TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Marko Jõgi 185829IABB
Henri Hummal 185732IABB

VEEBIRAKENDUS TALLINNA BÖRSIETTEVÕTETE FINANTSANDMETE KAJASTAMISEKS: 3. ITERATSIOON

Bakalaureusetöö

Juhendaja: Tõnn Talpsepp
PhD

Tallinn 2021

Autorideklaratsioon

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

Autorid: Marko Jõgi, Henri Hummal

18.05.2021

Tallinn 2021

Annotatsioon

Tallinna börsil puudub hetkel ühine platvorm, kus investoritele leiduks kogu vajalik ettevõtetega seotud majanduslik informatsioon investeerimisotsuste langetamiseks. Taolise platvormi loomisega on juba alustatud ja käesoleva töö eesmärgiks on panustada selle platvormi edasisse arengusse.

Käesolev lõpuprojekt käsitleb olulise osa juurdearendamist, kus lisatakse investeerimisotsuste langetamisel kriitilise tähtsusega ajalooliste andmete salvestamine. Töö jätkab Marko Jagori ja Katre-Helena Käppa tööde edasiarendamist. Marko Jagor arendas välja rakenduse fundamentaalarhitektuuri ja kasutajaliidese [1], Katre-Helena Käppa lisas väärtpaberite hindade reaalajas uuendamise, ettevõtetega seotud mõnede finantsnäitaja arvutamise, kasutajakontode ja ettevõtete võrdlemise funktsionaalsused [2]. Meie eesmärk on välja töötada korrektse börsiettevõtete andmete importimise; nende standardsele kujule viimine, et hiljem arvutatud finantsnäitajad oleksid erinevate ettevõtete vahel võrreldavad; võimalusel relevantsete finantsnäitajate genereerimise ja andmebaasis talletamise funktsionaalsused.

Kuna peale meid jätkab rakenduse edasiarendusega tõenäoliselt veel hulk lõputöö kirjutajaid, siis otsustasime töö nime standardiseerida, et lugejatel oleks selle finantsrakenduse arenduse dokumentatsiooni lihtsam kronoloogiliselt järjestada. Meie töö on Marko Jagori ja Katre-Helena Käppa järel kolmas ning meie tööst järgmine neljas iteratsioon.

Lõpuprojekt on kirjutatud eesti keeles ning sisaldab teksti 63 leheküljel, 5 peatükki, 37 joonist.

Abstract

Web Application for Presenting Financial Data of Companies Listed on Tallinn Stock Exchange: Iteration 3

Currently, the Tallinn Stock Exchange lacks a uniform platform, where all the relevant financial information about its partaking companies would be gathered and presented. The aim of this work is to continue the development of such an application. The authors believe that it would greatly improve the situational awareness and decision-making capabilities of investors.

The authors continue the work of previous contributors Marko Jagor [1] and Katre-Helena Käppa [2], adding support for importing real companies' data, if possible, generating relevant financial indicators, and storing the results in a real database.

The authors believe that other contributors will continue work on this application and have decided to standardize the name of this paper after Jagor's first report so that readers would more easily be able to chronologically order all papers in the series.

The thesis is written in Estonian and contains 63 pages of text, 5 chapters, and 37 figures.

Lühendite ja mõistete sõnastik

Agile development	*Agile development* viitab iteratiivsel arendamisel põhinevale tarkvaraarenduse metoodikate rühmale, kus nõuded ja lahendused arenevad isekorralduvate ristfunktsionaalsete meeskondade koostöö kaudu.
CSV	Comma seperated value fail.
DOM	*Document Object Model* – Objekt, kuhu kõikide komponentide genereeritud HTML saadetakse ja mille põhjal brauser kasutajale lõpliku vaate kuvab.
Endpoint	RESTful API-t rakendav rakendus määrab ühe või mitu URL-I lõpp-punkti (endpointi), millele vastab URL'i teenindavas kontrolleris meetod, mis teenindab päringut.
Feature	Feature on tarkvarasüsteemi funktsionaalsuse ühik, mis vastab nõudele.
Infra	Infrastruktuuri kiht.
Issue	*Issue'*d on suurepärane võimalus jälgida oma projektide ülesandeid, täiustusi ja vigu. Neid saab jagada ja arutada ülejäänud meeskonnaga.
Kasutajalugu	Kasutajalugu (Inglise keeles *user story*) on inimestele arusaadavalt kirja pandud süsteeminõue, mis kirjeldab mõnda infosüsteemi funktsionaalsust, mida mõni isik (näiteks lõppkasutaja) peab teha saama.
Komponent	Komponendid (Reacti kontekstis) on sisuliselt ehitusklotsid, millest koosneb terve Reacti klientrakendus [3]. Komponendid võimaldavad kasutajaliidese igat elementi vaadelda kui iseseisvat objekti, mida saab taaskasutada klientrakenduse erinevates vaadetes. Komponent võib omakorda koosneda komponentidest. Näitena võib välja tuua andmetabeli suurema komponendina. Tabel koosneb päistest, andmetest, nuppudest ja paljudest muudest elementidest, mis võivad omakorda olla väiksemad komponendid.
Milestone	*Milestone'*d on probleemide rühmad, mis vastavad projektile, funktsioonile või ajaperioodile. Inimesed kasutavad neid tarkvaraarenduses mitmel erineval viisil.
Mock	Andmestik, mida kasutatakse ajutiselt süsteemis selleks, et töödata ilma reaalsete andmeteta.

Olek	*React*'i komponendid hoiavad endas olekut (inglise keeles *state*), kuhu pannakse kõik selle komponendi jaoks vajalikud muutujad, mis võivad komponendi kasutamise käigus oma väärtust muuta. Need muutujad on kriitilise tähtsusega, sest võimaldavad näidata komponendis dünaamilist sisu.
Production	Tootmiskeskkond on seade, kuhu installitakse tarkvara uusim tööversioon ja tehakse see lõppkasutajatele kättesaadavaks.
Screener	Kraape tööriist, mis kogub infot väliselt veebilehelt.
SCRUM	*Scrum* on agiilne raamistik keerukate toodete väljatöötamiseks, tarnimiseks ja hooldamiseks, rõhuasetusega esialgu tarkvara arendamisel
SpEL	Spring Expression Language (lühidalt SpEL) on võimas väljendikeel, mis toetab andmete pärimist ja manipuleerimist programmi jooksutamise ajal
Sprint	*Sprindid* on kindla pikkusega sündmused, mille kestvus on üks kuu või vähem. Uus Sprint algab kohe pärast eelmise Sprindi sõlmimist.
Sprint boot	Spring Framework on Java platvormi rakenduste raamistik, mis võimaldab lihtsustada projekti üles seadmist ja edaspidist arendust.
Stand-Up	*Stand-up* on igapäevane koosolek, kuhu on kaasatud põhimeeskond
Tagarakendus (back-end) ja klientrakendus (front-end)	Projekt tervikuna koosneb tagarakendusest, kus toimuvad päringute tõlgendamised ja nende põhjal andmete töötlemine, suhtlus andmebaasiga ja palju muud. Klientrakendus kuvatakse kasutaja monitoril ja kasutaja saab seal visuaalselt näha tagarakenduse poolt saadud andmeid, vajutada nuppe, täita välju ja liikuda rakenduse erinevate võimaluste vahel ringi.
Teek	Teek (inglise keeles library) on funktsioonide, makrode, klasside, moodulite vms komponentide kogu.
UI	*User Interface*, kasutajaliides
Vaade	Vaade on see, mida programmi kasutaja lõpuks monitoril näeb.
Waterfall	*Waterfall* mudelis peavad kõik faasid olema lõpule viidud enne järgmise etapi algust ja faasides ei ole kattumist.
Web API	Web API ehk veebiliides on rakenduse programmeerimisliides kas veebiserveri või veebilehitseja jaoks.

Sisukord

Jooniste loetelu

Tabelite loetelu

1. Sissejuhatus

Käesolev peatükk hõlmab endas probleemi ja projekti eesmärgi lahti seletamist, olemasoleva ja planeeritava funktsionaalsuse lühiülevaadet ning töö struktuuri.

Kirjutatud bakalaureusetöö käsitleb Tallinna börsil viibivate ettevõtete finantsandmete veebirakenduse arendust, toetudes REST-rakenduse põhimõtetele. Rakenduse esimese iteratsiooni autor on Marko Jagor, kes tegi selle oma lõputöö raames. Hiljem lisati uue lõputöö raames Katre-Helena Käppa poolt uut funktsionaalsust ning nüüd, kolmanda iteratsioonina, on veebirakendus jõudnud käesoleva töö autorite kätte. Lõpuprojektina jätkatakse rakenduse edasiarendust, arvestades vajadusel eelnevate autorite soovitustega.

1.1 Probleem ja projekti eesmärk

Väärtpaberi turgudel tegutsev investor kasutab investeerimisotsuste langetamisel mitmeid tööriistu, konkreetsete ettevõtete finantskordajaid ning aruandeid. Eelnevates iteratsioonides on välja toodud, et reaalajas pakutavad olulisemate finantsnäitajate kohta infot andvad kraape-programmid ehk *screener*'id on enamasti suunatud välismaistele börsidele ning Balti börsidele ei leidu praegu analoogi, mis oligi ajendiks taolise tööriista loomiseks. Samaaegselt nenditi asjaolu, et pikema perspektiiviga investeeringute suhtes otsuste langetamiseks vajataks ettevõtte sügavamaks analüüsiks ka finantsaruandeid, bilansse, kasumiaruandeid ning muid dokumente: Käppa töös väideti, et „Tallinna börsil tegutseval investoril puudub hetkel mugav veebirakendus, mis võimaldaks kuvada kasutajale aktsiate andmeid reaalajas ning samas pakuks lihtsat võimalust ettevõtte ajaloolise tervikpildi analüüsiks." [2]

Käesoleva töö, investeerimisrakenduse kolmanda iteratsiooni eesmärgiks on luua firmadele aruannete lisamise tehniline võimekus finantsteadmistega administraatori poolt, mille järel saaks näha börsiettevõtete ajaloolisi andmeid. Lisatud aruannetest omakorda tuletatakse standardiseeritud aruanded, millega saab iga investor tutvuda.

Arendatud funktsionaalsus viib projekti tervikuna lähemale üldeesmärgile: kogu vajalikku informatsiooni pakkuv platvorm, millele tuginedes saaksid investorid langetada läbimõeldud investeerimisotsuseid. Visiooni kohaselt pakub rakendus börsil kaubeldavate ettevõtete andmeid nii rakenduses vaadatavana kui ka CSV-kujul allalaetavana.

1.2 Funktsionaalsus

Rakenduse eelmises versioonis loodi rakendusele väärtpaberite reaalajas hinna saamise funktsionaalsus. Andmeid uuendatakse teatud intervalli tagant ja loodi mõne olulisema finantssuhtarvu automaatne arvutamine ning kuvamine. Samuti lisati kasutaja loomise võimalus, millega võimaldati lisada jälgimisnimekirja Tallinna börsi aktsiaid ning teha börsifirmade finantsandmete üks-ühele võrdlust.

Antud lõpuprojekti raames otsustasid autorid jätkata tööd, kasutades Java programmeerimiskeelt ning Spring Boot raamistikku tagarakenduses ja JavaScript React teeki klientrakenduses. Andmete kraapimise lahendust ega sõnumivahetust kraapija ning tagarakenduse vahel ei muudetud.

Võrreldes eelneva lõputööga, on loodud juhend RabbitMQ kasutamiseks tulevastele arendajatele, et nad ei peaks selle seadistamisega palju aega raiskama. Andmebaasi valikuks jäi eelnevalt kasutusele võetud PostgreSQL, kuhu on juurde lisatud hulganisti andmetabeleid, et kogu uut ajalooliste andmete lisamise ja töötlemise funktsionaalsust toetada. Projekti arenduses jätkati puhta koodi põhimõtete kasutamist eesmärgiga hoida koodi kvaliteeti ja hallatavust kõrgel.

1.3 Töö struktuur

Eesseisvad peatükid heidavad pilgu olemasolevatele lahendustele, kolmandas iteratsioonis kasutatud tehnoloogiatele, arhitektuurile, disainile, kirjutatud koodile, tööprotsessile, kasutajalugudele ning töö tulemustele. Töö lõpeb analüüsiga ning edasiarenduste ettepanekutega.

Töö tulemuste all leiab projekti tehnilise dokumentatsiooni. Analüüsi ja järelduste peatükis analüüsitakse töö tulemusi ja antakse hinnang projekti teostuse ja protsessi kohta. Projekti viimases faasis tuuakse välja ka iga individuaalse meeskonnaliikme isiklik panus ning eneseanalüüs.

2. Projekti ülevaade

Selleks, et täita hea koodi dokumentatsiooni nõuet, otsustati säilitada sarnane töö struktuur, mis eelmistelgi iteratsioonidel. Sellisel juhul on tulevastel arendajatel lihtsam võrrelda töid ning näha, kuhu on jõutud etapp-etapi haaval. Projekti ülevaate peatükk katab endas lõpuprojekti nõutud "Metoodika" peatükki.

2.1 Objekti detailne kirjeldus

Järjekordne projekti edasiarendus on suunatud uue funktsionaalsuse juurde arendamisele, tänu millele viiakse sisse esimesed reaalsed andmed administraatori lisatud ettevõtte raamatupidamisaruannete põhjal. Senimaani oli kasutatud rohkesti *mock*-andmeid, millega ei olnud midagi peale hakata. Idee tekkis juhendaja ja tudengite koostöös, kes soovisid ühendada enda head majanduse ja tarkvaraarenduse oskused ning luua praktiline töö, kus saaks midagi uut luua. Valmis produkt on erinevate tudengite tööde põhjal valminud investeerimisrakendus, mis on mõeldud Balti börsi turule.

Projekt käsitleb peamiselt vaid reaalsete andmete importimist ja kasutuselevõttu, kuid see ei ole kindlasti kõik, mida annab selle rakenduse puhul realiseerida. Edaspidi järgneb võimalusel veel palju tööd, kust sisendit saab nii eelmistest kui ka praeguse arenduse iteratsioonist. Sisuliselt plaanitakse valmis saada tööriist, mille läbi finantsteadmistega administraator saab sisestada raamatupidamisaruandeid läbi firmalt kogutud csv failide. Lisaks eelnevale saab sisestatud aruannete põhjal luua standardiseeritud aruannete grupid, mis tehakse järgmiste rakenduse iteratsioonidega kättesaadavaks investoritele.

2.2 Olemasolevad lahendused

Selles alapeatükis antakse ülevaade eelmiste tudengite tööst ning tuuakse välja tööriistad, mida kasutatakse ajalooliste andmete kättesaamiseks.

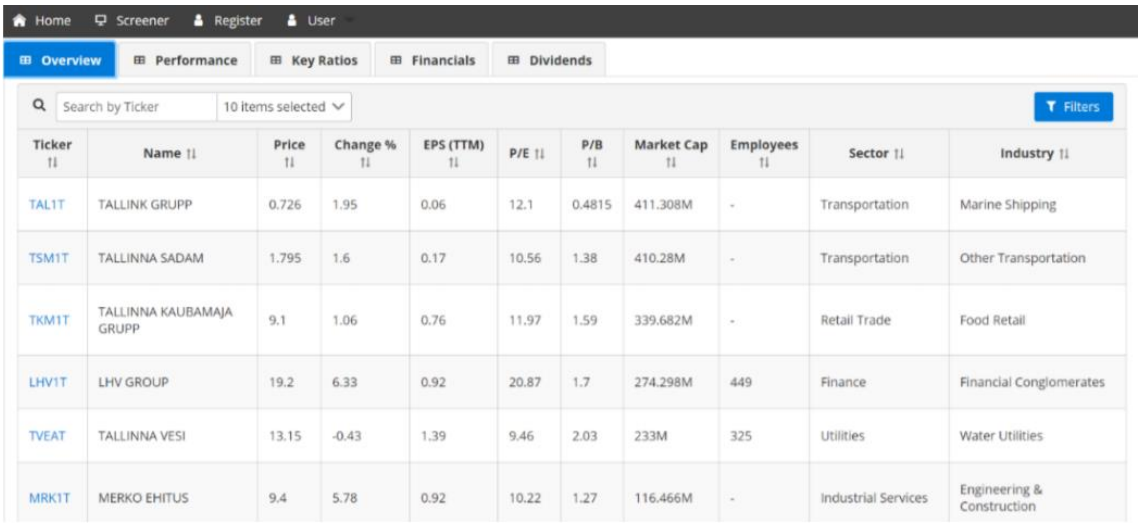
2.2.1 Käesoleva veebirakenduse esimene iteratsioon

Finantsrakenduse esimese iteratsiooni lõi Marko Jagor oma bakalaureusetöö raames 2020. aastal [1]. Veebirakendus kasutas *mock* ettevõtete andmeid, mida rakendus kasutab osaliselt siiamaani. Tagarakendus suhtles ja suhtleb siiani kasutajaliidesega läbi REST arhitektuuril loodud tagarakenduse, mida selle iteratsiooni käigus märgatavalt täiendati.

Kõige esimesena valmisid esimese arenduse iteratsiooni käigus aktsiasümboli järgi ettevõtte info otsimine, ettevõttega seotud andmete kuvamine tabelites ning üksik detailivaade.

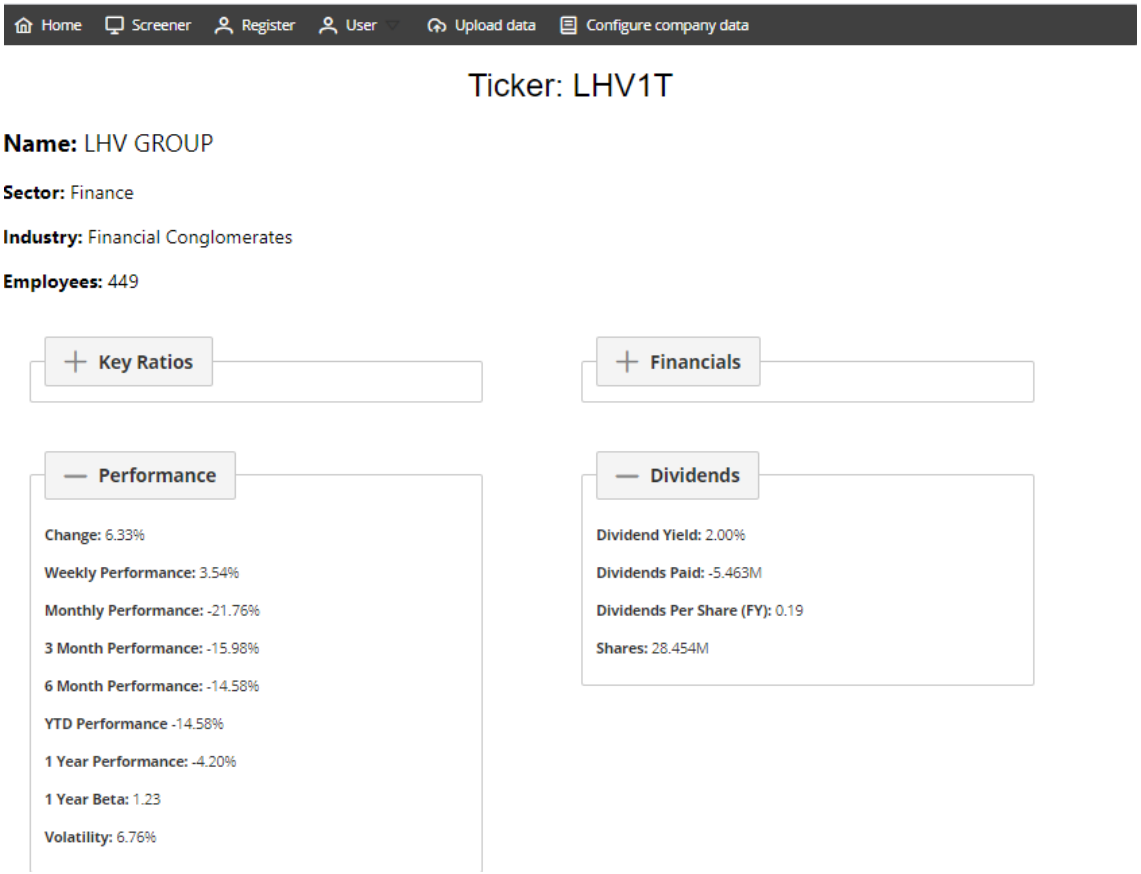
Joonis 1. Esialgse rakenduse esilehe ekraanisalvestus

Tabelivaates on võimalik valida viie alateema vahel: *Overview*, *Performance*, *Key Ratios*, *Financials* ning *Dividends*. Selles sisaldub siiamaani peamiselt programmi poolt lisatud *mock*-informatsioon, kuid hind ja mõned muud näitajad arvutatakse reaalajas projekti teise iteratsiooni tulemusena. *Filter*-nupu abil on võimalik tabelit filtreerida erinevate parameetrite alusel.



Ticker	Name	Price	Change %	EPS (TTM)	P/E	P/B	Market Cap	Employees	Sector	Industry
TAL1T	TALLINK GRUPP	0.726	1.95	0.06	12.1	0.4815	411.308M	-	Transportation	Marine Shipping
TSM1T	TALLINNA SADAM	1.795	1.6	0.17	10.56	1.38	410.28M	-	Transportation	Other Transportation
TKM1T	TALLINNA KAUBAMAJA GRUPP	9.1	1.06	0.76	11.97	1.59	339.682M	-	Retail Trade	Food Retail
LHV1T	LHV GROUP	19.2	6.33	0.92	20.87	1.7	274.298M	449	Finance	Financial Conglomerates
TVEAT	TALLINNA VESI	13.15	-0.43	1.39	9.46	2.03	233M	325	Utilities	Water Utilities
MRK1T	MERKO EHITUS	9.4	5.78	0.92	10.22	1.27	116.466M	-	Industrial Services	Engineering & Construction

Joonis 2. Esialgne *mock*-andmebaasiga täidetud ettevõtete andmete kuvamine



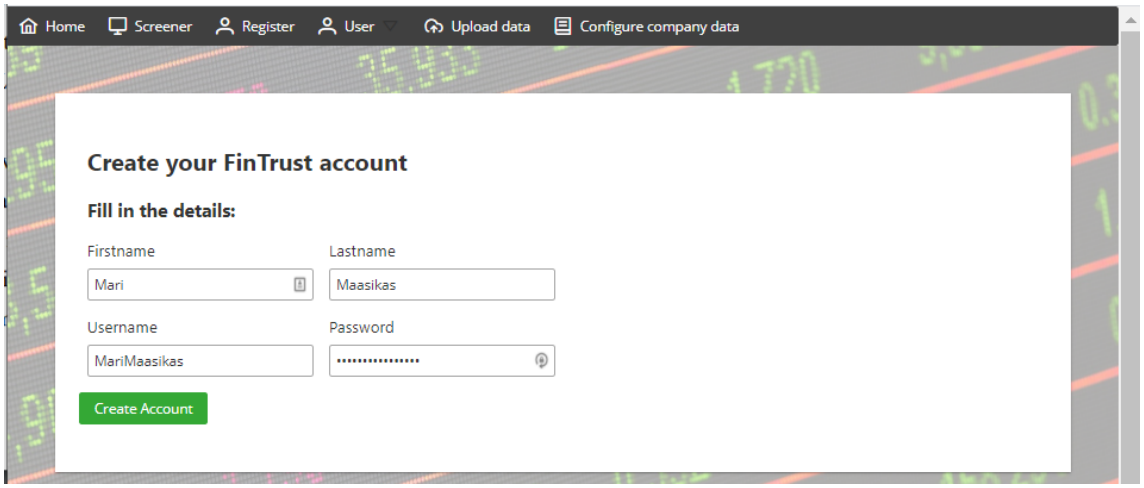
Joonis 3. Börsiettevõtte detailivaade

2.2.2 Veebirakenduse teine iteratsioon

Esimeses versioonis valmisid vaated *Home*, *Screener* ja *CompanyInfo*. Teise iteratsiooni käigus lisati juurde hulk uusi vaateid: *Login*, *Register* ja *Profile*. Muudatuse tegi läbi ka menüüriba.

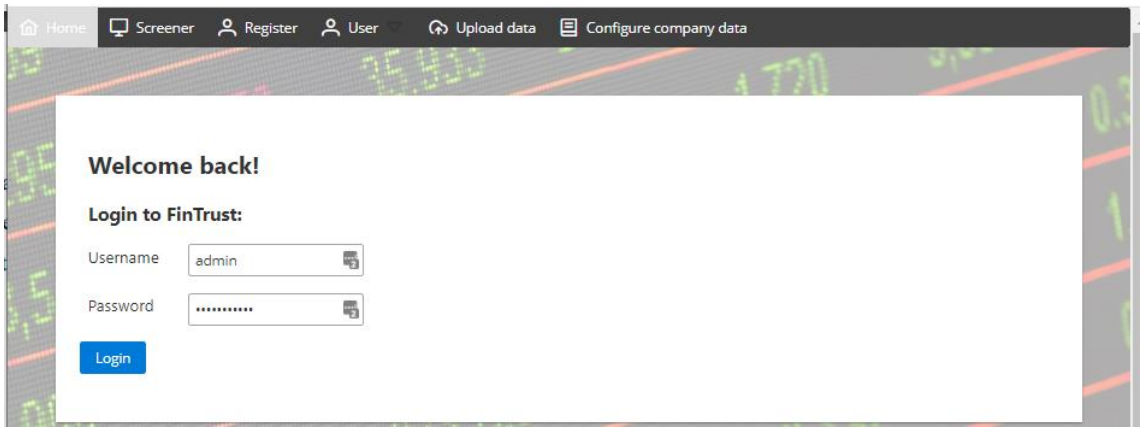
Register-lehekülje eesmärk on kasutaja registreerimine, lisades vajaminevad parameetrid vormi väljade sisse. Väljad olid ja on siiamaani kohustuslikud ning andmete sisestamise järgselt viiakse läbi nende valideerimine. Antud kasutajaliidese komponent on kasutusel muutmata kujul ka kolmanda iteratsiooni järgselt.

Täpsem info valideerimise ja tagarakendusega suhtluse loogika kohta leiab projekti teise iteratsiooni dokumentatsioonist. [2]



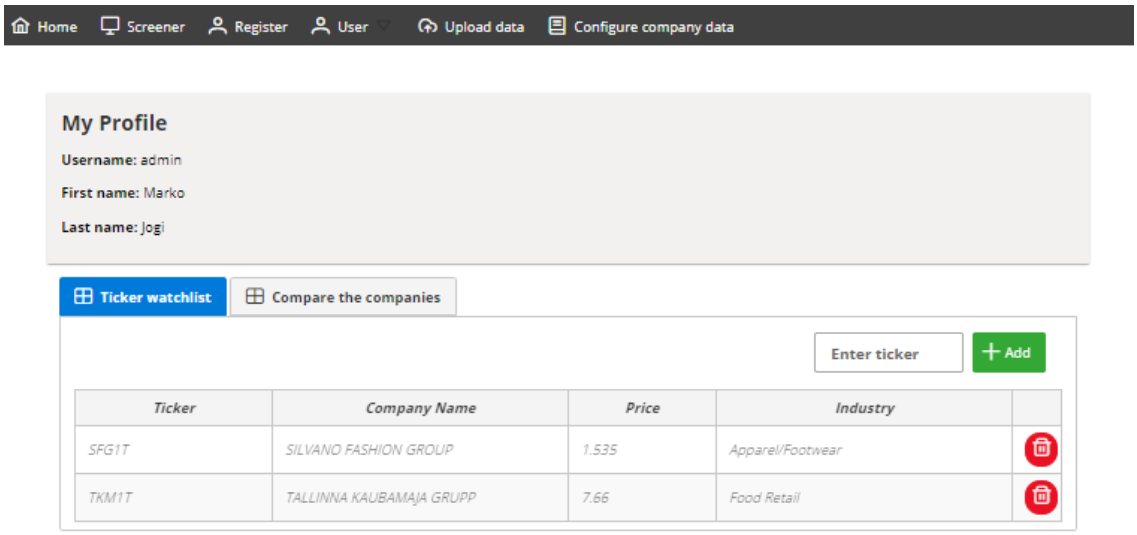
Joonis 4. Registreerimise komponendi vaate sisselogimise vorm

Samaaegselt lisati juurde ka *Login* vaate, mis võimaldab kasutajal end autoriseerida. Täpsem info selle kohta on samuti eelmise iteratsiooni dokumentatsioonis. [2]



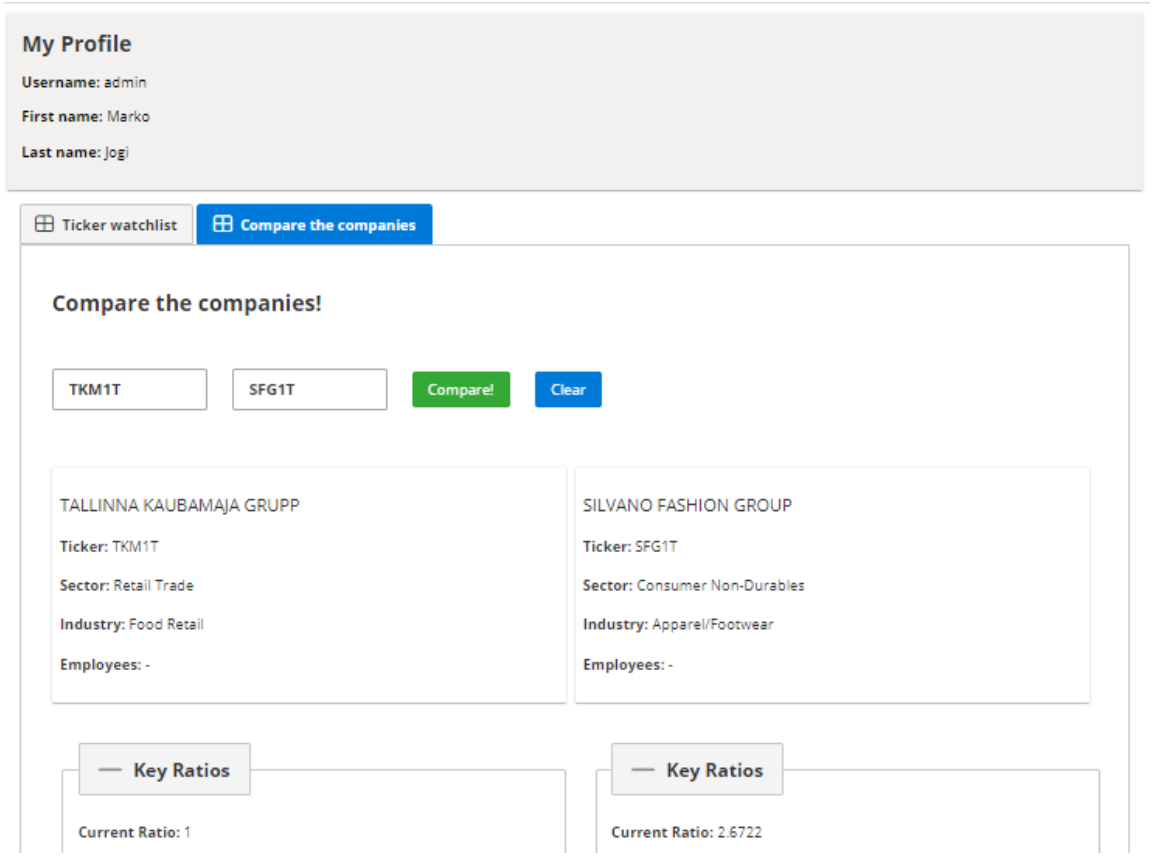
Joonis 5. Kasutajaliidese sisselogimise (*Login*) vaade

Järgnevalt lisati profiili nägemise vaate. Selle vaate võimalusteks oli kasutaja informatsiooni kuvamine, personaalse aktsiate jälgimisnimekirja koostamine ja kahe balti börsil asuva ettevõtte omavaheline võrdlemine.



Joonis 6. Kasutajaliidese Profile vaade

Loodi börsiettevõtete võrdlemise komponent, mis vajab sisendiks kahte võrreldavat ettevõtet.



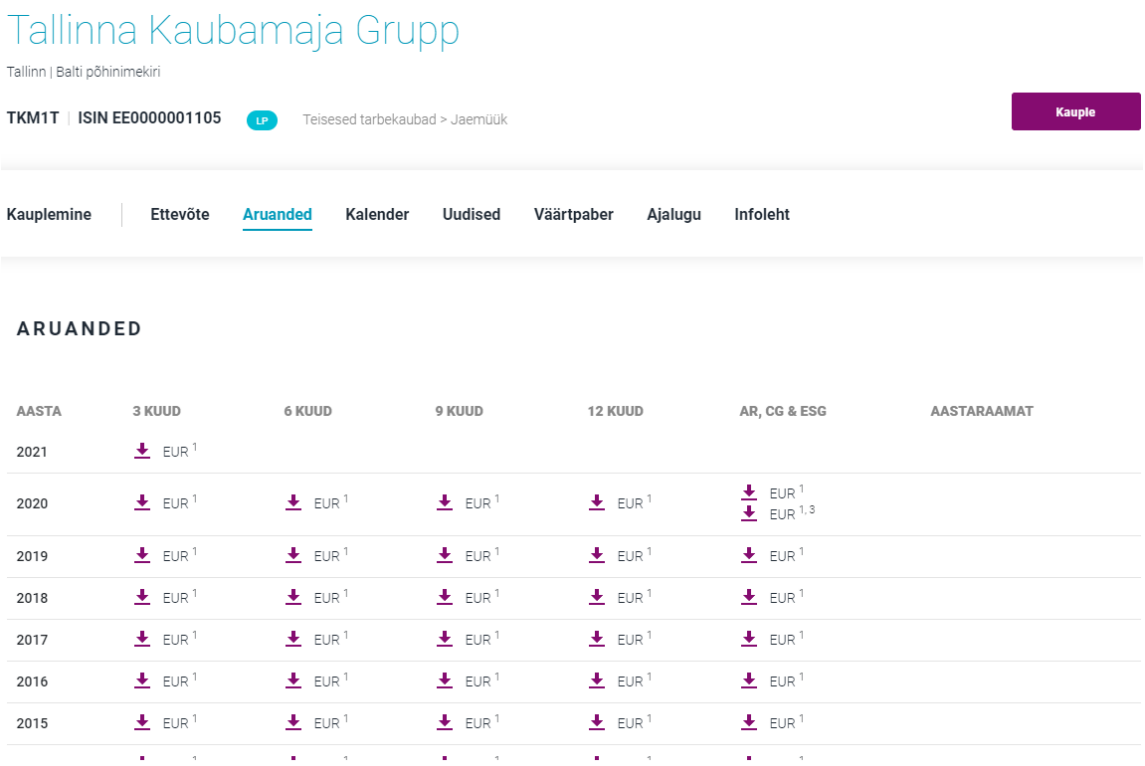
Joonis 7. Börsiettevõtete võrdlus Tallinna Kaubamaja ja Silvano Fashion Group

2.2.3 Nasdaq Balti börsi veebilehel ettevõtte ajalooliste väärtuste kuvamine

„Nasdaq Balti börside visiooniks on see, et Balti börsidel oleks noteeritud kõik ambitsioonikad Balti ettevõtted ning siin investeerivad kõik ambitsioonikad balti inimesed.“ [4]

„Börs võimaldab kauplemist erinevate turuinstrumentidega – Baltikumis noteeritud aktsiate, võlakirjade ja fondidega“, märkis Käppa. [2].

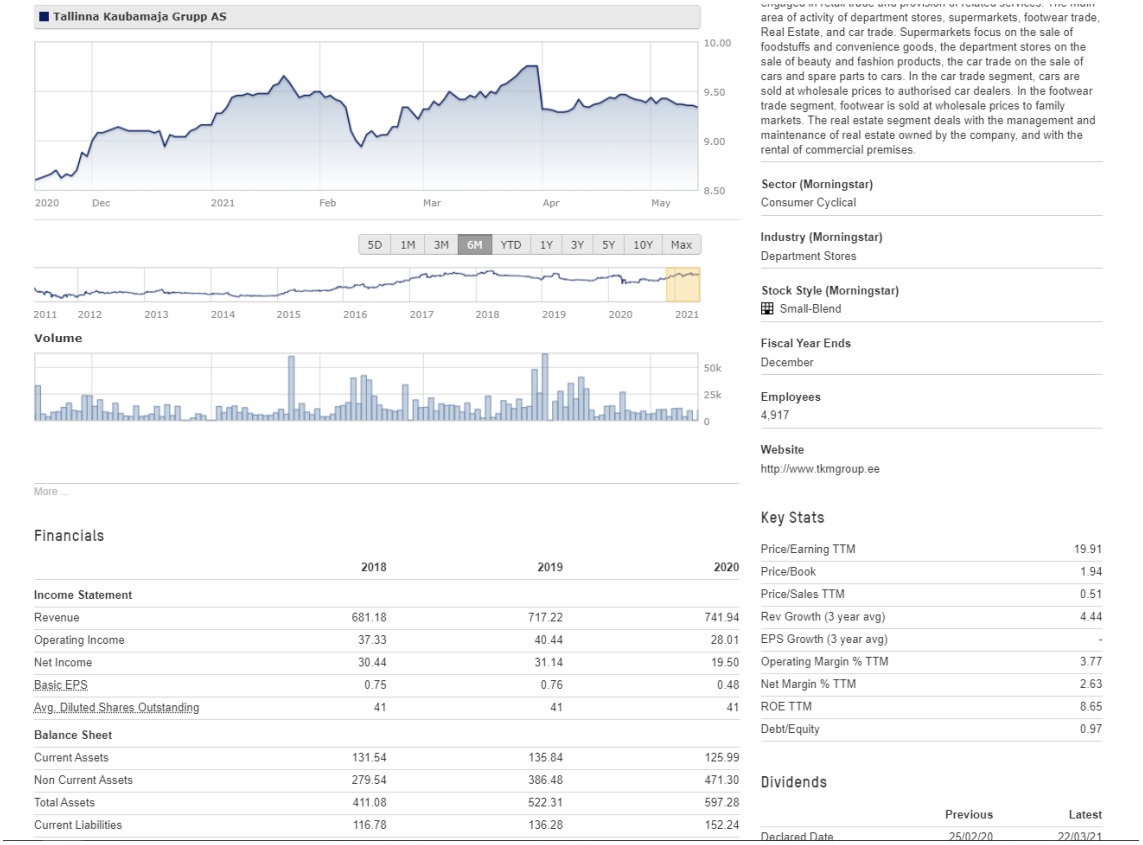
Ka eelmistes iteratsioonides toodi välja, et selle platvormi tugevuseks on riigi kohalik järvevalveamet ning et seal on võimalik leida ajaloolisi andmeid aastate lõikes. Sellest tulenevalt võimaldatakse ka loodavas rakenduses alla laadida ettevõtte majandusaruandeid CSV-failidena, mis on varasemalt administraatori poolt süsteemi sisestatud. [5]



Joonis 8. Tallinna kaubamaja aktsia aruannete allalaadimise võimalused

Lisaks toodi välja, et ajaloolise info leidmine on raskendatud, vajaliku info kättesaamiseks peab läbi otsima pikad majandussaasta aruanded. Selle funktsionaalsuse mugavama lahendusega tegeletakse kolmandas projektiarenduse iteratsioonis.

Nasdaq Baltic veebilehelt on võimalik infolehe alt näha konkreetse firma kohta finantstulemuste andmeid, milleks on finantsaruanded, põhistatistika ja info dividendide kohta. Hetkel on sarnane info rakenduses olemas *mock*-andmestiku kujul, mida näeb *screener* tööriistas ettevõtte peale vajutamisel. Selle iteratsiooni käigus võimaldatakse administraatori poolt koostada standardiseeritud finantsaruanded, millega on investoril võimalik tutvuda. [6]



Joonis 9. Tallinna kaubamaja finantsinfo ülevaade

Eraldi on loodud võimalus tutvuda ettevõtte suhtarvudega, mida tihti kasutatakse investeerimisotsuste tegemisel. Selline tegevus annab infot ettevõtte käekäigu kohta ning teeb erinevad ettevõtted samast valdkonnaks omavahel võrreldavaks. Sarnane lahendus tuleks kindlasti luua tulevastes investeerimise rakenduse iteratsioonides, kuid hetkel on see väljaspool arenduse skoopi. [6]

FINANTSANDMED

Ülevaade | Tootlus | **Suhtarvud** | Finantsid | Infoleht | Metoodika

Margins (% of Sales)

	2016	2017	2018	2019	2020
Revenue	100.00%	100.00%	100.00%	100.00%	100.00%
Cost of Revenue	75.32%	75.29%	75.26%	75.53%	75.96%
Gross Margin	**24.68%**	**24.71%**	**24.74%**	**24.47%**	**24.04%**
SG&A	1.17%	1.15%	1.13%	1.15%	1.05%
Research and development	-	-	-	-	-
Operating Margin	**5.30%**	**5.70%**	**5.48%**	**5.64%**	**3.77%**
Net Int. Inc and Other	10.47%	11.30%	10.87%	10.89%	7.00%
EBT Margin	**5.18%**	**5.60%**	**5.39%**	**5.25%**	**3.23%**

Profitability

	2016	2017	2018	2019	2020
Tax Rate	0.88%	1.02%	0.92%	0.91%	0.60%
Net Margin	4.30%	4.58%	4.47%	4.34%	2.63%
Return on Assets	**6.98**	**7.59**	**7.53**	**6.67**	**3.48**
Financial Leverage	1.91	1.91	1.82	2.29	2.68
Return on Equity	**13.42**	**14.49**	**14.04**	**13.73**	**8.65**

Joonis 10. Tallinna Kaubamaja suhtarvude ülevaade

Võimalik on ka näha igat finantsaruannet eraldi, mis tuleb kasuks investoritele, kellele meeldib ettevõtte finantsnäitajate detailidesse süveneda. Näiteks on toodud kuvatõmmis kasumiaruande kohta. [6]

FINANTSANDMED

Ülevaade | Tootlus | Suhtarvud | **Finantsid** | Infoleht | Metoodika

Income Statement

	2016	2017	2018	2019	2020
Revenue	598.41	651.26	681.18	717.22	741.94
Cost of Revenue	450.74	490.30	512.65	541.73	563.61
Gross Operating Profit	**147.67**	**160.96**	**168.53**	**175.50**	**178.33**
Operating expenses					
Research and development	-	-	-	-	-
Sales, general and administrative	7.02	7.51	7.72	8.23	7.81
Staff cost	-	-	-	-	-
Depreciation and amortization	15.59	13.36	13.43	30.74	35.14
Other Operating Expenses	93.37	102.98	110.06	96.09	107.37
Total Operating Expenses	115.98	123.85	131.20	135.06	150.32
Operating income before interest and taxes	**31.69**	**37.11**	**37.33**	**40.44**	**28.01**
Non-operating income	-0.71	-0.61	-0.60	-2.78	-4.05
Income before income taxes	30.98	36.50	36.74	37.66	23.96
Provision for income taxes	5.26	6.67	6.30	6.52	4.46
Net income from continuing operations	25.73	29.83	30.44	31.14	19.50
Net Income	**25.73**	**29.83**	**30.44**	**31.14**	**19.50**
Net income available for common shareholders	25.73	29.83	30.44	31.14	19.50
Earnings per share					
Basic	0.63	0.73	0.75	0.76	0.48
Diluted	0.63	0.73	0.75	0.76	0.48

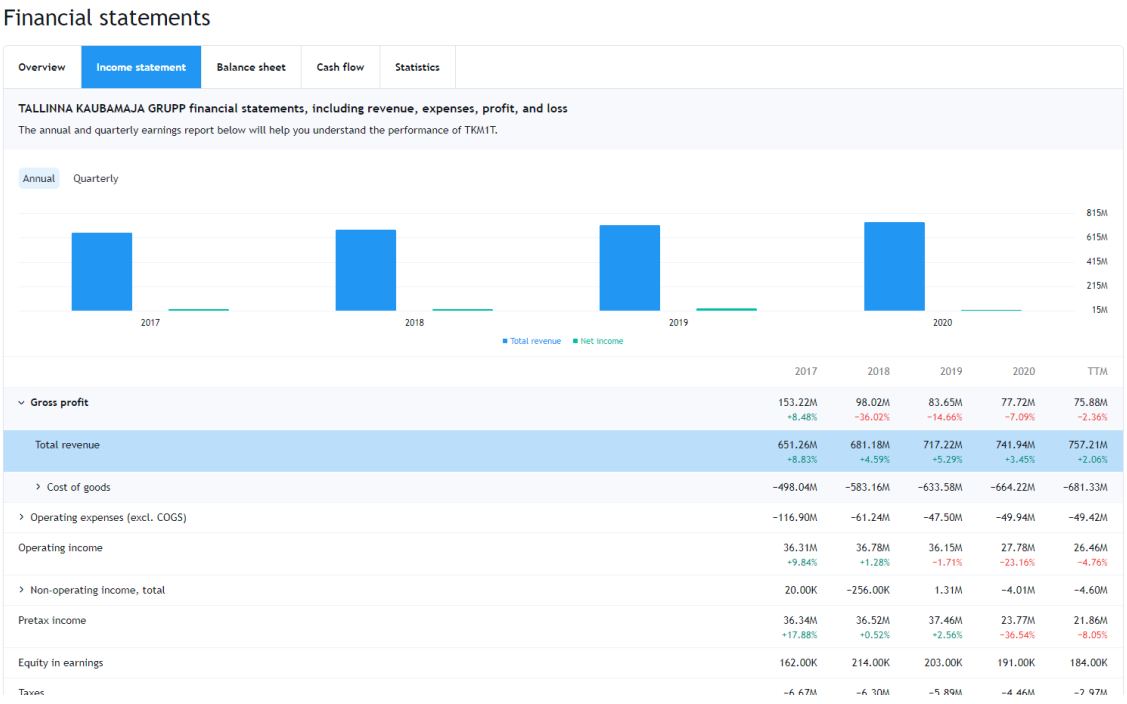
Joonis 11. Tallinna Kaubamaja kasumiaruande ülevaade

2.2.4 TradingView veebilehel ettevõtte ajalooliste väärtuste kuvamine

TradingView on internetist saadav tarkvara, mille abi kasutatakse finantsmaailmas erinevate finantsalaste graafikute ning jälgimisnimekirjade koostamisel. Selle abil on võimalik luua tabelivaateid, mis sisaldab endas kasutaja poolt lisatud parameetreid, selle sisu ja kasutatavad andmed on lihtsasti konfigureeritavad. [7]

TradingView on Nasdaq Baltic'ule alternatiivne rakendus, mis võimaldab tutvuda üle 50 börsiga üle maailma ning seal esineb ka Tallinna börs [7]. Kuna see veebileht keskendub ka suurematele börsidele, siis kahjuks jääb väiksemate börside detailisus puudulikuks.

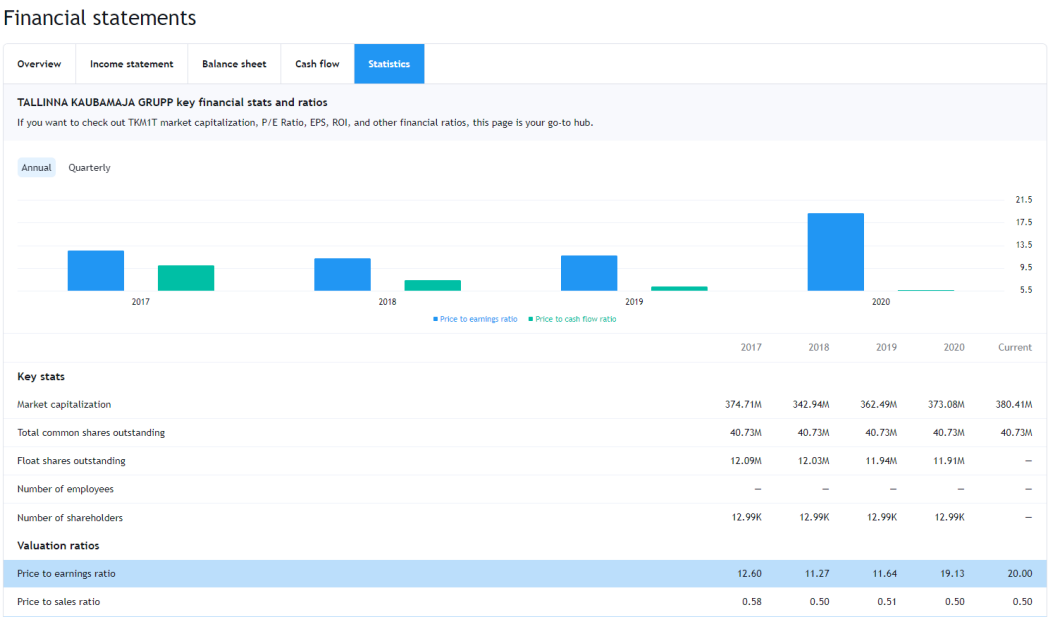
TradingView's saab sarnaselt Nasdaq Tallinnale tutvuda ettevõtte ajalooliste andmetega. Selleks tuleb otsingusse kirjutada otsitava ettevõtte lühinimi, mille järel suunduda vahelehele *financials*. Seal on võimalik tutvuda bilansi-, rahavoogude- ning kasumiaruandega. Eriti ülevaatlikuks teeb need lehed võimalus näha väärtuste muutumist üle mitme aasta [8]. Vt joonis 14.



Joonis 12. Tallinna kaubamaja kasumiaruannete ülevaade – TradingView

Siin eksisteerib ka sarnaselt Nasdaq Tallinn veebilehele firma üldise finantsstatistika vaheleht *Statistics*, millele navigeerides avaneb info ettevõtte põhistatistika, väärtuskordajate, kasumikordajate, likviidsuskordajate ning maksuvõime suhtarvude kohta (joonis 15). Kõik need asjaolud võiksid kättesaadavad olla ka veebirakenduses, mis

kajastab Tallinna börsiettevõtete finantsandmeid. Kõik selle arenduse iteratsiooni käigus realiseerimata jäänud kasulikud vaateid tasuks kaaluda tulevastes arenduse iteratsioonides. [8]



Joonis 13. Tallinna Kaubamaja finantsnäitajate ülevaade - TradingView

2.3 Kasutatud tehnoloogiad

Peatükis leiab ülevaate bakalaureusetöös kasutatud tehnoloogiate ja tööriistadest.

2.3.1 Projekti arenduskeskkonnad ja koodi hoidla

Tagarakenduse koodi kirjutamise keskkonnaks valisid autorid IntelliJ IDEA versiooni 2019.3. Tegemist on rikkaliku sisseehitatud analüüsitööriistadega koodiredaktoriga, mis enda reaalaja soovitustega võimaldas paremini kirjutada puhast koodi. SQL Server käivitatakse keskkonda integreeritud Apache TomCat'i abil. Samuti võimaldas IDEA otsest juurdepääsu PostgreSQL andmebaasi sisuga. Lisaks kasutati paljusid Java Spring ökosüsteemi kuuluvaid tehnoloogiaid, mis tegi arenduse kergemaks ning mis võimaldasid keskkonna konfigureerimise asemel tegeleda äriloogika ja funktsionaalsuse arendamisega.

Klientrakenduse koodi kirjutati Visual Studio Code redaktoris.

Andmete kraapimise lahendust, mis on realiseeritud Pythoni programmeerimiskeeles, kolmandas iteratsioonis edasi ei arendatud.

Arendatav investeerimisrakendus on eelmiste lõputööde jätk. Et võimaldada potentsiaalsetele tulevastele arendajatele ligipääsu koodibaasile, jäetakse projekti lähtekood avalikesse GitHubi repositooriumitesse.

2.3.2 *Back-end* komponendid ja andmebaas

Projekti *back-end* koosneb kahest osast: Java keeles kirjutatud tagarakendus ise ja veebist ajakohaste väärtpaberihindade leidmiseks Pythoni keeles kirjutatud *scraper*, mille funktsionaalsuse automatiseerimiseks kasutati Task Scheduleri ja PowerShelli keelt [2] ja mida kolmandas iteratsioonis ei muudetud.

Tagarakendus ja *scraper* suhtlevad omavahel läbi RabbitMQ sõnumiedastaja. RabbitMQ kasutamiseks tuleb eelnevalt installida arvutisse Erlang programmeerimise keel [2].

Uusi osasid *back-end*'i juurde ei tulnud, sest kolmanda iteratsiooni eesmärk oli REST-stiilis tagarakenduse äriloogika edasiarendus ning sisestatud andmete korrektne salvestamine andmebaasi.

Andmebaas kasvas selle arendustöö etapi vältel mitmekordselt. Andmete salvestamise keerukuse paremaks mõistmiseks on loodud Enterprise Architect tarkvara abil andmebaasi arhitektuurimudel[9].

2.3.3 *Front-end* komponent

Klient- ehk *front-end* rakenduse puhul jätkati eelmiste autorite loodud klientrakenduse arendamist, mis tähendab, et lähtuti samadest tehnoloogiatest, mida eelmised autorid kasutasid. Arendustegevust jätkati, kasutades Reacti teeki.

Erinevalt projekti eelnevatest iteratsioonidest polnud seekord paljuski võimalik klientrakenduse komponentide kiire väljatöötamine PrimeReacti komponente kasutades, sest need ei arvestanud tagarakenduse koodibaasi oluliste nüanssidega. Seetõttu läks kasutajaliidese arendamine üldpildis võrreldes projekti eelmiste iteratsioonidega aeglasemalt. Küllaga peeti komponentide väljatöötamisel silmas nende paindlikust, et projekti tulevastel arendajatel oleks võimalik neid klientrakenduse erinevates osades võimalikult lihtne oma otstarbel taaskasutada. Siiski kasutati PrimeReacti komponente seal, kus võimalik, et hoida kokku arendusele kuluvat aega.

3. Töö tulemused

Peatükis tuuakse välja kasutajalugudena kirja pandud projekti funktsionaalsed ja mittefunktsionaalsed nõuded, selgitatakse rakenduse arhitektuuri ja disaini. Lõpuks antakse ülevaade kirjutatud koodist ning testidest.

3.1 Kasutajalood

Lugenud ja analüüsinud läbi rakenduse teise versiooni ning hinnates, millised rakenduse osad annaksid edaspidi kõige enam väärtust, tulid tudengid koostöös juhendajaga välja esialgsete kasutajalugudega, millest arendati välja allpool toodud.

Kasutajalugusid saab esitada mitmel kujul. Neist kõige levinum on Connextra-mall, mida antud lõpuprojektis ka kasutatakse [10].

Arendaja kasutajalood:

- Arendajana soovin dokumenteerida kogu arendustegevust piisavalt täpselt, et uued arendajad saaksid tulevikus varasemate arendajate poole pöördumata iseseisvalt rakendust edasi arendada.
- Arendajana soovin katta testidega tagarakenduse eesmärgiga vältida märkamatute vigade tekkimist ja projekti edasiarenduse olulist aeglustumist tulevikus.
- Arendajana soovin rakendust arendada nii, et rakendus töötaks uuemates kõige sagedamini kasutatavates veebilehitsejates, selleks, et kasutajatel ei tekiks veebilehitsejaga seoses probleeme. Nõue tuleneb eelmisest rakenduse iteratsioonist [2].
- Arendajana soovin rakendust arendada, jälgides tööstuse parimaid tavasid, et tulevastel arendajatel oleks lihtsam tööd jätkata.

Finantsteadmistega administraatori kasutajalood:

- Administraatorina tahan lisada uusi aruannete komplekte läbi CSV-faili üleslaadimise eesmärgiga võimaldada seda informatsiooni investoritele nii standardiseerimata kui standardiseeritud aruannete kujul.
    - Eelnevalt tuleb fail muuta vastavale kujule, et programm suudaks seda töödelda. Juhend on välja toodud lisas 5.
- Administraatorina tahan lisada mitu komplekti CSV-faile korraga ühele ettevõttele aja kokkuhoiu eesmärgiga.
- Administraatorina tahan CSV-failide üleslaadimisel saada programmilt kohest märguannet CSV-faili nõuete rikkumise kohta, et saaksin faili viia korrektuurid.
- Administraatorina tahan luua standardiseerimata aruannete põhjal firma standardiseeritud finantsaruanded, et luua investoritele paremad tingimused ettevõtete võrdlemiseks ja et standardiseeritud aruannete baasil arvutada erinevaid finantskordajaid.
    - Luua võimekus kindlal perioodil kehtivate aruannete konfiguratsioonide loomisele andes konfiguratsiooni kehtimise alguskuupäeva ja lõppkuupäeva.
    - Standardsete aruannete kirjeridade valemite loomine toimub kasutajaliideses vastavate väljade täitmisel. Andmebaasi jääb märge valemist, mida kasutati standardse aruande väärtuste arvutamiseks.
- Administraatorina tahan aruannete konfiguratsioonide loomisel sisestada MS Excelis kasutatavatele sarnaseid valemeid eemärgiga lihtsustada valemite kirjutamise õppimist.

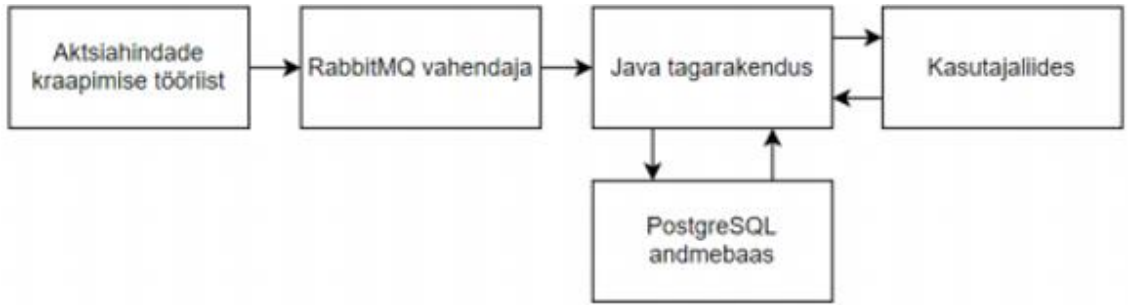
3.2 Arhitektuur

Rakenduse arenduse kolmandas iteratsioonis on säilitatud kihiline arhitektuur. Erinevate rakenduse osade vahelisi sõltuvusi hoiti minimaalsetena. Kood on grupeeritud otstarbe järgi erinevatesse pakettidesse.

Reactis kirjutatud klientrakenduse eesmärgiks on päringute tegemine tagarakendusse ning kuvada infot erinevate kasutajarollide jaoks. Java tagarakendus ja *scraper*-tööriist tegelevad äriloogikaga. Teise iteratsiooni ajal valminud tagarakendus tegeleb

andmebaasipäringutega, finantskordajate ja suhtarvude kalkuleerimise, kasutajate autentimise ning autoriseerimisega. [2]

Lisaks teises iteratsioonis valminud täpse arhitektuuri kirjeldusele on käesoleva edasiarenduse jooksul lisandunud juurde CSV-failide süsteemi originaalsel kujul salvestamine, lugemise, töötlemise ning andmebaasi salvestamise funktsionaalsused; standardiseerimata aruannete ridade kaardistamine tabelitesse muudetavate valemitena, mida saab rakendada erinevatele ajaperioodidele. Peale valemite salvestatakse arvutatud väärtused eraldi andmebaasi tabelitesse. Üldine teise iteratsiooni käigus loodud arhitektuur jäi samaks ning on nähtav joonise 14 peal.



Joonis 14. Teise rakenduse versioonist tulnud kehtiv komponentide omavaheline seos

3.2.1 Java *back-end* komponendi arhitektuur

Kood on jaotatud pakettidesse: *config*, *controller*, *exception*, *model*, *repository*, *security*, *service*, *utils*, mis omakorda sisaldavad kaustasid koodifailidega.

Kihilise arhitektuuriga jätkamine võimaldab täita alguses püstitatud ülesannet: hoida koodi võimalikult loetavana, organiseerituna ja lihtsasti edasiarendatavana. Eraldi töökindlust tagavad laia ulatusega enne lõpuprojekti kaitsmist valmivad tagarakenduse testid, mis annavad rakenduse mitte-eeldustekohase käitumise korral vigadest märku.

Projekti tagarakenduse arhitektuur on nähtav mitmel ekraanitõmmisel, kus kolmandas iteratsioonis edasi arendamata äriloogika ja *infra* üritatakse välja jätta. Vt lisa 8.

Järgnevalt on välja toodud kolmanda iteratsiooni käigus muudetud või lisandunud klasside ülesanded. Tervikliku pildi nägemiseks võiks paralleelselt uurida ka teise iteratsiooni kaustade selgitusi [2].

- Model.statement paketis paiknevad klassid sisaldavad abstraktseid finantsaruannete klasse.
- Model.statement.attribute paketis paiknevad klassid, mis sisaldavad standardiseerimata kujul saadud erinevate raamatupidamisaruannete kirjeid koos väärtusega.
- Model.statement.balance paketis paiknevad bilansi mudelid.
- Model.statement.income paketis paiknevad kasumiaruande mudelid.
- Model.statement.cashflow paketis paiknevad rahavoogude mudelid.
- Model.statement.configuration paketis paiknevad kõikide finantsaruannete konfigureerimise mudelid.
- Model.statement.groupOfStatements paketis paiknevad nii standardiseerimata kui ka standardiseeritud aruannete kogumiku mudelid.
- Model.statement.sourceFile paketis paikneb CSV-faili mudel, mille alusel säilitatakse info rakenduses varasemalt üleslaetud failide kohta.
- Service.configuration paketis paiknev klass teostab konfiguratsiooni loomise toe kontrollerile.
- Service.csvreader paketis paiknev klass teostab andmete CSV-failist lugemist ja töötlemist.
- Service.groupOfStandardStats paketis paiknev klass vastutab kas uue standardse aruannete grupi loomise või olemasoleva uuendamise eest.
- Service.rawStats paketis paiknev klass teeb csvReader klassi poolt tagastatud tulemustega vastavad andmebaasi objektid.
- Service.upload paketis paiknev klass toetab FileControllerit ning võimaldab originaalsel kujul kasutajaliidesest kätte saadud faili salvestada projekti kausta ning selle allalaetavaks muuta.
- Util.payload.request.statementMapping klassid kirjeldavad JSON objekte, millisel kujul nad peaksid tagarakendusse jõudma kasutajaliidesest.

3.2.2 Andmebaasi arhitektuur

Rakendus kasutab kolmkümmend tabelit. Kakskümmend kaks neist on uued, mida läheb vaja csv failidest imporditud aruannete salvestamiseks, tabelitevahelise seoste loomiseks, aruannete töötlemiseks ning aruannete standardiseerimiseks.

Esimeses rakenduse iteratsioonis lisati *company_dimension*, *financials_daily* ja *financials_quarterly* andmetablid. *company_dimension* nimeline tabel sisaldab Balti börsil tegutseva ettevõtte üldandmeid. *financials_daily* sisaldab finantsnäitajad, mis muutuvad päeva lõikes. *financials_quartely* sisaldab kvartalis korra muutuvaid finantsnäitajaid [2].

Teise iteratsioonina lisandusid uued andmetabelid *roles*, *users*, *user_roles* ja *user_tickers* [2].

Käppa töös toodi välja, et „Roles tabel defineerib kõik võimalikud eksisteerivad kasutajatasemed, mis erinevad nendele antud õiguste osas.“ [2]. Teises iteratsioonis eksisteeris vaid üks roll: ROLE_USER. Kolmandas iteratsioonis lisandus administraatori roll.

Users tabelisse salvestatakse eelnevalt registreeritud isikute andmed. *Roles* ja *users* tabelid on seotud mitu-mitmele seosega ning *user_roles* defineerib, mis rollid kuuluvad kasutajale. [2]

Rakenduse kasutajad saavad lisada jälgimisnimekirja neile huvipakkuvaid ettevõtteid. Seos kasutaja ja jälgitavate ettevõtete vahel luuakse *user_ticker* tabeli abil, kasutades mitu-mitmele annotatsiooni. Tehes päringuid kasutaja jälgitud ettevõtete kohta tagastatakse JSON formaadis ka seotud *financials_daily* ja *financials_quarterly* tabelites seotud objektid. Nende vahel on loodud seos üks-ühele. [2]

Kolmanda iteratsiooni käigus lisati juurde 22 uut andmetabelit, sealhulgas 12 uut mudeliklassi. Suur hulk vahetabeleid (inglise keeles *cross-table*), kus ühendatakse erinevad objektid nende identifikaatorite kaudu on vajalikud objektide vahel mitmesuguste seoste loomiseks. Taoline lahendus valiti, kuna *JPA* poolt tabelite poolautomaatne generatsioon ja nendevaheliste seoste loomine tundus autoritele kõige kergemini realiseeritav ning sel moel tuli välja puhtam kood. Kuna siiski lisandus liiga suur hulk tabeleid (võrreldes lisandunud koodifunktsionaalsuse hulgaga), hindavad autorid retrospektiivselt sellist lähenemist mitte jätkusuutlikuks ning soovitavad edaspidi luua seoseid alternatiivsel viisil, näiteks läbi võõrvõtme veeru (*foreign-key*). Võõrvõti on relatsioonilise andmebaasi tabeli veerg, mis loob seose kahe tabeli andmete vahele [11].

Kõik need tabelid toetavad administraatori tööriistade funktsionaalsust, mille abil saab CSV-faili sisestades standardiseerimata aruanded süsteemi importida. Hiljem võib ettevõtetele luua standardiseeritud aruannete konfiguratsioon, kus kasutatakse standardiseerimata aruandeid sisendina.

Enterprise Architect tarkvara abil on koostatud kaks mudelit, mis visualiseerivad andmebaasi arhitektuuri erinevaid osi (vaata lisa 6). Lihtsuse eesmärgil on mudelitest ära jäetud vahetabelid, kus seotakse kokku erinevate objektide identifikaatorid. Enterprise Arhitect'i tarkvara fail on saadaval tagarakenduse koodi repositooriumis.

Attributes tabel sisaldab erinevates aruannetes leitud standardiseerimata raamatupidamiskirjeid, millel on olemas nimetus *field_name* ja väärtus *value*. Kirjed on seotud konkreetse standardiseerimata (kasumi-, rahavoo- või bilansi-) aruandega. *Balance_statement_as_imported*, *cashflow_statement_as_imported*, *income_statement_as_imported* andmetabelid sisaldavad kindla kuupäeva/perioodi kohta süsteemi üles laetud aruandeid. Nende aruannetega on seotud atribuudid läbi vahetabelite *balance_statement_as_imported_attributes*, *cashflow_statement_as_imported_attributes*, *income_statement_as_imported_attributes*.

Kuna süsteemis on võimekus luua standardiseerimata aruannete baasil standardiseeritud aruanded, siis selle tegemiseks on tarvis eelnevalt defineeritud firma aruannete konfiguratsioone. Need sisaldavad kõiki standardiseeritud aruannete välju String andmetüübina, kuhu finantsteadmistega administraator sisestab kindlas formaadis kirjutatud valemeid. Nende valemite baasil hiljem tekitatakse või uuendatakse standardiseeritud aruanded.

Konfiguratsiooniobjektid luuakse andmetabelitesse *company_balance_statement_config*, *company_cashflow_statement_config*, *company_income_statement_config*. Iga aruande konfiguratsioon sisaldab endas kehtivuse alguse- ning lõpptähtaega. Lisaks kuupäevadele sisaldab konfiguratsioon ka kollektsiooni identifikaatorit, mille abil saab leida aruannete kolmiku. Selle kolmiku korraga valimisel saab luua standardiseeritud aruannete grupi kindla kuupäevaga.

Grupeerimise andmetabelite puhul tuleb arvestada mõne nüansiga. *Group_of_statements* tabel sisaldab endas kolme erinevat CSV-failist loetud standardiseerimata aruannetet. Mõne konkreetse aruande puudumisel seatakse sellele *null* väärtus.

Konfiguratsiooniobjektide ja standardiseerimata aruannete põhjal saab luua standardiseeritud aruanded, mis salvestatakse *Balance_statement_stand, cashflow_statement_stand* ja *income_statement_stand* tabelitesse. Standardiseeritud aruanne sisaldab endas konfiguratsioonide alusel välja arvutatud väljade väärtusi ning kohustuslikku lisainformatsiooni [12].

Kolmanda iteratsiooni jooksul loodud äriloogika realisatsioon on selline, et aruandeid käsitletakse arvutustes vaid gruppidena. Seetõttu läks vaja varasemalt mainitud tabelit *group_of_statements* ning analoogselt töötab lahendus ka standardiseeritud aruannete jaoks. *Group_of_statements_stand* sisaldab endas viidet standardiseeritud aruannetele ning seotud firmale *ticker_id*.

Loodud on ka võimalus jäädvustada CSV-faili nimi ja tema asukoht kettal (inglise keeles *path)*, et hiljem saaks vajadusel seda alla laadida. Fail on seotud konkreetse firmaga läbi välja *ticker_id*. Tabeli nimetus on *source_csv_file*.

Kuna andmebaasi arhitektuuri loomisel eelistati vahetabelite loomist andmetabeli objektide vaheliste seoste tegemiseks, siis eksisteerib veel palju eelnevalt mainitud vahetabeleid, kus seotakse *id*'de abil erinevad objektid. Need on eelkõige mõeldud objektide seostamiseks konkreetse Balti börsil tegutseva firmaga. Tabelite nimedeks on *company_dimension_balance_configs, company_dimension_cashflow_configs, company_dimension_income_configs, company_dimension_balance_raw_stats, company_dimension_cashflow_raw_stats, company_dimension_income_raw_stats*.

3.2.3 *Front-end* komponendi arhitektuur

Vajaduse puudumise tõttu jäi *front-end* komponendi arhitektuur muutmata. Klientrakenduse kood jäi juba varasemates töödes väljatoodud [1] [2] monoliitsele kujule.

Investeerimisrakenduse arendamise projekti kolmandas iteratsioonis seati eesmärgiks luua võimalus finantsettekannete üleslaadimiseks, andmete standardiseerimiseks, standardiseeritud andmete põhjal finantsindikaatorite loomiseks ja kogu käsitletava informatsiooni seostamiseks mõne kindla ettevõttega. See funktsionaalsus on suunatud

rakenduse administraatoritele ja ülalhoidjatele ning erineb kontseptuaalselt rakendusele varem arendatud funktsioonidest, mis olid suunatud rakenduse lõppkasutajatele – investoritele.

Kuna administratiivne funktsionaalsus erineb klientidele suunatud funktsionaalsusest, peeti töö planeerimise järgus diskussioone, kas tuleks juurde luua iseseisev administratiivne klientrakendus olemasoleva investorile suunatud klientrakenduse kõrvale või paigutada erinevad funktsionaalsused siiski ühte klientrakendusse. Lõpuks otsustati siiski ühise klientrakenduse kasuks, sest investeerimisrakenduse projekt tervikuna on praeguseni väga varajases järgus ning ühte ühist klientrakendust on praeguses staadiumis lihtsam hallata. Küsimus, kas ka tulevikus tasub hoida administratiivne klientrakendus koos lõppklientidele suunatud klientrakendusega, jääb praegu lahtiseks.

Domeeniloogikast lähtudes on kõige olulisemad juurde lisandunud komponendid *DataImportComponent*, mis tegeleb failide üleslaadimisega, ja *DataConfigurationComponent,* mille peamiseks ülesandeks jääb imporditud andmetest standardiseeritud ettekannete koostamine.

Klientrakendusse lisati failide üleslaadimise esmane vaade täielikult ja failide konfiguratsioonide vaade osaliselt.

3.3 Disain

Peatükis saab ülevaate projekti tarkvarakomponentide disainimustrite eripäradest.

3.3.1 Java *back-end* komponendi disain

Projekti järjekordses edasiarenduse iteratsioonis on jätkati REST API disainimustri kasutamist. Selline stiil on tarkvaraarenduse maailmas hästi populaarne tänapäevaste *web API* teenuste disainis. Disaini puhul jätkati M. Masse raamatu disainimustrite reeglite kasutamist, millega alustati juba esimeses iteratsioonis. [13]

Käppa töös toodi välja tähtsad tähelepanekud: „API-d kasutavad URI-sid (Uniform Resource Identifiers) ressursside täpse aadressi defineerimiseks ja nende kättesaamiseks. Uute URI-de disainis on lähtutud reeglitest, et kasutatakse vaid väikseid tähti, URI ei tohiks lõppeda „/“ märgiga, kuna iga sümbol on oluline unikaalse identiteedi osas. Samuti

peaks olema kollektsioonile viitav URI nimisõna mitmuses ja URI-d, mis viitavad controller-i ressursile, peavad olema nimetatud tegusõnaga.“ [2]. Kõiki tsiteeritud reegleid täitsid ka käesoleva lõpuprojekti autorid.

Käesoleva arenduse käigus lisati peamiselt uusi GET päringuid, kuid konfiguratsioonide loomine ja faili üleslaadimine toimub läbi POST päringute. Konfiguratsioone uuendatakse PUT päringu abil. Käppa töös toodi välja, et “DELETE-i kasutatakse ressursi eemaldamiseks kollektsioonist ning POST-i uue ressursi lisamiseks kollektsiooni ja kontrollerite töö täitmiseks.” [2]. Meie lisatud kontrolleri töö põhineb samadel reeglitel.

Eelmistes töödes soovitatud JSON formaadi säilitamist sõnumite struktureerimiseks täideti. JSON’id kujutavad endast ette õigel struktuuril vormindatud võti-väärtus paaride kogumikku.

Tagarakenduse poolel jätkati edukalt Spring raamistiku kasutamist ning ühe kõige tähtsama tehnika kasutamist, mida nimetatakse *dependency injection’*iks. Selle tehnika kasutamisel luuakse objektile seos teise objektiga. See objekt on *singleton* tüüpi, mis tähendab, et terve programmi eluea jooksul kasutatakse ainult ühte objekti instantsi. [14].

3.3.2 *Front-end* komponendi disain

Klientrakenduses ei muudetud juba eksisteerivate komponentide lähtekoodi.

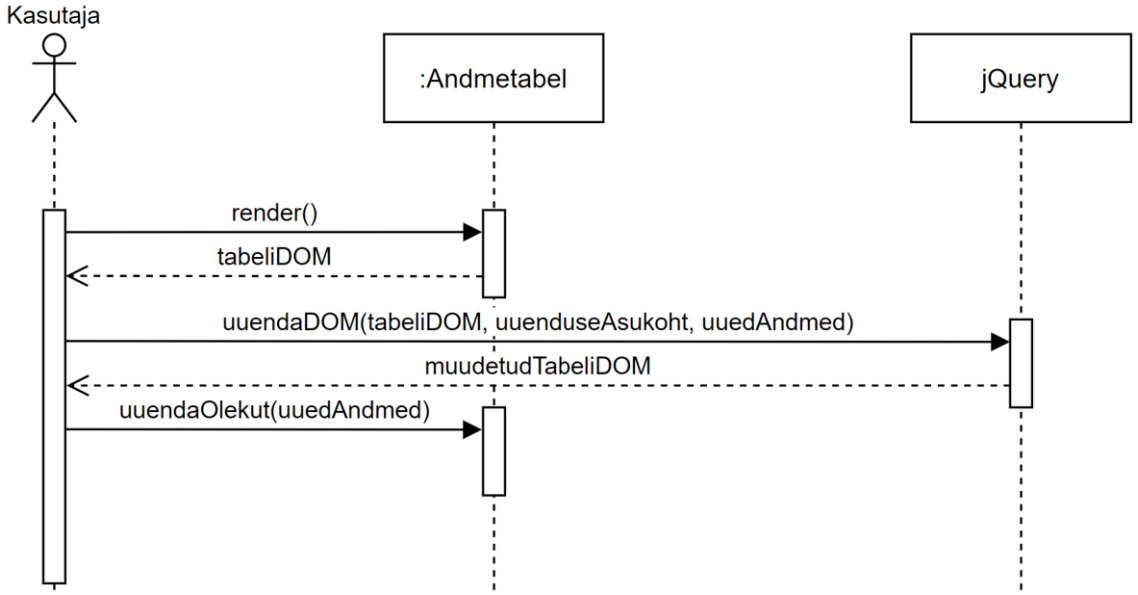
Tehnilise poole pealt võib ühe suurema erinevusena välja tuua funktsionaalsete komponentide kasutuselevõtu ja nende eelistamise üle klassikomponentide seal, kus võimalik [15]. Töö autorite hinnangul on funktsionaalsed komponendid intuitiivsemad: neid on lihtsam koodis realiseerida ja teistel arendajatel (eriti nendel, kellel puudub eelnev kogemus Reactiga) on koodist lihtsam aru saada.

Uute komponentide loomisel jälgiti ühe vastutuse printsiipi: igal komponendil peaks olema ainult üks selgelt määratletud vastutusala. [16]

Uusi komponente prooviti kirjutada võimalikult kooskõlas Reacti põhimõtetega. See võttis rohkem aega, sest varajasema arenduskogemuse puudumise tõttu tuli esmalt Reacti kohta õppida.

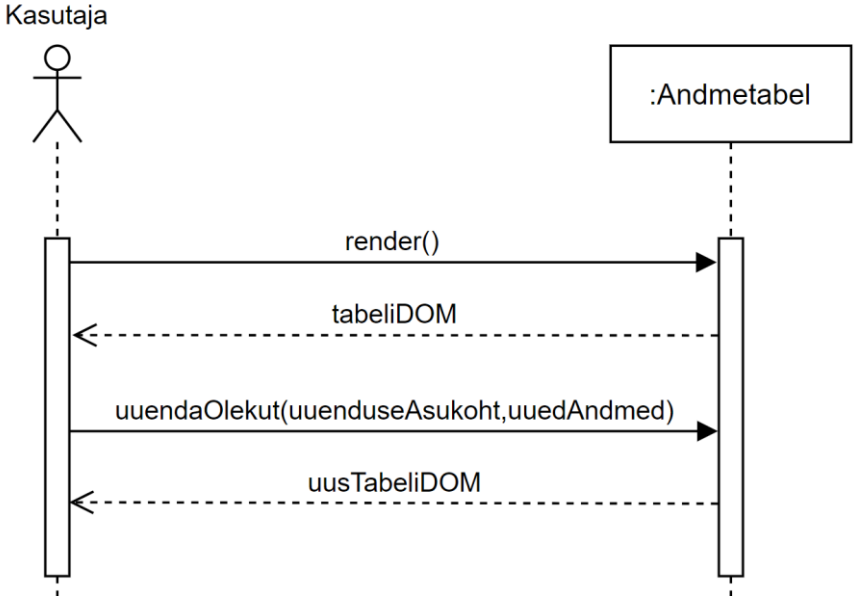
Eelnevalt väidetu näiteks võib välja tuua andmetabeli komponendi kirjete muutmise. Kirjete muutmist oleks võinud teha ka näiteks kasutades jQuery teeki, aga see oleks

tähendanud esmalt DOMi elementide otsest modifitseerimist ja siis muudatuste kajastamist komponendi olekus.



Joonis 15. Lihtsustatud joonis andmetabeli kirjete väärtuste muutmiseks jQuery abil

React aga on disainitud selliselt, et juba renderdatud DOMi elemente ei muudeta, vaid mõne sündmuse (näiteks kasutaja trükib tekstiväljale teksti) puhul kajastuvad muutused kõigepealt komponendi olekus (inglise keeles *Component State*) ning oleku muutumise tõttu renderdatakse muutunud olekuga komponent uuenenud olekut arvestades.



Joonis 16. Lihtsustatud joonis andmetabeli kirjete väärtuste muutmiseks Reacti tööpõhimõtteid arvestades

3.4 Projekti kood

Projekti koodi kirjutamisel jätkatakse heade tavade kasutamist, mida on juba teinud varasemad rakenduse arendajad. Aluseks võeti sama raamat, mis eelmiseski iteratsioonis. [17]. See on oluline sellepärast, et tulevikus rakenduse arenduse üle võtvad arendajad saaksid koodi, mille osad täidavad kindlat eesmärki, on hästi loetav ning täidab käske efektiivselt, raiskamata arvuti ressursse. Ka selles iteratsioonis järgitakse rohkesti puhta koodi põhimõtteid.

Eriti tähelepanu pöörati ning võimalusel likvideeriti:

- asjakohastele nimedele, mis pidid olema muutujatel, funktsioonidel ning klassidel
- funktsioonid võiksid olla võimalikud kompaktsed ning teha ühte asja
- Koodi korduvkasutamise vähendamine
- koodi vormindamine
  [17]

Koodi kvaliteedi kontrolli all hoidmiseks kasutati Intellij IDE'sse sisseehitatud võimsat koodi analüüsimise tööriista, mis tõi välja stiilivead ja muudatusettepanekud koodis.

3.4.1 Java *back-end* komponendi kood - äriloogika

Selles ja järgmises peatükis käsitletakse ükshaaval iga kolmanda iteratsiooni jooksul juurde tulnud rakenduse klassi ja kontrollerit. Tuuakse ülevaade csv faili üleslaadimisest, selle lugemisest, andmete töötlusest ning kuidas esitatud standardiseerimata aruannetest luuakse lõppkokkuvõttes standardiseeritud aruannetega grupid. Käiakse läbi uued REST API päringud ja nende andmebaasi salvestamise loogika.

Kolmanda iteratsiooni jooksul loodi viis uut ja peamist äriloogika klassi, mis teostavad eelpool nimetatud operatsioonid. Klassi nimetus viitab kontrollerile ja funktsionaalsusele, mida ta teostab läbi vastava Kontrolleri. Uued äriloogika klassid on: *ConfigCreation*, *CsvReaderImpl, StandardGroupOfStatsCreation, RawStatsCreation* ning *FileStorageService*. Klassid sisaldab endas privaatseid muutujaid, millele saadakse ligi väljastpoolt klassi OOP *encapsulation* printsiibi kohaselt ainult läbi getter ja setter meetodite.

FileStorageService.java klassis on realiseeritud csv faili originaalkujul vastuvõtmine läbi *FileController*'i ning selle salvestamine */resources/upload/files* kausta. Selle ülesanne on lihtne: salvestada administraatori poolt sisestatud csv fail õigesse kausta ning luua URL, millele navigeerimisel on võimalik varasemalt administraatori poolt lisatud fail alla laadida. *Storefile* meetod teostab oma tööd try-catch ploki sees, sest failide salvestamisel võib tulla csv faili lugemise veateade *IOException*. Esialgu kontrollitakse, kas faili nimi on korrektsel kujul ja luuakse koopia sisestatud failist õigesse *upload/files* kausta. *LoadFileAsResource* tagastab soovi korral *upload/files* kaustas asuva csv faili originaalsel kujul, võimaldades csv faili alla laadida.

CsvReaderImpl klassis on realiseeritud csv faili lugemine ja tulemuste tagastamine. Tähtis on see, et csv fail oleks korrektsel kujul varasemalt üles laetud. Faili korrektsuse kontrolli tuleb teostada administraatoril ise. Selle kohta on tehtud õpetus: vt lisa 5. *Parser* väli on vajalik selleks, et objekti loomise etapis loodaks korrektne konfiguratsioon csv faili lugemiseks. Nüüd on klass valmis lugema csv faili ning vastavalt äriloogikale andmeid töötlema.

Meetodid sellel klassil on järgmised. *isCurrentRowStatementBlock* nagu ka edaspidi enamik meetodeid võtab vastu hetkel töötluses oleva rea csv failist ning kontrollib, kas tegemist on aruande algusega. *determineCurrentSpecificStatementBlock* sätib muutuja globaalse välja *currrentStatement*, millest sõltub järgnev äriloogika õigetesse kollektsioonidesse aruande ridade asetamine. *isEndingStatementBlock* meetod kontrollib, kas tegemist on reaga, mis lõpetab ära viimase aktiivse aruande. *doesHeaderColContainNote* kontrollib, kas csv faili päis sisaldab „Note" või sellele sarnast teksti. Kui jah, peab sellega edaspidi arvestama, sest faili struktuur on veidi teistsugune. *addFieldToCurrentStatement* lisab loetud rea õigesse kollektsiooni. *addNoteToFieldIfNecessary* meetod lisab note'i identifikaatori atribuudi väärtusele ise, et see veerg ei läheks kaduma. *determineUnnecessaryCols* võtab sisendiks ühe aruande kollektsiooni ning tühja kollektsiooni indeksitest, mille veerud peaks eemaldama. Meetod lisab varemalt tühja kollektsiooni indeksid ehk veerud, mis peaks eemaldama. *removeUnnecessaryCols* lõpetab töö, eemaldades reaalselt varem identifitseeritud read, mille tuleb eemaldada. *DetermineAndRemoveUnnecessaryCols* meetod teeb korraga kogu vajaliku funktsionaalsuse.

Suurem meetod on *readCsvAndReturnLists*. Meetod loeb läbi csv faili read samm-sammu haaval, kasutades õiges järjekorras varasemalt mainitud meetodeid. Lisab aruande read õigetesse aruanne kollektsioonidesse ning hiljem tagastab kollektsiooni kolmest aruande kollektsioonist (*incomeList*, *cashflowList*, *balanceList*).

Eelnevalt loetud ja kollektsioonidesse salvestatud info aruannete kohta tuleb kindla struktuuri järgi salvestada andmebaasi. Selle funktsionaalsusega tegeleb äriloogika klass *RawStatsCreation*. Sellel klassil on ligipääs neljale repositooriumile: *CashflowStatRawRepository*, *BalanceStatRawRepository*, *IncomeStatRawRepository* ning *GroupOfStatementsRepository*. Ligipääs andmebaasile, uute objektide lisamise ning olemasolevate manipuleerimisega teostatakse repositooriumite teel. Kuna esitatud csv failid sisaldavad mitme perioodi aruanded, siis tuleb läbi käia kõik perioodid.

Klassis on olemas erinevad meetodid, mis toetavad andmetöötlust ja seejärel andmete õigetesse tabelitesse salvestamist. *parseNumToDouble* meetod konverteerib sõne väärtuse numbriliseks tüübiks. *iterateDataLinesAndCreateFinStatementsAttrs* loob atribuudid hetkel töötluses oleva aruande jaoks. Kuna esitatud standardiseerimata aruannete read ei ole võimalik ette planeerida, ei saanud nende väärtuste salvestamise lahendada traditsioonilisel teel, tehes iga väärtuse jaoks andmebaasi veerg. Selleks, et salvestada kindla aruande jaoks raamatupidamiskirje, luuakse *Entity-Attribute-Value* mustrit [18] kasutades Attribute objekt, millel on *field_name* ja *value*. Hiljem seotakse kindla standardiseerimata aruandega kõik atribuudid läbi selleks eraldi vahetabeli. Meetod *createNewFinStatementForSpecPeriod* võtab sisendiks kõik olemasolevad kindla aruande perioodid, mille alusel tehakse iga perioodi jaoks eraldi standardiseerimata aruande objekt. Peale perioode on vaja ka atribuutide listi ise, et käivitada meetodi keskel teise seotud meetodi *iterateDataLinesAndCreateFinStatementsAttributes* ning mis aruande tüübiga on hetkel tegemist, et meetod teaks, mis tüüpi aruande standardiseerimata objekte luua. Selle sama meetodi tulemusena luuakse uued standardiseerimata aruanded.

EAV kasutamise peamine eelis on selle paindlikkus. Mudeli üksust kirjeldavad atribuudid ei piirdu kindla veergude arvuga, mis tähendab, et see ei nõua skeemi ümberkujundamist iga kord, kui on vaja uue atribuudi loomine. Atribuutide arv võib andmebaasi arenedes vertikaalselt kasvada ilma, et struktuur muutuks [18]. Selline lahendus sobis väga hästi standardiseerimata aruannete kirjete jaoks, sest administraator ei saa kunagi ette teada,

kuidas ettevõtte enda raamatupidamiskirjeid nimetab. Hiljem saab administraator luua valemid, tänu millele seotakse standardiseerimata aruande read standardiseeritud aruannete ridadega.

Veel on olemas kaks lisa meetodit, millest üks on *findMaxLengthOfStatementsInFile*. See on abimeetod, mille ülesandeks on leida maksimaalse aruannete perioodide pikkuse csv failis. *createGroupsOfStatementsForCompany* teostab gruppide koostamist, mis on vajalikud standardsete aruannete gruppide tegemiseks. Töö autorid on leidnud, et aruandeid ei tasuks vaadata ükshaaval vaid gruppidena, sest nad on csv failides asetatud nii, et neid võib vaadata tervikuna. Nende põhjal koostatud standardiseeritud aruannete grupi põhjal annab teha arvutusi firma finantskordajate jaoks.

Järgmine klass tegeleb konfiguratsiooni objektide loomisega ning selle nimetus on *ConfigCreation*. Tegemist on äriloogika klassiga, mille ainuke eesmärk on vastu võtta tagarakendusele saadetud konfiguratsioone JSON formaadis ning teha neist objekt, mis salvestatakse andmebaasi. Olemas on kolm meetodit, mille idee on sama. *setIncomeConfigObjectFields*, *setCashflowConfigObjectFields*, *setBalanceConfigObjectFields*. Meetodid võtavad vastu eelnevalt loodud tühja objekti ning tagarakenduse poolt kinni puutub JSON formaadis kätte saadud info *requesti* kohta. Meetodi sees toimub päringu lahti pakkimine ning konfiguratsiooniobjekti väljade täitmine.

Kõige viimasena lisatud äriloogika klass on *StandardGroupOfStatsCreation*, millel on oluline roll kogu uue lisatud funktsionaalsuse juures. Seal peitud kogu *Spring Expression Language*'i kasutuselevõtu idee. Kogu selle klassi mõte on teostada vastavad funktsioonid ning lõpptulemusena väljastada grupp kolmest standardiseeritud aruandest, millega saavad investorid tulevastes rakenduse iteratsioonides tutvuda. Nende alusel arvutatakse ka finantskordajad ettevõttete jaoks, mis on hetkel lisatud *mock* andmetena. Kolmanda iteratsiooni skoopi see kahjuks ei mahtunud piiratud ajaressurssi ning projekti liikmete arvu tõttu. Küll aga mõeldi rohkelt selle funktsionaalsuse peale ning tehti ette *endpointe* andmete küsimiseks, mis peaks lihtsustama järgmise arenduse iteratsiooni tööd. Lähemalt sellest räägitakse töö edasiarenduse võimaluste peatükis.

Klassis eksisteerib rohkelt muutujate välju. *Parser* [19] vastutab väljendussõnede parsimise eest. Parsimine kujutab endast ette sisendi analüüsi andmete korrastamiseks

vastavalt grammatika reeglitele. Parsimise määratlemiseks on mõned võimalused. Ent põhisisu jääb samaks: parsimine tähendab meile antud andmete aluseks oleva struktuuri leidmist. Mõnes mõttes võib parsimist nimetada mallide pöördteisenduseks: struktuuri tuvastamiseks ja andmete väljavõtmiseks. [20].

„Parsimine vajalik, kuna erinevad süsteemi olemid vajavad, et andmed oleksid erinevas vormis. Parsimine võimaldab andmeid teisendada viisil, mis on konkreetsele tarkvarale arusaadav. Hea näide on programmid, mis on kirjutatud inimeste poolt, kuid nad jooksutatakse arvutite abil. Nii kirjutavad inimesed teksti neile arusaadavas vormis, seejärel interpreteerib tarkvara teksti sel viisil, mida arvuti oskab kasutada. [20]

Projektis võeti kasutusele Spring Expression Language teegi (edaspidi *SpEL*), kuna see toetab kõiki meile vajalikke aritmeetilisi tehteid: liitmine, lahutamine, korrutamise, jagamine ning jälgib sulge. Vajadusel saab juurde luua ka oma matemaatilisi operaatoreid, näiteks ruutjuure võtmiseks. [21] Kõik see on vajalik ka edaspidi siis, kui jõutakse erinevate firma finantsnäitajate arvutamiste juurde standardiseeritud aruannete grupi põhjal.

Rakendus on loodud nii, et konfiguratsiooni loomisel tuleb iga standardiseeritud välja juure kirjutada valem, kuidas standardiseeritud väli arvutatakse. Valemi sees saab kasutada standardiseerimata aruandes olevaid *field_name*'e muutujatena, mis osalevad arvutustes. Valemitel on oma kuju ning selle õpetus on olemas lisades. Vt lisa 4.

SpEL koosseisu kuulub ka stContext, mida kasutatakse valemite loomisel ning arvutustehetes [22]. Iga loodava objekti jaoks tehti oma context: *stContextBalance*, *stContextCashflow*, *stContextIncome*.

EvalutationContext *interface*'i kasutatakse avaldise hindamisel omaduste, meetodite, väljade lahendamiseks ja tüübi teisendamiseks. StandardEvaluationContext on koht, kus määrata töös olevat aruande objekti. Võimalus on ka ise määrata muutujaid, mida hakatakse *expression*'ites kasutama. [22]

Lisaks eelnevale, on globaalsete muutujatena lisatud rohkelt standartsete aruannete välju String andmetüübina, sest kõigile neile luuakse ettevõtte ja kindla ajaperioodi spetsiifile aruande konfiguratsioon. Näiteks toon välja *revenueString* ja *grossProfitString*. Selleks, et kõiki neid standardsete aruannete nimelisi välju valemitena hoida, on loodud kolm

kollektsiooni: *incomeStandardFieldFormulas, cashflowStandardFieldFormulas, balanceStandardFieldFormulas*.

Klassi initsialiseerimisel luuakse tühjad kollektsioonid, kuhu asetada valemeid kindlatele aruande väljadele. Seejärel järgneb meetod *createStContexts*, mille tulemusena luuakse *balanceStatement, cashflowStatement, incomeStatement* tühjade standartsete aruannete jaoks kontekst, tänu millele saab hakata teostama *SpEL* manipulatsioone ja arvutusi nende objektide väljadega. *populateContextsWithValues* võtab sisendina erinevate aruannete atribuutide kollektsioonid ning lisab teostades meetodi *createAttributeWithValuesContext* igale kontekstile. Selle järel on atribuudid saadaval erinevatel aruannete kontekstides ning nende abil saab teostada aritmeetilisi tehteid.

createValuesForStatementFromFormulas meetod võtab sisendiks valemid kindlale aruandele ja lisaks veel selle konteksti. Selles meetodis toimub enamik *SpEL* võlust ning tänu sellele saavad algselt tühjad standardsed aruanded valemite põhjal välja arvutatud väärtused.

createIncomeString, createCashflowStrings, createBalanceString loovad valemite struktuurid, mida siis *createValuesForStatementFromFormulas* meetodis kasutatakse. Selle näide on järgmine: "*revenue=#Revenue + #Other_income*". Lisaks on veel abimeetodid *createListOfIncomeStrings*, *createListOfCashflowString*, *createListOfBalanceString*, mis lisavad individuaalsed valemid kollektsioonidesse, et nendega oleks lihtsam tegeleda.

Teine suurem osa, millega klass tegeleb on standardse aruannete grupi tegemine, mida hilisemates arendustes läheb vaja, et kuvada standardseid andmeid kindla kuupäeva seisuga ning välja arvutada firma finantskordajad nende aruannete alusel.

findRightGroupOfStatements leiab üles standardiseerimata aruannetest koosneva grupi, mille bilanss on koostatud otsitava kuupäevaga. See on vajalik selleks, et teha selle alusel standardiseeritud aruannetes grupp. *findRightBalalanceConfig* tegeleb sellega, et leiab üles õigel ajavahemikul kehtiva bilansi konfiguratsiooni, et seda kasutada hiljem standardiseeritud aruande väärtuse arvutamiseks. *findRightConfig* teeb nii, et leitakse muud aruanded, mis kuuluvad eelnevalt leitud bilansi konfiguratsiooniga samase gruppi. Näiteks kui luuakse teine komplekt konfiguratsioone firma jaoks, mis kehtivad aastas

2014 kuni 2018, siis igal aruande konfiguratsioonil on config_collection_id võrdne kahega.

createStatement on meetod, mis tegeleb standardse aruande tegemisega. Sellele meetodile antakse sisendiks õige konfiguratsioon ning info selle kohta, mis aruanne tuleb teha. Selle meetodiga saadakse kätte objekt kõikide välja arvutatud ridadega..

Lõpuks jõutakse standardse grupi koostamise juurde, kus meetod *createGroupUsingPreviouslyFoundData*, mille sisendiks on standardiseeritud aruanded ise ning firma, millele see kuulub. Meetod tagastab lõpliku grupi standardsetest aruannetest.

3.4.2 Java *back-end* komponendi kood - kontrollerid

Kasutajaliides teostab päringuid tagarakendusele kasutades selleks REST API arhitektuuri. Eelnevates rakenduse versioonides oli võimalik küsida andmeid kõikide ettevõtete kohta kasutades GET */companies* päringut. GET */companies/{tickerId}* päring tagastas kindla ettevõtte andmed JSON'I formaadis. Täpsemat infot leiab eelmise rakenduse iteratsiooni tööst.

Hiljem lisandus teise iteratsiooni käigus viis päringut, millest uued olid POST ja DELETE. Need lisandunud päringud on senimaani kättesaadavad *AuthenticationController.java ja UserController.java* klassides. Täpsemat infot nende kohta leiab eelmise iteratsiooni lõputööst.

Kolmanda iteratsiooni käigus loodi 25 uut päringut, millega tegeleb 9 kontrollerit. Uued kontrollerid on: *ConfigurationController*, *GroupOfStatementsController*, *GroupOfStandardStatementsController*, *RawBalanceSheetController*, *RawCashflowStatController, RawIncomeStatController, CsvController, FileController*, *SourceCsvFileController*.

ConfigurationController.java:

- POST */configuration/create/{ticker}/income* lisab üksiku kasumiaruande konfiguratsiooni ettevõtte kasumiaruannete konfiguratsioonide hulka. Sisendiks võetakse *IncomeRequest* JSON objekt, mis sisaldab kogu infot, mis on vaja uue konfiguratsiooni loomiseks.

- Päringut kasutatakse kasutajaliideses uue konfiguratsiooni loomiseks. Neid saab olla mitu. Iga perioodi jaoks erinev.
- Edu korral kuvatakse ekraanil *"Mapping created, check if income configuration containing new formulas has been created"*

- POST */configuration/create/{ticker}/cashflow* lisab üksiku rahavoogude aruande konfiguratsiooni ettevõtte rahavoogude konfiguratsioonide hulka. Sisendiks võetakse *CashflowRequest* JSON objekt, mis sisaldab kogu infot, mis on vaja uue konfiguratsiooni loomiseks.
  - Päringut kasutatakse kasutajaliideses uue konfiguratsiooni loomiseks. Neid saab olla mitu.
  - Edu korral kuvatakse ekraanil *"Mapping created, check if cashflow configuration containing new formulas has been created"*

- POST */configuration/create/{ticker}/balance* lisab üksiku bilansi konfiguratsiooni ettevõtte bilansi konfiguratsioonide hulka. Sisendiks võetakse *BalanceRequest* objekt, mis sisaldab kogu infot, mis on vaja uue konfiguratsiooni loomiseks.
  - Päringut kasutatakse kasutajaliideses uue konfiguratsiooni loomiseks. Neid saab olla mitu.
  - Edu korral kuvatakse ekraanil *"Mapping created, check if balance configuration containing new formulas has been created"*.

POST */configuration/update/{ticker}/[*balance;cashflow;income*]/{id}* võimaldab olemasolevat konfiguratsiooni uuendada.

- Päringut kasutatakse siis, kui otsustatakse mõnda konfiguratsiooni valemit muuta. Hiljem võib kohe uuesti arvutada standardiseeritud väärtused arvestades uuendatud valemeid.
- Edu korral kuvatakse ekraanil *"[statementName] configuration updated"*.

GroupOfStandStatementsController.java:

- GET */groupOfStandardStatements/* päring tagastab kõik andmebaasis olevad grupid standardiseeritud andmetest.

- GET */groupOfStandardStatements/*{id} päring tagastab id järgi grupi standardsetest aruannetest.

- GET */groupOfStandardStatements/for/{ticker_id}* päring tagastab andmebaasist kõik standardiseeritud aruannete grupid, mis kuuluvad firmale, mille id on {ticker_id}

- GET */{ticker}/create/forPeriod/{balance_date}/* päring loob standardiseeritud grupi, standardiseerimata aruannete grupist, kasutades korrektseid eelnevalt loodud konfiguratsioone. Kontrolleri sisendiks on firma lühinimi {ticker} ja bilansi kuupäev, mille standardiseerimata gruppi me soovitakse leida.

    - Päringut kasutatakse selleks, et luua täielikust ehk standardiseerimata firma aruannete grupist, mille kõik standardiseerimata aruanded on olemas, standardiseeritud grupp.

    - Edu korral kuvatakse kasutaja kaks võimalikku sõnumit

        - Kui standardiseeritud gruppi pole veel olemas, luuakse uus ning kasutajale kuvatakse status.ok ning tekst *"Group of standard statements created"*.

        - Kui standardiseeritud aruannete grupp on juba olemas, siis selle finantsaruannete väärtused uuendatakse vastavalt kehtivatele aruannete konfiguratsioonidele.

    - Mingis etapis veateate tõttu programmi töö lõpetamise kohta antakse infot.

        - Kui ei leita gruppi standardiseerimata aruannetest, mille bilanss on {balance_data} kuupäevaga, siis kuvatakse: "*Couldn't find suitable balance configuration. Does it exist*?"

        - Kui ei leita rahavoogude või kasumiaruande konfiguratsioone, siis kuvatakse: *"Couldn't find suitable cashflow or income configuration. Does two of them exist?"*

GroupOfStatementsController:

- GET */groupOfStatements/* päring tagastab kõik andmebaasis olevad standardiseerimata grupid, mis sisaldavad endas standardiseerimata aruandeid: kasumiaruanne, rahavoogude aruanne, bilansi aruanne.

- GET /groupOfStatements/{id} päring tagastab id järgi grupi standardiseerimata aruannetest.

- GET */groupOfStatements/onlyFullGroups/{ticker_id}* päring tagastab andmebaasist kõik standardiseerimata aruannete grupid, mille kõik kolm aruannet on olemas (ükski ei ole null) ning mis kuuluvad firmale, mille id on {ticker_id}

CsvController:

- GET */csv/readAndSave/{ticker}/{fileName}* päring võimaldab lugeda rakenduse *resources/upload/files* asuvat originaalset csv faili ning salvestada organiseeritud selle tulemused andmebaasi tabelitesse.
    - Päring saadetakse kohe, kui csv fail on üles laetud rakendusse.
    - Edu korral tagastatakse teade *"As is statements are stored to database successfuly."*.
    - Kahel juhul võib kontroller enda töö peatada.
        - Siis kui selle nimega fail on juba varasemalt olnud sisse loetud. Tagastatakse veateade *"File is already in the database. No need to add it a second time."*
        - Juhul kui ühtegi aruannet ei õnnestu kätte saada, siis kuvatakse kasutaja veateade: *"Something went wrong with reading in values from CSV file. Result contains null"*

FileController:

- POST */uploadfile* võimaldab lisada üksiku faili vormi sisse ning tagarakenduses originaalsel kujul vastu võtta.
    - Päringut kasutatakse administraatori poolt. Administraator saab valida firma ja lisada selle standardiseerimata aruandeid lihtsalt üles laadides firma poolt esitatud csv faili, mis on algselt viidud korrektsele kujule (vt lisa 4).
    - Edu korral kuvatakse ekraanil csv faili allalaadimise link.
- POST */uploadMultipleFiles* võimaldab teostada sama funktsionaalsust, mis *uploadfile*, kuid see *endpoint* võtab vastu juba mitu faili.
- GET */downloadFile/{fileName:.+}* võimaldab varasemalt rakendusse lisatud csv faili alla laadida. See võib olla kasulik investoritele, kui nende jaoks saavad tulevastes iteratsioonides kasutajaliidese vaated valmis.

SourceCsvFileController:

- GET /*sourceCsvFiles* päring tagastab kõik andmebaasis registreeritud varasemalt üles laadidud csv failid. Neil on olemas nimi ning nendega seotud standardiseerimata aruanded.
- GET */groupOfStandardStatements/*{id} päring tagastab id järgi csv faili andmed.

Käesoleva rakenduse iteratsiooni tulemusena saab osa tagarakenduse funktsionaalsust kasutada läbi Postmani päringute. Exporditud päringute faili leiab tagarakenduse projekti *resources* kaustas. Päringute näited leiab lisa 7.

Ka selle iteratsiooni lõpus kasutatakse siiamaani teises iteratsioonis programmeeritud: “andmetabelite osaliseks täitmiseks *data.sql* faili, mis täidavad andmetabelid *roles, company_dimension, financials_daily* ja *financials_quarterly*” [2].

Rakenduses on jäetud selline teise arenduse jooksul loodud Spring Boot konfiguratsioon, mille korral iga käivitamise aeg kustutatakse vanad ja luuakse uued andmetabelid ning seejärel lisatakse uued andmed. See on vajalik selleks, et andmebaasi tabelite

edasiarendust teostada ka järgmistes iteratsioonides. Ilma selleta võivad tekkida dubleeritud *mock* andmed. [2]

3.4.3 *Front-end* komponendi kood

3.4.3.1 Failide üleslaadimine

Et mõne konkreetse börsiettevõtte profiilile lisada majandusettekanded, tuleb need kõigepealt süsteemi importida, navigeerides klientrakenduses vaatele 'Upload Data'. Seal tuleb valida üleslaetav fail ja sisestada börsiettevõtte sümbol, kelle profiiliga andmeid soovitakse seostada. Sümbolit võib sisestada nii väikeste kui suurte tähtedega.

Joonis 17. Failide importimise vaade

Imporditavate failide formaadinõuded on välja toodud lisa 5 all.

Failide üleslaadimise jaoks tuleb tagarakendusele praegu reaalsuses teha kaks järjestikkust päringut, kuid seda praegune PrimeReacti pakutav failide üleslaadimise komponent ei võimaldanud. Oleks võinud kasutada mõne muu kolmanda osapoole failide üleslaadimiskomponenti, aga otsustati, et pikas perspektiivis on parem hoida kolmandate osapoolte teekide, millest klientrakendus sõltub, hulka võimalikult madalal. Seega tuli valmis kirjutada spetsiaalne failide üleslaadimise komponent, mis saadaks tagarakendusele kaks järjestikkust päringut. Esimene päring saadab CSV-faili spetsiaalsesse hoidlasse. Teine päring annab käsu failis sisalduva info töötlemiseks ja relatsioonilisse andmebaasi salvestamiseks.

Failide üleslaadimise vaate eest vastutab klientrakenduses DataImportComponent. Komponendi sees on näha funktsiooni *uploadHandler()*, mis kutsutakse välja, kui kasutaja vajutab vaates nupule *Upload*. Funktsiooni raskuspunktiks osutus faili jaoks sobiva objekti loomine ning selle objekti saatmine POST-päringuga. Autorid pidid ennast kurssi viima MIME standardiga RFC 2045 [23]. Selle jaoks, et faili saatmine

tagarakendusse töötaks korrektselt, tuli POST-päringusse lisada päringupäised *„Content-type:multipart/form-data“* ja *„Accept:*/*“*

```
function uploadHandler(){
    let fileName = fileContent.name.split(".",1)[0];
    let formData = new FormData();
    let file = new File([fileContent], fileContent.name, {type: "text/csv"});
    formData.append("file", file);
    axios.post(API_URL + "/uploadFile", formData, {
        headers:{
            "Content-Type": "multipart/form-data; boundary=" + FORMDATA_BOUNDARY,
            "Accept": "*/*"
        }
    }).then(axios.get(API_URL + "/csv/readAndSave/" +
    companySymbol.toUpperCase() + "/" + fileName));
}
```

„*Content-Type*“ päis edastab juhised päringu sisu kodeerimiseks ja ka lugemiseks. Selle päringu oluliseks atribuudiks on „*boundary*“, mis deklareerib päringuga saadetud infoobjektide piirid, kust lõppeb ühe objekti informatsioon ja kust algab teise objekti informatsioon. Kuigi funktsioonis *uploadHandler()* saadetakse päringuga ainult üks objekt nimega *formData*¸ peab see ikkagi olema piiritletud atribuudis „*boundary*“ deklareeritud piirdega.

Selle dokumendi valmimise seisuga jäeti „*boundary*“ väärtuseks „xxxxxxxxxxxxxxxx“. „*boundary*“ väärtus peab olema selline, mis mingil moel ei esine päringu sisuga saadetatavate objektide (antud kontekstis „*formData*“) sees. Valitud väärtus võeti aja kokkuhoidmise eesmärgil katse-eksituse meetodil ning praegu tundub, et see sobib. Sellegipoolest on probleemide tekkimise korral soovitatav tulevastel töö edasiarendajatel uurida antud valdkonna kohta ja valida uus väärtus mõne levinud ja tunnustatud meetodi põhjal, mitte jätkata katse-eksitusmeetodiga.

„*Accept*“ päis teatab tagarakendusele, mis formaadis informatsiooni on klientrakendus valmis vastu võtma. Lihtsuse mõttes jäeti praegu väärtuseks „*/*“, mis näitab, et klientrakendus on vastu võtma mis tahes formaadis informatsiooni, kuid ka seda on soovitatav tulevastel arendajatel muuta, et tõsta klientrakenduse töökindlust ja mingil määral ka turvalisust.

Koodist on näha, kuidas peale esimest päringut paikneb teine päring axiose meetodi *then()* sees. Kuna päringute saatmine tagarakendusse on asünkroonne protsess, võiks juhtuda olukord, kus server hakkab esimese päringuga tegelemise kõrvalt samaaegselt ka teise päringuga tegelema. Teise päringu eelduseks on aga, et esimene päringuga tegelemine oleks täielikult lõppenud. Seepärast tuli teine päring mähkida meetodi *then()* sisse.

3.4.3.2 Andmetabeli komponent

Mahu poolest kõige keerulisem komponent, millega valmis saadi, oli andmetabeli komponent (DataTableComponent). Andmetabeli komponendi ülesanne on luua üldine HTML-tabeli struktuur ja renderdada tabeli päised.

Andmetabel toetab ühe- ja kahedimensioonilisi andmehulkasid.

Profit for the period	*9764*
Depreciation and amortization of non current assets	*2655*
Share of profit of equity accounted investees	*-3*
Gains / losses on the sale of PPE and IA	*23*
Net finance income / costs	*-2546*
Provision for impairment losses on trade receivables	*0*
Provision for long term benefits	*5*
Income tax expense	*2875*
Change in inventories	*241*

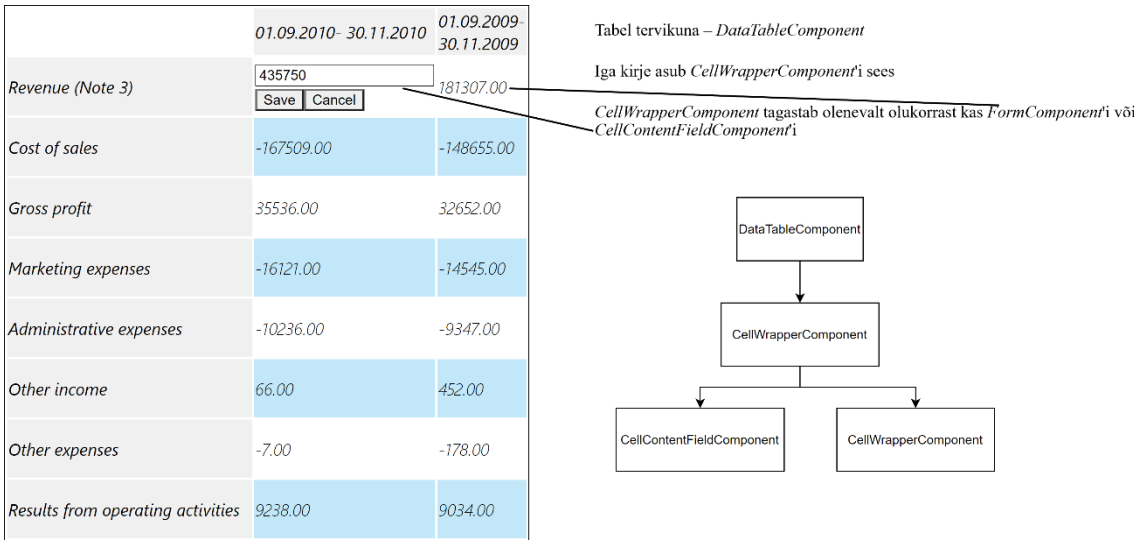
Joonis 18. Ühedimensiooniline tabel

	01.09.2010-30.11.2010	01.09.2009-30.11.2009
Revenue (Note 3)	203045.00	181307.00
Cost of sales	-167509.00	-148655.00
Gross profit	35536.00	32652.00
Marketing expenses	-16121.00	-14545.00
Administrative expenses	-10236.00	-9347.00
Other income	66.00	452.00
Other expenses	-7.00	-178.00
Results from operating activities	9238.00	9034.00

Joonis 19. Kahedimensiooniline tabel

Andmetabeli igale päisele (kahedimensiooniliste andmete puhul veeru- ja reapäiste kombinatsioonile) kuuluva väärtuse renderdamine on delegeeritud CellWrapper komponendile, kus sees kasutatakse *Conditional Rendering tehnikat* [24]. CellWrapper on tabeli reaalse kirje väärtust ümbritsev komponent, mis otsustab, kas selles tabeli lahtris tuleks kuvada tavaline tekst (CellContentFieldComponent) või teksti sisestamise väli (FormComponent). Vaikimisi kuvatakse tabelis kõik väärtused tavatekstina, aga kui

kasutaja klõpsab mõne kirje peale, asendab CellWrapper tagataustal tavateksti teksti sisestamise väljaga, kus kasutaja saab muuta selle konkreetse kirje väärtust. Kui väärtus muudetakse ja muudatused salvestatakse, uuendatakse andmetabeli komponendi olekus vastav kirje, mille tulemusena renderdatakse muudetud tabeli lahter uue väärtusega.

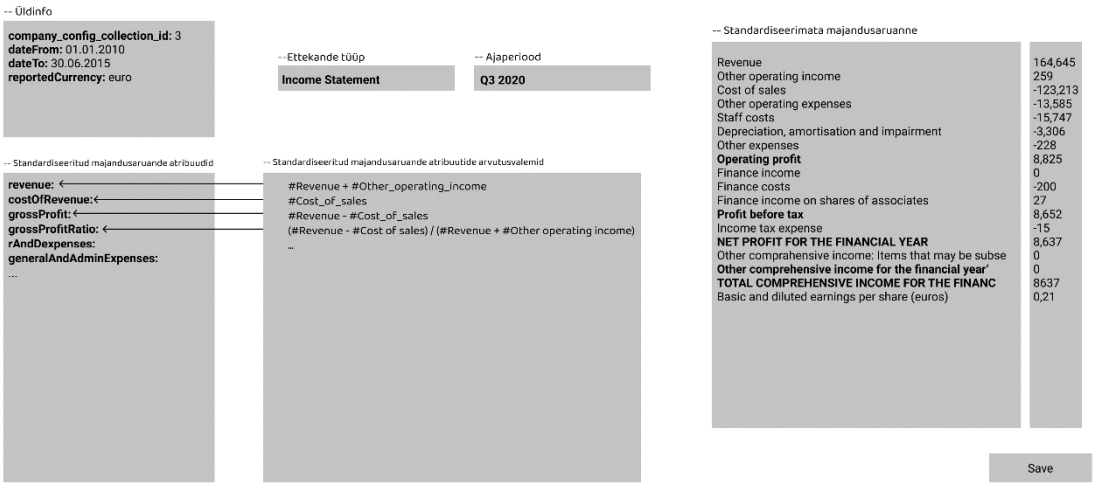


Joonis 20. Tabeli alamkomponentide selgitus ja komponentide hierarhia

Kuna andmetabel võib koosneda väga paljudest ridadest ja tulpadest, mis kõik võivad muutuda, oli oluline igale kirjele välja mõelda sobiv võti [25]. Kui on võimalik adresseerida tabeli igat unikaalset kirjet, siis saab React kirje muutmisel säästa meeletult aega, taasrenderdades ainult muutunud väärtusega kirje. Ilma unikaalse adresseerimiseta tuleks näiteks ühe kirje muutmisel taasrenderdada terve tabeli sisu. [26]

3.4.3.3 Failide konfigureerimise komponent

Iga ettevõtte majandusaruanded omavad üldjuhul mõningaid nüansse, mistõttu võib investorile analüüsi käigus erinevate firmade üleslaetud majandusettekannete võrdlemine tekitada ebamugavusi. Failide konfigureerimise komponendi *DataConfigurationComponent* ülesanne on pakkuda vaadet, kus administraatorkasutaja saab firma üleslaetud majandusettekanded kaardistada standardiseeritud ettekanneteks *Spring Expression Language*'it kasutades [21]. See vaade jäi aja puuduse tõttu pooleli, aga allpool on ikkagi toodud välja, mis sai valmis, ning mida sellele vaatele kavandati.



Joonis 21. Konfiguratsioonivaate idee

Tuleb mainida, et sellise konfiguratsioonivaate realiseerimine eeldaks ettevõtte profiili administraatorvaate olemasolu. Praeguses staadiumis taoline vaade puudub ning seetõttu lisati arendamisjärgus konfiguratsioonivaatele tekstiväli, kuhu tuleb sisestada börsiettevõtte sümbol, ja nupp, millele vajutades tuuakse komponendi olekusse vastava ettevõtte informatsioon. Seejärel saab soovitud ettevõttele luua standardiseeritud majandusettekandeid.

- Üldinfo näitab firmaga seonduvaid üldiseid andmeid.
- Ettekande tüüp ja ajaperiood on *Dropdown* menüüd, kus saab vastavalt valida loodavat standardiseeritud ettekande tüüpi ja ajaperioodi, millele see standardiseeritud ettekanne kehtima hakkab.
- Standardiseerimata majandusaruande komponent näitab valitud ettekande tüübi ja ajaperioodi põhjal vastavat standardiseerimata majandusaruannet.
- Standardiseeritud majandusaruande atribuudid on kõik eeldefineeritud standardsete ettekannete väljad ning nende kõrval paikneb teksti sisestusväli, kuhu saab vastava välja jaoks kirjutada *Spring Expression Language*'iga valemi, kuidas seda välja standardiseerimata aruandest arvutada

Valmis saadi ettekande tüübi ja ajaperioodi *Dropdown*-menüüdega, kus toorikuna sai kasutada PrimeReact'i *Dropdown*-komponenti. Võimalikke ettekande tüüpe on igal ajahetkel 3: bilanss, rahavoogude aruanne ja kasumiaruanne. Ajaperioodiks saab praegu

valida neid perioode, mille kohta on üles laetud standardiseerimata majandusaruandeid ehk kui näiteks laetakse üles standardiseerimata kasumiaruanne 2020. aasta 3. kvartali kohta ja standardiseerimata bilanss 2020. aasta 2. kvartali kohta, siis standardiseeritud kasumiaruannet saab samuti luua ainult 2020 aasta 3. kvartali kohta ja standardiseeritud bilanssi saab luua ainult 2020. aasta 2. kvartali kohta.

Standardiseerimata ja standardiseeritud aruannete kuvamiseks sobib hästi projekti jaoks loodud andmetabeli komponent, nende sektsioonide implementeerimise ülesanne lasub tulevatel klientrakenduse arendajatel. Vajalik on tabelite olekust kätte saada ja moodustada andmeobjektid, mida tagarakendustele päringutega saata, et standardiseeritud aruanded serverisse salvestada.

Samuti jääb järgmiste arendajate ülesandeks komponentide paigutamine lehele CSS-kujunduskeelt kasutades.

3.5 Projekti komponentidele kirjutatud testid

Selle iteratsiooni käigus planeeriti luua võimalikult laia katvusega testid tagarakendusele ning vajalikumad testid kasutajaliidesele. Kuna arenduse lõpupoole tegeleti funktsionaalsuse juurde arendusega, otsustati, et testimine viiakse lõpuni enne lõpuprojekti kaitsmist. Tagarakenduse äriloogika testitakse *unit testidega* ning peamised kontrolleride *entpoint*'id samuti. Selle jaoks jätkati kasutamist @SpringBootTest annotatsiooni ning JUnit5 testimise *teeki.*

Antud rakenduse esimeses versioonis tehti testid CompanyDimensionControlleri jaoks ning selle arenduse käigus lisati uued testid. Teise iteratsiooni jooksul lisandusid *AuthenticationController, UserController ja Calculator* testid, mida ei täiendatud.

Selle arenduse käigus lisanduvad kaitsmise hetkeks testid järgmistele äriloogika klassidele: *ConfigCreation, CsvReaderImpl, StandardGroupOfStatsCreation, RawStatsCreation* ning *FileStorageService.* Kontrolleritest testitakse *ConfigurationController, GroupOfStandardStatementsController, GroupOfStatementsController, CsvController* ning *SourceCsvFileController*.

4. Analüüs ja järeldused

Peatükk sisaldab kasutatud tehnoloogiate analüüsi ning miks just need võeti kasutusele. Lisaks pannakse kõrvale rakenduse iteratsiooni jooksul planeeritud kasutajalood, mille abil pandi kirja funktsionaalsed ja mittefunktsionaalsed nõuded, ning realiseeritud lõpptulemused. Analüüsitakse rakenduse arhitektuuri ning disaini. Lõpetuseks tuuakse välja edasiarenduse ideid, mida võiks tulevaste iteratsioonide käigus realiseerida.

4.1 Kasutatud tehnoloogiate analüüs

Rakenduse esimese demoversioonis võeti kasutusele Java Spring Boot raamistiku ja kasutajaliidese poolel JavaScripti React raamistiku. Selleks, et esialgne funktsionaalsus kiiresti tööle saada, kasutati PrimeReact'i komponentide kogu.

Kolmanda iteratsiooni töö autorid omasid vähest eelnevat kokkupuudet Java programmerimise keelega ning seetõttu oli nõuete täitmise produktiivsus esialgselt planeeritust väiksem. Küll aga see keel on piisavalt sarnane ülikoolis kolme aasta vältel õpitud C# programmeerimise keelega ja selle tõttu said autorid päris kiiresti hakata kvaliteetset koodi kirjutama. Mis puudutab kasutajaliidest, siis React raamistikuga puudus autoritel varasem kokkupuude. Selle tõttu kasutajaliidese osa ei jõudnud tagarakendusele järele. Samas leiti, et kasutatud tehnoloogiad on oma võimekuselt projekti tegemiseks rohkem kui sobivad ning välja vahetada esi- ja tagarakenduse programmeerimise keeled ei ole tarvis ka tulevastes rakendustes.

Projekti kolmanda iteratsiooni arendusfaasis loodi rohkelt äriloogika funktsionaalsust, mis toetab csv failide lugemist ning korrastatud kujul andmebaasi salvestamiseks. See on vajalik edaspidiseks töötluseks ning erinevatele lõppkasutajatele kuvamiseks. Kõige keerulisem funktsionaalsus antud arenduse puhul on *SpEL* kasutuselevõtt, mis võimaldab sisestada ja tõlgendada kasutaja poolt sisestatud kindlas formaadis valemeid. Valikuks osutus see raamistik, sest käsitsi valemite kirjutamine oleks ebamõistlik tegevus ning annaks mahult lõputöö iseenesest välja. Alternatiivsed lahendused ei arvesta aga *Spring boot* raamistiku eripärasid, mille peale on kogu projekt üles ehitatud. Seetõttu osutus

loogiliseks otsuseks projekti puhul kasutusele võtta spetsiaalselt selle valdkonna jaoks loodud võimas väljenduskeel, mis toetab andmete SQL päringute tegemist ja sellega manipuleerimist programmi *runtime*'i ajal [19].

Csv failist andmete lugemist teostati kasutades OpenCSV teeki [27] [28], mis on spetsiaalselt Java programmerimise keele jaoks loodud kolmanda osapoole raamisik, mille abil on võimalik teostada lihtsasti konfigureeritavat ning turvalist csv faili lugemist. Eelnevalt nimetatud omadused osutusid ka põhjuseks, miks ei kasutatud tavalist standardset Java Core teeki csv failide lugemiseks. Selliseid kolmanda osapoole *teek*'e on mitmeid, kuid kõige parem dokumentatsioon ja populaarsus tarkvaraarendajate hulgas on autorite arvates Apache Commons CSV ja OpenCSV'l. Kuna OpenCSV osutus kasutajasõbralikumaks, otsustati sellega jätkata [29].

4.2 Töö tulemuste analüüs nõete baasil

„Kasutajalood" peatükis mainiti kasutajalugusid, mida plaanitakse ära teha selle rakenduse iteratsiooni jooksul. Kasutajalugudena olid kirja pandud funktsionaalsed ja mittefunktsionaalsed nõuded. Edaspidi räägitakse sellest, kuivõrd palju jõuti soovitud funktsionaalsusest ära teha.

4.2.1 Funktsionaalsed nõuded kasutajalugudena

Esimene kasutajalugu oli see, et finantsteadmistega administraator saaks üles laadida standardiseerimata ettevõtte aruandeid läbi csv faili. See funktsionaalsus sai realiseeritud. Kasutades csv faili lugejat ja selleks mõeldud domeeni klassi võimaldati administraatoril lisada fail, mille järel see salvestatakse projekti ressursside kausta. Tänu sellele saab faili hiljem erinevate rollide poolt alla laadida originaalsel kujul. Investorile võib see olla kasulik selleks, et ta saaks detailsemalt uurida, kuidas on saadud standardiseeritud andmed standardiseerimata aruannetest. Kohe kui fail saab üles laetud rakendusse, hakkab rakendus seda lugema ning sisu andmetabelitesse salvestama.

Kuna failide hulk, mida sisestada süsteemi on suur, sooviti, et üles laadida oleks võimalik ka mitu faili. See funktsionaalsus osaliselt realiseeriti. Kuna React baasil üles ehitatud kasutajaliides oli meile uus nähtus, ei jõudnud me selle jaoks luua kasutaliidese vahelehte, kus saaks mitu faili korraga üles laadida. Hetkel on olemas tagarakenduses kontrolleri *endpoint*, mis võimaldab POST käsu abil saata mitu csv faili ning need failid salvestatakse

originaalsel kujul ressursside kausta. Nende sisu automaatne andmebaasidesse salvestamine ei toimu. Arvestades saadaolevat arendusressurssi, otsustasime, et üksikfaili üleslaadimisest peaks alguses piisama ning otsustasime suunata oma aja teiste vaadete peale.

Kasutajaloona oli kirjas, et administraator peaks saama ekraanile tagasisidet selle kohta, kui sisestatud fail on vales formaadis ning selle lugemine ning andmete andmebaasi salvestamine ei ole võimalik. Rakenduse arenduse käigus jõudsime järeldusele, et iga väiksemat pisiasja kontrollida faili formaadi puhul ei ole tarvis. Ülesanne hoida csv failid korrektse formaadina jääb finantsteadmistega administraatori vastutusalasse. Korrektse formaadi kohta leiab infot lisades: vt lisa 5. Rakendus oskab anda osalist tagasisidet faili lugemise kohta:

1) Fail on juba sisse loetud andmebaasi

2) Midagi läks valesti csv faili lugemisel, tulemuseks on tühi aruannete jada.

3) Andmed on edukalt salvestatud andmebaasi tabelitesse.

Järgmine nõue oli luua funktsionaalsus, kus administraator saab kaardistada csv failist kätte saadud standardiseerimata aruandeid standardiseeritud aruannetega. See oli kõige tähtsam ja suurimat pingutust nõudev ülesanne selle arenduse iteratsiooni jooksul. Funktsionaalsus sai edukalt loodud esialgsel kujul.

Lõpuprojektis on see funktsionaalsus realiseeritud nii, et administraatoril tuleb esialgu luua iga aruande jaoks konfiguratsioon, mida kasutatakse standardiseeritud aruannete loomisel standardiseerimata finantsaruannetest. Aruanded luuakse gruppidena, millele antakse valminud lahenduses kollektsiooni number, mille abil saab tuvastada komplekti kuuluvad konfiguratsioonid. Selleks, et genereerida standardiseeritud aruannete komplekt (grupp), on vaja luua kolm konfiguratsiooni (bilansi-, rahavoogude- ning kasumiaruande konfiguratsioon), mis kuuluvad sama kollektsiooni. See tähendab, et neil on sama kollektsiooni identifikaator.

Kõige tähtsam standardiseerimata aruannete grupi element on bilansi aruanne, mille järgi vaadatakse, mis konfiguratsiooni komplekti tuleks kasutada. Näiteks on üks konfiguratsiooni komplekt kehtiv alates 2014 kuni 2018 (vaadatakse bilansi kuupäeva

järgi). Sellisel juhul genereeritakse selle perioodi (nt 31.08.2016 või 31.08.2017) standardiseerimata aruannete komplektidest standardiseeritud komplektid just seda konfiguratsiooni komplekti kasutades. Selle iteratsiooni lõpuks on kogu konfiguratsioonide loomise funktsionaalsus realiseeritud tagarakenduses ning täielikult kasutatav läbi Postman'i tarkvara, mis imiteerib kasutajaliidest. Kahjuks React'ile mugava vahelehe selle funktsionaalsuse kasutamiseks ei jõudnud me täielikul kujul arenduse perioodi jooksul realiseerida. Seega tuleb kasutajaliidese osa valmis teha soovitatavalt järgmise arenduse iteratsiooni jooksul.

Viimaseks kirja pandud funktsionaalseks nõudeks kasutajaloo kujul oli selline soov, kus administraator võiks standardiseeritud aruannete konfiguratsioonide loomisel kasutada valemeid, et rakendus teaks, kuidas standardiseeritud aruannete väärtused hiljem arvutada. Need valemid võiksid meenutada Exceli valemeid. Ülesanne sai täidetud, kuid süntaks on veidi erinev, kuna *SpEL* parseri kuju on raske muuta. Küll aga valemite koostamise keerukus on suhteliselt lihtne ja sellele vastav juhend on olemas: vt lisa 4.

4.2.2 Mittefunktsionaalsed nõuded kasutajalugudena

Kasutajalugudena said kirja ka mõned mittefunktsionaalsed nõuded, millest mõned olid päritud eelmisest rakenduse iteratsioonist. Esiteks sooviti, et autorid dokumenteeriksid enda tööd piisavalt hästi, et tulevikus oleks rakendust hästi ja suhteliselt kergesti edasiarendatav. Sellele on pööratud eraldi hoolsalt tähelepanu, seega peavad autorid selle kasutajaloona kirja pandud nõude tehtuks. Nõude täitmise tulemusena valmis lõpuprojekti jooksul hea tehniline dokumentatsioon, kuhu pöörati palju rõhku.

Järgmiseks sooviti hoida kood võimalikult testitud, mis on hea koodi üks olulisi näitajaid. Sellega jõuti alustada, kuid enamik testimist jäeti perioodile enne kaitsmist. See tähendab, et projekti dokumentatsiooni esitamise seisuga on see nõue osaliselt täidetud ning kaitmise ajal on autoritel ette näidata testitud tagarakenduse kood.

Eelmisest arenduse iteratsioonist päriti kasutajaloona kirja pandud nõue, kus kasutajaliidese funktsionaalsus peaks toimima kõigis uuemates veebilehitsejates. Kasutajaliides on kontrollitud erinevate veebilehitsejatega ning kõik töötab ootuspäraselt. Nõue on täidetud.

Viimaseks nõudeks oli rakendust edasi arendada nii, et kirjutatud kood võimaldaks hiljem selle laiendamist ja edasiarendamist. See nõue oli kirja pandud seetõttu, et suure tõenäosega arendatakse seda rakendust edasi ka tulevikus. Koodi refaktoorimisele on pööratud erilist tähelepanu, et järgida võimalikult hästi puhta koodi ja objektorienteeritud programmeerimise printsiipe. Ülesandega tuldi toime.

4.3 Arhitektuuri analüüs

Esimese iteratsiooni jooksul jaotati arhitektuur kaheks: tekkis tagarakendus ja kasutajaliides. Esialgse autori sõnul tekitatakse niiviisi projektide vahel nõrk seos, mis muudab koodi paremini hallatavaks ning nii on ka mugav projekte testida. [1]. Edaspidi teise arenduse käigus lisandusid uued komponendid nagu Python'i keeles kirjutatud *scraper* ja RabbitMQ sõnumiedastaja, mis aitasid hoida funktsionaalsuse eraldatuse printsiipi. Need komponendid olid eraldi projektides. Tagarakendusele lisati juurde ka kalkulatsioonide funktsionaalsus ning selle eraldi tõstmine ei olnud tollel hetkel otstarbekas selle väikese mahu tõttu. Kuna viimase iteratsiooni käigus ei tehtud finantskordajate kalkulatsioonide osas muudatusi, siis kogu äriloogika osa, mis seda puudutab jäeti puutumata.

Selle iteratsiooni käigus lisandus rohkelt äriloogika funktsionaalsust tagarakendusse ning selle eraldi tõstmist teise projekti ei olnud vajadust teha, kuna äriloogika funktsionaalsus võiks ideaalise stsenaariumi järgi jääda tagarakenduse osaks. Klasside vaheliste sõltuvuste minimeerimiseks kasutati juba eelnevates lõputöödes mainitud *dependency injection* tehnikat selleks, et minimiseerida klassidevahelisi sõltuvusi. Paigutati kõik sarnast funktsionaalsust teostavad klassid eraldi kaustadesse. Kasutati üleval mainitud *OOP* ja puhta koodi soovitusi ning selleks võeti kasutusse Rovert C. Martini *Clean Code – A HandBook of Agile Software Craftsmanship* raamat, kuid sellega ei mindud äärmustesse [30]. Jälgiti üldiseid printsiipe, kus selles vajadust nähti.

4.4 Disaini analüüs

Tagarakenduse API disaini küsimustes jätkati REST arhitektuuri stiili kasutamist. Seda arhitektuuri on võrreldes konkureerivate lahendustega lihtsam rakendada. See kasutab vähem ressursse ja on märgatavalt kiirem, kui varem olemasolnud alternatiivid [2].

Kolmanda iteratsiooni arenduse käigus lisandus tagarakendusele peamiselt äriloogikat ning neid toetavad klassid. Juurde tulid üheksa uut kontrollerit, mille peamised päringu liigid on GET.

„Päringute hulga suurenemisel ja keerukuse kasvamisel peaks järgnevatel arendusetappidel mõtlema kuidas neid paremini grupeerida ja milline oleks optimaalseim URI tee.“, on väitnud Käppa [2]. Päringute arv tõesti suurenes ning tagarakenduses paigutati kontrollerid otstarbe järgi grupeeritult.

Lõpuprojekti autorid andsid endast parima, et anda URI’dele korrektsed nimed. Algselt üritati anda neile sellised nimed, mis oleksid arusaadavad tavalisele interneti kasutajale, hiljem aga mindi korrektsema vastupanu teed ning kasutati juba soovitatud URI nimetusi tagarakenduse refaktoorimise järgselt.

Nagu ka varasemas Katre-Helene töös, said ka meie *frontend* disaini komponendid tükeldatud ajapikku väiksemateks juppideks. Lisaks tehti komponendid taaskasutatavateks ja koodi struktureeriti [2]. Näiteks väljaarendatud taaskasutatava andmetabeli kirjete komponente *CellContentFieldComponent* ja *FormComponent* on omakorda võimalik taaskasutada klientrakenduse muudes komponentides.

4.5 Projekti teostamise põhjendus

Projekti tegemisel võeti aluseks mitmed aruannete standardid ning raamatute õpetused. Raamatud hangiti O’Reilly andmebaasist ja aruannete standardid leiti internetist. Raamatute kasutamise eesmärk oli saada ülevaate, kas valitud tehnoloogiad on antud rakenduse jaoks sobilikud.

Koodi kirjutamisel lähtuti Robert C. Martini raamatust “Clean Code”. Alustuseks võeti koodi kirjutamise reegliks poiss-skaut’i reegeli ehk “Jäta laagriplats puhtamaks, kui sa selle leidsid”. Selle tõttu toimus koodi pidev refaktoorimine. Järgiti nimetamise reegleid. Kasutati asjalikke nimetusi: klassid on nimisõnad ning meetodid tegusõnad. Kolmandaks tehti nii, et võimalusel iga funktsioon ja meetod teeks ühte asja ning et kood ei korduks. Tegime erandi kontrolleritele, sest seal tuleb teha palju funktsionaalsust, kus kasutatakse rohkesti meetodeid. Viimaseks kasutati veateadete jaoks nende kuvamist [17].

Agiilse arenduse metoodika Scrum raamistik valiti tudengite poolt, kuna varasemates autorite arendustöödes on see metoodika tõstnud arendajate produktiivsust.

Tööprotsess algas aktiivse suhtlusena juhendajaga, kus analüüsiti ning pandi kirja funktsionaalsed ja mittefunktsionaalsed nõuded *user-story'*dena [31]. Arutati läbi projekti eesmärk, töökäik ning esimesed võimalikud *issue*'d, mille järel planeeriti edasine tegevus. Hiljem tehti esimese koosoleku, kus tudengid, tarkvaraarendajate rollis, lõid esimesed *issue*'d ning asusid kohe uute programmeerimise keeltega lähemalt tutvuma. Peale tutvumist asuti esimeste tarkvaraarenduse ülesannete juurde.

Kasutajalood muutusid ajapikku ning nende kasutuselevõtt oli tingitud sellest, et lõputöö puhul kasutati agiilse arenduse võtteid. Seega said traditsioonilised funktsionaalsed- ja mittefunktsionaalsed nõuded kirja kasutajaloodena.

Kasutajalugu on tarkvaraarenduses mingisuguse funktsionaalsuse ja kasutaja vahelise kokkupuute sõnaline kirjeldus. See lähtub tavaliselt süsteemi lõppkasutaja vaatenurgast. [32]. Kasutajaloo peamiseks ülesandeks meie kontekstis on suurendada mõistmist tarkvaraarendaja ja investori vahel. See aitab panna funktsionaalsusi reaalsesse konteksti, mille tulena muutub tarkvara loomine lihtsamaks ning kõikide osapoolte soovid saavad paremini täidetud. Lõppkasutajaks loetakse selle arenduse käigus finantsteadmistega administraatorit.

Võrreldes varasema lõputööga, tehti seda lõpuprojekti varem mainitud agiilseid tarkvaraarenduse võtteid kasutades. Iga nädal kujutas endast ette *sprinti*, mille jooksul tuli ära teha kokkulepitud ülesanded. Nädala lõpus tehti meeskonnaliikmetele sobival ajal *sprindi* kokkuvõte. Autorid näitasid üksteisele ette, mis oli nädala jooksul tehtud. Räägiti, mis takistused üksteisel ees on ning mida plaanitakse teha järgmisel nädalal.

Suhtlus arendajate ja juhendajate vahel käis läbi Microsoft Teamsi ja kiirsõnumite rakenduste. Meeskonnaliikmed tegid tööd distantsilt ning tööd teostati võimalikult individuaalselt. Töö oli jaotatud kahte põhilisse ossa: Henri Hummal tegeles põhiliselt kasutajaliidesega ja tagarakenduse funktsionaalsus, loogika ning arhitektuur oli Marko Jõgi vastutusala. Selline tööjaotus aitas meid võõraste tööriistadega kiiremini meistriks saada ning spetsialiseerimine aitas retrospektiivselt hinnates kaasa produktiivsusele.

Rakendus, mis lõpuprojekti jooksul valmis saadi, võimaldab finantsteadmistega administraatoril lisada süsteemi reaalseid andmeid, mis seni puudusid rakendusest ning olid siia kaua oodatud. Lisaks saab luua standardiseeritud aruannete grupid, tänu millele saab tulevastes arendustes teha investoritele kättesaadavaks standardiseeritud andmed. Sel moel saab ettevõtteid omavahel ning firma siseselt tulemusi võrrelda läbi erinevate perioodide. Rakendust arendanud meeskond omandas vajalikud teadmised, kuidas lugeda andmeid csv failidest ning luua dünaamiliselt valemeid, tänu millele luuakse standardiseeritud aruanded. Samuti omandati teadmisi investeerimise vallast, mis aruandeid on vaja tervikliku ettevõtte finantsilise pildi saamiseks ning mis finantssuhtarve selleks kasutatakse.

Kokkuvõttes loodi lisandväärtusena tagarakenduse poolt järgmine funktsionaalsus, mida võib lugeda selle iteratsiooni lõplikuks tulemiks:

- Administraator saab lisada reaalsed ja standardiseerimata ajaloolised andmed ettevõtte kohta, kasutades selleks csv faile, mida administraator on ettevõtetelt hankinud.
    - Selleks kasutatakse upload vaadet kasutajaliideses.
- Administraator saab luua erinevate perioodide ja aruannete jaoks konfiguratsiooni objekte, kus kaardistatakse ära, mis moel kavatsetakse tulevikus genereerida standardiseeritud aruande gruppe.
    - Tuleb kasutada Postmani päringuid. Tulevikus on soovitatav luua selle jaoks vaated kasutajaliidesesse.
- Administraator saab muuta olemasolevaid konfiguratsioone ning uuenenud konfiguratsioonide alusel uuendada standardiseeritud aruannete väärtuseid.
    - Selleks tuleb kasutada Postmani päringuid.
- Administraator saab luua esialgse standardiseeritud aruannete grupi standardiseerimata aruannete grupist.
    - Selleks tuleb kasutada GET päringut Postmanis või veebilehitsejas.

- Administraator saab uuendada juba varasemalt arvutatud standardiseeritud aruannete gruppide väärtusi, kui neile aruannetele vastavad konfiguratsiooni objektid on muutunud. Näiteks otsustati arvutada standardiseeritud aruande kasumi väli teiste standardiseerimata aruande väljade põhjal.
    - Selleks tuleb kasutada GET päringut Postmanis või veebilehitsejas.

Kõik Postmani teel tehtav funktsionaalsus peab tulevaste iteratsioonide käigus olema tehtav rakenduse kasutajaliideses. See asi jäi töö autoritel pooleli.

4.6 Hinnang projekti teostamise protsessi kohta

4.6.1 Projekti juhtimine ning teostamise protsess

Lõpuprojekti arendustöö kestis 12 nädala jooksul. Projektiga alustati 29. veebruaril 2021 ning lõpetati 18. mail 2021. Projektiga tegeleti algselt kaks-kolm päeva nädalas ning graafik oli vaba. Projekti viimasel kuul läks arenduse intensiivsus suuremaks ning projekti panustati enam kui täistööajaga. Kõik tööpäevad kestsid vähemalt 4 tund. Iga nädal panustati projekti iga liikme poolt ca 21 tundi tööd. Kokku saadi 12 nädalaga projekti panustatud üle 500 tunni tööaega.

Projekti juhendajad ülikooli poolt olid Tõnn Talpsepp, kes andis nõuandeid majandust puudutavatel teemadel ning koodi kohta esitati küsimused kaasjuhendajale, kelleks oli Viljam Puusep.

4.6.2 Hinnang projektile

Projekti arenduse käigus esines palju takistusi, mis tulenesid autoritele võõraste tehnoloogiate kasutuselevõtust, kuid need saadi ajapikku ise lahendatud. Selle tõttu aga venis arendusaeg ning ei suudetud kõik planeeritud kasutajalood realiseerida, eriti puudutab see kasutajaliidese vaateid. Suhtlus juhendajatega oli hea, kiire ning küsimustele leiti kiiresti vastused.

Kokkuvõtvalt võib öelda, et protsess sujus edukalt, kuna lõpuprojekti tulemuseks sai algselt planeeritud funktsionaalsus teostatud.

4.6.3 Hinnang üldisele protsessile

Töökorraldus oli paindlik ja algusest peale arusaadav. Kohe alguses saadi kokkulepitud töökorraldus autorite vahel ning igaüks teadis, mida teha.

Tiimiliikmete vaheline läbisaamine oli positiivne. Ajapikku õpiti üksteist paremini tundma, tänu millele saadi ülevaate üksteise tugevuste ja nõrkuste üle. See aitas kohe alguses jaotada õigesti vastutusalad parima lõpptulemuse saavutamiseks. Autorid panustasid projekti pluss-miinus võrdselt.

4.7 Töö edasiarenduse võimalused

Selle töö skoopi ei olnud planeeritud finantskordajate arvutamist standardiseeritud aruannetest, kuid need tuleks igal juhul tulevikus valmis teha. Selle abil saaksid investorid parema pildi ettevõtte finantsseisundist. Hetkel näidatakse neis vaid *mock* andmeid. See võimekus on soovitatav realiseerida järgmise arenduse iteratsiooni jooksul, kui selline võimalus peaks tulema.

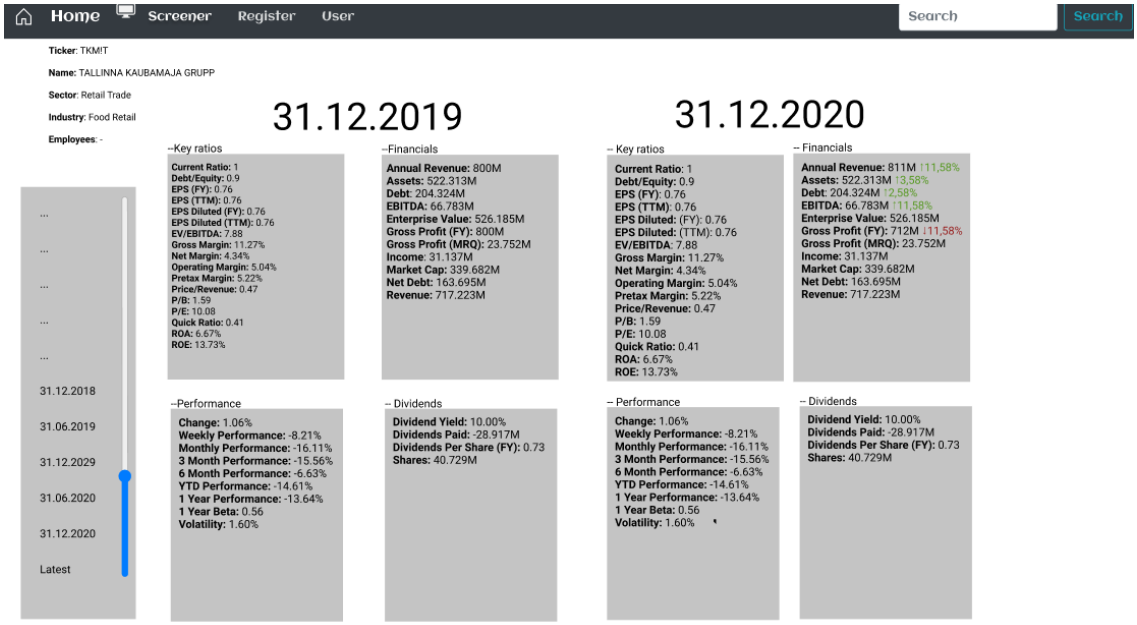
Töö alguses tekkisid autoritel lisaks minimaalsele nõutud funktsionaalsusele ka oma mõtted, kuidas võiksid investorite jaoks kasutajaliidese vaated välja näha ning mis seal saaks teha. Ühel hetkel mõistsid aga autorid, et ei jõua seda ekstra funktsionaalsust rakendada ning need kasutajalood jäeti kõrvale. Küll aga tehtud mõttetöö ei tuleks raisku lasta ning tänu sellele said valmis ka investori kasutajalood, mis võiksid olla realiseeritud tulevastes rakenduse iteratsioonides.

Investori kasutajalood:

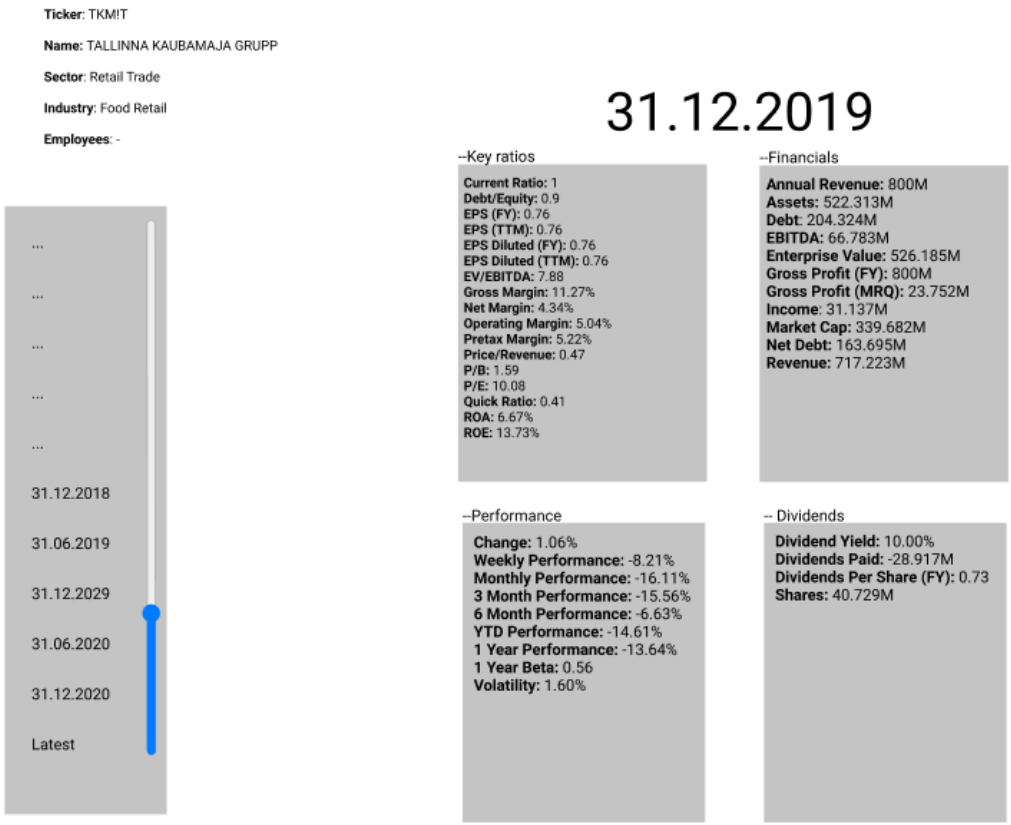
- Investorina ma saan näha ajaloolisi ettevõte finantsaruandeid kronoloogilises järjestuses, sest nii ma saan valida mind huvitava perioodi ning sellega lähemalt tutvuda investori seisukohast.
- Investorina ma saan näha konkreetse perioodi finantsaruandeid, nii, et ma ei peaks neid eraldi otsima internetist või firma poolt esitatud csv failidest.
- Investorina ma saan näha ettevõtte finantsilisi suhtarve, mis on välja arvutatud kindla perioodi standardiseeritud finantsaruannete baasil
    - Rahandussuhtarvud
    - Likviidsus ja maksevõime suhtarvud
    - Väärtussuhtarvud

- Investorina soovin ma näha kahte kõrval olevat perioodi, et näha firma arengu tendentse.
- Investorina soovin ma näha väärtuskordajaid ja suhtarve konkreetsel perioodil
- Investorina tahan ma võrrelda arvutatud finantssuhtarve erinevate perioodide kohta.

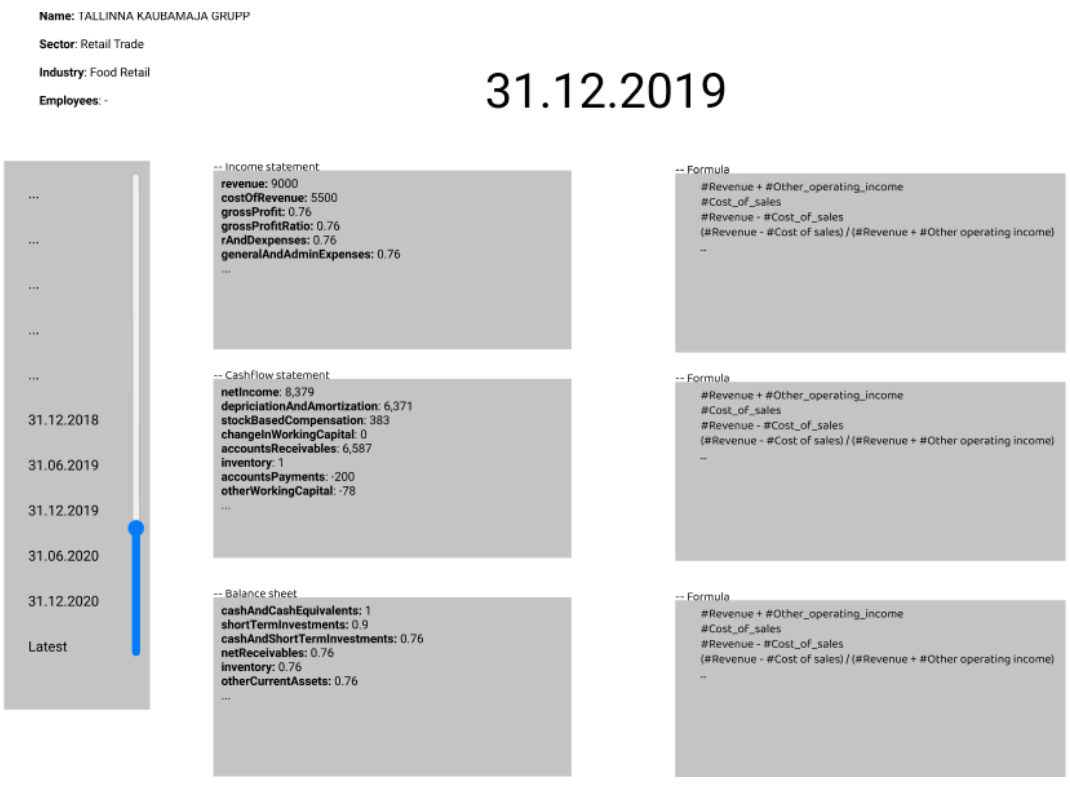
Lõime ka esialgsed kasutajaliidese kiired *mock*-vaated, mille põhjal võiksid valmida investoritele mõeldud vaated.



Joonis 22. Firma finantskordajate ülevaade (võrdlus)



Joonis 23. Firma finantskordajate ülevaade



Joonis 24. Firma perioodi standardiseeritud aruannete ülevaade

4.7.1 Klientrakendus

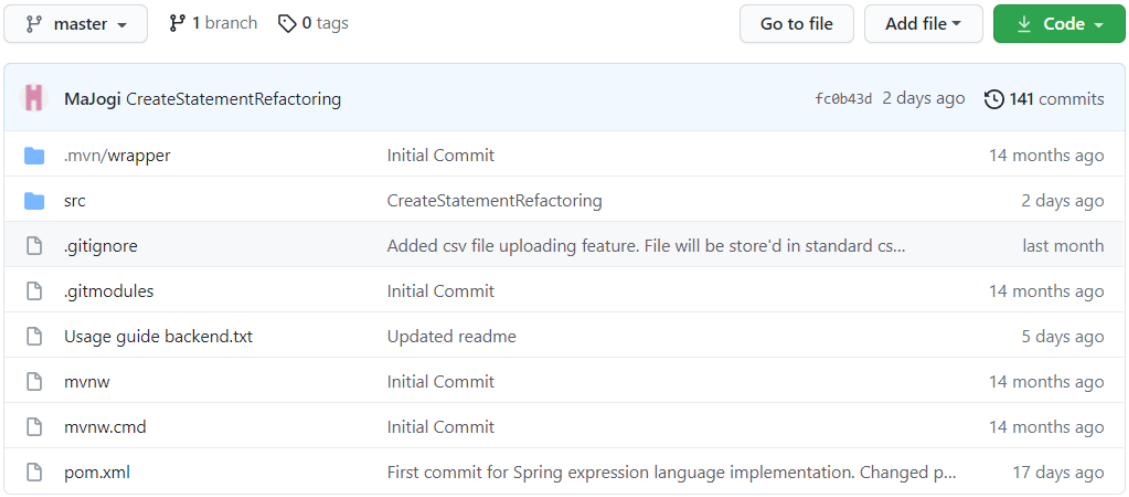
Klientrakenduse poole pealt peaks arendajate esimeseks ülesandeks saama andmete konfigureerimise vaate lõpetamine. Seejärel tuleks luua administraatorkasutajatele spetsiaalne „*dashboard*“, mis seoks failide üleslaadimise ja konfigureerimise üheks loogiliseks tervikuks ning kust oleks võimalik ettevõtte teisigi parameetreid muuta.

Andmete konfiguratsioonikomponendis saab majandusaruannete standardiseerimisprotsessi luua sujuvamaks, luues näiteks võimaluse kasutada ühe ajaperioodi valemeid ka teiste ajaperioodide puhul.

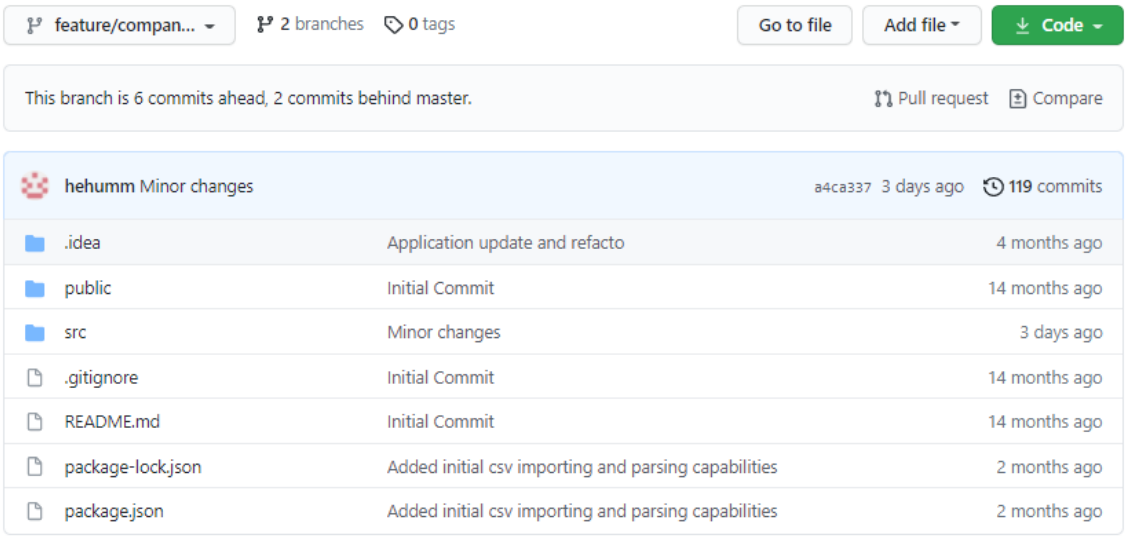
Klientrakenduse varasemates iteratsioonides arendatud ettevõttete profiilide vaadet tuleks uuendada, et *mock*-andmete asemel kuvataks seal reaalseid selle ettevõttega seonduvaid andmeid.

4.8 Teostatud tööde logi

Peatükk sisaldab Toggle tarkvara logisid ja lühikest informatsiooni teostatud tööde kohta. Toggle tarkvara võimaldab hästi analüüsida kuupäevade ja ülesannete täpsusega, palju aega kulus tarkvaraarendusele.

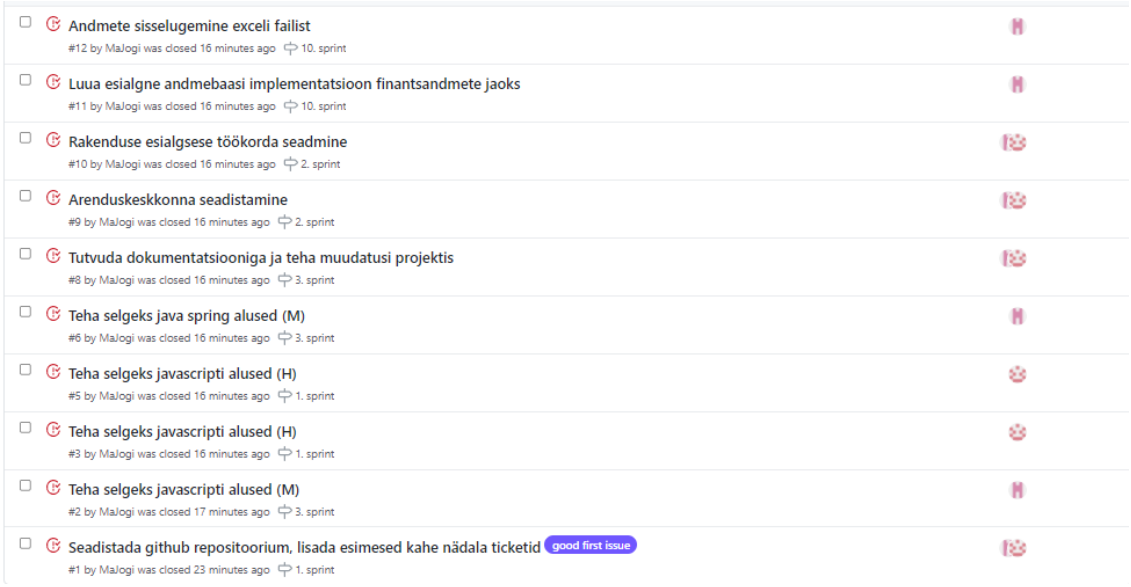


Joonis 25. Back-end commitide arv ja projekti struktuur

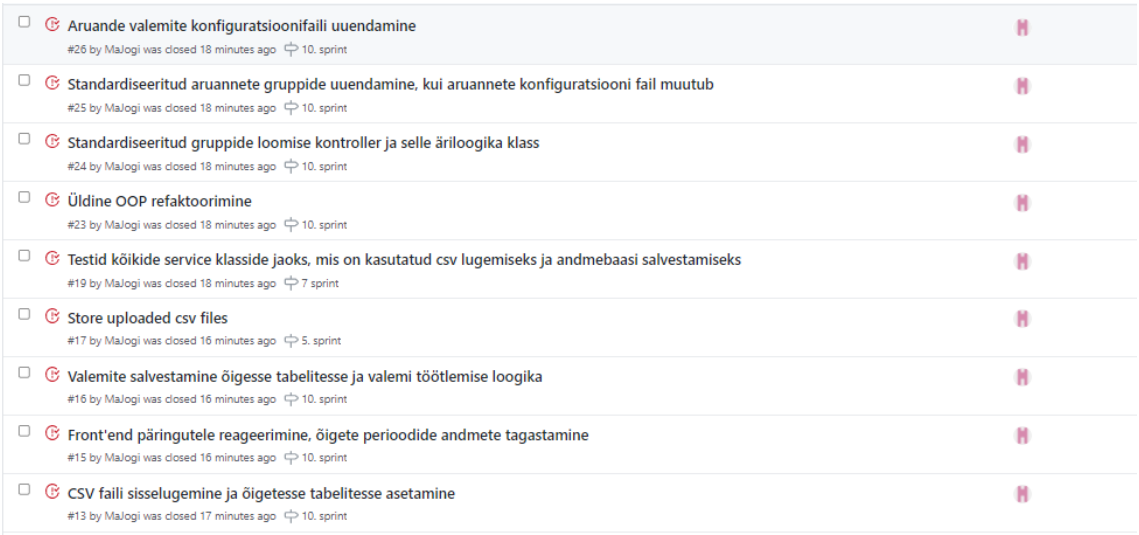


Joonis 26. Kasutajaliidese commitide arv ja projekti struktuur

(a)



(b)

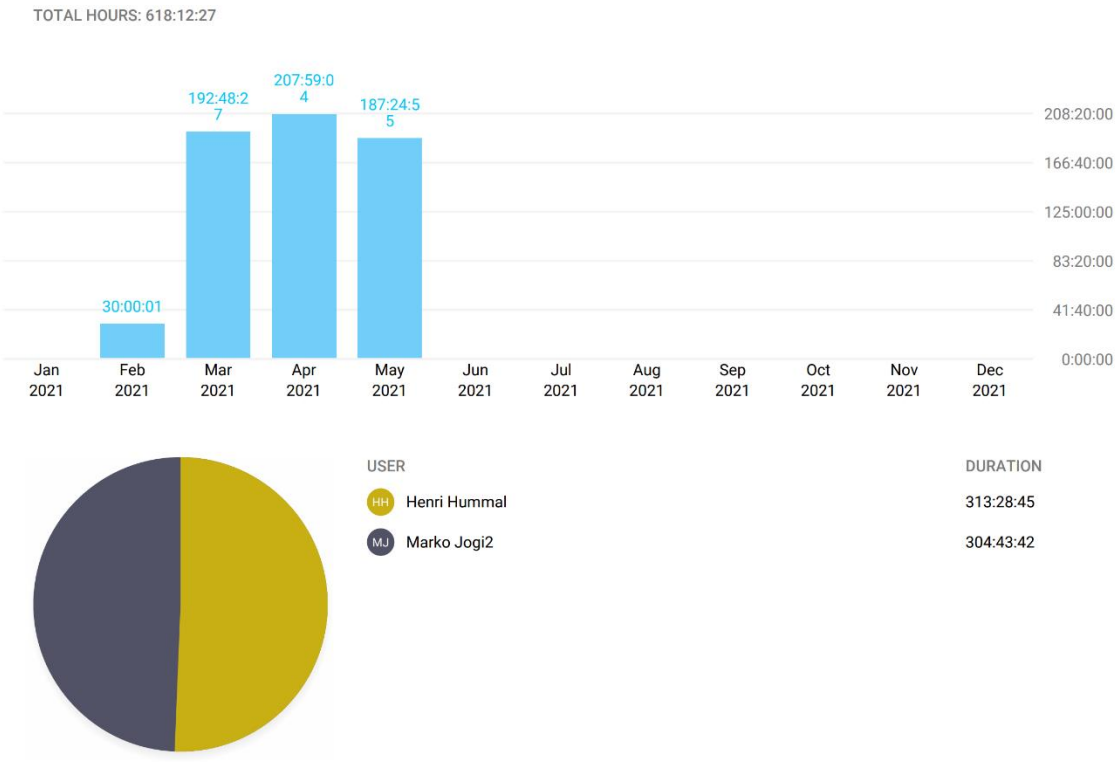


Joonis 27. Ülevaade projekti tagarakenduse tehtud töödest (a) ja (b)

4.8.1 Autorite logid

Table 1. Meeskonnaliikmete logid

Kasutaja	Tegevus	Kulunud aeg
Henri Hummal	Dokumentatsiooni kirjutamine	46:32:25
Henri Hummal	Ideede formuleerimine	03:31:34
Henri Hummal	Kasutajaliides: failide üleslaadimine	23:07:50
Henri Hummal	Kasutajaliides: finantsinfo konfiguratsioon	34:29:14
Henri Hummal	Kasutajaliides: kirjete muutmine	39:37:58
Henri Hummal	Kasutajaliides: komponentide testimine	01:40:47
Henri Hummal	Kasutajaliides: standardsete tabelite implementeerimine	01:02:52
Henri Hummal	Kasutajaliides: standardvaate implementeerimine	00:39:04
Henri Hummal	Kasutajaliidese arendamine	76:26:39
Henri Hummal	Kohtumine	10:02:57
Henri Hummal	Koosolek	03:02:15
Henri Hummal	Projekti eelne planeerimine	15:00:00
Henri Hummal	UI disainimine	02:33:31
Henri Hummal	Õppimine	50:37:42
Henri Hummal	Õppimine, lõputöö kirjutamine	05:03:57
Marko Jogi	Dokumentatsiooni kirjutamine	62:23:39
Marko Jogi	Kohtumine	04:46:16
Marko Jogi	Koosolek	11:27:25
Marko Jogi	Projekti eelne planeerimine	15:00:01
Marko Jogi	Tagarakendus: andmebaasi arhitektuuri disainimine	32:44:38
Marko Jogi	Tagarakendus: andmebaasi mudelite loomine	04:33:22
Marko Jogi	Tagarakendus: andmete sisselugemine ja andmetöötlus	16:35:50
Marko Jogi	Tagarakendus: csv andmete andmebaasi salvestamine	34:51:15
Marko Jogi	Tagarakendus: failide üleslaadimine	12:17:39
Marko Jogi	Tagarakendus: finantsinfo konfiguratsioon	12:51:37
Marko Jogi	Tagarakendus: koodi esialgne tööle saamine	07:13:12
Marko Jogi	Tagarakendus: standardiseeritud aruannete grupi loomine	29:57:43
Marko Jogi	Tagarakendus: testimine	08:16:25
Marko Jogi	Tagarakendus: üldine refaktoorimine	16:38:01
Marko Jogi	Õppimine	35:06:39
	Kokku	**618:12:27**



Joonis 28. Projektile kulunud aeg

4.9 Meeskondlik hinnang tiimiliikmete panuste kohta

Meeskonnaliikmed on panustanud lõpuprojekti võrdselt ja olnud alati üksteisele kättesaadavad. Tiimiliikme poole pöördunud küsimusega ei pidanud kaua vastust ootama. Kaheliikmeline meeskond innustas üksteist tööle ja keerulisemad probleemid arutati koos läbi. Mõlemad meeskonnaliikmed said lõppkokkuvõttes hinnanguks 0 punkti meile antud skaalal, mis tähendab, et iga meeskonnaliige panustas lõpuprojekti võrdselt.

5. Kokkuvõte

Lõpuprojekti jooksul arendati edasi Tallinna börsi aktsiate finantsandmete veebirakendust. Bakalaureusetöö põhieesmärgiks seati rakendusele uue funktsionaalsuse lisamine. See oli seotud reaalsete ajalooliste andmete sissetoomisega projekti.

Projekti arenduse kolmas iteratsioon viib projekti tervikuna lähemale eesmärgile luua Tallinna börsile platvorm, kus investor leiab investeerimisotsuste langetamise jaoks mugavalt informatsiooni paljude oluliste aspektide kohta:

- Ettevõtte finantsiline seisund
- Väärtpaberi ajakohane ja ajalooline hind
- Ettevõtte ajaloolised ja hiljutised majandusaruanded standardiseerimata ja standardiseeritud kujul
- Ettevõtte ajaloolised ja hiljutised finantsindikaatorid

Suurtematel turgudel tegutsevate firmade ajaloolisi andmeid leidub paljudel platvormidel, kuid Tallinna turu väiksusest tingituna pole siinse turu kohta pakutavate andmete kvaliteet eelkõige USA turule mõeldud tööriistadest piisav ning taolise informatsiooni hankimine iseseisvalt nõuaks kasutajalt spetsiaalseid finantsteadmisi.

Arenduse kolmanda iteratsiooni tulemusena valmis uuenenud projekti versioon, kus kasutatakse juba reaalseid ajaloolisi andmeid. Projekti tootmiskeskkonda suunamiseks on veel vara, sest selle iteratsioonis loodud ajalooliste andmete lisamise ja hoiustamise juurde on vajalik juurde luua vaated investoritele. See aga tähendab, et töö järgnevates etappides tuleb kindlasti täiendada ka finantskordajate arvutamise süsteemi, mis on lisandunud teise arenduse iteratsiooni käigus, et need arvutataks automaatselt, lähtudes ettevõtte standardiseeritud andmetest.

Kolmas iteratsioon jättis vahele eelmise töö soovitust tegeleda turvalisusega. Sellega on aga võimalik tegeleda tulevastes iteratsioonides peale vajaliku domeeniloogika ja -funktsionaalsuse valmimist.

Kasutatud kirjandus

[1] M. Jagor, „Veebirakendus Tallinna börsiettevõtete finantsandmete kajastamiseks“, [Bakalaureusetöö], Infotehnoloogia teaduskond, TalTech, Tallinn, Eesti, 2020. [Online]. Loetud aadressil: https://digikogu.taltech.ee/et/Item/62b79499-c091-48b6-898e-799b442c864f.

[2] K.-H. Käppa, „Veebirakendus Tallinna börsiettevõtete finantsandmete kajastamiseks“, [Bakalaureusetöö], Infotehnoloogia teaduskond, TalTech, Tallinn, Eesti, 2021, unpublished.

[3] *Components and Props*, *version 17.0.2*, React Official Documentation [Online]. Loetud aadressil: https://reactjs.org/docs/components-and-props.html Kasutatud 01.05.2021

[4] Nasdaq Balti börsid, *Meist | Nasdaq Balti börsid*, 2021, [Online]. Loetud aadressil: https://nasdaqbaltic.com/et/meist/nasdaq-balti-borsid/ Kasutatud: 3.04.21.

[5] Nasdaq Balti börsid, *Tallinna Kaubamaja Grupp – Aruanded*, 2021, [Online]. Loetud aadressil: https://nasdaqbaltic.com/statistics/et/instrument/EE0000001105/reports Kasutatud: 3.04.2021

[6] Nasdaq Balti börsid, *Tallinna Kaubamaja Grupp - Infoleht*, 2021, [Online]. Loetud aadressil: https://nasdaqbaltic.com/statistics/et/instrument/EE0000001105/fact_sheet Kasutatud: 5.04.2021

[7] TradingView, *TradingView Features*, 2021, [Online]. Loetud aadressil: https://www.tradingview.com/features/ Kasutatud 5.04.2021.

[8] TradingView, *TALLINNA KAUBAMAJA GRUPP - Financial statements*, [Online]. Loetud aadressil: https://www.tradingview.com/symbols/OMXTSE-TKM1T/financials-income-statement/ Kasutatud 6.04.2021.

[9] Sparx Systems Pty Ltd., *Getting to know Documentation*, [Online]. Loetud aadressil: https://www.sparxsystems.com/enterprise_architect_user_guide/14.0/guidebooks/tools_ba_documentation.html Kasutatud 23.04.2021.

[10] P. W. Szabo, „The Three Rs or the Connextra format“ in *User Experience Mapping*, Packt Publishing, 2017. [Online]. Loetud aadressil:

https://learning.oreilly.com/library/view/user-experience-mapping/9781787123502/92d21fe3-a741-49ff-8200-25abf18c98d0.xhtml Kasutatud: 23.05.2021.

[11] J. Rabelo, „Foreign Key“, Techopedia, August 2020. [Online]. Loetud aadressil: https://www.techopedia.com/definition/7272/foreign-key#:~:text=A%20foreign%20key%20is%20a,establishing%20a%20link%20between%20them.. Kasutatud 25.04.2021.

[12] FmpCloud Inc, *Financials Statements as Reported On The U.S. Securities and Exchange Commission (SEC),* 2021, [Online]. Loetud aadressil: https://fmpcloud.io/documentation#financialStatementsAsReportedOnSEC Kasutatud: 5.04.2021.

[13] M. Masse, *REST API Design Rulebook,* USA: O'Reilly Media, Inc., 2011. [E-book]. Loetud aadressil: https://learning.oreilly.com/library/view/rest-api-design/9781449317904/ Kasutatud: 15.04.2021.

[14] A. L. Davis, *Spring Quick Reference Guide: A Pocket Handbook for Spring Framework, Spring Boot, and More,* USA: Apress, 2020. [E-book]. Loetud aadressil: https://learning.oreilly.com/library/view/spring-quick-reference/9781484261446/ Kasutatud: 10.04.2021.

[15] *Introducing Hooks, version 17.0.2,* React Official Documentation [Online]. Loetud aadressil: https://reactjs.org/docs/hooks-intro.html Kasutatud: 01.05.2021

[16] R. C. Martin, *Agile Principles, Patterns, and Practices in C#,* Lebanon, Indiana, USA.: Prentice Hall, 2006. [E-book]. Loetud aadressil: https://learning.oreilly.com/library/view/agile-principles-patterns/0131857258/cover.xhtml Kasutatud: 16.04.2021.

[17] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship,* USA.: Pearson, 2008. [E-book]. Loetud aadressil: https://learning.oreilly.com/library/view/clean-code-a/9780136083238/chapter01.html Kasutatud: 16.04.2021.

[18] R. Raszczynski, „Understanding the EAV data model and when to use it“, Invica, October 2010. [Online]. Loetud aadressil: https://inviqa.com/blog/understanding-eav-data-model-and-when-use-

it#:~:text=Entity%2DAttribute%2DValue%20design%20is,the%20facts%20that%20it%20describes Kasutatud: 20.04.2021

[19] Baeldung, *Spring Expression Language Guide*, 2021, [Online]. Loetud aadressil: https://www.baeldung.com/spring-expression-language Kasutatud: 12.05.2021.

[20] G. Tomassetti, „A Guide to Parsing: Algorithms and Terminology“, Strumenta SRL, October 2017. [Online]. Loetud aadressil: https://tomassetti.me/guide-parsing-algorithms-terminology Kasutatud: 12.05.2021

[21] *Spring Expression Language (SpEL), version 3.0.x*, Spring Framework Official Documentation [Online]. Loetud aadressil: https://docs.spring.io/spring-framework/docs/3.0.x/reference/expressions.html Kasutatud: 12.05.2021

[22] *6.3.1 The EvaluationContext interface, version 3.0.0*, Spring Framework Official Documentation [Online]. Loetud aadressil: https://docs.spring.io/spring-framework/docs/3.0.0.M4/reference/html/ch06s03.html Kasutatud: 13.05.2021.

[23] N. Borenstein ja N. Freed, „Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies“, 1996, [Online]. Loetud aadressil: https://datatracker.ietf.org/doc/html/rfc2045 Kasutatud: 12.05.2021.

[24] *Conditional Rendering*, React Official Documentation [Online]. Loetud aadressil: https://reactjs.org/docs/conditional-rendering.html Kasutatud: 03.05.2021

[25] *Lists and Keys*, React Official Documentation [Online]. Loetud aadressil: https://reactjs.org/docs/lists-and-keys.html Kasutatud: 03.05.2021

[26] *Reconciliation*, React Official Documentation [Online]. Loetud aadressil: https://reactjs.org/docs/reconciliation.html#recursing-on-children Kasutatud: 03.05.2021

[27] *Opencsv Users Guide*, OpenCsv Official Documentation [Online]. Loetud aadressil: http://opencsv.sourceforge.net/ Kasutatud: 10.04.2021

[28] *Opencsv API*, OpenCsv Official Documentation [Online]. Loetud aadressil: http://opencsv.sourceforge.net/apidocs/index.html Kasutatud: 10.04.2021.

[29] Stack Abuse, *Libraries for Reading and Writing CSVs in Java*, 2021, [Online]. Loetud aadressil: https://stackabuse.com/libraries-for-reading-and-writing-csvs-in-java/ Kasutatud: 10.04.2021.

[30] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, USA.: Pearson, 2008. [E-book]. Loetud aadressil: https://learning.oreilly.com/library/view/clean-code-a/9780136083238/chapter01.html Kasutatud: 16.04.2021.

[31] V. Devedzic ja J. Jovanovic, *A comparative study of software tools for user story management*, Belgrad: Mihailo Pupin Institute, 2015. Loetud aadressil: https://www.sciencedirect.com/science/article/abs/pii/S0950584914001293?via%3Dihub

[32] M. Rehkopf, „User Stories with Examples and Template“, Atlassian, [Online]. Loetud aadressil: https://www.atlassian.com/agile/project-management/user-stories. Kasutatud 15.05.2021.

[33] L. S. Sterling, *The Art of Agent-Oriented Modeling*, London: The MIT Press, 2009.

[34] D. Singh, „Uploading Files with Spring Boot“, Stackabuse, 2019. [Online]. Loetud aadressil: https://stackabuse.com/uploading-files-with-spring-boot Kasutatud: 10.04.2021.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Meie, Marko Jõgi ja Henri Hummal

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Veebirakendus Tallinna börsiettevõtete finantsandmete kajastamiseks: 3. iteratsioon“, mille juhendaja on Tõnn Talpsepp.
   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. Oleme teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

18.05.2021

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 - Kirjalik ülevaade ning kirjeldus meeskonnaliikmete panusest ning tegevustest projektis

Peatükk sisaldab endas kokkuvõtliku ülevaadet meeskonnaliikmete panusest ja tegevustest projektis.

1.1 Eneseanalüüs:

- Marko

Enne projektiga alustamist ei olnud mul Java programmeerimise keelega olnud arvestatavat kogemust, küll aga sain kergete pingutustega programmeerimise keele alused selgeks, kuna ülikoolis kolmandat aastat õpitud programmeerimise keel C# on üles ehitatud Java keele baasil. Tänu sellele oli 75 protsenti oskustest ülekantavad uute keskkonda. Selle projekti teeb minu jaoks eriliseks see, et see oli esimene kord, kus ma võtsin midagi ise projekteerida väikse tiimi ning ilma kõrvalise ettevõtte toeta. Mulle tundub, et kõige suurema arengu, mis puudutab minu tarkvaraarenduse oskust, saavutasin ma just siin. Seda seetõttu, kuna kõigile probleemidele lahenduse pidi leidma ise, mis ei olnud vahetevahel sugugi kerge. Kokkuvõttes olen ma väga rahul enda panusest projekti ning mis sellest on välja tulnud. Usun, et lõpuprojekt oli kõige kasulikum ja huvitavam asi, mida ma pidin ülikooli jooksul olles tegema. Lisaks on see hea alternatiiv neile, kes ei soovi teha liialt analüütilist klassikalist lõputööd. Mina isiklikult tegelesin peamisest tagarakenduse arendamisega, sest teadsin, et suudan pakkuda selles keskkonnas kõige enam lisaväärtust ning ise areneda.

- Henri

Võiks öelda, et mingil määral tegime lõputööd enesekeskselt: seadsime prioriteediks eelkõige isikliku arengu ja projekti kirjutamine oli pigem uute õpitud teadmiste kohene rakendamine. Usun, et oleksime suutnud projekti üldkokkuvõttes rohkem panustada, kui mina oleksin arendanud tagarakendust ja Marko klientrakendust. Põhjus peitub selles, et mina olin Javaga rohkem kokku puutunud ning lisaks õppisin spetsiaalselt Java veebirakenduste arendamist Märt Kalmo kursusel „Veebirakendused Java baasil“ lõputöö kirjutamisele eelneval semestril. Markol polnud väidetavalt küll kogemusi Javaga ega Reactiga, aga kui tema oleks õppinud *front-end* arendamist, oleksin mina saanud ilma

suurema õppimiseta samaaegselt tagarakendust arendada. Selle tõttu jõudsime investeerimisrakendusse reaalselt vähem panustada, sest meie tööjaotuse juures pidime mõlemad märkimisväärse aja veetma uusi tehnoloogiaid õppides. Siiski tegin ise ettepaneku arendada klientrakenduse poolt ja mõlemale tiimiliikmele sobis kokkulepitud tööjaotus. Tagasivaadates jäin väga rahule, sest kogu ülikoolis veedetud 3 õppeaasta jooksul olin alati tegelenud justnimelt *back-end*-tehnoloogiatega ja kindlasti olid minu *front-end* teadmised kõvasti nõrgemad *back*-end maailmaga võrreldes. Veebirakenduste arendamisest terviklikuma pildi saamiseks kulub praktiline kogemus praeguse aja ühe populaarseima klientrakenduse loomise tehnoloogia, Reactiga kindlasti ära.

Kuna koodi kirjutamise kõrvalt tuli paralleelselt hulganisti uusi asju selgeks õppida ning puudus veel „kõhutunne“, mis olukorras mingit kindlat teadmist vaja läheb, tuli palju eksperimenteerida. Kui mõnele probleemile sai leitud lahendus ja mõni aeg hiljem lahendusele pilk peale visatud, selgus (sest vahepeal olin omandanud uusi teadmisi), et vana lahenduse võiks lühemalt/elegantsemalt ümber kirjutada. Seetõttu jäi lõpuks koodi alles vähem, kui tegelikult seda kirjutasin.

1.2 Kirjalik ülevaade meeskonnaliikmete panusest ja logidest:

- Marko

Kokku kulus mul projekti peale veidi üle 300 tunni. Selle saavutamiseks pidin viimastel nädalatel tegelema muude ainete kõrvalt projektiga rohkem kui täistööajaga.

Enne kõike kulus aega kasutatavate tehnoloogiate õppimiseks, kuna nendega ei olnud autoritel projekti alguses palju kogemusi. Kokku kulus aega erinevate kursuste vaatamiseks ca 35 tundi tööaega.

Koosolekutele, projekti eelsele planeerimisele ja muudele suhtlust nõudvate asjadega tegelemiseks kulus mul ca 30 tundi projektile kulutatud ajast. See teeb ca 10% töömahust, mis on päris arvestatav ajakulu. Samas tuleb ära märkida, et kommunikatsioon projektis osalejate vahel on agiilse arenduse puhul oluline punkt, seega on see näitaja põhjendatud.

Rakenduse eluea jooksul tekkivate andmete salvestamiseks oli vaja luua hulga andmebaasi tabeleid. Andmebaasi tabelite projekteerimise, arhitektuuri loomisele ning nende tabelite implementeerimisele kulus mul 37 tundi tööaega. Selline tulemus saavutati

pideva refaktoorimise teel. Algselt oli raske välja mõelda, kuidas rakendada *entity-attribute-value* mustrit ning seetõttu algne arhitektuur muutus ajapikku.

Esimene asi, mis ma tegin oli csv faili üleslaadimise ning selle ja mitme faili korraga originaalkujul salvestamine projekti kausta. Selle tegevuse jaoks kulus mul 12 tundi tööd.

Järgmiseks ja üheks tähtsaimatest ülesannetest oli realiseerida korrektne andmete lugemine csv failist, mis arvestaks näidisfailidega, mis meile algselt juhendaja poolt anti. Töö juurde käis ka hilisem refaktoorimine. Selle ülesande käigus tekkis ka juhend, missugune peaks rakendusele antav fail, et selle lugemine oleks võimalik. Ülesandele kulus ca 16h tööd.

Selle järgnev ülesanne on tihedalt seotud eelmisega, kuid ma otsustasin ta lisada eraldi. Nüüd kui on olemas arvuti mälus loetud andmed, tuleb need korrektselt salvestada tabelitesse, et hiljem neist saaks tekitada standardiseeritud aruanded. See oli üks kõige aeganõudvaimaid ülesandeid. Funktsionaalsus sai realiseeritud 35h-ga.

Järgnes konfiguratsioonide loomise ja uuendamise funktsionaalsus. See oli vajalik samm selleks, et hiljem luua standardiseeritud aruannete komplekt. Ülesandele kulus 13h.

Kõige keerulisem osa, mis sai üsna mõistliku ajaga lahendatud on standardiseeritud aruannete grupist standardiseeritud aruannete grupi loomine. Siin kasutati *SpEL* teeki, et teostada dünaamist standardiseeritud aruannete grupi loomist standardiseerimata aruannete grupi ja hetkel kehtivate firma aruannete konfiguratsioonidest alusel. Ülesandele kulus 30 tundi.

Eraldi logide grupis tõin välja üle 16h tööd, mis kulus kogu rakenduse üldise funktsionaalsuse refaktoorimisele. Loeti puhta koodi raamatut ning rakendati soovitusi, mida ei tehtud ülesande tegemise ajal. Siia alla käib näiteks objektorienteeritud programmeerimise soovituste rakendamine. Tänu selle muutub kood loetavamaks, kergemini hallatavamaks ning üldiselt puhtamaks.

Kõige suurem aeg kulus dokumentatsioonile ehk lõpuprojekti kirjutamisele. Sellele kulus üle 60 tunni. Üks põhjus, miks sellele nii palju aega pöörati on see, et algselt sooviti head koodi dokumenteeritust. Selle tõttu tekkis dokumentatsiooni tugev tehniline kirjeldus, mis aitab tulevasi arendajaid rakendust edasi arendada. Autorite arvates tuli hea ja

läbimõeldud bakalaureusetöö ning dokumentatsiooni kirjutamise ajal leiti üles probleemid, mida kohe ei märgatud.

- Henri

Kogu lõpuprojekti peale läks mul üle 300 tunni. Sellest ajast kulus programmeerimisele ligikaudu 180 tundi, lõputöö dokumentatsioonile ligikaudu 50 tundi, puhta õppimisele ligikaudu 50 tundi, erinevatele koosolekutele ligikaudu 10 tundi ja muudele tegevustele ligikaudu 20 tundi. Puhta õppimise alla lugesin Puhta õppimise alla lugesin aja, kui vaatasin näiteks videokursuseid, kuid õppimist tuli ka ette koodi kirjutamise faasis erinevate probleemide lahendamisel.

Kõigepealt tuletasin meelde JavaScripti põhitõed, sest need on Reacti kasutamise eelduseks. Enne lõputööd puutusin JavaScriptiga kokku vaid põgusalt 2 õppeaastat varem ja praktiline töökogemus puudus. Seejärel töötasin läbi Reacti sissejuhatava kursuse, millest saadud teadmisi proovisin koheselt koodis rakendada.

Alustasin tööd failide üleslaadimise lehega. Alguses asusin looma lahendust, kus klientrakendusse imporditakse fail, millest loetakse välja informatsioon, mida saab kasutajaliideses muuta ja hiljem JSON-objektina tagarakendusse saata. Peale nõuete selginemist aga tuli failide importimine realiseerida selliselt, et klientrakendus saadab faili otse tagarakendusse, kus toimub nii faili enda salvestamine kui ka failist loetud info põhjal koostatud objektide salvestamine PostgreSQL andmebaasi.

Paralleelselt failide üleslaadimislehe arendamisega mõtlesime Markoga koos välja kasutajaliidese disaini. Kuna mõlemal puudus eelnev kogemus kasutajaliidese disainimisega, võttis see etapp samuti veidi aega, sest pidime endale ka disainitööriistad selgeks tegema.

Kuna isiklikult näen tulevikus mõistliku funktsionaalsusena imporditud andmete mõne rea muutmist rakenduse sees, asusin looma andmetabelit, mis samaaegselt toetaks kirjete muutmist ja kahedimensioonilisi andmeid. Kahedimensionaalsus tuleneb sellest, et süsteemi üleslaetavates majandusaruannetes on palju erinevaid vaatlusaluseid objekte (kasumiaruandes näiteks käive, kasum, kulum jne) erinevatel ajahetkedel (2019,2020,2021…). Selle tabeli loomine osutus parajaks väljakutseks, sest seal pidi samaaegselt kasutama väga palju erinevaid Reacti ja JavaScripti teadmisi.

Kui andmetabel sai valmis, hakkasin uuesti tegelema üleslaadimise funktsionaalsusega, et viia see kooskõlla eelnevalt mainitud üleslaadimise tööpõhimõttega. Töös väljatoodud põhjustel tuli luua spetsiaalne üleslaadimise komponent. Selle jaoks pidin õppima, kuidas moodustada failist spetsiaalne JavaScripti *File*-tüüpi objekt. Palju aega kulus ka reaalsete päringute loomiseks, sest keeruliseks osutus POST-päringute päiste konfigureerimine.

Kui failide üleslaadimine hakkas tööle, liikusin edasi failide konfigureerimise vaatesse, kus saab luua standardiseerimata aruannete põhjal standardiseeritud aruandeid. See vaade jäi poolikuks ajapuuduse tõttu. Esimese sammuna alustasin komponentide loomisega paigutusele rõhku pööramata. *Dropdown*-menüüde käimasaamiseks tuli kasutada reflektsiooni, millega ma polnud varem kokku puutunud. Lisaks mõistsin andmete konfiguratsiooni lehel, et varasemalt mainitud andmetabel ei värskenda enda olekut, kui tema *props* muutub (üks paljudest Reacti nüanssidest, mille peale ma vähese kogemuse tõttu koheselt ei tulnud). Proovisin erinevaid lahendusi, aga lõpuks pidin vea kõrvaldamiseks terve andmetabeli komponendi teisaldama JavaScripti funktsiooni süntaksiga komponendist klassi süntaksiga komponendiks.

Lõputöö dokumentatsioonile kulus märkimisväärselt palju aega, sest minu ülesandeks oli stiili ühtsustamine. See võib alguses tunduda küllaltki triviaalse ülesandena, kuid lausete ümbersõnastamine nii mahukas dokumendis selliselt, et need säilitaksid oma algse mõtte ja sobiksid kokku ülejäänud tekstiga, nõuab palju mõtlemist.

Lisa 3 – Juhend RabbitMQ kasutamiseks

Link algallikale, kus saab täiendavat infot tarkvara kohta: https://www.rabbitmq.com/download.html

RabbitMq tuleb korrektselt installida. Peab jälgima, et enne RabbitMQ'd peab olema arvutisse installitud Erlang programmeerimise keel. Installitud Erlangi versioon peab olema ühilduv RabbitMQ versiooniga.

Kui peaks juhtuma olukord, kus ei teki Erlangi installeerimisel Windows environment muutujad, siis tuleb need ise tekitada:

1. Tuleb kirjutada otsingusse environment variables ning avada: Edit the system environment variables -> environment variables.
    a. Tuleb lisada mõlemale PATH muutjale juurde „%ERLANG_HOME%\bin" ning luua uus muutuja pannes selle väärtuseks korrektne viide kaustale, kus asub Erlang. See võib olla näiteks: ERLANG_HOME „C:\Program Files\erl-23.3"
2. Tuleb avada web-scraper projekt ning luua pyinstaller'i abil exe fail, kasutades käsku: „pyinstaller – onefile –windowed failinimi.py"
3. Seejärel tuleks käivitada ps1 script powershelli abil, mille tagajärjel luuakse *task schedulerisse* ülesanne, mis käivitab programmi iga tööpäev kindlal kellaajal ning andmeid uuendatakse iga n minuti tagant.
    a. Võib juhtuda, et *task scheduleri* programm ei leia faili ülesse ning selle lahendamiseks on kaks võimalust:
        i. Lisada fail kausta, mida jälgib PATH environment variable
        ii. Muuta asukoht – antakse ette täpne path, kus exe fail asub.

RabbitMq käivitamine:

rabbitmq-server start -detached
rabbitmq-plugins.bat enable
rabbitmq-plugins enable rabbitmq_management

Lisa 4 – Konfiguratsiooni vormil valemite koostamise juhend

Valemites on lubatud kasutada korrutamist, jagamist, liitmist ning lahutamist. Selleks, et kasutada muutujaid valemis tuleb need tuua lahtrisse sisse „#“ algusega. Näiteks #Revenue.

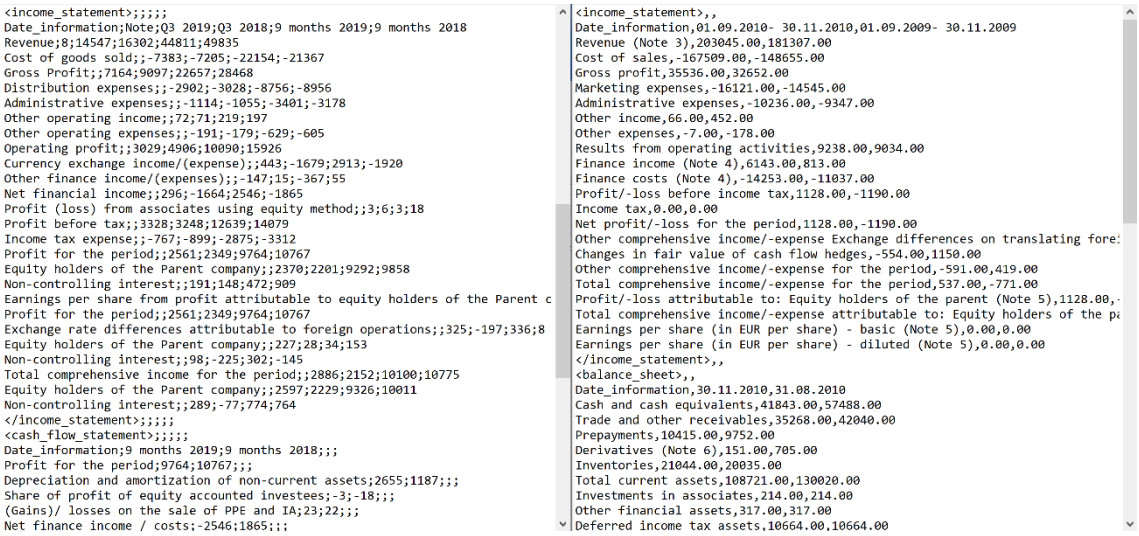
Standardiseeritud aruande rida standardIncomeStat.revenue võidakse saada Tallinna kaubamaja näitel näiteks nii: standardIncomeStat.revenue = „#Revenue + #Other_income“. Tühiku asemel tuleb kasutada alakriipsu, et *SpEL* parser suudaks iterpreteerida korrektselt, mis muutujaid soovitakse kasutada. Kasutatavad on sulud, et tehete järjekorda muuta.

Lisa 5 – Korrektne csv faili formaat

Korrektne csv faili ülesehitus on järgmine:

1) Aruanded algavad ja lõppevad aruande nimega, kusjuures alguses mähitakse aruande nimi <> ja lõpus </> sümbolitega
    - Näide: algab <balance_sheet> ja lõppeb </balance_sheet>
2) Peale aruande algust peab järgmine rida sisaldama esimeses veerus Date_information nimelist rida, millele järgnevad perioodid või kuupäevad. Erandina võib perioodidele eelneda veerg päisega „Note“, millega arvestab tagarakenduse kood.
    - Näide: Date_information, Note, 30.06.2017, 31.12.2016
    - Näide_2: Date_information, 30.06.2017, 31.12.2016
    - Näide_3: Date_information, Q2 2017, Q2 2016, 6 months 2017, 6 months 2016.
3) Võimalikud perioodide ja kuupäevade väärtused, mille seast valida (kui kuupäeva/perioodi struktuur erineb, tuleb see käsitsi muuta):
    - Bilanss
        - Kindla kuupäeva seisuga: dd.mm.yyyy
            - 31.12.2018
            - 30.06.2018
            - 15.05.2018
    - Kasumi ja rahavoogude aruanne
        - Ajaperiood
            - 3 months yyyy, 6 months yyyy, 9 months yyyy, 12 months yyyy
        - Aasta
            - 2017
            - 2018
        - Kvartal
            - Q1 yyyy, Q2 yyyy, Q3 yyyy, Q4 yyyy

Imporditavatele failidele kehtib nõue, et need peavad olema csv-formaadis ja väärtuste eraldajateks peab olema semikoolon (;).

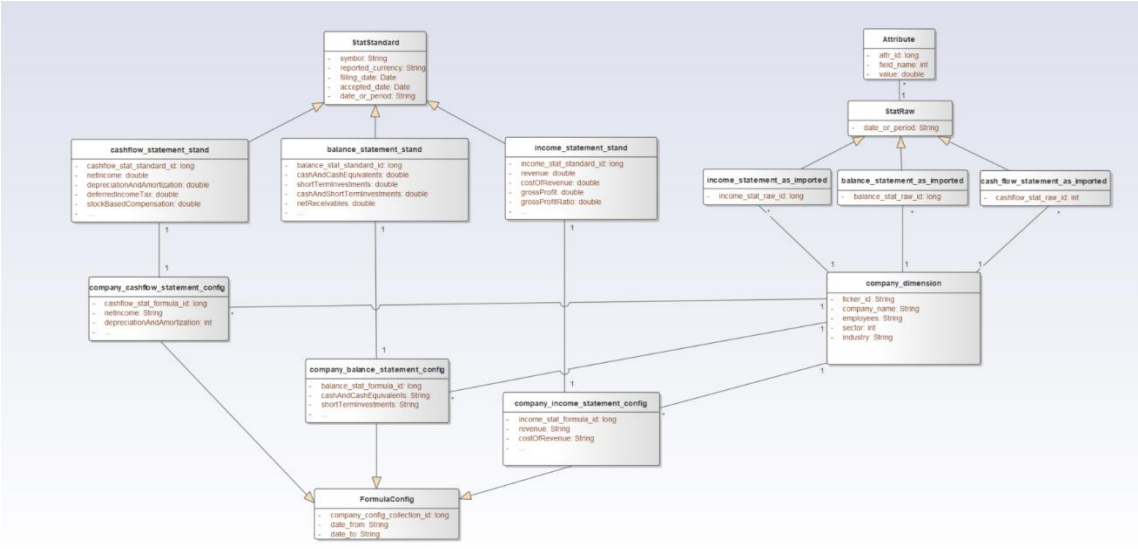


```
<income_statement>;;;;;
Date_information;Note;Q3 2019;Q3 2018;9 months 2019;9 months 2018
Revenue;8;14547;16302;44811;49835
Cost of goods sold;;-7383;-7205;-22154;-21367
Gross Profit;;7164;9097;22657;28468
Distribution expenses;;-2902;-3028;-8756;-8956
Administrative expenses;;-1114;-1055;-3401;-3178
Other operating income;;72;71;219;197
Other operating expenses;;-191;-179;-629;-605
Operating profit;;3029;4906;10090;15926
Currency exchange income/(expense);;443;-1679;2913;-1920
Other finance income/(expenses);;-147;15;-367;55
Net financial income;;296;-1664;2546;-1865
Profit (loss) from associates using equity method;;3;6;3;18
Profit before tax;;3328;3248;12639;14079
Income tax expense;;-767;-899;-2875;-3312
Profit for the period;;2561;2349;9764;10767
Equity holders of the Parent company;;2370;2201;9292;9858
Non-controlling interest;;191;148;472;909
Earnings per share from profit attributable to equity holders of the Parent c
Profit for the period;;2561;2349;9764;10767
Exchange rate differences attributable to foreign operations;;325;-197;336;8
Equity holders of the Parent company;;227;28;34;153
Non-controlling interest;;98;-225;302;-145
Total comprehensive income for the period;;2886;2152;10100;10775
Equity holders of the Parent company;;2597;2229;9326;10011
Non-controlling interest;;289;-77;774;764
</income_statement>;;;;;
<cash_flow_statement>;;;;;
Date_information;9 months 2019;9 months 2018;;;
Profit for the period;9764;10767;;;
Depreciation and amortization of non-current assets;2655;1187;;;
Share of profit of equity accounted investees;-3;-18;;;
(Gains)/ losses on the sale of PPE and IA;23;22;;;
Net finance income / costs;-2546;1865;;;
```

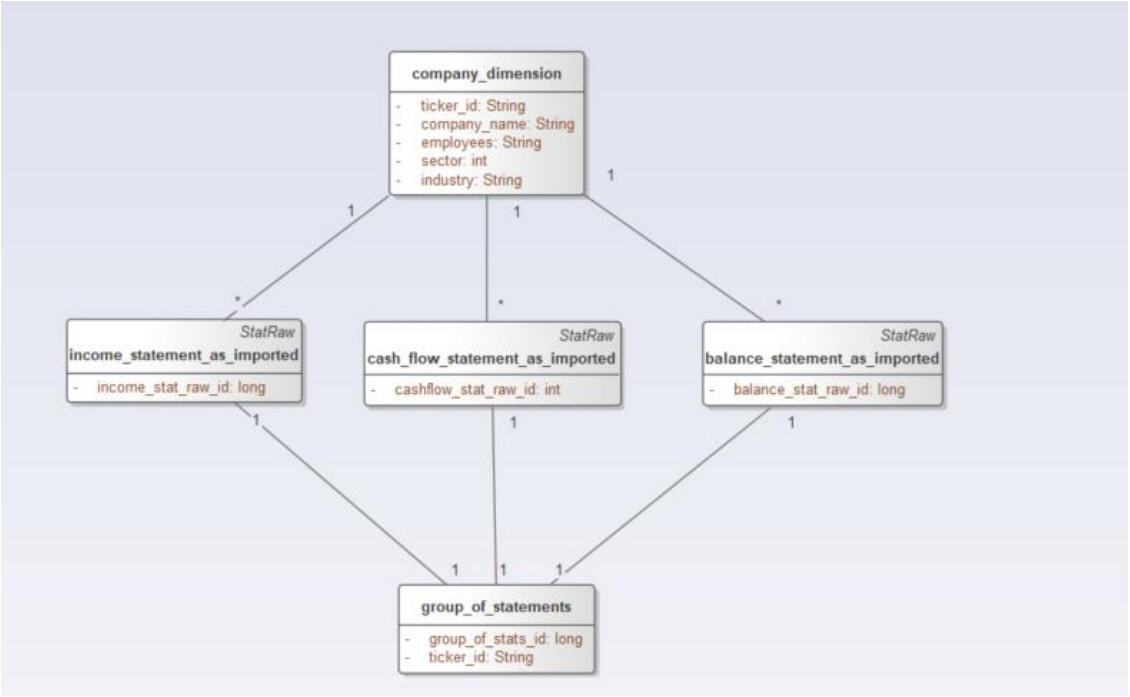
```
<income_statement>,,
Date_information,01.09.2010- 30.11.2010,01.09.2009- 30.11.2009
Revenue (Note 3),203045.00,181307.00
Cost of sales,-167509.00,-148655.00
Gross profit,35536.00,32652.00
Marketing expenses,-16121.00,-14545.00
Administrative expenses,-10236.00,-9347.00
Other income,66.00,452.00
Other expenses,-7.00,-178.00
Results from operating activities,9238.00,9034.00
Finance income (Note 4),6143.00,813.00
Finance costs (Note 4),-14253.00,-11037.00
Profit/-loss before income tax,1128.00,-1190.00
Income tax,0.00,0.00
Net profit/-loss for the period,1128.00,-1190.00
Other comprehensive income/-expense Exchange differences on translating fore
Changes in fair value of cash flow hedges,-554.00,1150.00
Other comprehensive income/-expense for the period,-591.00,419.00
Total comprehensive income/-expense for the period,537.00,-771.00
Profit/-loss attributable to: Equity holders of the parent (Note 5),1128.00,-
Total comprehensive income/-expense attributable to: Equity holders of the pa
Earnings per share (in EUR per share) - basic (Note 5),0.00,0.00
Earnings per share (in EUR per share) - diluted (Note 5),0.00,0.00
</income_statement>,,
<balance_sheet>,,
Date_information,30.11.2010,31.08.2010
Cash and cash equivalents,41843.00,57488.00
Trade and other receivables,35268.00,42040.00
Prepayments,10415.00,9752.00
Derivatives (Note 6),151.00,705.00
Inventories,21044.00,20035.00
Total current assets,108721.00,130020.00
Investments in associates,214.00,214.00
Other financial assets,317.00,317.00
Deferred income tax assets,10664.00,10664.00
```

Joonis 29. Korrektne CSV-formaat semikoolonitega (vasakul) ja ebakorrektne formaat komadega (paremal)

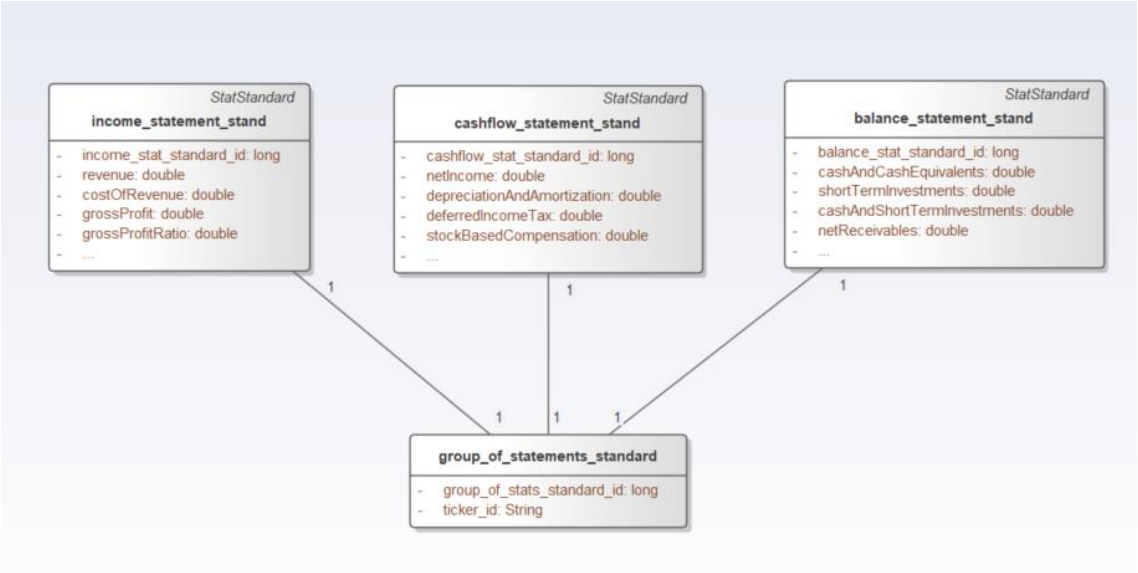
Lisa 6 – Andmebaasi mudelid



Joonis 30. Balti börsile mõeldud investeerimise rakenduse globaalne mudel

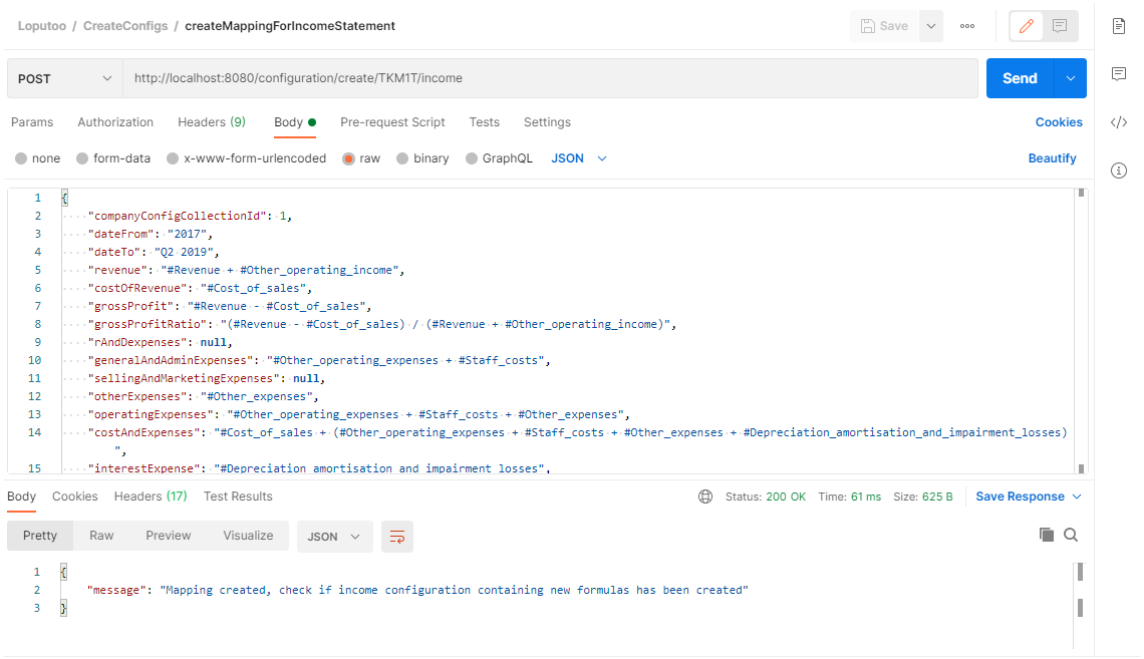


Joonis 31. Group of statements mudel

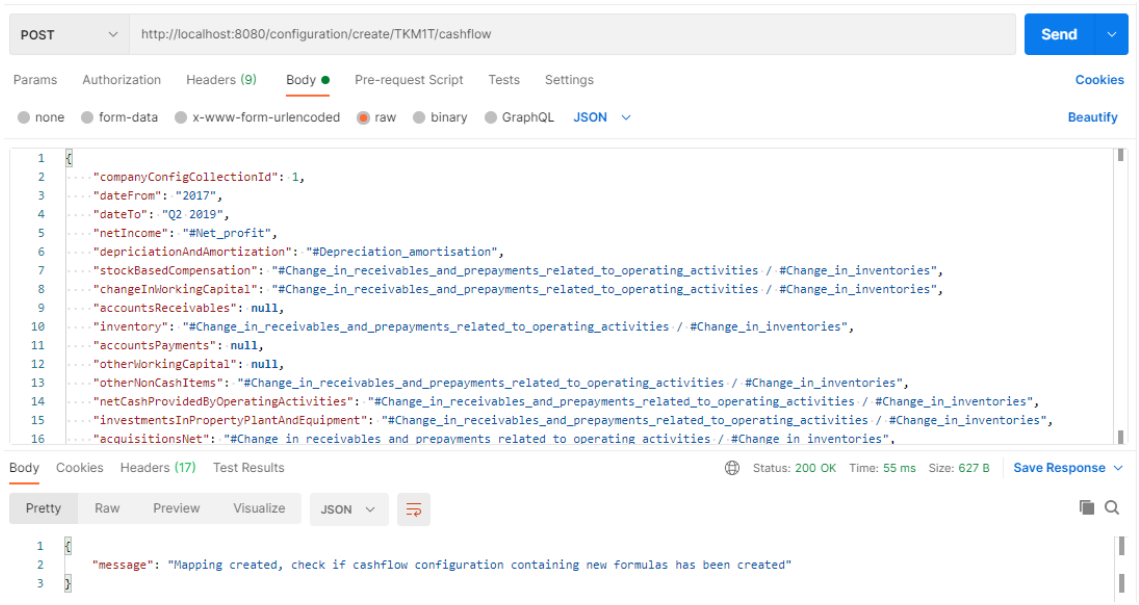


Joonis 32. Group of standard statements mudel

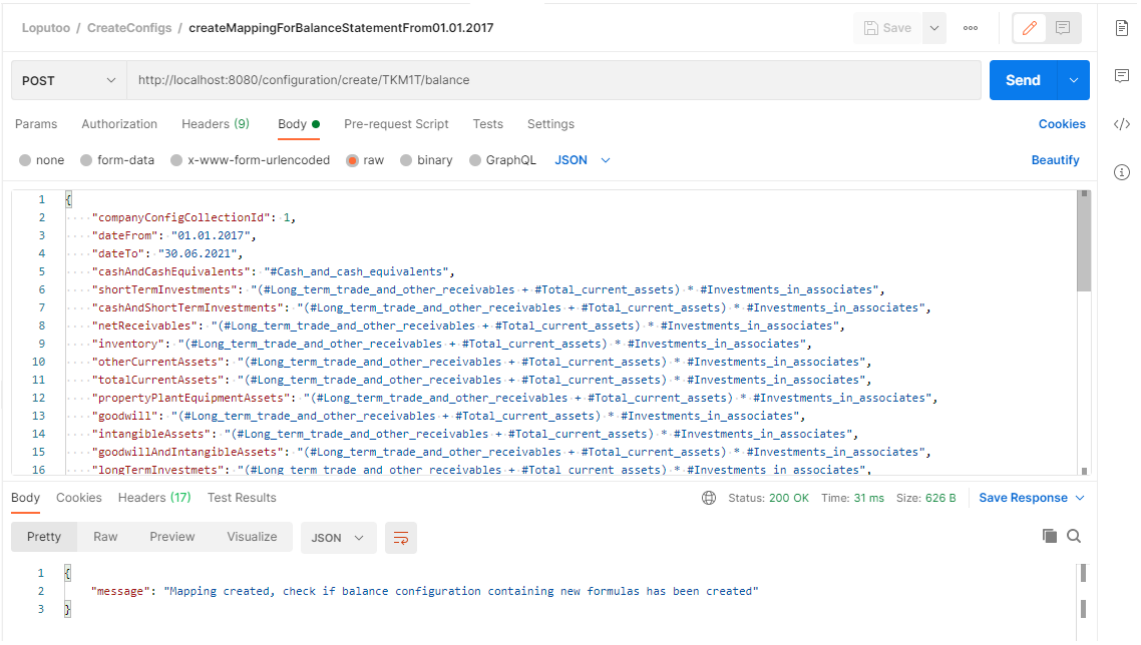
Lisa 7 – Postmani päringud ja tagarakenduse funktsionaalsuse testimine



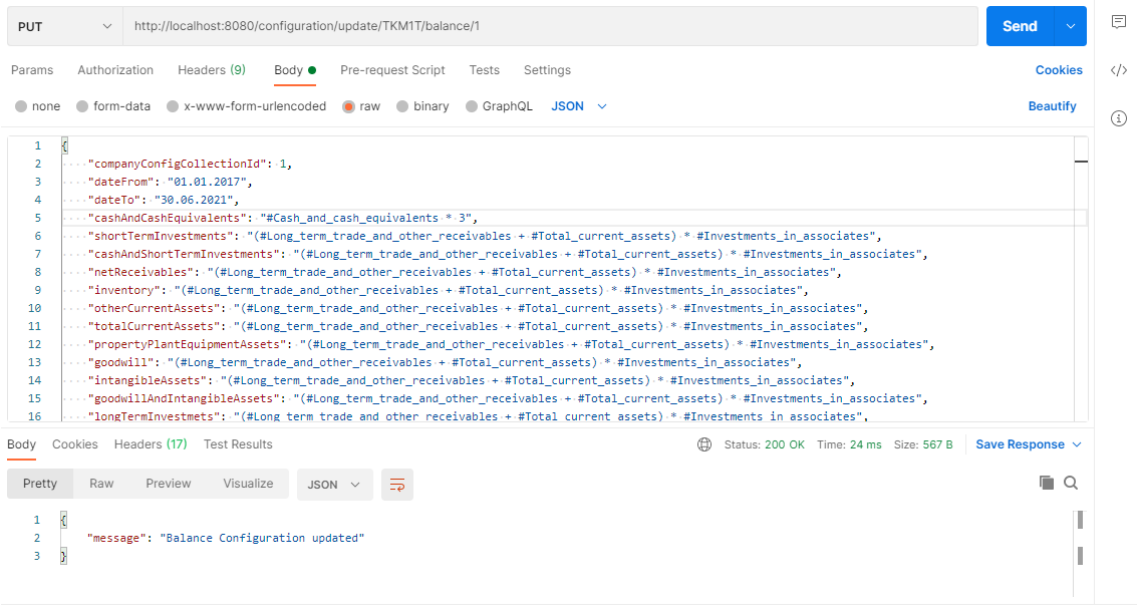
Joonis 33. Kasumiaruande loomine



Joonis 34. Rahavoogude aruande loomine

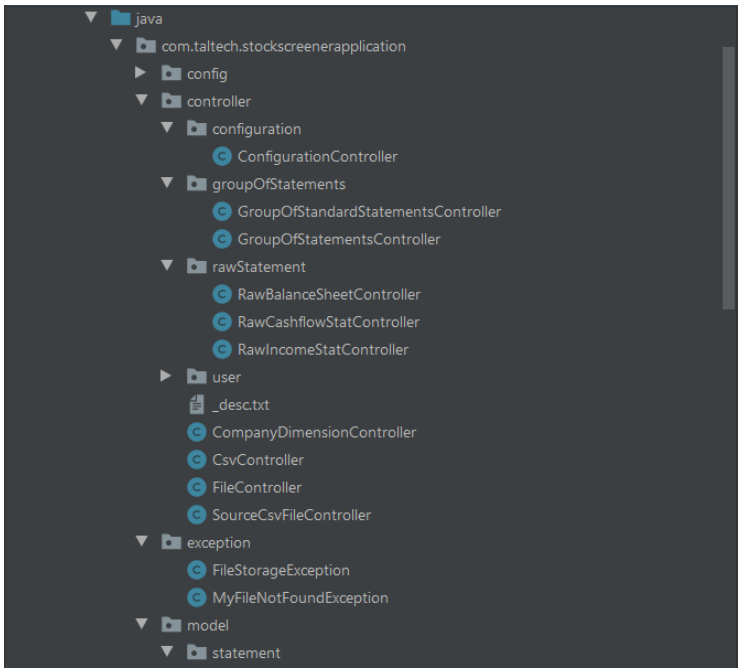


Joonis 35. Bilansi aruande loomine

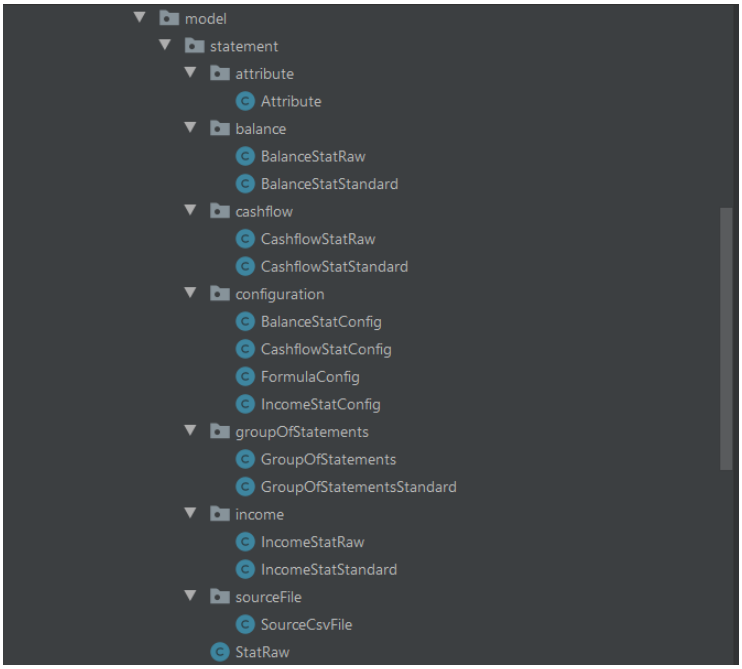


Joonis 36. Bilansi aruande uuendamine

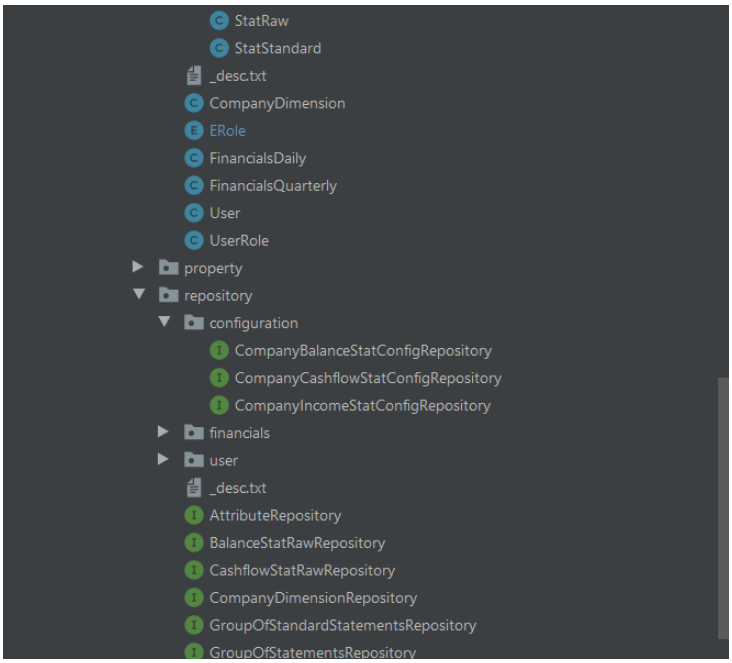
Lisa 8 – Projekti tagarakenduse arhitektuur



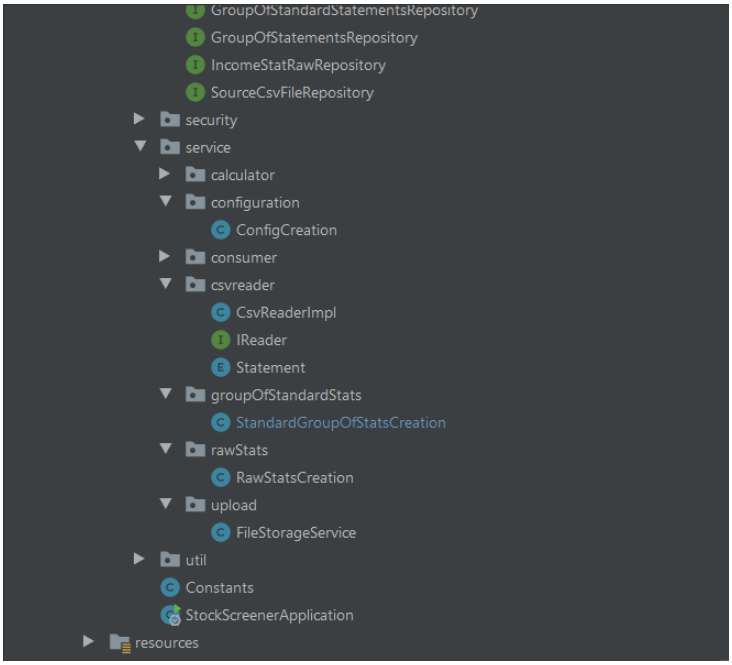
(a)



(b)



(c)



(d)

Joonis 37. Projekti tagarakenduse arhitektuur (a)…(d)

Lisa 9 – Projekti repositooriumite ligipääs

Projekti *scraper*: https://bitbucket.org/KatreHelena/web-scraper/

Tagarakendus: https://github.com/MaJogi/stock-screener-application

Kasutajaliides: https://github.com/MaJogi/stock-screener-react-application