. На страници су присутна поља која се налазе и на страници за додавање нове биљке, без поља за напомену.

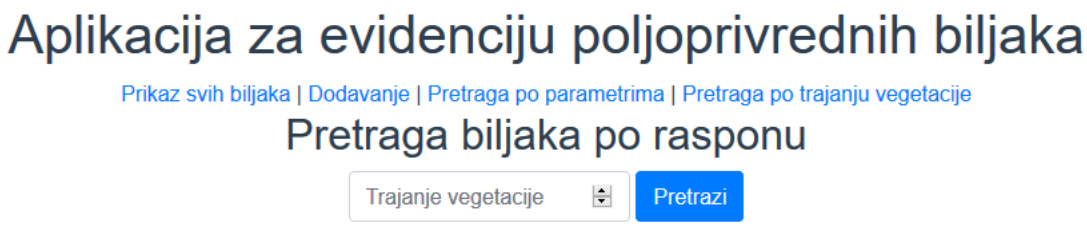


Слика 5 – Страница за претрагу биљака по параметрима

Кликом на дугме *Pretrazi*, уколико у бази постоје биљке које одговарају параметрима претраге, испод форме биће приказане у виду картица са свим подацима о њима, као што се може видети на Слици 6.

Слика 6 – Резултат претраге

Одабиром везе *Pretraga po trajanju vegetacije* прелази се на страницу за претрагу биљака по броју месеци који протекне од месеца садње до месеца бербе, која је приказана на Слици 7.



Слика 7 – Страница за претрагу биљака по трајању вегетације

Кликом на дугме *Pretrazi* излиставају се резултати претраге испод форме, слично као у претходном случају, приказаном на Слици 5.

5. Ограничења и могућа побољшања решења

Ограничења која ће бити разматрана у овом поглављу углавном су последица тежње да команде доменског језика буду компактније и једноставније за коришћење, како би корисници који немају пуно искуства са програмским језицима могли да их разумеју и користе.

Први од избора за поједностављење команди је тај да се за означавање месеци у коду користе почетна три слова назива сваког месеца. Ограничење које произилази из ове одлуке је то да је потребно имплементирати посебне функције за улепшавање приказа у клијентским апликацијама, како би подаци о биљкама били читљивији. Слично важи и за називе биљака и напомене, у којима је забрањено појављивање размака, иако је ово ограничење ублажено у интерфејсу, тако што контролер клијентске апликације мења карактере размака доњим цртама.

Једно од могућих побољшања решења могло би бити и избацивање команди као таквих, и директно коришћење описа биљака у оквиру текстуалних датотека приликом извршавања операција над њима. Ово би олакшало разумевање описа крајњим корисницима, али постоји могућност да би оптеретило тимове који би радили на измени или креирању нових клијентских апликација јер би захтевало прилагођавање нетипичној конвенцији што се тиче садржаја текстуалних датотека, па се при изради овог решења задржало на употреби концепта команди.

Даља ограничења су претежно су техничке природе, како би се омогућило функционисање решења на што већем броју платформи. Једно од тих ограничења последица је и одлуке да се у командама доменског језика користе искључиво карактери *ASCII* табеле, што значи да у називима биљака и напоменама није дозвољено користити посебна слова српске латинице (*č, ć, š, ž и đ*), као ни ћирилично писмо или интерпункцијске знакове (‘,’, ‘*!*’, ‘*?*’ и слично). Највећи број ових ограничења односе се на

читљивост команди, која је субјективна, и не би имала посебног утицаја на употребљивост језика или програмског решења.

6. Закључак

Овај рад представио је једно од могућих решења за проблем евидентирања пољопривредних биљака. Главна идеја изложеног решења је у томе да се део за само евидентирање биљака издвоји у посебан независан програмски код, са којим је могуће комуницирати посредством команди доменског језика. На овај начин омогућује се прилагођавање великом броју потенцијалних корисника, од оних искусних који би могли да искористе брзину евидентирања биљака у систему посредством директног уноса команди, без губљења времена до ког долази навигирањем кроз графички интерфејс, до корисника који нису вични у раду са рачунаром, који би ценили једноставност клијентске апликације. Такође, оставља се могућност унапређења или креирања потпуно нове клијентске апликације, тиме што је главни део решења, који је задужен за саму евиденцију биљака, у потпуности одвојен од клијентске апликације.

7. Литература

[1] *JFlex User's Manual*, https://jflex.de/manual.pdf (август 2019)

[2] *CUP User's Manual*, http://www2.cs.tum.edu/projects/cup/docs.php (август 2019)

[3] Zorica Suvajdžin Rakić, Predrag Rakić, *flex & bison*, http://www.acs.uns.ac.rs/sr/filebrowser/download/4392010 (август 2019)

[4] *Spring Boot Reference Documentation*, https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/ (септембар 2019)

[5] *Vue.js Introduction*, https://vuejs.org/v2/guide/ (септембар 2019)