TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kuldar Vakker 206576IADB

# Polümorfsete andmeedastusobjektide väljade ja väljadevaheliste sõltuvuste valideerimine

Bakalaureusetöö

Juhendaja: Andres Käver
Magistrikraad
Anton Sauh
Bakalaureusekraad

Tallinn 2024

# Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kuldar Vakker

04.01.2024

# Annotatsioon

Antud bakalaureusetöö eesmärk on välja töötada jätkusuutlik süsteem polümorfsete andmeedastusobjektide väljade ja nendevaheliste sõltuvuste valideerimisreeglite kirjeldamiseks, mida saab kasutada Java veebirakenduses, mille programmikood on kirjutatud Kotlinis. Lahendus peab tagama kirjeldamise kasutatavust ning kasutust agregeeritud ärimudeliga rakendusel.

Lõputöö käigus kasutati andmemudeli agregeerimise tarbeks polümorfseid andmeedastusobjekte toetavat objektimis- ja jadastusteeki. Analüüsiti valideerimisteekide rakendamist agregeeritud ärimudelil. Valideerimisteekide võrdlus teostati kasutades analüütiliste hierarhiate meetodit.

Analüüsi põhjal töötati välja vähendatud äririskiga agregeeritud ärimudeli kirjeldamise süsteem. Antud töö tulemused on kasulikud Kotlinis kirjutatud veebirakendustega tegutsevatele arendajatele, kes soovivad optimeerida andmeedastusobjektide valideerimist. Lisaks keerukate valideerimisreeglite efektiivsemale rakendamisele soodustab tulemus uute, tõhusamate ja kvaliteetsemate tarkvaralahenduste loomist.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 4 peatükki, 20 joonist, 2 tabelit.

# Abstract

Field and Cross-Field Validation of Polymorphic Data Transfer Objects

The purpose of the thesis is to find a sustainable solution for validating polymorphic data transfer object fields and cross-fields on Java web application written in Kotlin. The solution must ensure the usability of validation rules and its use in an application with an aggregator business model.

The thesis covers usage of serialization and deserialization library, which supports polymorphic data transfer objects. The implementation of various validation libraries on an aggregator business model was analysed. The comparison of validation libraries was performed using the Analytic Hierarchy Process.

A system for describing an aggregated business model with reduced business risk was developed based on the analysis. The findings of this thesis are valuable for software developers operating on the Java web application written in Kotlin and who want to optimise validation of the object. In addition to simplifying the formulation of complex validation rules and thereby making the process more efficient. The result of this thesis promotes the creation of new, more efficient, and higher quality software solutions.

The thesis is in Estonian and contains 35 pages of text, 4 chapters, 20 figures, 2 tables.

# Lühendite ja mõistete sõnastik

| | |
|---|---|
| Agregeerima | Koondama, kokku võtma |
| AHP | *Analytic hierarchy process*, analüütiliste hierarhiate meetod |
| Annotatsioon | *Annotation*, Java metaandmete märgend, mis lisavad täiendavat teavet või programmikoodi |
| API | *Application programming interface*, rakendusliides |
| Asünkroonne | Sõltumatu |
| DRY | *Don't repeat yourself,* üldlevinud koodi kirjutamise reegel, mis toetab taaskasutatava koodi kirjutamist |
| DSL | *Domain-Specific Language*, valdkonnaspetsiifiline keel |
| Eesrakendus | *Front end application*, esitluskomponent kasutajale, visualiseerib tagarakenduse |
| Erind | *Exception*, erandolukord programmi täitumisel |
| *Fluent interface* | Meetodite defineerimise viis, mis tagastab objekti, millele ise kuulub |
| Gradle Wrapper | Skript, mis vajadusel laeb alla vastava versiooniga Gradle tarkvara |
| HTTP | *HyperText Transfer Protocol*, veebis kasutatav andmeedastusprotokoll |
| IDE | *Integrated development environment*, liidestatud tarkvaraarendus keskkond |
| Jadastus | *Serialization*, andmeobjekti muutmine baidi- või tekstijadaks |
| Jakarta EE | *Jakarta Enterprise Edition,* kogukonna juhitud standard Java rakenduste arendamiseks, asendab Java EE ning haldab Eclipse Foundation |
| Java EE | *Java Platform, Enterprise Edition*, kogukonna juhitud standard Java rakenduste arendamiseks, mida haldab Oracle |
| JSON | *JavaScript Object Notation*, andmete formaat |
| Korduvalt kasutatav | *Boilerplate*, rutiinselt lisatav |
| KSP | *Kotlin Symbol Processing*, Kotlin kompilaatori lisafunktsionaalsuste arendamiseks mõeldud API |
| Kõrvutama | Samale tasemele seadma |
| Liidestus | *Integration*, integratsioon, rakenduste vaheline ühendus |

| | |
|---|---|
| Objektimine | *Deserialization*, baidi- või tekstijadast muutmine andmeobjektiks, jadastuse pöördtoiming |
| OpenAPI | Standard kirjeldamaks veebirakendusliidest |
| *Race condition* | Olukord, kus ühele ressursile on ligipääs mitmel lõimel ning mille väljundväärtus sõltub välistest sündmustest |
| Skript | *Script*, väike käsujada |
| SMS | *Short message service*, lühisõnumiteenus |
| Sõltuvus | *Dependency*, rakenduse tööks vajalik projektiväline koodiosa |
| Tagarakendus | *Back end application*, kasutajale nähtamatu komponent, töötlemissüsteem |
| Tagasiühilduvus | *Backward compatibility*, tarkvara omadus toimida eelneva versiooni nõuetega |
| Täitmisaegne | *Runtime*, käitumisaegne |

# Sisukord

# Jooniste loetelu

# Tabelite loetelu

# 1 Sissejuhatus

Tarkvaraarendaja kulutab pea pool koodi kirjutamise ajast silumisele [1]. Mitmete tegurite tõttu ei ole võimalik koodi esmakordsel kirjutamisel saavutada täielikku korrektsust. Nende tegurite hulka võivad kuuluda keerulised äriloogilised protsessid, ajaline piirang, tarkvarateekide ja -raamistike eripärad, edasiarendused, koostööprotsessi ebakõlad teiste arendajatega ning muutuvad nõuded.

2022. aasta sügisel ning jätkudes veel ka 2023. aastal toimus suurtes infotehnoloogia firmades mitmeid koondamislaineid [2]. Selle tõttu tekkis isegi spekulatsioone Twitteri teenuste lõppemisest, kuna arvati, et ei suudeta hoida töös olemasolevat tarkvara [3]. Siiski toimib ettevõte tänaseni ning see on näide tarkvaraarendajate suurest vastutuskoormast, mis on seotud muutuvast keskkonnast või koodi ebatäpsusest jäljendada keskkonda.

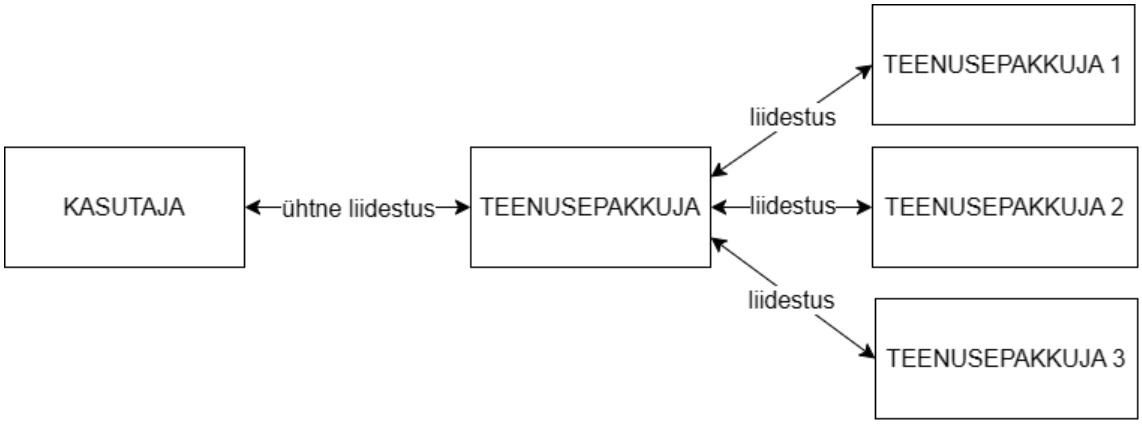
Samal ajal tuleb arendajal toetada uute äriliste eesmärkide saavutamist, mis üldjuhul eeldab ka kiiret arendusprotsessi. Antud protsessi kiirust on võimalik tõsta ja dikteerida kasutades valmis tarkvarakomponente ning äriloogiliste protsesside lihtsamaks mõistmisele suunatud arendusprintsiipe.

## 1.1 Agregeeritud ärimudel

Tehnoloogiahiidude Lyft ja Uberi kiire raha kaasamise edu tugineb agregeeritud ärimudelil. Selle mudeli kontseptsioon seisneb erinevate teenusepakkujate ja toodete koondamist ühisele platvormile, mille teenused on kasutajale standardiseeritud ja struktureeritud. [4] Sarnase ärimudeli rakendamist võib leida ka Eestis tegutsevatelt ettevõttetelt nagu Twilio ja Maksekeskus. Twilio pakub mobiilioperaatorite lühisõnumiteenuste (SMS) vahendust [5]. Maksekeskus pakub pangalinkide vahendust [6].

All järgneval joonisel (Joonis 1) on nummerdamata teenusepakkuja standardiseeritud lahenduse pakkuja. Näitlikustamiseks võib olla selleks Maksekeskus ning nummerdatud

teenuspakkujad on sama näite puhul Eestis tegutsevad krediidiasutused. Kasutaja rollis on tooteid või teenuseid pakkuv e-pood, kes on teostanud liidestuse Maksekeskusega. Kasutaja ja teenusepakkuja liidestust tõlgendab lõputöö autor ühtseks. Seda toetab Maksekeskuse võimekus pakkuda üheksast Eesti krediidiasutuse nimistust seitsme asutuse teenuseid [7], [8]. Sellega suudab e-pood pakkuda enda klientidele suuri arenduskulusid tegemata suurel hulgal erinevaid makseteenuseid standardiseeritud kujul.




Joonis 1. Lihtsustatud agregeeritud ärimudel.

Standardiseerimise peab tegema teenuseid koondav ettevõte arvestades kõikide integreeritavate teenusepakkujate nõutavate andmeväljadega ja andmete üleandmise viisidega. Selle tulemusena luuakse kliendile ühtne liidestus, mis võimaldab lihtsama vaevaga kasutada valdavalt kõiki teenusepakkujaid. Pakutava teenuse keerukus võib sõltuda valdkonniti ning üldjuhul on laiapõhjalisematel tarkvarasüsteemidel agregeeritud andmeväljadega seotud komplitseeritud reeglistik.

## 1.2 Andmete valideerimine veebirakenduses

Andmeväljadega seotud reeglite kontrollimist nimetatakse valideerimiseks. Antud protsess hõlmab andmeedastusobjekti väljadega seotud kriteeriumite täitmist, et tagada rakendusse sisestatavate andmete vastavust ette antud nõuetele. See on oluline veebirakenduse tõrgeteta toimimiseks. Ebatäpsete või mittetäielike andmete käitlemine võib põhjustab programmi täitmisaegsete vigade tekkimist või äriprotsessi tõrkeid.

Veebirakendused põhinevad klient-server-mudelil, mis võimaldab teostada valideerimist mõlemal osapoolel. Kliendi poolne valideerimine parandab kasutajakogemust, pakkudes kiiret tagasisidet ning hoides ära liigse serveri poole pöördumise. Kuid kergete manipuleerimise võimaluste tõttu ei piisa ainuüksi kliendi poolsest valideerimisest [9].

Server tagab andmeedastusobjektide töötlemiseks stabiilsema ja kontrollituma keskkonna, millele puudub otseligipääs kõrvalistel isikutel. Enne andmete kasutusele võtmist võimaldab see kontrollida ja salvestada neid ettemääratud kvaliteeditasemele.

Rakenduse arendamisel on andmete valideerimine alati olnud üks põhilisi väljakutseid, mistõttu on ka saadaval arvukalt valideerimisteeke. Kõik need erinevad suuremal või väiksemal määral tingituna esialgseks kasutuseks mõeldud rakendusest, selle eripäradest ja vajadusest ning sobivusest rakenduse äriloogikale. Sobiv andmete valideerimise kirjeldamise süsteem peab olema jätkusuutlik, et vältida hilisemas arendusfaasis kulukaid muutusi.

## 1.3 Probleem

Agregeeritud ärimudelil põhineva veebirakenduse teenusepakkujate liidestus võib sisaldada olenevalt valdkonnast rohkelt andmevälju. Need andmeväljad kirjeldavad integreeritava rakenduse toimimisloogikat ning seda tuleb kajastada ka arendatava rakenduse valideerimisreeglistikus. Olemasolevaid valideerimisteeke on mitmeid, kuid puudub ülevaade nende kasutuselevõtust agregeeritud ärimudelit kasutavas Java veebirakenduses.

## 1.4 Eesmärk

Käesoleva töö eesmärk on välja töötada jätkusuutlik süsteem polümorfsete andmeedastusobjektide väljade ja väljadevaheliste sõltuvuste valideerimisreeglite kirjeldamiseks, mida saab kasutada Java veebirakenduses, mille programmikood on kirjutatud Kotlinis. Antud eesmärk on tingitud laialdasemast tarkvaratoodete liikumisest agregeeritud ärimudelile [10].

Analüüsi eesmärk on luua detailsem arusaam probleemist, kasutada andmemudeli agregeerimise tarbeks polümorfseid andmeedastusobjekte toetavat objektimise teeki ning teostada sama reeglistiku kirjeldamist erinevate valideerimisteekidega. See järel tulemeid võrrelda kasutades hierarhiate meetodit (AHP). Võrdlemiseks loodud rakendus teha avalikuks ja kõigile kättesaadavaks.

# 2 Analüüs

Analüüsi peatüki ülesehitus on loodud vastavalt Lisas 2 toodud tehniliste komponentide suurusele, alustades suurimast ning liikudes edasi väiksemale tasemele. Samal ajal on arvesse võetud protsessi täitmise järjekorda rakenduse tööseisundis, mis on välja toodud Lisas 3. Selle teostamiseks on loodud Kotlinis kirjutatud Java veebirakendus, mis teostab polümorfsete andmemudelite vastuvõttu ning mille eesmärk on kõrvutada erinevad valideerimisteegid. Rakenduse loomisel kasutatakse üldlevinud praktikaid.

## 2.1 Programmeerimiskeel Kotlin

Java rakenduste arendamiseks võib JetBrainsi toodet IntelliJ IDEA nimetada laialdaselt kasutatavaks standardtarkvaraks. Sama organisatsioon lõi 2010. aastal staatiliselt tüübitud programmeerimiskeele Kotlin, mille versioon 1.0 avaldati 2016. Kotlin on kergesti õpitav ning see on inspireeritud erinevatest programmeerimiskeelest nagu Java, C#, JavaScript, Scala ja Groovy. [11]

JetBrainsi initsiatiivil arendatava keele üheks suureks rõhuasetuseks on Java ja Kotlini omavaheline sobitamine. Nii on võimalik Javas kutsuda välja Kotlinis kirjutatud koodi ja vastupidi. See soosib Java asemel Kotlinit kasutama, kuna olemasolevat koodibaasi on võimalik vähendada sellega ligikaudu 40%. Samuti on keel tüübikindlam ning hõlpsasti saab sellega omistada nii objektorienteeritud kui ka funktsionaalse programmeerimise stiile. [11]

Google poolt soovitatud keel Android rakenduste kirjutamiseks on Kotlin, tuues välja samu argumente, mis on mainitud Kotlini ametlikus dokumentatsioonis [12]. Lisaks, leidub Kotlini dokumentatsioonis hulgaliselt firmade ja teekide edukaid Kotlinile adopteerimise lugusid, mis julgustavad selle kasutuselevõtmist [11].

Kotlini võimekus kasutada Javas kirjutatud teeke annab võimaluse kasutada Java veebirakenduse raamistike koos moodsamate kõrgkeelevõimekustega. Kuigi Kotlini püüdlus on jõuda veel rohkematele platvormidele, siis juba hetkel on Kotlin edukalt populariseerinud ennast nii Androidi rakendustes kui ka Java platvormi tagarakendustes

[13]. Antud keele populariseeritust kinnitab ka käesoleva lõputöö autori eelnev erialane töökogemus erinevates etappides tegutsevates tarkvaraarendusettevõtetes, kus uue funktsionaalsuse kirjutamine toimus täielikult Kotlinis.

## 2.2 Veebirakendus

Antud rakenduse versioonihaldussüsteemiks on valitud tasuta kättesaadav ja vabavaraline hajus versioonihaldussüsteem nimega Git. Seda kasutavad mitmed suurkorporatsioonid, sealhulgas Microsoft ja Google. [14] Samuti on olemas mitmeid veebirakendusi, mis pakuvad tasuta Giti hoidlate varundust ning üht neist on ka käesolevas lõputöös kasutatud.

Koodi varundamiseks on kasutatud platvormi GitHub, mis pakub tasuta võimalust varundada ja jagada Giti hoidlaid. Sama funktsionaalsust teatavate erisustega pakub nii Bitbucket kui ka GitLab [15]. GitHub langes valikuks peamiselt käesoleva töö autori varasemate kokkupuudete põhjal, mille tulemusel välditi ajalist kulu varasema kasutaja seadistuste rakendamisega. Samuti edestab GitHub avatud lähtekoodiga projektide vaadatavuselt teisi eelmainitud platvorme [15]. See annab parema eelduse teadmiste jagamiseks läbi koodi.

Rakenduse erinevate tarkvarateekide sõltuvuste haldamiseks ning ka tööle seadmiseks kasutatakse tarkvaratööriista Gradle. Gradle on ülesehitatud võttes arvesse selle eelkäijat Maven. Võrreldes Maveniga on Gradle paindlikum, parema jõudlusega ja kiirema *integrated development environment* (IDE) abistavate tarkvarajuppide arendusega [16].

### 2.2.1 Veebirakenduse raamistiku valik

Kotlini dokumentatsioonis on toodud välja mitmeid raamistikke tagarakenduste ehitamiseks. Nendeks on Spring Boot, Ktor, Micronaut, Quarkus, Vaadin, Cuba, Vert.x, Http4k ja Javalin. [17] Alljärgnevas tabelis (Tabel 1) on eelmainitud raamistike populaarsuse ülevaade, millest on eemaldatud tasulisi teenuseid pakkuvad Vaadin ja Cuba, olles sellega rohkem platvorm kui raamistik. Võrdluse eesmärk on leida kõige laiema kõlapinnaga raamistik, milles luua ka antud lõputöö näitekood.

Tabelis 1 on toodud välja veebirakenduste raamistike populaarsus, kasutades avaliku lähtekoodi hoidla GitHub *stars* väärtust, mis näitab kasutajate arvu, kes on märkinud

antud projekti *stars* väärilisek. Vastupidiselt on Maven Repository tulemus, kus kõige populaarsem teek omab väikseimat väärtust. Tulemusena selgub, et kõige populaarsem veebirakenduste ehitamiseks on Spring Boot raamistikkomplekt. Samuti on ka autori senine tööalane kogemus võrdluses olevatest raamistikest suurim Spring Bootiga. Tabeli põhjal saab luua järelduse, et lõputöös esitatud näitekood leiab suurima tõenäosusega kõlapinna just antud raamistikkomplektis kirjutatud veebirakenduse näol.

Tabel 1. Veebirakenduste raamistike populaarsuse võrdlus.

| | **Spring Boot** | **Ktor** | **Micronaut** | **Quarkus** | **Vert.x** | **Http4k** | **Javalin** |
|---|---|---|---|---|---|---|---|
| **Github *stars*** | 70854 [18] | 11809 [19] | 5868 [20] | 12618 [21] | 13867 [22] | 2434 [23] | 6887 [24] |
| **Maven Repo-sitory *ranking*** | 50 [25] | 1814 [26] | 673 [27] | 437 [28] | 321 [29] | 2446 [30] | 3774 [31] |

Spring Boot on komplekt erinevatest teekidest, mis toetuvad Springi baastehnoloogiale. Sellel on lai kasutajaskond ning mahukas dokumentatsioon koos näidetega. Kuigi võrreldes teiste keelte ja raamistikega jääb antud raamistik mälukasutuse ja jõudluse osas alla [32], on see siiski tänu suurele kasutajaskonnale jätkusuutlik. Samuti leidub arendusfoorumis Stackoverflow erinevaid probleemilahendusi läbi aastakümnete. Spring käib kaasas Java üldise arenguga ning on siiani populaarne valik veebirakenduse ehitamiseks. Viimaseks suureks muutuseks võib lugeda Java EE spetsifikatsioonilt üleminekut Jakarta EE spetsifikatsioonile, mis kindlustas avatud lähtekoodiga projektide ladusamat jätku. [33] Lisaks eelmainitule, toetab Spring Booti valikut ka eelnevalt teostatud veebirakenduse raamistike populaarsuse võrdlus.

### 2.2.2 Rakenduse loomine

Spring Boot rakenduse loomiseks kasutatakse Spring Initializr veebitööriista, mille veebikuva on nähtav Lisas 4. Sellega saab koostada vastavalt kasutaja valitud sõltuvustele eelseadistatud rakenduse skeletikoodi kokku pakitud failina.

Pärast skeletikood lahti pakkimist saab seada üles Giti ja initsialiseerida kaasas oleva Gradle Wrapper skripti. Wrapper paigaldab projektile vastava versiooniga Gradle

tööriista [34]. Seejärel saab paigaldada rakenduse sõltuvused, mis on kirjeldatud failis *build.gradle.kts*. Rakenduse käivitamiseks on mõeldud Gradle käsk „*bootRun*“ [35].

### 2.2.3 Veebirakenduse põhifunktsionaalsus

Antud veebirakenduse puhul on tegu prototüübiga, mille ainuke ärireeglitele kohandumine ja rakendamine toimub päringute vastuvõtmisel ja tagastamisel, edasist andmete salvestamist ja käitlemist ei toimu. Päringute töötlus toimub vastavalt määratud polümorfsele andmemudelile. Igale andmemudeli kujule peab rakenduma vastav valideerimisreeglistik ning tagastama kõik ebakõlad.

### 2.2.4 Veebirakenduse lisafunktsionaalsus

Avaliku liidesega tagarakendusel on olulised mitmed funktsionaalsused, mis parandavad kasutatavust, töökindlust ja hilisemat rakenduse töös hoidmist. Üheks jätkusuutliku rakenduse sujuvamaks edasiarendamise tagamiseks on testide kirjutamine. Sellega suudetakse luua ja kontrollida koodi funktsionaalsuse täituvust. Samuti lisab see loetavust ja defineerib tarkvaraarendaja teostatud töö sisu. Kuigi käesoleva lõputöö rakenduse näitekood on vähene ja äriliselt tarbetu, on siiski sellele kirjutatud integratsiooni testid, mis jäljendavad rakenduse käimisoleku seisu. Sellega kontrollitakse rakenduse tegelikku käitumist arendaja soovidele.

Rakendusliidese dokumenteerimine on oluline rakenduse kasutajale. Vastava dokumendi lugemisel peab tekkima aimdus, milliseid andmeid on võimalik rakendusele saata ning millised on nende päringute vastused. Antud juhul on *HyperText Transfer Protocol* (HTTP) *application programming interface* (API) dokumenteerimiseks ka vastav OpenAPI standard [36]. OpenAPI versioone ja vastava dokumentatsiooni kuvamise rakendusi on mitmeid. Käesolevas töös on kasutatud OpenAPI standardit 3.0 ning API dokumentatsiooni kuvamiseks kasutatakse Redoc eesrakendust, mille kasutajaliidese vaade on toodud Lisas 5.

## 2.3 Andmemudel

Agregeeritud ärimudeli puhul tuleb andmemudeli koostamisele eelnevalt teha kindlaks grupeerimise tunnus. Grupeerida saab mitmeti ning seda üldjuhul dikteerivad ärilised nõuded. Juhul kui ärinüanss ei luba vahendatava teenusepakkuja teadlikust kliendile

väljastada, siis on otstarbekas grupeerida andmemudel konkreetse vajaliku andmevälja kaudu.

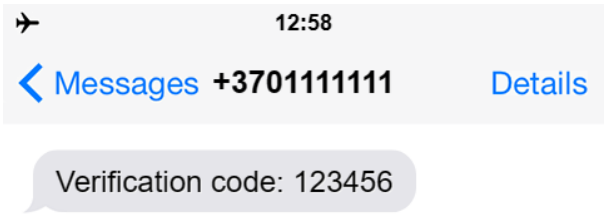
### 2.3.1 Grupeerimata andmemudel

Kui pakutav toode on SMS teenus, siis vastavalt asukoha riigile võib seda sõnumit käsitleda teenusepakkuja erinevad partnerid. Joonisel 2 on SMS saatmise andmemudel grupeeriva andmeväljata ning sellisel juhul on pakutav toode kõige lihtsakoelisem ja valideerimine peab API dokumentatsiooni genereerimisest lähtudes toimuma kõige lihtsama mudeliga ja rangeima nõuetega partneri alusel.

```
{
    "to": "+37212345679",
    "message": "Verification code: 123456"
}
```

Joonis 2. Grupeerimiseta SMS saatmise andmemudel.

Antud juhul ei ole kliendil võimalik valida lisafunktsionaalsust, mida mõne riigi telekomi ettevõtted võivad pakkuda. Seetõttu ei ole võimalik ka Joonisel 3 kuvatud sõnumil kindlaks teha, kellelt see pärineb. See võib eksitada ja häirida adressaati, kuna puudub ära tuntav viide saatja kohta.




Joonis 3. Grupeerimiseta SMS saatmise lõpptulemus.

Grupeerimata andmemudel on lihtsaim ning samas ka kõige piiravam agregeeritud andmemudeli liik. Üldjuhul on antud mudeli võimalikkus reguleeritud äritegevuses, kus on pikalt püsinud kindlad ja lihtsad standardid.

### 2.3.2 Kohaldatud grupeeringuga andmemudel

Kui SMS teenusepakkuja tahab pakkuda rohkem väljundeid, siis järgmiseks normaliseerimise sammuks võib liigitada telekomi ettevõtteid riigiti. Ärikitsendusena peab teenusepakkuja arvestama, et teenust kasutav klient ei pruugi ise olla võimeline riiki määrama, kuid samas on võimalus riigipõhiselt paremat funktsionaalsust pakkuda.

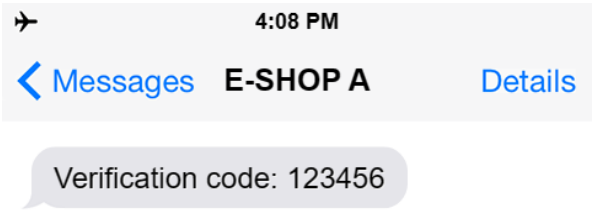
Pakkudes riigiti ainult ühe telekomi teenuseid, tekitatakse ärisaladus, mis varjab pakutava telekomi ettevõtte nime. Sellega suureneb ka risk kui antud telekomi ettevõte peaks lõpetama tegevuse konkreetses riigis. Pakkudes riigiti mitme telekomi pakkuja teenuseid, väheneb risk, kuid nii nagu grupeerimata andmemudeli puhul, tuleb valideerimisreeglid ja andmemudeli koostamisel lähtuda väikseimat funktsonaalsust pakkuva ettevõttega.

Joonisel 4 on andmemudeli grupeering lähtudes riigist. Sõnumi loomisel on arvestatud võimalusega määrata selle saatja nimi. Sama riigi telefoninumbri puhul puudub kohustus lisada sellele suunakood.

```
{
    "destinationCountry": "EE",
    "from": "E-SHOP A",
    "to": "12345679",
    "message": "Verification code: 123456"
}
```

Joonis 4. Riigipõhise grupeeringuga SMS saatmise andmemudel.

Joonisel 5 on kuvatud eelnevalt määratud nime ja tekstiga sõnum. See loob lõpptarbijale täiendava konteksti ning seetõttu saab selle teenuse eest, võrreldes tavalise sõnumiga, küsida kõrgemat hinda.




Joonis 5. Riigipõhise grupeeringuga SMS saatmise lõpptulemus.

Kohaldatud grupeeringuga andmemudel lisab kasutajale keerukust, luues võimaluse kasutada täiendava väärtusega funktsionaalsust. Antud mudelit saab kasutada ka teenusepakkuja varjamiseks, kuna vajalikud andmed on üldised.

### 2.3.3 Täpse grupeeringuga andmemudel

Kohaldatud grupeerimise näide oli teostatud kasutades võimalike vajaminevaid andmevälju, mida konkreetsel juhul oli võimalik pakkuda ühel vastava riigi telekomi ettevõttel või kuni kõikidel selle riigi telekomi ettevõtetel. Võimalik on luua veel

konkreetsem andmemudel, mis suudaks täielikult katta telekomi firma funktsionaalsust ning mille saatmise vorm on Joonisel 6.

```
{
    "serivceProvider": "TEL_COM_COMPANY_A",
    "from": "E-SHOP A",
    "to": "+37212345679",
    "message": "Verification code: 123456",
    "attachments": [
      "https://url.to/attachment.png"
    ]
  }
```

Joonis 6. Ettevõttepõhise grupeeringuga SMS saatmise andmemudel.

Klient saab valida sobiva telekomi ettevõtte, kelle vahendusel sõnum lõpptarbijale edastatakse. Rakenduses kohaldatakse igale ettevõttele omast valideerimisreeglistikku ning võimaldatakse kasutada antud teenusepakkuja maksimaalset funktsionaalsust. Konkreetsel juhul on kasutatud piltide lisamist ning mille lõpptulem on Joonisel 7.


Joonis 7. Ettevõttepõhise grupeeringuga SMS saatmise lõpptulemus.

Ettevõttepõhise grupeeringu osas tuleb arvestada, et kohati võib sellise mudeliga kaasneda kliendi ja lõppteenusepakkuja vahel eraldi kokkulepped, mis pikendavad teenuse kasutuselevõtmise protsessi, kuid samas vähendab vastutus integreerimise teenust pakkuval firmal.

### 2.3.4 Polümorfse andmemudeli koostamine

Polümorfism tähendab mitmekujulisust. Antud lõputöö käigus loodavas täpse grupeeringuga andmemudelis tähistab seda üks andmeväli. See võimaldab

integreerimisteenust pakutaval veebirakendusel vähendada enda poolset äririski ning lubada klientidele maksimaalset pakutavat funktsionaalsuse olemasolu. Samuti ei toimu sellisel juhul agregeerimine mitme välja abil, mis omakorda lisaks eraldi keerukust.

Andmemudeli koostamise ärikontekstiks on modelliagentuuride vahendust pakkuva ettevõtte rakenduse arendus. Antud teenusepakkuja juures saab kandideerida erinevate agentuuride juurde täites selleks vastav *Curriculum Vitae* stiilis vorm. Samuti on võimalik täita üldine vorm, mis on toodud Lisas 6. Kogu vorm lähtub inimest iseloomustavatest andmetest nagu nimi, sünniaasta, kaal, pikkus ning kontaktandmed.

## 2.4 Jätkusuutlik tarkvarateek

Tarkvarateek on korduvkasutatavate koodikomponentide ja funktsioonide kogum ning neid saab kasutada erinevates rakendustes või teistes teekides. Tarkvarateegi arukal kasutamisel säästab arendaja aega ja vaeva probleemi lahendamiseks.

Avatud lähtekoodiga tarkvarateekide kiire kasv on kaasa toonud sarnasele probleemile erinevate lahenduste kasutamise. Tekkinud on mitmeid eriotstarbelisi lahendusi ning nii palju kui on inimesi on ka erinevaid arvamusi ja teguviise. Seetõttu on ka avalikult kättesaadaval mitmeti lahenduvatele probleemidele hulgaliselt teeke. Võttes arvesse eelnevat, ei tohiks ühtegi tarkvarateeki kasutusele võtta enne selle esmast analüüsimist. Esmasel vaatlusel on võimalik analüüsida konkreetse teegi jätkusuutlikust konkreetse eesmärgi saavutamiseks. Jätkusuutlikkuse staatust ei ole võimalik ühiselt sõnastada, kuna seda kasutatava rakenduse eesmärgid võivad olla erinevad. Siiski saab lähtuda mõnest printsiibist, mis on omane igale teegile.

Tarkvarateegi käekäiku mõjutab oluliselt sellele kohalduv litsents. Tarkvara saab üldjuhul liigitada omandvaraks või vaba ja/või avatud lähtekoodiga tarkvaraks, millest viimane on rohkem lubavate litsentsidega, kuid mitte tingimata tasuta. Kohati võib rakenduda ka vastavalt litsentsile ärilisel eesmärgil kasutamise keeld. [37] Tarkvara tootvad ettevõtted eelistavad kasutada nõrga õiguste edasikandumise klausliga teeke, kuna sellisel juhul ei pea muretsema litsentsitasude ning autoriõiguste rikkumiste osas. Sageli kasutatakse Apache ja MIT litsentse, mis kuuluvad nõrga õiguste edasikandumise litsentside hulka ning neid on lubatud vabalt kasutada ärilistel eesmärkidel.

Suur kasutajaskond võib luua aktiivse ja toetava kogukonna tarkvarateegi ümber, mis omakorda soodustab teegi jätkuvat arengut ja kasutamist. Aktiivne jututuba või sarnane suhtluskanal lihtsustab ja kiirendab probleemide lahendamist, kuna on võimalik esitada kogenud kasutajatele küsimusi ning aidata seeläbi uutel kasutajatel kohaneda. Samuti leiavad kinnituspinda uued funktsionaalsused, mis vastaksid kasutajate vajadustele.

Head dokumentatsiooni võib pidada oluliseks esmaseks eelduseks kasutajaskonna kasvatamiseks ja teegi edukaks kasutamiseks. Dokumentatsioon sisaldab üldjuhul erinevaid õpetusi, korduma kippuvad küsimused sektsiooni, teegi lühikirjeldust ja eesmärki ning näiteid parimatest kasutustavadest. Kvaliteetne dokumentatsioon mitte ainult ei aita kasutajatel edukalt teeki kasutada, vaid vähendab ka teegi autorite ja arendajate ajakulu vastamaks kasutajate probleemidele ja küsimustele.

Tarkvarateegi valimisel tuleb arvestada kiirusega, kuid mitte liialt keskenduda varasele koodi optimeerimisele. Olulisem on teegi kiire omaksvõtmine ja selle kasutamise kiirus. Seda väidet toetavad ka eelnevad lõigud dokumentatsioonist ja aktiivsest kasutajaskonnast. Seetõttu peab teegi kasutajaliides olema piisvalt lihtne, et lahendada keerukaid probleeme.

Tarkvarateegi jätkusuutlikust mõjutavad mitmed tegurid, ning kuigi kindlad omadused määratakse sõltuvalt konkreetse teegi kasutusest, on siiski võimalik lähtuda kõikidele teekide ühistest omadustest. Aktiivne kommuun aitab teegi funktsionaalsusel kiiremini valmida, kuid samas võib piiravamaks saada hoopis esialgne litsentsivalik. Samuti on vajalik, et teegil oleks olemas dokumentatsioon, mis aitaks kasutajatel mõista, kuidas teeki kasutada ja vähendada sellega ajakulu probleemi lahendamisel. Tarkvarateegid on tehtud inimestelt inimestele ning on oluline, et teegi kasutajaliides oleks lihtne ja võimaldaks arendajal probleemi lahenduseni hõlpsamini jõuda.

## 2.5 Objektimise ja jadastuse teegid

Veebirakenduse andmeedastus väliste osapooltega võib kasutada erinevaid protokolle ja formaate. Populaarne veebirakendustes kasutatav andmeedastusformaat on *JavaScript Object Notation* (JSON). JSON on andmevahetuse vorming, mida on nii inimestel kui ka masinatel kerge lugeda ja kirjutada [38].

Objektimine on protsess, mis muudab baidi-või sõnejada andmeobjektiks [39]. Objektimise käigus on võimalik JSON formaadis sõnejada muuta klassina kirjeldatud andmeobjektiks. Spring Boot pakub kolmele objektimise ja jadastuse teegile automaatseid konfiguratsioone. Nendeks teekideks on Gson, Jackson ja Yasson. [35]. Tabelis 2 on eelmainitud teekide lühivõrdlus.

Tabel 2. Objektimise ja jadastuse teekide võrdlus.

| | **Gson** | **Jackson** | **Yasson** |
|---|---|---|---|
| **Litsents** | Apache-2.0 [40] | Apache-2.0 [41] | Eclipse Public License 2.0 [42] |
| **Viimane versioon** | 2.10.1 [40] | 2.16.0 [41] | 3.0.2 [42] |
| **Versiooni avaldamise kuupäev** | 6 jaanuar 2023 [40] | 16 november 2023 [41] | 13 september 2022 [42] |
| **Ametlik staatus** | Hooldusrežiim [40] | Aktiivne [41] | Küps [43] |
| **Haldaja** | Google [40] | FasterXML [41] | Eclipse Foundation [42] |
| **Toetab polümorfset objektimist** | Jah [44] | Jah [45] | Jah [46] |
| **Autori kogemus** | Vähene | Kõige rohkem | Puudub |

Teekide esmasel uurimisel sai tõdemuseks, et kuigi tehniliselt on kõik võrdluses olevad teegid sobilikud antud lõputöö eesmärgiga, omades selleks polümorfset objektimise funktsionaalsust, siiski ei toeta seda teised tegurid. Gson teeki ei saa liigitada jätkusuutlikuks objektimise teegiks, kuna edasiarendusi ei toimu ning projekt on ametlikult hooldusrežiimis. Sarnane seis on ka JSON-B standardit järgiva Yasson teegiga, mille ametlik staatus on märgitud kui kõlbulik kasutamiseks, kuid mille edasiarendused on selgelt jäänud liiga harvaks, et pidada seda aktiivselt hooldatud teegiks. Võrdluse põhjal osutus sobilikuks Jackson-nimeline objektimise ja jadastuse teek. Antud teegil on olemas jätkusuutliku tarkvarateegi tunnused ning on ka Spring Booti vaikimisi valik.

Eelmainitud tagarakenduste ehitamiseks mõeldud raamistike võrdlustabeli nimistus on Jacksoni teek samuti kasutamise esirinnas ning suuremahuliste raamistike esmane valik. Vert.x ja Javalin erinevad teistest raamistikest enda nägemusega valideerimisreeglistiku kasutamisest. Seda on tehtud enda ehitatud teegi [47] või eraldiseisva JSON Schema

tööriista näol [48], kuid siiski annab nendel raamistikel kasutada Jackson teeki. Http4k puhul on võrreldes teistega toetatud rohkem erinevaid objektimis- ja jadastusteeke. Samuti on sealne tendents rohkem suunitletud Kotlinis kirjutatud teekidele, pakkudes tuge JetBrainsi enda Kotlinx.Serialization teegile ning ka vähem tuntumatele Moshi, Klaxon ja Kondorile [49].

Kuigi objektimise ja jadastuse teegi võrdlus oli esmalt loodud põhinedes ainult Spring Bootil, on Jackson saavutanud populaarsuse ka teistes raamistikes. Samuti omab see *de-facto* staatust objektimise ja jadastuse teekide seas [50].

### 2.5.1 Rakenduse seadistus

Rakendusega suheldes on vajalik saada piisvalt infot, miks päringut ei aktsepteeritud. Seetõttu on vajalik rakenduses päringute vastuvõtu täiendav seadistamine. Saates päringu vales või vigases formaadis, tekib rakenduses erind. Erindeid saab lahendada, kasutades annotatsiooni *RestControllerAdvice*. See võimaldab päringule vastates tagada, et rakendus kuvab vastuse vastavalt defineeritud erindi tekkele.

Käesolevas rakenduses on kirjutatud loogika Jackson spetsiifiliste erinditega tegelemiseks, mis juhtuvad HTTP päringu loetamatuse korral. Antud loogika lõpptulem annab kasutajale vihje valesti läinud päringu põhjusest, mille lahendamiseks on otstarbekas kasutada genereeritud API dokumentatsiooni. Erindi tegelemise loogika on kirjeldatud Lisas 7.

## 2.6 Valideerimisteegid

Kotlinis rakendatavaid andmete valideerimiseks mõeldud tarkvarateeke on mitmeid. Nende seast sobiva teegi valimine on rakenduse arendamisel oluline otsus, mis võib oluliselt mõjutada projekti hilisemat püsimist ja arendamise kiirust [51]. Valik peaks toetuma valideerimisteegi võimalikule tehnilisele võimekusele, mis on seatud konkreetses projektis [52]. Teisisõnu peab valideerimisteek olema projekti raames jätkusuutlik.

### 2.6.1 Valideerimisteegi nõuded

Valideerimisteeki valides võiks eelistada neid tarkvarateeke, millel on olemas kõik esmased jätkusuutlikkusega seotud tunnused nagu dokumentatsioon koos näidistega ning

oleks kasutatav Gradle tarkvara ülesseadmise tööriistaga. Samuti võiks antud teeki ümbritseda aktiivne kommuun, mis suurendaks püsivust.

Andmeedastusobjektide valideerimisel peab olema tagatud koodi hilisem loetavus. Ideaalne valideerimisteek peaks koondama kogu andmemudeliga seotud reeglistiku ühte kohta. Sellega suureneb konkreetsus, lihtsus ja selgus kaasates alamobjektidega seotud reeglistiku kirjeldust.

Valideerimisteegi lihtsust, selgust ja konkreetsust suurendab veelgi annotatsioonivaba lähenemine. Annotatsioonide kasutamine küll vähendab korduvkoodi kirjutamist, kuid samas viib arendaja sammu võrra kaugemale selle tegelikust toimimisloogikast. Samuti võib annotatsioon olla kasutuses alles rakenduse käimisaegses olekus, mis tõttu ei ole võimalik vigu tuvastada programmi kompileerides.

API väljastamisel peab tagama selle järjekindluse, mistõttu ei tohiks olemasolevat versiooni muuta. Uuendused peaksid olema saadaval uue API versioonina, mida saavad kasutada nii uued kui ka olemasolevad kasutajad. Samuti peaks valideerimisreeglite kirjeldus olema järjekindel ja muutumatu oma reeglites. See tähendab, et valideerimisteek peab olema staatiline ja annaks sisendil alati sama vastuse. Samuti on kasutajale mugavam kui kõik reeglirikkumised oleks kuvatavad ühe vastena.

### 2.6.2 Valideerimisreeglid

Agregeeritud andmemudelil võib olla mitmeid erikujusid. Lõputöö käigus valminud rakendus kontrollib andmemudelit vastavalt agregeerimise tunnusele ning antud juhul on neid mudeleid kokku kolm. Need mudelid erinevad nii nõutud andmeväljade kui ka nendele seatud reeglite osas. Esimese andmemudeli kuju nõuab lihtsamate väljade ja väljadevaheliste sõltuvuste valideerimist, järgmistega lisandub keerukust andmekollektsioonide ja erinevate objektide kontrollimisega.

### 2.6.3 Valim

Valideerimisteekide valimiks on mugavusvalim. Mugavusvalim toetub kättesaadavusele, mis on ühtlasi ka üheks teegi jätkusuutlikkuse tunnuseks. Valim koostati GitHubi teemalehtede põhjal, mis keskendusid Javas ja Kotlinis kirjutatud valideerimisteekidele [53]. Esmasel vaatlusel elimineeriti teegid, mis keskendusid Androidi rakenduse vaadete valideerimisele. Samuti kõrvaldati nimistust need, millel puudus esmane paketihalduse

poolt pakutav versioon, dokumentatsioon või mis ei toeta mitme valideerimisreegli rikkumisteadete tagastust ning on suurel hulgal koodibaasi suurendav. Nimistus esineb kokku 9 erinevat teeki, millest üks otseselt ette seatud nõuetele ei vasta, kuid kuna tegu on *de-facto* valideerimisteegiga Java veebirakendustes [54], täidab see võrdluses nii kaitsva staatuse kui ka metoodika kinnitavat rolli. Teekide nimistusse kuuluvad Thing, Akkurate, Java Fluent Validator, Valiktor, Konform, Kalidation, YAVI, Hibernate Validator, Validoctor.

Iga valideerimisteegi kohta on alljärgnevates alapeatükkides toodud välja teegi lühitutvustus, kasutatavate baastehnoloogiate kirjeldus, hinnang dokumentatsioonile, uue reegli kasutamise võimalused ning erinevad lisad. Samuti on teeki rakendatud vastavalt eelmainitud valideerimisreeglite astmetele kasutades polümorfset andmemudelit. Antud valideerimisreeglite esile toomine igas alapeatükis aitab võrrelda samade reeglite defineerimist erinevate teekidega.

### 2.6.4 Hibernate Validator

Hibernate Validator on Apache-2.0 litsentsil baseeruv valideerimisteek, mis võimaldab arendajatel määratleda ja rakendada Java objektidele valideerimisreegleid tuginedes Jakarta Bean Validation spetsifikatsioonile. Reeglite kirjeldamine käib annotatsioonidega ning antud teegiga tuleb kaasa mitmeid üldlevinud reegleid, mida andmeväljadele rakendada. [55] Tagatud on sujuv ühildumine Springiga, mis tõttu on tegu populaarse valikuga paljudes veebirakendusi loovates ettevõtetes [56] ning peetakse *de-facto* valideerimisteegiks Java veebirakendustes [54].

Antud valideerimisteegi kasutusliidese baastehnoloogia pärineb 2004. aastast kui avalikustati Java 1.5 versiooniga *metadata* ehk annotatsioonid. Annotatsioonide eesmärk on kirjeldada korduvkasutatav kood ning läbi selle elimineerida mitmes kohas koodi deklareerimist [57]. Annoteerida saab nii klassi, meetodit kui ka defineeritava klassi väljasid. Java põhiannotatsioonide hulka kuulub nii *Override*, mis kirjutab üle klassi meetodi kui ka funktsionaalse programeerimise tarbeks kasutatav *FuntionalInterface*. Vaatamata vastakatele arvamustele on annotatsioonid tekitanud rakenduse kirjutamise viisi, mis suurendab abstraktsiooni ja on deklareerivam, kuid samas vähendab selgust ning kontrolli. [58]

Hibernate Validator kõikide reeglite kohta on toodud näiteid mahukas ja põhjalikus dokumentatsioonis. Lisaks näidetele, selgitab dokumentatsioon probleemikäsitlust, lahendust ning tarkvaraprojekti ülesseadmist. [55] Samuti on võimalik andmete valideerimisreegleid kirjeldada mitmel eriviisil. Kõik esmavajalike kergemate tüüpide reeglite annotatsioonid on olemas ning saab kasutada ka vabamate reeglite kirjeldamise annotatsioone, kuid annotatsiooni rakendumise ja defineerimise eripärale puudub staatiline kontroll programmi kompileerides. Tingituna teegi kohmakusest on olemas ka mitmeid teisi nii ametlikke kui ka kommuunipõhiseid lisateeke, mis sisaldavad hulgaliselt keerulisemate objektitüüpide kontrolle ja ka võimaluse rakenduse kompileerides leida vigu valele tüübile rakendatud reegli kasutamisest [55] [59].

Uue reegli kirjeldamiseks on vajalik luua uus või kasutada spetsifikatsioonile omaseid *AssertTrue* ja *AssertFalse* meetodi annotatsioone. Hibernate ametlik dokumentatsioon soovitab luua uue annotatsiooni. Sedasi on tagatud sujuvam ühilduvus kuvada sõnumeid erinevates keeltes ning olemasolevate väärtuste kuvamist sõnumis. Annotatsiooni limiteeritud tüübikontroll suureneb riski programmiaegse vea tekkeks, mida võibki pidada ka antud teegi suurimaks varjuküljeks. Lisaks on annotatsiooni loomine mahukas ning see eeldab arendajalt rohkem tähelepanu ja täiendava korduvkoodi kirjutamist.

Tänu Hibernate Validator laialdasele kasutuskonnale ning põhjalikule dokumentatsioonile on suur tõenäosus, et esmane lahendus leitakse kiiresti ja hõlpsalt. Samuti tuleb kasuks ka Hibernate Validatori pikaajaline stabiilsus, mistõttu on see paljude arendajate esimene valik andmete valideerimiseks.

Joonisel 8 on teostatud andmemudeli valideerimine kasutades Hibernate Validatorit. Vormil on rakendatud nii väljade kui ka väljadevaheliste sõltuvuste valideerimisreegleid. Väljadevaheliste sõltuvuste valideerimisreeglid tuleb määrata klassile. Antud andmemudeli tarbeks on loodud kaks eraldiseisvat annotatsiooni, millest üks kontrollib emaili aadressi või telefoninumbri välja täituvust ning teine *LocalDate* ajatüübi võrdlust. Jooniselt puudub annotatsioon Valid, mis tähistab märgendatud osa valideerimist. Kui andmemudelil on mitmeid kaasuvaid objektide ahelaid, halveneb koodi jälgitavus ja selgus, sest puudub ülevaade tervele ahelale rakendatavatest reeglitest.

```
@EmailOrPhoneIsPresent
data class HibernateCompanyAAgencyForm(
    override val agency: Agency = Agency.COMPANY_A,
    @field:Size(min = 1, max = 128)
    override val firstName: String,
    @field:Size(min = 1, max = 128)
    override val lastName: String,
    @field:CheckDateIsAfterOrEquals(year = 2000, month = 1, day
= 1)
    override val birthDate: LocalDate,
    @field:Size(min = 5, max = 35)
    override val phoneNumber: String?,
    @field:Email
    @field:Size(min = 1, max = 128)
    override val email: String?,
) : HibernateAgencyForm()
```

Joonis 8. Hibernate Company A vormi valideerimise näide.

Hibernate Validator on *de-facto* valideerimisteek Java veebirakendustes. Antud teegi populaarsus ning staatus võib olla tingitud vanusest või Java arendajate kasutusharjumusest kasutada annotatsioone veidikene laiemalt kui ainult deklareerimiseks. Kõikidest võrdluses olevatest teekidest eristab antud teeki annotatsioonide laialdane kasutus ning ka OpenAPI dokumentatsiooni automaatse lisamise võimalus. Antud teeki on lihtne ja mugav kasutada kui on kasutusel lihtsamad valideerimisreeglid ning mille andmemudel ei ole keeruline.

### 2.6.5 Java Fluent Validator

Java Fluent Validator on inspireeritud .Net raamistikust pärit FluentValidatorist, mis väljendub valideerimisreeglite kirjeldamises inimese kõnelemisele sarnases stiilis. Sedasi üritatakse luua kirjeldamisviis, mida on kerge lugeda ja mõista. Samuti tuuakse teegi tutvustuses välja probleem, kus samad või sarnased valideerimisreeglid on projektipõhiselt eraldatud ning keeruline on pidada kinni *don't repeat yourself* (DRY) reeglist. [60]

Antud teek on ülesehitatud kasutades Java Generic meetodeid ning klasse, mis sarnaselt annotatsioonidega tuli Java maailma versiooniga 1.5 ning lubab kompileerimiskindla koodi loomist teatud määramatusega tüüpi kirjeldamata. Teiseks on teostatud hulgaliselt liidestatuse valmidust, mis lisab kasutajale paindlikust. Andmete lugemisel kollektsioonist hoitakse selle järjekorraväärtust lõimedele avatuna. Sellega võib tekkida *race condition*. Antud teek on võrdluses olevatest teekidest üks kolmest, mis kasutab ainult Javat.

Java Fluent dokumentatsioon sisaldab rohkelt valideerimisreeglite kirjeldusi. Näitena tuuakse lihtsamate reeglite rakendamist, kuid puudub konkreetne näide mitme välja valideerimisest. Samuti on jäänud antud edasiarendused harvaks, mis väljendub viimases teadaolevast teegi uuendusest, mis pärineb 2021. aasta lõpust. Vaatamata GitHub *stars* arvukusele, ei ole suudetud hoida teeki aktiivses arenduses. Sellest tulenevalt võib kahelda teegi elujõulisuses ja hilisemas toetuses.

Iga andmemudeli tarbeks tuleb luua uus objekt, mis pärineb *AbstractValidator* klassist. Sellega määratakse ära valideeritava objekti tüüp. Reeglite kirjeldamine toimub üle kirjutatavas meetodis. Meetodis kasutatakse kergesti loetava süntaksiga reeglite kirjeldamist, kus iga reegel algab kontrollitava välja või objekti defineerimisega. Igal uuel real järgneb uus reegel või täpsustus kontrollsõnumi edastamisel.

Uue reegli kirjeldamine on mahukas, kuna eeldab iga reegli täismanuaalset väljakirjutust ning tekib korduva välja defineerimine, mis on suurte andmemudelite puhul tülikas ja väsitav. Samuti on keerukam kogu andmemudeli kontroll, kuna iga uue objekti tarbeks peab kasutama eraldi klassi, mis on oma protsessi poolest sarnane Hibernate Validator tegumoele. Dokumentatsioonis on olemas ka kollektsioonides olevate objektide valideerimine, kuid ei suudeta kuvada nii andmemudeli välja kui ka kollektsioonides

paiknemist. Probleem leiab kinnitust ka teegi testidest, kus antud olukordasid ei ole käsitletud.

```
class JavaFluentCompanyAValidator :
FluentValidator<JavaFluentCompanyAAgencyForm>() {

    override fun rules() {
        setPropertyOnContext("COMPANY_A")

        fieldRuleOf(
            JavaFluentCompanyAAgencyForm::firstName,
            stringSizeBetween(1, 128),
            "size must be between 1 and 128"
        )
        fieldRuleOf(
            JavaFluentCompanyAAgencyForm::lastName,
            stringSizeBetween(1, 128),
            "size must be between 1 and 128"
        )
        fieldRuleOf(
            JavaFluentCompanyAAgencyForm::birthDate,
            localDateAfterOrEqual(LocalDate.ofYearDay(2000, 1)),
            "Date must be after or equals to 2000-1-1"
        )
        fieldRuleOf(
            field = JavaFluentCompanyAAgencyForm::phoneNumber,
            predicate = stringSizeBetween(5, 35),
            errorMessage = "size must be between 5 and 35"
        )
         fieldRuleOf(
            field = JavaFluentCompanyAAgencyForm::email,
            predicate = stringSizeBetween(1, 128),
            errorMessage = "size must be between 1 and 128"
        )
        entityRuleOf(
            fieldName =
JavaFluentCompanyAAgencyForm::phoneNumber.name,
            predicate = {!it.phoneNumber.isNullOrBlank() ||
!it.email.isNullOrBlank()},
            message = "phoneNumber or email must be present"
        )
    }
}
```

Joonis 9. Java Fluent Validator Company A vormi valideerimise täiendatud näide.

Lõputöö autor on täiendanud antud teegi kitsaskohti, lisades valideeritava välja reegli rikkumise korral selle asukoha andmemudelis. Samuti on autor piiranud andmemudeli väljade automaatset nimetamist vastavalt klassi määratletud väljade nimedele, kasutades

selleks Kotlini reflektsiooni teeki. Lisaks avastati kollektsiooni elemendi asukoha väärtuse väära positsiooni kuvamine, mis on nähtav lõputöö näitekoodis antud teegi testis.

Java Fluent Validator puhul on tegu lihtsakoelise ja järele aitamist vajava valideerimisteegiga. Kuigi esmavaatluse põhjal võiks lugeda antud teeki kõlbulikuks kasutamiseks, siis antud lõputöö eesmärkide ja nõuetega antud teek siiski ei vasta. Lisaks on teegis arhitektuuriliselt kriitilisi kõrvalekaldeid, mis muudavad teegi kasutuskõlbmatuks, sest rakendamisel puudub kindlustunne.

### 2.6.6 Konform

Konform on Kotlin Multiplatformi toetav valideerimisteek, mis kasutab reeglite kirjeldamist tüübikindlat valdkonnaspetsiifilist keelt [61]. Kotlin Multiplatformi eesmärk seisneb platvormide üleses korduvkasutatava koodi konsolideerimises [62], kuid see ei too kaasa otseseid märkimisväärseid eeliseid tagarakenduse loomisel. See vastu avaneb valdkonnaspetsiifilise keele abil võimalus koostada intuitiivselt mõistetavaid ja tüübikindlaid reegleid, mis sobituvad sujuvalt Kotlini programmeerimisparadigmaga. Samuti on oluline märkida, et antud teek ei kasuta eraldiseisvaid väliseid teeke, mistõttu on ka antud teegi edasiarendused stabiilsemad ja kooskõlas keele enda suunitlusega.

Konform on ülesehitatud Kotlin Multiplatformil. See võib lisada kitsendusi, mis on seotud platvormi erisuste ühildamiseks mõeldud koodi deklareerimisega, mis sisuliselt tähendab arendajale ikkagi platvormi spetsiifiliste koodi juppide arendamist. Käesolev teek lahendab üldlevinud probleemi ning selle tarbeks ei ole vaja kasutada eriotstarbelist platvormipõhist lähenemist. Lisaks, kasutatakse Kotlini *Domain-Specific Language* (DSL) võimalusi loomaks tüübikindlaid valideerimisreegleid. Antud juhul saadakse reflektsiooni kaudu andmemudeli välja nimi ja väärtus ning kasutatakse edasisi funktsioone kontrolli teostuseks.

Konformi dokumentatsioon sisaldab näiteid lihtsamate reeglite kirjeldamiseks. Keerukamate ja vähem levinud reeglite kirjeldamise näited puuduvad. Sarnaselt paljudele valideerimisteekidele, on ka Konformil aktiivne arendus seiskunud ning otsitakse osalisi, kes aitaks teeki edasi arendada [63].

Vaatamata aktiivse arenduse puudusele, toimib antud valideerimisteek siiski eeskujulikul tasemel. Uue reegli kirjeldamiseks saab kasutada DSL võimalusi, andes funktsioonile

sisendiks valideeritav väärtus ning väljundiks *Boolean* tüüpi objekt. Samuti saab määrata sõnumisisu, kui defineeritud valideerimisreeglit rikutakse, kuid sõnumisisule ei saa anda kaasa valideeritavat väärtust ning ei ole võimalik eraldi defineerida valideerimisreeglile kehtivat asukoha märget. Käesoleva lõputöö autor on nii sõne kui ka kollektsiooni minimaalse ja maksimaalse pikkuse valideerimise ühildanud üheks funktsiooniks, mis vähendab koodis kasutatavate ridade arvu.

```
val validate = Validation<KonformCompanyAAgencyForm> {
            KonformCompanyAAgencyForm::agency {
enum(Agency.COMPANY_A) }
            KonformCompanyAAgencyForm::firstName { length(1,
128) }
            KonformCompanyAAgencyForm::lastName { length(1, 128)
}
            KonformCompanyAAgencyForm::birthDate {
                val reqDate = LocalDate.ofYearDay(2000, 1)
                addConstraint("Date must be after or equals to
${reqDate.year}-${reqDate.monthValue}-${reqDate.dayOfMonth}") {
                    checkLocalDateIsAfterOrEqualsTo(it, reqDate)
                }
            }
            KonformCompanyAAgencyForm::phoneNumber ifPresent {
length(5, 35) }
            KonformCompanyAAgencyForm::email ifPresent {
length(1, 128) }

            addConstraint("phoneNumber or email must be
present") { person ->
                (!person.phoneNumber.isNullOrBlank() ||
!person.email.isNullOrBlank())
            }
        }
```

Joonis 10. Konform Company A vormi valideerimise täiendatud näide.

Konform on Kotlinis kirjutatud valideerimisteek, mis kasutab suurel määral Kotlinile omast funktsionaalsust. Teegiga on suudetud katta suurem osa kasutajate vajadusi, kuid edasiarendusi ei toimu. Siin võib joonistuda välja uute teekide murekoht, kus ühalt suudetakse nädalaga luua baasvajadusi kattev teek, kuid edasine motivatsioon puudub.

### 2.6.7 Yavi

Yavi on Java programmeerimiskeeles loodud valideerimisteek, mis tugineb valideerimisreeglite konstrueerimisel funktsionaalse programmeerimise paradigmal. Antud teeki saab kasutada reflektsiooni, annotatsiooni ning väliste sõltuvuste vabalt. [64] Yavi sarnaneb Java Fluent Validator teegile, mis baseerub geneeriliselt ülesehitatud

funktsioonidele. Yavi kasutab reeglite defineerimiseks *fluent* stiili ning lõppobjekti loomeks ehitamise meetodit.

Yavi dokumentatsiooni ülesehitus ning näited on inspireeritud Hibernate Validator dokumentatsioonist. Samuti on lisatud õpetus ühildumaks Spring raamistikuga. Näiteid jagub nii lihtsate väljade kui ka väljadevaheliste sõltuvuste valideerimiseks. [64] Teegi arendused toimuvad konstantselt selle algataja poolt ning GitHubi vahendusel tõstatatud lisafunktsionaalsuse vajalikkust kontrollitakse pingsalt.

Yavi valideerimisreeglite kirjeldamiseks luuakse ehitaja tüüpi objekt, millele saab lisada erinevaid kitsendusi. Kitsendused saab luua ise või kasutada olemasolevaid eeldefineeritud piiranguid üldlevinud andmetüüpide kohta. Lisatud on ka tugi kasutamaks Kotlinit. Selle tarbeks saab kasutada valdkonnaspetsiifilist keelt, mis muudab kirjutatava koodi lihtsamini loetavaks.

```
val yaviCompanyAAgencyFormValidator =
ValidatorBuilder.of<YaviCompanyAAgencyForm>()
    .constraint(YaviCompanyAAgencyForm::firstName, "firstName")
{ c -> c.length(1, 128) }
    .constraint(YaviCompanyAAgencyForm::lastName, "lastName") {
c -> c.length(1, 128)}
    .constraint(YaviCompanyAAgencyForm::phoneNumber,
"phoneNumber") { c -> c.length(5, 35) }
    .constraint(YaviCompanyAAgencyForm::email, "email") { c ->
c.length(1, 128) }
    .constraint(YaviCompanyAAgencyForm::birthDate, "birthDate")
{ c -> c.afterOrEqual { LocalDate.ofYearDay(2000,1) }}
    .constraintOnTarget(
        { form -> !form.phoneNumber.isNullOrEmpty() &&
!form.email.isNullOrEmpty() },
        "phoneNumber",
        ViolationMessage.of(
            "",
            "phoneNumber or email must be present"
        )
    ).build()
```

Joonis 11. Yavi Company A vormi valideerimise täiendatud näide.

Kuigi Yavil on olemas kasutajasõbralik ja sisukas dokumentatsioon, on siiski teegi kasutusele võtt olnud käesoleva lõputöö autori kogemusel võrreldavatest teekidest kõige tülikam, kuna Yavi Kotlin DSL kasutamisel ei ole võimalik defineerida väljadevahelisi sõltuvusreegleid. Samuti puudub *enum* tüüpi objektide valideerimine. See vastu on antud teegil võimalik defineerida reeglistik ühe suure objektina või eraldiseisvate tükkidena, mis lisab paindlikust.

### 2.6.8 Valiktor

Valiktor on Kotlini DSL kasutav tüübikindel valideerimisteek, mis resideerub GitHub keskkonnas Apache-2.0 litsentsiga. Antud teegi esimene versioon väljastati 2019. aasta jaanuaris. Kuigi teegi nimi võiks justkui viidata Ktor [65] raamistikule, on pigem keskendatud Springiga ühildumisega. [66]

Antud teeki kasutab Kotlini reflektsiooni teeki tüübikindla *fluent* DSL loomiseks ning Java *ResourceBundle* tõlgete haldamiseks. Tõlkeid ning ka erinevaid reegleid on võimalik ise luua, muuta ja täiendada. Kõike seda katab ka detailne dokumentatsioon, mis on loogilise ning selge ülesehitusega. Samuti on hulgaliselt lisateeke, mis lihtsustavad erinevate andmetüüpidega tegelemist. Reflektsiooni on kasutatud minimaalselt ning teegi koodibaas on kergesti loetav, kuna kirjeldused on kompaktsed.

Viimane versioon 0.12.0 väljastati 2020. aasta juunis ning edasisi arendusi ei ole toimunud. See seab kahtluse alla teegi jätkusuutlikkuse, kuna ükski esialgsetest autoritest ei ole GitHub keskkonnas inimestele vastanud. Samuti on keeruline kui mitte võimatu hinnata, kas valideerimisteek on loojate poolt mõeldud kasutamiseks, kuna kõik teegid ei pruugi järgida Semantic Versioning spetsifikatsiooni [67]. Seda järgides on teek alles esialgses arendusfaasis ning antud teeki ei tohiks pidada stabiilseks. Spetsifikatsiooni järgi peaks 1.0.0 versioon defineerima teegi avaliku liidese ning järgnevad numbrikombinatsioonid on sellest sõltuvad [67].

Uue reegli kirjeldamiseks on dokumentatsioonis vastav juhend. Seda järgides on võimalik luua tõlkevalmidusega valideerimisreegel. Enne reegli kasutamist tuleks kontrollida sõnumi ja sõnumis sisalduvate asenduste korrektsuses, sest sõnumit kuvatakse alles rakenduse käimisaegses olekus. Kui erinevates keeltes sõnumi kuvamine ei ole oluline, siis on võimalik eraldiseisvalt lisada funktsioonid, mis suudaksid valideerimisreeglite kirjeldamise juures defineerida veasõnumit ning andmevälja nime.

```
validate(this) { form ->
            validate(ValiktorCompanyAAgencyForm::agency).isValid
{ it == Agency.COMPANY_A }

validate(ValiktorCompanyAAgencyForm::firstName).hasSize(min = 1,
max = 128)

validate(ValiktorCompanyAAgencyForm::lastName).hasSize(min = 1,
max = 128)

            val reqDate = LocalDate.ofYearDay(2000, 1)

validate(ValiktorCompanyAAgencyForm::birthDate).withMessage("Dat
e must be after or equals to ${reqDate.year}-
${reqDate.monthValue}-${reqDate.dayOfMonth}") {
                checkLocalDateIsAfterOrEqualsTo(form.birthDate,
reqDate)
            }


validate(ValiktorCompanyAAgencyForm::phoneNumber).hasSize(min =
5, max = 35)

validate(ValiktorCompanyAAgencyForm::email).hasSize(min = 1, max
= 128)

validate(ValiktorCompanyAAgencyForm::phoneNumber).withMessage("p
honeNumber or email must be present") {
                (!form.phoneNumber.isNullOrBlank() ||
!form.email.isNullOrBlank())
            }
        }
```

Joonis 12. Valiktor Company A vormi valideerimise täiendatud näide.

Andmemudeli valideerimisreeglite kirjeldamine on kompaktne ning võimalik on defineerida ka vajadusel üksikuid reegleid või reeglikomplekte. Valiktori kõikide reeglite loogika tõlgendab null väärtust kui reegli edukat täitmist. Null väärtuse vältimiseks on eraldi reegel *isNotNull()*.

Valiktor pakub tüübikindlat valideerimislahendust kasutades minimaalselt Kotlini reflektsiooni teegi funktsionaalsust. Valideerimisloogika on kompaktne, kuid aktiivseid edasiarendusi ei toimu. Seetõttu ei ole antud valideerimisteek pikemas perspektiivis jätkusuutlik, kuigi omab arvestavas koguses GitHub keskkonnas *stars* märgiseid.

### 2.6.9 Akkurate

Akkurate on 2023. aasta sügisel loodud valideerimisteek, mis kasutab Kotlini intuitiivse API loomise võimekust. Teegi loomisel on võetud arvesse keerukat äriloogikat ja

jätkusuutlikkust. Samuti leitakse, et otstarbekam on kasutada Kotlini DSL kui annotatsioone ning lahendus peab toetama asünkroonsust ja pakkuma võimalikult palju esmajärgulisi valideerimisreegleid. [68]

Antud teek kasutab *Kotlin Symbol Processing* (KSP) API, mille kasutajaliides on sarnane Kotlini reflektsiooni teegile. KSP on võrreldes reflektsiooniga oma eesmärkide saavutamiseks rangem, lubades olemasolevat koodi lugeda, kuid mitte muuta.

Võrreldes teiste võrdluses olevate teekidega, on antud teegi lahendus kõige uudsemate võtetega lahendatud ning on ka seetõttu kasutajakogemuse poolest veidikene võõras. GitHub keskkonnas on antud teek lühikese ajaga saavutanud arvestatava *stars* koguse.

```
val validateAkkurateCompanyAAgencyForm =
Validator<AkkurateCompanyAAgencyForm> {
    agency.isEqualTo(Agency.COMPANY_A)
    firstName.hasLengthBetween(1..128)
    lastName.hasLengthBetween(1..128)

    val reqDate = LocalDate.ofYearDay(2000, 1)
    birthDate.isAfterOrEqualTo(reqDate) otherwise {
        "Date must be after or equals to ${reqDate.year}-
${reqDate.monthValue}-${reqDate.dayOfMonth}"
    }

    constrain {
        (!it.phoneNumber.isNullOrBlank() ||
!it.email.isNullOrBlank())
    } otherwise { "phoneNumber or email must be present" }

    if (email.unwrap() != null) {
        email.hasLengthBetween(1..128)
    }

    if (phoneNumber.unwrap() != null) {
        phoneNumber.hasLengthBetween(5..35)
    }
}
```

Joonis 13. Akkurate Company A vormi valideerimise näide.

Valideerimisreegleid on võimalik kirjutada väga sarnaselt tavalisele koodile, kasutades selleks näiteks numbrite vahemiku defineerimist, mida tavaliselt kasutatakse *for-loop* tarbeks. Lõpptulemusena saadakse kasutajasõbralik ning kompaktne reeglistik.

Akkurate valideerimisteek on üks silmapaistvamaid teeke antud lõputöö võrdlusnimistus, mis kasutab Kotlini võimalusi maksimaalselt. Eesmärk on luua intuitiivne ja kompaktne

kasutajaliides, mis võimaldaks hoida koodi loetavana. Esimesed katsetused on olnud positiivsed ning kindlasti tasuks antud teeki uuesti vaadelda kui saab valmis stabiilne versioon.

### 2.6.10 Thing

Thing on reeglipõhine olemihaldusteek, mis lahendab samaaegselt andmete normaliseerimist, valideerimist ja redigeerimist. Sageli jõuavad andmed rakenduseni valel kujul. Mõnikord on see paratamatu, kuid mõningane parandamine aitab teha andmed loetavamaks. Sellisel juhul saab kasutada teegi normaliseerimise funktsionaalsust. Juhul kui töötlemata informatsiooni kasutajale kuvamine ei ole võimalus, siis selle eest vastutab redigeerimise funktsionaalsus ning andmete korrektse vastuvõtmise eest vastutab valideerimine. [69]

Baastasemel on antud teek sõltuv Kotlin Reflect teegist ning sellega loodavast tüübikindla DSL võimekusest. Spring Boot toeks kaasatakse hetkel vanema tüübi kompilaatori lisa nimega Kapt, mis on hetkel ametlikult hooldusrežiimis ning mille asendus on KSP [70], kuid antud teeki on võimalik edukalt kasutada ka ilma selleta.

Dokumentatsioon sisaldab teegi ülesseadmist ning põhilist funktsionaalsust, kuid puuduvad detailsemad näited kõikidest saadaolevatest funktsionaalsustest. Samuti ei suuda IDE antud funktsionaalsust reeglite defineerimisel kasutajale esialgselt kuvada. Kiiremaks kohanemiseks oleks vaja täiendada teegi dokumentatsiooni olemasolevast teegi võimekusest, mis väldiks lähtekoodis kasutajaliidese analüüsimist.

```
Validation {
                ThingCompanyAAgencyForm::agency {
equal(Agency.COMPANY_A) }
                ThingCompanyAAgencyForm::firstName { length(1,
128) }
                ThingCompanyAAgencyForm::lastName {
length(1,128) }
                ThingCompanyAAgencyForm::birthDate {
                    addValidator("Date must be after or equals
to 2000-01-01") {
                        checkLocalDateIsAfterOrEqualsTo(it,
LocalDate.ofYearDay(2000, 1))
                    }
                }
                ThingCompanyAAgencyForm::phoneNumber.ifPresent {
length(5, 35) }
                ThingCompanyAAgencyForm::email.ifPresent {
length(1, 128) }
                ThingCompanyAAgencyForm::phoneNumber {  }

                addValidator("phoneNumber or email must be
present") {
                    (!it.phoneNumber.isNullOrBlank() ||
!it.email.isNullOrBlank())
                }
            }
```

Joonis 14. Thing Company A vormi valideerimise täiendatud näide.

Kasutajaliidest tundes on uue reegli kirjeldamine intuitiivne ning ühtne. Üldine ülesehitus on sarnane teiste teekidega ning seetõttu on ka teiste teekide reegleid võimalik ümber tõsta Thing teegile.

Thing teek lahendab samaaegselt andmete normaliseerimist, valideerimist ja redigeerimist. Kuigi pakutav funktsionaalsus võib olla vajalik, ei ole antud teeki otstarbekas kasutada ainult valideerimiseks, kuna väga sarnase reeglite kirjeldamisviisiga on ka teisi teeke.

### 2.6.11 Kalidation

Kalidation on teek, mis pakub sarnaselt paljudele teistele Kotlini teekidele reflektsioonil põhinevat tüübikindlat DSL kasutust, kuid antud juhul olles pigem ühenduslüliks erinevate valideerimisteekide kasutamiseks. Selle saavutamiseks järgitakse lähenemist, kus valideerimist hoitakse andemudelist eraldi ning teegi toimimiseks on vajalik eraldi kaht valideerimisteeki ning üht funktsionaalse programmeerimisparadigmade kasutuseks. [71]

Baastehnoloogia kattuvus on antud teegil sarnane teiste Kotlin Reflect kasutatavate teekidega. Andmete valideerimiseks on kasutatud standardiseeritud Jakarta Validation API implementeerivat Hibernate Validatorit. Seega toimib teek pigem ühenduslülina, kuigi kasutatava valideerimisteegi välja vahetamise võimalus on tülikas ja ilmselt väikse tõenäosusega, kuna Jakarta Validation standardit kasutab ametlikult ainult Hibernate Validator. Kalidationi võrdlusesse toomine ei anna erilisi eeliseid eelmainitud põhjustel. Lisaks on väljadevaheliste sõltuvuste kirjeldamine raskendatud.

### 2.6.12 Validoctor

Validoctor on universaalne valideerimisteek nii Java kui ka Kotlini tagarakendustele. Valideerimist teostatakse funktsiooni abil, mille sisendiks on valideeritav objekt ja määratud reeglistik. Sellise lahenduse eesmärk on luua keeruka struktuuriga andmemudelile lihtsasti järgitav valideerimisreeglistik. [72]

Antud teek kasutab Java reflektsiooni leidmaks andmemudeli väljad, millele reeglit rakendada. Reflektsiooni kasutus on minimaalne ning ei raskenda üldise koodibaasi mõistmist. Valideerimise vastust on võimalik konfigureerida nii tavalise funktsiooni vastena kui ka erindina. [72]

Dokumentatsioon sisaldab kasutatavaid termineid, lahendatava probleemi ajendeid, seadistust ning näiteid. Samuti on näidatud erinditega tegelemist Springis. Uue reegli kirjeldamise näitest leidub eripära võrreldes teiste teekidega. Reegli sõnumid on *enum* tüüpi, mis annab oletust teeki haldava ettevõtte kasutavast tõlke lahendusest, kuid puudub täielik nimistu eeldefineeritud väärtustest. [72]

Aktiivne kommuun on üks olulisi tegureid teegi jätkusuutlikkuses. Antud teeki haldab Poola ettevõtte Miquido ning leiab nende projektides kasutust. Seetõttu saab pidada antud teegi seisundit aktiivseks, sest nende poolt on tehtud pidevalt edasiarendusi. Kasutatud on Semantic Versioning spetsifikatsioonile omast versioniseerimist ning praegune viimane versioon on 2.1.3. Teek on kasutuskõlbulik lõpptoodanguks.

Uue reegli kirjeldamine on toodud näitematerjalides. väljadevaheliste reegli puhul on võimalik kirjutada reegel otse valideerimisreeglistiku, kuid välja reegel tuleb vajadusel defineerida eraldi. Samuti võib eraldi reegleid koguneda oma jagu, sest antud teeki ei ole lisanud erinevate aega hoidvate tüüpide reegleid. Samas võib tekkida hulga lihtsustusi,

kuna mitmele väljale saab samaaegselt defineerida samad reeglid, mis muudab omakorda kogu reeglistiku selgemaks ja kompaktsemaks.

```
val validoctorCompanyAAgencyFormValidator:
Rule<ValidoctorCompanyAAgencyForm> =

Validoctor.rulesFor(ValidoctorCompanyAAgencyForm::class.java)
        .field(ValidoctorCompanyAAgencyForm::agency,
equalTo(Agency.COMPANY_A))
        .field(ValidoctorCompanyAAgencyForm::firstName,
stringLengthInRange(1,128))
        .fields(listOf(ValidoctorCompanyAAgencyForm::firstName,
ValidoctorCompanyAAgencyForm::lastName), stringLengthInRange(1,
128))
        .field(ValidoctorCompanyAAgencyForm::birthDate,
isAfterOrEqualTo(LocalDate.ofYearDay(2000, 1)))
        .rule("phoneNumber or email must be present",
ValidoctorCompanyAAgencyForm::phoneNumber.name) {
            !it.phoneNumber.isNullOrBlank() ||
!it.email.isNullOrBlank()
        }
        .field(ValidoctorCompanyAAgencyForm::phoneNumber,
stringLengthInRange(5, 35))
        .field(ValidoctorCompanyAAgencyForm::email,
stringLengthInRange(1, 128))
        .build()
```

Joonis 15. Validoctor Company A vormi valideerimise täiendatud näide.

Esialgselt leitakse andmemudeli väljad reflektsiooniga kasutades välja nime. Seda saab täiendada Kotlini reflektsiooni teegiga, mis pakub sama funktsionaalsust tüübikindlamalt. Antud lahendust kasutavad mitmed teised teegid ning käesoleva lõputöö autor on näitekoodibaasis defineerinud ka selle lihtsustuse.

Validoctor pakub teistest valideerimisteekidest erinevat lahendust valideerimisreeglite sõnumite tõlkimiseks, kus tõlke transleerimise vastutus on viidud valideerimisteegist välja. Vaatamata Kotlini standard teegi sõltuvusest on antud teek kirjutatud täielikult kasutades Javat ning sobib mõlemale platvormile.

# 3 Süntees

Analüüsi peatükis tutvustati programmeerimiskeelt Kotlin ja leiti kõige suurema kõlapinnaga Java veebirakenduse raamistik. Selleks on Spring Boot ning selle kaudu on ka suurem tõenäosus käesoleva töö tulemite jagamiseks. Loodi polümorfne andmemudel ning selle objektimiseks kasutatakse Jackson nimelist teeki, mis on leidnud heakskiitu kõikides võrdluses olevates raamistikes. Samuti defineeriti jätkusuutliku tarkvarateegi üldiseid omadusi ning analüüsiti valideerimisteeke.

Valideerimisteekide mugavusvalimi alusel analüüsiti üheksat valideerimisteeki. Nimistus esines kolm Javas ning kuus Kotlinis kirjutatud tarkvarateeki. Käesolev lõputöö kasutab lõpliku valideerimisteekide võrdlusel Saaty ehk analüütiliste hierarhiate meetodit. Antud meetodit kasutatakse mitme kriteeriumi ja alternatiiviga otsustuse tegemiseks. [73] Võrdluste teostamiseks on loodud mitmeid interaktiivseid veebilehti, mis selgitavad meetodi kasutust ning samuti abistavad maatriksite ja graafikute loomist. Antud töö raames on kasutatud ettevõtte Comcast vabavaralise tarkvaraosakonna loodud analüütiliste hierarhiate meetodit kasutava tööriista veebilehte [74].

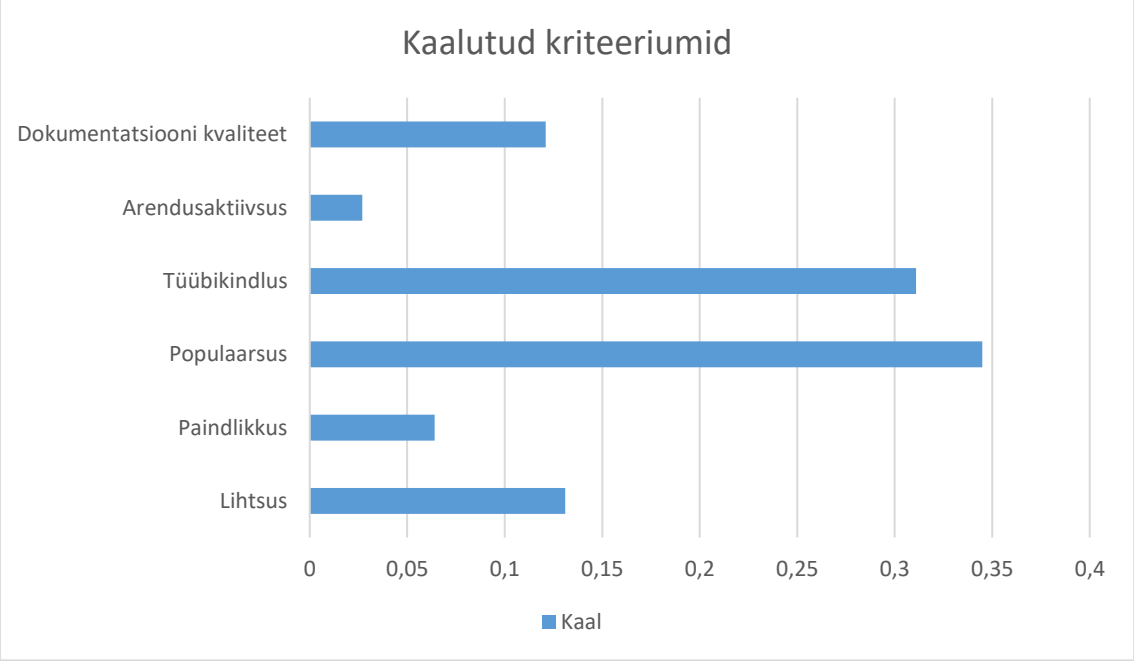
## 3.1 Valideerimisteekide võrdlus

Lõplikust võrdlusest on täiendavalt eemaldatud Java Fluent Validator ja Kalidation vastavalt teekide tutvumisel leitud puudujääkidele. Võrdlemise teostamiseks on valitud järgnevad kriteeriumid: lihtsus, tüübikindlus, paindlikus, populaarsus, arendusaktiivsus ja dokumentatsiooni kvaliteet. Kriteeriumite võrdlemiseks kasutatakse tabelites arvväärtusi üks, kolm, viis ja üheksa. Väärtus üks tähistab kriteeriumi samastumist ning üheksa tähistab kriteeriumi tugevaimat võimalikku eelistust teise kriteeriumi suhtes.

Lihtsuse kriteerium tähistab valideerimisteegi kasutajaliidese kasutamise mugavust, intuitiivsust ja hilisemat koodi loetavust. Valideerimisteegi tüübikindlus annab arendajale kindlustunde reeglite defineerimisel, võimaldades staatilisel koodikontrollil tuvastada juba tekkivaid kasutusvigu. Paindlikkuse kriteerium tähistab võimalust kasutada teeki erinevate konfiguratsioonidega ning valideerimisreegli defineerimist mitmel viisil. Populaarsus on suure kaaluga kriteerium, kuna vastav omadus on mõjukaks eelduseks teegi edasisel püsimisel. Arendusaktiivsus sõltub kohati populaarsusest, kuid käesolevas võrdluses vaadeldakse seda eraldi, kuna teek võib olla alles arendamisjärgus ning esmast

populaarsust ei pruugi olla saavutatud. Dokumentatsiooni olemasolu ning kvaliteet on uue teegi puhul oluline, sest nii on võimalik juba enne teegi võrdlusesse võtmist teha algkriteeriumite põhjal otsuseid. Samuti teeb hea dokumentatsioon hõlpsamaks teegi kasutuselevõttu.

Kriteeriumite võrdlustabeli (Lisa 8) põhjal on loodud alljärgnev joonis (Joonis 16), millele on kantud iga kriteeriumi kaalulised väärtused. Kaalude väärtused tagavad kriteeriumite võrreldavuse. Samuti aitavad samal tasemel määratletud kaalud valida vastavate kriteeriumite põhjal parima valideerimisteegi.



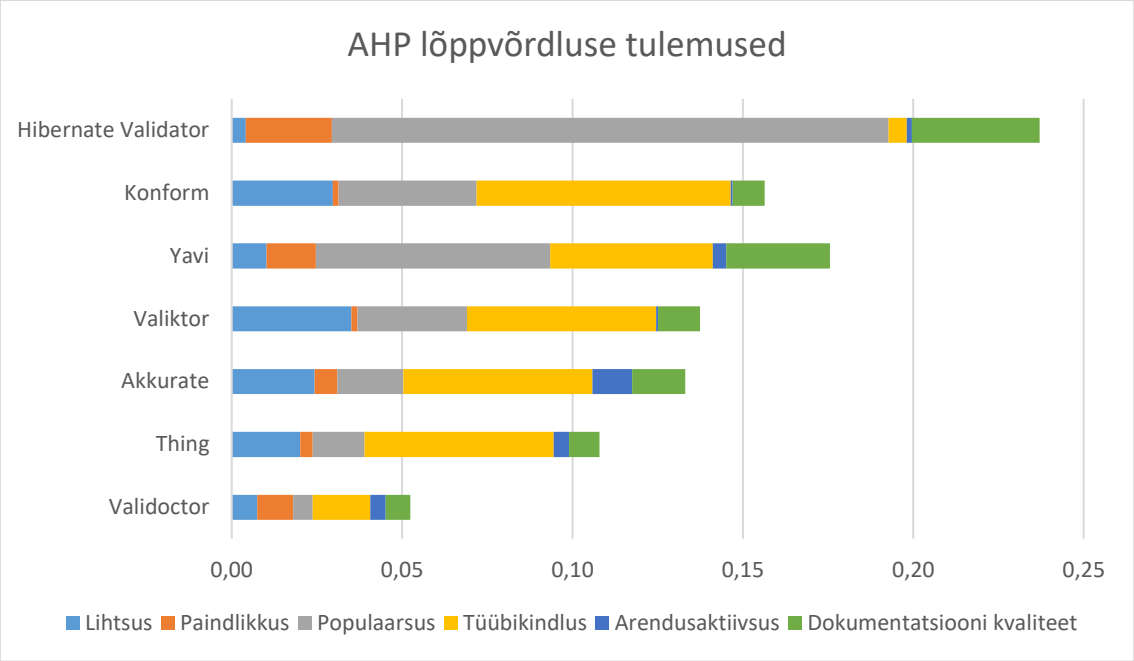

Joonis 16. AHP kaalutud kriteeriumid.

Kõige enim mõjutavad valideerimisteegi jätkusuutlikkust populaarsus ja tüübikindlus. Tüübikindlus on implementatsiooni kirjeldav omadus ning populaarsus kasutajaskonna. Lihtsus ja dokumentatsiooni kvaliteet moodustavad koos kasutajakogemuse. Arendusaktiivsus ja paindlikkus näitavad kaudselt teegi keerukusastet.

### 3.1.1 Tulemus

Kaalude rakendamiseks peab hindama ning määratlema iga kriteeriumi eelistust igal teegil. Antud kriteeriumite määratluste (Lisad 9-14) põhjal tekib alternatiivide võrdlus, mis on allolevalt jooniselt (Joonis 17) intuitiivselt loetav ning kohati ka ettearvatava tulemusega. Hibernate Validator kui kõige kõrgem tulemus annab veelgi kinnitust, et tegu on *de-facto* staatuses valideerimisteegiga, kuid eelmainitud eesmärkide ja teegi nõuetele

ei ole see teek sobiv. Üheks valideerimisteegi nõudeks oli valideerimisreeglistiku kompaktne ning ühtne kirjeldus, mis näitaks tervet andmemudelit. Samuti oli nõue hoiduda annotatsioonidel põhinevatest teekidest, kuna sellega peitub tegelik toimimisloogika ning tüübikindlust on raskem tagada.




Joonis 17. AHP lõppvõrdluse tulemused.

Jättes kõrvale Hibernate Validator teegi, on tegelikult tulemus üllatav, kuna kõige suurema väärtusega valideerimisteek Yavi on täielikult kirjutatud Javas ning Kotlinile omased võimalused ei ole seda üle kaalunud. Küll aga peab ära märkima, et Yavi teeki kasutades tuleb eelnevalt teada mõningaid erisusi. Yavi ei toeta *enum* tüüpi väljasid ning nende jaoks tuleb luua eraldiseisvad reeglid. Samuti ei ole võimalik kuvada valideerimisreegli sõnumis sisestatud andmeid. Juhul kui eelmainitud puudused on aktsepteeritavad, siis on tegu hea valideerimisteegiga ning seda on sobilik kasutada projektides, kus on nõutud keerukamaid valideerimisreegleid ja nendega seotud kitsendusi.

## 3.2 Kasutus rakenduses

Analüüsi peatükis käsitletud osade ühendamisel luuakse valideerimisreeglite kirjeldamise süsteem. Lahendusena kombineeritakse täpse grupeeringuga andmemudel ning objektimis- ja valideerimisteegid. Kasutatakse Kotlini tüüpimissüsteemi [75] ning dokumenteeritakse kasutajaliides. Kogu lahendus on üles laetud GitHub keskkonda [76].

### 3.2.1 Andmemudeli defineerimine

Andmemudelite kirjeldamiseks luuakse abstraktne klass (Joonis 18). Antud klassis on defineeritud kõik äriks vajalikud väljad vaikimisi väärtusega null. Väljade objektimise ignoreerimiseks kasutatakse *JsonIgnore* annotatsioone, mis väldivad üleliigsete väärtuste objektimist andmemudeli kujul.

```
abstract class AgencyForm {
    @field:JsonIgnore
    open val firstName: String? = null
    @field:JsonIgnore
    open val lastName: String? = null
    @field:JsonIgnore
    open val gender: Gender? = null
    @field:JsonIgnore
    open val birthDate: LocalDate? = null
    @field:JsonIgnore
    open val phoneNumber: String? = null
    @field:JsonIgnore
    open val email: String? = null
    @field:JsonIgnore
    open val socialMedia: List<SocialMedia>? = null
    @field:JsonIgnore
    open val height: Height? = null
    @field:JsonIgnore
    open val weight: Weight? = null
}
```

Joonis 18. Andmemudeli abstraktne klass agregeerimistunnuseta.

Täpse grupeeringuga andmemudeli agregeerimistunnus lisatakse järgmises abstraktses klassis (Joonis 19). Antud klassi eesmärk on agregeerimistunnuse lisamine ning objektimisteegi abil sellele kohaldumine, kus igale tunnusele vastab teatud mudeli kuju.

```
@JsonTypeInfo(
    use = JsonTypeInfo.Id.NAME,
    include = JsonTypeInfo.As.EXISTING_PROPERTY,
    property = "agency")
@JsonSubTypes(
    JsonSubTypes.Type(value =
YaviGeneralAgencyForm::class, name = "GENERAL"),
    JsonSubTypes.Type(value =
YaviCompanyAAgencyForm::class, name = "COMPANY_A"),
    JsonSubTypes.Type(value =
YaviCompanyBAgencyForm::class, name = "COMPANY_B"))
abstract class YaviAgencyForm: AgencyForm() {
    abstract val agency: Agency
}
```

Joonis 19. Andmemudeli abstraktne klass agregeerimistunnusega.

Andmemudeli lõppkuju loomiseks tuleb abstraktset klassi pärides üle kirjutada kasutatavad väljad ning defineerida nende null väärtuse võimalikkus. Alljärgnevas andmemudeli kujus (Joonis 20) on võimalikust üheksast väljast defineeritud viis. Välja üle kirjutamisega vähendatakse suurte andmemudelite erikujude kirjeldamise mahtu ning säilitatakse kujude loomisel kindlaks määratud väljade defineerimine.

```
data class YaviCompanyAAgencyForm(
   override val agency: Agency = Agency.COMPANY_A,
   override val firstName: String,
   override val lastName: String,
   override val birthDate: LocalDate,
   override val phoneNumber: String?,
   override val email: String?,
) : YaviAgencyForm()
```

Joonis 20. Andmemudeli lõppkuju.

Andmemudeli kujud defineeritakse API dokumentatsioonis vastavalt andmemudeli struktuurile. Eraldi väljade täpsemaks defineerimiseks on vajalik kasutada *Schema* annotatsiooni. Selle kaudu saab lisada väljale vajalikku infot andmemudeli edukaks kasutamiseks. Täidetud API dokumentatsiooni kasutajaliidese näide on Lisas 5.

### 3.2.2 Andmemudeli valideerimine

Päringu esmane andmemudelile vastavuse valideerimine toimub objektimisel. Kontrollitakse sisendi vastavust JSON formaadile ning konstantsete väärtuste väljade kattuvust. Samuti kontrollitakse andmemudeli null väärtuseta väljade täituvust. Täpsemaks väljade ja nendevaheliste sõltuvuste valideerimiseks kasutatakse valideerimisteekide võrdluse tulemusel teeki nimega Yavi. Igale andmemudeli kujule on defineeritud valideerimisreeglid. Nende kontroll toimub pärast objektimist.

# 4 Kokkuvõte

Antud lõputöö eesmärk oli välja töötada jätkusuutlik süsteem polümorfsete andmeedastusobjektide väljade ja nendevaheliste sõltuvuste valideerimisreeglite kirjeldamiseks, mida saab kasutada Java veebirakenduses, mille programmikood on kirjutatud Kotlinis. Kirjeldamissüsteemi vajalikkus tulenes agregeeritud ärimudelit kasutava veebirakenduse korrektsete ja lihtsasti hallatavate valideerimisreeglite nõudest ning laialdasemast tarkvaratoodete liikumisest agregeeritud ärimudelile.

Töö analüüsi peatükis koostati rakenduse raamistiku ja teekide valim ning tugineti nende seast valimisel objektiivsetest ja otstarbekatest hinnangutest. Veebirakenduse raamistikuks valiti Spring Boot, mis on kõige populaarsem Java veebirakenduste raamistikkomplekt. See annab eelduse käesolevas lõputöös valminud lähtekoodi suurema kõlapinna loomeks. Andmemudeli objektimisel kasutati Jackson nimelist teeki, mis lihtsustas polümorfsete kujude kirjeldamist ning Kotlini koosmõjul ka nullide elimineerimist. Analüüsi kõige mahukamaks osaks olid valideerimisteegid, kuna nende valik oli suurim ning nende valikul lähtuti mitmest nõudest. Valideerimisteekide analüüsimisel kasutati hierarhiate meetodit, mille tulemusel kujunes sobilikuks teek nimega Yavi.

Analüüsi tulemusena leiti sobiv valideerimisreeglite kirjeldamise süsteem, mis on piisavalt paindlik suutmaks täita reeglite kirjeldamise vajadusi, mis on tingitud agregeeritud ärimudeli kasutusest. Sellega täideti ära lõputöö eesmärk ning laiendati autori teadmisi valideerimisteekidest ja nende ülesehitusest.

# Kasutatud kirjandus


[1] R. Chmiel ja M. C. Loui, „Debugging: From Novice to Expert,“ märts 2004. [Võrgumaterjal]. Available: https://dl.acm.org/doi/pdf/10.1145/1028174.971310. [Kasutatud 12 aprill 2023].

[2] K. Kivil, „Google koondab 12 000 töötajat,“ Eesti Rahvusringhääling, 20 jaanuar 2023. [Võrgumaterjal]. Available: https://www.err.ee/1608857861. [Kasutatud 12 aprill 2023].

[3] N. Bilton, „Big Tech Companies Are Testing How Far They Can Slash Staff,“ Vanity Fair, 29 märts 2023. [Võrgumaterjal]. Available: https://www.vanityfair.com/news/2023/03/big-tech-layoffs-2023-twitter-meta-amazon-google. [Kasutatud 12 aprill 2023].

[4] R. Sama, “Aggregator Business Model: A Complete Guide,” 6 juuni 2022. [Online]. Available: https://www.linkedin.com/pulse/aggregator-business-model-complete-guide-raman-sama/. [Accessed 28 veebruar 2023].

[5] Twilio, “SMS Pricing in Estonia for Text Messaging,” Twilio, 6 märts 2023. [Online]. Available: https://www.twilio.com/sms/pricing/ee. [Accessed 6 märts 2023].

[6] Maksekeskus AS, „Pangalink,“ Maksekeskus AS, 6 märts 2023. [Võrgumaterjal]. Available: https://maksekeskus.ee/service/pangalink/. [Kasutatud 6 märts 2023].

[7] Maksekeskus AS, „Kõik saadaolevad makseviisid,“ 4 mai 2023. [Võrgumaterjal]. Available: https://maksekeskus.ee/saadaolevad-makseviisid/. [Kasutatud 4 mai 2023].

[8] Finantsinspektsioon, „Krediidiasutused,“ 4 mai 2023. [Võrgumaterjal]. Available: https://www.fi.ee/et/pangandus-ja-krediit/krediidiasutused. [Kasutatud 4 mai 2023].

[9] R. Mk, „Differentiate between client side validation and server side validation,“ 5 mai 2023. [Võrgumaterjal]. Available: https://net-informations.com/faq/asp/validation.htm. [Kasutatud 5 mai 2023].

[10] R. Raj, „Understanding the Aggregator Business Model,“ 8 juuni 2023. [Võrgumaterjal]. Available: https://medium.com/@nagendrakumarsahktr7777/understanding-the-aggregator-business-model-79d7c5cfe052. [Kasutatud 4 detsember 2023].

[11] Kotlin Foundation, „FAQ | Kotlin Documentation,“ 27 oktoober 2023. [Võrgumaterjal]. Available: https://kotlinlang.org/docs/faq.html. [Kasutatud 28 november 2023].

[12] Google Inc., „Android’s Kotlin-first approach | Android Developers,“ [Võrgumaterjal]. Available: https://developer.android.com/kotlin/first. [Kasutatud 29 oktoober 2023].

[13] A. Dolgikh, „KontlinConf 2023: A Look at the Opening Keynote,“ JetBrains s.r.o, 13 aprill 2023. [Võrgumaterjal]. Available: https://blog.jetbrains.com/kotlin/2023/04/kotlinconf-2023-opening-keynote/. [Kasutatud 18 detsember 2023].

[14] „Git,“ Software Freedom Conservancy, 5 September 2023. [Võrgumaterjal]. Available: https://git-scm.com/. [Kasutatud 5 September 2023].

[15] K. Hughes, „Bitbucket vs GitHub vs GitLab | What are the differences?,“ StackShare, [Võrgumaterjal]. Available: https://stackshare.io/stackups/bitbucket-vs-github-vs-gitlab. [Kasutatud 25 oktoober 2023].

[16] Gradle Inc., „Gradle | Gradle vs Maven Comparison,“ [Võrgumaterjal]. Available: https://gradle.org/maven-vs-gradle/. [Kasutatud 25 oktoober 2023].

[17] JetBrains s.r.o, „Productive Server-Side Development with Kotlin Programming Language,“ [Võrgumaterjal]. Available: https://kotlinlang.org/lp/server-side/. [Kasutatud 30 oktoober 2023].

[18] „spring-projects/spring-boot: Spring Boot,“ [Võrgumaterjal]. Available: https://github.com/spring-projects/spring-boot. [Kasutatud 15 detsember 2023].

[19] JetBrains s.r.o., „Framework for quickly creating connected applications in Kotlin with minimal effort,“ [Võrgumaterjal]. Available: https://github.com/ktorio/ktor. [Kasutatud 15 detsember 2023].

[20] „Micronaut Application Framework,“ [Võrgumaterjal]. Available: https://github.com/micronaut-projects/micronaut-core. [Kasutatud 15 detsember 2023].

[21] „Quarkus: Supersonic Subatomic Java.,“ [Võrgumaterjal]. Available: https://github.com/quarkusio/quarkus. [Kasutatud 15 detsember 2023].

[22] Eclipse Foundation, Inc., „Vert.x is a tool-kit for building reactive applications on the JVM,“ [Võrgumaterjal]. Available: https://github.com/eclipse-vertx/vert.x. [Kasutatud 15 detsember 2023].

[23] http4k, „The Functional toolkit for Kotlin HTTP applications. http4k provides a simple and uniform way to serve, consume, and test HTTP services.,“ [Võrgumaterjal]. Available: https://github.com/http4k/http4k. [Kasutatud 15 detsember 2023].

[24] D. Åse, „A simple and modern Java and Kotlin web framework,“ [Võrgumaterjal]. Available: https://github.com/javalin/javalin. [Kasutatud 15 detsember 2023].

[25] MvnRepository, „Maven Repository: org.springframework.boot > spring-boot-starter-web,“ [Võrgumaterjal]. Available: https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web. [Kasutatud 15 detsember 2023].

[26] MvnRepository, „Maven Repository: io.ktor > ktor-server-core,“ [Võrgumaterjal]. Available: https://mvnrepository.com/artifact/io.ktor/ktor-server-core. [Kasutatud 16 detsember 2023].

[27] MvnRepository, „Maven Repository: io.micronaut > micronaut-inject,“ [Võrgumaterjal]. Available: https://mvnrepository.com/artifact/io.micronaut/micronaut-inject. [Kasutatud 16 detsember 2023].

[28] MvnRepository, „Maven Repository: io.quarkus > quarkus-resteasy,“ [Võrgumaterjal]. Available:

https://mvnrepository.com/artifact/io.quarkus/quarkus-resteasy. [Kasutatud 16 detsember 2023].

[29] MvnRepository, „Maven Repository: io.vertx > vertx-core,“ [Võrgumaterjal]. Available: https://mvnrepository.com/artifact/io.vertx/vertx-core. [Kasutatud 16 detsember 2023].

[30] MvnRepository, „Maven Repository: org.http4k > http4k-core,“ [Võrgumaterjal]. Available: https://mvnrepository.com/artifact/org.http4k/http4k-core. [Kasutatud 16 detsember 2023].

[31] MvnRepository, „Maven Repository: io.javalin > javalin,“ [Võrgumaterjal]. Available: https://mvnrepository.com/artifact/io.javalin/javalin. [Kasutatud 16 detsember 2023].

[32] TechEmpower, „Round 21 results - TechEmpower | Web Framework Benchmarks,“ 19 juuli 2022. [Võrgumaterjal]. Available: https://www.techempower.com/benchmarks/#section=data-r21&test=db&hw=cl. [Kasutatud 25 oktoober 2023].

[33] R. Graciano, „Java EE vs J2EE vs Jakarta EE,“ Baeldung, 13 juuli 2023. [Võrgumaterjal]. Available: https://www.baeldung.com/java-enterprise-evolution. [Kasutatud 25 oktoober 2023].

[34] Gradle Inc., „Gradle Wrapper,“ [Võrgumaterjal]. Available: https://docs.gradle.org/current/userguide/gradle_wrapper.html. [Kasutatud 25 oktoober 2023].

[35] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis, S. Deleuze, M. Simons, V. Pavić, J. Bryant, M. Bhave, E. Meléndez, S. Frederick ja M. Halbritter, „Spring Boot Reference Documentation,“ 19 oktoober 2023. [Võrgumaterjal]. Available: https://docs.spring.io/spring-boot/docs/3.1.5/reference/htmlsingle/. [Kasutatud 24 oktoober 2023].

[36] Linux Foundation, „OpenAPI Initiative,“ [Võrgumaterjal]. Available: https://www.openapis.org/. [Kasutatud 2023 november 2023].

[37] K. Kikkas, „ITSPEA 6. loeng - Arvutid ja paragrahvid II: tarkvara- ja sisulitsentsid,“ 2023. [Võrgumaterjal]. Available: https://akadeemia.kakupesa.net/ITSPEA/loengud/loeng6.pdf. [Kasutatud 22 oktoober 2023].

[38] „Introducing JSON,“ [Võrgumaterjal]. Available: https://www.json.org/json-en.html. [Kasutatud 25 oktoober 2023].

[39] Cybernetica AS, „Andmekaitse ja infoturbe portaal,“ [Võrgumaterjal]. Available: https://akit.cyber.ee/. [Kasutatud 25 oktoober 2023].

[40] Google Inc., „google/json: A Java serialization/deserialization library to convert Java Objects into JSON and back,“ [Võrgumaterjal]. Available: https://github.com/google/gson. [Kasutatud 17 detsember 2023].

[41] FasterXML, LLC, „FasterXML/jackson-databind: General data-binding package for Jackson (2.x): works on streaming API (core) implementation(s),“ [Võrgumaterjal]. Available: https://github.com/FasterXML/jackson-databind. [Kasutatud 17 detsember 2023].

[42] Eclipse Foundation Inc., „eclipse-ee4j/yasson: Eclipse Yasson project,“ [Võrgumaterjal]. Available: https://github.com/eclipse-ee4j/yasson. [Kasutatud 17 detsember 2023].

[43] Eclipse Foundation Inc., „Eclipse Yasson,“ [Võrgumaterjal]. Available: https://projects.eclipse.org/projects/ee4j.yasson. [Kasutatud 17 detsember 2023].

[44] I. Sharipov, „Handling polymorphism with Gson,“ 22 juuli 2020. [Võrgumaterjal]. Available: https://medium.com/@iliamsharipov_56660/handling-polymorphism-with-gson-f4a702014ffe. [Kasutatud 17 detsember 2023].

[45] O. M. Tacu, „@JsonSubTypes vs. Reflections for Polymorphic Deserialization in Jackson,“ Baeldung, 17 august 2023. [Võrgumaterjal]. Available: https://www.baeldung.com/java-jackson-polymorphic-deserialization. [Kasutatud 17 detsember 2023].

[46] Eclipse Foundation Inc., „Jakarta JSON Binding 3.0.0,“ [Võrgumaterjal]. Available: https://projects.eclipse.org/projects/ee4j.jsonb/releases/3.0.0. [Kasutatud 17 detsember 2023].

[47] D. Åse, „Documentation - Javalin,“ [Võrgumaterjal]. Available: https://javalin.io/documentation. [Kasutatud 20 detsember 2023].

[48] Eclipse Foundation Inc., „JSON Schema | Eclipse Vert.x,“ [Võrgumaterjal]. Available: https://vertx.io/docs/vertx-json-schema/java/. [Kasutatud 20 detsember 2023].

[49] http4k, „http4k JSON Message Format Modules,“ [Võrgumaterjal]. Available: https://www.http4k.org/guide/reference/json/. [Kasutatud 29 november 2023].

[50] S. Ganesh, „Jackson Mixin — A simple guide to a powerful feature,“ Medium, 11 juuni 2018. [Võrgumaterjal]. Available: https://medium.com/@shankar.ganesh.1234/jackson-mixin-a-simple-guide-to-a-powerful-feature-d984341dc9e2. [Kasutatud 29 november 2023].

[51] M. Majka, „Choosing the Right Framework or Library for a Web Application: A Comprehensive Guide,“ Dev Community, Cracow, 2023.

[52] L. Soares, „Choosing a software library or framework,“ Medium, 2018.

[53] GitHub, Inc., „validation GitHub Topics,“ [Võrgumaterjal]. Available: https://github.com/topics/validation. [Kasutatud 20 detsember 2023].

[54] baeldung, „Java Bean Validation Basics,“ 6 november 2023. [Võrgumaterjal]. Available: https://www.baeldung.com/java-validation. [Kasutatud 20 november 2023].

[55] H. Ferentschik, G. Morling ja G. Smet, „Hibernate Validator 8.0.0.Final - Jakarta Bean Validation Reference Implementation,“ 9 september 2022. [Võrgumaterjal]. Available: https://docs.jboss.org/hibernate/stable/validator/reference/en-US/pdf/hibernate_validator_reference.pdf. [Kasutatud 14 aprill 2023].

[56] A. Ugarte, „Validation in Spring Boot,“ Baeldung, 15 aprill 2022. [Võrgumaterjal]. Available: https://www.baeldung.com/spring-boot-bean-validation. [Kasutatud 14 aprill 2023].

[57] Oracle, „Annotations,“ [Võrgumaterjal]. Available: https://docs.oracle.com/javase/1.5.0/docs/guide/language/annotations.html. [Kasutatud 24 oktoober 2023].

[58] A. Warski, „The case against annotations,“ 13 oktoober 2017. [Võrgumaterjal]. Available: https://blog.softwaremill.com/the-case-against-annotations-4b2fb170ed67. [Kasutatud 15 november 2023].

[59] A. Ciobanu, D. Gautier ja W. Cai, „JBVE (Java Bean Validation Extension),“ [Võrgumaterjal]. Available: https://github.com/nomemory/java-bean-validation-extension. [Kasutatud 23 detsember 2023].

[60] M. T. Vallim ja P. Sergio, „Java Fluent Validator,“ GitHub, 28 detsember 2021. [Võrgumaterjal]. Available: https://github.com/mvallim/java-fluent-validator. [Kasutatud 23 november 2023].

[61] N. Lochschmidt, „konform-kt/konform: Portable validations for Kotlin,“ [Võrgumaterjal]. Available: https://github.com/konform-kt/konform. [Kasutatud 23 detsember 2023].

[62] Kotlin Foundation, „Kotlin Multiplatform | Kotlin Documentation,“ 12 november 2023. [Võrgumaterjal]. Available: https://kotlinlang.org/docs/multiplatform.html. [Kasutatud 23 detsember 2023].

[63] N. Lochschmidt, „konform-kt/konform: Project Maintenance #38,“ 3 mai 2022. [Võrgumaterjal]. Available: https://github.com/konform-kt/konform/discussions/38. [Kasutatud 23 detsember 2023].

[64] T. Maki, „Yet Another Validation for Java (A lambda based type safe validation framework),“ [Võrgumaterjal]. Available: https://github.com/making/yavi. [Kasutatud 17 detsember 2023].

[65] JetBrains s.r.o, „Ktor,“ [Võrgumaterjal]. Available: https://ktor.io/. [Kasutatud 29 november 2023].

[66] R. Couto, M. Ohtake, O. Canalias, J. Feinstein, M. Perazzo ja Y. Ayad, „Valiktor is a type-safe, powerful and extensible fluent DSL to validate objects in Kotlin,“ GitHub, 29 juuni 2020. [Võrgumaterjal]. Available: https://github.com/valiktor/valiktor. [Kasutatud 29 november 2023].

[67] T. Preston-Werner, „Semantic Versioning 2.0.0,“ [Võrgumaterjal]. Available: https://semver.org/. [Kasutatud 29 november 2023].

[68] J. Pardanaud, „Overview | Akkurate,“ 29 november 2023. [Võrgumaterjal]. Available: https://akkurate.dev/docs/overview.html. [Kasutatud 30 november 2023].

[69] S. Lee, S. Oh ja J. S. Hyeon, „Thing - A rule-based entity management library written in Kotlin,“ 6 juuli 2023. [Võrgumaterjal]. Available: https://github.com/kciter/thing. [Kasutatud 1 detsember 2023].

[70] Kotlin Foundation, „kapt compiler plugin | Kotlin documentation,“ 27 oktoober 2023. [Võrgumaterjal]. Available: https://kotlinlang.org/docs/kapt.html. [Kasutatud 1 detsember 2023].

[71] R. Capraro, J. Basson ja M. Ilgner, „Kalidation = A Kotlin validation DSL,“ 13 juuli 2023. [Võrgumaterjal]. Available: https://github.com/rcapraro/kalidation. [Kasutatud 1 detsember 2023].

[72] Miquido Sp. z o.o. Sp. k., „Validoctor,“ 5 juuni 2023. [Võrgumaterjal]. Available: https://github.com/miquido/validoctor. [Kasutatud 3 detsember 2023].

[73] R. W. Saaty, „The analytic hierarchy process—what it is and how it is used.,“ 1987. [Võrgumaterjal]. Available: https://www.sciencedirect.com/science/article/pii/0270025587904738. [Kasutatud 1 mai 2023].

[74] Comcast, „Analytic Hierarchy Process (AHP) Tool,“ 9 märts 2021. [Võrgumaterjal]. Available: https://github.com/ComcastSamples/ahp-tool. [Kasutatud 1 mai 2023].

[75] Kotlin Foundation, „Null safety | Kotlin Documentation,“ 11 september 2023. [Võrgumaterjal]. Available: https://kotlinlang.org/docs/null-safety.html. [Kasutatud 3 jaanuar 2024].

[76] K. Vakker, „kuldarvakker/cross-field-validation,“ [Võrgumaterjal]. Available: https://github.com/kuldarvakker/cross-field-validation. [Kasutatud 3 detsember 2023].

# Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

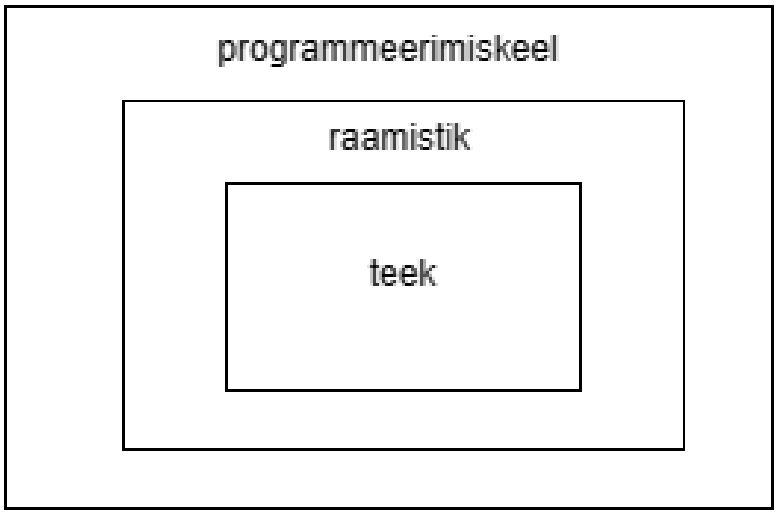
Mina, Kuldar Vakker

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Polümorfsete andmeedastusobjektide väljade ja väljadevaheliste sõltuvuste valideerimine“, mille juhendaja on Andres Käver ja Anton Sauh.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

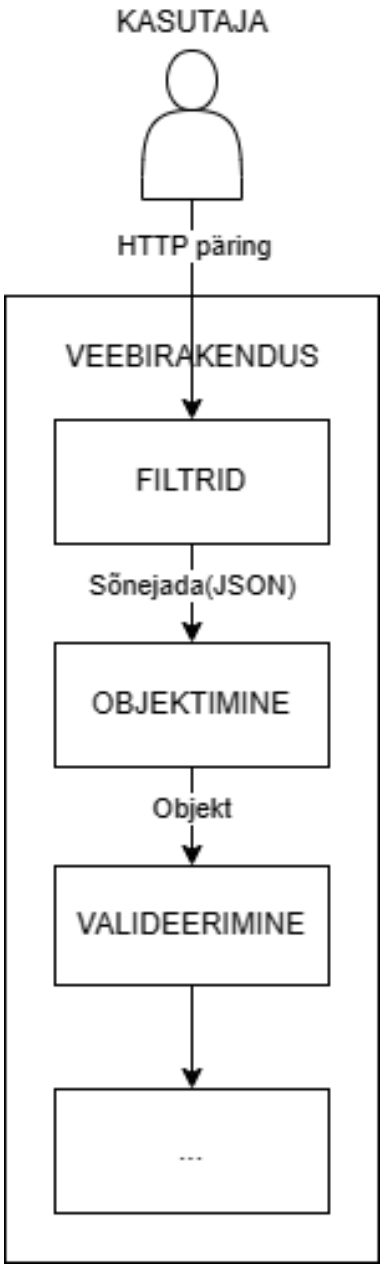
04.01.2024

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

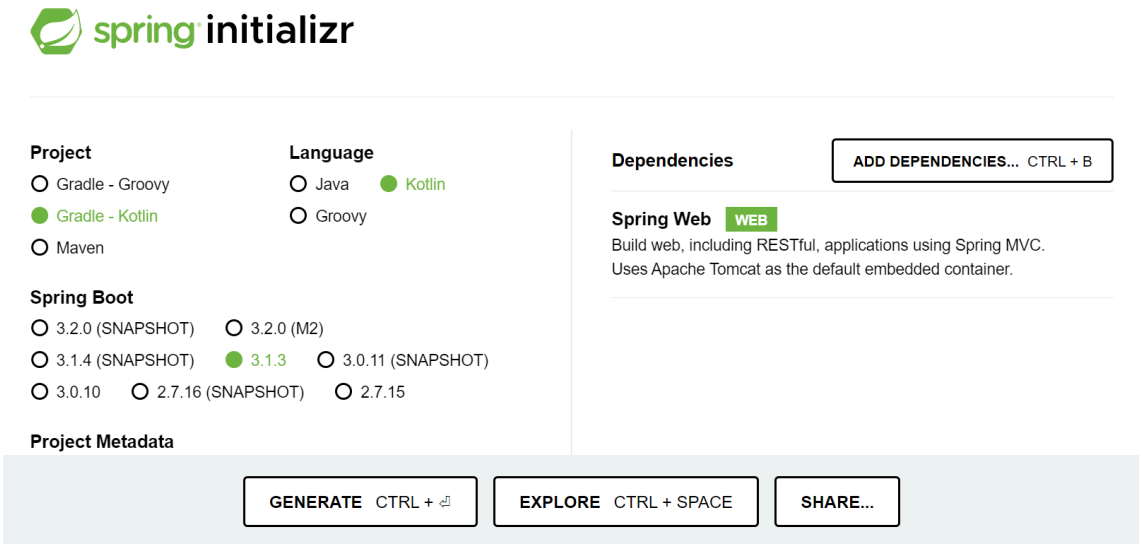
## Lisa 2 – Tarkvarakomponentide sõltuvused



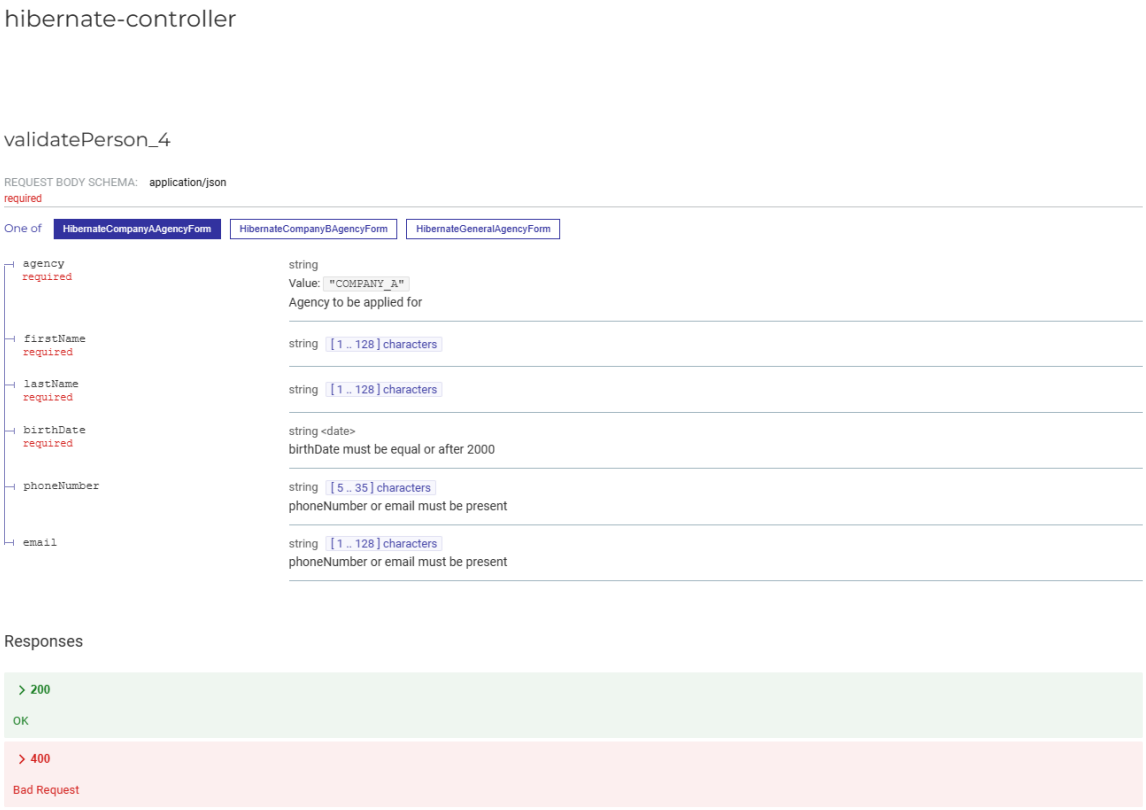
# Lisa 3 – Päringu üldine rakendusaegne täitmine



# Lisa 4 – Spring Initializr seadistuse veebikuva



# Lisa 5 – Redoc OpenAPI kasutajaliides



# Lisa 6 – Rakendusele saadetav üldine JSON tüüpi vorm

```
{
  "agency": "GENERAL",
  "firstName": "string",
  "lastName": "string",
  "gender": "MALE",
  "birthDate": "2019-08-24",
  "phoneNumber": "string",
  "email": "string",
  "socialMedia": [
    {
      "platform": "FACEBOOK",
      "profileUrl": "string"
    }
  ],
  "height": {
    "value": 0,
    "unit": "M"
  },
  "weight": {
    "value": 0,
    "unit": "KG"
  }
}
```

# Lisa 7 – Rakenduse HTTP sõnumite lugematuse erindiga tegelemise loogika

```
@RestControllerAdvice
class ErrorControllerAdvice {

    @ExceptionHandler(HttpMessageNotReadableException::class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    fun processConversionException(e:
HttpMessageNotReadableException): ValidationErrors {
        var msg: String? = null
        val cause = e.cause
        if (cause is JsonParseException) {
            val jpe: JsonParseException = cause
            msg = jpe.originalMessage
        } else if (cause is InvalidTypeIdException) {
            val itie: InvalidTypeIdException = cause
            msg = itie.originalMessage
        }
        else if (cause is MismatchedInputException) {
            val mie = cause
            msg = if (mie.path != null && mie.path.size > 0) {
                "Invalid request field: " +
mie.path[0].fieldName
            } else {
                "Invalid request message"
            }
        } else if (cause is JsonMappingException) {
            val jme = cause
            msg = jme.originalMessage
            if (jme.path != null && jme.path.size > 0) {
                msg = "Invalid request field: " +
jme.path[0].fieldName +
                        ": " + msg
            }
        }
        return
ValidationErrors(listOf(ValidationError("constraint", msg!!)))
    }
}
```

# Lisa 8 – AHP kriteeriumite võrdlus lähtuvalt eesmärgist

| **Eesmärk** | **Lihtsus** | **Paindlikkus** | **Populaarsus** | **Tüübikindlus** | **Arendusaktiivsus** | **Dokumentatsiooni kvaliteet** |
|---|---|---|---|---|---|---|
| **Lihtsus** | 1 | 3 | 1/5 | 1/3 | 5 | 3 |
| **Paindlikkus** | 1/3 | 1 | 1/5 | 1/5 | 5 | 1/9 |
| **Populaarsus** | 5 | 5 | 1 | 1 | 9 | 5 |
| **Tüübikindlus** | 3 | 5 | 1 | 1 | 9 | 5 |
| **Arendusaktiivsus** | 1/5 | 1/5 | 1/9 | 1/9 | 1 | 1/3 |
| **Dokumentatsiooni kvaliteet** | 1/3 | 9 | 1/5 | 1/5 | 3 | 1 |

# Lisa 9 – Valideerimisteekide lihtsuse kriteeriumi määramine

| Lihtsus | Hibernate | Konform | Yavi | Valiktor | Akkurate | Thing | Validoctor |
|---|---|---|---|---|---|---|---|
| **Hibernate** | 1 | 1/9 | 1/3 | 1/5 | 1/3 | 1/5 | 1/3 |
| **Konform** | 9 | 1 | 3 | 1/3 | 3 | 1 | 5 |
| **Yavi** | 3 | 1/3 | 1 | 1/3 | 1 | 1/5 | 1 |
| **Valiktor** | 5 | 3 | 3 | 1 | 1 | 3 | 5 |
| **Akkurate** | 3 | 1/3 | 1 | 1 | 1 | 5 | 3 |
| **Thing** | 5 | 1 | 5 | 1/3 | 1/5 | 1 | 3 |
| **Validoctor** | 3 | 1/5 | 1 | 1/5 | 1/3 | 1/3 | 1 |

# Lisa 10 – Valideerimisteekide paindlikkuse kriteeriumi määramine

| **Lihtsus** | **Hibernate** | **Konform** | **Yavi** | **Valiktor** | **Akkurate** | **Thing** | **Validoctor** |
|---|---|---|---|---|---|---|---|
| **Hibernate** | 1 | 9 | 3 | 9 | 5 | 5 | 5 |
| **Konform** | 1/9 | 1 | 1/9 | 1 | 1/9 | 1/3 | 1/5 |
| **Yavi** | 1/3 | 9 | 1 | 9 | 5 | 5 | 1 |
| **Valiktor** | 1/9 | 1 | 1/9 | 1 | 1/3 | 1/3 | 1/5 |
| **Akkurate** | 1/5 | 9 | 1/5 | 3 | 1 | 3 | 1/3 |
| **Thing** | 1/5 | 3 | 1/5 | 3 | 1/3 | 1 | 1/5 |
| **Validoctor** | 1/5 | 5 | 1 | 5 | 3 | 5 | 1 |

# Lisa 11 – Valideerimisteekide populaarsuse kriteerumi määramine

| **Lihtsus** | **Hibernate** | **Konform** | **Yavi** | **Valiktor** | **Akkurate** | **Thing** | **Validoctor** |
|---|---|---|---|---|---|---|---|
| **Hibernate** | 1 | 9 | 9 | 9 | 9 | 9 | 9 |
| **Konform** | 1/9 | 1 | 1/3 | 3 | 5 | 5 | 9 |
| **Yavi** | 1/9 | 3 | 1 | 5 | 9 | 9 | 9 |
| **Valiktor** | 1/9 | 1/3 | 1/5 | 1 | 5 | 5 | 9 |
| **Akkurate** | 1/9 | 1/5 | 1/9 | 1/5 | 1 | 3 | 9 |
| **Thing** | 1/9 | 1/5 | 1/9 | 1/5 | 1/3 | 1 | 9 |
| **Validoctor** | 1/9 | 1/9 | 1/9 | 1/9 | 1/9 | 1/9 | 1 |

# Lisa 12 – Valideerimisteekide tüübikindlus kriteeriumi määramine

| **Lihtsus** | **Hibernate** | **Konform** | **Yavi** | **Valiktor** | **Akkurate** | **Thing** | **Validoctor** |
|---|---|---|---|---|---|---|---|
| **Hibernate** | 1 | 1/9 | 1/9 | 1/9 | 1/9 | 1/9 | 1/9 |
| **Konform** | 9 | 1 | 5 | 1 | 1 | 1 | 5 |
| **Yavi** | 9 | 1/5 | 1 | 1 | 1 | 1 | 5 |
| **Valiktor** | 9 | 1 | 1 | 1 | 1 | 1 | 5 |
| **Akkurate** | 9 | 1 | 1 | 1 | 1 | 1 | 5 |
| **Thing** | 9 | 1 | 1 | 1 | 1 | 1 | 5 |
| **Validoctor** | 9 | 1/5 | 1/5 | 1/5 | 1/5 | 1/5 | 1 |

# Lisa 13 – Valideerimisteekide arendusaktiivsus kriteeriumi määramine

| **Lihtsus** | **Hibernate** | **Konform** | **Yavi** | **Valiktor** | **Akkurate** | **Thing** | **Validoctor** |
|---|---|---|---|---|---|---|---|
| **Hibernate** | 1 | 5 | 1/5 | 5 | 1/9 | 1/9 | 1/9 |
| **Konform** | 1/5 | 1 | 1/9 | 1 | 1/9 | 1/9 | 1/9 |
| **Yavi** | 5 | 9 | 1 | 9 | 1/5 | 1 | 1 |
| **Valiktor** | 1/5 | 1 | 1/9 | 1 | 1/9 | 1/9 | 1/9 |
| **Akkurate** | 9 | 9 | 5 | 9 | 1 | 5 | 5 |
| **Thing** | 9 | 9 | 1 | 9 | 1/5 | 1 | 1 |
| **Validoctor** | 9 | 9 | 1 | 9 | 1/5 | 1 | 1 |

# Lisa 14 – Valideerimisteekide dokumentatsiooni kvalideedi kriteeriumi määramine

| **Lihtsus** | **Hibernate** | **Konform** | **Yavi** | **Valiktor** | **Akkurate** | **Thing** | **Validoctor** |
|---|---|---|---|---|---|---|---|
| **Hibernate** | 1 | 3 | 3 | 3 | 5 | 3 | 3 |
| **Konform** | 1/3 | 1 | 1/3 | 1/3 | 1/3 | 1 | 3 |
| **Yavi** | 1/3 | 3 | 1 | 5 | 3 | 5 | 5 |
| **Valiktor** | 1/3 | 3 | 1/5 | 1 | 1/3 | 3 | 1 |
| **Akkurate** | 1/5 | 3 | 1/3 | 3 | 1 | 3 | 1 |
| **Thing** | 1/3 | 1 | 1/5 | 1/3 | 1/3 | 1 | 3 |
| **Validoctor** | 1/3 | 1/3 | 1/5 | 1 | 1 | 1/3 | 1 |