TARTU ÜLIKOOL

Arvutiteaduse instituut

Informaatika õppekava

Artjom Aralov

Integreeritud arenduskeskkonna IntelliJ IDEA Community ja lähtekoodiredaktori Visual Studio Code haldamisvõimaluste võrdlemine Spring Boot ja Angular raamistike jaoks

Bakalaureusetöö (9 EAP)

Juhendaja: Helle Hein

Tartu 2020

Integreeritud arenduskeskkonna IntelliJ IDEA Community ja lähtekoodiredaktori Visual Studio Code haldamisvõimaluste võrdlemine veebirakenduse loomisel Spring Boot ja Angular raamistike jaoks

Lühikokkuvõte

Spring Boot tagarakenduse raamistiku ja Angular eesrakenduse raamistiku kasutamise koos toimetamine veebiarenduses levib suure kiirusega üle maailma. Põhinedes nendele raamistikele kasutatakse veebirakenduste loomiseks kindlaid arendusvahendeid, et kiirendada tarkvara arendamise protsessi. Erilist tähelepanu tuleb pöörata kahele arendusvahendile: integreeritud arenduskeskkond IntelliJ IDEA Community ja lähtekoodiredaktor Visual Studio Code. Töö eesmärk on luua veebirakendus Spring Boot ja Angular raamistikke kasutades ning veebirakenduse arendamise käigus võrrelda vahendite haldamisvõimalusi mõlema raamistiku jaoks. Vastavalt võrdlemisele otsustada, milline vahend sobib paremini Spring Boot ja Angular raamistikega töötamiseks järgmiste kriteeriumite suhtes: Spring Initializr, Flyway andmebaaside migreerimise tööriist, MySQL relatsioonandmebaasi haldussüsteemi tugi, andmebaasi päringumeetodid, Project Lombok annotatsioonid, *IntelliSense* ja versioonikontrollisüsteem.

Võtmesõnad

Veebiarendus, Spring Boot, Angular, IntelliJ IDEA Community, Visual Studio Code, VCS

CERCS

T120 Süsteemitehnoloogia, arvutitehnoloogia

Comparison of Management Options of Integrated Development Environment IntelliJ IDEA Community and Source Code Editor Visual Studio Code by Creating a Web Application for Spring Boot and Angular Frameworks

Abstract

Interaction for web development purposes between Spring Boot backend framework and Angular frontend framework is spreading intensively around the world. To build web applications based on these frameworks, robust development tools are used to accelerate software development processes. Special attention must be paid to the following two tools: integrated development environment IntelliJ IDEA Community and source code editor Visual Studio Code. The aim of this research is to create web application by using Spring Boot and Angular frameworks and to compare management options for both frameworks during web development. Based on comparison determine which development tool suits better for web development based on the following criteria: Spring Initializr, Flyway database-migration tool, MySQL relational database management system support, database query methods, Project Lombok annotations, *IntelliSense* and Version Control System.

Keywords

Web development, Spring Boot, Angular, IntelliJ IDEA Community, Visual Studio Code, MySQL, VCS

CERCS

T120 Systems technology, computer technology

Sisukord

Sissejuhatus 7

1 Integreeritud arenduskeskkond ja lähtekoodiredaktor 9

1.1 Lähtekoodi redigeerimine 9

1.1.1 Süntaksi esiletõstmine 10

1.1.2 *IntelliSense* 10

1.1.4 Koodi silumine 11

1.2 Raamistike haldamisvõimalused 11

2 Spring Boot ja Angular raamistike struktuur ja koostoimetamine veebiarenduses 13

2.1 Spring ja Spring Boot 13

2.2 Angular 15

2.3 Spring Boot ja Angular raamistike koostoimetamise lühikirjeldus 16

3 Integreeritud arenduskeskkonna IntelliJ IDEA Community ja lähtekoodiredaktori Visual Studio Code haldamisvõimalused Spring Boot raamistiku jaoks 18

3.1 Spring Initializr 18

3.1.1 Spring Initializr (IntelliJ IDEA Community) 18

3.1.2 Spring Initializr (Visual Studio Code) 19

3.2 Andmebaaside migreerimise tööriist Flyway 19

3.2.1 Flyway (IntelliJ IDEA Community) 20

3.2.2 Flyway (Visual Studio Code) 20

3.3 Relatsioonandmebaasi haldussüsteemi MySQL tugi 20

3.3.1 Relatsioonandmebaasi haldussüsteemi MySQL tugi (IntelliJ IDEA Community) 21

3.3.2 Relatsioonandmebaasi haldussüsteemi MySQL tugi (Visual Studio Code) 21

3.4 Andmebaasi päringumeetodid 22

3.4.1 Andmebaasi päringumeetodid (IntelliJ IDEA Community) 23

3.4.2 Andmebaasi päringumeetodid (Visual Studio Code) 24

3.5 Project Lombok 25

3.5.2 Project Lombok (IntelliJ IDEA Community) 27

3.5.3 Project Lombok (Visual Studio Code) 27

4 Veebirakendus *iBuy* ning projekti *IntelliSense* Spring Boot ja Angular raamistike jaoks 28

4.1 Veebirakendus *iBuy* 28

4.2 Veebirakenduse *iBuy* Spring Boot raamistiku SpringApplication klass, *@SpringBootApplication* 31

4.2.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community ja lähtekoodi redaktori Visual Studio Code veebirakenduse *iBuy* Spring Boot raamistiku peaklassi *IntelliSense* 31

4.3 Veebirakenduse *iBuy* Spring Boot raamistiku olemid, *@Entity* 32

4.3.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community veebirakenduse *iBuy* Spring Boot raamistiku olemite *IntelliSense* 32

4.4 Veebirakenduse *iBuy* Spring Boot raamistiku hoidlad, *@RepositoryRestResource* 33

4.4.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community veebirakenduse *iBuy* Spring Boot raamistiku hoidlate *IntelliSense* 33

4.5 Veebirakenduse *iBuy* Spring Boot raamistiku restkontrollerid, *@RestController* 34

4.5.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community veebirakenduse *iBuy* Spring Boot raamistiku restkontrollerite *IntelliSense* 34

4.6 Veebirakenduse *iBuy* Spring Boot raamistiku teenuseklass, *@Service* 35

4.6.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community veebirakenduse *iBuy* Spring Boot raamistiku teenuseklassi *IntelliSense* 35

4.7 Veebirakenduse *iBuy* Spring Boot raamistiku turvakonfiguratsiooni klass, *@EnableWebSecurity* 36

4.7.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community veebirakenduse *iBuy* Spring Boot raamistiku turvakonfiguratsiooni klassi *IntelliSense* 36

4.8 Üldine *IntelliSence'i* võrdlus integreeritud arenduskeskkonna IntelliJ IDEA Community ja lähtekoodiredaktori Visual Studio Code Veebirakenduse *iBuy* põhjal Spring Boot raamistiku jaoks 38

4.9 Angular raamistiku TypeScript programmeerimiskeel ja selle *IntelliSense* 38

4.10 Veebirakenduse *iBuy* Angular raamistiku mudelid 39

4.11 Veebirakenduse *iBuy* Angular raamistiku teenuseklassid 39

4.11.1 Lähtekoodiredaktori Visual Studio Code veebirakenduse *iBuy* Angular raamistiku TypeScript programmeerimiskeele teenuseklasside *IntelliSense* 39

5 Integreeritud arenduskeskkonna IntelliJ IDEA Community ja lähtekoodiredaktori Visual Studio Code versioonikontrollisüsteem 40

5.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community versioonikontrollisüsteem 40

5.2 Lähtekoodiredaktori Visual Studio Code versioonikontrollisüsteem 42

6 Tulemused 43

6.1 Spring Initializr 43

6.2 Flyway andmebaaside migreerimise tööriist 43

6.3 Relatsioonandmebaasi haldussüsteemi MySQL tugi 44

6.4 Andmebaasi päringumeetodid 44

6.5 Project Lombok 45

6.6 *IntelliSense* Spring Boot ja Angular raamistike jaoks 46

6.8 Üldised tulemused 47

7 Kokkuvõte 49

Kasutatud kirjandus 50

Lisad 53

Sissejuhatus

Veebiarenduse tööriistu on täiustatud pidevalt. On loodud uusi raamistikke ja programmeerimiskeeli, mille eesmärk on muuta iga programmeerija jaoks veebiarendus kiiremaks ja mugavamaks protsessiks. Tutvudes tööpakkumistega Eesti turul kasutades kahte tööportaali, milleks on CV ja CV Keskus, on pandud tähele, et infotehnoloogia valdkonnas on Eestis nõutud arendajad, kes omavad teadmisi Java programmeerimiskeelest ja lisaks teadmisi Spring Boot raamistikust. Samuti soovitatakse Java arendajatel peale Spring Boot raamistiku tundmist omada teadmisi ka veebi tagarakenduse raamistikest (nt Angular).

Selleks, et koodi kirjutamine oleks lihtsam ning ka koodi edastamine programmeerijate vahel oleks kiirem, kasutatakse erinevaid arendusvahendeid, mis võimaldavad kiirendada koodi haldamist. Tavaliselt sellised vahendid sisaldavad endas tekstiredaktorit ja annavad võimaluse seadistada kasutatava programmeerimiskeele kompilaatorit ja interpretaatorit. Mõned arendusvahendid sisaldavad automaatse täitmise võimalust (ingl. keeles *autocomplete,* vt paragrahv *1.1.2*), mis teeb koodi kirjutamise lihtsamaks, andes vihjeid koodi "mõtete" lõpetamiseks. Kuid mõned keskkonnad sisaldavad endas lisavahendeid erinevate raamistikega töötamiseks. Näiteks on Spring raamistiku initsialiseerimise tugi Spring Initializr (vt paragrahv *3.1*), mis aitab rakenduse projekti struktuuri üles ehitada. Arenduskeskkondi on mitmeid ning tuginedes tootesoovituste kogukonna Slant pakkumistele arenduskeskkonna valikul Java programmeerimiskeelega töötamiseks, hakatakse selle bakalaureusetöö raames vaatlema kahte tuntud arendusvahendit, milleks on integreeritud arenduskeskkond IntelliJ IDEA Community (kuna just Community version on tasuta kasutamiseks) ja lähtekoodiredaktor Visual Studio Code.

Töö eesmärk on luua veebirakendus Spring Boot ja Angular raamistikke kasutades ning veebirakenduse arendamise käigus võrrelda vahendite haldamisvõimalusi mõlema raamistiku jaoks. Vastavalt võrdlemisele otsustada, milline vahend sobib paremini veebi arendamiseks nende raamistike kasutamisel. Haldamisvõimalusi raamistikega töötamisel on mitmeid ning võrdlemise kriteeriumite valikul on arvestusse võetud märkused veebilehelt *Programming Notes*[1] ja autori enda kogemuse põhjal mõned tööriistad. Tulemusena keskkondade haldamisvõimalusi võrreldakse järgmiste kriteeriumite alusel: Spring Initializr (vt paragrahv *3.1*), Flyway andmebaaside migreerimise tööriist (vt paragrahv *3.2*), MySQL relatsioonandmebaasi haldussüsteemi tugi (vt paragrahv *3.3*), andmebaasi päringumeetodid (vt

[1] https://www3.ntu.edu.sg/home/ehchua/programming/howto/EditorIDE.html

paragrahv *3.4*), Project Lombok annotatsioonid (vt paragrahv *3.5*), *IntelliSense* (vt paragrahv *1.1.2* ja ptk *4*) ja versioonikontrollisüsteem (VKS, vt ptk *5*).

Antud töö koosneb seitsmest põhiosast. Töö esimeses peatükis kirjeldatakse, milleks kasutatakse integreeritud arenduskeskkondi ja lähtekoodiredaktoreid ning mille poolest integreeritud arenduskeskkond erineb lähtekoodiredaktorist. Lisaks kirjeldatakse lähtekoodi redigeerimise aspekte ja raamistike haldamisvõimalusi. Järgmises peatükis kirjeldatakse lühidalt Spring ja Spring Boot raamistike erinevusi. Samuti kirjeldatakse projekti struktuuri Spring Boot ja Angular raamistikke kasutades. Töö kolmandas peatükis kirjeldatakse integreeritud arenduskeskkonna IntelliJ IDEA Community ja lähtekoodiredaktori Visual Studio Code haldamisvõimalusi Spring Boot raamistiku jaoks. Järgmises peatükis kirjeldatakse loodud veebirakendust *iBuy* ja selle tagarakenduse süsteemi põhikomponente ning paralleelselt analüüsitakse integreeritud arenduskeskkonna IntelliJ IDEA Community ja lähtekoodiredaktori Visual Studio Code *IntelliSense'i* võimalusi Spring Boot ja Angular raamistike jaoks. Töö viiendas peatükis kirjeldatakse versioonkontrollisüsteeme mõlema arendusvahendi puhul. Töö lõpus analüüsitakse saadud tulemusi.

1 Integreeritud arenduskeskkond ja lähtekoodiredaktor

Integreeritud arenduskeskkond ja lähtekoodiredaktor on süsteemid, mida programmeerijad kasutavad tarkvara arendamiseks. Sellised arendusvahendid võimaldavad programmeerijatel konsolideerida tarkvara kirjutamise vajalikke aspekte, mis kiirendavad nende tootlikkust ning pakuvad kasutajaliidese abi lähtekoodi redigeerimiseks, koodi süntaksvigade tuvastamiseks, käivitatavate failide loomiseks ning koodi silumiseks [1]. Tavaliselt integreeritud arenduskeskkondi eristatakse lähtekoodiredaktoritest järgmiste aspektide poolest:

- lähtekoodiredaktorid ei võimalda kompileerida ja siluda lähtekoodi, samas integreeritud arenduskeskkondades on see võimalik
- lähtekoodiredaktorid vajavad lisaks pistikprogrammide installimist osade programmeerimiskeelte või raamistikega töötamisel
- integreeritud arenduskeskkonnad on mõeldud kitsama hulga programmeerimiskeelte või raamistike jaoks kui lähtekoodiredaktorid

Tänapäeval on aga raske kindlaks teha, mille poolest integreeritud arenduskeskkond erineb lähtekoodiredaktorist, sest paljud lähtekoodiredaktorid sisaldavad endas integreeritud arenduskeskkondade jaoks mõeldud funktsionaalsusi. Näiteks lähtekoodiredaktor Visual Studio Code (lühend VSCode) võimaldab kompileerida ja siluda lähtekoodi. VSCode peetakse lähtekoodiredaktoriks, sest seda kasutades saab koodi redigeerida paljude programmeerimiskeelte korral. Arendusvahend VSCode nõuab pistikprogrammide installimist programmeerimiskeeltega töötamisel ning on väiksema mälumahuga (umbes 100 MB installimisel).

1.1 Lähtekoodi redigeerimine

Programmeerimine on mõeldud tarkvaraarendamiseks konkreetsete ülesannete lahendamiseks ning lähtekoodi kirjutamine on väga tähtsal kohal programmeerimisel. Paljud arendusvahendid võimaldavad koodi kirjutamisel tuvastada süntaksvigu (ingl. keeles *syntax highlighting*), näidata koodirea lõpetamiseks loetelu (automaatne täitmine) ning pakuvad ka silumise võimalusi.

1.1.1 Süntaksi esiletõstmine

Kaasaegsed arendusvahendid annavad võimaluse kasutada süntaksi esiletõstmist visuaalsete näpunäidetega. Sellised vahendid aitavad tuvastada, millisel koodireal on viga tekkinud ning milline oleks võimalik lahendus selle vea eemaldamiseks (kui lahendus eksisteerib). Vigade esiletõstmine sõltub interpretaatorist, mis vastutab lähtekoodi transleerimise eest masinkoodiks. Samuti aitavad arenduskeskkonnad eristada programmeerimiskeele komponente ehk värviliselt eristada programmeerimiskeele klassid, väljad, meetodid jms. Süntaksi esiletõstmine sõltub kasutatavast programmeerimiskeelest [2].

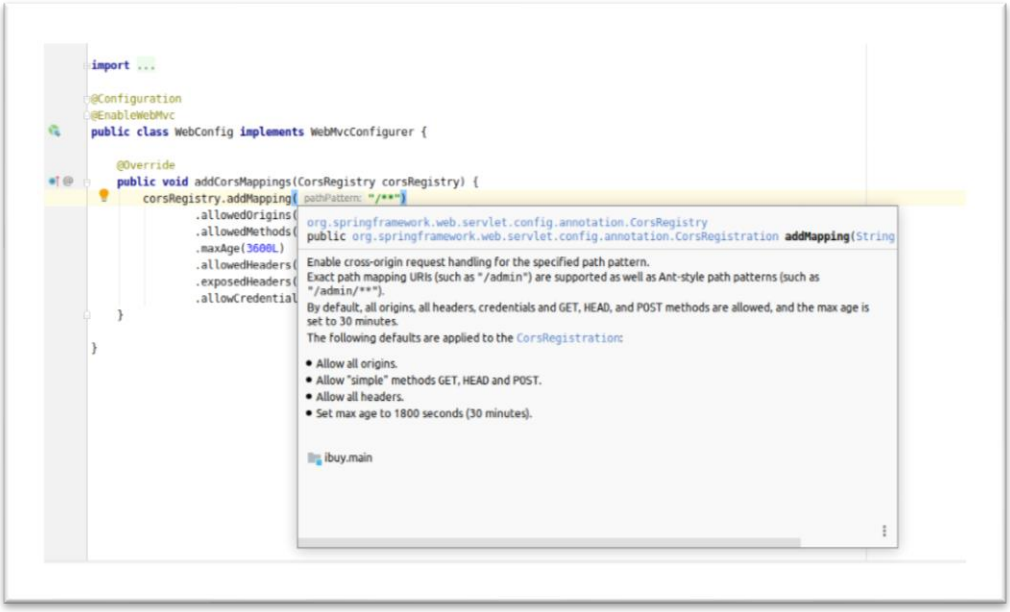
1.1.2 *IntelliSense*

IntelliSense on koodi lõpuleviimise abivahend, millel on kaks peamist funktsiooni: liikmete loend ja kiirteave. Need funktsioonid aitavad jälgida sisestatavaid parameetreid ja väärtustada atribuute ning meetodeid vaid mõne klahvivajutusega. Paljud *IntelliSense* funktsioonid on programmeerimiskeele spetsiifilised.

Tüübi (või nimeruumi) kehtivate liikmete loend ilmub näiteks pärast klassinime või selle muutuja algusmärkide sisestamist (vt *Joonis 1.1*). Kui jätkata trükimärkide tippimisega, filtreeritakse loendisse ainult liikmed, mis algavad nende märkidega. Pärast loendi liikme valimist saab selle koodi sisestada, vajutades *Tab* klahvi, sisestades tühiku või vajutades *Enter* klahvi (sõltub kasutatavast arenduskeskkonnast ja selle *IntelliSense-ist*). Loend võib olla üsna pikk, nii et loendis üles või alla liikumiseks tuleb kasutada *PgUp* ja *PgDn* klahve. *IntelliSense'i* liikmete loend on samas ka programmeerimise automaatse täitmise võte.

Joonis 1.1. Automaatne täitmine arendusvahendis IntelliJ IDEA Community.

Parameetriteave annab informatsiooni meetodi/funktsiooni parameetrite arvu, nimede ja tüüpide kohta. Paksus kirjas olev parameeter tähistab järgmist parameetrit, mida funktsiooni sisestamisel nõutakse. Ülekoormatud ehk pika parameetrite nimekirjaga meetodite/funktsioonide puhul saab kasutada *PgUp* ja *PgDn* klahve, et vaadata meetodi/funktsiooni alternatiivset teavet. Lisaks näiteks *Ctrl + q* klahvide kombinatsiooniga (sõltuvalt arenduskeskkonnast, klahvide kombinatsioon võib ka teine olla) saab informatsiooni klassi, liidese või meetodi/funktsiooni vastava komponendi kasutamise kohast (vt *Joonis 1.2).*



Joonis 1.2. IntelliJ IDEA Community arendusvahendi kiirteave.

1.1.4 Koodi silumine

Tarkvara arendajad võivad koodi kirjutamisel teha vigu. Selleks, et vead kiiremini üles leida, võimaldavad kaasaegsed arendusvahendid koodi silumist. Koodi silumise abil saab uurida, millisel real viga tekib ning millised väärtused on muutujatel sellel real. Antud protsess kiirendab vigade ülesleidmist ning aitab korrektset koodi kirjutada, et arendatav tarkvara vastaks püstitatud nõuetele.

1.2 Raamistike haldamisvõimalused

Raamistik on tarkvararakenduste arendamise platvorm. See loob aluse, mida kasutades tarkvaraarendajad saavad konkreetse platvormi jaoks programme luua. Näiteks raamistik võib

sisaldada eelnevalt määratletud klasse ja funktsioone, mida saab kasutada sisendi töötlemiseks, riistvaraseadmete haldamiseks ja süsteemitarkvaraga "suhtlemiseks". See lihtsustab arendusprotsessi, kuna programmeerijad ei pea uue rakenduse väljatöötamisel baasfunktsionaalsust leiutama. Raamistikke on mitmeid ja nende funktsionaalsed võimalused sõltuvad sihtarenduse grupist. Tavaliselt raamistikke jaotakse kolme rühma:

- andmeteaduse raamistikud, mis keskenduvad andmete analüüsimisele
- mobiilsed arendusraamistikud, mis keskenduvad mobiilsete rakenduste tootmisele
- veebiraamistikud, mis keskenduvad veebirakenduste tootmisel

Antud töö on seotud veebiarendusega ja selles töös vaadeldakse Spring Boot ja Angular veebiraamistikke. Spring Boot raamistik vastutab tagarakenduse süsteemi poolt (vt paragrahv *2.1*) ja Angular raamistik vastutab eesrakenduse süsteemi poolt (vt paragrahv *2.2*). Veebiraamistikud vastutavad veebiteenuste, veebiressursside ja veebi programmeerimisliideste arendamise toetamise eest. Haldamisvõimaluste võrdlemise kriteeriumite valikul on arvestusse võetud märkused veebilehelt *Programming Notes*[2] ja autori enda kogemuse põhjal mõned tööriistad. Selle tulemusena võrreldakse keskkondi järgmiste kriteeriumite suhtes: Spring Initializr (vt paragrahv *3.1*), Flyway andmebaaside migreerimise tööriist (vt paragrahv *3.2*), MySQL relatsioonandmebaasi haldussüsteemi tugi (vt paragrahv *3.3*), andmebaasi päringumeetodid (vt paragrahv *3.4*), Project Lombok annotatsioonid (vt paragrahv *3.5*), *IntelliSense* (vt paragrahv *1.1.2* ja ptk *4*) ja versioonikontrollisüsteem (VKS, vt ptk *5*).

[2] https://www3.ntu.edu.sg/home/ehchua/programming/howto/EditorIDE.html

2 Spring Boot ja Angular raamistike struktuur ja koostoimetamine veebiarenduses

2.1 Spring ja Spring Boot

Spring on Java põhine raamistik rakenduste loomiseks. Selle üks parimaid omadusi on sõltuvuse süstimine (ingl. keeles DI, Dependency Injection) või samuti kontrolli ümberpööramine (ingl. keeles IOC, Inversion Of Control), mis võimaldab luua ube (ingl. keeles *beans*, vt paragrahv *4.2*) Java klassidest ning siduda *(*ingl. keeles *autowire)* need klassid omavahel. Spring Boot raamistik on laiendus Spring raamistikule ja seda kasutatakse mikroteenuse loomiseks. Spring Boot raamistiku on välja töötanud Pivotal Team, kelle eesmärk oli luua raamistik, mis võimaldab konstrueerida eraldiseisvaid ja tootmisvalmivaid Spring rakendusi, keskendudes veebiarendusele. Spring Boot on tagarakenduse süsteemi raamistik, mille peamisteks ülesanneteks on rakenduse programmeerimisliideste kaitsmine kolmandate osapoolte rünnakute eest, kasutajatele volituste andmine, veatu suhtluse hõlbustamine andmebaasidega ja ühtsete ressursside lokaatorite marsruutimine andmebaasidesse, et saada klientidelt küsitavat teavet jpm. Selle üks peamisi eripärasid on AutoConfiguration tunnuse olemasolu, mis arendaja eest automaatselt konfigureerib kõik kasutusele võetud sisemised funktsionaalsed sõltuvused (ingl. keeles *dependencies*) [3]. Spring Boot raamistik sisaldab nelja peamist ehituskomponenti:

- Spring Boot starterid
- Spring Boot AutoConfigurator
- Spring Boot CLI
- Spring Boot Actuator

Spring Boot starterid on Spring Boot raamistiku üks peamisi komponente. Spring Boot starteri peamine vastutus on kombineerida eraldiseisvate sõltuvuste või seotud sõltuvuste grupp eraldi sõltuvusteks. Selleks, et kasutada starterite välispakettide funktsionaalsust, tuleb käsitsi lisada projekti ehitusskripti (pom.xml või build.gradle) .jar faile, mis on aeganõutav ning suurendab ka ehitusskripti mälu kasutust. Spring Boot raamistiku starteri komponent ühendab kõik vajalikud .jar failid ühte .jar faili, nii et kasutajal tuleb lisada ehitusskripti ainult funktsionaalseid sõltuvusi. Näiteks, kui lisada *spring-boot-starter-web* .jar faili sõltuvust

ehitusskripti, siis Spring Boot raamistik laeb alla kõik vajalikud .jar failid välispaketist ning lisab vastavalt klassiteele projekti.

Teine oluline Spring Boot raamistiku komponent on Spring Boot AutoConfigurator. Spring Boot raamistiku automaatse konfigureerimise tööriista peamine vastutus on Spring raamistiku tööks vajaliku konfiguratsiooni vähendamine. Töötades Spring Boot raamistikuga ei pea määratlema XML konfiguratsiooni ja annoteerimiskonfiguratsiooni. Selle teabe edastamise eest hoolitseb Spring Boot raamistiku komponent AutoConfigurator. Näiteks, kui üles ehitada Spring IO platvormi abil Spring MVC rakendust, tuleb määratleda XML konfiguratsioone, näiteks vaateid, vaadete lahendajaid jne. Spring MVC (Model-View-Controller) on Java raamistik, mida kasutatakse veebirakenduste loomiseks. See järgib *mudeli-vaate-kontrolleri* kujundusmustrit. Antud raamistik rakendab kõiki Spring põhifunktsioone, nagu juhtimise ümberpööramine, funktsionaalse sõltuvuse süstimine (ingl. keeles *dependency injection*). Spring MVC pakub lahendust *mudeli-vaate-kontrolleri* kasutamiseks DispatcherServlet abiga. DispatcherServlet on klass, mis võtab vastu sissetuleva päringu ja ühendab selle õige ressursiga, milleks on need samad kontrollerid, mudelid ja vaated. Kui kasutada projekti ülesehituse failis .jar faili *spring-boot-starter-web*, siis Spring Boot raamistiku komponent AutoConfigurator haldab vaateid automaatselt. Lisaks vähendab Spring Boot raamistik märkuste konfiguratsiooni määratlemist. Kui kasutada klassi tasemel märkust *@SpringBootApplication*, siis Spring Boot raamistiku komponent AutoConfigurator lisab Java baitkoodi automaatselt kõik nõutavad märkused (vt paragrahv *4.2*).

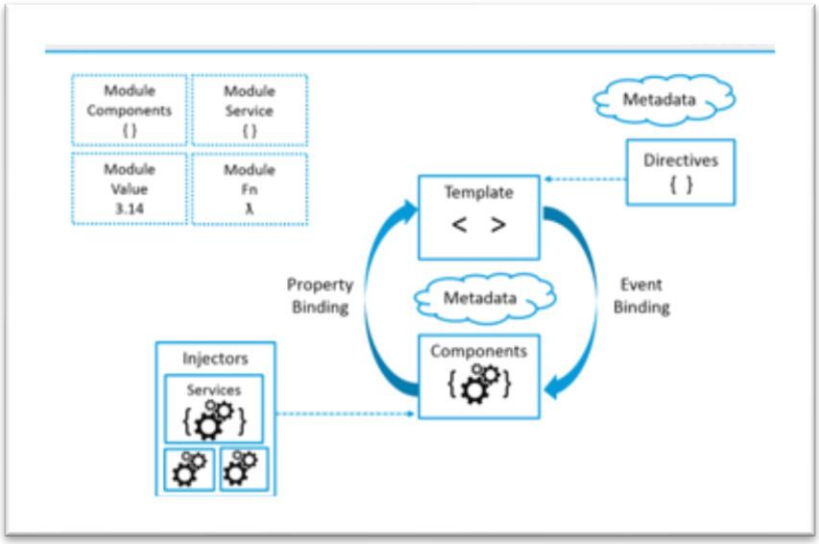
Spring Boot käsurea liides (ingl. keeles CLI, Command-line interface) on Spring Boot tarkvara rakenduste käivitamiseks ja testimiseks käsurealt. Kui käivitada rakendus Spring Boot CLI liidese abil, kasutab see sisemiselt Spring Boot starterite ja Spring Boot AutoConfigurate komponente, et hallata kõiki funktsionaalseid sõltuvusi.

Spring Boot Actuator on Spring Boot raamistiku alamprojekt. See sisaldab lisafunktsioone, mis aitavad Spring Boot rakenduse protsesse hallata. Spring Boot Actuator-i peamisteks komponentideks on täiturmehhanismi lõpp-punktid (keskkonnad, kus paiknevad ressursid). Spring Boot rakenduse haldamiseks ja jälgimiseks saab kasutada HTTP ja JMX lõpp-punkte vaadete renderdamisel. Projekti esitlusomaduste kasutamiseks võetakse kasutusele Spring Boot täiturmehhanism. Täiturmehhanismi lõpp-punktid võimaldavad rakendusega koos toimetada. Spring Boot pakub mitmeid sisseehitatud lõpp-punkte ja saab luua ka enda lõpp-

punkti. Enamikes rakendustes kasutatakse HTTP protokolli, kus lõpp-punkti ID koos täituri prefiksiga kaardistatakse URL-iks [4].

2.2 Angular

Angular on kliendirakenduste raamistik, mis on loodud ettevõte Google poolt. Selle eesmärk seisneb SPA-lahenduste välja töötamises (ingl. keeles SPA, Single Page Application) ehk üheleheliste rakenduste arenduses. Angular on arendatud Angular JS raamistikust ning on ülesehitatud TypeScript programmeerimiskeeles.



Joonis 2.1. Angular raamistiku eesrakenduse süsteemi struktuur [5].

Angular raamistiku struktuur põhineb teatavatel põhimõistetel ning selle ehitusplokide koostoimetamine on illustreeritud joonisel *2.1*. Peamised ehitusplokid on moodulid nimega NgModules, mis võimaldavad kompileerida rakenduse komponente. Iga selline moodul kogub vastava koodi funktsionaalsetesse komplektidesse. Rakendustel, mis on kirjutatud Angular raamistikku kasutades, on alati vähemalt juurmoodul, mis vastutab rakenduse initsialiseerimise eest ja tavaliselt on sellel palju rohkem funktsioonimooduleid. Komponendid määratlevad vaateid, mis on ekraanielementide komplektid, mida saab vastavalt rakenduse loogikale ja andmetele muuta. Komponendid kasutavad teenuseid, mis sõltuvad spetsiifilistest funktsioonidest ja mis pole vaadetega otseselt seotud, vaid vastutavad just funktsionaalsuse eest. Teenusepakkujaid saab kokku siduda komponentidega sõltuvustena, muutes rakenduse koodi modulaarseks, korduvkasutatavaks ja tõhusaks. Nii komponendid kui ka teenused on

lihtsalt klassid dekoraatoritega, mis tähistavad vastava klassi tüüpi ja mis pakuvad metaandmeid Angular raamistiku käitumiseks. On olemas neli peamist dekoraatorite tüüpi [6]:

- Klasside dekoraatorid, nt. *@Component* ja *@NgModule*. Klasside dekoraatorid on tipptasemel sisekujundajad, mida kasutatakse klassi kasutamise kavatsuste väljendamiseks. Need dekoraatorid võimaldavad määrata klassi moodulina või komponendina ja koodi klassi määramiseks pole vaja kirjutada, vaid piisab vastavast dekoraatori määramisest.
- Omaduste dekoraatorid, nt. *@Input* ja *@Output*. Omaduste dekoraatorid võimaldavad klassides konkreetseid välju kujundada. Näiteks välja sidumine *@Input* dekoraatoriga kutsub välja Angular raamistiku kompilaatori, mis automaatselt loob atribuudi nimest sisestussideme ja seob klassi välja koos dekoraatoriga, mida pärast saab klassi vaates atribuudi nime järgi kasutada.
- Meetodite dekoraatorid, nt. *@HostListener*. Meetodite dekoraatorid on väga sarnased omaduste dekoraatoritega, kuid väljade asemel kasutatakse meetodeid. Meetodite dekoraatoritega kujundatakse vastavas klassis konkreetsed meetodid funktsionaalsusega. Hea näide sellest on *@HostListener* dekoraator. See võimaldab Angular raamistiku kompilaatoril välja kutsuda kliendi poolt seotud *@HostListener* dekoraatoriga meetodeid sündmuste püüdmiseks.
- Klassi konstruktorite parameetrite dekoraatorid, nt. *@Inject*. Parameetrite dekoraatorid võimaldavad kujundada parameetreid klassi konstruktorites. Näiteks on *@Inject* dekoraator, mis laseb Angular raamistiku kompilaatoril siduda parameetri vastava klassiga. Metaandmete tõttu, mida TypeScript avaldab, ei pea *@Inject* dekoraatorit kasutama vaid piisab parameetri tüübi määramisest.

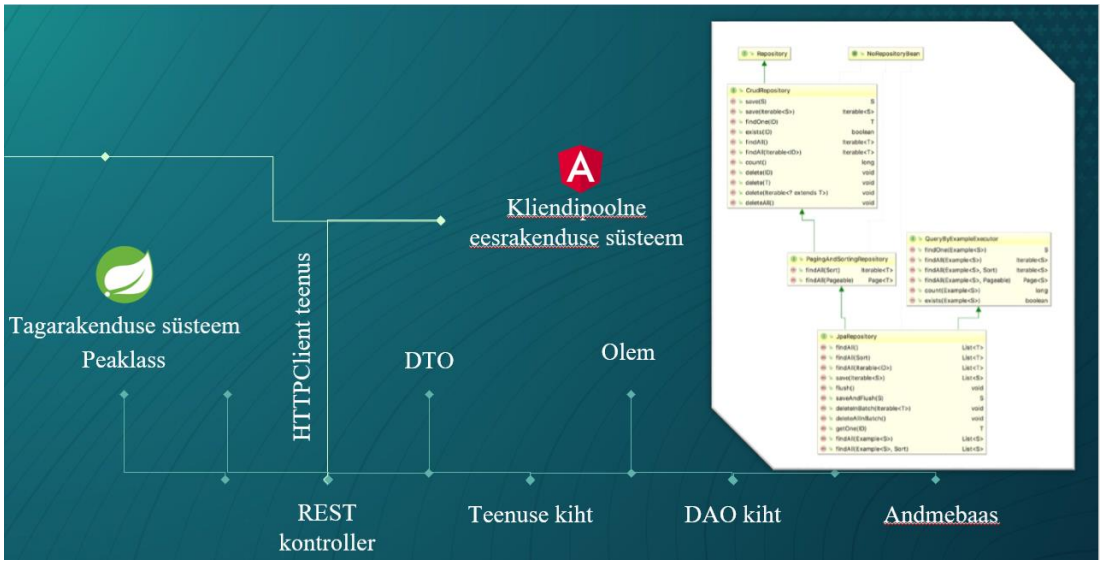
Komponentklassi metaandmed seostavad klassi malliga, mis määrab vaatet. Mall ühendab tavalise HTML-i Angular raamistiku direktiividega ja sidumismärgistustega, mis võimaldavad Angular raamistikul HTML renderdamist muuta enne selle kuvamist. Rakenduse komponendid määratlevad tavaliselt palju hierarhiliselt paigutatud vaateid. Angular raamistik pakub ruutimisteenust, mis aitab vaadete vahel navigeerimise teid määratleda [7].

2.3 Spring Boot ja Angular raamistike koostoimetamise lühikirjeldus

Spring Boot raamistiku andmetele juurdepäsu kiht alustab oma tööd SpringApplication klassi tasemel märgendatud *@SpringBootApplication* annotatsiooniga Java peameetodi

väljakutsumisel. Meetodi väljakutsumisel skaneeritakse kõik kasutusele võetud Spring raamistiku välispaketid ning registreeritakse bean-objektid (vt paragrahv *4.2*). Kui kõik bean-objektid on initsialiseeritud, siis projektis implementeeritud REST-kontrollerid väljastavad vastavalt HTTP protokollile andmebaasis olevad andmed vaikimisi JSON formaadis ja edastavad need andmed läbi HTTP-kliendi teenuse eesrakenduse süsteemile. REST (Representational State Transfer) on olekuülekanne, mis määratleb veebiteenuste loomisel kasutatavate piirangute komplekti DTO objektidena. DTO (ingl. keeles Data Transfer Object) on andmeedastusobjekt, mis kirjeldab ülekandel olemi struktuuri (klassi väljad, väljade tüübid) klassina. Neid andmeedastusobjekte võtab kasutusele rakenduse teenusekiht, milles toimub rakenduse äriloogika (nt kasutaja registreerimine, kasutaja sisselogimine jm). Teenusekihis realiseeritakse olemid vastavalt andmeedastusobjekti abstraktsele kirjeldusele. JPA (Java Persistence API) olemid pole muud kui andmete POJO-d (vt paragrahv *4.3*). Iga Spring boot raamistiku olemit hallatakse andmebaasis vastavalt hoidla kapseldamise mehhanismile, mis jäljendab olemite kogumit. Kogum toimib peamiselt markerliidesena, et jäädvustada töötavaid objektide tüüpe ja nende manipuleerimist. Objekti manipuleerimine sõltub laiendavast liidesest (vt paragrahv *3.4*).

Läbi HTTP-kliendi-teenuse saadud andmed (vaikimisi JSON formaadis) edastab Angular raamistiku teenusekihile, kus toimub äriloogika. Andmete struktuuri piiratakse DTO andmeedastusobjektidega. Vastavalt DTO struktuurile realiseeritakse olemid ja nende metaandmed antakse üle mallile ja kuvatakse kasutades Angular raamistiku direktiive. Spring Boot ja Angular raamistike koostoimetamine on illustreeritud joonisel *2.2*.



Joonis 2.2. Spring Boot ja Angular raamistike koostoimetamine.

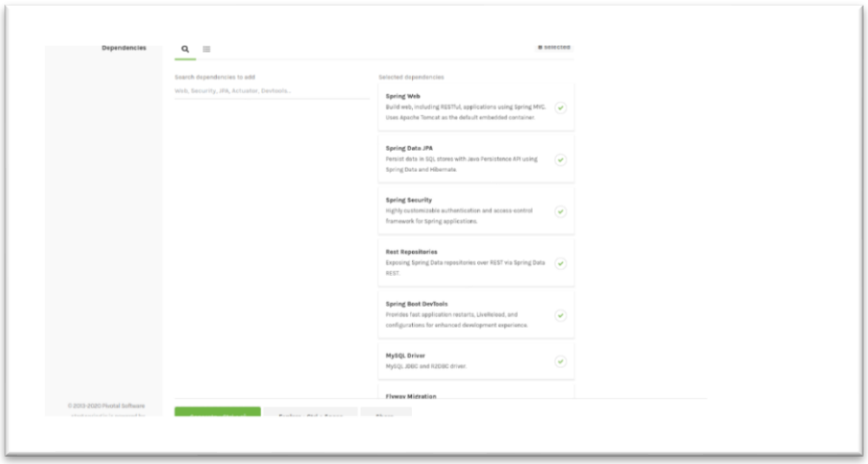
3 Integreeritud arenduskeskkonna IntelliJ IDEA Community ja lähtekoodiredaktori Visual Studio Code haldamisvõimalused Spring Boot raamistiku jaoks

3.1 Spring Initializr

Spring Initializr on tööriist, mis võimaldab luua projekti struktuuri Gradle või Maven automatiseerimissüsteemi abil Spring Boot raamistikuga töötamiseks. Spring Initializr metaandmemudel annab võimaluse seadistada projekti startereid (sisemisi sõltuvusi), nt *spring-boot-starter-thymeleaf*, *spring-boot-starter-web* jt. Iga sõltuvuskirjeldus vastutab veebiarenduse konkreetse funktsionaalsuse eest. Näiteks *spring-boot-starter-thymeleaf* sõltuvust kasutatakse MVC veebirakenduste ehitamiseks, kasutades Thymeleaf malli mootori vaateid. Lisaks annab antud tööriist võimaluse seadistada kasutatavat programmeerimiskeelt, projekti nimetust ning Spring Boot raamistiku versiooni.

3.1.1 Spring Initializr (IntelliJ IDEA Community)

Integreeritud arenduskeskkonnal IntelliJ IDEA Community (lühend IIC) puudub sisseehitatud tööriist Spring Initializr. Selleks, et lahendada probleemi seoses selle puudumisega, võetakse kasutusele veebipõhine generaator (vt *Joonis 3.1*) Spring Initializr[3], mis moodustab projekti struktuuri ühise .zip kaustana.

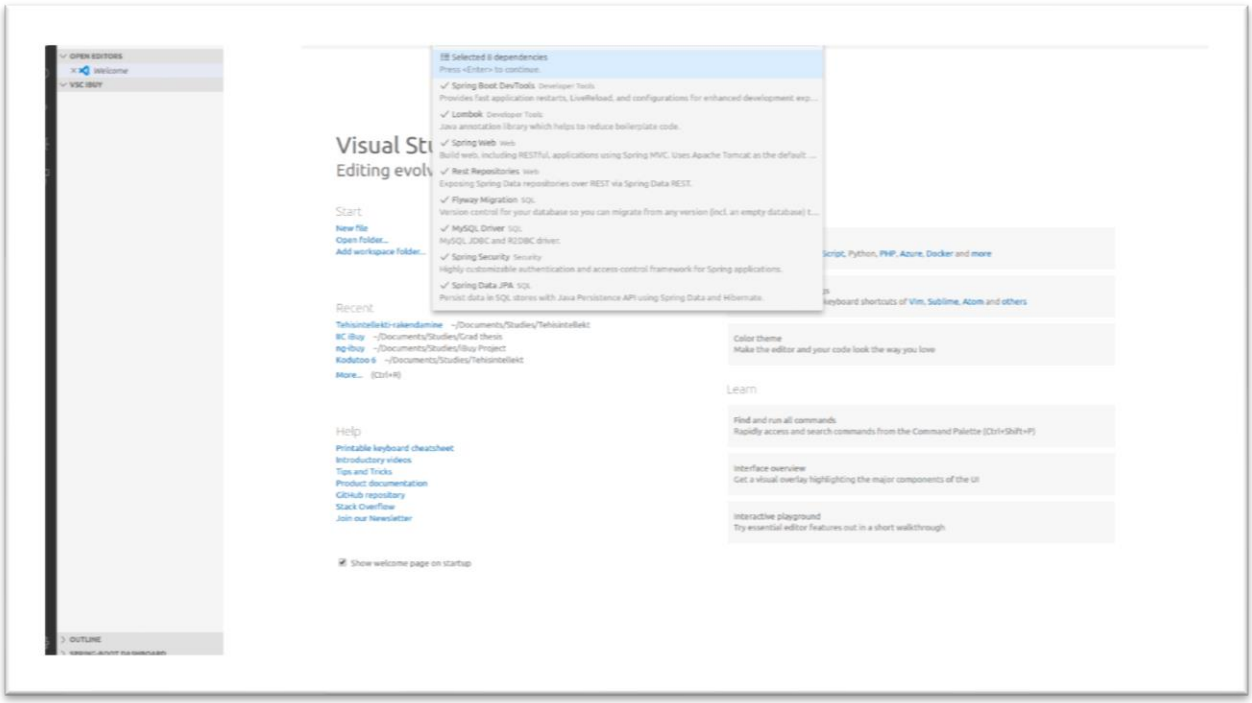


Joonis 3.1. Tööriista Spring Initializr veebipõhine kasutajaliides.

[3] https://start.spring.io

3.1.2 Spring Initializr (Visual Studio Code)

Lähtekoodiredaktoril VSCode on olemas sisseehitatud tööriist Spring Initializr. Kuna VSCode on lähtekoodiredaktor, siis selleks, et alustada Spring Boot raamistikuga arendamist, on vaja installida *Spring Boot Extension Pack* laiend. Kuna Spring Boot on Java-põhine raamistik, siis lisaks tuleb installida *Java Extension Pack* laiend. Lähtekoodiredaktori VSCode tööriist Spring Initializr annab projekti genereerimisel võimaluse määrata projektis kasutatavat programmeerimiskeelt, projekti nimetust, Spring Boot raamistiku versiooni ja funktsionaalseid sõltuvusi.



Joonis 3.2. Tööriista Spring Initializr kasutajaliides lähtekoodiredaktoril Visual Studio Code.

3.2 Andmebaaside migreerimise tööriist Flyway

Flyway on avatud lähtekoodiga andmebaaside migreerimise tööriist, mis värskendab migratsiooni abil andmebaasi struktuuri ja andmeid. Flyway tööriista kasutatakse andmebaasi migreerimiseks ühelt platvormilt teisele muutmata andmebaasi struktuuri, mis tulemusena kiirendab arendamist. Migreerimisi saab kirjutada SQL domeenispetsiifilises keeles või Java keeles [8].

3.2.1 Flyway (IntelliJ IDEA Community)

Töötades .sql failidega Intellij keskkond võimaldab *Database Navigator* ja *MaxCompute Studio* pistikprogrammide installimist. *Database navigator* on andmebaaside arendus-, skriptimis- ja navigeerimistööriist. See tööriist võimaldab koos IntelliJ keskkonnaga ühilduvat SQL ja PL/SQL keelte redigeerijat, andmebaasiühenduse haldust, skriptide tuge, andmebaasiobjektide sirvimist ja käivitatud päringute ajaloo vaatamist. [9]. *MaxCompute* Studio on IntelliJ keskkonna pistikprogramm, mis võimaldab SQL-skriptide autoriseerimist ja *MaxCompute* modelleerimisteenuse silumist, andmete sirvimist ja analüüsimist [10].

3.2.2 Flyway (Visual Studio Code)

Töötades .sql failidega pakub VSCode pistikprogrammi *SQL Server (mssql)* installimist. Pistikprogramm *mssql* on mõeldud Microsoft SQL serveri, Azure SQL andmebaasi ja SQL Data andmebaasi arendamiseks. Programm annab võimaluse kirjutada T-SQL-skripte kasutades *IntelliSense'i* [11]. Pistikprogrammi *mssql* abil on võimalik käivitada SQL-päringud, salvestada päringute tulemusi .json või .csv failivormingusse ja vaadata tulemusi redaktoris [12]. Töötamisel .sql failidega on mõistlik installida *Language PL/SQL* pistikprogrammi, mis pakub koodiviimistlust PL/SQL päringutele. Pistikprogramm sisaldab korduvkasutatavaid koodiplokke (ingl. keeles *snippets*[4]), mis on mõeldud taaskasutatava lähtekoodi genereerimiseks [13].

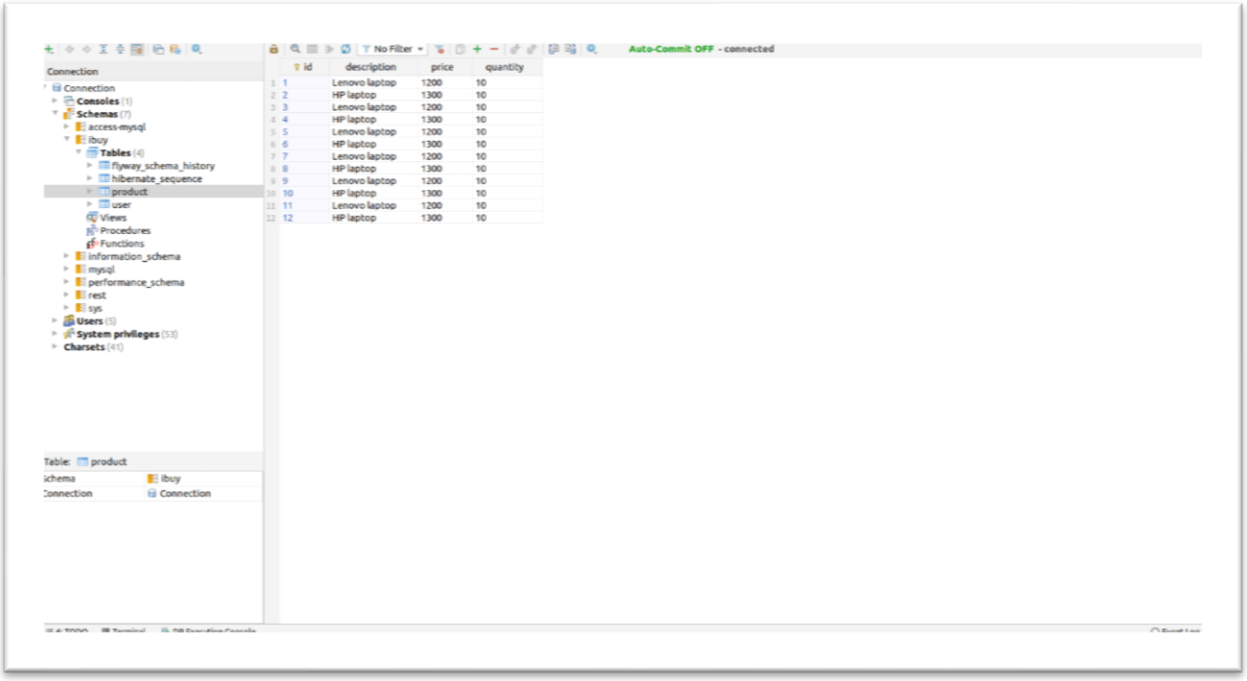
3.3 Relatsioonandmebaasi haldussüsteemi MySQL tugi

MySQL on Oracle'i toega avatud lähtekoodiga relatsioonilise andmebaasi haldussüsteem (ingl. keeles RDBMS, Relational Database Management System), mis põhineb struktureeritud päringute keelel SQL ja mis töötab mitmetel platvormidel (Microsoft Windows, Linux, macOS, Open BSD jt). MySQL põhineb klient-server mudelil. Mudeli tuumaks on MySQL server, mis haldab kõiki andmebaasi juhiseid või käske [11].

[4] https://techterms.com/definition/snippet

3.3.1 Relatsioonandmebaasi haldussüsteemi MySQL tugi (IntelliJ IDEA Community)

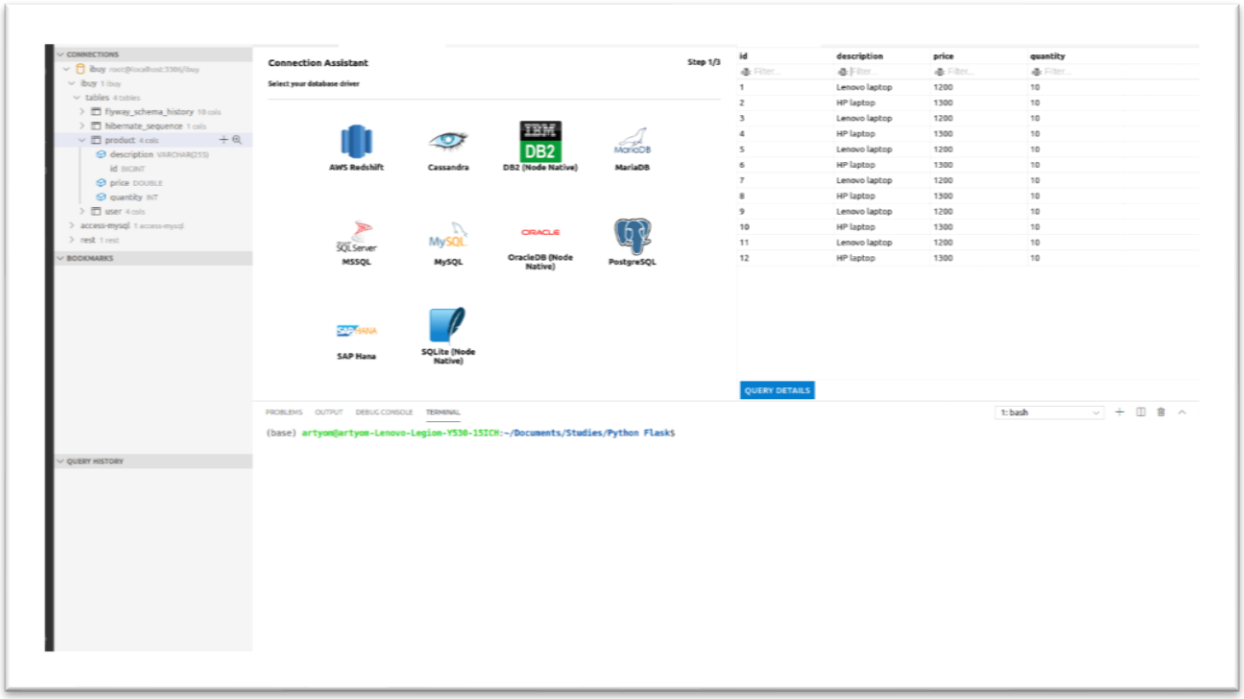
Andmebaasidega töötades arenduskeskkond IIC keskkond pakub *Database navigator* pistikprogrammi installimist (vt paragrahv *3.2.1*). Selle kasutajaliides on demonstreeritud joonisel *3.3*.



Joonis 1.3. Pistikprogrammi Database Navigator kasutajaliides arenduskeskkonnal IntelliJ IDEA Community.

3.3.2 Relatsioonandmebaasi haldussüsteemi MySQL tugi (Visual Studio Code)

Relatsioonandmebaasi haldussüsteemiga MySQL töötamisel tuleks installida *SQLTools - Database tools* pistikprogramm. Antud pistikprogramm annab võimaluse kontrollida andmebaasiga ühendamist ja päringupaneeli andmeid ning võimaldab muuta andmebaasi andmeid päringute käivitamisel. Lisaks pakub programm koodiviimistlust ning ka käivitatud päringute ajalugu [14]. Selle kasutajaliides on demonstreeritud joonisel *3.4.*



Joonis 3.4. Pistikprogrammi SQLTools kasutajaliides lähtekoodiredaktoril Visual Studio Code.

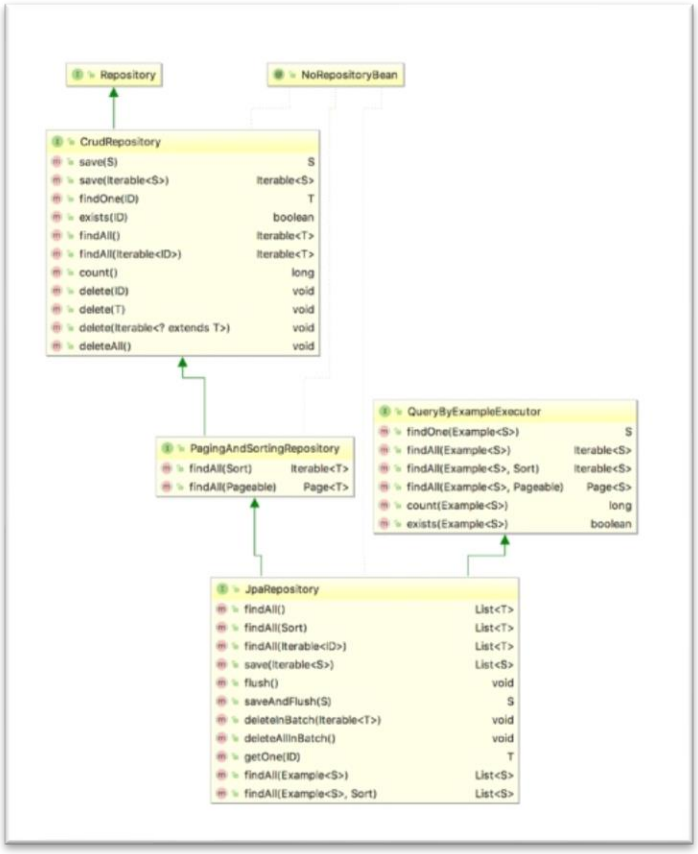
3.4 Andmebaasi päringumeetodid

Päringumeetodid on põhiaspekt, mis teeb andmebaasid võimsaks vahendiks andmete filtreerimisel ja hoidmisel. Päringumeetodeid kirjutatakse vastavalt SQL domeenispetsiifilisele keelele tuginedes programmeerimiskeele süntaksile. Tavaliselt päringumeetodeid kasutatakse programselt, kuid päringuid saab genereerida ka kasutajaliidest kasutades (nt andmebaasihaldussüsteemi Microsoft Access kasutades) [14]. Andmete manipuleerimiseks kasutatakse lehel Codecademy[5] loetletud käske.

Spring Data JPA Java Püsivuse API kirjeldab relatsioonandmete haldamist rakendustes. Programm hõlbustab selliste rakenduste väljatöötamist, mis peavad saama juurdepääsu JPA andmeallikatele. Spring Data peamine liides on hoidla. See liides toimib peamiselt markerliidesena, et säilitada töötavaid objektide tüüpe ja aidata leida seda liidest laiendavaid liideseid, et andmetega manipuleerida. Laiendatavateks liideteks on CrudRepository, PagingAndSortingRepository, QueryByExampleExecutor ja JpaRepository. Iga laiendatav liides pakub oma meetodeid andmetega töötamiseks (vt *Joonis 3.5*).

[5] https://www.codecademy.com/articles/sql-commands

SQL päringumeetodeid saab kirjutada nimega päringute (*@NamedQuery*) ja nimeta päringute (*@Query*) annotatsioonide kasutamisel. Nimega päringute kasutamine üksuste päringute deklareerimiseks on õige lähenemisviis ja sobib väikese arvu päringute korral hästi. Kuna päring on seotud Java meetodiga, mis neid täidab, siis saab päringu tegelikult siduda, kasutades Spring Data JPA *@Query* annotatsiooni, et vältida domeeniklassi külge annoteerimist. See vabastab domeeniklassi püsivalt seotud teabest ja seob päringu hoidla liidesega [16].



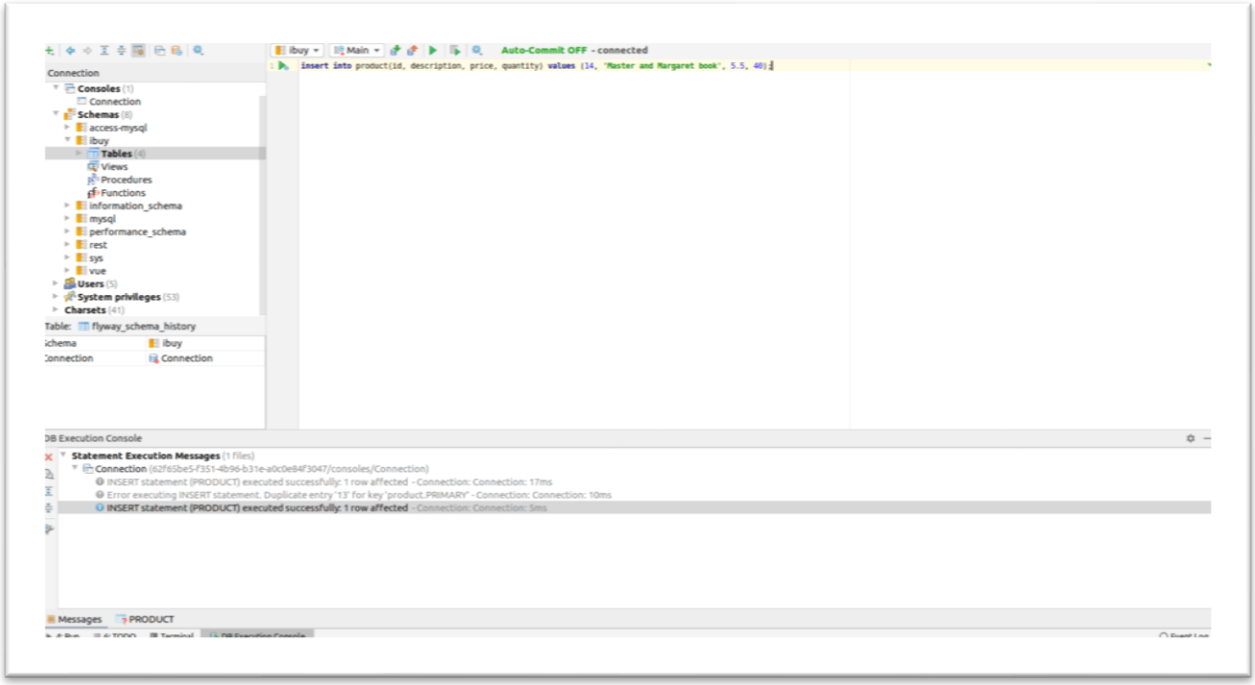
Joonis 3.5. Spring Data JPA liideste CrudRepository, PagingAndSortingRepository, QueryByExampleExecutor ja JpaRepository meetodid [17]

.

3.4.1 Andmebaasi päringumeetodid (IntelliJ IDEA Community)

Arenduskeskkond IIC ei toeta automaatset täitmist Spring Data JPA päringumeetodite annotatsioonide *@NamedQuery* ja *@Query* kasutamisel. Tööriista *DB navigator* liitumise konsooli abil saab käivitada päringumeetodeid, et filtreerida ja muuta andmebaasi andmeid.

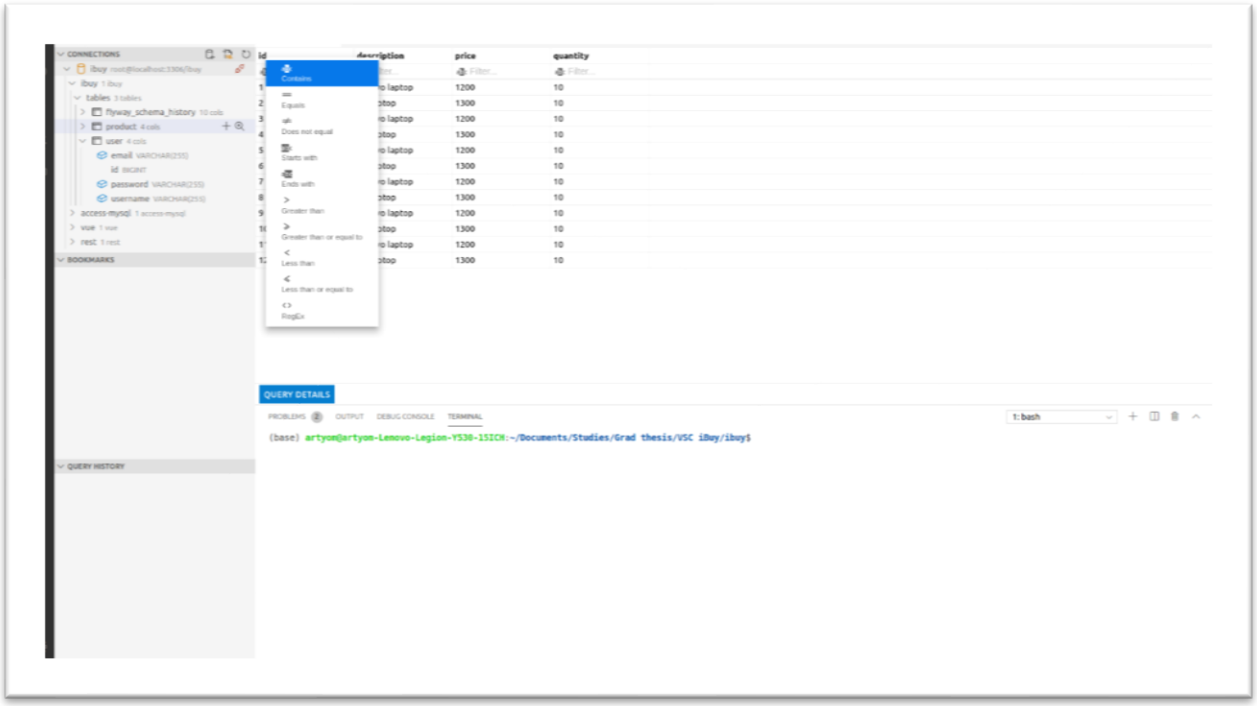
Konsool pakub automaatset täitmist päringumeetodite koodi kirjutamisel. Selle kasutajaliides on demonstreeritud joonisel *3.6*.



Joonis 3.6. Pistikprogrammi Database Navigator liitumise konsool arenduskeskkonnal IntelliJ IDEA Community.

3.4.2 Andmebaasi päringumeetodid (Visual Studio Code)

Lähtekoodiredaktor VSCode ei toeta automaatset täitmist Spring Data JPA päringumeetodite annotatsioonide *@NamedQuery* ja *@Query* kasutamisel. *SQLTools* pistikprogrammi kasutamisel on võimalik genereerida ainult *INSERT* päringumeetodeid, mis ei ole aga parametriseeritud. Selle kasutajaliides on esitatud joonisel *3.7*. Parameetritega päringute peamine eelis seisneb võimaluses sama meetodit välja kutsuda erinevate argumentidega. Lähtekoodiredaktor ei anna võimalust käivitada päringumeetodeid, on aga võimalik filtreerida andmebaasi tabelite andmeid järgmisi filtreid kasutades: *sisaldab, võrdub, ei võrdu, lõpeb sümboli(ga/tega), suurem kui, suurem või võrdne, väiksem ku*i, *väiksem või võrdne*, regulaaravaldis.



Joonis 3.7. Pistikprogrammi SQLTools kasutajaliides andmete filtreerimisel lähtekoodiredaktoril Visual Studio Code.

3.5 Project Lombok

Project Lombok on Java teek, mida integreeritakse kasutatavatesse arenduskeskkondadesse vastava pistikprogrammi installimisel. Mõned keskkonnad nagu näiteks IntelliJ IDEA (Community/Ultimate) nõuavad peale pistikprogrammi installimist annotatsioonide redigeerimise luba, et töödelda annotatsioone projekti moodulitel [18]. Teegi Project Lombok idee on asendada Vanilla Java koodiplokid Lombok annotatsioonidega, mis teeb klasside koodi kompaktsemaks ja loetavamaks.

3.5.1 Tihti kasutatavate Project Lombok annotatsioonide kirjeldused

@val annotatsiooni kasutatakse objekti tüübi määramisel. Selle kasutamisel muutuja teisendub lõplikuks, mis tähendab, et selle kasutamisel muutujat ei tohi enam väärtustada. Tavaliselt *@val* annotatsiooni kasutatakse sõnastikega töötamisel.

@var annotatsiooni kasutatakse muutuja tüübi määramisel. Selle annotatsiooni vahe võrreldes *@val* annotatsiooniga seisneb selles, et selle kasutamisel muutuja ei muutu lõplikuks, mis omakorda tähendab, et muutujat saab väärtustada.

@Getter annotatsioon on mõeldud klassi meetodi asendamiseks, mis väljastab klassi konkreetse välja väärtuse (ingl. keeles *getter*) ning *@Setter* annotatsioon on mõeldud klassi meetodi asendamiseks, mis omistab klassi konkreetsele väljale väärtuse (ingl. keeles *setter*).

Java teek Project Lombok võimaldab ka klassi konstruktorite tööd kirjeldada. Selleks on olemas annotatsioonid: @NoArgsConstructor, *@RequiredArgsConstructor*, *@AllArgsConstructor*. *@NoArgsConstructor* annotatsioon genereerib tühja konstruktorit, *@RequiredArgsConstructor* genereerib konstruktorit väljadega, mis vajavad erikäsitsemist (nt *@NonNull* annotatsioonidega väljad) ja *@AllArgsConstructor* annotatsioon genereerib konstruktorit kõikidest klassi väljadest ning iga *@NonNull* välja korral kontrollib, kas väärtus sellel väljal eksisteerib või mitte. Kui väärtus *@NonNull* väljal puudub, siis viskab erandi [19].

Iga klassimääratlusele võib lisada annotatsiooni *@EqualsAndHashCode*, et lasta Lombok teegil genereerida objektide *equals(Object other)* ja *hashCode()* meetodid. Vaikimisi kuuluvad *@EqualsAndHashCode* annotatsiooni alla kõik klassi mittestaatilised, mittesalvestatavad väljad, kuid on võimalik ka annoteerida konkreetseid välju kasutades vastavatel väljadel *EqualsAndHashCode.Include* (välja kuulumine) või *@EqualsAndHashCode.Exclude* (välja mitte kuulumine) annotatsioone. Kui on soov kasutada ülemklassi väljade *@EqualsAndHashCode* annotatsiooni, siis tuleb lisaks *@EqualsAndHashCode* annotatsiooni seada *(callSuper=true)* väärtusega. [20].

Mistahes klassi määratlusele võib lisada annotatsiooni *@ToString*, et genereerida vastava klassi *toString()* meetodi rakendamist, mis vaikimisi väljastab klassi väljade väärtusi. Pärilusel ülemväljade väljakutsumisel tuleb kasutada @ToString(callSuper = true) annotatsiooni [21].

@Data on mugav annotatsioon, mis koondab *@Getter*, *@Setter*, *@RequiredArgsConstructor*, *@EqualsAndHashCode* ja *@ToString* annotatsioonid kokku. *@Value* annotatsioon sarnaneb *@Data* annotatsiooniga. Kõik väljad seadistatakse vaikimisi privaatseteks ja lõplikeks ning settereid ei genereerita. Klass ise tehakse vaikimisi ka lõplikuks. Nagu *@Data*, genereeritakse ka kasulikud *toString()*, *equals()* ja *hashCode()* meetodid, igast väljast saadakse getter-meetod ja genereeritakse ka konstruktor, mis katab kõik argumendid (välja arvatud lõplikud väljad, mis on toodud väljadeklaratsioonis). Praktikas *@Value* annotatsiooni kuuluvad järgmised alamannotatsioonid: lõplik *@ToString*, *@EqualsAndHashCode*, *@AllArgsConstructor*, *@FieldDefaults (makeFinal = true, level = AccessLevel.PRIVATE)* ja *@Getter* [22].

3.5.2 Project Lombok (IntelliJ IDEA Community)

Project Lombok annotatsioonidega töötamisel integreeritud arenduskeskkond IIC pakub Lombok pistikprogrammi. Sellega saab kasutada järgmisi annotatsioone Vanilla Java koodi asendamiseks [23]:

@Getter, @Setter, @FieldNameConstants, @ToString, @EqualsAndHashCode, @AllArgsConstructor, @RequiredArgsConstructor, @NoArgsConstructor, @Log, @Log4j, @Log4j2, @Slf4j, @XSlf4j, @CommonsLog, @JBossLog, @Flogger, @CustomLog, @Data, @Builder, @SuperBuilder, @Singular, @Delegate, @Value, @Accessors, @Wither, @With, @SneakyThrows, @val, @var, experimental @var, @UtilityClass.

3.5.3 Project Lombok (Visual Studio Code)

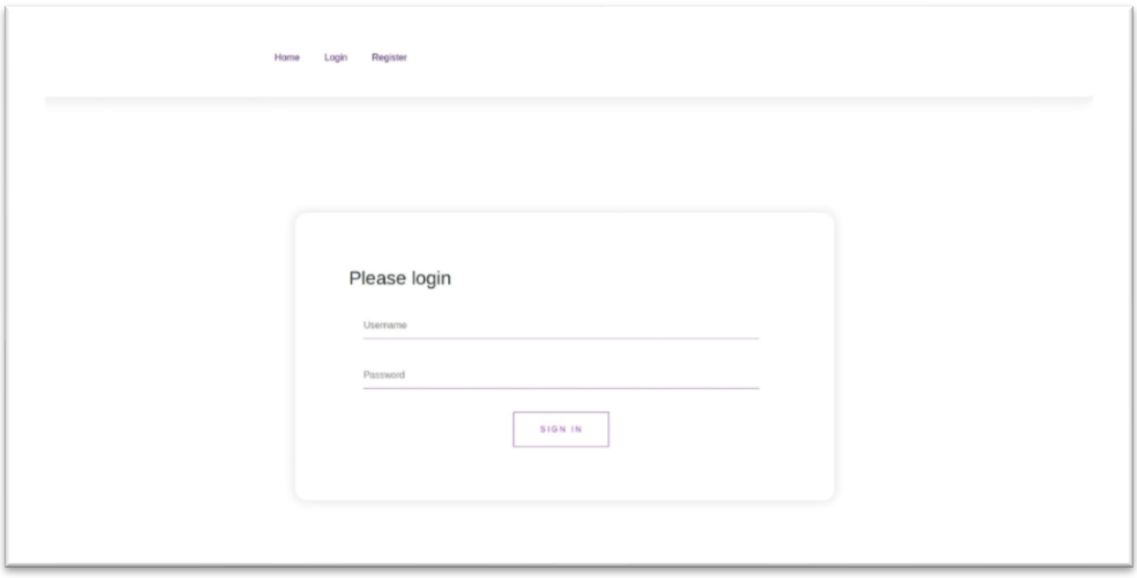
Project Lombok annotatsioonidega töötamisel lähtekoodiredaktor VSCode pakub *Lombok Annotations Support for VS Code* pistikprogrammi. Sellega saab kasutada järgmisi annotatsioone Vanilla Java koodi asendamiseks [24]:

@Getter, @Setter, @ToString, @EqualsAndHashCode, @AllArgsConstructor, @RequiredArgsConstructor, @NoArgsConstructo, @Log, @Slf4j, @Data, @Builder, @Singular, @Delegate, @Value, @Accessor, @Wither, @SneakyThrows, @val, @UtilityClass.

4 Veebirakendus *iBuy* ning projekti *IntelliSense* Spring Boot ja Angular raamistike jaoks

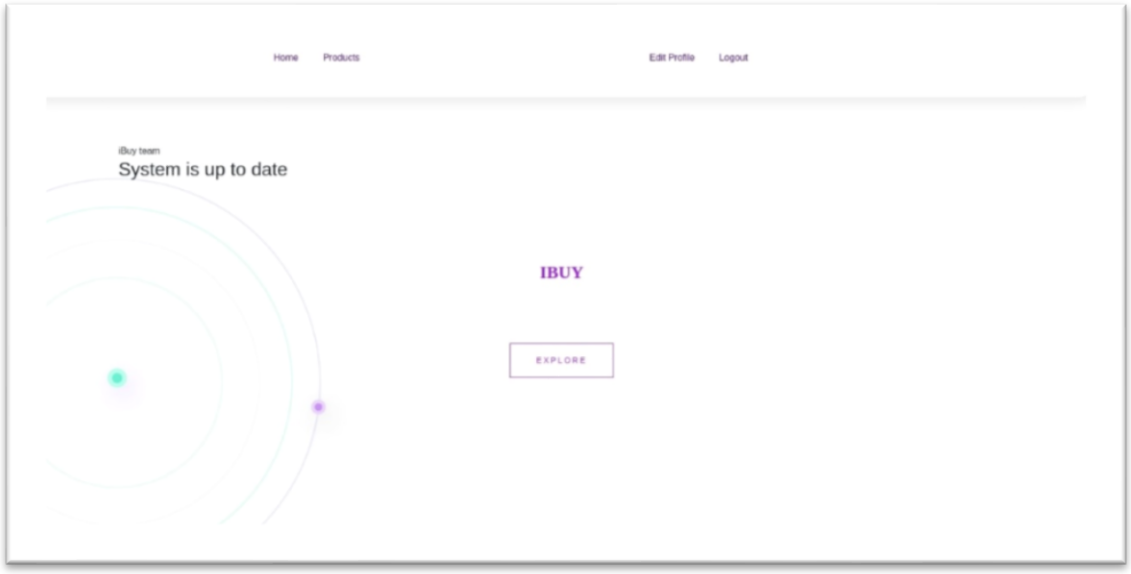
4.1 Veebirakendus *iBuy*

Integreeritud arenduskeskkonnad ja lähtekoodiredaktorid annavad võimaluse kasutada erinevaid tööriistu tarkvara mugavaks arendamiseks. Selleks, et võrrelda integreeritud arenduskeskkonna IIC ja lähtekoodiredaktori VSCode haldamisvõimalusi mõlema raamistiku jaoks, luuakse veebirakendus nimega *iBuy* mõlemat arendusvahendit kasutades. Veebirakendus *iBuy* on ehitatud Spring Boot andmetele juurdepääsu kihist (tagarakenduse süsteemi raamistik) ja Angular esitluskihist (eesrakenduse süsteemi raamistik). *iBuy* on veebirakendus, kus veebikülastaja saab registreerida ennast kasutajaks ning registreerimise õnnestumisel loodud konto abil siseneda veebipoodi nimega *iBuy*. Sisse logimise leht on illustreeritud joonisel *4.1*.



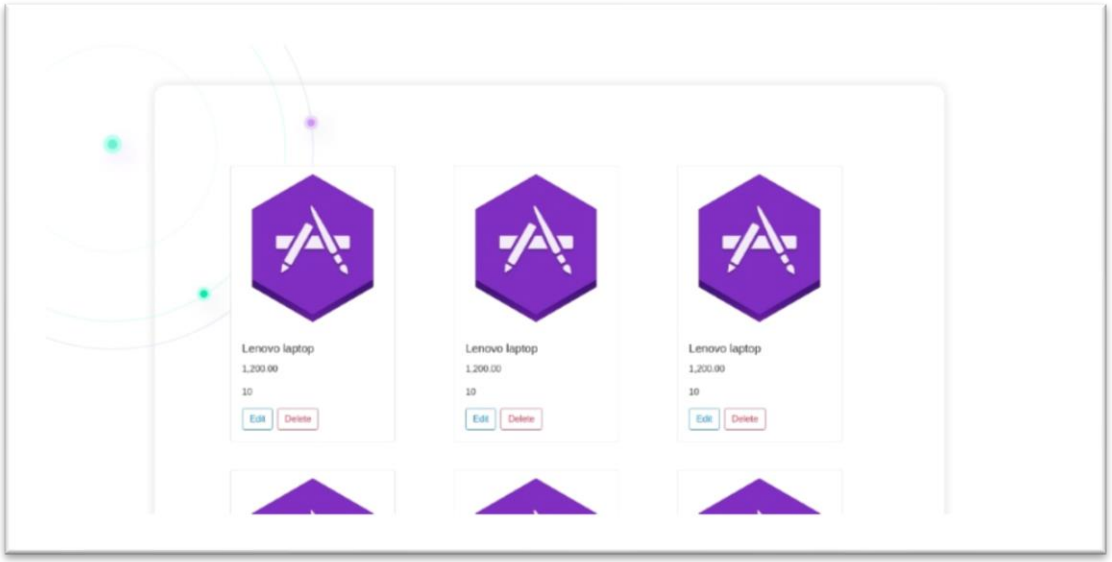
Joonis 4.1. Veebirakenduse iBuy sisse logimise leht.

Veebipoodi sisenedes ilmub veebirakenduse pealeht, kus kasutaja saab külastada veebipoe toodete lehte, lisada enda poolt tooteid andmebaasi ning muuta oma profiili andmeid. Pealeht on illustreeritud joonisel *4.2*.



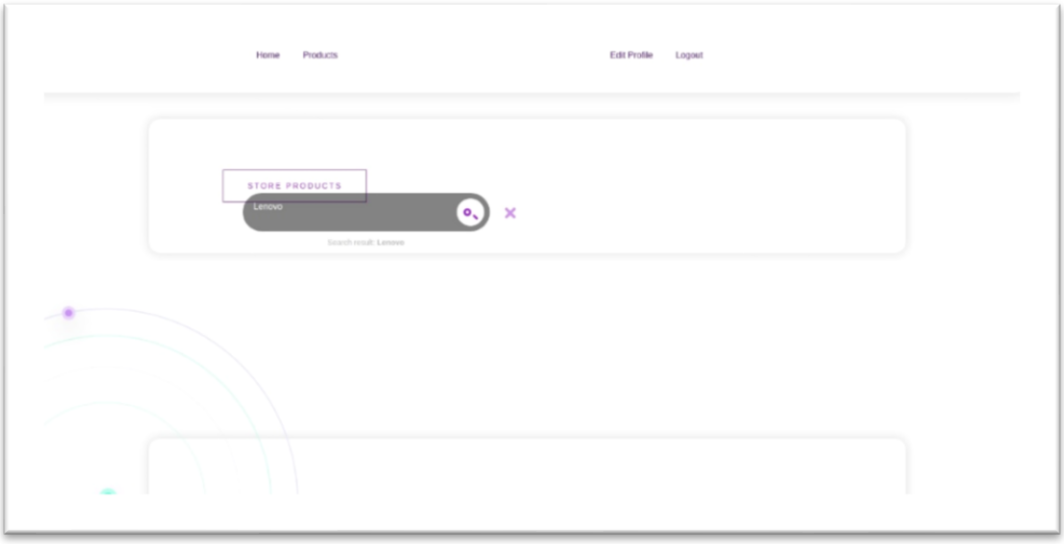
Joonis 4.2. Veebirakenduse iBuy pealeht.

Külastades veebipoe toodete lehte saab kasutaja vaadata, millised tooted on poes kättesaadavad ning mis on iga toote nimetus, kogus ja hind. Toodete lehe toodete nimekiri on illustreeritud joonisel *4.3*.



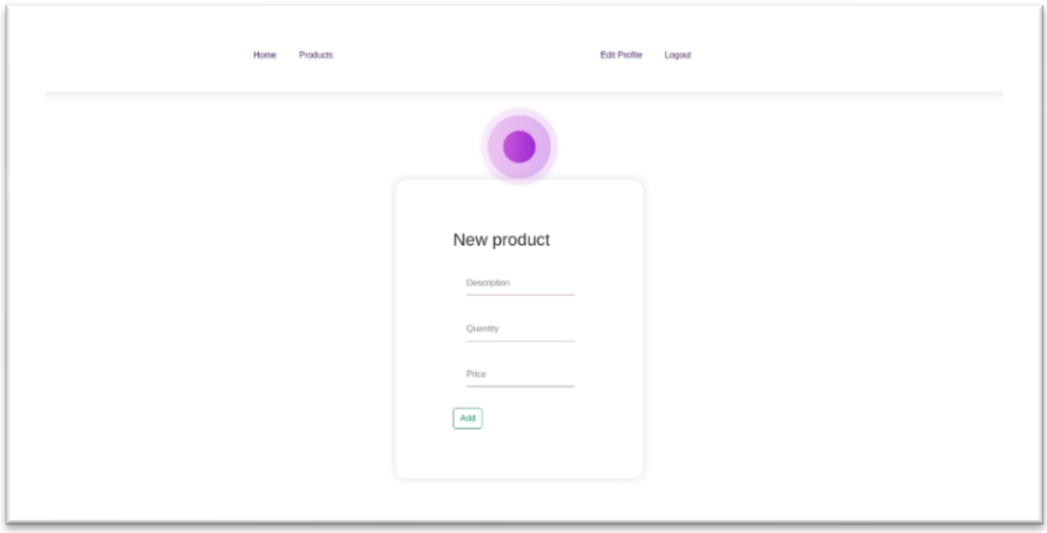
Joonis 4.3. Veebirakenduse iBuy toodete lehe toodete nimeriki.

Lisaks annab toodete leht võimaluse filtreerida poes olevaid tooteid toodete nimetuse järgi. Toodete lehe toodete filtreerimine nimetuse järgi on illustreeritud joonisel *4.4*.



Joonis 4.4. Veebirakenduse iBuy toodete lehe filtreerimine.

Veebipoodi kasutades saab lisada erinevaid tooteid andmebaasi, määrates iga toote kirjelduse, koguse ning hinna. Samuti saab lisatud toodet redigeerida ning vajadusel ka kustutada andmebaasist. Registreeritud kasutaja saab muuta oma profiili andmeid. Toote lisamise, toote muutmise ja kasutaja profiili muutmise lehed on sarnased kasutajaliidese poolest ning toote lisamise leht on illustreeritud joonisel *4.5*.



Joonis 4.5. Veebirakenduse iBuy toote lisamise leht

4.2 Veebirakenduse *iBuy* Spring Boot raamistiku SpringApplication klass, *@SpringBootApplication*

Spring Boot raamistiku SpringApplication klassi ehk peaklassi kasutatakse Java peamise peameetodi abil Spring raamistukuga loodud rakenduse alglaadimiseks ja käivitamiseks. See klass loob klassiruumist automaatselt rakenduse konteksti, skanneerib rakenduse konfiguratsiooniklassid ja käivitab rakenduse. SpringApplication klass on abiks Spring MVC või Spring REST rakenduse käivitamisel Spring Boot raamistiku abil, sest on märgitud *@SpringBootApplication* annotatsiooniga [25]. *@SpringBootApplication* annotatsioon on *@Configuration*, *@ComponentScan* ja *@EnableAutoConfiguration* annotatsioonide kombinatsioon ja pakub kolme annotatsioonide funktsionaalsust vaid ühe koodireaga.

@Configuration annotatsioon näitab, et klass deklareerib ühe või mitu *@Bean*-meetodit, mida töödeldakse Spring raamistiku konteineriga, et genereerida käitusajal @Bean-meetodite jaoks määratlusi ja teenusetaotlusi. Alates Spring Boot raamistiku teisest versioonist kirjutatakse bean-ide (või ubade) konfiguratsioonid .xml failidesse. Kuid alatest kolmandast versioonist saab Java failides kasutada bean-ide määratlusi. Seda nimetatakse Spring Java Config omaduseks (kasutades märkust *@Configuration*) [26]. Oad on objektid, mille meetodid on märgitud *@Bean* annotatsiooniga ja mida haldab Spring raamistiku IoC konteiner. IoC mõiste on kirjeldatud 2.1 pealõigus. Spring raamistiku *@ComponentScan* annotatsiooni kasutatakse koos *@Configuration* annotatsiooniga selleks, et märkida rakenduse pakette skaneerimiseks. Ilma argumentideta *@ComponentScan* annotatsiooni korral Spring raamistik skaneerib aktiivset paketti ja kõiki selle alampakette. *@EnableAutoConfiguration* annotatsioon automaatselt konfigureerib klassiruumis olevad oad [27]. Tagarakenduse süsteemi peaklassi kood on esitatud lisas *I*.

4.2.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community ja lähtekoodi redaktori Visual Studio Code veebirakenduse *iBuy* Spring Boot raamistiku peaklassi *IntelliSense*

Spring Boot raamistiku peaklassi kasutamise peamine eesmärk seisneb projekti käivitamises ning konfiguratsiooniklasside skaneerimises. Täpsemalt peaklassi tööst on kirjeldatud paragrahvis 4.2. Peaklassi realiseerimisel võetakse kasutusele erinevad paketid. Selleks, et paremini aru saada, kuidas suvalise paketi komponenti tuleks kasutatada, tuleb pöörata tähelepanu kasutatava arenduskeskkonna poolt pakutava *IntelliSense* peale. Integreeritud

arenduskeskkonna IIC ja lähtekoodiredaktori VSCode poolt võimaldatav *IntelliSense* sisaldab automaatset täitmist ning kiirteavet selles paragrahvis loetletud gruppide jaoks.

SpringFramework Boot grupi *IntelliSense*:

- liides *CommandLineRunner*
- annotatsioon *SpringApplication*
- klass *autoconfigure.SpringBootApplication*

Spring Beans grupi *IntelliSense*:

- annotatsioon *factory.annotation.Autowired*

SpringFramework Data grupi *IntelliSense*:

- klass *rest.core.config.RepositoryRestConfiguration*

4.3 Veebirakenduse *iBuy* Spring Boot raamistiku olemid, *@Entity*

POJO (ingl. keeles Plain Old Java Objects) on olemite objektid, mida saab andmebaasis säilitada. Olem esindab andmebaasi salvestatud tabelit. Olemi iga eksemplar tähistab tabeli rida andmebaasis [28]. Veebirakenduse *iBuy* tagarakenduse süsteemi kasutaja olem on esitatud lisas *II*. Toote olemi kood on esitatud lisas *III*.

4.3.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community veebirakenduse *iBuy* Spring Boot raamistiku olemite *IntelliSense*

Spring Boot raamistiku olemite kasutamise eesmärk seisneb objektide realiseerimises, et andmebaasi tabalites säilitada andmed nõutud kujul. Täpsemalt olemite tööst on kirjeldatud paragrahvis 4.3. Olemite realiseerimisel võetakse kasutusele erinevad paketid. Integreeritud arenduskeskkonna IIC ja lähtekoodiredaktori VSCode poolt pakutav *IntelliSense* sisaldab automaatset täitmist ning kiirteavet selles paragrahvis loetletud grupi ja paketi jaoks.

Projectlombok gruppi annotatsioonide *IntelliSense*:

- *lombok.Getter*
- *lombok.Setter*
- *lombok.AllArgsConstructor*

- *lombok.NoArgsConstructor*
- *lombok.ToString*

javax.persistence paketi *IntelliSense*:

- annotatsioon *javax.@Entity*
- annotatsioon *javax.@Id*
- klass *javax.@Column*
- annotatsioon *javax.@GenerationValue*

4.4 Veebirakenduse *iBuy* Spring Boot raamistiku hoidlad, *@RepositoryRestResource*

@Repository on hoidlat kirjeldav annotatsioon. Hoidla on ladustamise, otsimise ja otsingukäitumise kapseldamise mehhanism, mis jäljendab olemite kogumit. Seda imiteeritakse *@Component* annotatsiooni määramisel, mis võimaldab automaatselt tuvastada rakendusklassi klassiteed annotatsiooni *@ComponentScan* skaneerimise kaudu (vt paragrahv *4.1.1*) [29]. Annotatisoon *@RepositoryRestResource* sisaldab endas annotatsiooni *@Repository*, kuid lisaks annab võimaluse määrata teed REST-teenuste kasutamiseks *collectionResourceRel* ja *path* atribuutide määramisel. Vaikimisi on REST-teenuste tee määratud vastava hoidla olemi mitmuse nimetusega. Veebirakenduse *iBuy* kasutaja hoidla kood on esitatud lisas *IV*. Veebirakenduse *iBuy* toote hoidla kood on esitatud lisas *V*.

4.4.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community veebirakenduse *iBuy* Spring Boot raamistiku hoidlate *IntelliSense*

Spring Boot raamistiku hoidlate kasutamise eesmärgiks on andmebaasis olevate objektide filtreerimine. Täpsemalt hoidlate tööst on kirjeldatud paragrahvis 4.4. Hoidlate realiseerimisel võetakse kasutusele erinevad paketid. Integreeritud arenduskeskkonna IIC ja lähtekoodiredaktori VSCode poolt pakutav *IntelliSense* sisaldab automaatset täitmist ning kiirteavet selles paragrahvis loetletud grupi jaoks.

SpringFramework Data grupi *IntelliSense*:

- klass *domain.Pageable*
- klass *jpa.repository.JpaRepository*

- annotatsioon *repository.query.Param*
- annotatsioon *rest.core.annotation.RepositoryRestResource*
- annotatsioon *rest.core.annotation.RestResource*

4.5 Veebirakenduse *iBuy* Spring Boot raamistiku restkontrollerid, *@RestController*

Spring raamistiku annotatsioon *@RestController* on mugavusmärkus, mis on kombineeritud annotatsioonidest *@Controller* ja *@ResponseBody*. Seda annotatsiooni rakendatakse klassile, et märkida seda päringukäsitlejaks. Spring raamistiku annotatsiooni *@RestController* kasutatakse asünkroonsete veebiteenuste loomiseks, päringu andmete vastendamise eest vastutab päringukäsitleja meetod. Reageerimiskeha (ingl. keeles response body) genereeritakse käitleja meetodist ja vastus teisendatakse JSON- või XML- formaati [30]. *@Controller* annotatsioon toimib *@Component* annotatsiooni spetsialiseerumisena, võimaldades rakendusklasse automaatselt tuvastada klassitee skaneerimise kaudu [31]. Annotatsioon *@ResponseBody* annab kontrollerile teada, et tagastatud objekt teisendatakse automaatselt JSON-formaati ja edastatakse tagasi *HttpResponse* objekti.

Spring Web MVC raamistik võimaldab kasutada *mudeli-vaate-kontrolleri* arhitektuuri ja valmiskomponente, mida saab kasutada paindlike veebirakenduste arendamiseks. MVC mustri tulemusena eraldatakse rakenduse erinevad aspektid (sisendloogika, äriloogika ja kasutajaliidese loogika), pakkudes samal ajal nende elementide sidumist. Mudel kapseldab rakenduse andmed ja moodustab nendest POJO-d. Vaade vastutab mudeli andmete renderdamise eest ja genereerib HTML väljundi, mida kliendi brauser saab tõlgendada. Kontroller vastutab kasutaja taotluste töötlemise ja sobiva mudeli koostamise eest ning edastab selle renderdamiseks vaadetele [32].Veebirakenduse *iBuy* kasutaja restkontrolleri kood on esitatud lisas *VI*. Veebirakenduse *iBuy* toote REST-kontrolleri kood on esitatud lisas *VII*.

4.5.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community veebirakenduse *iBuy* Spring Boot raamistiku restkontrollerite *IntelliSense*

Spring Boot raamistiku kontrollerite kasutamise peamine eesmärk seisneb tarkvara kasutajatele vaadete renderdamises. Täpsemalt kontrollerite tööst on kirjeldatud paragrahvis 4.5. Kontrollerite realiseerimisel võetakse kasutusele erinevad paketid. Integreeritud

arenduskeskkona IIC ja lähtekoodiredaktori VSCode poolt pakutav *IntelliSense* sisaldab automaatse täitmise ning kiirteabe selles paragrahvis loetletud pakettide jaoks.

javax.persistence paketti *IntelliSense*:

- *enum org.springframework.http.HttpStatus*
- klass *org.springframework.http.ResponseEntity*

org.springframework.web.bind.annotation paketti annotatsioonide *IntelliSense*:

- *RequestBody*
- *PostMapping*
- *RequestMapping*
- *RestController*

4.6 Veebirakenduse *iBuy* Spring Boot raamistiku teenuseklass, *@Service*

Rakenduse teenuseklassid tähistatakse annotatsiooniga *@Service*, et näidata, et vastav klass hoiab äriloogikat teeninduskihina [33]. Teeninduskiht on selleks, et pakkuda käitumisloogikat kliendile saadetavate DAO[6]-de (vt *Lisa VIII*) andmete haldamiseks. Teenusekiht pakub koodi modulaarsust, äriloogikat ja reegleid, mis on täpsustatud teenusekihis, mis omakorda kutsub välja DAO kihti. DAO kiht vastutab sellisel juhul ainult andmebaasiga "suhtlemise" eest [34]. Veebirakenduse *iBuy* teenuseklassi kood on esitatud lisas *IX*.

4.6.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community veebirakenduse *iBuy* Spring Boot raamistiku teenuseklassi *IntelliSense*

Spring Boot raamistiku teenuseklassi kasutamise peamine eesmärk seisneb DAO-de andmete haldamises. Täpsemalt on teenuseklassi tööd kirjeldatud paragrahvis 4.6. Teenuseklassi realiseerimisel võetakse kasutusele erinevad paketid. Integreeritud arenduskeskkona IIC ja lähtekoodiredaktori VSCode poolt pakutav *IntelliSense* sisaldab automaatset täitmist ning kiirteavet selles paragrahvis loetletud grupi ja paketi jaoks.

[6] https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm

SpringFramework Security grupi klasside *IntelliSense*:

- *authentication.AuthenticationManager*
- *authentication.UsernamePasswordAuthenticationToken*
- *core.context.SecurityContextHolder*
- *crypto.password.PasswordEncoder*

org.springframework.stereotype paketi *IntelliSense*:

- annotatsioon *stereotype.Service*

4.7 Veebirakenduse *iBuy* Spring Boot raamistiku turvakonfiguratsiooni klass, *@EnableWebSecurity*

Spring Security on võimas ja lihtsasti kohandatav autentimis- ja juurdepääsukontrolli raamistik. Selle kasutamiseks tuleb määrata automatiseeritud rakendamiseks Spring Boot Security starteri *spring-boot-starter-security* sõltuvust. Spring security raamistiku annotatsioon *@EnableWebSecurity* on määratud klassi tasemel koos *@Configuration* annotatsiooniga, et tagada veebi turvalisust määratletud WebSecurityConfigurer liidese abiga. WebSecurityConfigurerAdapter on WebSecurityConfigurer liidese rakendusklass. Annotatsioon *@EnableWebSecurity* aotomaatselt lubab WebSecurityConfigurerAdapter klassiga määratletud veebiturvalisust. Selleks, et realiseerida oma veebirakenduse turvalisuse seadeid, tuleb WebSecurityConfigurerAdapter klassi laiendada ning selle meetodeid üle katta (ingl. keeles overriding methods) [35]. Veebirakenduse *iBuy* turvakonfiguratsiooni klassi kood on esitatud lisas *X*.

4.7.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community veebirakenduse *iBuy* Spring Boot raamistiku turvakonfiguratsiooni klassi *IntelliSense*

Spring Boot raamistiku turvakonfiguratsiooni klassi kasutamise eesmärgiks on veebiturvalisuse määramine. Turvakonfiguratsiooni klassi tööd on kirjeldatud paragrahvis 4.7. Turvakonfiguratsiooni klassi realiseerimisel võetakse kasutusele erinevad paketid. Integreeritud arenduskeskkona IIC ja lähtekoodiredaktori VSCode poolt pakuv *IntelliSense* sisaldab automaatset täitmist ning kiirteavet selles paragrahvis loetletud grupi ja paketi jaoks.

SpringFramework Security grupi *IntelliSense*:

- liides *authentication.AuthenticationManager*
- klass *config.annotation.authentication.builders.AuthenticationManagerBuilder*
- klass *config.annotation.web.builders.HttpSecurity*
- klass *config.annotation.web.builders.WebSecurity*
- annotatsiooon *config.annotation.web.configuration.EnableWebSecurity*
- klass *config.annotation.web.configuration.WebSecurityConfigurerAdapter*
- klass *crypto.bcrypt.BCryptPasswordEncoder*
- liides *crypto.password.PasswordEncoder*
- klass *web.authentication.UsernamePasswordAuthenticationFilter*
- klass *web.session.SessionManagementFilter*

org.springframework.http paketi *IntelliSense*:

- annotatsioon *config.annotation.web.configuration.EnableWebSecurey*

4.8 Üldine *IntelliSence'i* võrdlus integreeritud arenduskeskkonna IntelliJ IDEA Community ja lähtekoodiredaktori Visual Studio Code Veebirakenduse *iBuy* põhjal Spring Boot raamistiku jaoks

Analüüsides kõiki veebirakenduse *iBuy* Spring Boot raamistikku põhikomponte võib järeldada, et mõlemad arendusvahendid võimaldavad kasutada *IntelliSense'i* projektis olevate klasside, liideste ja annotatsioonide jaoks. Paljude pakettide kirjeldused on võetud Spring raamistiku viitedokumentatsioonist (https://docs.spring.io/spring-security/site/docs/current/api/), kuid mitte kõik (nt AuthenticationManager liides). Arenduskeskkonna IIC *IntelliSense'i* vihjes kuvatakse iga kasutatava starteri sõltuvuse kohta selle nimetus, kuid seda ei paku lähtekoodiredaktor VSCode.

Samas, lähtekoodiredaktor VSCode iga *IntelliSense'i* vihjes kuvatakse autor, mida ei leidu üheski arenduskeskkonna IIC *IntelliSense'i* vihjes. Lähtekoodiredaktor VSCode annab hoiatuse kui serialiseeritav klass ei deklareeri otseselt versiooninumbrit *serialVersionUID*. Seda kasutatakse objekti realiseerimise ajal, et kontrollida, kas serialiseeritud objekti saatja ja vastuvõtja on laadinud selle objekti jaoks klassid, mis ühilduvad serialiseerimisega. Kui vastuvõtja on laadinud klassi objekti, millel on erinev versiooninumber *serialVersionUID* kui saatjal, siis põhjustab selle väärtuse muutmine InvalidClassException erindi. Kui seerialiseeritav klass ei deklareeri *serialVersionUID* versiooninumbrit, siis omistatakse käivitamisel selle klassi *serialVersionUID* vaikimisväärtus [36]. Sellest hoiatusest saab loobuda muutes redaktori VS Code *IntelliSense'i* vaikeseadeid. Arenduskeskkonna IIC *IntelliSense'i* vaikeseaded sellist hoiatust ei kuva.

4.9 Angular raamistiku TypeScript programmeerimiskeel ja selle *IntelliSense*

TypeScript on Angular raamistikuga rakenduste arendamisel peamine programmeerimiskeel. See on JavaScript programmeerimiskeele ülemkomplekt, millel on disaini tugi turvalisuse ja tööriistade abil koodi redireegimise jaoks. Need tööriistad pakuvad täiustatud automaatset lõpuleviimist ja koodi navigeerimist. Integreeritud arenduskeskond IIC ei paku *IntelliSense'i* Angular raamistikul baseeruva programmeerimiskeele TypeScript jaoks.

4.10 Veebirakenduse *iBuy* Angular raamistiku mudelid

Angular raamistiku mudelid on objektide struktuuri andmehoidlad. Iga objekti struktuur on määratud klassis väljade tüüpidega ja nende väljade nimetustega. Need andmehoidlad reeglina toimetavad koos eesrakenduse süsteemi teenuseklassidega, mis omakorda kas edastavad need andmed kasutajale või annavad üle tagarakenduse süsteemi vastava teenuseklassile.

4.11 Veebirakenduse *iBuy* Angular raamistiku teenuseklassid

Angular raamistiku teenused on üksikud objektid, mida initsialiseeritakse rakenduse elutsükli jooksul ühe korra. Nad sisaldavad meetodeid, mis säilitavad andmeid kogu rakenduse elutsükli jooksul, st andmeid ei värskendata ja need on kogu aeg saadaval. Teenuse peamine eesmärk on korraldada ja jagada äriloogikat, mudeleid või mudelite funktsioone Angular raamistiku komponentide vahel [37].

4.11.1 Lähtekoodiredaktori Visual Studio Code veebirakenduse *iBuy* Angular raamistiku TypeScript programmeerimiskeele teenuseklasside *IntelliSense*

Komponentides pole lubatud andmete salvestamine. Komponentide loomisel peaks keskenduma andmete esitamisele ja komponentides tuleb delegeerida andmetele juurdepääsu teenusele. Lähtekoodiredaktor VSCode pakub *IntelliSense'i* järgmiste klasside jaoks: *@angular/common/http moodulist HttpClient*, *rxjs/operators* moodulist *map*, *rxjs* moodulist *Observable*.

5 Integreeritud arenduskeskkonna IntelliJ IDEA Community ja lähtekoodiredaktori Visual Studio Code versioonikontrollisüsteem

Versioonikontrollisüsteem (ingl. keeles VCS, Version Control System) on tööriist, mis aitab tarkvara arendusmeeskonnal lähtekoodi muudatusi hallata. Kui kasutatava teenuse hoidlasse üleslaetud kood sisaldab vigu, siis on võimalik vastava koodi varasemaid versioone üle vaadata, mis võiks aidata neid vigu parandada. Peamised eelised, mida versioonikontrollisüsteem pakub [38]:

- Säilitatakse iga faili täielik pikaajaline muudatuste ajalugu. See tähendab, et versioonikontrollisüsteem annab võimaluse vaadata igas failis tehtud muudatust tarkvaraarendaja poolt. Muudatused hõlmavad failide loomist ja kustutamist, samuti nende sisu muutmist. Erinevad VCS-i tööriistad erinevad selle poolest, kui hästi nad failide ümbernimetamist ja teisaldamist käsitlevad. Ajalugu peaks sisaldama ka autorit, kuupäeva ja kirjalikke märkusi iga muudatuse kohta. Tervikliku ajaloo omamine võimaldab naasta eelmistesse versioonidesse, et aidata tõrgete algpõhjuseid analüüsida, mis on hädavajalik tarkvara probleemide varasemate versioonide parandamiseks.
- Hargnemist ja kokku sidumist (ingl. keeles branching and merging). Meeskonnaliikmete ühiste failidega samaaegne töötamine ei ole mõistlik. Versioonikontrollisüsteemi tööriistades haru loomine hoiab mitu töövoogu üksteisest sõltumatuna, pakkudes samal ajal ka võimalust need vood uuesti kokku siduda, võimaldades arendajatel kontrollida, kas iga haru muudatused on vastuolus või mitte. Arendamise protsessis tekib palju erinevaid töövooge, millest meeskonnad saavad valida, kuidas versioonikontrollisüsteemis hargnemis- ja ühinemisvõimalusi kasutada.

5.1 Integreeritud arenduskeskkonna IntelliJ IDEA Community versioonikontrollisüsteem

Faili muutmisel integreeritud arenduskeskkonna IIC versioonikontrollisüsteem märgendab muudetud koodiread ja projekti struktuuri vastava failinime. Märgendatud muudatust saab tagasi lükata või vaadata täpsemalt, mis muudatus on tehtud ehk millised koodiread on muutunud.

Projektis saab *Version Control* menüüst *Local Changes* tööriba valikul vaadata muudetud failide nimekirja. Valides konkreetse faili sellest nimekirjast, kuvab versioonikontrollsüsteem

vastava faili muudetud ridu, märgendades uusi koodiridu rohelise värviga ning eelmistel koodiridadel (nendel, mis olid kirjutatud enne muudatusi) kuvab nuppu *Revert*, mis annab võimaluse vastava koodirea muudatust taastada. Valides faili *Local Changes* ning vajutades hiire parema klahvi, ilmub hajutatud versioonikontrollisüsteemi Git tööriba, mille peamisteks tööülesannete valikuteks on koodiridade annoteerimine, faili kehtestamine ja nii lokaalsete kui ka kaughoidla töövoogude võrdlemine. Kasutades annoteerimise valikut, märgendab versioonikontrollisüsteem vastava faili kõik koodiread kuupäevaga, mis näitab, millal kehtestatud koodirida on lükatud kaughoidlasse ja töövoogu nimetusega, mis näitab, millise töövoo pealt muudatust tehti.

Kasutades faili kehtestamise valikut, kuvab versioonikontrollisüsteem muudatuste nimekirja, kust saab valida faile kehtestamiseks. Peale failide positiivset kehtestamist saab muudetud faile lükata initsialiseeritud projekti kaughoidlasse. Peale failide valikut saab kasutada järgmisi kontrollvõimalusi:

- Koodi vormindamine. Koodi vormindamise reeglid on määratud arenduskeskkonna seadmetes *File/Settings/Editor/Code Style.*
- Imporditavate pakettide optimeerimine. Valitud failid kontrollitakse üle ning mittekasutatavate pakettide import kustutakse failidest ära.
- *TODO* märgendite kontroll. Versioonikontrollisüsteem kontrollib, kas koodis on jäänud kusagil *TODO* märgend või mitte. Kui jah, siis valitud muudatuste nimekirjast faile pole võimalik kehtestada.

Kasutades lokaalsete ja kaughoidla töövoogude võrdlemist, saab aktiivses töövoos valitud faili võrrelda mõne muu lokaalses või kaughoidla voos oleva sama failiga. Versioonikontrollsüsteem märgendab võrreldavas voos faili kustatud ridu halli värviga, aktiivses voos muudetud ridu sinise värviga ning lisatud ridu rohelise värviga.

Arenduskeskkonna *Version Control* menüüst *Log* valikul saab vaadata projekti töövoogude struktuuri ehk töövoogude puud, mis kujutab endast projektis olemasolevate töövoogude hargnemist ja kokku sidumist ning iga hargnemise ja sidumise muudatuste kehtestamise nimetust. Iga muudatuste kehtestamine sisaldab endas GIT-kasutaja nimetust ja kehtestamise kuupäeva. Puud saab filtreerida töövoogude, kuupäevade jt parameetrite kaudu [39].

5.2 Lähtekoodiredaktori Visual Studio Code versioonikontrollisüsteem

Lähtekoodiredaktori VSCode versioonikontrollsüsteemi *Source* juhtimise kasutamisel kuvatakse projekti muudetud failide nimekiri. Nimekirjast faili valimisel märgendab versioonikontrollisüsteem aktiivse töövoo muudetud ridu rohelise värviga ning võrreldava töövoo vastavad read punase värviga. Peamised tööülesanded, mida *Source* juhtimise kontroll pakub, on muudatuste kehtestamine ja kaughoidlasse lükkamine ning lokaalse hoidla uuendamine vastavalt kaughoidla faili versioonidele. Töövoogude puud lähtekoodiredaktor VSCode ei paku.

6 Tulemused

Arenduskeskkonnad võimaldavad programmeerijatel lihtustada koodi kirjutamist, pakkudes selleks konkreetseid haldamisvõimalusi. Antud töö raames on haldamisvõimaluste valik orienteeritud Spring Boot ja Angular raamistikega töötamisele ja keskkondi võrreldakse järgmiste kriteeriumite suhtes: Spring Initializr, Flyway andmebaaside migreerimise tööriist, MySQL relatsioonandmebaasi haldussüsteemi tugi, andmebaasi päringumeetodid, Project Lombok annotatsioonid, IntelliSense (vt paragrahv 1.1.2) ja VCS (vt peatükk 5).

6.1 Spring Initializr

Spring Initializr on tööriist, mis võimaldab luua projekti struktuuri Gradle või Maven ehitamis automatiseerimissüsteemi abil Spring Boot raamistikuga töötamiseks (vt paragrahv *3.1*). Töö raames on uuritud integreeritud arenduskeskkonna IIC ja lähtekoodiredaktori VSCode tööriista Spring Initializr haldamisvõimalused ning saadud tulemused on illustreeritud tabelis *6.1*.

Tabel 6.1. Tööriista Spring Initializr haldamisvõimaluste võrdlemine.

	IntelliJ IDEA Community	**Visual Studio Code**
Programmeerimiskeele määramine	Puudub	Olemas
Projekti nimetuse määramine	Puudub	Olemas
Spring Boot raamistiku versiooni määramine	Puudub	Olemas
Funktsionaalsete sõltuvuste lisamine	Puudub	Olemas

6.2 Flyway andmebaaside migreerimise tööriist

Flyway on avatud lähtekoodiga andmebaaside migreerimise tööriist, mida kasutatakse andmebaasi migreerimiseks ühelt platvormilt teisele muutmata andmebaasi struktuuri, mis tulemusena kiirendab arendamist (vt paragrahv *3.2*). Töö raames on uuritud integreeritud arenduskeskkonna IIC ja lähtekoodiredaktori VSCode andmebaaside migreerimise tööriista Flyway haldamisvõimalusi ning saadud tulemused on illustreeritud tabelis *6.2.*

Tabel 6.2. Andmebaaside migreerimise tööriista Flyway haldamisvõimaluste võrdlemine.

	IntelliJ IDEA Community	**Visual Studio Code**
SQL skriptide automaatne täitmine	Piiratud *snipettid*	Piiratud (Vastavalt olemasolevatele skriptidele)
SQL-päringute tulemused .json- või .csv-failivormingusse	Puudub	Olemas
Korduvkasutatavad koodiplokid	Puudub	Olemas

6.3 Relatsioonandmebaasi haldussüsteemi MySQL tugi

MySQL on Oracle'i toega avatud lähtekoodiga relatsioonilise andmebaasi haldussüsteem (vt paragrahv *3.3*). Töö raames on uuritud integreeritud arenduskeskkonna IIC ja lähtekoodiredaktori VSCode relatsioonandmebaasi haldussüsteemi MySQL toe haldamisvõimalusi ning saadud tulemused on illustreeritud tabelis *6.3*.

Tabel 6.3. Relatsioonandmebaasi haldussüsteemi MySQL toe haldamisvõimaluste võrdlemine.

	IntelliJ IDEA Community	**Visual Studio Code**
Andmebaasiühenduse haldus	Olemas	Olemas
Skriptide tugi	Olemas	Olemas
Päringute ajalugu	Olemas	Olemas

6.4 Andmebaasi päringumeetodid

Päringumeetodid muudavad andmebaasid võimsaks vahendiks andmete filtreerimisel ja hoidmisel. Päringumeetodeid kirjutatakse vastavalt SQL domeenispetsiifilisele keelele tuginedes programmeerimiskeele süntaksile (vt paragrahv *3.4*). Töö raames on uuritud integreeritud arenduskeskkonna IIC ja lähtekoodiredaktori VSCode andmebaasi päringumeetodite haldamisvõimalusi ning saadud tulemused on illustreeritud tabelis *6.4.*

Tabel 6.4. Andmebaasi päringute haldamisvõimaluste võrdlemine.

	IntelliJ IDEA Community	**Visual Studio Code**
Spring Data JPA päringumeetodid	Puudub	Puudub
Andmete filtreerimine	Olemas	Olemas

6.5 Project Lombok

Project Lombok on Java teek, mille idee on asendada Vanilla Java koodiplokid Lombok annotatsioonidega, mis teeb klasside koodi kompaktsemaks ja loetavamaks (vt paragrahv *3.5*). Töö raames on uuritud integreeritud arenduskeskkonna IIC ja lähtekoodiredaktori VSCode Project Lombok teeki haldamisvõimalusi ning saadud tulemused on illustreeritud Venni diagrammi kasutades joonisel *6.5*.

Joonis 6.5. Project Lombok teeki haldamisvõimaluste võrdlemine.

6.6 *IntelliSense* Spring Boot ja Angular raamistike jaoks

IntelliSense on koodi lõpuleviimise abivahend, mis sisaldab kaks peamist funktsiooni: liikmete loendit ja kiirteavet (vt paragrahv *1.1.2*). Töö raames on uuritud integreeritud arenduskeskkonna IIC Community ja lähtekoodiredaktori VSCode *IntelliSense*'i haldamisvõimalusi Spring Boot ja Angular raamistike jaoks. Üldine järeldus Spring Boot raamistiku jaoks on kirjeldadud paragrahvis *4.8.* Integreeritud IIC ei paku *IntelliSense'i* Angular raamistiku baseeruva programmeerimiskeele TypeScript jaoks ja lähtekoodiredaktori VSCode veebirakenduse *iBuy* Angular raamistiku TypeScript programmeerimiskeele teenuseklasside *IntelliSense* on kirjeldatud paragrahvis *4.11.1*.

6.7 Versioonikontrollisüsteem

Versioonikontrollisüsteem on tarkvarariist, mis aitab tarkvarameeskonnal lähtekoodi muudatusi hallata. Kui kasutatava teenuse hoidlasse üleslaetud kood tekitab vea, siis saab vastava koodi varasemaid versioone üle vaadata, mis võiks aidata seda viga parandada (vt peatükk *5*). Töö raames on uuritud integreeritud arenduskeskkonna IIC ja lähtekoodiredaktori VSCode versioonikontrollisüsteemi haldamisvõimalusi ning saadud tulemused on illustreeritud tabelis *6.7.*

Tabel 6.7. Versioonikontrollisüsteemi haldamisvõimaluste võrdlemine.

	IntelliJ IDEA Community	**Visual Studio Code**
Muudetud koodiridade märgendamine	Olemas	Olemas
Muudatuse tagasilükkamine	Olemas	Olemas
Uute ning vanade koodiridade värviliselt märgendamine	Olemas	Olemas
Koodiridade annoteerimine	Olemas	Puudub
Faili kehtestamine	Olemas	Olemas
Lokaalsete ja kaughoidla töövoogude võrdlemine	Olemas	Puudub
Koodi vormindamine	Olemas	Puudub
Imporditavate pakketide optimeerimine	Olemas	Olemas
TODO märgendite kontroll	Olemas	Puudub
Töövoogude puu hargnemine ja kokku sidumine	Olemas	Puudub

6.8 Üldised tulemused

Tuginedes kuuendas peatükis toodud tulemustele saab teha järelduse, et nii integreeritud arenduskeskkonnal IIC kui ka lähtekoodiredaktoril VSCode on oma eelised ja puudused seoses Spring Boot ja Angular raamistike haldamisvõimalustega veebiarendusel. Üldiselt aga

tulemuste tabelid ning joonis näitavad, et lähtekoodiredaktor VSCode on rohkem orienteeritud just Spring Boot ja Angular raamistike kooskasutamisele, kuna see pakub haldamisvõimalust Spring Boot raamistiku Spring Initializr tööriista jaoks, mida ei paku IIC keskkond. Lähtekoodiredaktor VSCode pakub *IntelliSense'i* võimalusi Angular raamistiku TypeScript programmeerimiskeele jaoks, mida samuti ei paku IIC keskkond. Võib teha järelduse, et integreeritud arenduskeskkond IIC on rohkem suunatud Java programmeerimiskeele kasutamisele, kuna see annab võmaluse kasutada suuremal hulgal Project Lombok annotatsioone võrreldes lähtekoodiredaktoriga VSCode ning selle versioonikontrollisüsteem sisaldab rohkem funktsionaalseid võimalusi lähtekoodi haldamiseks.

7 Kokkuvõte

Käesoleva töö eesmärk oli luua veebirakendus Spring Boot ja Angular raamistikke kasutades ning veebirakenduse arendamise käigus võrrelda vahendite haldamisvõimalusi mõlema raamistiku jaoks. Vastavalt võrdlemise tulemusele otsustada, milline vahend sobib paremini veebi arendamiseks nende raamistike kasutamisel. Töö käigus uuriti põhilisi haldamisvõimalusi Spring Boot ja Angular raamistike jaoks integreeritud arenduskeskkonnas IIC ja lähtekoodiredaktoris VSCode. Ühtlasi loodi veebirakendus iBuy selleks, et rakenduse arendamise käigus võtta kasutusele töös vaadeldavad haldamisvõimaluste kriteeriumid ja võrrelda nende võimalusi mõlemas arendusvahendis. Tulemuste analüüsides järeldati, et lähtekoodiredaktor VSCode on rohkem orienteeritud Spring Boot ja Angular raamistike kooskasutamisele ja integreeritud arenduskeskkond IIC on rohkem suunatud Java programmeerimiskeele kasutamisele.

Antud töö teemaga seotud võimalikeks edasiarendusteks on testimise haldamisvõimaluste võrdlemine tagarakenduse süsteemi Spring Boot raamistiku jaoks ja eesrakenduse süsteemi Angular raamistiku jaoks. Uurimist vajaks ka mälukasutus mõlemas arenduskeskkonnas.

Kasutatud kirjandus

[1] Codecademy. What is an IDE? https://www.codecademy.com/articles/what-is-an-ide (30.11.2019)

[2] Red Hat. What is an IDE? https://www.redhat.com/en/topics/middleware/what-is-ide (30.11.2019)

[3] JournalDev. Key Components and Internals of Spring Boot Framework. https://www.journaldev.com/7989/key-components-and-internals-of-spring-boot-framework (28.03.2020)

[4] Spring Boot Starter Actuator. https://www.javatpoint.com/spring-boot-actuator (28.03.2020)

[5] Shubham Sinha. Edureka! Angular Tutorial: Getting Started With Angular 8. https://www.edureka.co/blog/angular-tutorial/ (19.02.2020)

[6] Todd Motto. A deep dive on Angular decorators. https://ultimatecourses.com/blog/angular-decorators (19.02.2020)

[7] Angular. Introduction to Angular concepts. https://angular.io/guide/architecture (19.02.2020)

[8] Baeldung. Database Migrations with Flyway. https://www.baeldung.com/database-migrations-with-flyway (26.01.2020)

[9] Database Navigator.

https://confluence.jetbrains.com/display/CONTEST/Database+Navigator (26.01.2020)

[10] JetBrains. MaxCompute Studio. https://plugins.jetbrains.com/plugin/9193-maxcompute-studio (26.01.2020)

[11] Microsoft. SQL Server (mssql).
https://marketplace.visualstudio.com/items?itemName=ms-mssql.mssql (26.01.2020)

[12] Margaret Rouse. MySQL. https://searchoracle.techtarget.com/definition/MySQL (26.01.2020)

[13] Microsoft. Language PL/SQL.
https://marketplace.visualstudio.com/items?itemName=xyz.plsql-language (28.01.2020)

[14] Microsoft. SQLTools - Database tools.
https://marketplace.visualstudio.com/items?itemName=mtxr.sqltools (28.01.2020)

[15] Querying a Database.
https://www.quackit.com/database/tutorial/querying_a_database.cfm (01.02.2020)

[16] Spring by Pivotal. Spring Data JPA - Reference Documentation.
https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods
(03.02.2020)

[17] Joby Wilson Mathews. What is difference between CrudRepository and JpaRepository interfaces in Spring Data JPA? Vastus 3, 10.03.2018.

https://stackoverflow.com/questions/14014086/what-is-difference-between-crudrepository-and-jparepository-interfaces-in-spring (06.02.2020)

[18] Jetbrains. Creating annotation profile. https://www.jetbrains.com/help/idea/configuring-annotation-processing.html#create_profile (07.02.2020)

[19] @NoArgsConstructor, @RequiredArgsConstructor, @AllArgsConstructor.
https://projectlombok.org/features/constructor (13.02.2020)

[20] @EqualsAndHashCode. https://projectlombok.org/features/EqualsAndHashCode
(16.02.2020)

[21] Lombok @ToString. http://www.javabyexamples.com/delombok-tostring/ (13.02.2020)

[22] @Value. https://projectlombok.org/features/Value (16.02.2020)

[23] Lombok plugin. https://plugins.jetbrains.com/plugin/6317-lombok (07.02.2020)

[24] Gabriel Basilio Brito. Microsoft. Lombok Annotations Support for VS Code
https://marketplace.visualstudio.com/items?itemName=GabrielBB.vscode-lombok
(07.02.2020)

[25] Spring Boot @SpringBootApplication, SpringApplication Class.
https://www.journaldev.com/21556/springbootapplication-springapplication (26.02.2020)

[26] What is a Spring Bean? https://www.baeldung.com/spring-bean (26.02.2020)

[27] The @SpringBootApplication Annotation Example in Java + Spring Boot.
https://dzone.com/articles/the-springbootapplication-annotation-example-in-ja (26.02.2020)

[28] Vivek Balasubramaniam. Defining JPA Entities. https://www.baeldung.com/jpa-entities (22.02.2020)

[29] Spring Boot @Repository. http://zetcode.com/springboot/repository/ (25.02.2020)

[30] Annotation Type Controller. https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/stereotype/Controller.html (28.02.2020)

[31] Spring RestController. https://www.journaldev.com/21536/spring-restcontroller (28.02.2020)

[32] Spring - MVC Framework. TutorialsPoint. https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm (28.02.2020)

[33] @Component vs @Repository and @Service in Spring. https://www.baeldung.com/spring-component-repository-service (29.02.2020)

[34] Alam Khan. Why to use Service Layer in Spring MVC? https://medium.com/stackavenue/why-to-use-service-layer-in-spring-mvc-5f4fc52643c0 (29.02.2020)

[35] Arvind Rai. Spring @EnableWebSecurity Example. https://www.concretepage.com/spring-5/spring-enablewebsecurity-example (29.02.2020)

[36] Jon Skeet. What is a serialVersionUID and why should I use it? Vastus 1, 10.03.2018. https://stackoverflow.com/questions/285793/what-is-a-serialversionuid-and-why-should-i-use-it (04.02.2020)

[37] Devquora Sharad. What Is a Service in Angular and Why Should You Use it? https://dzone.com/articles/what-is-a-service-in-angular-js-why-to-use-it (05.02.2020)

[38] Atlassian. What is version control. https://www.atlassian.com/git/tutorials/what-is-version-control (06.02.2020)

[39] JetBrains. Version control. https://www.jetbrains.com/help/idea/version-control-integration.html (06.03.2020)

Lisad

I. Tagarakenduse süsteemi peaklassi kood

```
@SpringBootApplication
public class IbuyApplication implements CommandLineRunner {

    @Autowired
    private ProductRepository productRepository;

    @Autowired
    private RepositoryRestConfiguration repositoryRestConfiguration;

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

```
    @Override
    public void run(String... args) {
        repositoryRestConfiguration.exposeIdsFor(Product.class);
        repositoryRestConfiguration.exposeIdsFor(User.class);

        Set<Product> products = Set.of(new Product(1L, "Lenovo laptop",
1200, 10),
                new Product(2L, "HP laptop", 1300, 10),
                new Product(3L, "Lenovo laptop", 1200, 10),
                new Product(4L, "HP laptop", 1300, 10),
                new Product(5L, "Lenovo laptop", 1200, 10),
                new Product(6L, "HP laptop", 1300, 10),
                new Product(7L, "Lenovo laptop", 1200, 10),
                new Product(8L, "HP laptop", 1300, 10),
                new Product(9L, "Lenovo laptop", 1200, 10),
                new Product(10L, "HP laptop", 1300, 10)
                );

        productRepository.saveAll(products);
        productRepository.findAll().forEach(p ->
System.out.println(p.toString()));
    }

}
```

II. Tagarekenduse süsteemi kasutaja olemi kood

```
@Entity
@Getter @Setter
@NoArgsConstructor @AllArgsConstructor
public class User {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique=true)
    private String username;

    @Column
    private String password;

    @Column
    private String email;

}
```

III. Tagarekenduse süsteemi toote olemi kood

```
@Entity
@Getter @Setter
@NoArgsConstructor @AllArgsConstructor
public class Product implements Serializable {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column
    private String description;

    @Column
    private double price;

    @Column
    private int quantity;

}
```

IV. Tagarekenduse süsteemi kasutaja hoidla kood

```
@RepositoryRestResource
public interface UserRepository extends JpaRepository<User, Long> {

    Optional<User> findByUsername(String username);

}
```

V. Tagarekenduse süsteemi toote hoidla kood

```
@RepositoryRestResource
public interface ProductRepository extends JpaRepository<Product, Long> {

    @RestResource(path = "/by_description")
    List<Product> findByDescriptionContains(@Param("mc") String
description);

    @RestResource(path = "/by_description_page")
    Page<Product> findByDescriptionContains(@Param("mc") String
description, Pageable pageable);

}
```

VI. Tagarekenduse süsteemi kasutaja REST-kontrolleri kood

```
@RestController
@RequestMapping(value = "/list_users")
public class UserController {

    private final UserRepository userRepository;

    public UserController(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @GetMapping()
    public List<User> listUsers() {
        return userRepository.findAll();
    }

    @GetMapping(value = "/{id}")
    public User getUser(@PathVariable(name = "id") Long id) {
        return userRepository.findById(id).orElse(null);
    }

    @DeleteMapping(value = "/{id}")
    public void deleteUser(@PathVariable(name = "id") Long id) {
        userRepository.deleteById(id);
    }

}
```

VII. Tagarekenduse süsteemi toote REST-kontrolleri kood

```
@RestController
@RequestMapping(value = "/list_products")
public class ProductController {

    @Autowired
    private ProductRepository productRepository;

    @GetMapping()
    public List<Product> listProducts() {
        return productRepository.findAll();
    }

    @GetMapping(value = "/{id}")
    public Product getProduct(@PathVariable(name = "id") Long id) {
        return productRepository.findById(id).get();
    }

    @PutMapping(value = "/{id}")
    public Product updateProduct(@PathVariable(name = "id") Long id,
@RequestBody Product product) {
        product.setId(id);

        return productRepository.save(product);
    }

    @PostMapping(value = "/{id}")
    public Product saveProduct(@RequestBody Product product) {
        return productRepository.save(product);
    }
```

```
    @DeleteMapping(value = "/{id}")
    public void deleteProduct(@PathVariable(name = "id") Long id) {
        productRepository.deleteById(id);
    }

}
```

VIII. DAO

Andmepöördusobjekt (DAO) on muster, mis annab abstraktse liidese teatud tüüpi andmebaasidele või muudele püsivusmehhanismidele. Selle peamised komponendid:

- Andmetele juurdepääsu objekti liides - see liides määratleb mudelobjektidega tehtavaid standardtoiminguid.
- Data Access Object konkreetne klass - see klass rakendab ülaltoodud liidest. See klass vastutab andmete hankimise eest andmeallikast, mis võib olla andmebaas või mis tahes muu salvestusmehhanism.
- Mudeliobjekt või väärtusobjekt - see objekt on lihtne POJO, mis sisaldab get/set-meetodeid DAO-klassi abil saadud andmete salvestamiseks.

IX. Tagarekenduse süsteemi teenuseklassi kood

```
@Service
public class AuthService {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private JwtProvider jwtProvider;

    public void signup(RegisterRequest registerRequest) {
        User user = new User();
        user.setUsername(registerRequest.getUsername());
        user.setPassword(encodePassword(registerRequest.getPassword()));
        user.setEmail(registerRequest.getEmail());
        userRepository.save(user);
    }

    private String encodePassword(String password) {
        return passwordEncoder.encode(password);
    }

    public AuthenticationResponse login(LoginRequest loginRequest) {
        Authentication authenticate = authenticationManager
                .authenticate(new
UsernamePasswordAuthenticationToken(loginRequest.getUsername(),
loginRequest.getPassword()));
        SecurityContextHolder.getContext().setAuthentication(authenticate);
        String authenticationToken =
jwtProvider.generateToken(authenticate);
```

```
        return new AuthenticationResponse(authenticationToken,
loginRequest.getUsername());
    }


}
```

X. Tagarekenduse süsteemi turvakonfiguratsiooni klassi kood

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsServiceImpl userDetailsService;

    @Bean
    public JwtAuthenticationFilter jwtAuthenticationFilter() {
        return new JwtAuthenticationFilter();
    }

    @Bean
    CorsFilter corsFilterCustom() {
        return new CorsFilter();
    }

    @Override
    public void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.csrf().disable().authorizeRequests()
                .antMatchers("/**")
                .permitAll()
                .anyRequest()
                .authenticated();

        httpSecurity.cors();
        httpSecurity.addFilterBefore(corsFilterCustom(),
SessionManagementFilter.class);
        httpSecurity.addFilterBefore(jwtAuthenticationFilter(),
UsernamePasswordAuthenticationFilter.class);
    }
```

```
@Autowired
    public void configure(AuthenticationManagerBuilder
authenticationManagerBuilder) throws Exception {

authenticationManagerBuilder.userDetailsService(userDetailsService).passwor
dEncoder(passwordEncoder());
    }

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

    @Override
    public void configure(WebSecurity web) {
        web.ignoring().antMatchers(HttpMethod.OPTIONS, "/**");
    }

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

}
```

XI. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, _____ Artjom Aralov_____,

(autori nimi)

1. Annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose _____ Integreeritud arenduskeskkonna IntelliJ IDEA Community ja lähtekoodiredaktori Visual Studio Code haldamisvõimaluste võrdlemine veebirakenduse loomisel Spring Boot ja Angular raamistike jaoks _____,

   (lõputöö pealkiri)

   Mille juhendaja on _____ Helle Hein _____,

   (juhendaja nimi)

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, alates pp.kk.aaaa kuni autoriõiguse kehtivuse lõppemiseni.
3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.
4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Artjom Aralov 08.05.2020