|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** |  |

Јелена Хрњак

***SECURADSL* – наменски језик за подршку брзог успостављања конфигурације безбедносних аспеката у радном оквиру *spring***

Мастер рад

- Мастер академске студије -

Нови Сад, 2023.

|  |  |  |  |
| --- | --- | --- | --- |
| 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, доцент | |
| Наслов рада, **НР**: | | *SecuraDSL* – Наменски језик за подршку брзог успостављања конфигурације безбедносних аспеката у радном оквиру *Spring* | |
| 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 / Факултет техничких наука | |
| Место и адреса, **МА**: | | Нови Сад, Трг Доситеја Обрадовића 6 | |
| Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога) | | 6/77/22/18/22/0/0 | |
| Научна област, **НО**: | | Електротехничко и рачунарско инжењерство | |
| Научна дисциплина, **НД**: | | Примењене рачунарске науке и информатика | |
| Предметна одредница/Кqучне 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 *securaDSL* за моделовање целокупне структуре *Spring* апликација са безбедносном конфигурацијом подржаном за три безбедносна механизма: основну аутентификациjу, *JWT* и *OAuth2.0.* Поред наменског језика, развијени су генератори у циљу генерисања извршивог кода на основу модела креираног помоћу језика *securaDSL*. За развој наменског језика i генератора коришћено је окружење *Eclipse Modeling Framework.* | |
| Датум прихватања теме, **ДП**: | |  | |
| Датум одбране, **ДО**: | |  | |
| Чланови комисије, **КО**: | Председник: |  |
| Члан: |  | Потпис ментора |
|  | Члан, ментор: | др Владимир Димитриески, доцент |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 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, Assistant 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 / Faculty of Technical Sciences | |
| Publication place, **PP**: | | Novi Sad, Faculty of Technical Sciences, Dositeja Obradovica sq. 6 | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | 6/77/22/18/22/0/0 | |
| 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 textal 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 the 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: |  |
|  | Member: |  | Menthor's sign |
|  | Member, Mentor: | Vladimir Dimitrieski, PhD, Assistant Professor |  |

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

| Студијски програм: | **Рачунарство и аутоматика** |
| --- | --- |
| Руководилац студијског програма: | др Мирна Капетина, ванредни професор |

*(Податке уноси предметни наставник - ментор)*

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | **Јелена Хрњак** | Број индекса: | **Е2 64/2022** |
| Област: | **Електротехничко и рачунарско инжењерство** | | |
| Ментор: | **др Владимир Димитриески, доцент** | | |
| НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА МАСТЕР РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:   * 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 *Spring*** |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| * Проучити основне концепте којисе користе прииком моделовања *Spring* апликација са безбедносном конфигурацијом * Анализирати постојеће језике за моделовање веб апликација * Имплементирати наменски језик и алат за моделовање *Spring* апликација са безбедносном конфигурацијом * Имплементирати генераторе који на основу модела описаног датим језиком генерише извршиви код *Spring* апликације са безбедносном конфигурацијом * Анализирати оправданост креирања наведеног решења за моделовање *Spring* апликација са безбедносном конфигурацијом и предложити даље правце развоја језика и алата |

|  |  |
| --- | --- |
| Руководилац студијског програма: | Ментор рада: |
|  |  |

|  |
| --- |
| Примерак за: - Студента; - Ментора |

**Садржај**

[1. Увод...............................................................................................................................1](#_Toc144365491)

[1.1 Структура рада….................................................................................................2](#_Toc144365492)

[2. Теоријске основе моделима вођеног развоја софтвера, наменских језика и](#_Toc144365493)

[безбедносних аспеката у радном оквиру](#_Toc144365493) *[Spring](#_Toc144365493)*[......................................................3](#_Toc144365493)

[2.1 Теоријске основе моделима вођеног развоја софтвера и наменских језика ..3](#_Toc144365494)

[2.1.1 Опис технологија коришћених за развој наменског језика *securaDSL*…..4](#_Toc144365495)

[2.1.2 Опис технологија коришћених за развој генератора…...............................5](#_Toc144365496)

[2.2 Преглед безбедносних механизама у *Spring* апликацијама…........................5](#_Toc144365497)

[2.2.1 Основна аутентификација…..........................................................................5](#_Toc144365498)

[2.2.2 Стандард *JSON* веб токен…...........................................................................6](#_Toc144365499)

[2.2.3 Стандард *Oauth*...............................................................................................6](#_Toc144365500)

[3. Преглед постојећег стања у области….....................................................................7](#_Toc144365501)

[3.1 Недостаци код постојећих начина за моделовање веб апликација…............7](#_Toc144365502)

[4. Наменски језик за подршку брзог успостављања конфигурације безбедносних](#_Toc144365503)

[аспеката у радном оквиру](#_Toc144365503) *[Spring](#_Toc144365503)*[…...........…...........…...........…...........…...........…9](#_Toc144365503)

[4.1 Апстрактна синтакса…...........…...........…...........…...........…...........…............9](#_Toc144365504)

[4.1.1 Концепт *Application*…...........…...........…...........…...........…...........…........11](#_Toc144365505)

[4.1.2 Концепт Database…...........…...........…...........…...........…...........…............12](#_Toc144365506)

[4.1.3 Концепт *Entity*…...........…...........…...........…...........…...........….................13](#_Toc144365507)

[4.1.4 Концепт *Attribute*…...........…...........…...........…...........…...........…............14](#_Toc144365508)

[4.1.5 Концепт *User*…...........…...........…...........…...........…...........…...........…....15](#_Toc144365541)

[4.1.6 Концепт *Role*…...........…...........…...........…...........…...........…...........….....16](#_Toc144365542)

[4.1.7 Концепт *RoleInstance*…...........…...........…...........…...........…...........…......16](#_Toc144365543)

[4.1.8 Концепт *Endpoint*…...........…...........…...........…...........…...........…............17](#_Toc144365544)

[4.1.9 Концепт *Controller*…...........…...........…...........…...........…...........…..........18](#_Toc144365545)

[4.1.10 Концепт *Authentication*…...........…...........…...........…...........…..................20](#_Toc144365546)

[4.1.11 Концепт *Security*…...........…...........…...........…...........…...........…..............21](#_Toc144365547)

[4.1.12 Концепт *BasicAuthentication*…...........…...........…...........…...........….........21](#_Toc144365548)

[4.1.13 Концепт *JWT*…...........…...........…...........…...........…...........…...........…....21](#_Toc144365549)

[4.1.14 Концепт *Claim*…...........…...........…...........…...........…...........…...........…..22](#_Toc144365550)

[4.1.15 Концепт *OAuth2*…...........…...........…...........…...........…...........…..............25](#_Toc144365551)

[4.1.16 Концепт *Provider*…...........…...........…...........…...........…...........…............26](#_Toc144365552)

[4.2 Конкретна синтакса…...........…...........…...........…...........…...........…............26](#_Toc144365553)

[4.2.1 Граматика наменског језика *securaDSL*…...........…...........…...........…......26](#_Toc144365554)

[4.3 Примери модела описаних наменским језиком *securaDSL*….......................30](#_Toc144365555)

[4.3.1 Пример модела](#_Toc144365556) *[Spring](#_Toc144365556)* [веб апликације са конфигурисаном основном](#_Toc144365556)

[аутентификацијом…...........…...........…...........…...........…...........…...........31](#_Toc144365556)

[4.3.2 Пример модела](#_Toc144365557) *[Spring](#_Toc144365557)* [веб апликације са конфигурисаним безбедносним](#_Toc144365557)

[механизмом](#_Toc144365557) *[JWT](#_Toc144365557)*[…...........…...........…...........…...........…...........….............32](#_Toc144365557)

[4.3.3 Пример модела](#_Toc144365558) *[Spring](#_Toc144365558)* [веб апликације са конфигурисаним](#_Toc144365558)

[безбедносним механизмом](#_Toc144365558) *[OAuth2.0](#_Toc144365558)*[..........…...........…...........…...............34](#_Toc144365558)

[5. Генерисање *Spring* веб апликација са безбедносном конфигурацијом…............35](#_Toc144365559)

[5.1 Генератор статичких датотека…...........…...........…...........…...........…..........35](#_Toc144365560)

[5.2 Генератор општих конфигурационих фајлова…...........…...........…..............37](#_Toc144365561)

[5.3 Генератор слоја који моделује податке из базе података …..........................42](#_Toc144365562)

[5.4 Генератор слоја за обраду захтева корисника…...........…...........…..............50](#_Toc144365563)

[5.5 Генератор конфигурационих фајлова за основну аутентификацију…........52](#_Toc144365564)

[5.6 Генератор конфигурационих фајлова за стандард *JWT* …...........…..............56](#_Toc144365565)

[5.7 Генератор конфигурационих фајлова за стандард *OAuth2.0*….....................66](#_Toc144365566)

[5.8 Тестирање генерисаних](#_Toc144365567) *[Spring](#_Toc144365567)* [апликација са конфигурисаном](#_Toc144365567)

[основном аутентификацијом или стандардом](#_Toc144365567) *[JWT](#_Toc144365567)*[…...........…...........…......67](#_Toc144365567)

[5.9 Тестирање генерисаних](#_Toc144365568) *[Spring](#_Toc144365568)* [апликација са конфигурисаним](#_Toc144365568)

[стандардом](#_Toc144365568) *[OAuth2.0](#_Toc144365568)*[…...........…...........…...........…...........…...........…..........69](#_Toc144365568)

[6. Закључак…...........…...........…...........…...........…...........…...........…...........…........71](#_Toc144365569)

[Скраћенице…...........…...........…...........…...........…...........…...........…...........…............73](#_Toc144365570)

[Литература…...........…...........…...........…...........…...........…...........…...........….............75](#_Toc144365571)

[Биографија…...........…...........…...........…...........…...........…...........…...........….............77](#_Toc144365572)

# 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 сигурних веб апликација.

Сигурност у веб апликацијама представља скуп мера и механизама 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 безбедноснти, представља основу ефикасне заштите. Аутентификација се односни на процес потврђивања идентитета корисника или ентитета који приступа систему. Ауторизација представља контролу приступа корисника или ентитета при приступању одређеним ресурсима или функционалностима система.

*Java* [1]представља један од најпопуларнијих програмских језика [2]. Радни оквир *Spring* [3]чини развој апликација у програмском језику *Java* бржим, једноставнијим и сигурнијим што га чини најпопуларнијим радним оквиром за овај програмски језик [4].

Међутим, обезбеђивање одговарајуће заштите за апликације у радном оквиру *Spring* представља сложен и временски захтеван процес, те је самим тим подложан грешкама. Обзиром да се безбедносни апекти 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 који задовољавају потребе и захтеве корисника на основу дефинисања основних параметара апликације. Аутоматизација 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 се односе на базу података, слој за моделовање података из базе, 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 овај језик, 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.

## Структура рада

Након уводног поглавља следи поглавње ,,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.

# 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 механизама у *Spring* апликацијама.

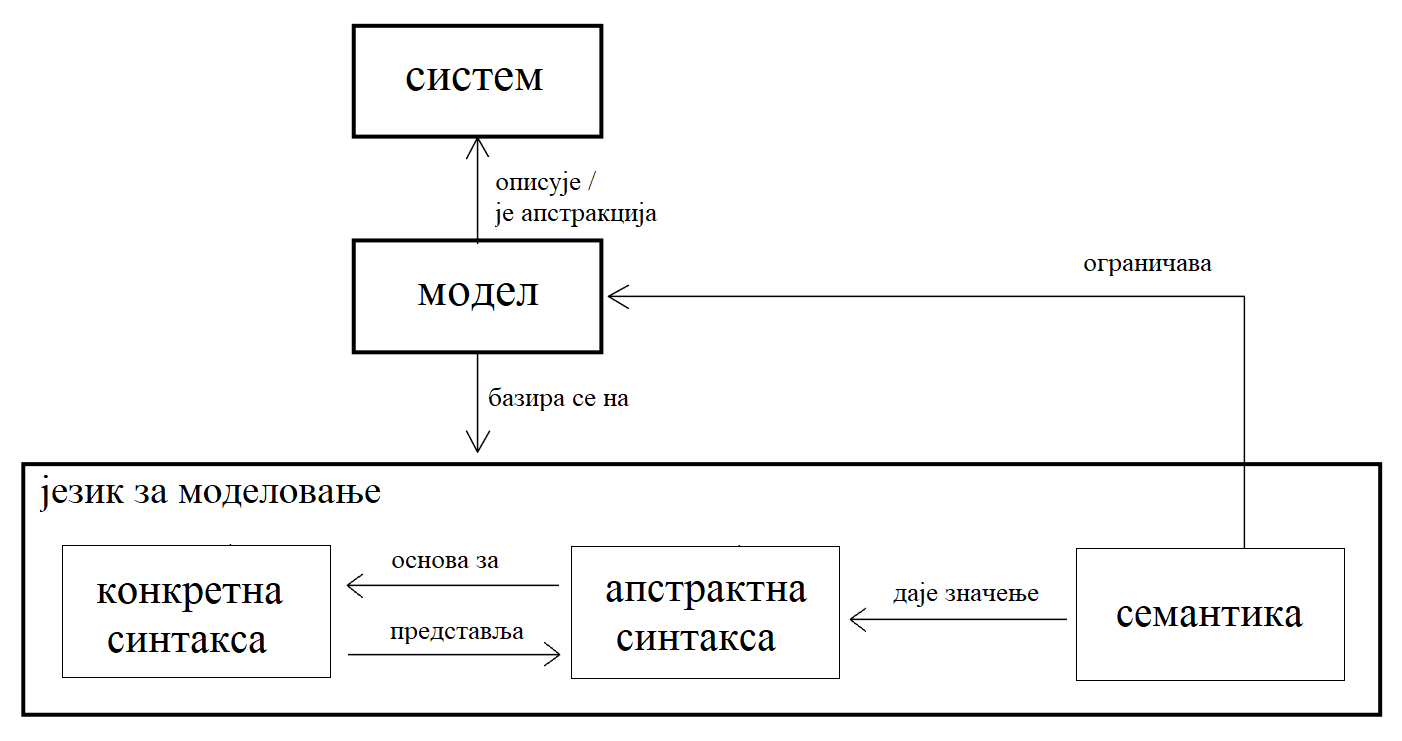
## Теоријске основе моделима вођеног развоја софтвера и наменских језика

Модел представља поједностављену слику, односно 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 релевантни делови.

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 на основу модела генерише извршиви код [5].

Сваки модел је заснован на језику, односно формализму. Језик за моделовање прецизно дефинише синтаксу, односно нотацију модела и његову семантику, тј. значење (слика 2.1) [6]. Синтаксу језика чине апстракнта 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 значење дефинисаних елемената, различитих начина за њихово комбиновање и представља ограничења за моделе који могу бити приказани коришћењем језика.

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 дизајнирани 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 и његови елементи су описани у поглављу „Наменски језик за подршку брзог успостављања конфигурације безбедносних аспеката у радном оквиру *Spring“.*



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

При развоју софтвера вођеним моделом, кључан корак представљају трансформације. 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 имплементиране су трансформације модела у текст, који представља извршиви код. Трансформације су постигнуте развојем генератора описаних у поглављу „Генерисање *Spring* веб апликације са безбедносном конфигурацијом”.

### 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.* На основу мета-модела описаног помоћу овог језика, *EMF* омогућава генерисање кода који имплеменитра 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-*a под називом *Eclipse OCL* [8]*.* Након тога, на основу апстрактне креирана је конкретна синтакса помоћу радног оквира *Xtext* [9]. Овај радни оквир се користи за развој програмских и наменских језика. Омогућава креирање свих неопходних алата за употребу језика на основу описаних елемената језика.

### Опис технологија коришћених за развој генератора

За развој генератора који омогућавају трансформацију модела у извршиви код, коришћен је програмски језик *Java* и језик *Xtend* [10]. *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.

## Преглед безбедносних механизама 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* [11]и *Spring Security* [12]. *Spring Boot* је модул којиомогућава 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* [13]*, MySQL* [14] и *Oracle Database* [15]*. 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 фукнционалности, 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.

Иако радни оквир *Spring* пружа решење за аутентификацију и ауторизацију помоћу модула *Spring Security,* оваква заштита веб апликација се не препоручује уколико постоји потреба за вишим нивоом безбедности. Због тога, подржана су три додатна безбедносна механизма: основна аутентификација (енгл. *Basic Authentication*) [16] , стандард *JSON* веб токен [17] и стандард *OAuth* [18].

### Основна аутентификација

Основна аутентификација (енгл. *Basic Authentication*) представља метод аутентификације у ком се корисник идентификује помоћу корисничког имена (или неког другог идентификационог параметра) и лозинке. Уколико су унешени подаци валидни, аутентификација корисника се сматра успешном. Ова два параметра се често прослеђују као обичан текст или се шифрују помоћу неког механизма шифровања, као што је *Base64*. Овај метод може бити ограничен у погледу ауторизације и заштите података, те се у случају потребе за вишим нивоима безбедности препоручује коришћење додатних безбедносних механизама.

### Стандард *JSON* веб токен

*JSON* вебтокен (*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.

### Стандард *OAuth*

*OAuth* (скраћено од *Open Authorization*)представља стандард за доделу права приступа који омогућава корисницима да доделе овлашћењa 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 кaо 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.*

# Преглед постојећег стања у области

Савремене методе развоја сотфвера теже брзини, безбедности и ефикасности. Тежи се смањењу потребе за ручно писаним кодом, што је довело до повећаног броја решења за генерисање различитих видова и нивоа софтвера.

*Spring Initializer [19]* је користан алат за брзо креирање основне структуре *Spring* апликације. Омогућава дефинисање основних мета-података 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 [20] је описан наменски језик *Silverа* 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 пограмском језику и радном оквиру, с обзиром да су коришћени спољни генератори.

Алат *MicroBuilder* представљен у раду [21] 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* [22] 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 механизама.

## 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 [19] која су корисна при развоју почетног пројекта, али захтевају ручно писање кода за иницијализацију већине елемената апликације. Нека решења [20,21] нису погодна за генерисање монолитних апликација, иако су се показала ефикасно за развој микросервисне архитектуре. Конфигурација безбедносних асплеката помоћу алата [22] може бити комплексна због сложености саме спецификације. Разумевање различитих параметара при дефинисању може захтевати додатно време, те 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.

# 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 односне на саму апликацију, односно њене мета податке, што омогућава брзу конфигурацију апликације. Параметри базе података који омогућавају складиштење и руковање подацима представљају другу целину, док следећа целину чини слој који моделује податке из базе, где је могуће дефинисање ентитета који описују структуру података у бази. 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 (*Basic Authentication),*
* аутентификација помоћу *JSON Web* токена (*JWT*) и
* *Open Authorization* (*OAuth2.0*)*.*

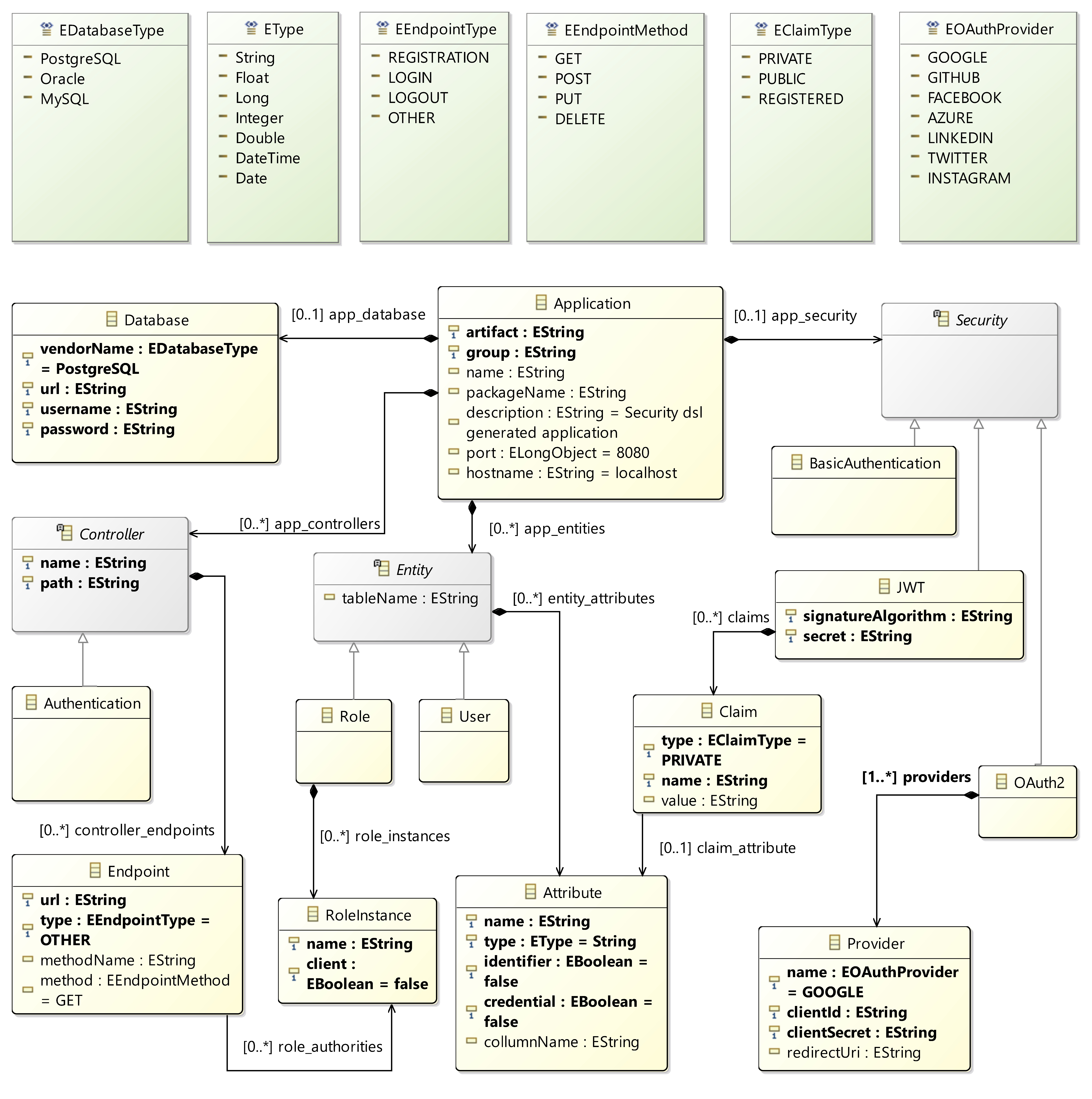
Овим се обезбеђује флексибилност и могућност одабира оптималног безбедносног механизма у зависности од потреба и захтева система.

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.*

## Апстрактна синтакса

Апстрактна синтаска 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 наведени курзивом.



Слика 4.1 – Апстракнта 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.*

### Концепт *Application*

Коренски концепт апстрактне синтаксе *Application* садржи обележја (табела 4.1) која се односе на основне параметре неопходне за иницијализацију апликације. Табела 4.2 приказује асоцијације концепта *Application* са другим концептима.

|  |  |  |  |
| --- | --- | --- | --- |
| Назив обележја | Тип податка | Обавезно | Опис |
| *artifact* | *EString* | да | Назив артефакта |
| *group* | *EString* | да | Назив групе који представља организацију, компанију или тим који развија апликацију |
| *name* | *EString* | не | Назив апликације |
| *packageName* | *EString* | не | Назив пакета у ком се налази изворни код апликације |
| *description* | *EString* | не | Опис апликације |
| *port* | *ELongObject* | не | Порт на ком апликација ослушкује захтеве |
| *hostname* | *EString* | 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

|  |  |  |  |
| --- | --- | --- | --- |
| Назив асоцијације | Референцирани концепт | Кардиналитет | Опис |
| *app\_database* | *Database* | 0..1 | База података апликације |
| *app\_entitites* | *Entity* | 0..\* | Eнтитети у апликацији |
| *app\_controllers* | *Controller* | 0..\* | Контролери у апликацији |
| *app\_security* | *Security* | 0..1 | Безбедносна конфигурација апликације |

Табела 4.2 – Асоцијације концепта *Application*

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 *OCL* ограничењем *hasDatabawseForEntity* (листинг 4.2).

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

**invariant** hasDatabaseForEntity('An application must have a database defined if it has entities!'):

**self**.*app\_entities* -> *isEmpty*() **or** **not** **self**.*app\_database*->*isEmpty*();

### Концепт Database

Апстрактни концепт *Database* обухвата параметре за повезивање апликације са базом података. Тип података *EDatabaseType* представља енумерацију заназив 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 |
| *vendorName* | *EDatabaseType* | да | Систем за управљање базама података, при чему су могуће вредности *PostgreSQL, MySQL* и *Oracle* |
| *url* | *EString* | да | Конекциони стринг за повезивање са базом података |
| *username* | *EString* | да | Корисничко име за приступ бази података |
| *password* | *EString* | да | Лозинка за приступ бази података |

Табела 4.3 – Обележја концепта *Database*

### 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 |
| *tableName* | *EString* | 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 ентиети 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*.*toLower*());

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

|  |  |  |  |
| --- | --- | --- | --- |
| Назив асоцијације | Референцирани концепт | Кардиналитет | Опис |
| *entity\_attributes* | *Attribute* | 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 *uniqueCollumnName* (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 окциру ентитета имају јединствене називе

**invariant** uniqueCollumnName('Column names must be unique if defined!'):

**self**.*entity\_attributes* -> *exists*(a | *a*.*collumnName* <> **null**) **implies**

**self**.*entity\_attributes* -> *isUnique*(a | *a*.*collumnName*);

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

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

**invariant** onlyOneIdentifier('Entity must have exactly one identifier attribute!'):

**self**.*entity\_attributes* -> *size*() > 0 **implies**

**self**.*entity\_attributes* -> *select*(a | *a*.*identifier*) -> *size*() = 1;

### 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 |
| *name* | *EString* | да | Назив обележја |
| *type* | *EType* | да | Тип обележја, при чему су могуће вредности *String, Float, Long, Integer, Double, DateTime* и *Date* |
| *identifier* | *EBoolean* | да | Да ли је обележје идентификатор? |
| *credential* | *EBoolean* | да | Да ли је обележје идентификациони параметар? |
| *collumnName* | *EString* | не | Назив колоне у табели базе података који се односи на обележје |

Табела 4.6 – Обележја концепта *Attribute*







### 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

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

**invariant** otherEntitiesDoesntHaveCredential('Entities other than User

cannot have a credential attribute!'):

**not** **self**.*oclIsTypeOf*(*User*) **implies** **self**.*entity\_attributes* -> *select*(a | *a*.*credential*) -> *isEmpty*();

### Концепт *Role*

Концепт *Role* наслеђује концепт *Entity* и моделује ентитет који се односи на улоге апликације. Улоге могу бити имплементиране 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 конфликата приликом додавања нових улога.

|  |  |  |  |
| --- | --- | --- | --- |
| Назив асоцијације | Референцирани концепт | Кардиналитет | Опис |
| *role\_instances* | *RoleInstance* | 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

### 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 |
| *name* | *EString* | да | Назив конкретне улоге |
| *client* | *EBoolean* | да | Ознака да ли је улога клијент. Уколико је вредност обележја *false* улога је администраторска |

Табела 4.8 – Обележја концепта *RoleInstance*

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

Табела 4.9 – Обележја концепта *Endpoint*

|  |  |  |  |
| --- | --- | --- | --- |
| Назив асоцијације | Референцирани концепт | Кардиналитет | Опис |
| *role\_authorities* | *RoleInstance* | 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 ‘/’

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

**invariant** uniqueRoleAuthorities('Role authorities must be unique for each endpoint!'):

**self**.*role\_authorities* -> *isUnique*(r | *r*.*name*);

### 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 |
| *name* | *EString* | да | Назив контролера |
| *path* | *EString* | 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 апликацјие морају да буду јединствени

**invriant** 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 караткером ‘/’

Листинг 4.18 – Ограничење назива за контролере

**invariant** controllerNotNamedUserRole('Controller names cannot be "User" or "Role"!'):

**not** *Controller*.*allInstances*() ->

*exists*(c | *c*.*name*.*toLower*() = 'user' **or** *c*.*name*.*toLower*() = 'role');

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). Методеза 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 |
| *controller\_endpoints* | *Endpoint* | 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 котнролеру за аутентификацију

### Концепт *Authentication*

Концепт *Authenticatiion* 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).

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

**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 4.24 – Могуће је постојање највише једног контролера за аутентификацију

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

**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;

### 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.

### 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 конепту *Role*

### Концепт *JWT*

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

|  |  |  |  |
| --- | --- | --- | --- |
| Назив обележја | Тип податка | Обавезно | Опис |
| *signatureAlgorithm* | *EString* | да | Алгоритам који се користи за потписивање *JWT* токена |
| *secret* | *EString* | да | Тајни кључ који се користи за потписивање *JWT* токена |

Табела 4.13 – Обележја концепта *JWT*

|  |  |  |  |
| --- | --- | --- | --- |
| Назив асоцијације | Референцирани концепт | Кардиналитет | Опис |
| *claims* | *Claim* | 0..\* | Тврдње у оквиру *JWT* 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*.*oclIsTypeOf*(*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*.*oclIsTypeOf*(*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*

### Концепт *Claim*

Концепт *Claim* представља трврдње које садржи *JWT* токен (табела 4.15), односно податке о кориснику, стању апликације или самом токену које се преносе путем токена. Постоје три типа тврдњи: регистроване (предефинисане), приватне и јавне. Поред типа, тврдња има назив који представља идентификатор пода тка 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 |
| *type* | *EClaimType* | да | Тип тврдње, при чему су могуће вредности *PRIVATE, PUBLIC* и *REGISTERED* |
| *name* | *EString* | да | Назив трвдње |
| *value* | *EString* | не | Вредност тврдње |

Табела 4.15 – Обележја концепта *Claim*

|  |  |  |  |
| --- | --- | --- | --- |
| Назив асоцијације | Референцирани концепт | Кардиналитет | Опис |
| *claim\_attributes* | *Attribute* | 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 *еxpirationTime, 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

### Концепт *OAuth2*

*OAuth2* концепт моделује *Open Authorization* конфигурацију 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 |
| *providers* | *Provider* | 1..\* | Конфигурисани провајдери у апликацији |

Табела 4.17 – Асоцијације концепта *OAuth2*

**invariant** doesntHaveUserForOauth('OAuth2 authentication requires no User entities!'):

*Entity*.*allInstances*() -> *select*(e | *e*.*oclIsTypeOf*(*User*)) -> *size*() = 0;

**invariant** doesntHaveRoleForOauth('OAuth2 authentication requires no Role entities!'):

*Entity*.*allInstances*() -> *select*(e | *e*.*oclIsTypeOf*(*Role*)) -> *size*() = 0;

**invariant** doesntHaveAuthControllerForOauth('OAuth2 authentication requires no Authentication controller!'):

*Controller*.*allInstances*() ->

*select*(e | *e*.*oclIsTypeOf*(*Authentication*)) -> *size*() = 0;

Листинг 4.34 – Није могуће дефинисани 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

### 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 |
| *name* | *EString* | да | Назив провајдера |
| *clientId* | *EString* | да | Идентификациони број клијента додељен од стране провајдера |
| *clientSecret* | *EString* | да | Тајни кључ клијента додељен од стране провајдера за сигурносну контролу приликом комуникације са провајдером |
| *redirectUri* | *EString* | не | Адреса за преусмеравање након успешне аутентификације |

Табела 4.18 – Обележја концепта *Provider*

## 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.

### 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  ('collumnName:' collumnName=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*

## Примери модела описаних наменским језиком *securaDSL*

Ово поглавље пружа преглед претходно описане конкретне синтаксе кроз примере за сваки од подржаних безбедносних механизама. 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 податакадефинишу се након кључне речи *database,* ентитети након кључне речи *entity*, контролери након кључне речи *controller*, а безбедносни аспекти након кључне речи *security.*

### 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 онсовном аутентификацијом у радном оквиру *Spring*

### 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**  **collumnName:** "username"  },  {  **name:** "firstName"  **type:** **String**  **collumnName:** "first\_name"  },  {  **name:** "lastName"  **type:** **String**  **collumnName:** "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 : "333"},  {**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*

### 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*

# Генерисање *Spring* веб апликација са безбедносном конфигурацијом

С обзиром да се *Spring* апликација може поделити у целине које су претходно описане и да постоји више подржаних безбедносних механизама, 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 механизама, неопходно је генерисати класуу којој је описано на који начин ће се потрвђивати 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 „Примери модела описаних наменским језиком *securaDSL“.* Концепт *Application* је дефинисан исто за сваки пример, те неће бити наглашено на који се пример односи.

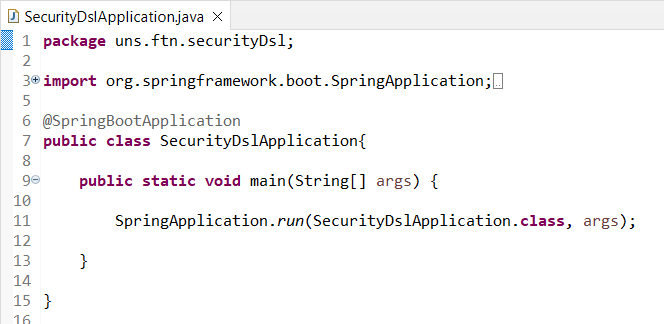
## Генератор статичких датотека

Статичке датотеке представљају датотеке за чије генерисање нису потребни парамтери из модела или су потребни само делови мета података 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 и главне датотеке за иницијализацију 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 у тестном фолдеру. Листинг 5.1 приказује шаблон за генерисање главне класе апликације, док је на листингу 5.2 приказан шаблон за генерисање главне класе тестног дела апликације. 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 тестног фолдера.

|  |
| --- |
| **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);  }  }''' |

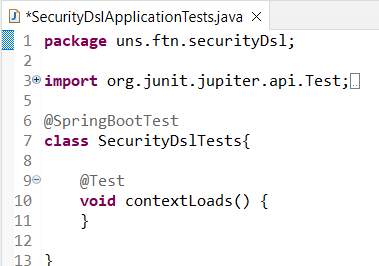
Листинг 5.1 – Шаблон за генерисање главне класе апликације



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 тестног дела



Слика 5.2 – Пример главне класе тестног дела апликације

## 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 безбедоносни механизам, те ће у зависности од одабира бити наведене спољне библиотеке. Листинг 5.3 приказује шаблон за генерисање *pom.xml* фајла*.* На слици 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*

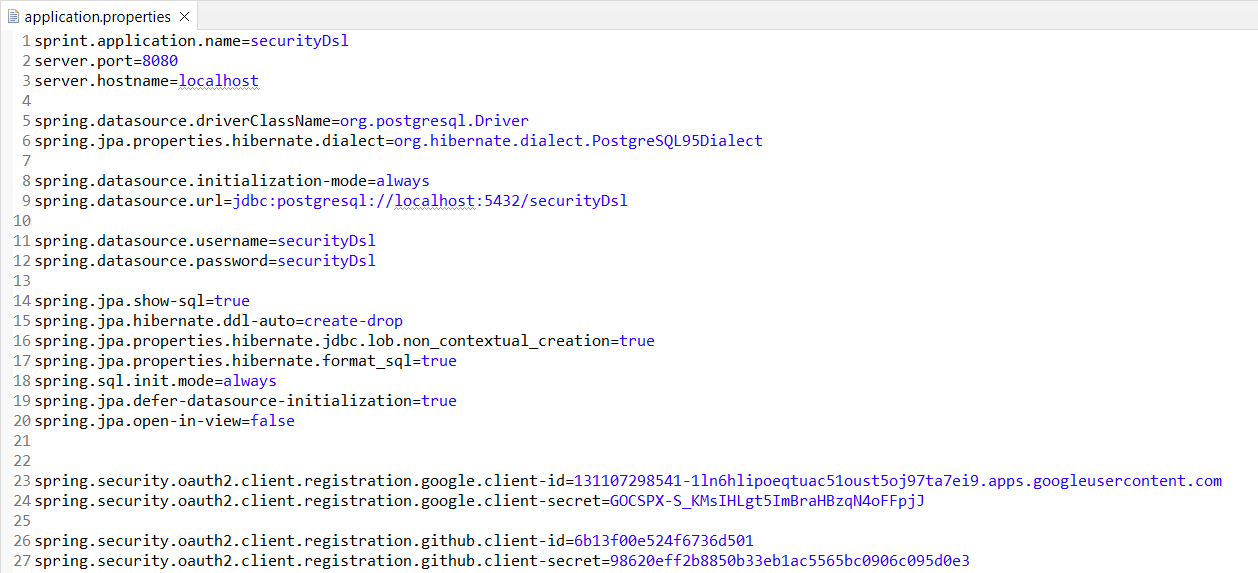
**

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 ослушкује 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 датотекте *application.properties* приказан је на листигу 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.MySQL8Dialec    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.PostgreSQL95Dialect    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.Oracle12cDialect    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*

**

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

## Генератор слоја који моделује податке из базе података

Генератор слоја који моделује податке из базе података крерира 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*).

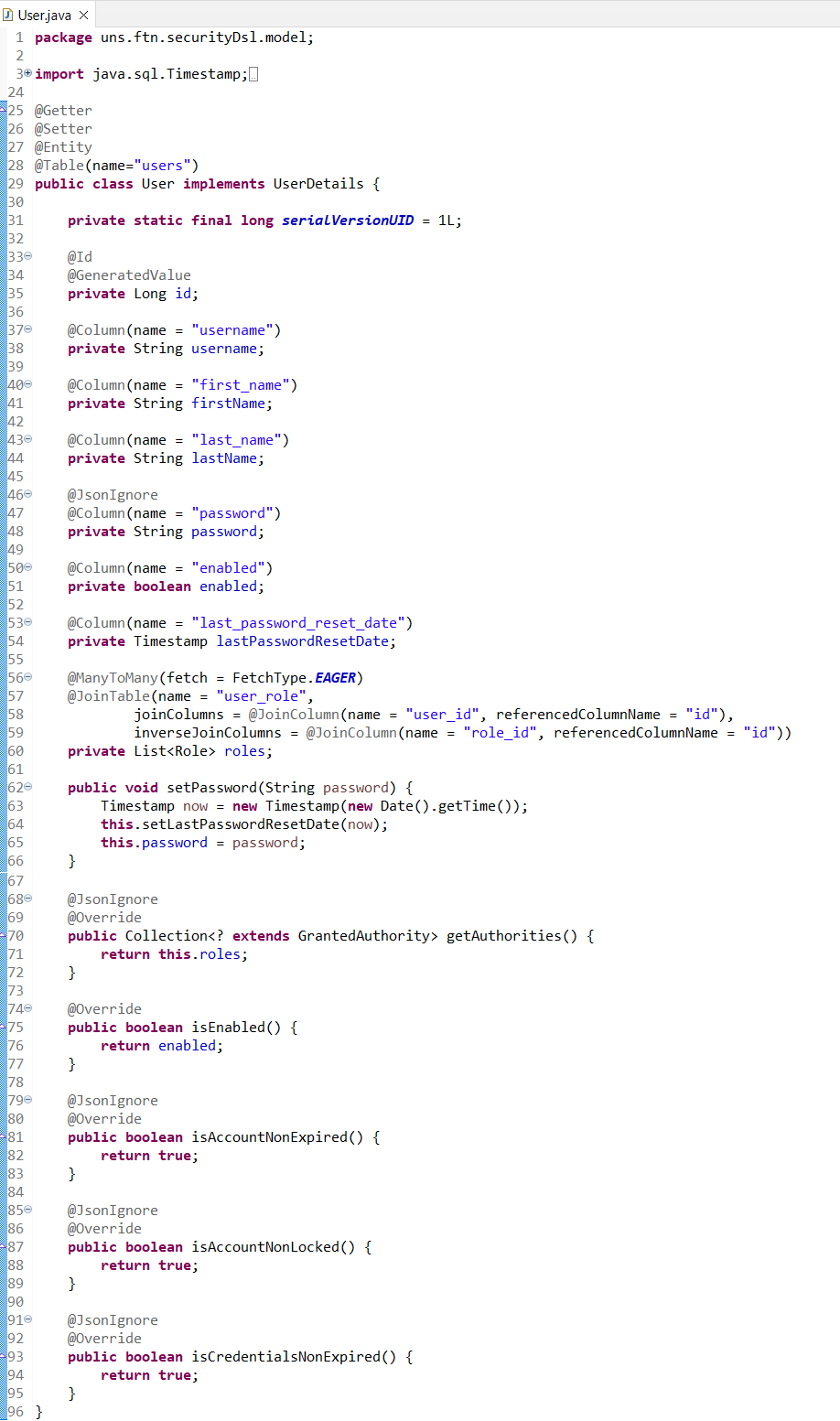
Класе модела описују структуру података у бази и креирају се на основу описа концепта *Entity,* односно концепата *User* и *Role.* Шаблон за генерисање класе која описује кориснике система приказан је на листину 5.5, док је на листингу 5.6 приказана помоћна метода која садржи шаблон за генерисање обележја 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.collumnName !== **null**»@Column(name = "«a.collumnName»")  «**ENDIF**»  private «a.type» «a.name»;    «**ENDFOR**»  '''} |

Листинг 5.6 – Шаблон за генерисање обележја класа

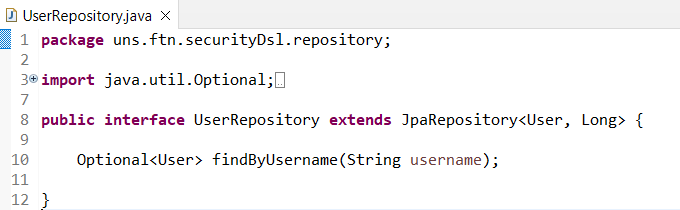


Слика 5.5 – Пример класе *User* у случају одабира безбедносног механизма *JWT*

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 ссе на листингу 5.7, док је пример генерисаног репозиторијума приказан на слици 5.6.

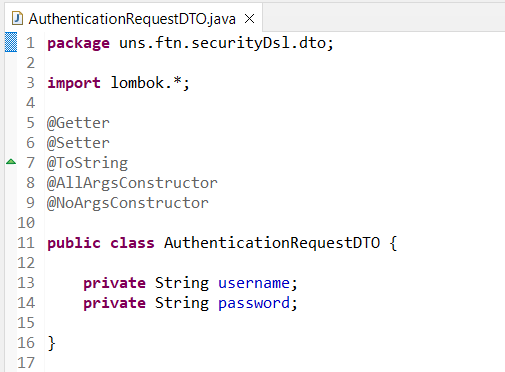
|  |
| --- |
| **def** generateUserRepository(User user)'''  package «packageName».repository;    import java.util.Optional;  import org.springframework.data.jpa.repository.JpaRepository;    import «packageName».model.User;    public interface UserRepository extends JpaRepository<User, «getIdentifier(user.entity\_attributes).type»> {    Optional<User> findBy«credentialUser.*toFirstUpper*»(String «credentialUser»);  }''' |

Листинг 5.7 – Шаблон за генерисање класе *UserRepository*

**

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.



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

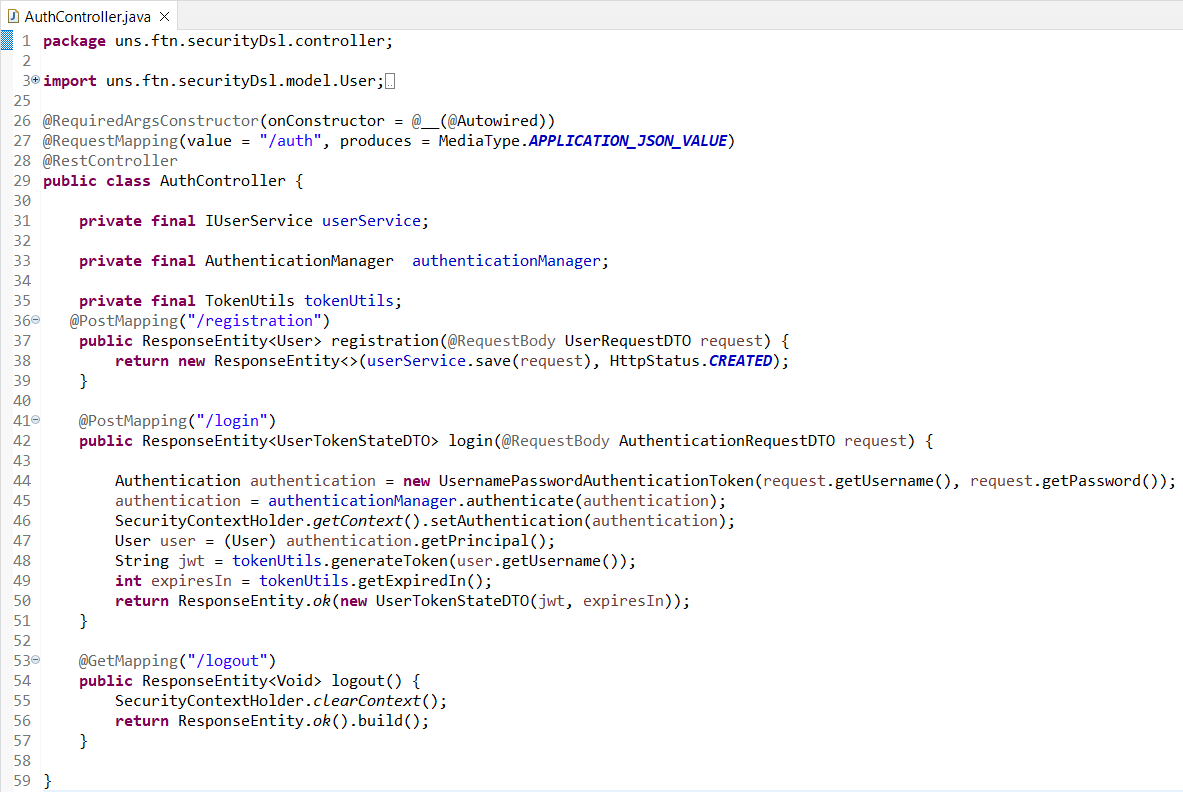
## Генератор слоја за обраду захтева корисника

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



Слика 5.8 – Пример генерисаног контролера за аутентификацију

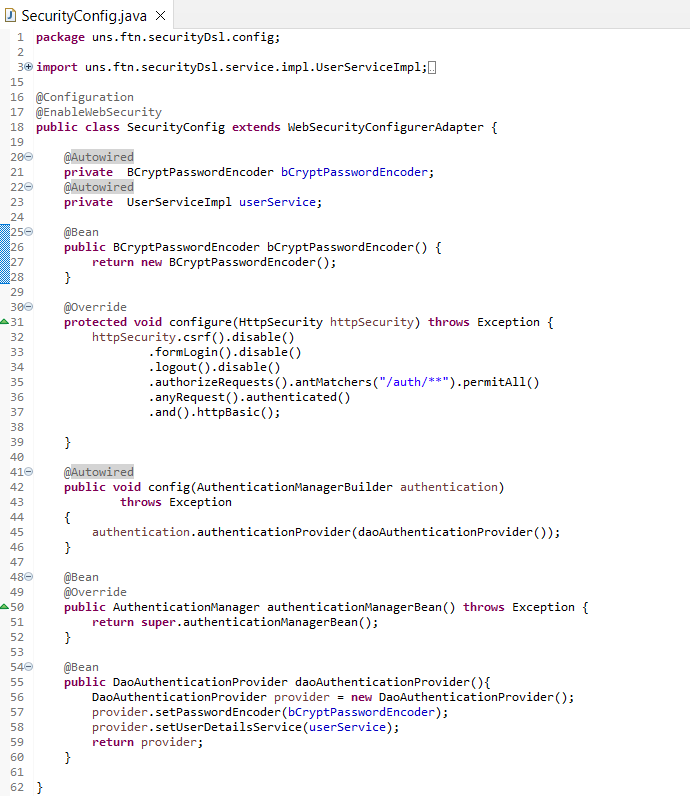
Сервиси представљају компоненте које обрађују логику захтева. У оквиру сервиса се налазе методе које врше потребне операције над подацима тако што комуницирају са одговарајућим репозиторијумом. Аутоматски се генеришу сервиси неопходни за логике захтева за регистрацију, пријаву и одјаву са система, а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

## 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 слици 5.9 представља класу *SecurityConfig* генерисану на основу модела описаног у поглављу „Пример модела *Spring* веб апликације са конфигурисаном основном аутентификацијом“.

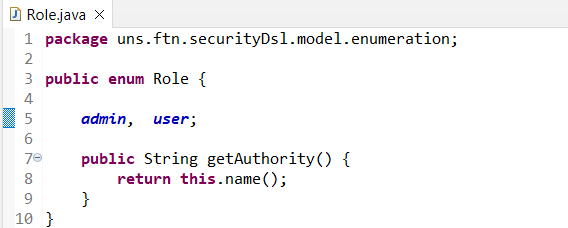
|  |
| --- |
| **def** generateSecurityConfig(Authentication authController)'''  package «packageName».config;    import «packageName».service.impl.UserServiceImpl;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.security.authentication.AuthenticationManager;  import org.springframework.security.authentication.dao.DaoAuthenticationProvider;  import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  import org.springframework.security.config.annotation.web.builders.HttpSecurity;  import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  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



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

Поред овога, генерисана је енумерација са вредностима дефинисаних улога корисника (слика 5.10).



Слика 5.10 – Енумерација за улоге корисника

Резултат извршавања свих неопходних генератора за основну аутентификацију је *Spring* апликација са безбедносном конфигурацијом основне аутентификације. Апликација је организована у пакете (слика 5.11) и садржи све класе неопходне за успешнo фунцкионисање апликације.



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

## 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 „Пример модела *Spring* веб апликације са конфигурисаним безбедносним механизмом *JWT*“.

|  |
| --- |
| **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.AuthenticationManagerBuilder;  import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;  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.WebSecurityConfigurerAdapter;  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).and()    .exceptionHandling().authenticationEntryPoint(restAuthenticationEntryPoint).and()    .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*

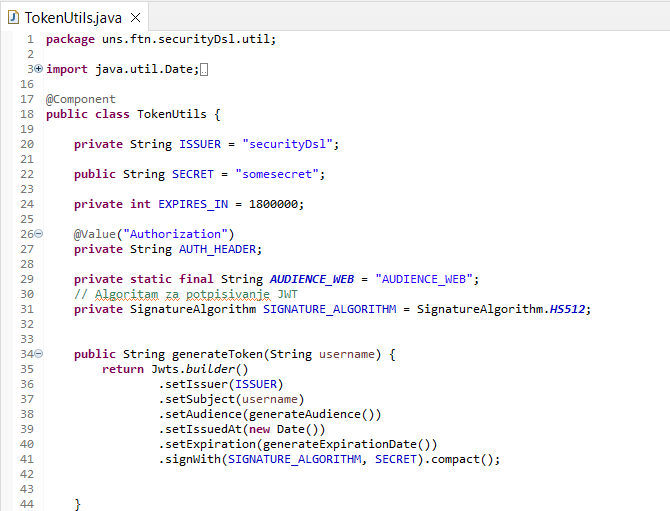
**

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»";  public 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*

**

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 аутентификован корисник. Листинг 5.12 приказује шаблон за генерисање ове класе, а пример је приказан на слици 5.14.

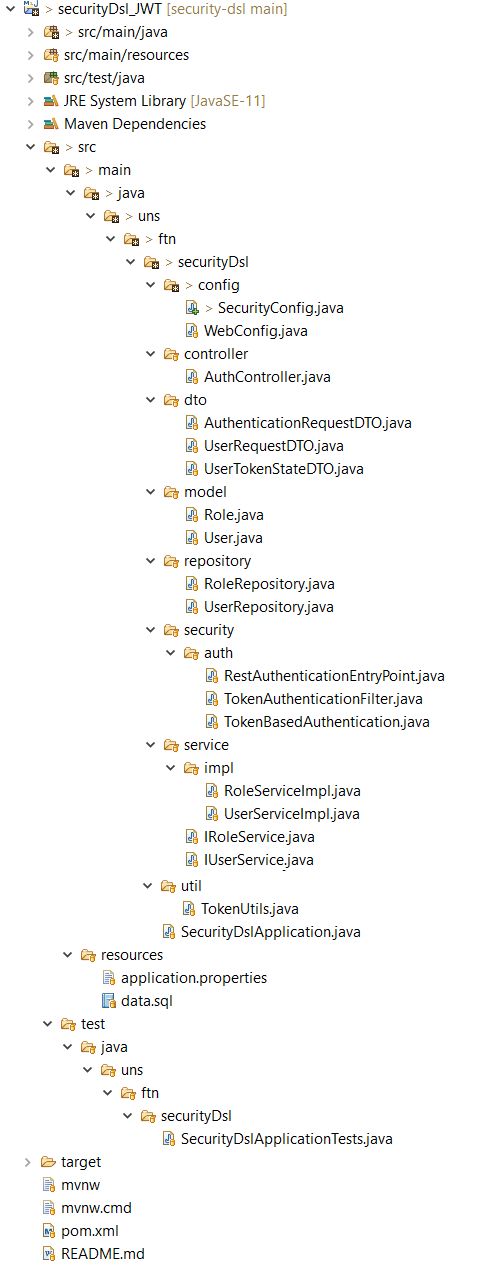
|  |
| --- |
| **def** String generateTokenAuthenticationFilter()'''  package «packageName».security.auth;    import java.io.IOException;    import javax.servlet.FilterChain;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    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 = LogFactory.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*

**

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.



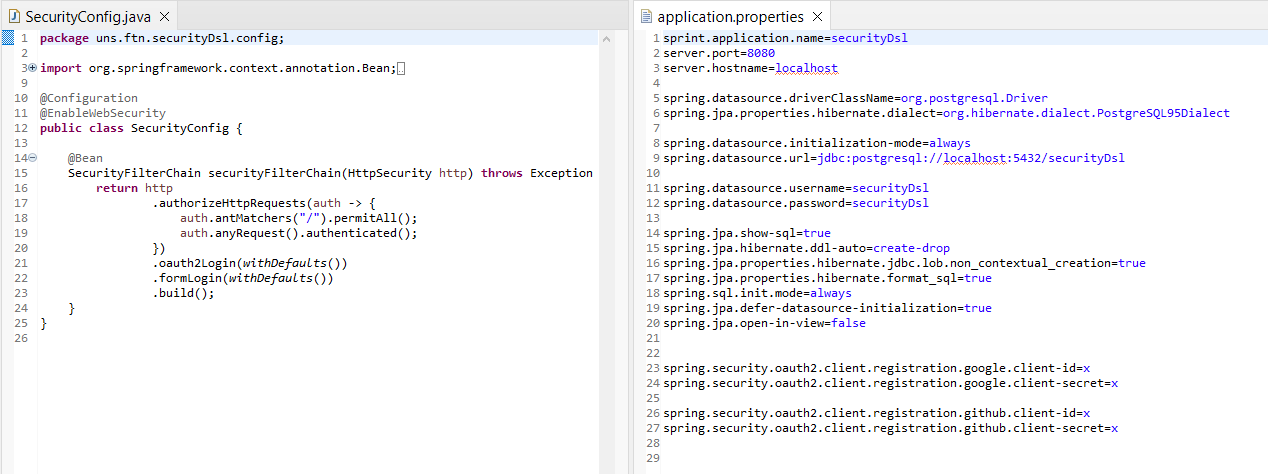
Слика 5.15 – Структура *Spring* апликације са конфигурисаним безбедносним механизмом *JWT*

## Генератор конфигурационих фајлова за стандард *OAuth2*

Примери приказани у овом поглављу сугенерисани на основу модела описаног у поглављу „Пример модела *Spring* веб апликације са конфигурисаним безбедносним механизмом *Oauth2.0*“. Листинг 5.13 приказује шаблон за генерисање класе *SecurityConfig* за конфигурацију безбедносног механизма *OAuth2.0.* Пример генерисане класе се налази на слици 5.16 са леве стране. Кључне елементе у аутенфитификацији помоћу безбедносног механизма *OAuht2.0* чине провајдери. Они се дефинишу у оквиру конфигурационе датотеке *application.properties.* На слици 5.16, са десне стране, приказана је конфигурација за провајдере *Google* и *Github.* Навођењем идентификационе ознаке и кључа корисника омогућава се аутентификација помоћу одабраних провајдера.

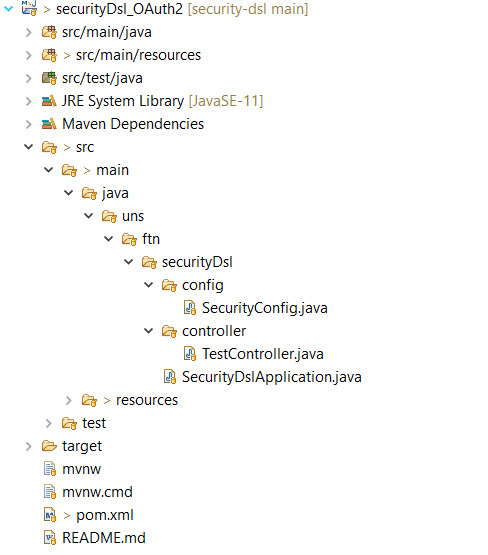
|  |
| --- |
| **def** generateSecurityConfig()'''  package «packageName».config;    import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.security.config.annotation.web.builders.HttpSecurity;  import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  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*

**

Слика 5.16 – Пример генерисане класе *SecurityConfig* и конфигурационе датотеке *application.properties* за безбедносни механизам *OAuth2.0*

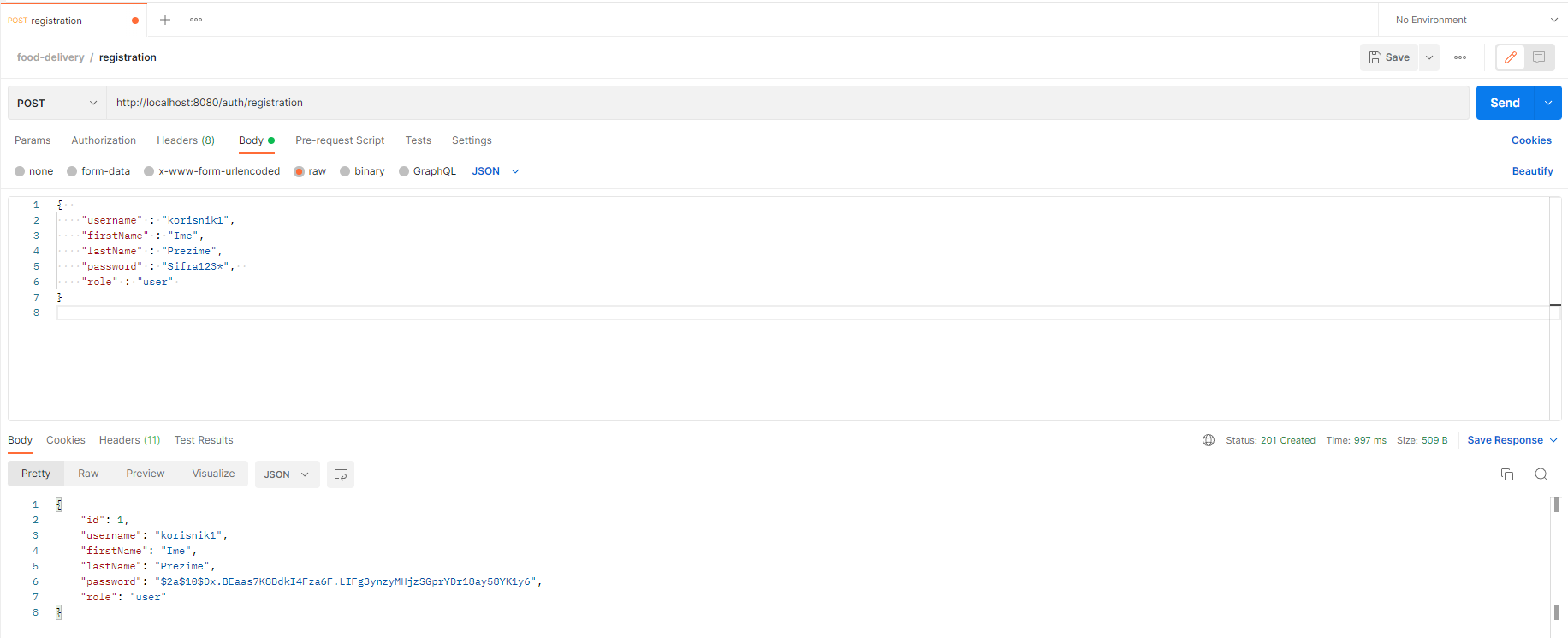
Након генерисања неопходних класа, омогућена је аутентификација помоћу конфигурисаних провајдера. Класе су организоване у пакете (слика 5.17) и генерисана *Spring* апликација је сигурна за коришћење.



Слика 5.17 – Структура *Spring* апликације са конфигурисаним безбедносним механизмом *OAuth2.0*

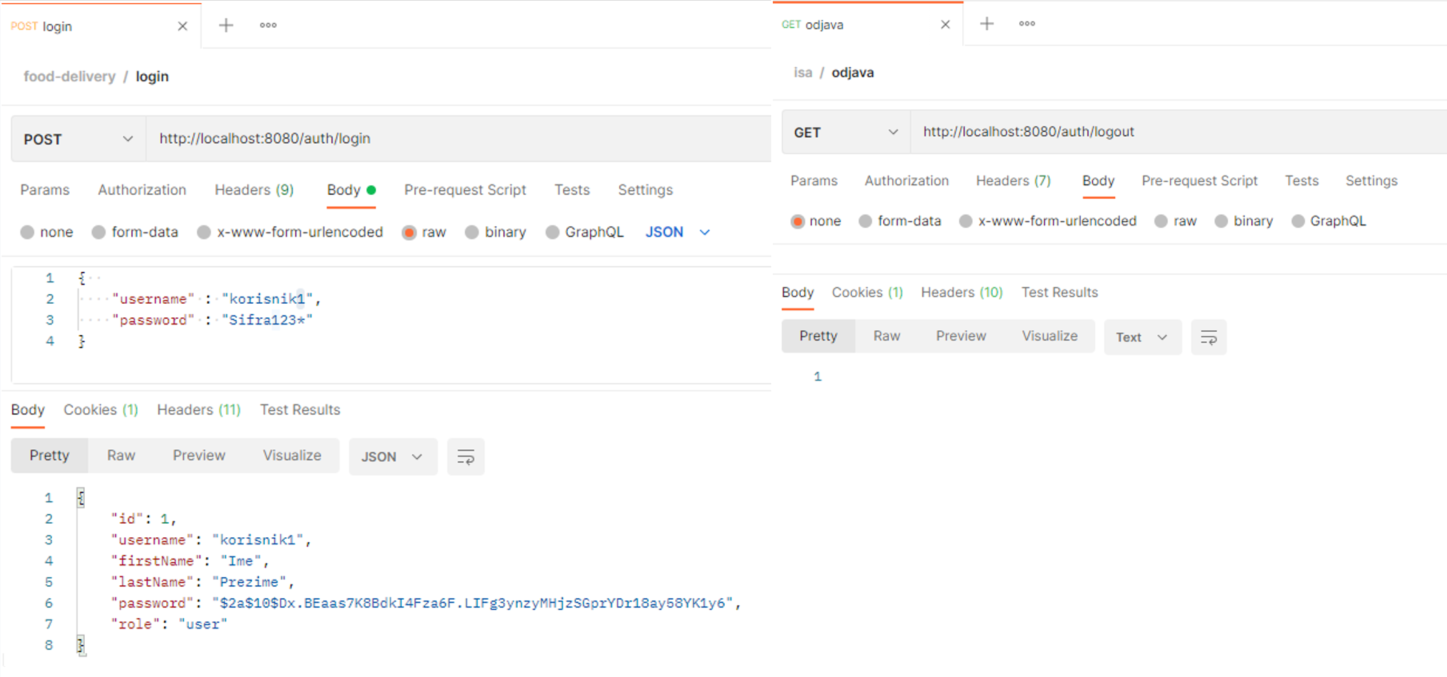
## Тестирање генерисаних *Spring* апликација са конфигурисаном основном аутентификацијом или стандардом *JWT*

*Spring* апликације са конфигурисаном основном аутентификацијом или безбедносним механизмом *JWT* тестиране су помоћу платформе *Postman.* Слика 5.18 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 одноварајућу путању. Регистрација на систем је могућа само за кориснике којима је додељена улога која је у моделу дефинисана као клијентска.



Слика 5.18 – Успешна регистрација на систем

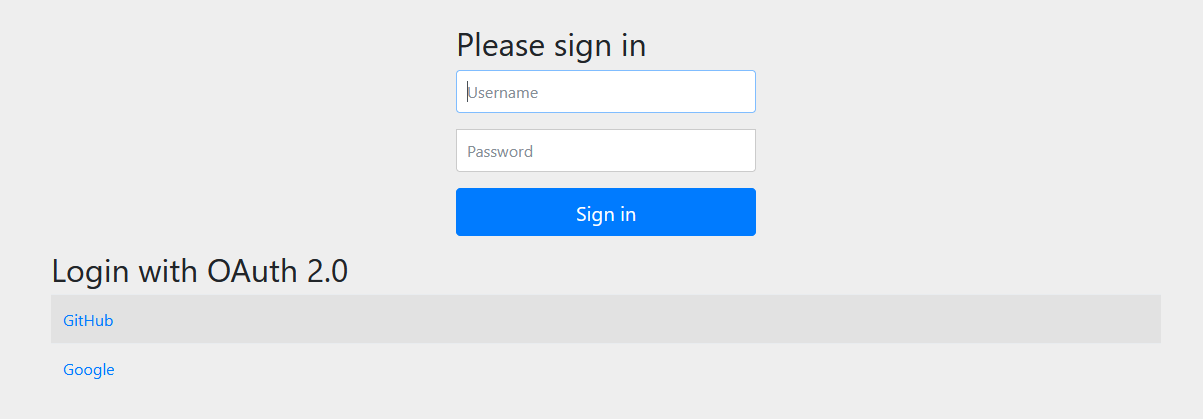
На слици 5.19 приказани су захтеви за успешну пријаву и одјаву са ситема. За успешну пријаву неопходно је унети валидну комбинацију идентификационог параметра и лозинке.



Слика 5.19 – Успешна пријава (лево) и одјава (десно) са система

## 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 5.20). Одабиром жељеног провајдера, корисник се преусмерава на страницу за аутентификацију.



Слика 5.20 – Пријава на систем у случају конфигурисаног безбедносног механизма *OAuth2.0*

# Закључак

Наменски језик *securaDSL* описан у овом раду садржи концепте неопходне за моделовање целокупне структуре *Spring* апликација, а додатно подржава конфигурацију три безбедносна механизма: основне аутентификациjе, *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. Развој 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 доприноси квалитету 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 апекта 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 намесног језика *securaDSL* и развијених генератора.

# Скраћенице

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

# Литература

1. Java Programming Language. <https://docs.oracle.com/en/java/>
2. Sujay, Vailshery, L. (2023). Most used programming langugages worldwide as of 2023. [https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/](#_Литература)
3. Spring Framework. [https://spring.io/](#_Литература)
4. Maple, S., & Binstock A. (2018). JVM Ecosystem Report 2018: About your Platform & Application. <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. <https://eclipse.dev/modeling/emf/>
8. Eclipse OCL (Object Constraint Language). <https://projects.eclipse.org/projects/modeling.mdt.ocl>
9. Xtext. <https://projects.eclipse.org/projects/modeling.tmf.xtext>
10. Xtend. <https://eclipse.dev/Xtext/xtend/documentation/index.html>
11. Spring Boot. <https://spring.io/projects/spring-boot>
12. Spring Security. <https://spring.io/projects/spring-security>
13. PostgreSQL. <https://www.postgresql.org/about/>
14. MySQL. <https://dev.mysql.com/doc/>
15. Oracle Database. <https://docs.oracle.com/en/database/>
16. Reschke, J., (2015). The ‘Basic’ HTTP authentication scheme. Internet Engineering Task Force
17. Jones, M., Bradley, J. & Sakimura. N. (2015). JSON Web Token (JWT). Internet Engineering Task Force
18. Hardt, D. Ed. (2012). The OAuth 2.0 Authorization Framework. Internet Engineering Task Force
19. Spring Initializer. <https://start.spring.io/>
20. Šuljkanović, A., Milosavljević, B., Inđić, V., Dejanović, I. (2022). Developing Microservice-Based Applications Using the Silvera Domain-Specific Language
21. Terzić, B., Dimitrieski, V., Kordić, S., Milosavljević, G., Luković, I. (2017). MicroBuilder: A Model-Driven Tool for the Specification of REST Microservice Architectures
22. *JHipster.* [*https://www.jhipster.tech/*](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 аутиматика Факултета теничких наука. Положила је све испите предвиђене планом и програмом мастер академских студија.