TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Jevgeni Tšerpak 155302IAPB

SMART-ID INTEGREERIMINE FINANTSTEENUSPAKKUJA INFOSÜSTEEMI

Bakalaureusetöö

Juhendaja: Martin Rebane
MSc

Tallinn 2018

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: Jevgeni Tšerpak

08.12.2017

Annotatsioon

Käesoleva lõputöö “Smart-ID integreerimine finantsteenuspakkuja infosüsteemi” eesmärgiks on teenuspakkujal olemasolevasse infosüsteemi lisada võimalus Tellerite jaoks Smart-Id kontode väljastamiseks klientidele.

Töö tulemusena valmis Spring Booti veebirakendus, mille osadeks on kontroller, retrofiti liides, teenused, teenuste adapter ja nende ühenduslüli. Kontroller on põhiline lüli, milles seisneb kogu loogika nii teenuste väljakutsumiseks kui ka vaadete kuvamiseks. Samuti vastutab kontroller ka andmete valideerimise eest.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 36 leheküljel, 5 peatükki, 10 joonist.

Abstract

Smart-ID integration to the financial service provider's data system

The primary goal of the thesis "Smart-ID integration to the financial service provider's data system" is developing a new interface for bank tellers to be able to issue a new Smart-ID account to the client. Currently there is nothing similar in the system as it is absolutely new service.

The result of this tesis is completely developed application on Spring Boot base that consists of a Controller, Model and multiple Views. The application must be able to collect data from current system, have possibility to add missing information and have a stable connection with Smart-ID API from the external service provider.

The thesis is in Estonian and contains 36 pages of text, 5 chapters, 10 figures.

Lühendite ja mõistete sõnastik

AJAX	Asynchronous JavaScript And XML
Bean	Objekt, Spring IoC konteineris.
HTTP	HyperText Transfer Protocol
IoC konteiner	Konteiner, mis initsialiseerib, hoiustab ja seadistab objektid
Java	Programmeerimise keel
JavaScript	Programmeerimise keel
JSON	JavaScript Object Notation
OkHttp	HTTP liides töö lihtsustamiseks
REST	Päringute tegemise disaini kontseptsioon
Spring	Raamistik Java aplikatsioonide loomiseks
ZK	Raamistik kasutajaliidese loomiseks Javas
ZUML	ZK User Interface Markup Language
XML	Extensible Markup Language

Sisukord

1 Sissejuhatus 9
2 Taust ja probleem 10
2.1 Olemasolev infosüsteem 10
2.2 Smart-ID 11
2.3 Spring Boot 11
2.4 Retrofit ja Gson 12
2.5 ZK Framework 12
2.6 Probleem ja eesmärk 13
2.7 Protsess kasutaja vaates 13
3 Arhitektuur 14
3.1 Kogu kasutusujuhtumi kontroller 16
3.2 Andmete hoidmise kontseptsioon 16
3.3 Päringute tegemise moodus 17
3.4 Kasutajaliidese vormimise moodus 18
4 Realisatsioon 19
4.1 Smart-ID konto loomise protsess 19
4.1.1 Klient 19
4.1.2 Esimene teller 19
4.1.3 Teine teller 20
4.2 Uus UseCase 20
4.3 Päringuteks nõutud objektid 22
4.3.1 Kliendi Smart-ID kasutajate päring 23
4.3.2 Smart-ID registratsiooni sessiooni alustamine 24
4.3.3 Smart-ID kinnitamine 26
4.4 Nõutud sisemised teenused 28
4.5 Smart-ID connector 30
4.6 Retrofit'i seadistamine 32
4.7 Vaated 35
5 Kokkuvõte 37
6 Kasutatud kirjandus 38

7 Lisa 1 – Kliendi Smart-ID kontode vaade 39
8 Lisa 2 – Uue konto loomise alustamise vaade 39
9 Lisa 3 – Uue konto loomise alustamise täidetud vaade 39
10 Lisa 4 – Teise osapoole valimise ja andmete lisamise vaade.....40
11 Lisa 5 – Teise osapoole valimise ja andmete lisamise täidetud vaade.....40
12 Lisa 6 – Teise telleri vaade.....41
13 Lisa 7 – Teise telleri vaade peale allkirjastamist 41
14 Lisa 8 - SmartIdAgreements.zul 42
15 Lisa 9 - SmartIdAgreementNew.zul 43
16 Lisa 10 - AddSecondTeller.zul 43
17 Lisa 11 - SmartIdFilesetDetails.zul.....44

Jooniste loetelu

Joonis 1 Smart-Id integratsiooni arhitektuur 15
Joonis 2.1 Kliendi kasutusjuhtum..... 19
Joonis 2.2 Esimese telleri kasutusjuhtum 19
Joonis 2.3 Teise telleri kasutusjuhtum..... 20
Joonis 3.1 Vaadeldavate vastuste visuaalne kujutus. 22
Joonis 3.2 SmartIdPersonAccounts andmete struktuur 23
Joonis 3.3 SmartId registratsiooni alustamise päringu struktuur..... 24
Joonis 3.4 SmartId registraatsiooni alustamise vastuse struktuur. 25
Joonix 3.5 Smart-ID kinnitamise päring..... 26
Joonis 3.6 Smart-ID kinnituse vastus 26

1 Sissejuhatus

Siiamaani on kasutuses olnud Eestis ainult kaks digiallkirjastamise meetodit:Mobiil-ID, ID-kaart [10]. Prognoosides SIM-kaardi vabade telefonide ilmumist turule tekkis vajadus uue allkirjastamise meetodi järele, mis ei sõltuks telefonis asuvast SIM-kaardist. Sertifitseerimiskeskuse poolt hakati looma aplikatsiooni nimega „Smart-ID“, mis seob isikut otse telefoni endaga. 2016. aasta teises pooles tekkis juba demo versioon ning finantsteenuspakkujad võisid hakata intregreerima „Smart-ID“ kasutajate haldamist oma infosüsteemidesse.

Käesoleva lõputöö eesmärgiks on integreerida Smart-ID kontode haldamist olemasolevasse infosüsteemi, tagades süsteemi korrektset suhtlust Sertifitseerimiskeskusega kasutades nende loodud API't. Lisaks tekkib vajadus andmete kuvamiseks ning visuaalsel moel nendega töötamiseks. Praeguses süsteemis puudub sarnane teenus, mida oleks võinud laiendada või teha uus arendus säilitades sarnast arhitektuuri.

Antud töö raames vaadeldakse ainult kliendi olemasolevate Smart-ID kontode pärimist ja uue konto loomist.

Tulemusena peab valmima süsteem, milles teller saab kuvada kliendi olemasolevad Smart-ID kontod ja vajadusel läbida uue konto loomise protsessi. Eelnimetatud funktsionaalsuse toimimiseks peab olema integreeritud 4 päringut: kliendi kontode pärimiseks, uue konto registreerimise sessiooni alustamiseks ja kaks protsessi kinnitust. Samuti on vaja luua kasutajaliides vastava informatsiooni kuvamiseks ja puuduvate andmete sisestamiseks.

Probleemi põhikeerukus seisneb autori poolses arusaamas infosüsteemist ja pidevalt muutuvates andmetes. Suur osakaal on ka süsteemi turvalisusel, millega peab arvestama iga protsessi sammul. Suurimaks väljakutseks pean korrektselt toimiva päringutevahetuse ja uue raamistiku abil kasutajaliidese loomist.

2 Taust ja probleem

2.1 Olemasolev infosüsteem

Antud finantsteenuspakkuja infosüsteemi, mida kasutavad tellerid on kirjutatud kasutades Spring Boot'i Java [2] programmeerimiskeeles. Kasutuses ja ringluses on suur hulk andmeid. Staatilised andmed hoitakse peamiselt andmebaasis, dünaamilised andmed on hoitud sessioonides Bean'i [8] salvestamiseks mõeldud failides. Nende puhul peab alati arvestama protsesside ristumist ja andemete muutust ning kadumist. Kogu kasutajaliides on kirjutatud kasutades ZK raamistikku, selle abil on kirjeldatud iga kasutaja jaoks mõeldud vaade.

Süsteemi omapäraks pean selle sobivust erinevates riikides toimimiseks. Selle võimalused ja nõudmised on igas seda kasutatatavas riigis erinevad ning selletõttu peab kood olema võimalikult dünaamiline. Lõputöö raames vaadeldatkse ainult Eesti infosüsteemi integreerimist.

Toimiv infosüsteem seab omakorda piirangud tehnoloogiate valikuks, nimelt ei ole võimalik kasutada teist programmeerimiskeelt kui Java, kasutajaliidese loomise põhimõtteid ja raamistikku ei ole võimalik muuta ning üldine integratsiooni arhitektuur peab sarnanema olemasolevate teenustega.

Lisades uut funktsionaalsust töö raames on plaanis kasutada Retrofit'i raamistikku ja Gson'i lisa, mis hakkavad vastutama süsteemiväliste päringute eest.

2.2 Smart-ID

Kasutaja vaates Smart-ID [3] kujutab endast ette turvalist digitaalset autentifitseerimise ja allkirjastamise võimalust läbi mobiilset aplikatsiooni. Üldplaanis on Smart-ID kolmeosaline süsteem, millest esimene on mobiilne rakendus, teiseks on serveripoolne rakendus, kus toimub andmete kontroll ja nende turvalisus ning kolmandaks on Smart-ID kasutamist implementeerinud erinevad teenuspakkujad. Antud aplikatsioon ning selle abil toimingute tegemine on realiseeritud SK ID Solutions poolt. Samuti on nende poolt realiseeritud Smart-ID turvalisus.

Smart-ID eeliseks on puuduv sõltuvus SIM-kaardist ja tasuta kasutamise võimalus. Samuti on seda võimalik luua individuaalselt kasutades Mobiil-ID'd või ID-kaarti [3] . Inimesed kellel pole antud võimalust, saavad luua konto oma finantsteenuspakkuja juures.

2.3 Spring Boot

Infosüsteemis on kasutatud Spring Boot'i [1] , mis teeb lihtsaks eraldiseisvate aplikatsioonide loomise, mida on võimalik jooksutada serveril. Erinevalt tavalisest Spring'ist [11] , teeb Spring Boot enamuse vajalikest konfiguratsioonidest ise, säästes aega aplikatsiooni käivitamiseks.

Mitmest alamprojektist koosneva aplikatsiooni käivitamine ja koos hoidmine võib olla tülikas, Spring Boot'i abil on aga projektidevaheliste seoste loomine tehtud lihtsalt ja arusaadavaks [1] . Selleks on vaja panna neid üksteisest sõltuma ja peamises projektis lisada alamprojektid moodulitena konfiguratsiooni.

2.4 Retrofit ja Gson

Retrofit on REST [12] päringute klient Java ja Androidi jaoks [5] . Selle abil saab üsna vähese vaevaga saata või vastu võtta JSON (või muu struktureeritud andmetüübi) päringud läbi REST'i põhise veebiteenuse [5] . Retrofiti jaoks on vaja konfigureerida konverter, mille abil saab liikuvad andmed konverteerida kas Java objekti kujule või JSON kujule [5] . Tüüpiline ja ka lõputöös kasutatud konverter on Gson [6] samuti tuntud kui Google Gson, kuid on võimalik lisada ka muud konvertorid näiteks XML'i [13] töötlemiseks.

Antud raamistik kasutab OkHttp raamistikku HTTP päringute jaoks [4] . Selles on võimalik teha nii sünkroonseid, kui ka asünkroonseid päringud ning saata päringuid, mis nõuavad autentimist saaja poolel [5] .

Retrofit'iga töötamiseks on vaja põhiliselt kolm klassi, milleks on: "*Model*" ehk objekti klass, mille Gson loob JSON päringu töötlemisel;"Interface", milles defineeritakse kõik võimalikud HTTP operatsioonid;"Retrofit.Builder", mis kasutab Interface ja Builder API't, mille abil saab defineerida sihtpunkti päringute jaoks [4] .

2.5 ZK Framework

ZK [7] on komponendipõhine UI raamistik, mis võimaldab luua veebi- ja mobiilirakendusi ilma, et peaks kasutama eraldi JavaScript'i või AJAX'it [7] . Võimalik on ehitada interaktiivseid ja reageerivaid AJAX veebirakendusi kasutades ainult Java't. ZK pakub kasutamiseks suure arvu komponente, mis on mõeldud erinevate kasutusviiside jaoks. Näiteks mõned on suure andmehulga kuvamiseks, teised on hoopis kasutaja poolt andmete sisestamiseks. Komponente on võimalik kergelt luua XML formaadis olevas keeles ZUL [14].

Kõik kasutaja interaktsioonid veebilehega on võimalik jälgida ja töötelda kontrolleris. Näiteks saab kasutaja andmesisestuse peale muuta lehe sisu ja muutus kuvatakse brauseris automaatselt. Arendajal pole vajadust brauseri ja serveri vahelise suhtluse eest hoolt kanda, see on juba tehtud ZK poolt.

2.6 Probleem ja eesmärk

Praeguse infosüsteemi probleemiks on puuduv valmis lahendus Smart-ID väljastamiseks klientidele. Süsteemis ei ole ühtegi sarnast teenust varasemalt implementeeritud, seega puudub taaskasutamise võimalus. Samuti ei eksisteeri ka lahendust vastavate andmete kuvamiseks ja töötlemiseks, mis vastaks projekti nõuetele.

Eesmärgiks on ehitada Smart-ID API dokumentatsiooni põhjal suhtluskanal, integreerides kõik vajalikud HTTP operatsioonid. Peale seda on vaja luua kõikide kasutusjuhtumite jaoks kasutajaliides ning siduda see kontrolleriga, mis hakkab töötlema saadetavad ja saadud andmed. Peab arvesse võtma ka võimalikud veateated.

Tulemusena peab valmima süsteem, milles teller saab kuvada kliendi olemasolevad Smart-ID kontod ja vajadusel läbida uue konto loomise protsessi.

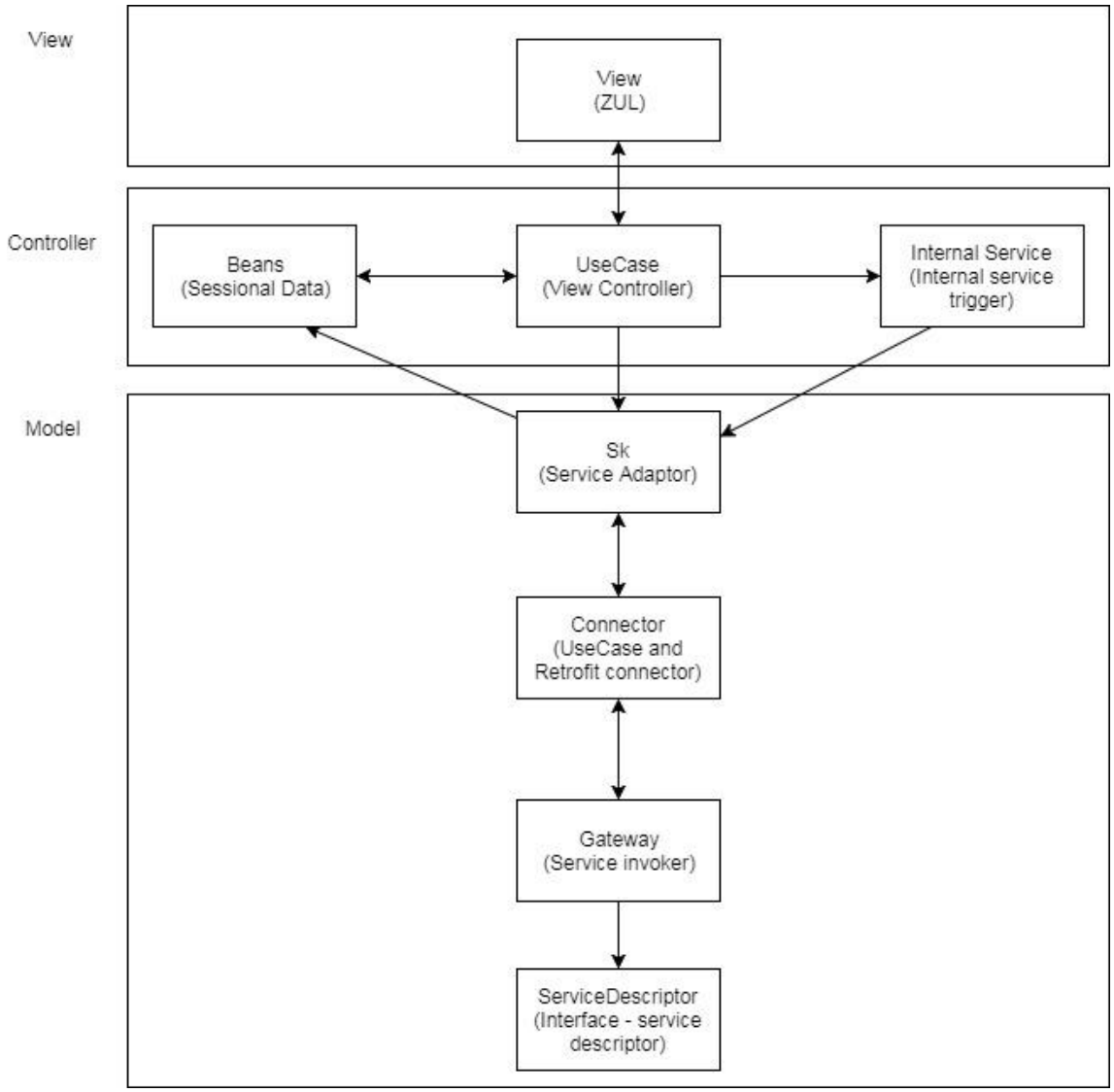
2.7 Protsess kasutaja vaates

Loodava tarkvara eesmärgiks on realiseerida järgnev protsess:

1. Esimene Teller avab Smart-ID menüüpunkti eelnevalt olles valinud klinedi, vt. Lisa 1.
2. Esimene Teller vajutab avanenud aknas nupule „Uus konto“.
3. Avanenud vaates, esimene Teller sisestab Kliendi poolt talle esitatud Token mobiilirakendusest, vt. Lisa 2.
4. Esimene Teller vajutab nupule „Trüki avaldus“.
5. Avanenud uues vaates (vt. Lisa 3), esimene Teller valib nimekirjast teise osapoole , laeb kliendi dokumendi koopia kasutades ülemist „Lae fail“ nuppu ning allkirjastatud avalduse koopia kasutades alumist „Lae fail“ nuppu.
6. Märkides andmete õigsuse linnukese, vajutab esimene Teller nupule „Kinnita“, mis saadab avalduse teisele osapoolele.
7. Teise Telleri vaates (vt. Lisa 4) on näha kõik lisatud koopiad.
8. Teine Teller avab avalduse pdfi, milles on kirjas kontrollkood ning sisestab selle väljale „Kontrollkood“.
9. Teine Teller märgib ära andmete õigsuse välja ning vajutab allkirjastamise nupule peale mida on konto loomise protsess lõppenud.

3 Arhitektuur

Rakenduse arhitektuuri (vt Joonis 1) planeerides oli tähtis säilitada olemasoleva infosüsteemi kasutusjuhtumitel baseeruva arhitektuuri samal ajal implementeerida Retrofit'i poolt nõutud arhitektuuri. Kogu projekti raames kasutatakse MVC [9] ja Mikroteenuste [15] arhitektuuri kombinatsiooni. Infosüsteem on ehitatud mikroteenuste baasil ehk antud juhul on see erinevate põhi kasutusjuhtumi laiendanud teenuste suur kogumik. Kõik teenused on omakorda ehitatud MVC baasil, kus *View* rollis on ZUL failid, *Controller*'iks on vastav kasutusjuhtumi klass ja *Model* on muutuv. Smart-ID puhul on *Model* rollis peamiselt *Connector*, mis vastutab korrektse päringu saatmise eest.



Joonis 1 Smart-Id integratsiooni arhitektuur

Kogu arhitektuuri põhjaks on Spring Boot. *View* osas on kasutatud vaid ZK raamistikku ja sellega kaasnevaid ZUL laiendiga faile. Nii Controller kui ka *Model* kasutavad bean'e ning on kirjutatud Java programmeerimiskeeles. Lisaks *Model* raames on kasutuses Retrofit koos Gson'iga.

3.1 Kogu kasutusujuhtumi kontroller

UseCase on kontroller, mis haldab endas kindla vaate äriloogikat, kirjeldab käsitletavate andmete struktuuri, hoiab endas informatsiooni beanidest, mis kasutatakse kindla operatsiooni läbiviimiseks.

UseCase'i vastutus:

- Väljade valideerimine
- Kombineeritud väljade valideeriminest
- Väljade parsimine
- Andmete Cachimine
- Visuaalse kuvamise loogika
- Autoriseerimine

UseCase haldab ja kontrollib igat telleri interaktsiooni süsteemiga. See ühendab endas vaadete (ZUL failide) kuvamist, bean'ides olevate andmete kontrollimist ja infosüsteemi sisemiste teenuste käivitamist.

3.2 Andmete hoidmise kontseptsioon

Bean on objekt, mis on kokkupandud, initsialiseeritud ja juhitud Spring IoC konteineris [8] . Igast olemasolevast java klassist on võimalik teha bean, salvestades selle sessiooni põhisesse vahemällu. Antud ajutiste andmete hoidmise viis on peamine andmete transportimise lahendus Smart-ID integreerimise käigus.
UseCase klassis on juba olemas toimiv lahendus objektide salvestamiseks ning teistele kasutuseks on tehtud kaks meetodit:

- **public Object addBeanToCache(Object bean)**, mis nõuab parameetriks lisatava objekti ning seejärel salvestab selle sessiooni.
- **public Object getBeanFromCache(Class<?> beanClass)**, mis nõuab soovitud objekti klassi ning seejärel tagastab leitud objekti või veateate, kui vastavat objekt puudub.

3.3 Päringute tegemise moodus

Testides Java'sse sisseehitatud HTTP päringute edastamise võimalust, jõudsin selgusele, et see ei võimalda omada piisavat paindlikust ja kerget andmete konverteerimist JSON päringute puhul.

Smart-ID integratsiooni käigus on vaja konverteerida mitmed erinevad päringud. Selleks kirjutada keerulised ja pikad HTTP päringud ja nende käsitlus ei ole otstarbeline ega eriti esteetiline. Retrofiti abiga on võimalik andmete töötlemiseks luua eelnevalt vastavad objektid ja hiljem vaid seadistada suhtluskanal ja käivitada päringud. Retrofit automaatselt tagastab vastava objekti, millega saab väga mugavalt edasi tegeleda.

SmartIdConnector klass hakkab tegelema päringute käivitamisega ja andmete spetsiifiliste modifikatsioonidega.

Lõputöö raames toon välja nelja päringu protsessi, mis hõlmavad põhilist ja primitiivset Smart-ID väljastamist.

3.4 Kasutajaliidese vormimise moodus

ZK Framework [7] abil on tehtud oluliselt kergemaks andmete kuvamise bean'idest ja ka nendesse salvestamine. ZUL failidega on kirjeldatud kõik vaated ning kuvatavad andmed. Kui muidu oleks vaja luua objektid, jälgida mida kasutaja on sisestanud, filtreerida, modifitseerida ja salvestada seda kõike, siis ZK abiga saab seda teha vaid kirjelades vaate XML'i adopteerinud keeles ZUML.

```
<form beanName="SkPersonInformation">
   <fields>
     <field fieldName="country" readonly="false" />
     <field fieldName="personIdentifier" readonly="false" />
     <field fieldName="forename" readonly="false" />
     <field fieldName="surname" readonly="false" />
     <field fieldName="email" readonly="false" />
     <field fieldName="phone" readonly="false" />
   </fields>
</form>
```

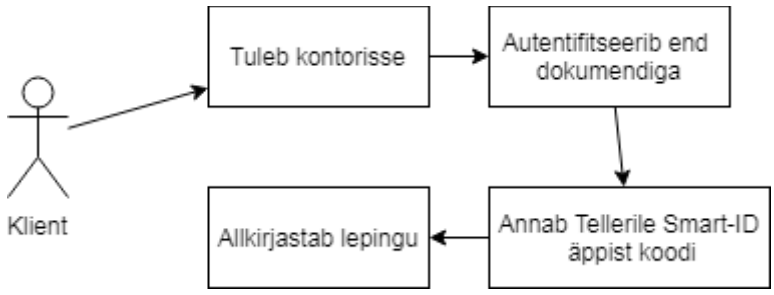
Antud koodilõik kuvab field'idega välja toodud andmed SkPersonInformation objektist, mis on bean'idesse salvestatud.

4 Realisatsioon

4.1 Smart-ID konto loomise protsess

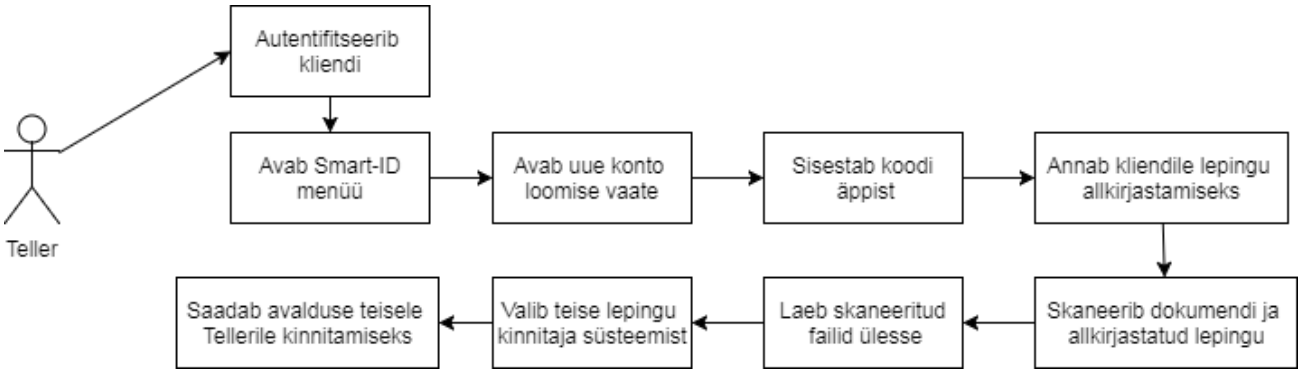
Smart-ID väljastamise protsess hõlmab kolme osapoolt, kellest esimene on klient ja teised kaks on erinevad tellerid. Kui klient tuleb kontorisse ja soovib luua omale konto, siis esimene teller teeb temaga kõik vajalikud toimingud ning edastab kogu informatsiooni teisele tellerile, kes klienti füüsiliselt nägemata kontrollib omalt poolt andmete korrektsust ning kinnitab konto loomise.

4.1.1 Klient



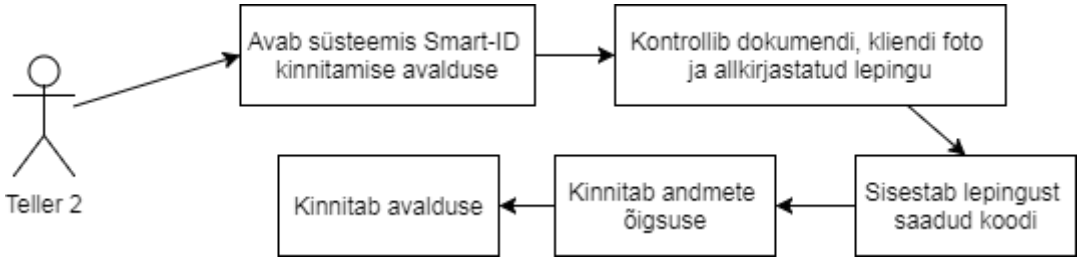
Joonis 2.1 Kliendi kasutusjuhtum

4.1.2 Esimene teller



Joonis 2.2 Esimese telleri kasutusjuhtum

4.1.3 Teine teller



Joonis 2.3 Teise telleri kasutusjuhtum

4.2 Uus UseCase

Uue UseCase'i loomise loogika seisneb failinime määramises ja UseCaseBase'i laiendamises. Antud klass vastutab kliendi olemasolevate Smart-ID kontode kuvamise ja uue konto loomise eest.
Objekt käivitatakse töötaja keskkonnas menüüpunkti avamisega.

Programmi avalik liides:

- **public void init()** – menüüpunkti avamisel käivitatav funktsioon, milles luuakse, initsialiseeritakse ning lisatakse bean'idesse SkPerson ning SkPersonInformation objektid edaspidiseks kasutamiseks.
- **public void showAgreementDetailsPage()** – kuvab SmartIdAgreements.zul ning lisab sellele nupu page.addAction('SmartIdAgreements') ja lisab tabelis olevatele ridadele funktsionaalsust, page.addRowAction("showAgreementDetails").setLabelId("select).
- **public void goBack()** – avab eelmise vaate.
- **public void getSmartIdAgreements()** – käivitab GetSkAgreementList_1.class teenuse ning valideerib vastuse SmartIdAccountsReponse.
- **public void showAgreementDetails()** – avab SmartIdAgreementDetails.zul eelnevalt tabelist valitud konto andmetega.
- **public void newAgreement()** – avab SmartIdAgreementNew.zul vaate ning lisab sellele uue konto loomise nupu page.addAction("createRegistrationProcess").

- **public void createRegistrationProcess()** – täidab konto loomise päringu andmed, loob tühjad vastuse bean'id, käivitab CreateRegistrationProcess_1.class teenuse. Edasi valideerib saadud vastuse, avab uue vaate AddSecondTeller.zul ning lisab sellele nupu page.addAction("SubmitCustomerSignatureConfirmation").
- **public void SubmitCustomerSignatureConfirmation** – käivitab valideerimise protsessi, selle läbides konfigureerib SmartIdRegistrationConfirmationRequest'i, seades selle tüübiks "CLIENT_SIGNATURE" ning käivitab SubmitCustomerSignatureConfirmation_1.class teenuse.
- **public void SubmitSecondaryConfirmation()** - selle läbides konfigureerib SmartIdRegistrationConfirmationRequest'i, seades selle tüübiks "SECONDARY_IDENTIFICATION" ning käivitab SubmitCustomerSecondaryConfirmation_1.class teenuse.
- **public void saveNbvsfBean()** – salvestab teise telleri protsessi jaoks nõutud andmed staatilisse bean'i, et hiljem neid kasutada, kuna iga telleri töö on sessioonipõhine ning tavalised bean'id on samuti sessioonipõhised.

Programmi abimeetodid:

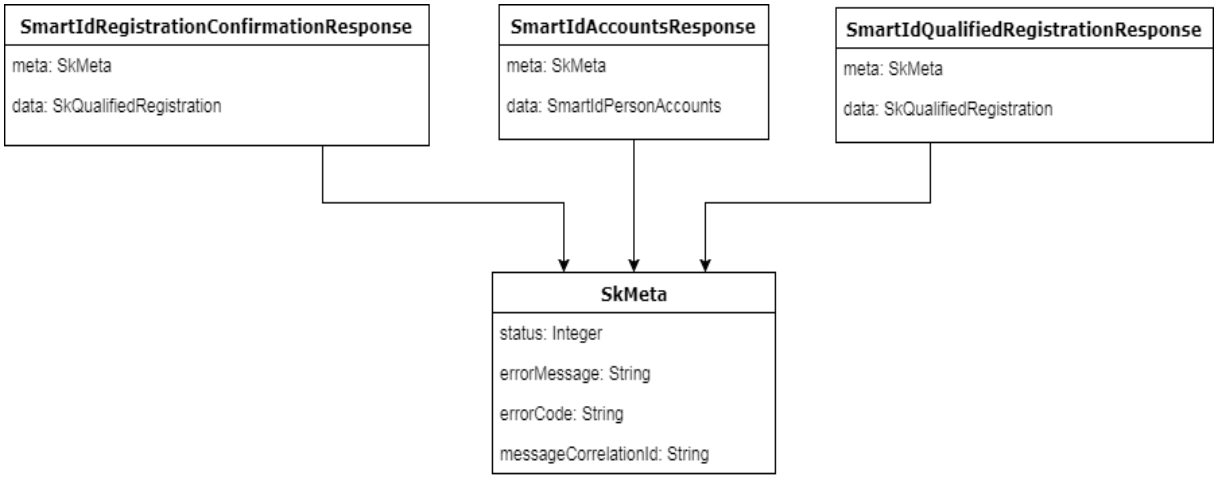
- **private void validateBeforeSave()** – valideerib järgmised punktid:
    - Teller valis teise lepingut kinnitatava poole
    - Teller ei ürita iseennast valida
    - Seisab kliendi allkirja märge
    - Allkirjastatud lepingu koopia on laetud
    - Kliendi dokumendi koopia on laetud
- **private boolean isDifferentTellerSelected()** – abifunktsiooni teise telleri valiku valideerimiseks.

Antud komponendi suurimaks probleemiks oli esimese ja teise Telleri tööprotsessi ühendamine, nimelt keerukus seisnes andmete hoidmises, kuna ühe kasutaja tööprotsessi jooksul on kõik andmed sessioonipõhised. Lahendusena olid kasutatud andmebaasi salvestatavad bean'id ehk staatilised bean'id. Nende abil on võimalik Teise telleri jaoks nõutud andmed saada hiljem kätte. Vastav lahendus oli valitud seoses infosüsteemis olevaoleva funktsionaalsuse taaskasutamisega.

4.3 Päringuteks nõutud objektid

Rakenduse töötamiseks ja Retrofit'i kasutamiseks on vaja luua kindla sisuga objektid andmete hoidmiseks. Iga objekt peab vastama kindla JSON'i päringu või vastuse lõigule, omades muutujaid täpselt sama nimega nagu päringus kirjas.

Vastuse sisu koosneb kahest objektist 'meta' ja 'data', millest 'meta' nimelises objektis on andmed päringu õnnestumise või vea kohta ning 'data' objektis on kõik muud saadetavad andmed seoses kontoga. Meta on kõigi päringute puhul sama, data on aga muutuv. Uute objektide mahu vähendamiseks on mõned objektid tehtud universaalseteks ja taaskasutatavateks. Nendeks on SkQualifiedRegistration ja SmartIdRegistrationConfirmationRequest.



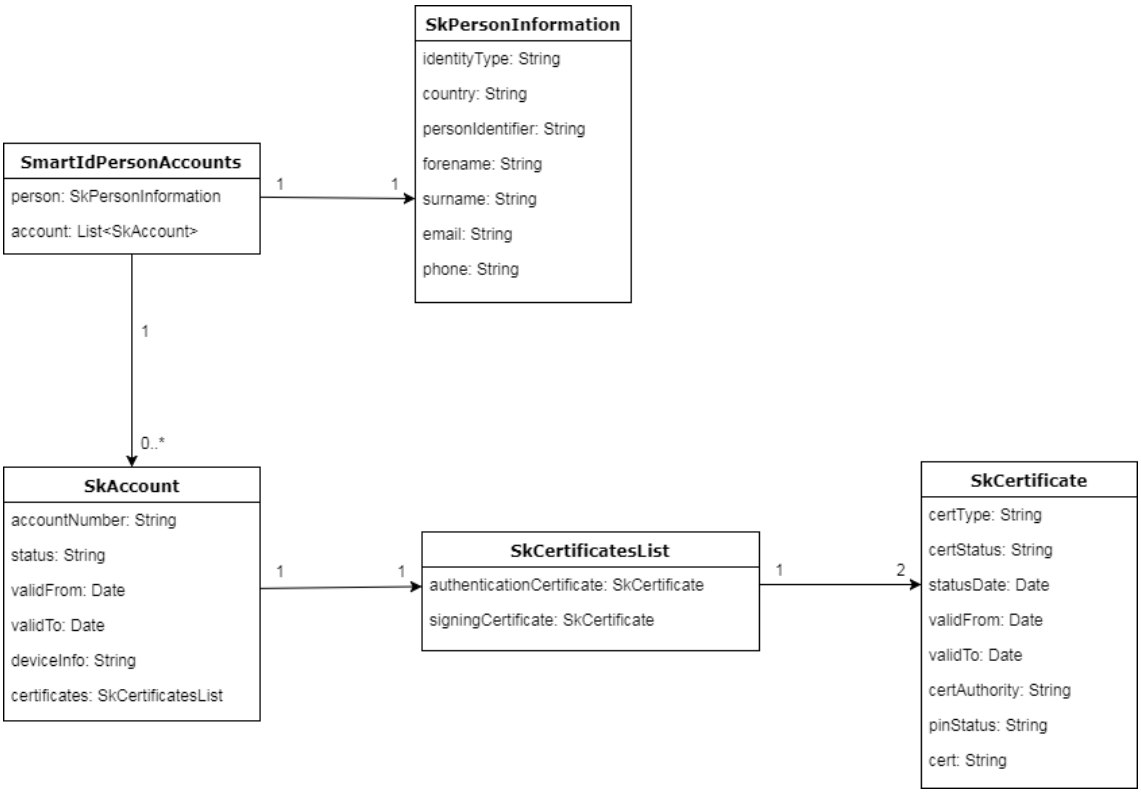
Joonis 3.1 Vaadeldavate vastuste visuaalne kujutus.

SkMeta klass vastutab päringu vastuse korrektsuse informatsiooni eest.

- **status** – päringu õnnestumise koodi, näiteks 200 või 500.
- **errorMessage** – sõnaline vea seletus.
- **errorCode** – sõnaline veakood, näiteks INVALID_REGISTRATION_TOKEN
- **meesageCorrelationId** – kood, mille abiga on võimalik päring kergemini leida logidest.

4.3.1 Kliendi Smart-ID kasutajate päring

Antud päringu puhul ei lähe vaja ühtegi bean'i, sest kõik nõutud parameetrid pandakse URL'i, vaja läheb vaid vastuse konverteerimiseks objekt SmartIdAccountsResponse.java



Joonis 3.2 SmartIdPersonAccounts andmete struktuur

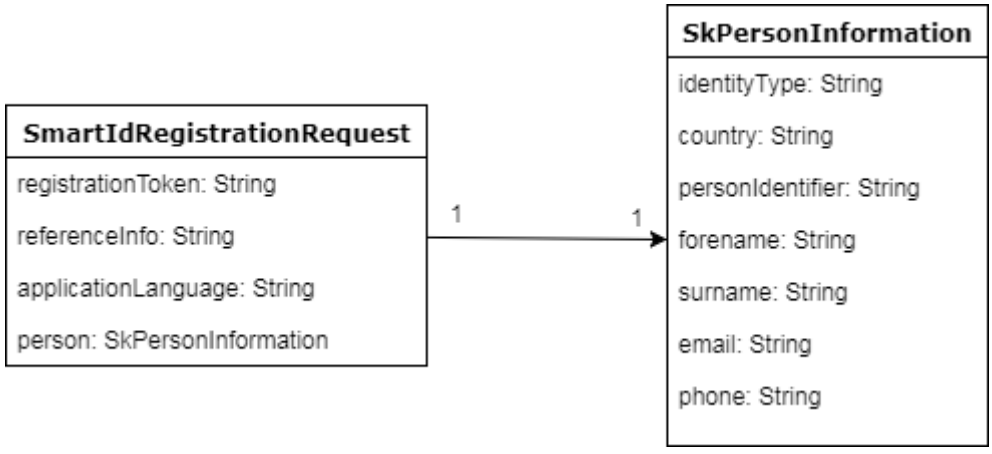
SmartIdPersonAccounts hoiab endas kõik vajalikud andmed, milleks on päritud isiku informatsioon ja nimekiri kõikide tema nime all tehtud kontodega.

SkAccount objekt on ühe Smart-Id konto informatsiooni hoidmiseks.

- **accountNumber** – konto number, mis koosneb konstandist „PNOEE“, isikukoodist ja unikaalsest koodilõpust.
- **status** – konto praegune staatus, näiteks IN_PROCESS, ACTIVE, REVOKED.
- **validFrom** – konto kehtivuse aeg.
- **validTo** – konto hehtivuse lõpp.
- **deviceInfo** – registreeritava telefoni täpne mudel.
- **certificates** – objekt, milles on omakorda kaks sertifikaati, millest üks on autentimiseks ja teine dokumentide allkirjastamiseks.

SkPersonInformation hoiab endas kõik nõutud kliendi andmed Smart-Id väljastamiseks

4.3.2 Smart-ID registratsiooni sessiooni alustamine

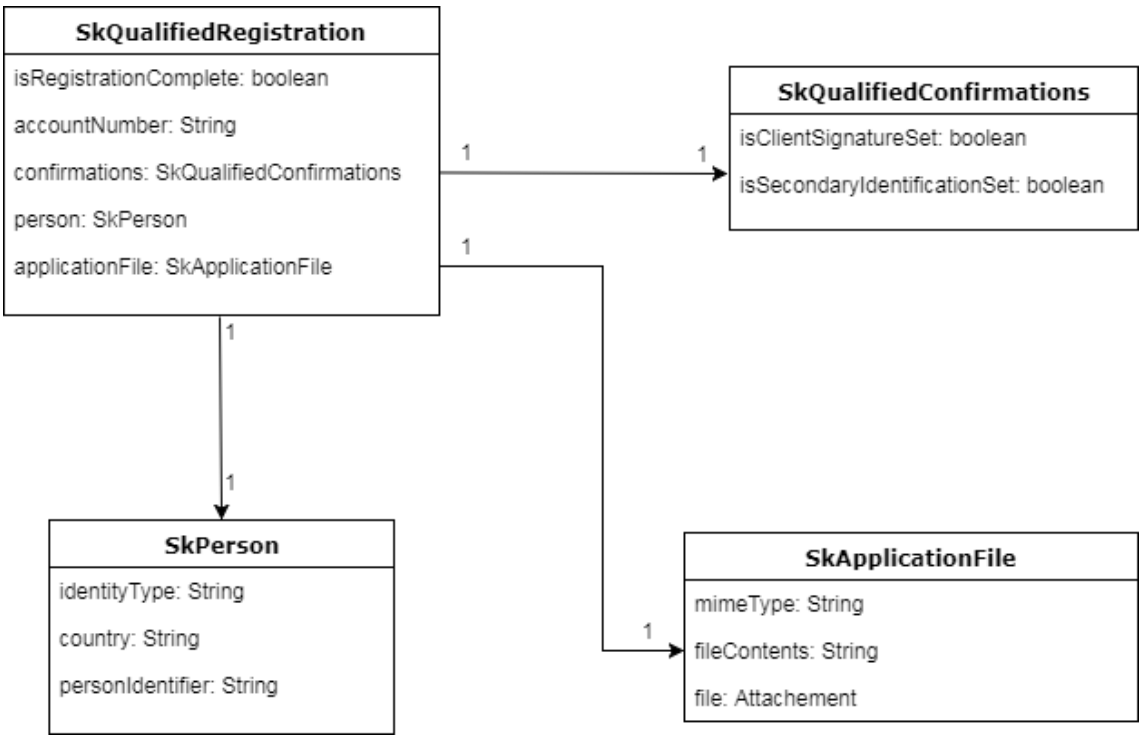


Joonis 3.3 Smart-ID registratsiooni alustamise päringu struktuur

Registratsiooni sessiooni alustamiseks on vaja päringusse kaasa panna SmartIdRegistrationRequest objekt.

Objekti seletus:

- **registrationToken** – kliendi äppist saadud kood.
- **referenceInfo** – koht kommentaari jaoks, mis on hiljem kuvatud lepingus.
- **applicationLanguage** – lepingu keel.
- **person** – kliendi info.

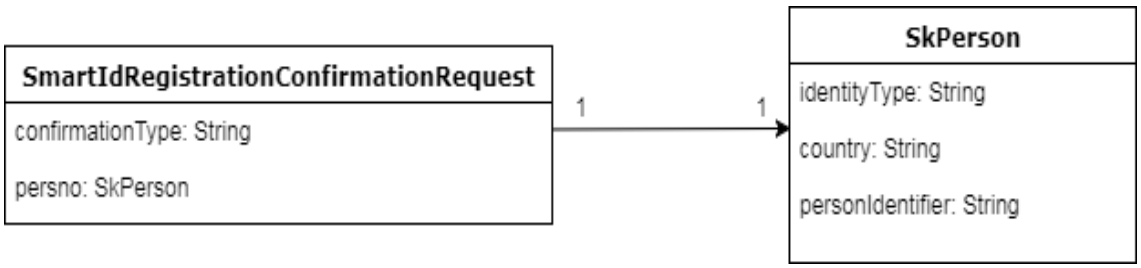


Joonis 3.4 Smart-ID registraatsiooni alustamise vastuse struktuur.

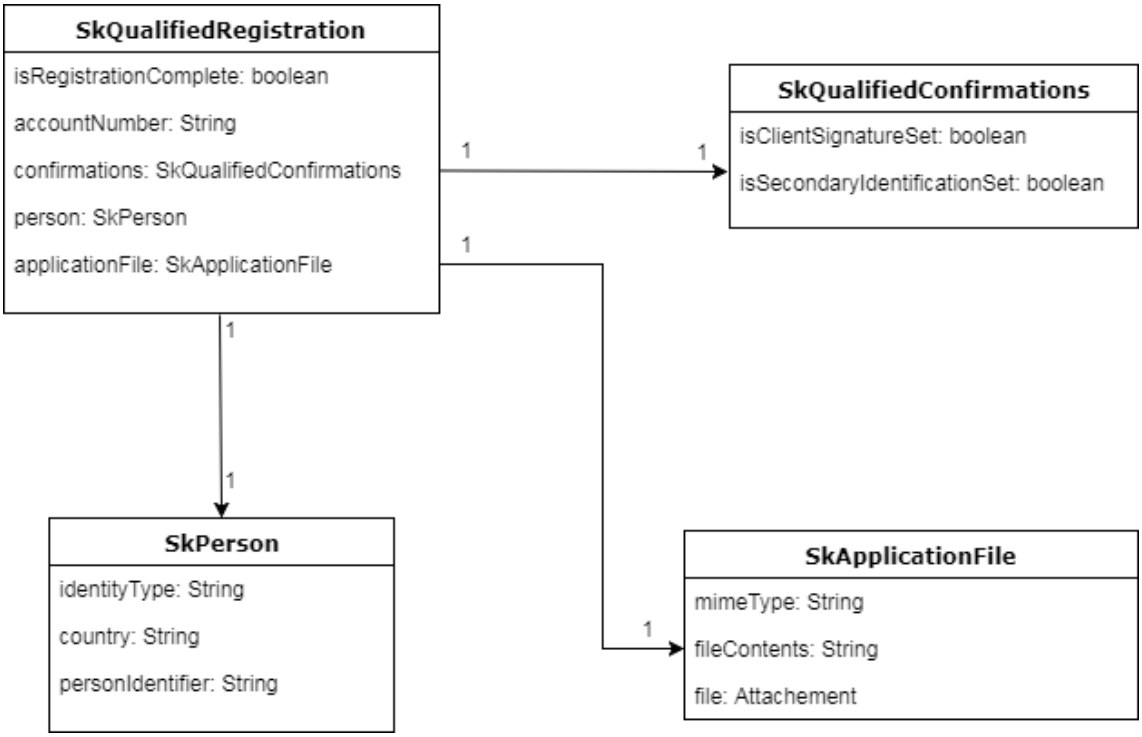
Registratsiooni esimese päringu vastuse peamiseks elemendiks on SkApplicationFile ehk baitkoodis saadetud leping, mille esimene teller peab printima ja laskma kliendil see allkirjastada.

4.3.3 Smart-ID kinnitamine

Uue kasutaja loomiseks on vaja saata kaks erinevat päringut sertifitseerimiskeskusele. Esimene päring saadetakse hetkel, kui klient allkirjastas lepingu ja esimene teller on selle süsteemi ülesse laadinud. Teine päring saadekse siis, kui teine teller on talle saadetud dokumendid kontrollinud ja kinnitanud andmete õigsuse.



Joonis 3.5 Smart-ID kinnitamise päring



Joonis 3.6 Smart-ID kinnituse vastus

Kinnituse päring koosneb vaid kahest osast:

- **confirmationType** – teskiline väli, mille väärtuse järgi määratakse kinnituse tüüp. Võimalikud väärtused ”CLIENT_SIGNATURE” ja ”SECONDARY_IDENTIFICATION”.
- **person** – objekt kliendi andmetega.

Vastuse tüüp on sama, mis esmase Smart-ID registratsiooni alustamise päringu puhul. Erinevus seisneb andmetes, nimelt muutub SkQualifiedConfirmations.java sisu ja SkApplicationFile.java sisu.

Mõlema päringu ärasaatmisel ja nende õnnestumisel tuleb viimane vastus, milles muutub SkQualifiedRegistration.isRegistrationComplete väärtus.

Luues ja töödeldes kõik nõutud objektid päringute ja vastuste jaoks oli kõige keerulisem andmetüüpide sobitamine ja õigeks seadmine protsessi toimiseks. Olemasolevas infosüsteemis võivad samasugusel väljal erineva kliendi puhul olla erinevat tüüpi või erineval kujul salvestatud andmed. Vastav probleem sai lahendatud peamiselt katsetusmeetodi abil, kuna info võimalikete väärtuste kohta infosüsteemi kindlal väljal puudub.

4.4 Nõutud sisemised teenused

SmartIdUseCase.java ja Sk.java suhtlevad omavahel teenuste väljakutsumise kaudu. Selleks on vaja:

1. Soovitud teenusele vastav objekt.
2. Sk.java's peab olema meetod vastava teenuse objekti nimega.

Sk.java klass on ServiceAdaptor ehk vastavalt konfigureeritud klass selles kirjeldatud teenuste vastuvõtmiseks, mis omakorda suunab need edasi SmartIdConnector'i poole avalike meetodite kaudu.

Toon näiteks GetSkAgreementList teenuse

```
@ServiceDescriptor
public class GetSkAgreementList_1 extends ServiceDescriptorBase
{
    @Override
    public Backends getBackend() {
              return Backends.sk;
    }

    @Override
    public Class<?>[] getRequestBeanClasses() {
            return    new    Class<?>[]    {    SkPerson.class,
    OfficerProfile.class };
    }
}
```

- GetSkAgreementList_1 – faili nimi määrab ära teenuse nime, sufiks "_1" määrab teenuse versiooni.
- getBackend() – tagastab kindla ServiceAdaptori suunaja, kuhu pöördub teenus.
- getRequestBeanClasses() – tagastab objektid, mis peavad olema bean'ides olemas teenuse väljakutsumise ajal, muul juhub tekib veateade.

Teenuse väljakutsumine SmartIdUseCase'is näeb välja järgnevalt:

```
this.invokeService(GetSkAgreementList_1.class);
```

Projekti raames on kasutuses 4 erinevat teenust:

- GetSkAgreementList_1
- CreateRegistrationProcess_1
- SubmitCustomerSignatureConfirmation_1
- SubmitSecondaryConfirmation_1

Sk.java failis asuvad teenuseid kirjeldatavad meetodid ning abimeetodid.

- **public void getSkAgreementList()** – käivitab SmartIdConnector'is getSmartIdAccoutByPerson, edastades nõutud parameetrid:
  - identityType - konstant
  - country – kliendi riigikood
  - personIdentifier – kliendi isikukood
  - accountStatus – päritavate kontode staatuse kriteerium
- **public void createRegistrationProcess()** - käivitab SmartIdConnector'is createSmartIdQualifiedRegistrationSession, edastades nõutud SmartIdRegistrationRequest objekt.
- **public void submitCustomerSignatureConfirmation()** - käivitab SmartIdConnector'is setSmartIdRegistrationConfirmation, edastades konto numbri ja nõutud SmartIdRegistrationConfirmationRequest'i objekt.
- **public void submitSecondaryConfirmation()** - lisab sessiooni bean'idesse SmartIdFileSetAddition.java, milles paiknevad teise telleri protsessi jaoks vajatud parameetrid, mis olid loodud või sisestatud esimese telleri poolt. Edasi samuti käivitab SmartIdConnector'is setSmartIdRegistrationConfirmation, edastades konto numbri ja nõutud SmartIdRegistrationConfirmationRequest'i objekt.
- **private void prepareGateway()** – seadistab SmartIdConnector'is kasutatavad telleri andmed, mis esinevad päringute päises.

Sisemiste teenuste realiseerimise keerukus põhines nende arhitektuuri mõistmisel ja suuremas osas mitme objekti edastamise õige kirjelduse leidmisel. Raskust tekitas sarnase olukorraga esmane kokkupuude ja dokumentatsiooni puudumine.

4.5 Smart-ID connector

SmartIdConnector on viimane lüli retrofiti ja smart-id vahel. Selles toimub lõplik päringute valideerimine, korrigeerimine ja käivitamine. *Connector'*is lisaks toimub esmane vastuste kontroll veateadete pihta ja info logimine.

Enne iga päringu saatmist korrigeeritakse aplikatsiooni keele ja kliendi riigi ISO standart vastavalt API poolt nõutud standardile. Nende teisendused hoitakse HashMap'is <String, String> kujul ja muudatuseks kasutatakse changeLanguageStandard ja changeApplicationLanguage meetodeid. Samuti korrigeeritakse kliendi telefoni number.

Klassi avalikud meetodid:

- **public SmartIdAccountsResponse getSmartIdAccountByPerson**
- **publik SmartIdQualifiedRegistrationResponse createSmartIdQualifiedRegistrationSession**
- **public SmartIdRegistrationConfirmationResponse setSmartIdRegistrationConfirmation**
- **public void setOfficerAndBranch**

Klassis olevad abimeetodid:

- **private boolean checkConnection**
- **private Attachment decodeApplicationFile**
- **private String changeLanguageStandard**
- **private String changeApplicationLanguage**

Klassi loomisel valmistas raskust andmete kontroll ja vastavalt olukorrale andmetüüpide ja formaatide teisendused. Lisaks antud klassis toimub kogu veateadete kontroll ja nende loogika seetõttu lisakeerukust valmistas veateadete filtreerimine ja nendest lähtuvalt protsesside edaspidise toimimise loogika arendamine.

Sisu näide:

```
public SmartIdRegistrationConfirmationResponse
setSmartIdRegistrationConfirmation(String accountNumber,

SmartIdRegistrationConfirmationRequest requestData) throws
FileNotFoundException, IOException, ValidationErrors {

requestData.getPerson().setCountry(changeLanguageStandard(requestData.
getPerson().getCountry()));
logger.info(SMARTID_PREFIX + " Starting to set SmartId registration
confirmation for account nr: " + accountNumber);

retrofit2.Response<SmartIdRegistrationConfirmationResponse> request =
gateway.sendQualifiedRegistrationConfirmation(accountNumber,
requestData);

logger.info(SMARTID_PREFIX + " Successfully got response on setting
SmartId registration confirmation for account nr: " + accountNumber);
SmartIdRegistrationConfirmationResponse response = null;
if (checkConnection(request)) {
    response = request.body();
    logger.info(SMARTID_PREFIX + " Response from SK OK!
messageCorrelationId: " +

    response.getMeta().getMessageCorrelationId());
    Attacement attacement =
decodeApplicationFile(response.getData().getApplicationFile().getFileC
ontents());

    response.getData().getApplicationFile().setFile(attacement);
    response.getData().getApplicationFile().setFileContents(null);
}
if (response.getMeta().getStatus() != 200) {
    throw new
ValidationErrors(getBasicErrors(response.getMeta().getErrorMessage()))
;
}
return response;
}
```

4.6 Retrofit’i seadistamine

SmartIdGateway

SmartIdGateway klassis toimub retrofiti loomine, konfigureerimine tööks ning konkreetsete päringute tegemine.

Retrofit’i on klass milles lokaalse API interface’id on muudetud kutsutavateks objektideks [5]. Vaikimisi Retrofit määratab mõistilud seaded, kuid võimaldab ka nende konfigureerimist. Samuti vaikimisi saab Retrofit deserialiseerida ainult HTTP body OkHttp ResponseBody tüübiks ja see aksepteerib vaid RequestBody tüüpi @Body’ks [4]. Muude tüüpide konverteerimiseks on vaja lisada konverter, mis võimaldab kasutada täielikult omi objekte.

Retrofit’i konfigureerimise erinevus vaikimisi seadetest:

- Lisatud GsonBuilder – Konverter ehk objektide deserialiseerija, mis võimaldab kasutada meie poolt loodud andmetüübid päringuteks ja vastusteks.
- GsonBuilder’is muudetud Date type format – Date format seatud ”yyyy-MM-dd'T'HH:mm:ssZ“
- Lisatud HttpLogginInterceptor – Interceptor on vajalik päringutes informatsiooni lisamiseks päisesse. SmartId puhul on vaja lisada telleri kood ja kontori kood.
- ConnectTimeout limiit muudetud – 45000 millisekundit
- ReadTimeout limiit muudetud – 45000 millisekundit

Päringu käivtamiseks on vaja luua uus objekt SmartIdConnectorServiceDescription’ist. Samuti enne igat päringut taaskonfigureeritakse retrofit arvestades vahepealset võimalikku päise muutust.

```
/**
 * First request to start Smart-ID registration process.
 * @param data
 * @return
 * @throws IOException
 */
public        retrofit2.Response<SmartIdQualifiedRegistrationResponse>
startQualifiedRegistration(SmartIdRegistrationRequest   data)   throws
IOException {
    configureGateway();
    SmartIdConnectorServiceDescriptor      smartIdService           =
this.retrofit.create(SmartIdConnectorServiceDescriptor.class);
    return smartIdService.startQualifiedRegistration(data).execute();
}
```

Täielik Gateway seadistus

```
private void configureGateway() {

logger.info(String.format("#smartid-ra Sk request header:  office=%s,
uid=%s", office, uid));
GsonBuilder builder = new GsonBuilder();
builder.registerTypeAdapter(Date.class, new JsonDeserializer<Date>() {
    public Date deserialize(JsonElement json, Type typeOfT,
JsonDeserializationContext context) throws JsonParseException {
        return new Date(json.getAsJsonPrimitive().getAsLong());
    }
});
Gson gson = builder.setDateFormat("yyyy-MM-dd'T'HH:mm:ssZ").create();
Retrofit retrofit;

HttpLoggingInterceptor logging = new
HttpLoggingInterceptor(LoggerNbsf.DEFAULT);
logging.setLevel(HttpLoggingInterceptor.Level.BODY);

final Builder httpClient = new OkHttpClient.Builder();
httpClient.connectTimeout(45000, TimeUnit.MILLISECONDS);
httpClient.readTimeout(45000, TimeUnit.MILLISECONDS);
httpClient.networkInterceptors().add(new Interceptor() {
    @Override
    public Response intercept(Chain chain) throws IOException {
        final Request request = chain.request().newBuilder()
            .header("uid", uid)
             .header("office", office)
            .build();
        return chain.proceed(request);
    }
});
httpClient.addInterceptor(logging);
retrofit = new Retrofit.Builder()
        .baseUrl(baseUrl)
        .addConverterFactory(GsonConverterFactory.create(gson))
        .client(httpClient.build())
        .build();
this.retrofit = retrofit;
}
```

SmartIdConnectorServiceDescriptor

Iga meetod SmartIdConnectorServiceDescriptor'is kirjeldab ühte võimaliku API päringut. Nõutud on HTTP annotatsioon (GET, POST jne) määramaks päringu tüübi ja päringu URL. Query ja Path parameetrid lisatakse vastavalt @Query ja @Path annotatsiooniga vahelduvalt parameetri ees. @Body annotatsioon määrab päringu Body. Smart-ID puhul on selleks eelnevalt valmis loodud objektid.

```
public interface SmartIdConnectorServiceDescriptor {

@POST("/v1/qualifiedRegistrations/{accountNumber}/confirmations/")
public                Call<SmartIdRegistrationConfirmationResponse>
sendQualifiedRegistrationConfirmation(@Path("accountNumber") String
accountNumber, @Body SmartIdRegistrationConfirmationRequest data);

@POST("/v1/qualifiedRegistrations/")
public                   Call<SmartIdQualifiedRegistrationResponse>
startQualifiedRegistration(@Body SmartIdRegistrationRequest data);

@GET("/v1/accounts/")
public                                Call<SmartIdAccountsResponse>
getPersonSmartIdAccounts(@Query("identityType")              String
identityType,       @Query("country")       String       country,
@Query("personIdentifier")        String        personIdentifier,
@Query("accountStatus") String accountStatus);

}
```

Põhiline keerukus seisnes retrofiti seadistuses seoses sertifitseerimiskeskuse nõuetega. Kuna antud raamistik polnud varasemalt kunagi kasutatud, siis tuli seadistused korrigeerida katsemeetodil. Suurim probleem tekkis päringute päise andmete lisamisel, kuna päise konfigureerimine toimub peamiselt protsessi alguses, siis tuli välja mõelda lahendus, kuidas saada töötaja andmed kätte SmartIdGateway klassis, mis on süsteemist kättesaadavad vaid SmartIdUseCase klassis.

4.7 Vaated

Smart-ID integratsiooni kuvamiseks on kasutatud neli erinevat zul vaadet, visualiseerides kõiki ärinmudelis nõutud protsesse.

- SmartIdAgreements.zul – Põhivaade kliendi olemasolevate kontode ülevaatuseks. Kuvatakse kliendi andmed SkPersonInformation bean'ist ja kõik temale kuuluvad kontod SmartIdAccountsResponse bean'ist. Esineb nupp uue konto registratsiooni alustamiseks.
- SmartIdAgreementNew.zul – Vaade uue konto registratsiooni sessiooni alustamiseks. Kuvatud on kliendi andmed SkPersonInformation bean'ist ja sisendväljaks on kliendi äppist saadud kood, mis salvestatakse SmartIdRegistartionRequest'i. Esineb nupp registratiooni sessiooni alustamiseks.
- AddSecondTeller.zul – Uue konto loomise põhivaade telleri jaoks.
  Kuvatud on :
  - kliendi dokumendi pilt, mis on olemas süsteemis
  - nupp klienti autentifitseerinud dokumendi koopia üleslaadimiseks
  - nupp kliendi poolt allkirjastatud lepingu koopia üleslaadimiseks
  - väli teise kinnitaja (telleri) valimiseks
  - väli tellerite grupi valikuks kinnitajateks
  - andmete õigsuse kontrolli kinnitus
  - kliendi allkirja olemasolu kontrolli kinnitus

  Protsessi jätkamiseks peavad olema üleslaetud dokumendi ja lepingu koopiad, valitud teine kinnitaja või grupp ning märgitud kaks kinnitust.

- SmartIdFilesetDetails.zul – Teise kinnituse vaade protsessi lõpetamiseks.
  Kuvatud on:
  - kliendi pilt
  - eelnevalt üleslaetud dokumendi koopia
  - allkirjastatud lepingu koopia
  - andmed seoses digidoc'i allkirjastatega (antud osa ei vaadelda lõputöö raames)
  - kontrollkoodi sisestamise väli
  - allkirjastamise / kinnitamise nupp

Vaadete loomise tegi keeruliseks protsessi äriloogika detailide täideviimine ja uue raamistiku ja ning programmeerimiskeele õppimine. Lisaks tuli integreerida olemasolevad komponendid nagu töötaja valik, mille info tüleb infosüsteemi siseteenusest.

SmartIdAgreementNew.zul

```
<zk>
    <form beanName="SkPersonInformation">
        <fields>
            <field fieldName="country" readonly="true" />
            <field fieldName="personIdentifier" readonly="true" />
            <field fieldName="forename" readonly="true" />
            <field fieldName="surname" readonly="true" />
            <field fieldName="email" readonly="true" />
            <field fieldName="phone" readonly="true" />
        </fields>
    </form>
    <form beanName="SmartIdRegistrationRequest">
        <fields>
            <field fieldName="registrationToken" />
        </fields>
    </form>
</zk>
```

5 Kokkuvõte

Lõputöö eesmärgiks oli integreerida finantsteenuspakkuja infosüsteemi Smart-Id API ning luua sellele kasutajaliides. Töö raames oli vaja realiseerida põhifunktsionaalsus ehk kliendile uue konto loomise võimalus.

Bakalaureusetöös valmis MVC baasil rakendus, mis on kirjutatud Java programmeerimiskeeles. Rakendus haldab kasutajale kuvatavat visuaalset pilti, mis on kirjeldatud ZK raamistiku abil kasutades ZUL programmeerimiskeelt. Lisaks rakendus suhtleb välisteenuspakkujaga REST päringute teel, mis on realiseeritud kasutades Retrofit raamistikku.

Protsessi käigus toimub andmete hoidmine Spring'i bean'ides. Eelnimetatud moodus hoiab andmed sessioonipõhiselt ehk hetk, kui esimene teller lõpetab oma töö, kõik andmed kaovad. Esimeses protsessis kasutatud andmed, mis on nõutud ka teises protsessis salvestatakse BeanStorage'isse ehk eraldi andmebaasi.

Põhiliseks takistuseks oli autori esialgne teadmatus infosüsteemi arhitektuurist ja muudest selles toimuvatest protsessidest. Lisakeerukust valmistasid andmete dünaamilisest hoidmistest tingitud muutused.

Töö eesmärk sai täidetud. Valmis toimiv olemasoleva rakenduse liides, mis võimaldab telleritel alustada Smart-Id kontode väljastamisega klientidele.

6 Kasutatud kirjandus

[1] Pivotal Software, Inc., „Spring Boot,“ [Võrgumaterjal]. Available: http://projects.spring.io/spring-boot/. [Kasutatud 19 November 2017].

[2] Oracle Corporation, „Java,“ [Võrgumaterjal]. Available: https://java.com/en/. [Kasutatud 19 November 2017].

[3] SK ID Solutions, ”Smart-ID” [Võrgumaterjal]. Available: https://www.smart-id.com/et/. [Kasutatud 19 November 2017]

[4] Square, Inc. ”Retrofit” [Võrgumaterjal]. Available: http://square.github.io/retrofit/ [Kasutatud 19 November 2017]

[5] Vogella GmbH, ”Retrofit” [Võrgumaterjal]. Available: http://www.vogella.com/tutorials/Retrofit/article.html [Kasutatud 19 November 2017]

[6] Goolge, Inc. ”Gson” [Võrgumaterjal]. Available: https://github.com/google/gson. [Kasutatud 19 November 2017]

[7] Potics Coproration. ”ZK’s Value and Strength” [Võrgumaterjal]. Available: http://books.zkoss.org/zkessentials-book/master/. [Kasutatud 19 November 2017]

[8] ”Spring - Bean Definition” [Võrgumaterjal]. Available: https://www.tutorialspoint.com/spring/spring_bean_definition.htm [Kasutatud 19 November 2017]

[9] ”MVC Architecture” [Võrgumaterjal]. Available: http://www.tutorialsteacher.com/mvc/mvc-architecture [Kasutatud 19 November 2017]

[10] SK ID Solutions ”Isikutuvastus ID-kaardi ja Mobiil-ID’ga” [Võrgumaterjal]. Available: https://www.id.ee/?lang=et [Kasutatud 19 November 2017]

[11] Pivotal Software, Inc., „Spring Framework“ [Võrgumaterjal]. Available: http://spring.io/ [Kasutatud 19 November 2017].

[12] Pivotal Software, Inc., „Understanding REST“ [Võrgumaterjal]. Available: https://spring.io/understanding/REST [Kasutatud 19 November 2017].

[13] W3C, „Extensible Markup Language“ [Võrgumaterjal]. Available: https://www.w3.org/XML/ [Kasutatud 19 November 2017].

[14] Potics Coproration. ”ZUML” [Võrgumaterjal]. Available: https://www.zkoss.org/wiki/ZUML%20Reference/ZUML/ . [Kasutatud 19 November 2017]

[15] Chris Richardson, ”Pattern: Microservice architecture” [Võrgumaterjal]. Available: http://microservices.io/patterns/microservices.html . [Kasutatud 19 November 2017]

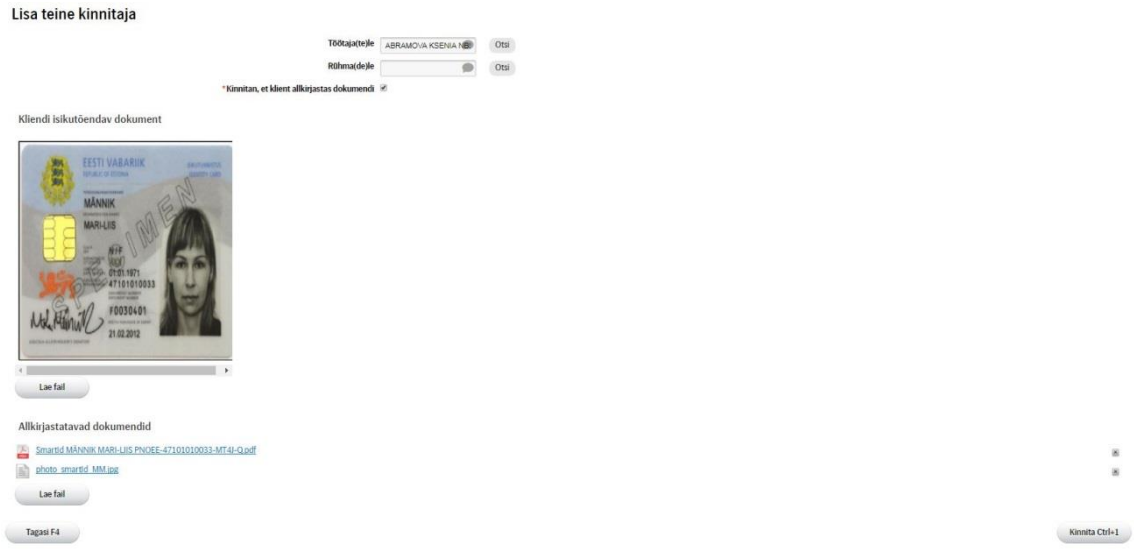
7 Lisa 1 – Kliendi Smart-ID kontode vaade

8 Lisa 2 – Uue konto loomise alustamise vaade

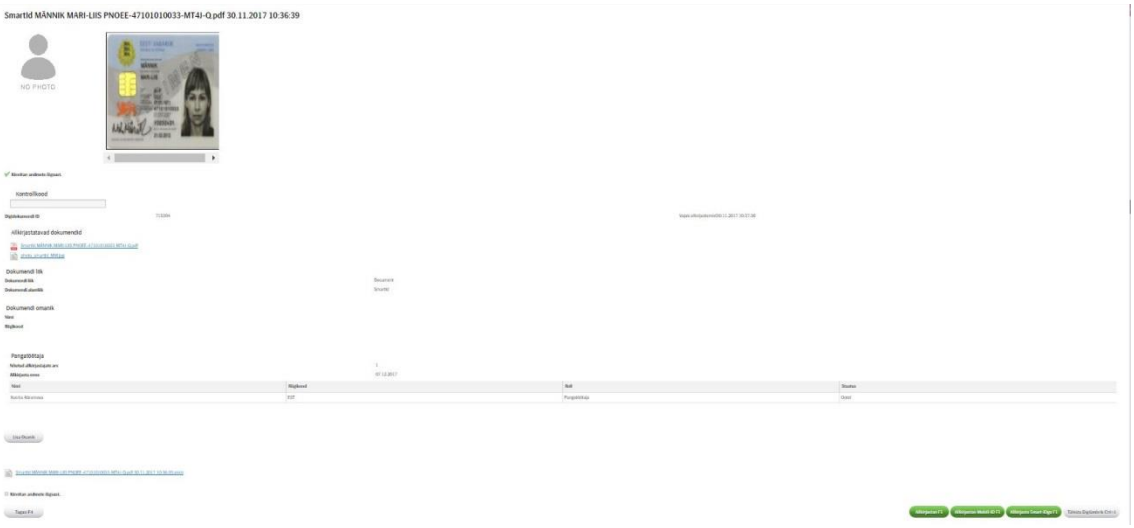
9 Lisa 3 – Uue konto loomise alustamise täidetud vaade

10 Lisa 4 – Teise osapoole valimise ja andmete lisamise vaade

11 Lisa 5 – Teise osapoole valimise ja andmete lisamise täidetud vaade



12 Lisa 6 – Teise telleri vaade



13 Lisa 7 – Teise telleri vaade peale allkirjastamist

14 Lisa 8 - SmartIdAgreements.zul

```
<zk xmlns:w="http://www.zkoss.org/2005/zk/client">
<zk if="${systemConfig.envConfig.country == 'LT'}">
    <form beanName="SmartIdCrmUrlConfig">
        <a href="" label="${trMessages.linkToLTCRM}"
w:onClick="gotoCrm('${defaultBean.privateCustomerLink}')" />
    </form>
    <form beanName="SkPersonInformation">
    <fields>
        <field fieldName="country" readonly="false"
rddFieldName="CountryCode"/>
        <field fieldName="personIdentifier" readonly="false" />
        <field fieldName="forename" readonly="false" />
        <field fieldName="surname" readonly="false" />
        <field fieldName="email" readonly="false" />
        <field fieldName="phone" readonly="false" />
    </fields>
    </form>
</zk>
<zk if="${systemConfig.envConfig.country != 'LT'}">
    <form beanName="SkPerson">
    <fields>
        <field fieldName="country" readonly="true" />
        <field fieldName="personIdentifier" readonly="true" />
    </fields>
    </form>
</zk>

<orilistbox beanName="SmartIdAccountsResponse.data.account">
    <oricolumn fieldName="accountNumber" />
    <oricolumn fieldName="status" />
    <oricolumn fieldName="validFrom" />
    <oricolumn fieldName="validTo" />
    <oricolumn fieldName="deviceInfo" />
</orilistbox>
<form beanName="SkTestBean">
    <fields>
        <field fieldName="text" readonly="true" />
    </fields>
</form>
</zk>
```

15 Lisa 9 - SmartIdAgreementNew.zul

```
<zk>
<form beanName="SkPersonInformation">
    <fields>
        <field fieldName="country" readonly="true" />
        <field fieldName="personIdentifier" readonly="true" />
        <field fieldName="forename" readonly="true" />
        <field fieldName="surname" readonly="true" />
        <field fieldName="email" readonly="true" />
        <field fieldName="phone" readonly="true" />
    </fields>
</form>
<form beanName="SmartIdRegistrationRequest">
    <fields>
        <field fieldName="registrationToken" />
    </fields>
</form>
</zk>
```

16 Lisa 10 - AddSecondTeller.zul

```
<zk xmlns:w="http://www.zkoss.org/2005/zk/client">
<zk if="${systemConfig.envConfig.country == 'LT'}">
<form beanName="NewCounterparty">
    <field fieldName="officerId" />
</form>
</zk>

<form beanName="CustomerSignatureConfirmation">
<zk if="${systemConfig.envConfig.country != 'LT'}">
    <field fieldName="crmMessageTo" styleClass="long" />
    <field fieldName="crmMessageToGroup" styleClass="long" />
</zk>
<zk if="${systemConfig.envConfig.country == 'LT'}">
    <field fieldName="sendMessageToGroup" />
</zk>
    <field fieldName="customerSigned"
styleClass="termsAndConditionsAcception" />
</form>

<groupbox sclass="transparent">
    <label sclass="zulHeaderSmall" value="${trMessages.photoPdf}" />
    <spacer class="vSpacer" />
```

```
    <div style="width:350px;float:left;">
    <orilistbox beanName="FileSet.files.categories.NORMAL">
           <oritableparameters lookLikeLinks="true"
showNoDataMessage="true" />
           <oricolumn fieldName="file_name_include1"
include="SmartIdFileSetDetailsInclude" />
    </orilistbox>

    <oributton label="${trMessages.Upload}" upload="true"
uploadEventListener="smartIdFbiFileUploadEventListener" />
    </div>
</groupbox>
<groupbox sclass="transparent">
    <label sclass="zulHeaderSmall" value="${trMessages.normalFiles}"
/>
    <spacer class="vSpacer" />
    <orilistbox beanName="FileSet.files.categories.NORMAL">
    <oritableparameters actionsGroup="fileSet" lookLikeLinks="true"
showNoDataMessage="true" />
    <oricolumn fieldName="file_name_include1"
include="FileSetDetailsInclude" />
    </orilistbox>
    <oributton label="${trMessages.Upload}" upload="true"
uploadEventListener="fbiFileUploadEventListener" />
</groupbox>
</zk>
```

17 Lisa 11 - SmartIdFilesetDetails.zul

```
<zk>
<div>
    <div style="float:left;">
    <form beanName="CustomerBasicData">
        <div style="width:300px; float:left;">
        <if visibleField="@{'not empty defaultBean.photo.file_body'}">
        <fields>
            <oriimage tooltip="popupRepresentative, delay=100"
fieldName="photo" style="width:300px;"
defaultImage="/images/noPhoto.gif" fieldTemplateEnabled="LABELEMPTY"
/>
        </fields>
        <popup id="popupRepresentative" zclass="zulPopup"
sclass="zulPopupPhoto">
            <oriimage fieldName="photo"
defaultImage="/images/noPhoto.gif" emptyLabel="true" />
        </popup>
        </if>
        <if visibleField="@{'empty defaultBean.photo.file_body'}">
            <image style="width:200px;" src="/images/noPhoto.gif" />
        </if>
```

```
    </div>
    <div style="width:350px;float:left;">
    <orilistbox beanName="FileSet.files.categories.NORMAL">
    <oritableparameters actionsGroup="fileSet" lookLikeLinks="true"
showNoDataMessage="true" />
        <oricolumn fieldName="file_name_include1"
include="SmartIdFileSetDetailsInclude" />
    </orilistbox>
    </div>
    </form>
    </div>
</div>
<div style="width:100%;float:left;">
<span class="vSpacer" />
<span class="vSpacer" />
<span class="zulTermsAndConditionsBlock">
<form beanName="FbiCounterparty" styleClass="dummyStyleClass">
    <fields width="auto">
    <field styleClass="termsAndConditionsAcception"
fieldName="termsAndConditionsAccepted"
fieldTemplateEnabled="LABELBEHIND"
editableExpr="${sessionData.session.channelTab}" />
    </fields>
</form>
    <div class="termsAndConditionsAcceptionQuestion"
if="${oriDataMgr.FileSet.confirmationQuestionTranslated!='#EMPTY'}">
    <html
content="@{oriDataMgr.FileSet.confirmationQuestionTranslated}" />
    </div>
</span>
<form beanName="SmartIdFileSetAddition" >
    <fields>
        <field fieldName="confirmationCode" editableExpr="true"
styleClass="long" fieldTemplateEnabled="BORDERTRANSPARENT" />
    </fields>
</form>
</div>

<include src="zul/common/FileSetDetails.zul" />
```

```
<div style="width:100%;float:left;">
<span class="vSpacer" />
<span class="zulTermsAndConditionsBlock">
<form beanName="SmartIDTerms" styleClass="dummyStyleClass">
    <fields width="auto">
        <field fieldName="termsAndConditionsAccepted"
fieldTemplateEnabled="LABELBEHIND" editableExpr="true" />
        <div class="termsAndConditionsAcceptionQuestion"
if="${oriDataMgr.FileSet.confirmationQuestionTranslated!='#EMPTY'}">
            <html
content="@{oriDataMgr.FileSet.confirmationQuestionTranslated}" />
        </div>
    </fields>
</form>
</span>
<span class="vSpacer" />
</div>
</zk>
```