



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Јелена Хрњак

***SECURADSL – НАМЕНСКИ ЈЕЗИК ЗА
ПОДРШКУ БРЗОГ УСПОСТАВЉАЊА
КОНФИГУРАЦИЈЕ БЕЗБЕДНОСНИХ
АСПЕКТА У РАДНОМ ОКВИРУ
SPRING***

Мастер рад
- Мастер академске студије -

Нови Сад, 2023.



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:	Мастер рад
Аутор, АУ:	Јелена Хрњак
Ментор, МН:	др Владимир Димитриески, ванредни професор
Наслов рада, НР:	<i>SecuraDSL</i> – Наменски језик за подршку брзог успостављања конфигурације безбедносних аспеката у радном оквиру <i>Spring</i>
Језик публикације, ЈП:	Српски / ћирилица
Језик извода, ЈИ:	Српски
Земља публиковања, ЗП:	Република Србија
Уже географско подручје, УГП:	Војводина
Година, ГО:	2023
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)	6/77/23/18/24/0/0
Научна област, НО:	Електротехничко и рачунарско инжењерство
Научна дисциплина, НД:	Примењене рачунарске науке и информатика
Предметна одредница/Кључне речи, ПО:	Доменски оријентисано моделовање и језици
УДК	
Чува се, ЧУ:	Библиотека Факултета техничких наука, Нови Сад
Важна напомена, ВН:	
Извод, ИЗ:	У овом раду описан је наменски језик <i>securaDSL</i> за моделовање <i>Spring</i> апликација са безбедном конфигурацијом подржаном за три безбедносна механизма: основну аутентификацију, <i>JWT</i> и <i>OAuth2.0</i> . Поред наменског језика, развијени су генератори извршивог кода на основу модела креираног помоћу језика <i>securaDSL</i> . За развој наменског језика и генератора коришћено је окружење <i>Eclipse Modeling Framework</i> .
Датум прихватања теме, ДП:	
Датум одбране, ДО:	
Чланови комисије, КО:	Председник: др Соња Ристић, редовни професор
	Члан: др Милан Челиковић, доцент
	Члан, ментор: др Владимир Димитриески, ванредни професор
	Потпис



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 :	Graduate-master Thesis	
Author, AU :	Jelena Hrnjak	
Mentor, MN :	Vladimir Dimitrieski, PhD, Associate Professor	
Title, TI :	SecuraDSL – A Domain-Specific Language for Supporting Rapid Configuration of Security Aspects in the Spring Framework	
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 :	2023	
Publisher, PB :	Author's reprint	
Publication place, PP :	Novi Sad, Faculty of Technical Sciences, Dositeja Obradovica sq. 6	
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	6/77/23/18/24/0/0	
Scientific field, SF :	Electrical and computer engineering	
Scientific discipline, SD :	Applied computer science and informatics	
Subject/Key words, S/KW :	Domain-Specific Modeling Languages	
UC		
Holding data, HD :	The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, N :		
Abstract, AB :	In this thesis, we present a domain-specific language securaDSL, designed for modeling Spring applications with security configuration, supporting three security mechanisms: Basic Authentication, JWT and OAuth2.0. Additionally, we present multiple generators that produce executable code based on models created using securaDSL. For the development of the domain-specific language and generators we used Eclipse Modeling Framework environment.	
Accepted by the Scientific Board on, ASB :		
Defended on, DE :		
Defended Board, DB :	President:	Sonja Ristić, PhD, Full Professor
	Member:	Milan Čeliković, PhD, Assistant Professor
	Member, Mentor:	Vladimir Dimitrieski, PhD, Associate Professor
		Mentor's sign

	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Датум:
	ЗАДАТАК ЗА ДИПЛОМСКИ РАД	Лист/Листова:

(Податке уноси предметни наставник - ментор)

Студијски програм:	Рачунарство и аутоматика		
Руководилац студијског програма:	др Мирна Капетина, ванредни професор		
Студент:	Јелена Хрњак	Број индекса:	E2 64/2022
Област:	Електротехничко и рачунарско инжењерство		
Ментор:	др Владимир Димитриески, ванредни професор		
НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА МАСТЕР РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА: <ul style="list-style-type: none"> - проблем – тема рада; - начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; - литература 			

НАСЛОВ ДИПЛОМСКОГ РАДА:

SecuraDSL – Наменски језик за подршку брзог успостављања конфигурације безбедносних аспеката у радном оквиру <i>Spring</i>

ТЕКСТ ЗАДАТКА:

<ul style="list-style-type: none"> • Проучити основне концепте који се користе приликом моделовања <i>Spring</i> апликација са безбедносном конфигурацијом. • Анализирати постојеће језике за моделовање веб апликација. • Имплементирати наменски језик и алат за моделовање <i>Spring</i> апликација са безбедносном конфигурацијом. • Имплементирати генераторе који на основу модела описаног датим језиком генерише извршиви код <i>Spring</i> апликације са безбедносном конфигурацијом. • Анализирати оправданост креирања наведеног решења за моделовање <i>Spring</i> апликација са безбедносном конфигурацијом и предложити даље правце развоја језика и алата.
--

Руководилац студијског програма:	Ментор рада:

Примерак за: <input type="checkbox"/> - Студента; <input type="checkbox"/> - Ментора
--

Садржај

1.	Увод.....	1
1.1	Структура рада.....	2
2.	Теоријске основе моделима вођеног развоја софтвера, наменских језика и безбедносних аспеката у радном оквиру <i>Spring</i>	3
2.1	Теоријске основе моделима вођеног развоја софтвера и наменских језика.....	3
2.1.1	Опис технологија коришћених за развој наменског језика <i>securaDSL</i>	4
2.1.2	Опис технологија коришћених за развој генератора.....	5
2.2	Преглед безбедносних механизма у <i>Spring</i> апликацијама.....	5
2.2.1	Основна аутентификација.....	5
2.2.2	Стандард <i>JSON</i> веб токен.....	6
2.2.3	Стандард <i>Open Authorization</i>	6
3.	Преглед постојећег стања у области.....	7
3.1	Недостаци код постојећих начина за моделовање веб апликација.....	7
4.	Наменски језик за подршку брзог успостављања конфигурације безбедносних аспеката у радном оквиру <i>Spring</i>	9
4.1	Апстрактна синтакса.....	9
4.1.1	Концепт <i>Application</i>	11
4.1.2	Концепт <i>Database</i>	12
4.1.3	Концепт <i>Entity</i>	13
4.1.4	Концепт <i>Attribute</i>	14
4.1.5	Концепт <i>User</i>	15
4.1.6	Концепт <i>Role</i>	16
4.1.7	Концепт <i>RoleInstance</i>	16
4.1.8	Концепт <i>Endpoint</i>	17
4.1.9	Концепт <i>Controller</i>	18
4.1.10	Концепт <i>Authentication</i>	20
4.1.11	Концепт <i>Security</i>	21
4.1.12	Концепт <i>BasicAuthentication</i>	21
4.1.13	Концепт <i>JWT</i>	21
4.1.14	Концепт <i>Claim</i>	22
4.1.15	Концепт <i>OAuth2</i>	25

4.1.16	Концепт <i>Provider</i>	26
4.2	Конкретна синтакса.....	26
4.2.1	Грамматика наменског језика <i>securaDSL</i>	26
4.3	Примери модела описаних наменским језиком <i>securaDSL</i>	30
4.3.1	Пример модела <i>Spring</i> веб апликације са конфигурисаном основном аутентификацијом.....	30
4.3.2	Пример модела <i>Spring</i> веб апликације са конфигурисаним безбедносним механизмом <i>JWT</i>	32
4.3.3	Пример модела <i>Spring</i> веб апликације са конфигурисаним безбедносним механизмом <i>OAuth2.0</i>	33
5.	Генерисање <i>Spring</i> веб апликација са безбедносном конфигурацијом.....	35
5.1	Генератор статичких датотека.....	35
5.2	Генератор општих конфигурационих фајлова.....	37
5.3	Генератор слоја који моделује податке из базе података.....	42
5.4	Генератор слоја за обраду захтева корисника.....	50
5.5	Генератор конфигурационих фајлова за основну аутентификацију.....	52
5.6	Генератор конфигурационих фајлова за стандард <i>JWT</i>	56
5.7	Генератор конфигурационих фајлова за стандард <i>OAuth2.0</i>	66
5.8	Тестирање генерисаних <i>Spring</i> апликација са конфигурисаном основном аутентификацијом или стандардом <i>JWT</i>	68
5.9	Тестирање генерисаних <i>Spring</i> апликација са конфигурисаним стандардом <i>OAuth2.0</i>	70
6.	Закључак.....	71
	Скраћенице.....	73
	Литература.....	75
	Биографија.....	77

1. Увод

Коришћење веб апликација представља неизоставни део свакодневног живота за велики део популације. Корисници веб апликација често нису свесни колико личних података те апликације прикупљају, обрађују и складиште и колико безбедносни пропусти могу да утичу на њих. С обзиром на осетљивост и важност података којима апликације неретко рукују, неовлашћен приступ подацима би могао да доведе до безбедносних ризика, укључујући злоупотребу и нарушавање приватности корисника. Стога, обезбеђивање високог нивоа заштите корисника и њихових података представља важан део развоја безбедних веб апликација.

Безбедност у веб апликацијама представља скуп мера и механизма који су примењени како би се корисници, систем и подаци заштитили од различитих видова напада, крађа и злоупотреба. Имплементација жељеног нивоа аутентификације и ауторизације, као два основна концепта у области безбедности, представља основу ефикасне заштите. Аутентификација представља процес потврђивања идентитета корисника или ентитета који приступа систему. Ауторизација се односи на контролу приступа одређеним ресурсима или функционалностима система.

Java [1] представља објектно-оријентисан програмски језик. Платформска независност и једноставност су особине које овај језик чине једним од најпопуларнијих избора при развоју софтвера [2]. Радни оквир *Spring* [3] чини развој серверског дела веб апликација у програмском језику *Java* бржим, једноставнијим и сигурнијим што га чини најпопуларнијим радним оквиром за ову намену [4].

Међутим, обезбеђивање одговарајуће заштите за апликације у радном оквиру *Spring* представља сложен и временски захтеван процес, те је самим тим подложен грешкама. Обзиром да се безбедносни аспекти изнова конфигуришу при почетној имплементацији сваке апликације, поред тога што је сложен и дуготрајан, овакав посао постаје и репетативан.

Са циљем уклањања наведених недостатака, тежи се оптимизацији и аутоматизацији развоја безбедних веб апликација како би се елиминисали безбедносни пропусти и грешке, али и уштедело време потребно за имплементацију. Једно од могућих решења представља аутоматско генерисање почетне *Spring* веб апликације са конфигурисаним безбедносним аспектима које се врши на основу параметара које корисник унесе, а који дефинишу основне карактеристике апликације и њених елемената. Аутоматизација процеса конфигурације безбедносних аспеката веб апликација би уштедела време експертима у пољу безбедносних конфигурација и самим тим олакшала рад, допринела квалитету софтвера и шансе за грешке свела на минимум.

За постизање овог циља креиран је наменски језик *Secura Domain-Specific Language* (*securaDSL*) за моделовање *Spring* веб апликација уз генераторе који модел

трансформишу у извршиви код. Иако је посебна пажња усмерена ка убрзању конфигурисања безбедносних аспеката, како би моделовање веб апликација било могуће, неопходно је да *securaDSL* садржи концепте за моделовање свих елемената апликације. Основни елементи односе се на метаподатке апликације, параметре базе података, слој за моделовање података складиштених у бази података, обраду захтева корисника и безбедносну конфигурацију. На основу модела и наведених параметара, генератори генеришу извршиви код написан у програмском језику *Java*, коришћењем развојног оквира *Spring*. Користећи језик *securaDSL*, експерти у пољу безбедносних конфигурација могу брзо и једноставно да дефинишу параметре апликација и конфигуришу различите безбедносне механизме помоћу синтаксе која им је лако читљива. Имплементација сигурних веб апликација на овај начин постаје једноставнија и ефикаснија, а уједно смањује могућност грешака у процесу развоја.

1.1 Структура рада

Након уводног поглавља следи поглавље „Теоријске основе моделима вођеног развоја софтвера, наменских језика и безбедносних аспеката у радном оквиру *Spring*“ у ком су описане теоријске основе моделима вођеног развоја и технологије коришћене при развоју.

Затим следи поглавље „Преглед постојећег стања у области“ где је направљен осврт на постојећа решења, пружајући увид у постојеће стандарде.

Четврто поглавље „Наменски језик за подршку брзог успостављања конфигурације безбедносних аспеката у радном оквиру *Spring*“ обухвата детаљан опис делова наменског језика и концепата које овај језик садржи уз примере модела описаних наменским језиком *securaDSL*.

У поглављу „Генерисање *Spring* веб апликација са безбедносном конфигурацијом“ описана је имплементација генератора који преводе модел у извршиви код. Детаљно су описани кораци генерисања апликације и конфигурације безбедносних аспеката и приказани су примери генерисаног кода.

Поглавље „Закључак“ садржи резултате истраживања и осврт на постигнућа у раду. Дате су препоруке за будућа унапређења и развој, као и могућност примене.

2. Теоријске основе моделима вођеног развоја софтвера, наменских језика и безбедносних аспеката у радном оквиру *Spring*

У овом поглављу дате су теоријске основе моделима вођеног развоја софтвера и наменских језика уз опис технологија коришћених при развоју. Након тога, дат је преглед безбедносних механизма у *Spring* апликацијама.

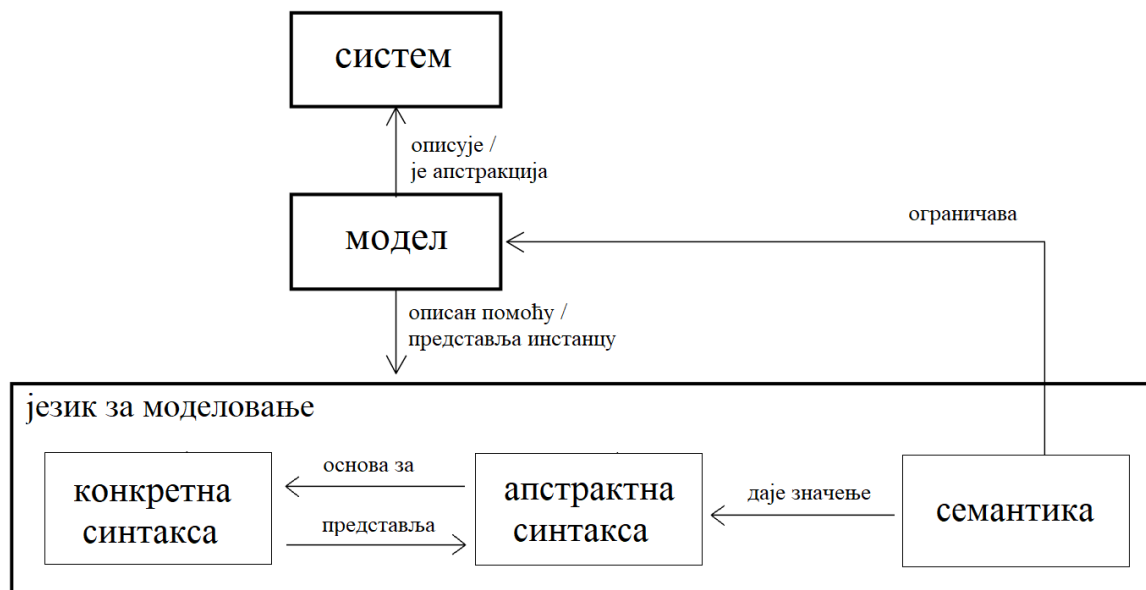
2.1 Теоријске основе моделима вођеног развоја софтвера и наменских језика

Модел представља поједностављени, апстрактни приказ неког реалног система, концепта, објекта или процеса. Доприноси бољем разумевању, анализи, развоју и тестирању. Модели не треба да описују реалност у целости, већ се у обзир узимају делови од интереса за решавање неког проблема.

Развој софтвера вођен моделима (енгл. *Model-Driven Software Development*) је методологија у којој модел представља централну тачку у процесу развоја софтвера. Развој софтвера може да постане комплексан, те његовом квалитету може да допринесе дискусија на различитим нивоима апстракције, зависно од укључених људи и фазе развоја. Модели могу да буду представљени као скице, нацрти или као програм, где се модел извршава или се на основу њега генерише извршиви код [5].

Сваки модел се креира помоћу неког језика за моделовање. Језик за моделовање прецизно дефинише синтаксу, односно нотацију модела и његову семантику, тј. значење (Слика 2.1) [6]. Синтаксу језика чине апстрактна и конкретна синтакса. Апстрактна синтакса описује структуру језика и начин на који се различити концепти могу комбиновати без обзира на репрезентацију [5]. Конкретна синтакса описује специфичну репрезентацију језика за моделовање, односно представља апстрактну синтаксу [5]. Конкретна синтакса може бити текстуална или графичка, а на основу апстрактне синтаксе може се извести више конкретних [5]. Семантика језика за моделовање описује значења и ограничења дефинисаних концепата и различите начине за њихово комбиновање. Помоћу дефинисаних концепата се креирају модели, те самим тим и модели имају јасно дефинисану семантику.

Постоје две класе језика за моделовање: наменски језици, познати и као језици специфични за домен (енгл. *Domain-Specific Languages*) и језици опште намене (енгл. *General-Purpose Languages*). Наменски језици су уско специјализовани и пројектовани за специфичан домен, контекст или компанију и развијају се како би олакшали моделовање и дискусију за специфичну намену. Неки од најпознатијих наменских језика су *HTML*, *VHDL* и *SQL*. Језици опште намене се могу применити за моделовање у било ком домену. Примери језика опште намене су *XML*, *UML* и Петријеве мреже [5]. Језик *securaDSL* представља наменски језик јер је пројектован и прилагођен домену безбедносних конфигурација *Spring* веб апликација, а његови концепти описани су у поглављу 4.



Слика 2.1 – Везе између система, модела и основних елемената језика за моделовање [5,6]

При развоју софтвера вођеним моделом, битан корак представљају и трансформације модела. Трансформација подразумева аутоматско генерисање циљног модела или текста на основу изворног модела. Уколико је резултат трансформације један или више модела, у питању је трансформација модела у модел, где програми и као улаз и као излаз имају један или више модела. Трансформација модела у текст представља генерисање текста на основу модела. У оквиру овог рада имплементирани су трансформације модела у текст, који представља извршиви код. Трансформације су постигнуте развојем генератора описаних у поглављу 5.

2.1.1 Опис технологија коришћених за развој наменског језика *securaDSL*

При развоју наменског језика *securaDSL*, први корак представља креирање апстрактне синтаксе која је приказана помоћу мета-модела. За креирање мета-модела коришћен је радни оквир за моделовање и генерисање кода *Eclipse Modeling Framework (EMF)* [7]. *EMF* као језик за мета-моделовање користи језик *Ecore* [8]. На основу мета-модела описаног помоћу овог језика, *EMF* омогућава генерисање кода који имплементира дати мета-модел. Овим језиком није било могуће приказати сва неопходна ограничења, те су додатна ограничења описана помоћу наменског језика *Object Constraint Language*, односно имплементације овог језика за моделе настале помоћу *EMF*-а под називом *Eclipse OCL* [9]. Након тога, на основу апстрактне креирана је конкретна синтакса помоћу радног оквира *Xtext* [10]. Овај радни оквир се користи за развој програмских и наменских језика. Омогућава креирање свих неопходних алата за употребу језика на основу описаних елемената језика.

2.1.2 Опис технологија коришћених за развој генератора

За развој генератора који омогућавају трансформацију модела у извршиви код, коришћен је програмски језик *Java* и језик *Xtend* [11]. *Xtend* представља дијалект програмског језика *Java* и доприноси детекцији типова података, побољшању синтаксе ламбда израза и омогућава употребу шаблона. Помоћу ламбда израза се подаци из модела лако претражују и издвајају, док се помоћу шаблона може описати изглед генерисаног кода.

2.2 Преглед безбедносних механизма у *Spring* апликацијама

Код који се генерише написан је у програмском језику *Java* коришћењем *Spring* радног оквира. *Java* је објектно-оријентисан програмски језик, а платформска независност, једноставност и објектна оријентисаност су особине које чине овај језик честим избором при развоју софтвера. *Spring* представља радни оквир за програмски језик *Java*. Омогућава лакши и бржи развој микросервисних, веб и многих других апликација. Овај радни оквир је организован у модуле који нуде функционалности за подршку различитих аспеката развоја апликација. Неки од најпознатијих модула су *Spring Boot* [12] и *Spring Security* [13]. *Spring Boot* је модул који омогућава креирање инфраструктуре за самосталне апликације које су спремне за продукцију. Конфигурација се врши помоћу алата *Maven* постављањем иницијалног *pom.xml* фајла. *Spring Security* нуди решење за аутентификацију и ауторизацију апликације. Има уграђену подршку за различите безбедносне механизме, односно стандарде и складиштење лозинки у шифрованом облику.

Подржана су три система за управљање базама података: *PostgreSQL* [14], *MySQL* [15] и *Oracle Database* [16]. *PostgreSQL* је релациони објектно-оријентисан систем за управљање базама података. Користи и проширује језик *SQL* и омогућава сигурно складиштење великог броја података. Издваја се по подршци великог скупа типова података, као и могућности креирања нових типова од стране корисника. *MySQL* представља релациони систем за управљање базама података који подржава широк скуп *SQL* операција и функционалности, укључујући рад са трансакцијама, индексирање и могућност вишекорисничког опслуживања. *Oracle Database* је релациони систем за управљање базама који користи језик *SQL* за манипулацију над подацима.

Радни оквир *Spring* пружа могућност безбедносне конфигурације веб апликација помоћу библиотеке *Spring Security*. Конфигурација се врши у складу са различитим безбедносним механизмима и на различитим нивоима, а конфигурацију је потребно прилагодити специфичним потребама система. Подржана су три безбедносна механизма: основна аутентификација [17], стандард *JSON* веб токен [18] и стандард *Open Authorization* [19].

2.2.1 Основна аутентификација

Основна аутентификација (енгл. *Basic Authentication*) представља метод у ком се корисник идентификује помоћу корисничког имена и лозинке. При сваком захтеву се у заглављу захтева налазе идентификациони параметри корисника и на основу тога се

потврђује идентитет и право приступа ресурсу или функционалности система. Ови параметри се често прослеђују као обичан текст или се шифрују помоћу неког механизма шифровања, као што је *Base64*. Овај метод се једноставно имплементира и користи, те је погодан за једноставније системе. Лозинке преносе у заглављу захтева, што их чини подложним нападима и може угрозити сигурност апликације, па се самим тим препоручује коришћење додатних безбедносних механизма.

2.2.2 Стандард *JSON* веб токен

JSON веб токен (енгл. *JSON web token, JWT*) представља формат за представу токена за аутентификацију. Састоји се од три дела: заглавље (енгл. *header*), главног дела (енгл. *payload*) и потписа (енгл. *signature*). Токен се генерише при свакој успешној аутентификацији и додељује се пријављеном кориснику, при чему садржи све неопходне податке о њему. При сваком захтеву се проверава валидност *JWT* токена на основу информација из њега и одређује се да ли је кориснику дозвољен приступ ресурсу или функционалности система.

2.2.3 Стандард *Open Authorization*

Стандард *Open Authorization (OAuth)* представља стандард за доделу права приступа који омогућава корисницима да доделе овлашћења апликацијама за приступ њиховим подацима који се налазе у другим апликацијама. Уместо уношења идентификационих параметара, као што су корисничко име и лозинка, сервер за доделу права приступа генерише токен који се користи за приступ ресурсима апликације. Овај стандард користи велики број компанија као што су *Google* и *Facebook*, како би омогућиле корисницима да поделе податке са својих корисничких налога са другим апликацијама. Последња верзија овог механизма је *OAuth2.0*.

3. Преглед постојећег стања у области

Савремене методе развоја софтвера теже брзом, безбедном и ефикасном развоју и смањењу потребе за ручно писаним кодом, што је довело до повећаног броја решења за генерисање различитих видова и нивоа софтвера.

Spring_INITIALIZER [20] је користан алат за брзо креирање основне структуре *Spring* апликација. Омогућава дефинисање основних метаподатака апликације, као и спољних библиотека и њихових верзија. Погодан је за једноставне пројекте, али је неопходна ручна конфигурација безбедносних аспеката, базе података и имплементација осталих слојева апликације.

У раду [21] је описан наменски језик *Silvera* за генерисање апликација са микросервисном архитектуром. Овај наменски језик је развијен како би омогућио једноставнији и бржи развој апликација помоћу језика који је лако читљив. Омогућава аутоматско генерисање документације за апликацију. Поред наменског језика, развијен је компајлер који може да произведе код у било ком програмском језику и радном оквиру, с обзиром да су коришћени спољни генератори.

Алат *MicroBuilder* представљен у раду [22] омогућава аутоматизацију и олакшава процес спецификације и конфигурације микросервисне архитектуре. На основу спецификације помоћу наменског језика *MicroDSL* и развијених генератора, генерише се извршив код у програмском језику *Java*.

JHipster [23] представља алат за развој апликација, укључујући апликације засноване на радном оквиру *Spring*. Омогућава генерисање кода на основу спецификације, укључујући серверски и клијентски део апликације. Подржава развој апликација са микросервисном архитектуром и монолитних апликација и интегрише разне технологије и радне оквире. Подржава безбедносну конфигурацију за неколико безбедносних механизма.

3.1 Недостаци код постојећих начина за моделовање веб апликација

Анализа постојећих решења за моделовање апликација довела је до закључка да ниједно од решења не испуњава све захтеве у потпуности. Постоје решења [20] која су корисна при развоју почетног пројекта, али захтевају ручно писање кода за иницијализацију већине елемената апликације. Нека решења [21,22] нису погодна за генерисање монолитних апликација, иако су се показала ефикасним за развој микросервисне архитектуре. Конфигурација безбедносних аспеката помоћу алата [23] може изискивати додатно време због комплексне синтаксе. Поред овога, генерисани код може да садржи више функционалности него што је захтевано, те обимност кода може довести до смањења перформанси апликације. Због наведеног се поставља питање колико заправо доприноси брзини и ефикасности развоја софтвера

Главни проблем који је уочен код скоро свих анализираних решења је недостатак могућности за конфигурисање безбедносних аспеката апликације. Неопходно је да решење буде довољно ефикасно, лако и брзо како би допринело брзини развоја софтвера, али да садржи довољно елемената како би апликације биле безбедне за коришћење и садржале све неопходне слојеве за правилно функционисање.

4. Наменски језик за подршку брзог успостављања конфигурације безбедносних аспеката у радном оквиру *Spring*

Недостаци описани у претходном поглављу, доводе до потребе за развојем наменског језика који би омогућио брзо успостављање конфигурације безбедносних аспеката. Да би наменски језик омогућио брзу и ефикасну конфигурацију у радном оквиру *Spring*, неопходно је подржати моделовање свих неопходних концепата за иницијализацију веб апликације. Ови концепти сврставају се у пет главних целина. На почетку, параметри који се односе на саму апликацију, односно њене метаподатке, што омогућава брзу конфигурацију апликације. Параметри базе података који омогућавају складиштење и руковање подацима представљају другу целину. Следећа целина обухвата модел структуре података складиштених у бази података. Модел података укључује ентитете који одговарају табелама у бази података. Слој за обраду захтева корисника је целина за себе и омогућава дефинисање контролера. Последња целина представља сигурносни слој који се односи на аутентификацију и контролу приступа корисника. Наменски језик *securaDSL* подржава конфигурацију три безбедносна механизма у радном оквиру *Spring*:

- основну аутентификацију,
- аутентификација помоћу *JWT* токена и
- *OAuth2.0*.

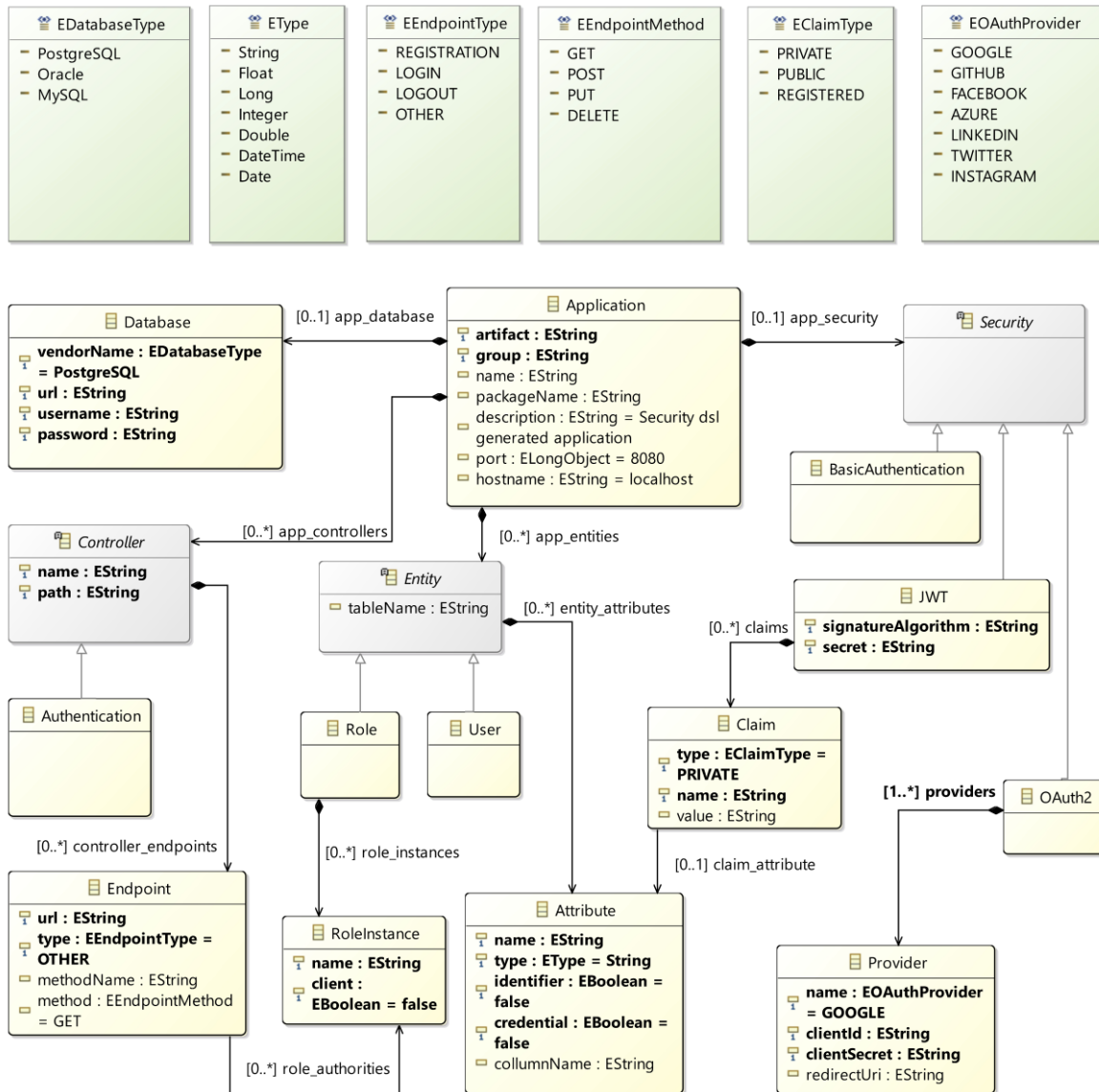
Овим се обезбеђује флексибилност и могућност одабира оптималног безбедносног механизма у зависности од потреба и захтева система.

Иако је примарна намена наменског језика *securaDSL* конфигурација безбедносних аспеката, корисницима је омогућена и конфигурација веб апликације без тог слоја. Претходно наведене целине је могуће комбиновати уз ограничења која ће бити наведена у наставку поглавља. Коришћење *securaDSL*, доменским експертима знатно убрзава и олакшава процес развоја сигурних веб апликација у радном оквиру *Spring*.

4.1 Апстрактна синтакса

Апстрактна синтакса омогућава опис структуре наменског језика *securaDSL* и представљена је помоћу мета-модела (Слика 4.1).

У даљем тексту дат је опис концепата апстрактне синтаксе где су енглески називи концепата који су приказани на Слици 4.1 наведени курзивом.



Слика 4.1 – Апстрактна синтакса наменског језика *securaDSL*

Коренски концепт апстрактне синтаксе је апликација (*Application*) и садржи податке о параметрима неопходним за иницијализацију апликације. Кориснику је остављена и могућност проширења апликације додатним концептима. Могуће је подесити параметре за повезивање са базом података (*Database*), а поред тога, могу се дефинисати ентитети (*Entity*) који се односе на кориснике (*User*) и улоге (*Role*). При дефинисању ентитета, неопходно је навести обележја (*Attribute*) за сваки ентитет. Концепт који се односи на обраду захтева корисника (*Controller*) је повезан са концептом *Endpoint* где је остављена могућност додавања метода које се односе на различите функционалности апликације. Навођењем инстанци улога (*RoleInstance*) које се налазе у систему и повезивањем са одређеним методама омогућена је контрола приступа. Контролер за аутентификацију (*Authentication*) може да садржи методе за регистрацију, пријаву на систем и одјаву са система.

Посебан део апстрактне синтаксе односи се на сигурносни слој (*Security*), где су подржана три безбедносна механизма: основна аутентификација (*BasicAuthentication*), аутентификација заснована на *JWT* токенима (*JWT*) и *OAuth2.0* аутентификација (*OAuth2*). У зависности од жељеног механизма могу се дефинисати додатни параметри описани адекватним концептима.

Сваки од наведених концепата могу се додати или изоставити у зависности од потреба корисника. Оваква апстрактна синтакса омогућава кориснику флексибилност и једноставно проширење генерисане апликације у складу са потребама.

У наставку поглавља описани су сви концепти апстрактне синтаксе наменског језика *securaDSL* уз опис обележја и асоцијација концепата. Поред тога, наведена су и појашњена ограничења имплементирана помоћу декларативног језика *OCL*.

4.1.1 Концепт *Application*

Коренски концепт апстрактне синтаксе *Application* садржи обележја (Табела 4.1) која се односе на основне параметре неопходне за иницијализацију апликације.

Назив обележја	Тип податка	Обавезно	Опис
<i>artifact</i>	<i>EString</i>	да	Назив артефакта, односно концизан назив који јасно описује функционалности апликације
<i>group</i>	<i>EString</i>	да	Назив групе који представља организацију, компанију или тим који развија апликацију
<i>name</i>	<i>EString</i>	не	Назив апликације
<i>packageName</i>	<i>EString</i>	не	Назив пакета у ком се налази изворни код апликације
<i>description</i>	<i>EString</i>	не	Опис апликације
<i>port</i>	<i>ELongObject</i>	не	Порт на ком апликација ослушкује захтеве
<i>hostname</i>	<i>EString</i>	не	Адреса рачунара или сервера на ком се извршава апликација

Табела 4.1 – Обележја концепта *Application*

Портови у опсегу од 1024 до 49151 представљају регистроване портове који се могу доделити апликацији. Ограничење *validRegisteredPort* (Листинг 4.1) гарантује да апликација користи исправан и регистрован порт.

```
invariant validRegisteredPort('Port must be in the valid range of
                                1024 to 49151!'):
    self.port >= 1024 and self.port <= 49151;
```

Листинг 4.1 – Порт апликације мора бити унутар опсега регистрованих портова

Табела 4.2 приказује асоцијације концепта *Application* са другим концептима. Апликација може садржати ентитете, контролере, базу података и аспекте безбедности. Кориснику је остављен простор да наведе или изостави одређене целине. Међутим, навођење неких целина повлачи одређена ограничења. Уколико се наведу ентитет, неопходно је да апликација има дефинисану базу података која складишти податке о тим ентитетима. Ово је омогућено *OC*L ограничењем *hasDatabaseForEntity* (Листинг 4.2).

Назив асоцијације	Референцирани концепт	Кардиналитет	Опис
<i>app_database</i>	<i>Database</i>	0..1	База података апликације
<i>app_entities</i>	<i>Entity</i>	0..*	Ентитети у апликацији
<i>app_controllers</i>	<i>Controller</i>	0..*	Контролери у апликацији
<i>app_security</i>	<i>Security</i>	0..1	Безбедносна конфигурација апликације

Табела 4.2 – Асоцијације концепта *Application*

```
invariant hasDatabaseForEntity('An application must have a database defined if
it has entities!'):
    self.app_entities -> isEmpty() or not self.app_database->isEmpty();
```

Листинг 4.2 – Уколико апликација има дефинисане ентитете, мора да има дефинисану базу података

4.1.2 Концепт Database

Апстрактни концепт *Database* обухвата параметре за повезивање апликације са базом података. Тип података *EDatabaseType* представља енумерацију за назив компаније која је развила жељени систем за управљање базама података. Корисник је дужан да унесе валидне идентификационе параметре и конекциони стринг како би апликација успешно успоставила везу са системом за управљање базом података. Ово омогућава да апликација чита, додаје и ажурира податке сачуване у бази података. Табела 4.3 приказује обележја концепта *Database*.

Назив обележја	Тип податка	Обавезно	Опис
<i>vendorName</i>	<i>EDatabaseType</i>	да	Систем за управљање базама података, при чему су могуће вредности <i>PostgreSQL</i> , <i>MySQL</i> и <i>Oracle</i>
<i>url</i>	<i>EString</i>	да	Конекциони стринг за повезивање са базом података
<i>username</i>	<i>EString</i>	да	Корисничко име за приступ бази података
<i>password</i>	<i>EString</i>	да	Лозинка за приступ бази података

Табела 4.3 – Обележја концепта *Database*

4.1.3 Концепт *Entity*

Ентитети апликације моделовани су помоћу концепта *Entity*. Табела 4.4 приказује обележја овог концепта. Асоцијација ентитета са концептом *Attribute* (Табела 4.5) омогућава да ентитети садрже сва релевантна обележја за домен апликације.

Назив обележја	Тип податка	Обавезно	Опис
<i>tableName</i>	<i>EString</i>	не	Назив табеле у бази података која се односи на ентитет

Табела 4.4 – Обележја концепта *Entity*

Ограничење *uniqueTableName* (Листинг 4.3) гарантује да сви ентитети имају јединствен назив табеле у бази података што спречава потенцијалне проблеме приликом рада са подацима. За ентитете за које није наведен, назив табеле биће изједначен са називом ентитета у множини (нпр. *users* или *roles*).

```
invariant uniqueTableName('Table names must be unique!'):
    Entity.allInstances() ->
        select(e | e.tableName <> null) -> isUnique(e | e.tableName.toLowerCase());
```

Листинг 4.3 – Сви ентитети имају јединствен назив табеле у бази података

Назив асоцијације	Референцирани концепт	Кардиналитет	Опис
<i>entity_attributes</i>	<i>Attribute</i>	0..*	Обележја ентитета

Табела 4.5 – Асоцијације концепта *Entity*

Асоцијација *entity_attributes* моделује придруживање одређених обележја ентитету. Јединственост назива обележја у оквиру ентитета омогућена је ограничењем *uniqueAttributeName* (Листинг 4.4), док јединственост назива колона у табели гарантује ограничење *uniqueColumnName* (Листинг 4.5). Неопходно је да ентитети поседују тачно један идентификатор, што је обезбеђено ограничењем *onlyOneIdentifier* (Листинг 4.6).

```
invariant uniqueAttributeName('Attribute names within an entity must be
unique!'):
    self.entity_attributes -> isUnique(a | a.name);
```

Листинг 4.4 – Сва обележја у оквиру ентитета имају јединствене називе

```
invariant uniqueColumnName('Column names must be unique if defined!'):
    self.entity_attributes -> exists(a | a.columnName <> null) implies
        self.entity_attributes -> isUnique(a | a.columnName);
```

Листинг 4.5 – Сва обележја имају јединствене називе колона унутар табеле у бази података

```
invariant onlyOneIdentifier('Entity must have exactly one identifier
attribute!'):
    self.entity_attributes -> size() > 0 implies
        self.entity_attributes -> select(a | a.identifier) -> size() = 1;
```

Листинг 4.6 – За ентитет мора да постоји тачно једно обележје које је идентификатор

4.1.4 Концепт *Attribute*

Концепт *Attribute* моделује обележја која описују различите карактеристике ентитета. *EType* представља енумерацију за тип податка обележја моделованог концептом. Обележје представља идентификатор ентитета уколико је вредност обележја *identifier* једнака *true*, док је обележје идентификациони параметар уколико је вредност обележја *credential* једнака *true*. Обележја овог концепта приказана су у Табели 4.6.

Назив обележја	Тип податка	Обавезно	Опис
<i>name</i>	<i>EString</i>	да	Назив обележја
<i>type</i>	<i>EType</i>	да	Тип обележја, при чему су могуће вредности <i>String</i> , <i>Float</i> , <i>Long</i> , <i>Integer</i> , <i>Double</i> , <i>DateTime</i> и <i>Date</i>
<i>identifier</i>	<i>EBoolean</i>	да	Да ли је обележје идентификатор?

<i>credential</i>	<i>EBoolean</i>	да	Да ли је обележје идентификациони параметар?
<i>columnName</i>	<i>EString</i>	не	Назив колоне у табели базе података који се односи на обележје

Табела 4.6 – Обележја концепта *Attribute*

4.1.5 Концепт *User*

Приликом генерисања кода, концепт *User* биће искоришћен за генерисање класе које представља ентитет корисника апликације. Наслеђује концепт *Entity*, те садржи обележја која ближе описују корисника. Могуће је постојање највише једне инстанце концепта *User* за исправну конфигурацију апликације (Листинг 4.7).

```
invariant uniqueUserEntity('There can be at most one entity of type "User" in the model!'):
    Entity.allInstances() -> select(e | e.oclIsTypeOf(User)) -> size() <= 1;
```

Листинг 4.7 – Могуће је постојање највише једне инстанце концепта *User*

Обележје које представља лозинку је подразумевано и биће генерисано са остатком кода, те не постоји потреба за експлицитним навођењем истог (Листинг 4.8). Како би аутентификација била омогућена неопходно је да класа *User*, поред постојеће лозинке, поседује још тачно једно обележје које представља идентификациони параметар (нпр. корисничко име). Ради једноставније провере идентификационог параметра, оно мора бити типа *String*. Ово је гарантовано ограничењем *oneStringTypeCredentialForUser* (Листинг 4.9). Увођењем идентификационих параметара, потребно је онемогућити их у класама које не представљају кориснике система (Листинг 4.10).

```
invariant noAttributeNamedPassword('User entity cannot have an attribute named "password"!'):
    self.entity_attributes -> forAll(a | a.name.toLower() <> 'password');
```

Листинг 4.8 – За концепт *User* не сме да постоји обележје које представља лозинку

```
invariant oneStringTypeCredentialForUser('User entity must have exactly one attribute of type String marked as a credential!'):
    self.entity_attributes -> select(a | a.credential) -> size() = 1
    and
    self.entity_attributes ->
        select(a | a.credential) -> forAll(a | a.type = EType::String);
```

Листинг 4.9 – За концепт *User* мора да постоји тачно једно обележје типа *String* који представља идентификациони параметар

```
invariant otherEntitiesDoesntHaveCredential('Entities other than User
cannot have a credential attribute!'):
    not self.oclIsTypeOf(User) implies self.entity_attributes ->
```

Листинг 4.10 – Само концепти *User* могу да поседују обележје које представља идентификациони параметар

4.1.6 Концепт *Role*

Концепт *Role* наслеђује концепт *Entity* и моделује ентитет који се односи на улоге апликације. Улоге могу бити имплементирани као класе са различитим обележјима или као енумерација, што зависи од изабране безбедносне конфигурације у оквиру апликације. Могуће је постојање највише једне инстанце концепта *Role* (Листинг 4.11).

```
invariant uniqueRoleEntity('There can be at most one entity of type "Role" in
the model!'):
    Entity.allInstances() -> select(e | e.oclIsTypeOf(Role)) -> size() <= 1;
```

Листинг 4.11 – Могуће је постојање највише једне инстанце концепта *Role*

Дефинисано је ограничење *uniqueRoleInstanceName* (Листинг 4.12), које осигурава да свака инстанца улоге (Табела 4.7) има уникатан назив. Ово ограничење омогућава избегавање конфликта приликом додавања нових улога.

Назив асоцијације	Референцирани концепт	Кардиналитет	Опис
<i>role_instances</i>	<i>RoleInstance</i>	0..*	Инстанце улога у апликацији

Табела 4.7 – Асоцијације концепта *Role*

```
invariant uniqueRoleInstanceName('Role instance names must be unique!'):
    self.role_instances -> isUnique(r | r.name);
```

Листинг 4.12 – Инстанце улога морају имати јединствен назив

4.1.7 Концепт *RoleInstance*

Инстанце концепта који моделује улоге, било да је у питању енумерација или ентитет са обележјима, моделоване су помоћу концепта *RoleInstance* (Табела 4.8). Представљају улоге које је могуће доделити корисницима система како би била омогућена контрола приступа ресурсима апликације.

Назив обележја	Тип податка	Обавезно	Опис
<i>name</i>	<i>EString</i>	да	Назив конкретне улоге

<i>client</i>	<i>EBoolean</i>	да	Ознака да ли је улога клијент. Уколико је вредност обележја <i>false</i> улога је администраторска
---------------	-----------------	----	---

Табела 4.8 – Обележја концепта *RoleInstance*

4.1.8 Концепт *Endpoint*

Endpoint је концепт који представља тачку комуникације између корисника и апликације, што га чини битним елементом за контролу приступа, односно саму безбедност апликације. Корисници комуницирају са сервером путем тачки комуникације тако што шаљу захтеве за извршавање одређених функционалности. Обележја концепта *Endpoint* су наведена у Табели 4.9. Асоцијација *role_authorities* (Табела 4.10) описује улоге које имају дозвољен приступ методи, односно улоге које имају овлашћење да приступе одређеној функционалности система.

Назив обележја	Тип податка	Обавезно	Опис
<i>url</i>	<i>EString</i>	да	Путања, односно <i>URL</i> адреса методе
<i>type</i>	<i>EEndpointType</i>	да	Тип методе, где су могуће вредности регистрација (<i>REGISTRATION</i>), пријава (<i>LOGIN</i>), одјава (<i>LOGOUT</i>) и друго (<i>OTHER</i>)
<i>methodName</i>	<i>EString</i>	да	Назив методе
<i>method</i>	<i>EEndpointMethod</i>	не	<i>HTTP</i> метод који означава каква је врста захтева, при чему су могуће вредности <i>GET</i> , <i>POST</i> , <i>PUT</i> и <i>DELETE</i>

Табела 4.9 – Обележја концепта *Endpoint*

Назив асоцијације	Референцирани концепт	Кардиналитет	Опис
<i>role_authorities</i>	<i>RoleInstance</i>	0..*	Улоге које имају право приступа методи

Табела 4.10 – Асоцијације концепта *Endpoint*

Ограничење *urlStartsWithForwardSlash* (Листинг 4.13) гарантује да путање метода започињу карактером '/' што доприноси конзистентности у апликацији. Поред овог ограничења, битно је да улоге којима је дозвољен приступ буду јединствене у оквиру методе (Листинг 4.14).

```
invariant urlStartsWithForwardSlash('Endpoint URL should start with a forward slash!'): self.url.at(1) = '/';
```

Листинг 4.13 – Путање метода започињу карактером '/'

```
invariant uniqueRoleAuthorities('Role authorities must be unique for each endpoint!'): self.role_authorities -> isUnique(r | r.name);
```

Листинг 4.14 – Улоге које имају дозволу приступа методи не могу да се дуплирају унутар исте

4.1.9 Концепт *Controller*

Концепт *Controller* описује контролере апликације (Табела 4.11). Садржи информације о називу и путањи контролера, при чему је неопходно да обе вредности буду јединствене унутар апликације (Листинг 4.15 и Листинг 4.16). Уобичајено је да се називи контролера разликују од путања како би се избегли конфликти приликом рутирања захтева што је гарантовано ограничењем *uniqueControllerPath* (Листинг 4.16). Путања контролера представља апсолутну путању у оквиру апликације, те је неопходно да почиње карактером '/'. Ово такође омогућава конзистентност генерисаног кода (Листинг 4.17). Ентитети који се односе на кориснике (*User*) и улоге (*Role*) доводе до постојања класа са истим називима, па самим тим постоји ограничење назива контролера (Листинг 4.18).

Назив обележја	Тип податка	Обавезно	Опис
<i>name</i>	<i>EString</i>	да	Назив контролера
<i>path</i>	<i>EString</i>	да	Путања контролера

Табела 4.11 – Обележја концепта *Controller*

```
invariant uniqueControllerName('Controllers should have unique names!'): Controller.allInstances() -> isUnique(c | c.name);
```

Листинг 4.15 – Називи контролера унутар апликације морају да буду јединствени

```

invariant uniqueControllerPath('Controller paths should be unique and
different from names!'):
    Controller.allInstances() -> isUnique(c | c.path)
and
    Controller.allInstances() ->
        forAll(c | '/' + c.name.toLower() <> c.path.toLower());

```

Листинг 4.16 - Путање контролера унутар апликације морају да буду јединствене и да се разлику од назива контролера

```

invariant controllerPath('Controller path should start with '/'!'):
    self.path.at(1) = '/';

```

Листинг 4.17 – Путања контролера мора да почиње карактером '/'

```

invariant controllerNotNamedUserRole('Controller names cannot be "User" or
"Role"!'):
    not Controller.allInstances() ->
        exists(c | c.name.toLower() = 'user' or c.name.toLower() = 'role');

```

Листинг 4.18 – Ограничење назива за контролере

За правилно рутирање захтева и рад апликације неопходна је јединственост назива (Листинг 4.19) и путања (Листинг 4.20) метода унутар контролера (Табела 4.12). Методе за регистрацију, пријаву и одјаву са система имају препоручене, унапред дефинисане *HTTP* методе: *POST*, *POST* и *GET* редом. За методе типа *OTHER* неопходно је навести *HTTP* метод (Листинг 4.21). Регистрација, пријава и одјава са система су функционалности контролера за аутентификацију, те се не могу наћи у другим контролерима (Листинг 4.22).

Назив асоцијације	Референцирани концепт	Кардиналитет	Опис
<i>controller_endpoints</i>	<i>Endpoint</i>	0..*	Методе контролера

Табела 4.12 – Асоцијације концепта *Controller*

```

invariant uniqueEndpointMethodNames('Endpoints within a controller must have
unique method names!'):
    self.controller_endpoints -> isUnique(e | e.methodName);

```

Листинг 4.19 – Методе унутар контролера морају имају јединствене називе

```

invariant uniqueEndpointURLs('Endpoints within a controller must have unique
URLs!'):
    self.controller_endpoints -> isUnique(e | e.url);

```

Листинг 4.20 – Методе унутар контролера морају имају јединствене путање

```

invariant methodRequiredForOtherType('Endpoints of type "OTHER" must have a
defined method!'):
    self.controller_endpoints -> select(e | e.type = EEndpointType::OTHER)
-> forAll(e | e.method <> null);

```

Листинг 4.21 – Методе типа *OTHER* морају да имају дефинисан *HTTP* метод

```

invariant endpointLimits('Controllers of type other than "Authentication" should
not have registration, login, or logout endpoints!'):
    not self.oclIsTypeOf(Authentication) implies (
        self.controller_endpoints ->
        select(e | e.type = EEndpointType::REGISTRATION) -> isEmpty()
        and
        self.controller_endpoints ->
        select(e | e.type = EEndpointType::LOGIN) -> isEmpty()
        and
        self.controller_endpoints ->
        select(e | e.type = EEndpointType::LOGOUT) -> isEmpty());

```

Листинг 4.22 – Методе за регистрацију, пријаву и одјаву са система могу да се налазе само у контролеру за аутентификацију

4.1.10 Концепт *Authentication*

Концепт *Authentication* наслеђује концепт *Controller*. Контролер за аутентификацију обрађује захтеве који се односе на регистрацију корисника, пријаву и одјаву корисника са система, те су информације о корисницима и њиховим улогама неопходне. Самим тим, уколико постоји контролер за аутентификацију, неопходно је да постоје инстанце концепата *User* и *Role* (Листинг 4.23). Могуће је постојање највише једног контролера за аутентификацију (Листинг 4.24). Уколико контролер за аутентификацију постоји, дозвољено је постојање највише једне методе за регистрацију, пријаву и одјаву са система што спречава вишеструко дефинисање истих функционалности (Листинг 4.25).

```

invariant hasUserAndRoleForController ('Authentication controller requires at
least one User entity and one Role entity!'):
    Entity.allInstances() -> exists(e | e.oclIsTypeOf(User))
    and
    Entity.allInstances() -> exists(e | e.oclIsTypeOf(Role));

```

Листинг 4.23 – За дефинисан контролер за аутентификацију, неопходно је постојање инстанци концепата *User* и *Role*

```

invariant uniqueAuthenticationController('There can be at most one controller
of type "Authentication" in the model!'):
    Controller.allInstances() ->
    select(c | c.oclIsTypeOf(Authentication)) -> size() <= 1;

```

Листинг 4.24 – Могуће је постојање највише једног контролера за аутентификацију

```

invariant authenticationLimits('Authentication can have at most one
registration, login, and logout endpoint!'):
    self.controller_endpoints ->
        select(e | e.type = EEndpointType::REGISTRATION) -> size() <= 1
    and
    self.controller_endpoints ->
        select(e | e.type = EEndpointType::LOGIN) -> size() <= 1
    and
    self.controller_endpoints ->
        select(e | e.type = EEndpointType::LOGOUT) -> size() <= 1;

```

Листинг 4.25 – Могуће је постојање највише једне методе за регистрацију, пријаву и одјаву са система

4.1.11 Концепт *Security*

Концепт *Security* представља важну апстракцију у моделу која је кључна за обезбеђивање сигурне апликације. Овај концепт омогућава имплементацију жељеног нивоа аутентификације и ауторизације, што представља основу ефикасне заштите података од потенцијалних напада и злоупотребе.

4.1.12 Концепт *BasicAuthentication*

Концепт *BasicAuthentication* наслеђује *Security* и моделује основну безбедносну конфигурацију. У случају одабира основне аутентификације, улоге ће у апликацији бити представљене као енумерација. Самим тим нису дозвољена обележја, већ само инстанце улога (Листинг 4.26).

```

invariant basicAuthNoRoleAttributes('Basic authentication cannot have role
attributes!'):
    Entity.allInstances() -> select(e | e.oclIsTypeOf(Role))
        > forAll(role | role.entity_attributes -> size() = 0)

```

Листинг 4.26 – У случају основне аутентификације нису дозвољена придружена обележја концепту *Role*

4.1.13 Концепт *JWT*

Аутентификацију помоћу *JWT* токена моделована је концептом *JWT* који наслеђује концепт *Security*. Обележја овог концепта су приказана у Табели 4.13. *JWT* токени садрже тврдње (Табела 4.14) које ће бити описане у даљем тексту.

Назив обележја	Тип податка	Обавезно	Опис
<i>signatureAlgorithm</i>	<i>EString</i>	да	Алгоритам који се користи за потписивање <i>JWT</i> токена
<i>secret</i>	<i>EString</i>	да	Тајни кључ који се користи за потписивање <i>JWT</i> токена

Табела 4.13 – Обележја концепта *JWT*

Назив асоцијације	Референцирани концепт	Кардиналитет	Опис
<i>claims</i>	<i>Claim</i>	0..*	Тврдње у оквиру <i>JWT</i> токена

Табела 4.14 – Асоцијације концепта *JWT*

За аутентификацију помоћу токена *JWT*, улоге представљају класу са обележјима. Како би инстанце улога биле омогућене, неопходно је да у случају одабира аутентификације на основу *JWT* токена за класу *Role* постоји тачно једно обележје које је типа *String*. Поред тога, дозвољено је само обележје које представља идентификатор. У случају да је идентификатор типа *String*, нису дозвољена додатна обележја. Ово је гарантовано обележјима *roleHasMaxTwoAttributes* и *roleHasStringAttribute* (Листинг 4.27).

```
invariant roleHasMaxTwoAttributes ('Role entities can have at most two
attributes!'):
    Entity.allInstances() -> select(e | e.ocIsTypeOf(Role)) ->
        forAll(role | role.entity_attributes -> size() <= 2);

invariant roleHasStringAttribute('Role entities must have either one identifier
attribute of type String or both identifier and non-identifier attributes of
type String!'):
    Entity.allInstances() -> select(e | e.ocIsTypeOf(Role)) ->
        forAll(role | (role.entity_attributes -> select(a | a.identifier
            and a.type = EType::_'String') -> size() = 1
            and role.entity_attributes ->
                select(a | a.type = EType::_'String') -> size() = 1)
            or (role.entity_attributes -> select(a | a.identifier
                and a.type <> EType::_'String') -> size() = 1
                and role.entity_attributes ->
                    select(a | a.type = EType::_'String') -> size() = 1)));
```

Листинг 4.27 – У случају аутентификације помоћу *JWT* токена, улога може да има тачно једно обележје типа *String*

4.1.14 Концепт *Claim*

Концепт *Claim* представља тврдње које садржи *JWT* токен (Табела 4.15), односно податке о кориснику, стању апликације или самом токenu које се преносе путем токена. Постоје три типа тврдњи: регистроване (предефинисане), приватне и јавне. Поред типа, тврдња има назив који представља идентификатор податка и вредност. Уколико се тврдња односи на податке о кориснику, она је повезана са обележјем које садржи њену вредност (Табела 4.16). Све тврдње морају имати јединствен назив (Листинг 4.28) и за једно обележје може бити везана највише једна тврдња (Листинг 4.29)

Назив обележја	Тип податка	Обавезно	Опис
<i>type</i>	<i>EClaimType</i>	да	Тип тврдње, при чему су могуће вредности <i>PRIVATE</i> , <i>PUBLIC</i> и <i>REGISTERED</i>
<i>name</i>	<i>EString</i>	да	Назив тврдње
<i>value</i>	<i>EString</i>	не	Вредност тврдње

Табела 4.15 – Обележја концепта *Claim*

Назив асоцијације	Референцирани концепт	Кардиналитет	Опис
<i>claim_attributes</i>	<i>Attribute</i>	0..1	Обележје на које се тврдња односи и који садржи додатне информације о њој

Табела 4.16 – Асоцијације концепта *Claim*

```
invariant uniqueClaimNames('Claims must have unique names'):
    Claim.allInstances() -> isUnique(c | c.name);
```

Листинг 4.28 – Тврдње морају имати јединствен назив

```
invariant uniqueClaimAttribute('Claim attributes must be unique!'):
    Claim.allInstances() -> select(c | c.claim_attribute <> null) ->
        isUnique(c | c.claim_attribute);
```

Листинг 4.29 – Обележје може бити повезано са највише једном тврдњом

Основне регистроване, односно предефинисане тврдње су време важења токена након ког он више није валидан (*expirationTime*), корисници или субјекти којима је токен намењен (*audience*), издавач токена (*issuer*) и идентификатор корисника (*subject*) те оне морају бити типа *REGISTERED*. Поред тога, тврдње *expirationTime* и *audience* су обавезне јер су неопходне за аутентификацију путем токена (Листинг 4.30). С обзиром да *expirationTime*, *audience* и *issuer* не садрже податке о кориснику, оне не могу бити повезане са обележјем корисника, већ морају имати дефинисану вредност (Листинг 4.31), док остале тврдње морају бити повезане са неким обележјем (4.32). Тврдња везана за време важења токена мора да има позитивну вредност (Листинг 4.33).

```

invariant subjectRegisteredClaim('If claim name is "subject", it must be of
type REGISTERED'):
    Claim.allInstances() -> select(c | c.name = 'subject') ->
        forAll(sc | sc.type = EClaimType::REGISTERED);

invariant issuerRegisteredClaim('If claim name is "issuer", it must be of type
REGISTERED'):
    Claim.allInstances() -> select(c | c.name = 'issuer') ->
        forAll(sc | sc.type = EClaimType::REGISTERED);

invariant hasExpirationTimeClaim('The claim "expirationTime" must exist and
be of type REGISTERED'):
    Claim.allInstances() ->
exists(c | c.name = 'expirationTime' and c.type = EClaimType::REGISTERED);

invariant hasAudienceClaim('The claim "audience" must exist and be of type
REGISTERED'):
    Claim.allInstances() ->
exists(c | c.name = 'audience' and c.type = EClaimType::REGISTERED);

```

Листинг 4.30 – Основне регистроване тврдње морају бити типа *REGISTERED* и тврдње *expirationTime* и *audience* су обавезне

```

invariant issuerValueNotNull('If the claim name is "issuer", it must have value
and must not be linked to an attribute'):
    Claim.allInstances() -> select(c | c.name = 'issuer') ->
        forAll(sc | sc.value <> null and sc.claim_attribute = null);

invariant audienceValueNotNull('If the claim name is "audience", it must have
value and must not be linked to an attribute'):
    Claim.allInstances() -> select(c | c.name = 'audience') ->
        forAll(sc | sc.value <> null and sc.claim_attribute = null);

invariant expirationTimeValueNotNull('If the claim name is "expirationTime", it
must have value and must not be linked to an attribute'):
    Claim.allInstances() -> select(c | c.name = 'expirationTime') ->
        forAll(sc | sc.value <> null and sc.claim_attribute = null);

```

Листинг 4.31 – Тврдње *issuer*, *audience* и *expirationTime* морају имати дефинисану вредност и не могу бити везане за обележје

```

invariant otherClaimsNoValue('Claim must be linked to an attribute'):
    Claim.allInstances() -> select(c | c.name <> 'issuer' and c.name <>
        'audience' and c.name <> 'expirationTime') ->
        forAll(sc | sc.value = null and sc.claim_attribute <> null);

```

Листинг 4.32 – Тврдње које нису *issuer*, *audience* и *expirationTime* не могу имати дефинисану вредност и морају бити везане за обележје ентитета *User*

```
invariant expirationTimeValueIsPositiveNumeric('If the claim name is
"expirationTime", the value must be a positive number'):
    Claim.allInstances() -> select(c | c.name = 'expirationTime') ->
        forAll(sc | sc.value <> null and sc.value.toInteger() > 0);
```

Листинг 4.33 – Тврдња везана за време важења токена мора да има позитивну вредност

4.1.15 Концепт OAuth2

OAuth2 концепт моделује конфигурацију безбедносног механизма OAuth2.0 и омогућава корисницима пријаву на систем посредством провајдера (Табела 4.17). У раду је подржана само пријава на систем за овај безбедносни механизам, те се подаци о корисницима не складиште у бази података апликације, већ у бази одабраног провајдера. Због тога су креирана ограничења која гарантују да неће бити дефинисани ентитети за кориснике и улоге, као ни контролер за аутентификацију (Листинг 4.34). Неопходно је да сваки тип провајдера буде конфигурисан највише једном, односно да назив провајдера буде јединствен (Листинг 4.35).

Назив асоцијације	Референцирани концепт	Кардиналитет	Опис
<i>providers</i>	<i>Provider</i>	1..*	Конфигурисани провајдери у апликацији

Табела 4.17 – Асоцијације концепта OAuth2

```
invariant doesn'tHaveUserForOauth('OAuth2 authentication requires no User
entities!'):
    Entity.allInstances() -> select(e | e.oclIsTypeOf(User)) -> size() = 0;

invariant doesn'tHaveRoleForOauth('OAuth2 authentication requires no Role
entities!'):
    Entity.allInstances() -> select(e | e.oclIsTypeOf(Role)) -> size() = 0;

invariant doesn'tHaveAuthControllerForOauth('OAuth2 authentication requires no
Authentication controller!'):
    Controller.allInstances() ->
        select(e | e.oclIsTypeOf(Authentication)) -> size() = 0;
```

Листинг 4.34 – Није могуће дефинисати ентитете за кориснике и улоге, као ни контролер за аутентификацију

```
invariant uniqueProviders('Providers must have unique names!'):
    self.providers -> isUnique(p | p.name);
```

Листинг 4.35 – Назив провајдера је јединствен

4.1.16 Концепт *Provider*

Концепт *Provider* (Табела 4.18) описује компоненту *OAuth2.0* протокола који представља ентитет који пружа услуге аутентификације, односно проверава идентификационе параметре које корисник уноси и утврђује њихову исправност. Провајдери могу бити различити, најчешће су то друштвене мреже, пружаоци услуга електронске поште итд.

Назив обележја	Тип податка	Обавезно	Опис
<i>name</i>	<i>EString</i>	да	Назив провајдера
<i>clientId</i>	<i>EString</i>	да	Идентификациони број клијента додељен од стране провајдера
<i>clientSecret</i>	<i>EString</i>	да	Тајни кључ клијента додељен од стране провајдера за сигурносну контролу приликом комуникације са провајдером
<i>redirectUri</i>	<i>EString</i>	не	Адреса за преусмеравање након успешне аутентификације

Табела 4.18 – Обележја концепта *Provider*

4.2 Конкретна синтакса

Коришћењем радног оквира *Xtext*, на основу мета-модела генерисана је почетна верзија текстуалне конкретне синтаксе, односно граматике која описује текстуалну репрезентацију наменског језика. Ова граматика прилагођена је домену, како би била лако читљива и интуитивна за развојне тимове којима је *securaDSL* намењен.

Ово поглавље обухватиће увид у структуру конкретне синтаксе и опис како су концепти мета-модела преведени у текстуалне елементе језика.

4.2.1 Граматика наменског језика *securaDSL*

Иницијална граматика наменског језика *securaDSL*, креирана на основу мета-модела помоћу радног оквира *Xtext*, креће од коренског концепта *Application*. Листинг 4.36 приказује почетно правило за опис коренског концепта. Примећено је да почетна верзија граматике подсећа на уобичајене синтаксе за дефинисање разних конфигурационих фајлова, те таква граматика не захтева велике измене узимајући у обзир да циљну групу чине експерти у пољу безбедносне конфигурације.

Ради прегледности, уведене су мале измене: уклањање витичастих заграда са одређених места, додавање двотачке након назива обележја а пре дефинисања вредности обележја, коришћења угластих заграда за обележавање листи итд. Ново правило за опис коренског концепта *Application* (Листинг 4.37) прилагођено је тако да буде уредније и прегледније, иако није дошло до значајних измена. Оваква ажурирана граматика омогућава једноставно дефинисање различитих аспеката безбедносне конфигурације, као и свих осталих концепата наменског језика. У остатку поглавља приказана су правила за опис свих концепата наменског језика *securaDSL*, укључујући и енумерације.

```
Application returns Application:
{Application}
'Application'
name=EString
'{'
    'artifact' artifact=EString
    ('name' = name=EString)?
    'group' group=EString
    ('packageName' packageName=EString)?
    ('description' description=EString)?
    ('port' port=ELongObject)?
    ('hostname' hostname=EString)?
    ('app_database' app_database=Database)?
    ('app_entities' '{' app_entities+=Entity
                        ( "," app_entities+=Entity)* '}' )?
    ('app_controllers' '{' app_controllers+=Controller
                        ( "," app_controllers+=Controller)* '}' )?
    ('app_security' app_security=Security)?
'}';
```

Листинг 4.36 – Иницијално правило за опис концепта *Application*

```
Application returns Application:
{Application}
'application:'
'artifact:' artifact=EString
('name:' name=EString)?
'group:' group=EString
('packageName:' packageName=EString)?
('description:' description=EString)?
('port:' port=ELongObject)?
('hostname:' hostname=EString)?
('database:' app_database=Database)?
('model:'
    ('user:' app_entities+=User)?
    ('role:' app_entities+=Role)?
)?
('security:' app_security=Security )?

('controller:'
    ('auth:' app_controllers+=Authentication)?
)?;
```

Листинг 4.37 – Правило за опис концепта *Application*

```

Database returns Database:
    'vendor:' vendorName=EDatabaseType
    'url:' url=EString
    'username:' username=EString
    'password:' password=EString;

enum EDatabaseType returns EDatabaseType:
    PostgreSQL = 'PostgreSQL' | Oracle = 'Oracle' | MySQL = 'MySQL';

```

Листинг 4.38 – Правило за опис концепта *Database*

```

Attribute returns Attribute:
    '{'
    (identifier?='identifier')?
    (credential?='credential')?
    'name:' name=EString
    'type:' type=EType

    ('columnName:' columnName=EString)? '}'

enum EType returns EType:
    String = 'String' |
    Float = 'Float' |
    Long = 'Long' |
    Integer = 'Integer' |
    Double = 'Double' |
    DateTime = 'DateTime' |
    Date = 'Date';

```

Листинг 4.39 – Правило за опис концепта *Attribute*

```

Role returns Role:
    {Role}
    ('tableName:' tableName=EString)?
    ('attributes:'
    '['entity_attributes+=Attribute ("," entity_attributes+=Attribute)*']')?
    ('roles:'
    '['role_instances+=RoleInstance ("," role_instances+=RoleInstance)*']')?;

```

Листинг 4.40 – Правило за опис концепта *Role*

```

RoleInstance returns RoleInstance:
    {RoleInstance}
    (client?='client')? name=EString;

```

Листинг 4.41 – Правило за опис концепта *RoleInstance*

```

User returns User:
    {User}
    ('tableName:' tableName=EString)?
    ('attributes:'
    '['entity_attributes+=Attribute ("," entity_attributes+=Attribute)*']')?;

```

Листинг 4.42 – Правило за опис концепта *User*

```

Endpoint returns Endpoint:
    '{'
        'type:' type=EEndpointType
        'url:' url=EString
        'methodName:' methodName=EString
        ('method:' method=EEndpointMethod)?
        ('roleAuthorities:'
            '['role_authorities+=[RoleInstance|EString]
                ( "," role_authorities+=[RoleInstance|EString])* ']' )?
        '}'

enum EEndpointType returns EEndpointType:
    REGISTRATION = 'REGISTRATION' |
    LOGIN = 'LOGIN' |
    LOGOUT = 'LOGOUT' |
    OTHER = 'OTHER';

enum EEndpointMethod returns EEndpointMethod:
    GET = 'GET' | POST = 'POST' | PUT = 'PUT' | DELETE = 'DELETE';

```

Листинг 4.43 – Правило за опис концепта *Endpoint*

```

Authentication returns Authentication:
    {Authentication}
    'name:' name=EString
    'path:' path=EString
    ('endpoints:'
        '['controller_endpoints+=Endpoint
            ( "," controller_endpoints+=Endpoint)* ']' )?;

```

Листинг 4.44 – Правило за опис концепта *Authentication*

```

BasicAuthentication returns BasicAuthentication:
    {BasicAuthentication} 'basicAuthentication';

```

Листинг 4.45 – Правило за опис концепта *BasicAuthentication*

```

JWT returns JWT:
    'jwt:'
    'signatureAlgorithm:' signatureAlgorithm=EString
    'secret:' secret=EString
    'claims:' '[' claims+=Claim ( "," claims+=Claim)* '];

```

Листинг 4.46 – Правило за опис концепта *JWT*

```

Claim returns Claim:
    '{'
        type=EClaimType', '
        name=EString ':'
        (value=EString)?
        ('attribute' claim_attribute=[Attribute|EString])?
    '}'

enum EClaimType returns EClaimType:
    PRIVATE = 'PRIVATE' | PUBLIC = 'PUBLIC' | REGISTERED = 'REGISTERED';

```

Листинг 4.46 – Правило за опис концепта *Claim*

```
OAuth2 returns OAuth2:
  'OAuth2.0:'
  'providers:' '[' providers+=Provider ( "," providers+=Provider)* '];
```

Листинг 4.47 – Правило за опис концепта *OAuth2*

```
Provider returns Provider:
  '{'
    'name:' name=EString ','
    'clientId:' clientId=EString ','
    'clientSecret:' clientSecret=EString
    (',' 'redirectUri:' redirectUri=EString)?
  '}'
;
```

Листинг 4.48 – Правило за опис концепта *Provider*

4.3 Примери модела описаних наменским језиком *securaDSL*

Ово поглавље пружа преглед претходно описане конкретне синтаксе кроз примере за сваки од подржаних безбедносних механизма. Кључне речи, називи обележја и вредности енумерација приказане су бордо бојом, вредности текстуалних обележја представљене су плавом, док су нумеричке вредности приказане сивом бојом. За сваки од примера, коренски концепт *Application* дефинисан је након кључне речи *application* навођењем вредности обележја овог концепта. Параметри за конфигурацију базе података дефинишу се након кључне речи *database*, ентитети након кључне речи *entity*, контролери након кључне речи *controller*, а безбедносни аспекти након кључне речи *security*.

4.3.1 Пример модела *Spring* веб апликације са конфигурисаном основном аутентификацијом

Концепт *Basic Authentication* не захтева додатан опис конфигурације. Битно је дефинисати једно обележје ентитета *User* које представља идентификациони параметар навођењем кључне речи *credential*. Инстанце улога наводе се након кључне речи *roles* у оквиру концепта *Role*. Уколико је инстанца клијент, наводи се кључна реч *client*.

```
application:
  artifact: "securaDSL"
  group: "uns.ftn"
  description: "This is an app that is generated with security DSL"
  port: 8080
  hostname: "localhost"

database:
  vendor: PostgreSQL
  url: "localhost:5432/securaDSL"
  username: "securaDSL"
```



```

password: "securaDSL"

entity:
  user:
    tableName: "users"
    attributes: [
      {
        identifier
        name: "id"
        type: Long
      },
      {
        credential
        name: "username"
        type: String
        collumnName: "username"
      },
      {
        name: "firstName"
        type: String
        collumnName: "first_name"
      },
      {
        name: "lastName"
        type: String
        collumnName: "last_name"
      }
    ]

  role:
    roles: ["admin", client "user"]

security:
  basicAuthentication

controller:
  auth:
    name: "AuthController"
    path: "/auth"
    endpoints: [
      {
        type: REGISTRATION
        url: "/registration"
        methodName: "registration"
      },
      {
        type: LOGIN
        url: "/login"
        methodName: "login"
      },
      {
        type: LOGOUT
        url: "/logout"
        methodName: "logout"
      }
    ]

```

Листинг 4.49 – Пример модела веб апликације са конфигурисаном основном аутентификацијом у радном оквиру *Spring*

4.3.2 Пример модела *Spring* веб апликације са конфигурисаним безбедносним механизмом *JWT*

Након кључне речи *jwt* дефинишу се неопходни параметри за аутентификацију помоћу JWT токена. Пре навођења назива и вредности тврдње, потребно је нагласити ког је она типа, а уколико је тврдња везана за обележје, неопходно је навести кључну реч *attribute*.

```
application:
  artifact: "securaDSL"
  group: "uns.ftn"
  description: "This is an app that is generated with security DSL"
  port: 8080
  hostname: "localhost"

database:
  vendor: PostgreSQL
  url: "localhost:5432/securaDSL"
  username: "securaDSL"
  password: "securaDSL"

entity:
  user:
    tableName: "users"
    attributes: [
      {
        identifier
        name: "id"
        type: Long
      },
      {
        credential
        name: "username"
        type: String
        columnName: "username"
      },
      {
        name: "firstName"
        type: String
        columnName: "first_name"
      },
      {
        name: "lastName"
        type: String
        columnName: "last_name"
      }
    ]

  role:
    attributes:
      [
        {
          identifier
          name: "id"
          type: Long
        },

```

```

        {
            name: "name"
            type: String
        }
    ]
    roles: ["admin", client "user"]

    security:
        jwt:
            signatureAlgorithm: "HS512"
            secret: "somesecret"
            claims:
                [
                    {REGISTERED , subject : attribute username},
                    {REGISTERED , audience : "AUDIENCE_WEB"},
                    {REGISTERED , expirationTime : "18000"},
                    {REGISTERED , issuer : "securaDSL"},
                    {PUBLIC , firstName : attribute firstName}
                ]

        controller:
            auth:
                name: "AuthController"
                path: "/auth"
                endpoints: [
                    {
                        type: REGISTRATION
                        url: "/registration"
                        methodName: "registration"
                    },
                    {
                        type: LOGIN
                        url: "/login"
                        methodName: "login"
                    },
                    {
                        type: LOGOUT
                        url: "/logout"
                        methodName: "logout"
                    }
                ]
    ]

```

Листинг 4.50 – Пример модела веб апликације са конфигурисаним безбедносним механизмом *JWT* у радном оквиру *Spring*

4.3.3 Пример модела *Spring* веб апликације са конфигурисаним безбедносним механизмом *OAuth2*

Придржавањем *OCL* ограничења, за безбедносну конфигурацију *OAuth2.0* механизма, нису дефинисани концепти *User*, *Role* и *Authentication*. При дефинисању концепта *OAuth2.0*, након кључне речи *providers* наведени су провајдери са свим неопходним обележјима.

```

application:
  artifact: "securaDSL"
  group: "uns.ftn"
  description: "This is an app that is generated with security DSL"
  port: 8080
  hostname: "localhost"

database:
  vendor: PostgreSQL
  url: "localhost:5432/securaDSL"
  username: "securaDSL"
  password: "securaDSL"

security:
  OAuth2.0:
    providers:[
      {
        name: "google",
        clientId: "x",
        clientSecret: "x"
      },
      {
        name: "github",
        clientId: "x",
        clientSecret: "x"
      }
    ]

```

Листинг 4.51 – Пример модела веб апликације са конфигурисаним безбедносним механизмом *OAuth2.0* у радном оквиру *Spring*

5. Генерисање *Spring* веб апликација са безбедносном конфигурацијом

Обзиром да се *Spring* апликација може поделити у целине које су претходно описане и да постоји више подржаних безбедносних механизма, ради прегледности је развијено више генератора. Развијени су следећи генератори:

- генератор статичких датотека,
- генератор општих конфигурационих фајлова,
- генератор слоја који моделује податке из базе података,
- генератор слоја за обраду захтева корисника,
- генератор конфигурационих фајлова за основну аутентификацију,
- генератор конфигурационих фајлова за стандард *JWT* и
- генератор конфигурационих фајлова за стандард *OAuth2.0*.

На основу података из модела *Spring* веб апликације описаног у претходном поглављу, генератори формирају излаз чиме се генерише део по део *Spring* веб апликације са конфигурисаним одабраним безбедносним механизмом. Који генератори ће генерисати излазни код зависи од тога који су концепти дефинисани у моделу. Нпр. уколико је дефинисана основна аутентификација, генератори конфигурационих фајлова за стандарде *JWT* и *OAuth2.0* неће имати излаз.

За конфигурацију сваког од безбедносних механизма, неопходно је генерисати класу у којој је описано на који начин ће се потврђивати идентитет корисника (основна аутентификација, *JWT* или *OAuth2.0*), који корисници имају приступ одређеним деловима апликације, функционалности и ресурси којима је приступ дозвољен без успешне аутентификације и догађаји након неуспешне аутентификације. Ова класа је названа *SecurityConfig*.

У наставку поглавља описани су генератори и дати су примери шаблона и генерисаног кода. Приказани примери су генерисани на сонову модела описаних у поглављу 4.3. Концепт *Application* је дефинисан исто за сваки пример, те неће бити наглашено на који се пример односи.

5.1 Генератор статичких датотека

Статичке датотеке представљају датотеке за чије генерисање нису потребни параметри из модела или су потребни само делови метаподатака апликације. С обзиром да се конфигурација *Spring Boot* апликација врши помоћу алата *Maven*, генерисање датотека *MavenWrapperDownloader.java*, *maven-wrapper.properties*, *mvnw* и *mvnw.cmd* осигурава конзистентно окружење на различитим уређајима.

Поред овога, важно је да се креирају класе помоћу које се врши иницијализација апликације. Ово укључује главну класу у којој се налази главна метода (енгл. *main*) апликације која представља почетну тачку извршавања. Ова метода служи за

иницијализацију апликације и конфигурацију свих компоненти. За покретање тестова који се могу додатно имплементирати, креира се главна метода за тестове. Листинг 5.1 приказује шаблон за генерисање главне класе апликације, док је на Листингу 5.2 приказан шаблон за генерисање главне класе за тестове. На Слици 5.1 приказан је пример главне класе генерисане на основу шаблона приказаног у Листингу 5.1. Слика 5.2 приказује пример генерисане главне класе за тестове.

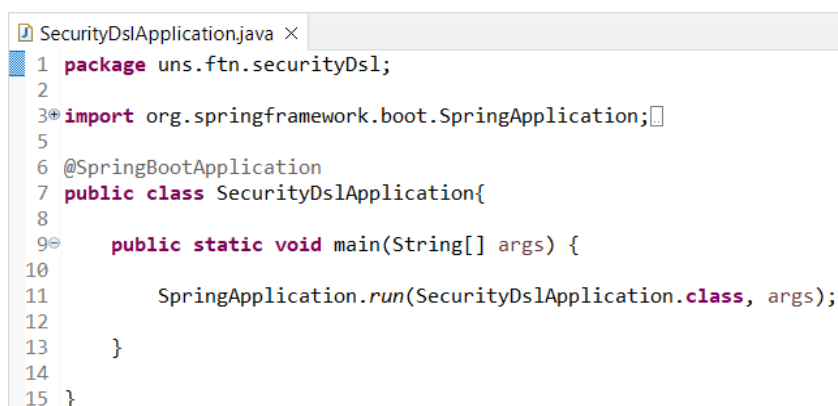
```
def static mainClassGenerator(String packageName, String appName) '''
    package «packageName»;

    import org.springframework.boot.SpringApplication;
    import org.springframework.boot.autoconfigure.SpringBootApplication;

    @SpringBootApplication
    public class «appName»{

        public static void main(String[] args) {
            SpringApplication.run(«appName».class, args);
        }
    }
'''
```

Листинг 5.1 – Шаблон за генерисање главне класе апликације



```
SecurityDslApplication.java ×
1 package uns.ftn.securityDsl;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class SecurityDslApplication{
8
9     public static void main(String[] args) {
10
11         SpringApplication.run(SecurityDslApplication.class, args);
12
13     }
14
15 }
```

Слика 5.1 – Пример главне класе апликације

```
def generateTests(String packageName, String appName) '''
    package «packageName»;

    import org.junit.jupiter.api.Test;
    import org.springframework.boot.test.context.SpringBootTest;

    @SpringBootTest
    class «appName»{

        @Test
        void contextLoads() {
        }
    }
'''
```

Листинг 5.2 – Шаблон за генерисање главне класе за тестове

```

SecurityDslApplicationTests.java ×
1 package uns.ftn.securityDsl;
2
3 import org.junit.jupiter.api.Test;
4
5
6 @SpringBootTest
7 class SecurityDslTests{
8
9     @Test
10    void contextLoads() {
11    }
12
13 }

```

Слика 5.2 – Пример главне класе за тестове

5.2 Генератор општих конфигурационих фајлова

Кључну улогу у конфигурацији и управљању апликацијом играју датотеке *pom.xml* и *application.properties*.

Датотека *pom.xml* се користи за конфигурацију и управљање спољним библиотекама апликације (енгл. *dependencies*) помоћу алата *Maven*. Неопходно је дефинисати све спољне библиотеке које апликација користи како би систем могао да их преузме и интегрише у апликацију. Неке од спољних библиотека су везане за одабрану базу података и безбедносни механизам, те ће у зависности од одабира бити наведене спољне библиотеке. Листинг 5.3 приказује шаблон за генерисање *pom.xml* фајла. На Сlici 5.3 приказан је пример генерисане датотеке у случају одабира система за управљање базама података *PostgreSQL* и безбедносног механизма *OAuth2.0*.

```

def generatePomXml(Application app)'''
    <?xml version="1.0" encoding="UTF-8"?>
    <project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.2</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>«app.group»</groupId>
    <artifactId>«app.artifact»</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>«app.name»</name>
    <description>«app.description»</description>
    <properties>
        <java.version>11</java.version>
    </properties>
    <dependencies>

```

```

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.modelmapper</groupId>
            <artifactId>modelmapper</artifactId>
            <version>2.4.4</version>
        </dependency>
        «IF
app.app_database?.vendorName.equals(EDatabaseType::MY_SQL)»
            <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
            </dependency>
        «ELSEIF
app.app_database?.vendorName.equals(EDatabaseType::POSTGRE_SQL)»
            <dependency>
                <groupId>org.postgresql</groupId>
                <artifactId>postgresql</artifactId>
            </dependency>
        «ELSEIF
app.app_database?.vendorName.equals(EDatabaseType::ORACLE)»
            <dependency>
                <groupId>com.oracle.database.jdbc</groupId>
                <artifactId>ojdbc8</artifactId>
            </dependency>
        «ENDIF»
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-security</artifactId>
            </dependency>
        «IF app.app_security instanceof JWT»
            <dependency>
                <groupId>io.jsonwebtoken</groupId>
                <artifactId>jjwt</artifactId>

```



```

        <version>0.6.0</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-annotations</artifactId>
    </dependency>
    «ELSEIF app.app_security instanceof OAuth2»
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-oauth2-
client</artifactId>
    </dependency>
    «ENDIF»
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-
plugin</artifactId>
            <configuration>
                <excludes>
                    <exclude>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                    </exclude>
                </excludes>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>

...

```

Листинг 5.3 – Шаблон генерисања датотеке *pom.xml*

```

securityDsl_OAuth2/pom.xml ×
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>2.5.2</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11
12  <groupId>uns.ftn</groupId>
13  <artifactId>securityDsl</artifactId>
14  <version>0.0.1-SNAPSHOT</version>
15
16  <name>securityDsl</name>
17  <description>This is an app that is generated with security DSL</description>
18
19  <properties>
20    <java.version>11</java.version>
21  </properties>
22
23  <dependencies>
24    <dependency>
25      <groupId>org.springframework.boot</groupId>
26      <artifactId>spring-boot-starter-data-jpa</artifactId>
27    </dependency>
28    <dependency>
29      <groupId>org.springframework.boot</groupId>
30      <artifactId>spring-boot-starter-web</artifactId>
31    </dependency>
32    <dependency>
33      <groupId>org.springframework.boot</groupId>
34      <artifactId>spring-boot-devtools</artifactId>
35      <scope>runtime</scope>
36      <optional>true</optional>
37    </dependency>
38    <dependency>
39      <groupId>org.projectlombok</groupId>
40      <artifactId>lombok</artifactId>
41      <optional>true</optional>
42    </dependency>
43    <dependency>
44      <groupId>org.springframework.boot</groupId>
45      <artifactId>spring-boot-starter-test</artifactId>
46      <scope>test</scope>
47    </dependency>
48    <dependency>
49      <groupId>org.modelmapper</groupId>
50      <artifactId>modelmapper</artifactId>
51      <version>2.4.4</version>
52    </dependency>
53    <dependency>
54      <groupId>org.postgresql</groupId>
55      <artifactId>postgresql</artifactId>
56    </dependency>
57    <dependency>
58      <groupId>org.springframework.boot</groupId>
59      <artifactId>spring-boot-starter-security</artifactId>
60    </dependency>
61    <dependency>
62      <groupId>org.springframework.boot</groupId>
63      <artifactId>spring-boot-starter-oauth2-client</artifactId>
64    </dependency>
65  </dependencies>
66  <build>
67    <plugins>
68      <plugin>
69        <groupId>org.springframework.boot</groupId>
70        <artifactId>spring-boot-maven-plugin</artifactId>
71        <configuration>
72          <excludes>
73            <exclude>
74              <groupId>org.projectlombok</groupId>
75              <artifactId>lombok</artifactId>
76            </exclude>
77          </excludes>
78        </configuration>
79      </plugin>
80    </plugins>
81  </build>
82
83 </project>

```

Слика 5.3 - Пример генерисане датотеке *pom.xml* у случају одабира система за управљање базама података *PostgreSQL* и безбедносног механизма *OAuth2.0*

Општи конфигурациони параметри, као што су подешавања базе података, портови на којима апликација послушнује захтеве, параметри за безбедност итд., дефинишу се у датотеци *application.properties*. Ова датотека омогућава лакше управљање подешавањима апликације без потребе за изменама у самом коду. Шаблон за генерисање датотеке *application.properties* приказан је на Листингу 5.4, док је пример за исти случај као и за датотеку *pom.xml* приказан на Слици 5.4.

```
def generateDatabaseProperties(Database database){

    if (database === null) return ''

    return '''
        «IF database.vendorName.equals(EDatabaseType::MY_SQL)»
        spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver

        spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dial
ec

        spring.datasource.initialization-mode=always
        spring.datasource.url=jdbc:mysql://«database.url»
        «ELSEIF database.vendorName.equals(EDatabaseType::POSTGRE_SQL)»
        spring.datasource.driverClassName=org.postgresql.Driver

        spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL
95Dialect

        spring.datasource.initialization-mode=always
        spring.datasource.url=jdbc:postgresql://«database.url»
        «ELSEIF database.vendorName.equals(EDatabaseType::ORACLE)»

        spring.datasource.driverClassName=oracle.jdbc.driver.OracleDriver

        spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle12cD
ialect

        spring.datasource.initialization-mode=always
        spring.datasource.url=jdbc:oracle:thin:@«database.url»
        «ENDIF»

        spring.datasource.username=«database.username»
        spring.datasource.password=«database.password»

        spring.jpa.show-sql=true
        spring.jpa.hibernate.ddl-auto=create-drop

        spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
        spring.jpa.properties.hibernate.format_sql=true
        spring.sql.init.mode=always
        spring.jpa.defer-datasource-initialization=true
        spring.jpa.open-in-view=false
    '''
}

def generateApplicationProperties(Application app) '''
    sprint.application.name=«app.name»
```

```

server.port=«app.port»
server.hostname=«app.hostname»

«generateDatabaseProperties(app.app_database)»
«IF app.app_security instanceof OAuth2»
    «var OAuth2 oauth = app.app_security as OAuth2»
    «FOR p : oauth.providers»

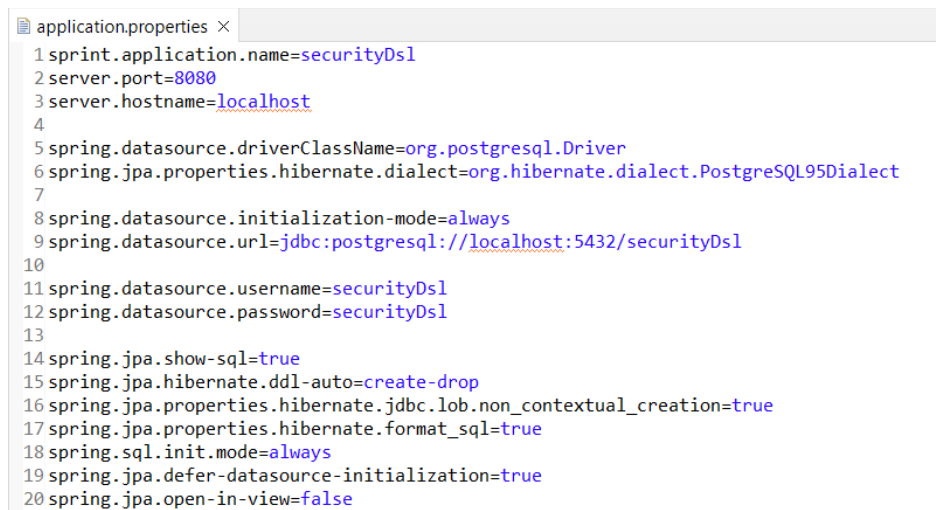
        spring.security.oauth2.client.registration.«p.name.toString.toLowerCase»
        .client-id=«p.clientId»

        spring.security.oauth2.client.registration.«p.name.toString.toLowerCase»
        .client-secret=«p.clientSecret»
        «IF p.redirectUri != null»

            spring.security.oauth2.client.registration.«p.name.toString.toLowerCase»
            .redirect-uri=«p.redirectUri»
        «ENDIF»
    «ENDFOR»
«ENDIF»
...

```

Листинг 5.4 – Шаблон генерисања датотеке *application.properties*



```

application.properties
1 sprint.application.name=securityDsl
2 server.port=8080
3 server.hostname=localhost
4
5 spring.datasource.driverClassName=org.postgresql.Driver
6 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL95Dialect
7
8 spring.datasource.initialization-mode=always
9 spring.datasource.url=jdbc:postgresql://localhost:5432/securityDsl
10
11 spring.datasource.username=securityDsl
12 spring.datasource.password=securityDsl
13
14 spring.jpa.show-sql=true
15 spring.jpa.hibernate.ddl-auto=create-drop
16 spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
17 spring.jpa.properties.hibernate.format_sql=true
18 spring.sql.init.mode=always
19 spring.jpa.defer-datasource-initialization=true
20 spring.jpa.open-in-view=false

```

Слика 5.4 - Пример генерисане датотеке *application.properties* у случају одабира система за управљање базама података *PostgreSQL* и безбедносног механизма *OAuth2.0*

5.3 Генератор слоја који моделује податке из базе података

Генератор слоја који моделује податке из базе података креира различите елементе неопходне за рад са подацима у апликацији. Ови елементи укључују класе модела (ентитета), репозиторијуме и класе за пренос података између слојева апликације (енгл. *Data Transfer Object - DTO*).

Класе модела описују структуру података у бази и креирају се на основу описа концепта *Entity*, односно концепата *User* и *Role*. Шаблон за генерисање класе која описује кориснике система приказан је на Листингу 5.5, док је на Листингу 5.6 приказана помоћна метода која садржи шаблон за генерисање обележја описане класе.

Пример класе *User* генерисане на основу овог шаблона у случају одабира безбедносног механизма *JWT* приказан је на Слици 5.5. У случају одабира безбедносног механизма *Basic Authentication* класа *User* неће садржати додатна обележја, док ће класа *Role* представљати енумерацију.

```
def generateUserModel(User user, Security security) '''
    package «packageName».model;

    «IF security instanceof BasicAuthentication»
    import «packageName».model.enumeration.Role;
    «ENDIF»

    «IF security instanceof JWT»
    import java.sql.Timestamp;
    import java.util.Date;
    import java.util.List;

    «ELSEIF security instanceof BasicAuthentication»
    import java.util.ArrayList;
    «ENDIF»

    import java.util.Collection;

    «IF security instanceof JWT»
    import javax.persistence.FetchType;
    import javax.persistence.JoinColumn;
    import javax.persistence.JoinTable;
    import javax.persistence.ManyToMany;

    «ELSEIF security instanceof BasicAuthentication»
    import javax.persistence.EnumType;
    import javax.persistence.Enumerated;
    «ENDIF»

    import javax.persistence.Column;
    import javax.persistence.Entity;
    import javax.persistence.GeneratedValue;
    import javax.persistence.Id;
    import javax.persistence.Table;
    import lombok.Getter;
    import lombok.Setter;

    import org.springframework.security.core.GrantedAuthority;
    «IF security instanceof BasicAuthentication»
    import
    org.springframework.security.core.authority.SimpleGrantedAuthority;
```

```

«ENDIF»

import org.springframework.security.core.userdetails.UserDetails;

import com.fasterxml.jackson.annotation.JsonIgnore;

@Getter
@Setter
@Entity
@Table(name="«user.tableName»")
public class User implements UserDetails {

    private static final long serialVersionUID = 1L;

    «generateAttributes(user.entity_attributes)»
    «IF security instanceof JWT»
        @JsonIgnore
        @Column(name = "password")
        private String password;

        @Column(name = "enabled")
        private boolean enabled;

        @Column(name = "last_password_reset_date")
        private Timestamp lastPasswordResetDate;

        @ManyToMany(fetch = FetchType.EAGER)
        @JoinTable(name = "user_role",
            joinColumns = @JoinColumn(name = "user_id",
referencedColumnName = "id"),
            inverseJoinColumns = @JoinColumn(name = "role_id",
referencedColumnName = "id"))
        private List<Role> roles;

        public void setPassword(String password) {
            Timestamp now = new Timestamp(new Date().getTime());
            this.setLastPasswordResetDate(now);
            this.password = password;
        }
    }

```

```

    @JsonIgnore
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return this.roles;
    }

    @Override
    public boolean isEnabled() {
        return enabled;
    }

    @JsonIgnore
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @JsonIgnore
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @JsonIgnore
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    «ELSEIF security instanceof BasicAuthentication»
    private String password;

    @Enumerated(EnumType.STRING)
    private Role role;

    @JsonIgnore
    @Override

```

```

        public boolean isEnabled() {
            return true;
        }

        @JsonIgnore
        @Override
        public boolean isCredentialsNonExpired() {
            return true;
        }

        @JsonIgnore
        @Override
        public boolean isAccountNonLocked() {
            return true;
        }

        @JsonIgnore
        @Override
        public boolean isAccountNonExpired() {
            return true;
        }

        @JsonIgnore
        @Override
        public Collection<? extends GrantedAuthority> getAuthorities() {
            Collection<GrantedAuthority> authorities = new
ArrayList<>();
            authorities.add(new
SimpleGrantedAuthority(role.getAuthority()));
            return authorities;
        }

        @Override
        public String getPassword() {
            // TODO Auto-generated method stub
            return password;
        }

```



```

@Override
public String getUsername() {
    // TODO Auto-generated method stub
    return «credentialUser»;
}
«ENDIF»
}'''

```

Листинг 5.5 – Шаблон за генерисање класе *User* која описује кориснике система

```

def generateAttributes(List<Attribute> unsortedAttributes){

    val ArrayList<Attribute> attributes = newArrayList

    for (a : unsortedAttributes) {
        if (a.isIdentifier) {
            attributes.add(0, a)
        } else {
            attributes.add(a)
        }
    }

    return '''
        @Id
        @GeneratedValue
        «FOR a : attributes»
            «IF a.columnName != null»@Column(name =
"«a.columnName»")
            «ENDIF»
            private «a.type» «a.name»;

        «ENDFOR»
    '''
}

```

Листинг 5.6 – Шаблон за генерисање обележја класа

```

1 package uns.ftn.securityDsl.model;
2
3 import java.sql.Timestamp;
4
5 @Getter
6 @Setter
7 @Entity
8 @Table(name="users")
9 public class User implements UserDetails {
10
11     private static final long serialVersionUID = 1L;
12
13     @Id
14     @GeneratedValue
15     private Long id;
16
17     @Column(name = "username")
18     private String username;
19
20     @Column(name = "first_name")
21     private String firstName;
22
23     @Column(name = "last_name")
24     private String lastName;
25
26     @JsonIgnore
27     @Column(name = "password")
28     private String password;
29
30     @Column(name = "enabled")
31     private boolean enabled;
32
33     @Column(name = "last_password_reset_date")
34     private Timestamp lastPasswordResetDate;
35
36     @ManyToMany(fetch = FetchType.EAGER)
37     @JoinTable(name = "user_role",
38         joinColumns = @JoinColumn(name = "user_id", referencedColumnName = "id"),
39         inverseJoinColumns = @JoinColumn(name = "role_id", referencedColumnName = "id"))
40     private List<Role> roles;
41
42     public void setPassword(String password) {
43         Timestamp now = new Timestamp(new Date().getTime());
44         this.setLastPasswordResetDate(now);
45         this.password = password;
46     }
47
48     @JsonIgnore
49     @Override
50     public Collection<? extends GrantedAuthority> getAuthorities() {
51         return this.roles;
52     }
53
54     @Override
55     public boolean isEnabled() {
56         return enabled;
57     }
58
59     @JsonIgnore
60     @Override
61     public boolean isAccountNonExpired() {
62         return true;
63     }
64
65     @JsonIgnore
66     @Override
67     public boolean isAccountNonLocked() {
68         return true;
69     }
70
71     @JsonIgnore
72     @Override
73     public boolean isCredentialsNonExpired() {
74         return true;
75     }
76 }

```

Слика 5.5 – Пример класе *User* у случају одабира безбедносног механизма *JWT*

Репозиторијуми су интерфејси који омогућавају комуникацију са базом података и самим тим манипулацију над подацима. Аутоматски генеришу методе за креирање, читање, модификацију и брисање података, што убрзава развој апликације. Пример шаблона за генерисање репозиторијума за ентитет *User* налази се на Листингу 5.7, док је пример генерисаног репозиторијума приказан на Слици 5.6.

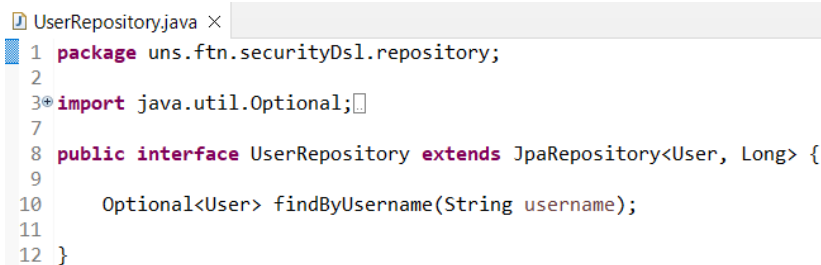
```
def generateUserRepository(User user)'''
    package «packageName».repository;

    import java.util.Optional;
    import org.springframework.data.jpa.repository.JpaRepository;

    import «packageName».model.User;

    public interface UserRepository extends JpaRepository<User,
«getIdentifier(user.entity_attributes).type»> {
        Optional<User> findBy«credentialUser.toFirstUpper»(String
«credentialUser»);
    }'''
```

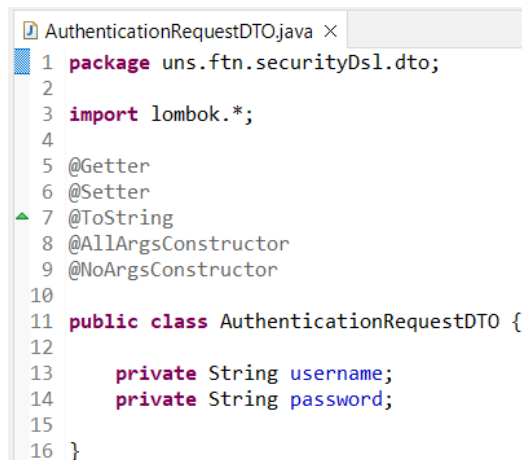
Листинг 5.7 – Шаблон за генерисање класе *UserRepository*



```
UserRepository.java ×
1 package uns.ftn.securityDsl.repository;
2
3 import java.util.Optional;
4
5
6
7
8 public interface UserRepository extends JpaRepository<User, Long> {
9
10     Optional<User> findByUsername(String username);
11
12 }
```

Слика 5.6 – Пример генерисаног репозиторијума за ентитет *User*

DTO класе омогућавају ефикасан пренос само неопходних података, чиме се повећавају перформансе, али и штите осетљиви подаци. Сличне су као и класе модела, а на Слици 5.7 приказана је *DTO* класа за пренос података при пријави на систем, где су неопходна само обележја која представљају идентификационе параметре.



```
AuthenticationRequestDTO.java ×
1 package uns.ftn.securityDsl.dto;
2
3 import lombok.*;
4
5 @Getter
6 @Setter
7 @ToString
8 @AllArgsConstructor
9 @NoArgsConstructor
10
11 public class AuthenticationRequestDTO {
12
13     private String username;
14     private String password;
15
16 }
```

Слика 5.7 – Пример генерисане *DTO* класе за пријаву корисника на систем

5.4 Генератор слоја за обраду захтева корисника

Генератор слоја за обраду захтева корисника аутоматски креира елементе неопходне за обраду корисничких захтева. Ови елементи укључују контролере и сервисе.

Контролери садрже приступне тачке (методе) које служе за повезивање одговарајућих сервиса и обраду резултата за жељену функционалност система. Овим методама се приступа помоћу путање (енгл. *path*) контролера и саме методе. Контролер за аутентификацију садржи методе за регистрацију, пријаву и одјаву са система. Шаблон за генерисање контролера за аутентификацију приказан је на Листингу 5.8, а пример на Слици 5.8.

```
def generateAuthController(Authentication authController, Security security,
String credentialNameUser){

    var Endpoint regEndpoint;
    var Endpoint loginEndpoint;
    var Endpoint logoutEndpoint;

    for (e : authController.controller_endpoints) {
        if(e.type == EEndpointType::REGISTRATION) regEndpoint = e
        if(e.type == EEndpointType::LOGIN) loginEndpoint = e
        if(e.type == EEndpointType::LOGOUT) logoutEndpoint = e
    }

    return '''
package «packageName».controller;

import «packageName».model.User;
import «packageName».dto.UserRequestDTO;
import «packageName».dto.AuthenticationRequestDTO;
import «packageName».service.IUserService;

«IF security instanceof JWT»
import «packageName».dto.UserTokenStateDTO;
import «packageName».util.TokenUtils;

«ENDIF»
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.http.MediaType;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import
org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.GetMapping;
```

```

import org.springframework.web.bind.annotation.PostMapping;

@RequiredArgsConstructor(onConstructor = @__(@Autowired))
@RequestMapping(value = "«authController.path»", produces =
MediaType.APPLICATION_JSON_VALUE)
@RestController
public class AuthController {

    private final IUserService userService;

    private final AuthenticationManager authenticationManager;

    «IF security instanceof JWT»
    private final TokenUtils tokenUtils;
    «ENDIF»

    @PostMapping("«regEndpoint.url»")
    public ResponseEntity<User> «regEndpoint.methodName»(@RequestBody
UserRequestDTO request) {
        return new ResponseEntity<>(userService.save(request),
HttpStatus.CREATED);
    }

    @PostMapping("«loginEndpoint.url»")
    public ResponseEntity<«IF security instanceof
JWT»UserTokenStateDTO«ELSEIF security instanceof
BasicAuthentication»User«ENDIF»> «loginEndpoint.methodName»(@RequestBody
AuthenticationRequestDTO request) {

        Authentication authentication = new
UsernamePasswordAuthenticationToken(request.get«credentialNameUser.toFirstUpper»()), request.getPassword());
        authentication =
authenticationManager.authenticate(authentication);

        SecurityContextHolder.getContext().setAuthentication(authentication);
        User user = (User) authentication.getPrincipal();

        «IF security instanceof JWT»
        String jwt = tokenUtils.generateToken(user.getUsername());
        int expiresIn = tokenUtils.getExpiredIn();
        return ResponseEntity.ok(new UserTokenStateDTO(jwt,
expiresIn));
        «ELSEIF security instanceof BasicAuthentication»
        return ResponseEntity.ok(user);
        «ENDIF»
    }

    @GetMapping("«logoutEndpoint.url»")
    public ResponseEntity<Void> «logoutEndpoint.methodName»() {
        SecurityContextHolder.clearContext();
        return ResponseEntity.ok().build();
    }

    ...
}

```

Листинг 5.8 – Шаблон за генерисање контролера за аутентификацију

```

1 package uns.ftn.securityDsl.controller;
2
3 import uns.ftn.securityDsl.model.User;
4
5
6 @RequiredArgsConstructor(onConstructor = @__(@Autowired))
7 @RequestMapping(value = "/auth", produces = MediaType.APPLICATION_JSON_VALUE)
8 @RestController
9 public class AuthController {
10
11     private final IUserService userService;
12
13     private final AuthenticationManager authenticationManager;
14
15     private final TokenUtils tokenUtils;
16
17     @PostMapping("/registration")
18     public ResponseEntity<User> registration(@RequestBody UserRequestDTO request) {
19         return new ResponseEntity<>(userService.save(request), HttpStatus.CREATED);
20     }
21
22     @PostMapping("/login")
23     public ResponseEntity<UserTokenStateDTO> login(@RequestBody AuthenticationRequestDTO request) {
24
25         Authentication authentication = new UsernamePasswordAuthenticationToken(request.getUsername(), request.getPassword());
26         authentication = authenticationManager.authenticate(authentication);
27         SecurityContextHolder.getContext().setAuthentication(authentication);
28         User user = (User) authentication.getPrincipal();
29         String jwt = tokenUtils.generateToken(user.getUsername());
30         int expiresIn = tokenUtils.getExpiredIn();
31         return ResponseEntity.ok(new UserTokenStateDTO(jwt, expiresIn));
32     }
33
34     @GetMapping("/logout")
35     public ResponseEntity<Void> logout() {
36         SecurityContextHolder.clearContext();
37         return ResponseEntity.ok().build();
38     }
39 }

```

Слика 5.8 – Пример генерисаног контролера за аутентификацију

Сервиси представљају компоненте које обрађују логику захтева. У оквиру сервиса се налазе методе које врше потребне операције над подацима тако што комуницирају са одговарајућим репозиторијумом. Аутоматски се генеришу сервис неопходни за логике захтева за регистрацију, пријаву и одјаву са система, а логика зависи од одабраног безбедносног механизма.

5.5 Генератор конфигурационих фајлова за основну аутентификацију

Сва потребна конфигурација за основну аутентификацију налази се у класи *SecurityConfig*. Помоћу шаблона (Листинг 5.9) омогућено је аутоматско генерисање ове класе чиме је обезбеђен основни ниво аутентификације и ауторизације. Пример приказан на Сlici 5.9 представља класу *SecurityConfig* генерисану на основу модела описаног у поглављу 4.3.1.

```

def generateSecurityConfig(Authentication authController)'''
    package «packageName».config;

    import «packageName».service.impl.UserServiceImpl;

    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.context.annotation.Bean;
    import org.springframework.context.annotation.Configuration;

```

```

import
org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.builders.Authent
icationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecu
rity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityCo
nfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;
    @Autowired
    private UserServiceImpl userService;

    @Bean
    public BCryptPasswordEncoder bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception
{
        httpSecurity.csrf().disable()
            .formLogin().disable()
            .logout().disable()

.authorizeRequests().antMatchers("«authController.path»/**").permitAll()
            .anyRequest().authenticated()
            .and().httpBasic();

    }

    @Autowired
    public void config(AuthenticationManagerBuilder authentication)
        throws Exception
    {
        authentication.authenticationProvider(daoAuthenticationProvider());
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws
Exception {
        return super.authenticationManagerBean();
    }

```

```

    }

    @Bean
    public DaoAuthenticationProvider daoAuthenticationProvider(){
        DaoAuthenticationProvider provider = new
        DaoAuthenticationProvider();
        provider.setPasswordEncoder(bCryptPasswordEncoder);
        provider.setUserDetailsService(userService);
        return provider;
    }

}'''

```

Листинг 5.9 – Шаблон за генерисање класе *SecurityConfig* за конфигурацију основне аутентификације

```

SecurityConfig.java ×
1 package uns.ftn.securityDsl.config;
2
3 import uns.ftn.securityDsl.service.impl.UserServiceImpl;
4
15
16 @Configuration
17 @EnableWebSecurity
18 public class SecurityConfig extends WebSecurityConfigurerAdapter {
19
20     @Autowired
21     private BCryptPasswordEncoder bCryptPasswordEncoder;
22     @Autowired
23     private UserServiceImpl userService;
24
25     @Bean
26     public BCryptPasswordEncoder bCryptPasswordEncoder() {
27         return new BCryptPasswordEncoder();
28     }
29
30     @Override
31     protected void configure(HttpSecurity httpSecurity) throws Exception {
32         httpSecurity.csrf().disable()
33             .formLogin().disable()
34             .logout().disable()
35             .authorizeRequests().antMatchers("/auth/**").permitAll()
36             .anyRequest().authenticated()
37             .and().httpBasic();
38     }
39
40
41     @Autowired
42     public void config(AuthenticationManagerBuilder authentication)
43         throws Exception
44     {
45         authentication.authenticationProvider(daoAuthenticationProvider());
46     }
47
48     @Bean
49     @Override
50     public AuthenticationManager authenticationManagerBean() throws Exception {
51         return super.authenticationManagerBean();
52     }
53
54     @Bean
55     public DaoAuthenticationProvider daoAuthenticationProvider(){
56         DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
57         provider.setPasswordEncoder(bCryptPasswordEncoder);
58         provider.setUserDetailsService(userService);
59         return provider;
60     }
61
62 }

```

Слика 5.9 – Пример генерисане класе *SecurityConfig* за конфигурацију основне аутентификације

Поред овога, генерисана је еnumerација са вредностима дефинисаних улога корисника (Слика 5.10).


```

1 package uns.ftn.securityDsl.model.enumeration;
2
3 public enum Role {
4
5     admin, user;
6
7     public String getAuthority() {
8         return this.name();
9     }
10 }

```

Слика 5.10 – Енумерација за улоге корисника

Резултат извршавања свих неопходних генератора за основну аутентификацију је *Spring* апликација са безбедноском конфигурацијом основне аутентификације. Апликација је организована у пакете (Слика 5.11) и садржи све класе неопходне за успешно функционисање апликације.

```

v securityDsl_BasicAuth [security-dsl main]
  > src/main/java
  > src/main/resources
  > src/test/java
  > JRE System Library [JavaSE-11]
  > Maven Dependencies
  > bin
  v src
    v main
      v java
        v uns
          v ftn
            v securityDsl
              v config
                SecurityConfig.java
              v controller
                AuthController.java
              v dto
                AuthenticationRequestDTO.java
                UserRequestDTO.java
              v model
                v enumeration
                  Role.java
                  User.java
              v repository
                UserRepository.java
            > service
              SecurityDslApplication.java
          v resources
            application.properties
        v test
          v java
            v uns
              v ftn
                v securityDsl
                  SecurityDslApplicationTests.java
    > target
      mvnw
      mvnw.cmd
      pom.xml
      README.md

```

Слика 5.11 – Организација генерисане *Spring* апликације за основну аутентификацију

5.6 Генератор конфигурационих фајлова за стандард JWT

За безбедносну конфигурацију стандарда JWT неопходне су додатне конфигурације. Поред класе SecurityConfig (Листинг 5.10, Слика 5.12) неопходно је конфигурисати класу за генерисање и верификацију токена (TokenUtils) и класу за проверу токена који се налазе у захтевима ка апликацији (TokenAuthenticationFilter). Примери приказани у овом поглављу су генерисани на основу модела описаног у поглављу 4.3.2.

```
def generateSecurityConfig(Authentication authController)'''
    package «packageName».config;

    import «packageName».security.auth.RestAuthenticationEntryPoint;
    import «packageName».security.auth.TokenAuthenticationFilter;
    import «packageName».service.impl.UserServiceImpl;
    import «packageName».util.TokenUtils;

    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.context.annotation.Bean;
    import org.springframework.context.annotation.Configuration;
    import org.springframework.http.HttpMethod;
    import
org.springframework.security.authentication.AuthenticationManager;
    import
org.springframework.security.config.annotation.authentication.builders.Authent
icationManagerBuilder;
    import
org.springframework.security.config.annotation.method.configuration.EnableGlob
alMethodSecurity;
    import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
    import
org.springframework.security.config.annotation.web.builders.WebSecurity;
    import
org.springframework.security.config.annotation.web.configuration.WebSecurityCo
nfigurerAdapter;
    import org.springframework.security.config.http.SessionCreationPolicy;
    import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
    import
org.springframework.security.web.authentication.www.BasicAuthenticationFilter;

    @Configuration
    @EnableGlobalMethodSecurity(prePostEnabled = true)
    public class SecurityConfig extends WebSecurityConfigurerAdapter {

        @Autowired
        private UserServiceImpl userService;

        @Autowired
        private RestAuthenticationEntryPoint
restAuthenticationEntryPoint;

        @Autowired
        private TokenUtils tokenUtils;
```

```

        @Bean
        public BCryptPasswordEncoder passwordEncoder() {
            return new BCryptPasswordEncoder();
        }

        @Bean
        @Override
        public AuthenticationManager authenticationManagerBean() throws
Exception {
            return super.authenticationManagerBean();
        }

        @Override
        public void configure(AuthenticationManagerBuilder auth) throws
Exception {
            auth

                .userDetailsService(userService)
                .passwordEncoder(passwordEncoder());
        }

        @Override
        protected void configure(HttpSecurity http) throws Exception {
            http

                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).an
d()

                .exceptionHandling().authenticationEntryPoint(restAuthenticationEntryPoint).an
d()

                .authorizeRequests().antMatchers("«authController.path»/**").permitAll()

                .anyRequest().authenticated().and()
                .cors().and()
                .addFilterBefore(new
TokenAuthenticationFilter(tokenUtils, userService),
BasicAuthenticationFilter.class);

            http.csrf().disable();
        }

        @Override
        public void configure(WebSecurity web) throws Exception {
            web.ignoring().antMatchers(HttpMethod.POST,
"«authController.path»«SecuraDslGenerator.getLoginEndpoint(authController)»");
            web.ignoring().antMatchers(HttpMethod.GET, "/", "/webjars/**",
"/*.html", "favicon.ico", "/*.html",
"/**/*.css", "/**/*.js");
        }
    }
}'''

```

Листинг 5.10 - Шаблон за генерисање класе *SecurityConfig* за конфигурацију безбедносног механизма *JWT*

```

1 package uns.ftn.securityDsl.config;
2
3 import uns.ftn.securityDsl.security.auth.RestAuthenticationEntryPoint;
4
5
6
7
8
9
10
11
12
13
14 @Configuration
15 @EnableGlobalMethodSecurity(prePostEnabled = true)
16 public class SecurityConfig extends WebSecurityConfigurerAdapter {
17
18     @Autowired
19     private UserServiceImpl userService;
20
21
22     @Autowired
23     private RestAuthenticationEntryPoint restAuthenticationEntryPoint;
24
25
26     @Autowired
27     private TokenUtils tokenUtils;
28
29
30     @Bean
31     public BCryptPasswordEncoder passwordEncoder() {
32         return new BCryptPasswordEncoder();
33     }
34
35
36     @Bean
37     @Override
38     public AuthenticationManager authenticationManagerBean() throws Exception {
39         return super.authenticationManagerBean();
40     }
41
42     @Override
43     public void configure(AuthenticationManagerBuilder auth) throws Exception {
44         auth
45             .userDetailsService(userService)
46             .passwordEncoder(passwordEncoder());
47     }
48
49     @Override
50     protected void configure(HttpSecurity http) throws Exception {
51         http
52             .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
53             .exceptionHandling().authenticationEntryPoint(restAuthenticationEntryPoint).and()
54             .authorizeRequests().antMatchers("/auth/**").permitAll()
55             .anyRequest().authenticated().and()
56             .cors().and()
57             .addFilterBefore(new TokenAuthenticationFilter(tokenUtils, userService), BasicAuthenticationFilter.class);
58
59         http.csrf().disable();
60     }
61
62     @Override
63     public void configure(WebSecurity web) throws Exception {
64         web.ignoring().antMatchers(HttpMethod.POST, "/auth/login");
65         web.ignoring().antMatchers(HttpMethod.GET, "/", "/webjars/**", "/*.html", "favicon.ico", "/*.html",
66             "/*.css", "/*.js");
67     }
68
69
70
71
72
73
74
75
76
77
78
79

```

Слика 5.12 – Пример класе *SecurityConfig* за конфигурацију безбедносног механизма *JWT*

У класи *TokenUtils* се врши генерисање и верификација токена на основу провере потписа, временског ограничења и слично. Листинг 5.11 приказује шаблон за генерисање ове класе, док се на Слици 5.13 налази пример дела класе са методом за генерисање токена.

```

def String generateTokenUtils(JWT jwt, String credentialUser)'''
    package «packageName».util;

    import java.util.Date;

    import javax.servlet.http.HttpServletRequest;

    import org.springframework.beans.factory.annotation.Value;
    import org.springframework.security.core.userdetails.UserDetails;
    import org.springframework.stereotype.Component;

    import io.jsonwebtoken.Claims;
    import io.jsonwebtoken.ExpiredJwtException;

```

```

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import «packageName».model.User;

@Component
public class TokenUtils {

    private String ISSUER =
"«SecuraDslGenerator.findClaimByName(jwt.claims, 'issuer').value»";

    private String SECRET = "«jwt.secret»";

    private int EXPIRES_IN =
«SecuraDslGenerator.findClaimByName(jwt.claims, 'expirationTime').value»;

    @Value("Authorization")
    private String AUTH_HEADER;

    private static final String AUDIENCE_WEB =
"«SecuraDslGenerator.findClaimByName(jwt.claims, 'audience')»";
    // Algoritam za potpisivanje JWT
    private SignatureAlgorithm SIGNATURE_ALGORITHM =
SignatureAlgorithm.«jwt.signatureAlgorithm»;

    public String generateToken(String «credentialUser») {
        return Jwts.builder()
            .setIssuer(ISSUER)
            .setSubject(«findSubjectClaim(jwt.claims).claim_attribute.name»)
            .setAudience(generateAudience())
            .setIssuedAt(new Date())
            .setExpiration(generateExpirationDate())
            .signWith(SIGNATURE_ALGORITHM,
SECRET).compact();
    }

    private String generateAudience() {
        return AUDIENCE_WEB;
    }

    private Date generateExpirationDate() {
        return new Date(new Date().getTime() + EXPIRES_IN);
    }

    public String getToken(HttpServletRequest request) {
        String authHeader = getAuthHeaderFromHeader(request);

        if (authHeader != null && authHeader.startsWith("Bearer "))
        {
            return authHeader.substring(7);
        }
    }
}

```

```

        return null;
    }

    public String getCredentialFromToken(String token) {
        String credential;
        try {
            final Claims claims =
this.getAllClaimsFromToken(token);
            credential = claims.getSubject();
        } catch (ExpiredJwtException ex) {
            throw ex;
        } catch (Exception e) {
            credential = null;
        }
        return credential;
    }

    public Date getIssuedAtDateFromToken(String token) {
        Date issueAt;
        try {
            final Claims claims =
this.getAllClaimsFromToken(token);
            issueAt = claims.getIssuedAt();
        } catch (ExpiredJwtException ex) {
            throw ex;
        } catch (Exception e) {
            issueAt = null;
        }
        return issueAt;
    }

    public String getAudienceFromToken(String token) {
        String audience;
        try {
            final Claims claims =
this.getAllClaimsFromToken(token);
            audience = claims.getAudience();
        } catch (ExpiredJwtException ex) {
            throw ex;
        } catch (Exception e) {
            audience = null;
        }
        return audience;
    }

    public Date getExpirationDateFromToken(String token) {
        Date expiration;
        try {
            final Claims claims =
this.getAllClaimsFromToken(token);
            expiration = claims.getExpiration();
        } catch (ExpiredJwtException ex) {
            throw ex;
        }
    }

```

```

        } catch (Exception e) {
            expiration = null;
        }
        return expiration;
    }

    private Claims getAllClaimsFromToken(String token) {
        Claims claims;
        try {
            claims = Jwts.parser()
                .setSigningKey(SECRET)
                .parseClaimsJws(token)
                .getBody();
        } catch (ExpiredJwtException ex) {
            throw ex;
        } catch (Exception e) {
            claims = null;
        }
        return claims;
    }

    public Boolean validateToken(String token, UserDetails
userDetails) {
        User user = (User) userDetails;
        final String credential = getCredentialFromToken(token);
        final Date created = getIssuedAtDateFromToken(token);

        return (credential != null
            &&
            credential.equals(userDetails.get«credentialUser.toFirstUpper»())
            && !isCreatedBeforeLastPasswordReset(created,
            user.getLastPasswordResetDate()));
    }

    private Boolean isCreatedBeforeLastPasswordReset(Date created,
    Date lastPasswordReset) {
        return (lastPasswordReset != null &&
            created.before(lastPasswordReset));
    }

    public int getExpiredIn() {
        return EXPIRES_IN;
    }

    public String getAuthHeaderFromHeader(HttpServletRequest request) {
        return request.getHeader(AUTH_HEADER);
    }
}'''

```

Листинг 5.11 – Шаблон за генерисање класе *TokenUtils*

```

TokenUtils.java ×
1 package uns.ftn.securityDsl.util;
2
3 import java.util.Date;
16
17 @Component
18 public class TokenUtils {
19
20     private String ISSUER = "securityDsl";
21
22     public String SECRET = "somesecret";
23
24     private int EXPIRES_IN = 1800000;
25
26 @Value("Authorization")
27     private String AUTH_HEADER;
28
29     private static final String AUDIENCE_WEB = "AUDIENCE_WEB";
30     // Algoritam za potpisivanje JWT
31     private SignatureAlgorithm SIGNATURE_ALGORITHM = SignatureAlgorithm.HS512;
32
33
34 public String generateToken(String username) {
35     return Jwts.builder()
36         .setIssuer(ISSUER)
37         .setSubject(username)
38         .setAudience(generateAudience())
39         .setIssuedAt(new Date())
40         .setExpiration(generateExpirationDate())
41         .signWith(SIGNATURE_ALGORITHM, SECRET).compact();
42
43
44 }

```

Слика 5.13 – Пример дела класе *TokenUtils* са методом за генерисање токена

Класа *TokenAuthenticationFilter* представља филтер који се користи за проверу валидности токена у корисничким захтевима. На овај начин апликација зна ко је тренутно аутентификовани корисник. Листинг 5.12 приказује шаблон за генерисање ове класе, а пример је приказан на Сlici 5.14.

```

def String generateTokenAuthenticationFilter()'''
    package «packageName».security.auth;

    import java.io.IOException;

    import javax.servlet.FilterChain;
    import javax.servlet.ServletException;
    import javax.servlet.http.HttpServletRequest;
    import javax.servlet.http.HttpServletResponse;

    import «packageName».util.TokenUtils;
    import org.apache.commons.logging.Log;
    import org.apache.commons.logging.LogFactory;
    import org.springframework.security.core.context.SecurityContextHolder;
    import org.springframework.security.core.userdetails.UserDetails;
    import org.springframework.security.core.userdetails.UserDetailsService;
    import org.springframework.web.filter.OncePerRequestFilter;

    import io.jsonwebtoken.ExpiredJwtException;

```



```

public class TokenAuthenticationFilter extends OncePerRequestFilter {

    private TokenUtils tokenUtils;

    private UserDetailsService userDetailsService;

    protected final Log LOGGER = LoggerFactory.getLog(getClass());

    public TokenAuthenticationFilter(TokenUtils tokenHelper,
UserDetailsService userDetailsService) {
        this.tokenUtils = tokenHelper;
        this.userDetailsService = userDetailsService;
    }

    @Override
    public void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain chain)
        throws IOException, ServletException {

        String credential;
        String authToken = tokenUtils.getToken(request);

        try {

            if (authToken != null) {

                credential =
tokenUtils.getCredentialFromToken(authToken);

                if (credential != null) {

                    UserDetails userDetails =
userDetailsService.loadUserByUsername(credential);

                    if (tokenUtils.validateToken(authToken,
userDetails)) {

                        TokenBasedAuthentication authentication = new
TokenBasedAuthentication(userDetails);
                        authentication.setToken(authToken);
SecurityContextHolder.getContext().setAuthentication(authentication);

                    }

                }

            }

            } catch (ExpiredJwtException ex) {
                LOGGER.debug("Token expired!");
            }

            chain.doFilter(request, response);

        }

    }
}

```

Листинг 5.12 – Шаблон за генерисање класе *TokenAuthenticationFilter*

```

TokenAuthenticationFilter.java ×
1 package uns.ftn.securityDsl.security.auth;
2
3 import java.io.IOException;
4
19
20
21 public class TokenAuthenticationFilter extends OncePerRequestFilter {
22
23     private TokenUtils tokenUtils;
24
25     private UserDetailsService userDetailsService;
26
27     protected final Log LOGGER = LoggerFactory.getLog(getClass());
28
29     public TokenAuthenticationFilter(TokenUtils tokenHelper, UserDetailsService userDetailsService) {
30         this.tokenUtils = tokenHelper;
31         this.userDetailsService = userDetailsService;
32     }
33
34     @Override
35     public void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
36         throws IOException, ServletException {
37
38         String credential;
39         String authToken = tokenUtils.getToken(request);
40
41         try {
42
43             if (authToken != null) {
44
45                 credential = tokenUtils.getCredentialFromToken(authToken);
46
47                 if (credential != null) {
48
49                     UserDetails userDetails = userDetailsService.loadUserByUsername(credential);
50
51                     if (tokenUtils.validateToken(authToken, userDetails)) {
52
53                         TokenBasedAuthentication authentication = new TokenBasedAuthentication(userDetails);
54                         authentication.setToken(authToken);
55                         SecurityContextHolder.getContext().setAuthentication(authentication);
56                     }
57                 }
58             }
59
60         } catch (ExpiredJwtException ex) {
61             LOGGER.debug("Token expired!");
62         }
63
64         chain.doFilter(request, response);
65     }
66
67 }

```

Слика 5.14 – Пример генерисане класе *TokenAuthenticationFilter*

Поред овога, генеришу се и класе које позивају методе из претходно наведених класа. Класе које су резултат генератора неопходних за безбедносну конфигурацију помоћу *JWT* токена омогућавају сигурну и ефикасну аутентификацију и ауторизацију. Генерисане класе су организоване у пакета (Слика 5.15) и чине *Spring* апликацију безбедну за коришћење.

- ▼ > securityDsl_JWT [security-dsl main]
 - > > src/main/java
 - > src/main/resources
 - > src/test/java
 - > JRE System Library [JavaSE-11]
 - > Maven Dependencies
 - ▼ > src
 - ▼ > main
 - ▼ > java
 - ▼ > uns
 - ▼ > ftn
 - ▼ > securityDsl
 - ▼ > config
 - > SecurityConfig.java
 - WebConfig.java
 - ▼ controller
 - AuthController.java
 - ▼ dto
 - AuthenticationRequestDTO.java
 - UserRequestDTO.java
 - UserTokenStateDTO.java
 - ▼ model
 - Role.java
 - User.java
 - ▼ repository
 - RoleRepository.java
 - UserRepository.java
 - ▼ security
 - ▼ auth
 - RestAuthenticationEntryPoint.java
 - TokenAuthenticationFilter.java
 - TokenBasedAuthentication.java
 - ▼ service
 - ▼ impl
 - RoleServiceImpl.java
 - UserServiceImpl.java
 - IRoleService.java
 - IUserService.java
 - ▼ util
 - TokenUtils.java
 - SecurityDslApplication.java
 - ▼ resources
 - application.properties
 - data.sql
 - ▼ test
 - ▼ java
 - ▼ uns
 - ▼ ftn
 - ▼ securityDsl
 - SecurityDslApplicationTests.java
 - > target
 - mvnw
 - mvnw.cmd
 - pom.xml
 - README.md

Слика 5.15 – Структура *Spring* апликације са конфигурисаним безбедносним механизмом *JWT*

5.7 Генератор конфигурационих фајлова за стандард OAuth2

Примери приказани у овом поглављу су генерисани на основу модела описаног у поглављу 4.3.3. Листинг 5.13 приказује шаблон за генерисање класе *SecurityConfig* за конфигурацију безбедносног механизма *OAuth2.0*. Пример генерисане класе се налази на Слици 5.16. Кључне елементе у аутенфитификацији помоћу безбедносног механизма *OAuth2.0* чине провајдери. Они се дефинишу у оквиру конфигурационе датотеке *application.properties*. На Слици 5.17 приказана је конфигурација за провајдере *Google* и *Github*. Навођењем идентификационе ознаке и кључа корисника омогућава се аутентификација помоћу одабраних провајдера.

```
def generateSecurityConfig()'''
    package «packageName».config;

    import org.springframework.context.annotation.Bean;
    import org.springframework.context.annotation.Configuration;
    import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
    import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
    import org.springframework.security.web.SecurityFilterChain;
    import static
org.springframework.security.config.Customizer.withDefaults;

    @Configuration
    @EnableWebSecurity
    public class SecurityConfig {

        @Bean
        SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
            return http
                .authorizeHttpRequests(auth -> {
                    auth.antMatchers("/").permitAll();
                    auth.anyRequest().authenticated();
                })
                .oauth2Login(withDefaults())
                .formLogin(withDefaults())
                .build();
        }
    }
'''
```

Листинг 5.13 –Шаблон за генерисање класе *SecurityConfig* за конфигурацију безбедносног механизма *OAuth2.0*

```

1 package uns.ftn.securityDsl.config;
2
3 import org.springframework.context.annotation.Bean;
4
5 @Configuration
6 @EnableWebSecurity
7 public class SecurityConfig {
8
9     @Bean
10    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
11    {
12        return http
13            .authorizeHttpRequests(auth -> {
14                auth.antMatchers("/").permitAll();
15                auth.anyRequest().authenticated();
16            })
17            .oauth2Login(withDefaults())
18            .formLogin(withDefaults())
19            .build();
20    }
21 }

```

Слика 5.16 – Пример генерисане класе *SecurityConfig* за безбедносни механизам *OAuth2.0*

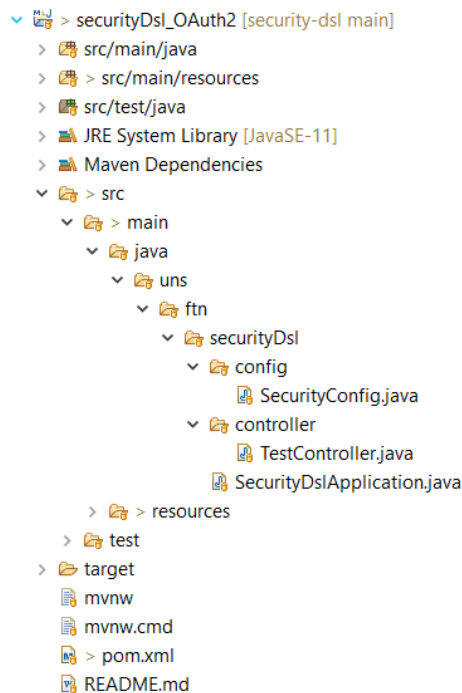
```

1 spring.application.name=securityDsl
2 server.port=8080
3 server.hostname=localhost
4
5 spring.datasource.driverClassName=org.postgresql.Driver
6 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL95Dialect
7
8 spring.datasource.initialization-mode=always
9 spring.datasource.url=jdbc:postgresql://localhost:5432/securityDsl
10
11 spring.datasource.username=securityDsl
12 spring.datasource.password=securityDsl
13
14 spring.jpa.show-sql=true
15 spring.jpa.hibernate.ddl-auto=create-drop
16 spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
17 spring.jpa.properties.hibernate.format_sql=true
18 spring.sql.init.mode=always
19 spring.jpa.defer-datasource-initialization=true
20 spring.jpa.open-in-view=false
21
22
23 spring.security.oauth2.client.registration.google.client-id=x
24 spring.security.oauth2.client.registration.google.client-secret=x
25
26 spring.security.oauth2.client.registration.github.client-id=x
27 spring.security.oauth2.client.registration.github.client-secret=x
28
29

```

Слика 5.17 – Пример конфигурационе датотеке *application.properties* за безбедносни механизам *OAuth2.0*

Након генерисања неопходних класа, омогућена је аутентификација помоћу конфигурисаних провајдера. Класе су организоване у пакете (Слика 5.18) и генерисана *Spring* апликација је сигурна за коришћење.

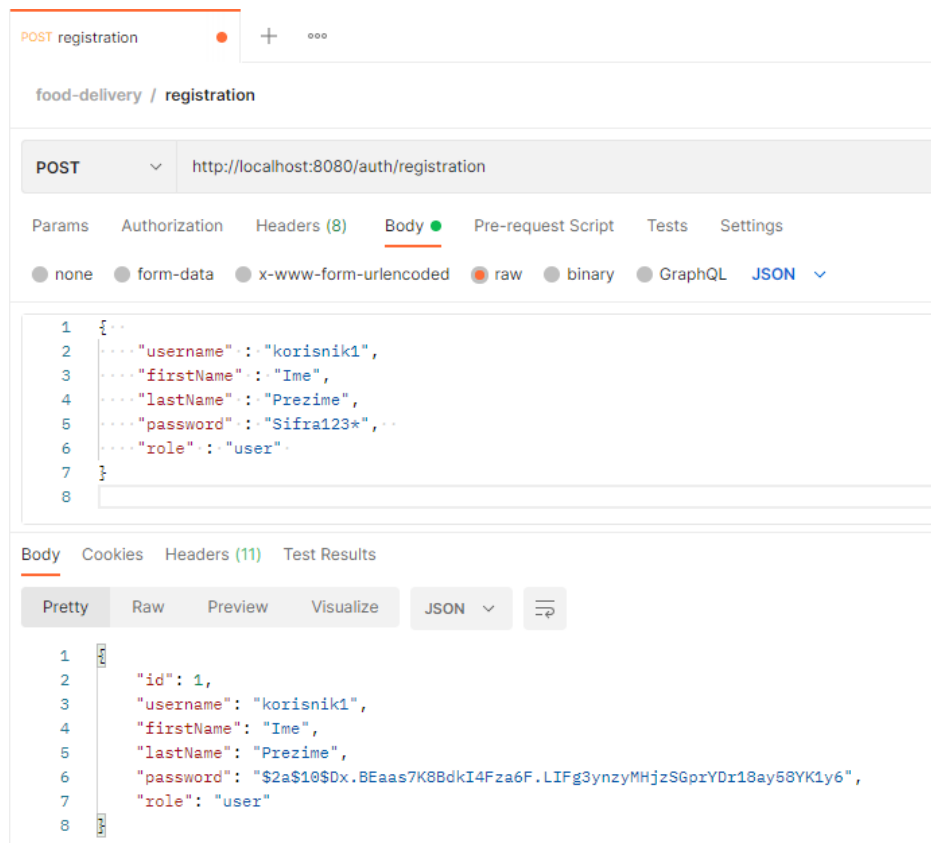


Слика 5.18 – Структура *Spring* апликације са конфигурисаним безбедносним механизмом *OAuth2.0*

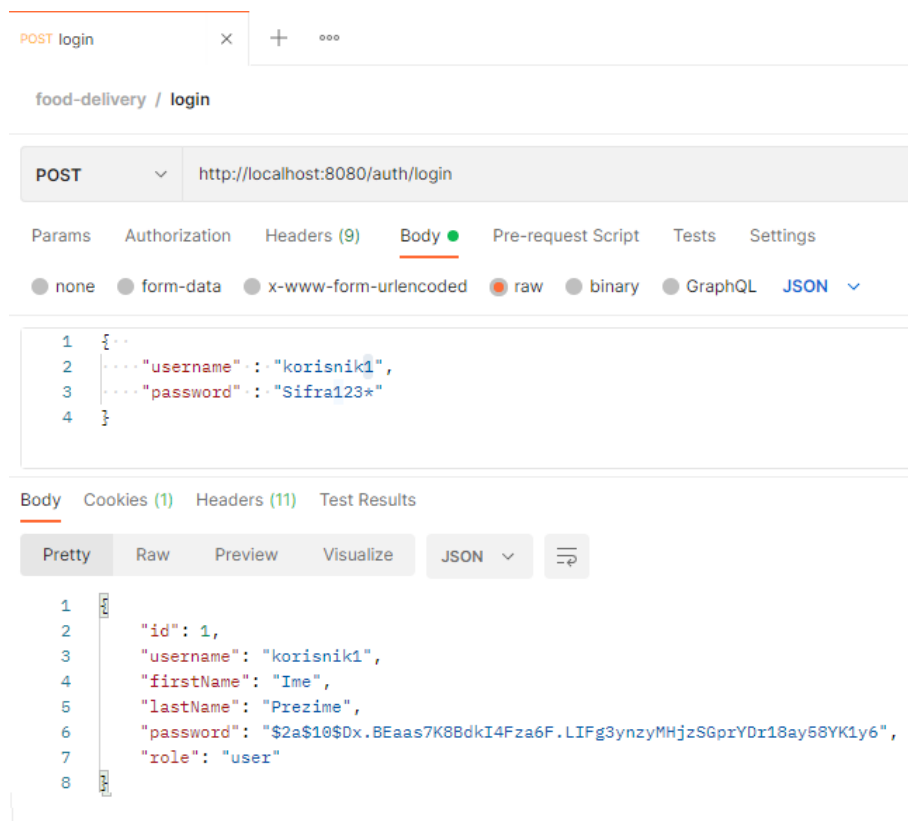
5.8 Тестирање генерисаних *Spring* апликација са конфигурисаном основном аутентификацијом или стандардом *JWT*

Spring апликације са конфигурисаном основном аутентификацијом или безбедносним механизмом *JWT* тестиране су помоћу платформе *Postman*. Слика 5.19 приказује успешну регистрацију на систем. За успешан одговор на захтев за регистрацију неопходно је да се у телу захтева налазе сва неопходна обележја корисника за регистрацију и да захтев буде послат на одговарајућу путању. Регистрација на систем је могућа само за кориснике којима је додељена улога која је у моделу дефинисана као клијентска.

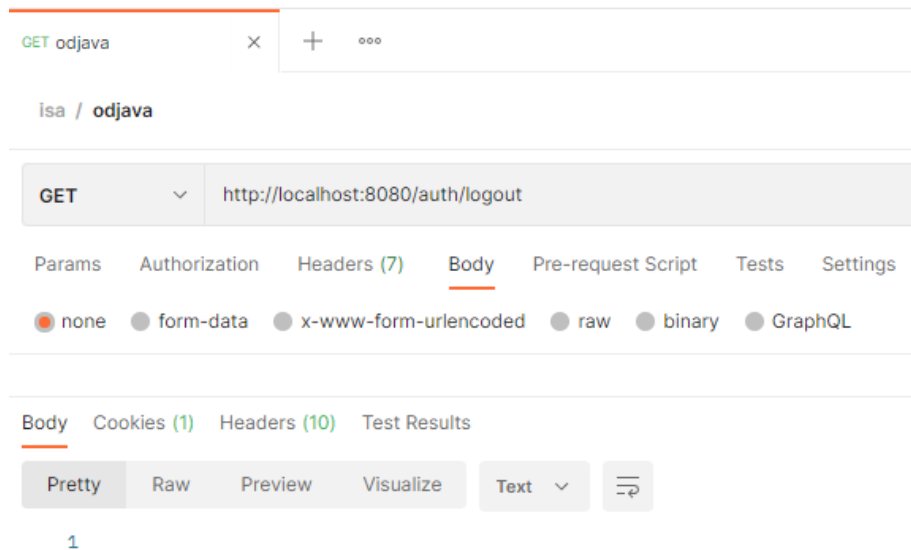
На Слици 5.20 приказан је успешан захтев за пријаву на систем, а на Слици 5.21 успешан захтев за одјаву са система. За успешну пријаву неопходно је унети валидну комбинацију идентификационог параметра и лозинке.



Слика 5.19 – Успешна регистрација на систем



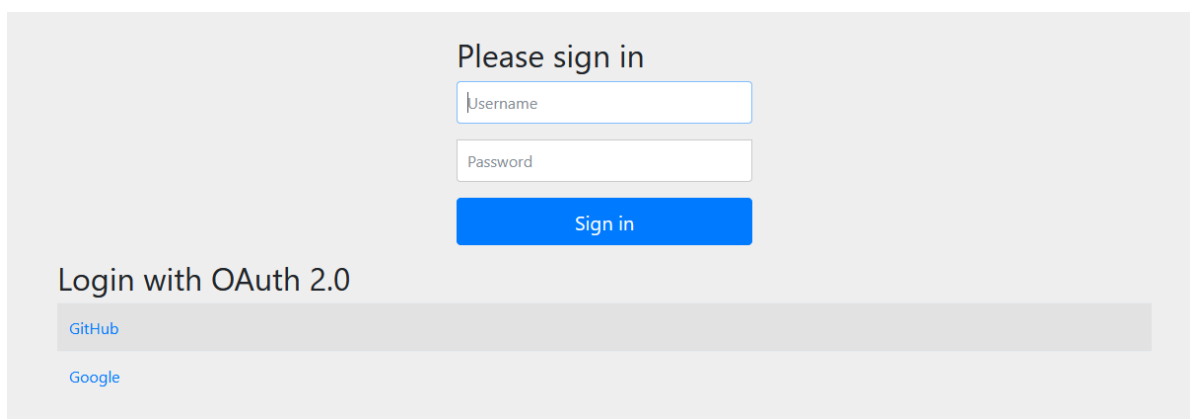
Слика 5.20 – Успешна пријава на систем



Слика 5.21 – Успешна одјава са система

5.9 Тестирање генерисаних *Spring* апликација са конфигурисаним стандардом *OAuth2.0*

Уколико је у питању генерисана *Spring* апликација са конфигурисаним безбедносним механизмом *OAuth2.0*, приликом пријаве на систем могуће је одабрати пријаву помоћу неког од конфигурисаних провајдера (Слика 5.22). Одабиром жељеног провајдера, корисник се преусмерава на страницу за аутентификацију.



Слика 5.22 – Пријава на систем у случају конфигурисаног безбедносног механизма *OAuth2.0*

6. Закључак

Наменски језик *securaDSL* описан у овом раду садржи концепте неопходне за моделовање *Spring* апликација, а додатно подржава конфигурацију три безбедносна механизма: основне аутентификације, *JWT* и *OAuth2.0*. Поред наменског језика, описани су генератори који су развијени у циљу генерисања извршивог кода на основу модела креираног помоћу језика *securaDSL*.

Наменски језик *securaDSL* намењен је експертима у пољу безбедносне конфигурације софтвера. Коришћење наменског језика за моделовање *Spring* апликација са конфигурисаним безбедносним аспектима и пропратних генератора значајно убрзава процес развоја софтвера и умањује могућност грешке која настаје при ручном писању кода. Постојање модела апликације помаже у уочавању делова које захтевају измену, а које би иначе изазвале ручну дораду која би утицала на велики број линија кода и самим тим довела до грешака које се тешко идентификују и отклањају. Поред овога, структура саме апликације је видљивија и подложнија дискусији унутар развојног тима. Аутоматско генерисање кода повећава ефикасност и доприноси конзистентности, што унапређује квалитет саме апликације.

Додатни развој и проширење језика *securaDSL* може да укључује увођење нових концепата за опис слоја за моделовање података из базе података и слоја за обраду захтева, што би допринело прилагођавању другим гранама индустрије. Остављен је простор за проширење тренутног начина за моделовање и генератора за безбедносни механизам *OAuth 2.0* увођењем регистрације на систем. Такође, могуће је увести подршку за додатне безбедносне механизме увођењем нових безбедносних концепата. Са аспекта подршке нових технологија, могуће је увођење подршке за друге програмске језике и радне оквире. Ово би захтевало измену апстрактне и конкретне синтаксе, али и развој нових генератора који би креирали код у различитим програмским језицима. Оваква унапређења би допринела значају и примени наменског језика *securaDSL* и развијених генератора.

Скраћенице

- *DTO – Data Transfer Object*
- *HTTP – Hypertext Transfer Protocol*
- *JWT - JSON Web Token*
- *OAuth2.0 – Open Authorization 2.0*
- *OCL – Object Constraint Language*
- *securaDSL – Secura Domain-Specific Language*
- *URL – Uniform Resource Locator*

Литература

- [1] Java Programming Language. [Online], Приступљено датума: 31.8.2023.
<https://docs.oracle.com/en/java/>
- [2] Sujay, Vailshery, L. (2023). Most used programming languages worldwide as of 2023. [Online], Приступљено датума: 31.8.2023.
<https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/>
- [3] Spring Framework. [Online], Приступљено датума: 31.8.2023. <https://spring.io/>
- [4] Maple, S., & Binstock A. (2018). JVM Ecosystem Report 2018: About your Platform & Application. [Online], Приступљено датума: 31.8.2023. <https://snyk.io/blog/jvm-ecosystem-report-2018-platform-application/>
- [5] Brambilla, M., Cabot, J., & Wimmer, M. (2012). Model-driven software engineering in practice. Kentfield, California. Morgan & Claypool Publishers
- [6] Beydeda, S., Book, M., & Gruhn, V. (1998). Model-Driven Software Development. Berlin, Germany. Springer-Verlag
- [7] Eclipse Modeling Framework. [Online], Приступљено датума: 31.8.2023.
<https://eclipse.dev/modeling/emf/>
- [8] Designing Ecore Models. [Online], Приступљено датума: 2.9.2023.
<https://eclipse.dev/ecoretools/doc/>
- [9] Eclipse OCL (Object Constraint Language). [Online], Приступљено датума: 31.8.2023. <https://projects.eclipse.org/projects/modeling.mdt.ocl>
- [10] Xtext. [Online], Приступљено датума: 31.8.2023.
<https://projects.eclipse.org/projects/modeling.tmf.xtext>
- [11] Xtend. [Online], Приступљено датума: 31.8.2023.
<https://eclipse.dev/Xtext/xtend/documentation/index.html>
- [12] Spring Boot. [Online], Приступљено датума: 31.8.2023.
<https://spring.io/projects/spring-boot>
- [13] Spring Security. [Online], Приступљено датума: 31.8.2023.
<https://spring.io/projects/spring-security>
- [14] PostgreSQL. [Online], Приступљено датума: 31.8.2023.
<https://www.postgresql.org/about/>
- [15] MySQL. [Online], Приступљено датума: 31.8.2023. <https://dev.mysql.com/doc/>
- [16] Oracle Database. [Online], Приступљено датума: 31.8.2023.
<https://docs.oracle.com/en/database/>

- [17] Reschke, J., (2015). The 'Basic' HTTP authentication scheme. Internet Engineering Task Force
- [18] Jones, M., Bradley, J. & Sakimura. N. (2015). JSON Web Token (JWT). Internet Engineering Task Force
- [19] Hardt, D. Ed. (2012). The OAuth 2.0 Authorization Framework. Internet Engineering Task Force
- [20] Spring Initializer. [Online], Приступљено датума: 31.8.2023. <https://start.spring.io/>
- [21] Šuljkanović, A., Milosavljević, B., Indić, V., Dejanović, I. (2022). Developing Microservice-Based Applications Using the Silvera Domain-Specific Language
- [22] Terzić, B., Dimitrieski, V., Kordić, S., Milosavljević, G., Luković, I. (2017). MicroBuilder: A Model-Driven Tool for the Specification of REST Microservice Architectures
- [23] *JHipster*. [Online], Приступљено датума: 31.8.2023. <https://www.jhipster.tech/>

Биографија

Јелена Хрњак рођена је 21. августа 1999. године у Бачкој Тополи где је стекла основно образовање у основној школи „Никола Тесла”. Даље школовање наставила је у Суботици где је завршила Гимназију „Светозар Марковић”, природно-математички смер. Школске 2018/2019 године уписује се на Факултет техничких наука Универзитета у Новом Саду, смер Рачунарство и аутоматика. Основне академске студије завршила је 2022. године и исте године уписује се на мастер академске студије на студијском програму Рачунарство и аутоматика Факултета техничких наука. Положила је све испите предвиђене планом и програмом мастер академских студија.