TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Marko Jagor 179582IABB

# VEEBIRAKENDUS TALLINNA BÖRSIETTEVÕTETE FINANTSANDMETE KAJASTAMISEKS

Bakalaureusetöö

Juhendaja: Tõnn Talpsepp
PhD

Tallinn 2020

# Autorideklaratsioon

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

Autor: Marko Jagor

12.05.2020

# Annotatsioon

Töö eesmärgiks on luua veebirakendus, mis kajastab Tallinna börsiettevõtete aktsiatega seotud finantsandmeid, mida investorid saaksid kasutada investeerimisotsuste tegemisel.

Internetis leidub erinevaid rakendusi, mis kajastavad börsiettevõtete finantsandmeid. Üldiselt keskenduvad need välismaistele börsidele, kus on suurem kauplemismaht ning investorite arv. Tallina börsi jaoks puudub mugav lahendus, mis annaks kiirülevaate ettevõtete hetkeseisust ning kuvaks ka ajaloolisi finantsandmeid.

Töö tulemusena valmis esmane demoversioon rakendusest. Rakendus võimaldab kasutajale anda kiirülevaate Tallinna börsiettevõtete finantsolukorrast. Rakenduse demoversioon ei kasuta ettevõtete reaalandmeid. Töös püstitatud eesmärk on väga mahukas ning selle täielik realiseerimine eeldab tulevikus rakenduse edasi arendamist.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 37 leheküljel, 5 peatükki, 16 joonist, 1 tabelit.

# Abstract

Web Application for Presenting Financial Data of Companies Listed on the Tallinn Stock Exchange

The goal of this thesis is to create a web application for presenting financial data of companies listed on the Tallinn Stock Exchange, which the investors could use for making investment decisions.

The Web contains different applications that present the financial data of publicly traded companies. Most of these applications focus on foreign stock exchanges that have a high trading volume and a large number of investors. There doesn't exist a convenient solution for the Tallinn Stock Exchange. A solution, which would give a quick overview of the current financial health of a company and also display historical financial data.

The result of the thesis is a demo version of the application. The application gives a quick overview of the current financial data of the companies listed on the Tallinn Stock Exchange. The demo version of the application doesn't use real financial data. In order to completely fulfil the goal of the application, further development is required in the future.

The thesis is in Estonian and contains 37 pages of text, 5 chapters, 16 figures, 1 tables.

# Lühendite ja mõistete sõnastik

| | |
|---|---|
| Aktsia *screener* | Rakendus/tööriist aktsiate filtreerimiseks kasutaja poolt määratud meetrikate abil. |
| API | *Application Program Interface*, rakendusliides |
| CORS | *Cross-Origin Resource Sharing*, domeeniväline ressursikasutus |
| CSS | *Cascading Style Sheets*, kaskaadlaadistik |
| CSV | *Comma-Separated Values*, komaeraldusega väärtused |
| DOM | *Document Object Model*, dokumendiobjektide mudel |
| HTTP | *Hypertext Transfer Protocol*, hüperteksti edastusprotokoll |
| IPO | *Initial Public Offering*, aktsia esmane avalik pakkumine |
| JSON | *JavaScript Object Notation*, lihtne andmevahetusvorming |
| JVM | *Java Virtual Machine*, Java virtuaalmasin |
| REST | *Representational State Transfer*, arhitektuuri stiil |
| SQL | *Structured Query Language*, struktuurpäringukeel |
| URI | *Uniform Resource Identifier*, ühtne ressursi-indikaator |

# Sisukord

# Jooniste loetelu

# Tabelite loetelu

# 1 Sissejuhatus

Käesoleva bakalaureusetöö teemaks on veebirakenduse arendus, mis kajastab Tallinna börsil kaubeldavate ettevõtete aktsiatega seotud finantsandmeid, mida investorid saaksid kasutada investeerimisotsuste vastuvõtmiseks.

Tänasel päeval leidub internetis väga palju erinevaid aktsiate *screener*-eid. Sellised rakendused kuvavad kasutajale erinevaid finantsandmeid börsil kaubeldavate ettevõtete kohta. Tihti keskendutakse just välismaistele börsidele, kus on suurem kauplemismaht ning investorite arv. Väiksemate kauplemismahtudega turgude jaoks, nagu näiteks Tallinna börs, leidub vähe rakendusi, mis annaks kasutajale terviklikku ning usaldusväärset infot ettevõtte kohta.

Lisaks eelnevalt mainitule, aktsiate *screener*-id näitavad kasutajale üldjuhul ainult reaalajas andmeid. See tähendab, et rakendusest saadud informatsiooni kasutamisel tuleb meeles pidada, et andmed kajastavad ettevõtte hetkeseisu ning see annab kõigest esmase pilgu ettevõttest tervikuna. Pikemaajalist investeerimishorisonti silmas pidades tuleks kindlasti tutvuda ka ettevõtte ajalooliste andmetega, et aru saada, kuidas näiteks erinevad turusituatsioonid on mõjutanud ettevõtte finantsilist käekäiku, näiteks kas ettevõte on kasvule orienteeritud. Sellise spetsiifilisema informatsiooni kättesaamiseks peab kasutaja järgmisest allikast otsima välja ettevõtte aastaaruanded. Selline lahendus ei ole mugav ning on aeganõudev.

Töö eesmärgiks on arendada veebirakendus, mida investorid saaksid kasutada Tallinna börsil igapäevaste investeerimisotsuste vastuvõtmiseks. Rakendus peab näitama kasutajale Tallinna börsi ettevõtete finantsilist hetkeseisu, kuid kasutaja peab soovi korral saama ka ülevaate ettevõttest tervikuna, ehk peab olema võimalik tutvuda ka ajalooliste tulemustega. Rakendus peab olema piisavalt usaldusväärne, et selle abil reaalsete börsitehinguteni jõuda.

Projekti tagarakendus, ehk *backend* pool on üles ehitatud kasutades Java Spring Boot raamistikku ning kasutajaliides, ehk *frontend* pool kasutades JavaScript React-i.

Kiirendamaks kasutajaliidese arendust, on projektis kasutatud PrimeReact-i komponentide kogu. Lisaks on rakenduse üldise väljanägemise muutmiseks kasutatud CSS-i (*Cascading Style Sheets)* võimalusi, et parandada mõningaid disainiprobleeme. Rakenduses kasutatavaid andmeid hoitakse PostgreSQL andmebaasis. Autor on projekti arenduses andnud endast parima, et järgida *clean code* põhimõtteid. Rakenduse töökindluse valideerimiseks on kood suures osas kaetud *unit* testidega.

Töö tulemusena valmis esmane demoversioon rakendusest, mis kuvab kasutajale ettevõtte hetkelise finantsolukorra. Seega töötab rakendus justkui aktsiate *screener*, ehk puudub võimalus näha informatsiooni ettevõtete ajalooliste andmete kohta. Lühidalt kirjeldades, rakenduse praeguse versiooni peamine funktsionaalsus on kuvada kasutajale tabelivaadet Tallinna börsi põhinimekirjas kaubeldavatest aktsiatest.

Töö järgnevates osades antakse täpsem ülevaade projekti läbiviimisest. Kirjeldatakse projekti ideed, olemasolevaid rakendusi, erinevate tehnoloogiate kasutamist ning tööprotsessi ülesehitust. Samuti tuuakse välja, millised olid peamised töö tulemused. Seejuures selgitatakse projektile seatud nõudeid, kasutatud arhitektuuri ja disaini. Lisaks tuuakse välja olulisemaid osasid koodist ning antakse ülevaade testimisest. Lõpetuseks analüüsitakse projektis tehtud valikuid, korratakse üle peamised töö tulemused ning arutatakse tulevikus edasiarendamise võimalusi.

# 2 Projekti ülevaade

Töö idee sai alguse autori huvist investeerimise vastu, mis on tekkinud seoses selliste Tallinna Tehnikaülikooli õppeainetega nagu Eraisiku rahandus ning Rahanduse alused, lisaks üleüldine ühiskonna teadlikkuse kasvamine investeerimisest ja olukord, kus investeerimist saab alustada väga väikeste summadega, ilma et sellel oleks mõju investori tootlusele.

Nagu sissejuhatuses mainitud, on projekti eesmärgiks pikas perspektiivis arendada valmis Tallinna börsile keskenduv toimiv veebirakendus, mis samaaegselt annaks kiire ülevaate ettevõtte hetkelisest finantsseisust ning lubaks tutvuda ka ettevõtete ajalooliste tulemustega. Selle rakenduse abil peab investor jõudma otsusele, kas investeerida ettevõttesse või mitte. Kuigi internetis on palju erinevaid veebilehti ning rakendusi, mida kasutada informatsiooni kogumiseks, sealhulgas Tallinna börsilt, siis autor leiab, et olemasolevad lahendused on kas puudulike andmetega või jätab soovida nende kasutamismugavus.

## 2.1 Olemasolevad lahendused

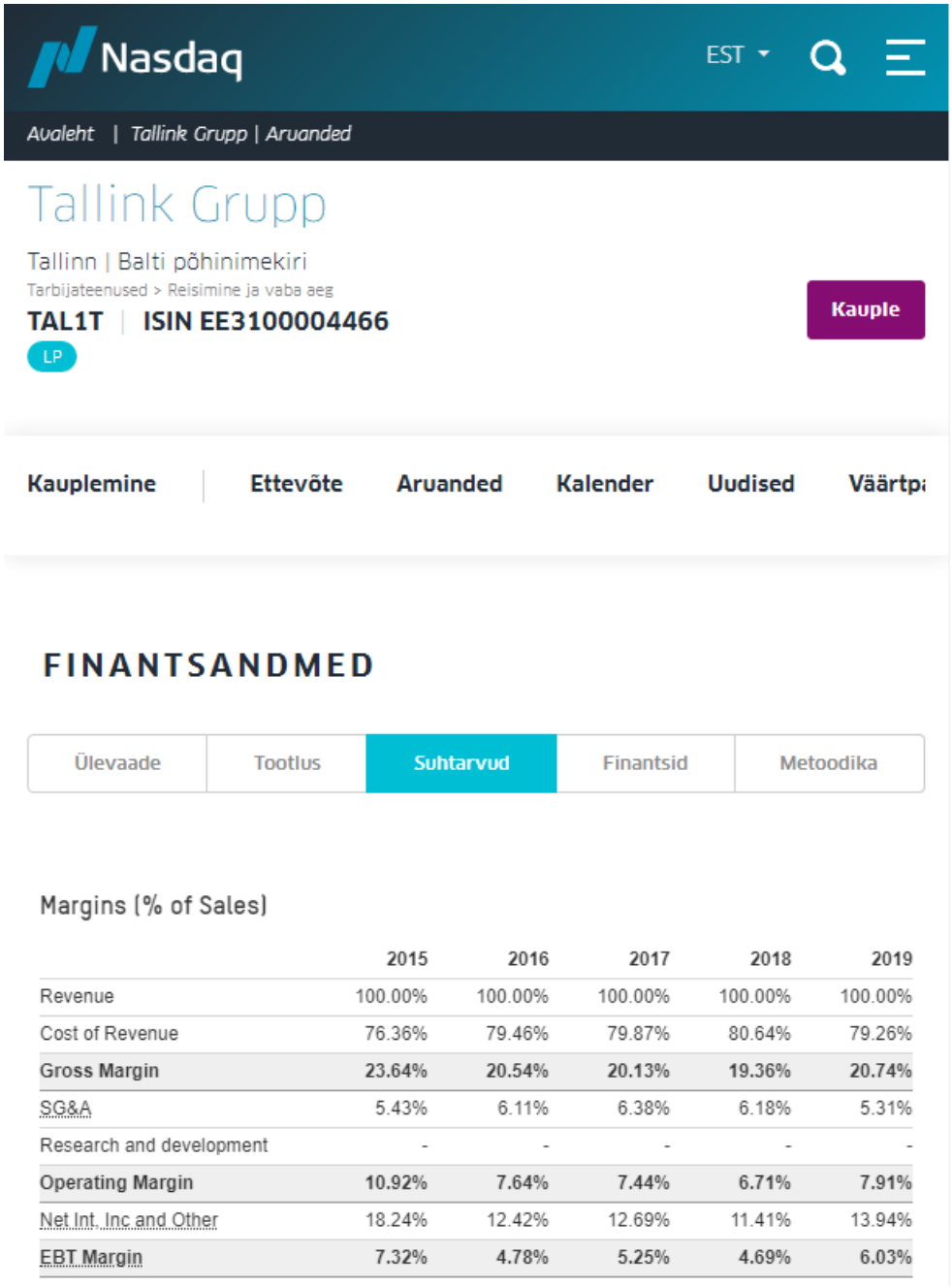
Järgnevalt tutvustatakse mõningaid olemasolevaid lahendusi, mille abil Tallinna börsilt informatsiooni leida.

### 2.1.1 Nasdaq Baltic koduleht

Nasdaq Balti turg esindab Nasdaqi börse Tallinnas, Riias ja Vilniuses. Investori vaatenurgast moodustavad Nasdaq Balti börsid ühe koha, kus investeerida suurimatesse ja tuntuimatesse ettevõtetesse. Ettevõtte vaatenurgast on Nasdaqi börsid Tallinnas, Riias ja Vilniuses ainukeseks reguleeritud börsiks Baltikumis, mis pakub börsil olevate ettevõtetele kõiki teenuseid [1].

Antud leht on ilmselt kõige usaldusväärsem allikas Tallinna ning üleüldse kogu Balti börsil investeerimiseks. Investori seisukohast võib leheküljelt leida informatsiooni nii

aktsiate, võlakirjade, fondide ning indeksite kohta (Joonis 1). Samuti võimaldab leht kursis olla erinevate börsiteadetega.

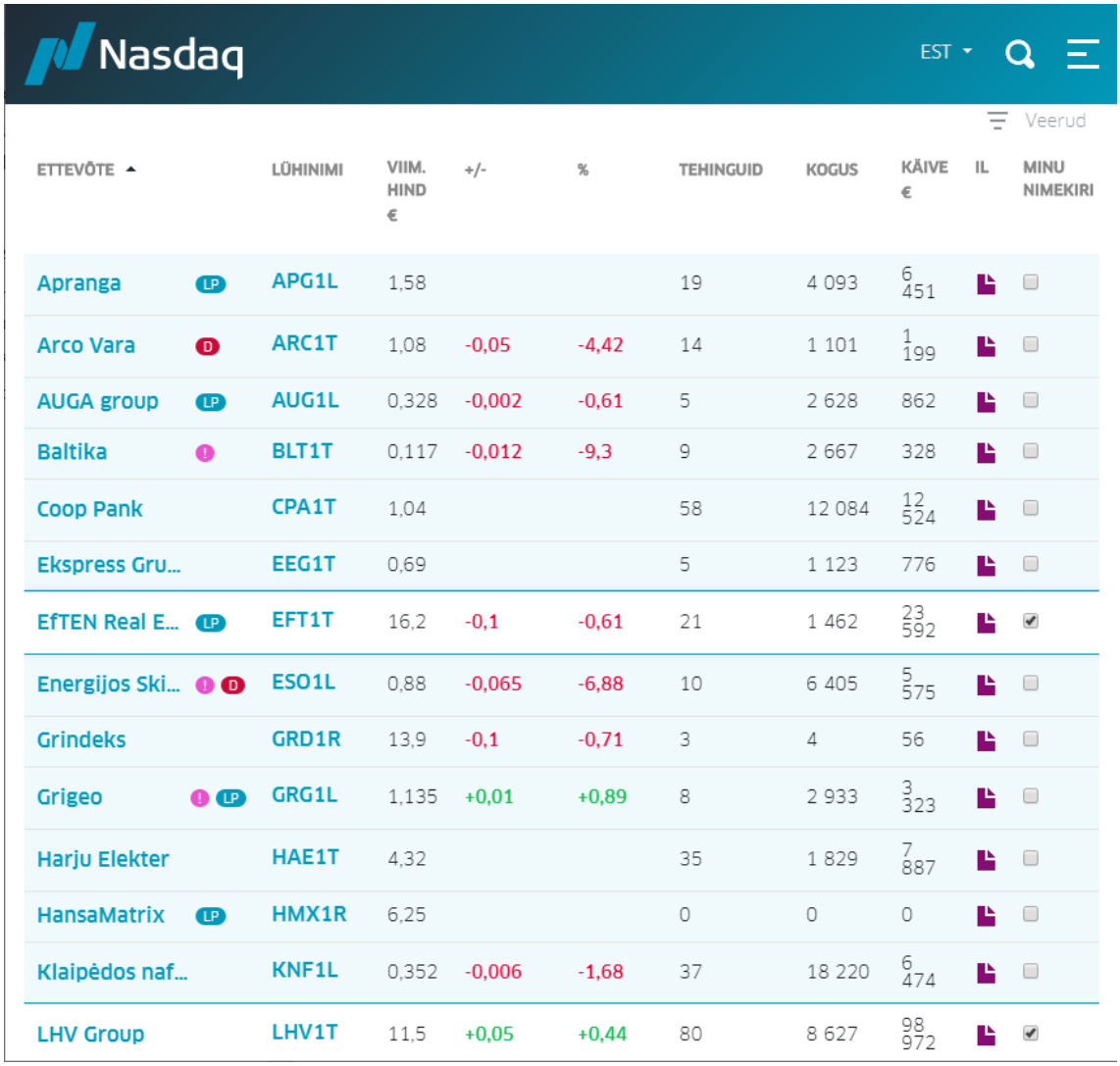



| | 2015 | 2016 | 2017 | 2018 | 2019 |
|---|---|---|---|---|---|
| Revenue | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% |
| Cost of Revenue | 76.36% | 79.46% | 79.87% | 80.64% | 79.26% |
| **Gross Margin** | **23.64%** | **20.54%** | **20.13%** | **19.36%** | **20.74%** |
| SG&A | 5.43% | 6.11% | 6.38% | 6.18% | 5.31% |
| Research and development | - | - | - | - | - |
| **Operating Margin** | **10.92%** | **7.64%** | **7.44%** | **6.71%** | **7.91%** |
| Net Int. Inc and Other | 18.24% | 12.42% | 12.69% | 11.41% | 13.94% |
| **EBT Margin** | **7.32%** | **4.78%** | **5.25%** | **4.69%** | **6.03%** |

Joonis 1. Kuvatõmmis Nasdaq Baltic veebilehelt ettevõtte finantsandmetest Tallink Grupp näitel.

Veebilehe suurimaks eeliseks ongi just selle usaldusväärsus, kuna tegemist on siiski reguleeritud süsteemiga, millele teostatakse finantsinspektsiooni poolt järelvalvet. Seega on tegemist leheküljega, kuhu esmajoones suunduda, kui on huvi Tallinna või kogu Balti börsi vastu.

Kuna Nasdaq Baltic on kogu Baltikumi börsi esindaja, siis selle suurimaks miinuspooleks ongi see, et informatsiooni kättesaadavus ettevõtete kohta on vaid üks osa selle veebilehe funktsionaalsusest. See ei paku teenuseid ainult investorile ning informatsiooni leidmine võib teatud määral aega võtta. Samuti puudub võimalus saada kiiret ülevaadet kõikide ettevõtete kohta ning igat huvipakkuvat objekti tuleb uurida eraldi ja soovi korral võrdlusmoment endale käsitsi tekitada (Joonis 2).




| ETTEVÕTE | LÜHINIMI | VIIM. HIND € | +/- | % | TEHINGUID | KOGUS | KÄIVE € | IL | MINU NIMEKIRI |
|---|---|---|---|---|---|---|---|---|---|
| Apranga LP | APG1L | 1,58 | | | 19 | 4 093 | 6 451 | | |
| Arco Vara D | ARC1T | 1,08 | -0,05 | -4,42 | 14 | 1 101 | 1 199 | | |
| AUGA group LP | AUG1L | 0,328 | -0,002 | -0,61 | 5 | 2 628 | 862 | | |
| Baltika ! | BLT1T | 0,117 | -0,012 | -9,3 | 9 | 2 667 | 328 | | |
| Coop Pank | CPA1T | 1,04 | | | 58 | 12 084 | 12 524 | | |
| Ekspress Gru... | EEG1T | 0,69 | | | 5 | 1 123 | 776 | | |
| EfTEN Real E... LP | EFT1T | 16,2 | -0,1 | -0,61 | 21 | 1 462 | 23 592 | | ✓ |
| Energijos Ski... ! D | ESO1L | 0,88 | -0,065 | -6,88 | 10 | 6 405 | 5 575 | | |
| Grindeks | GRD1R | 13,9 | -0,1 | -0,71 | 3 | 4 | 56 | | |
| Grigeo ! LP | GRG1L | 1,135 | +0,01 | +0,89 | 8 | 2 933 | 3 323 | | |
| Harju Elekter | HAE1T | 4,32 | | | 35 | 1 829 | 7 887 | | |
| HansaMatrix LP | HMX1R | 6,25 | | | 0 | 0 | 0 | | |
| Klaipėdos naf... | KNF1L | 0,352 | -0,006 | -1,68 | 37 | 18 220 | 6 474 | | |
| LHV Group | LHV1T | 11,5 | +0,05 | +0,44 | 80 | 8 627 | 98 972 | | ✓ |

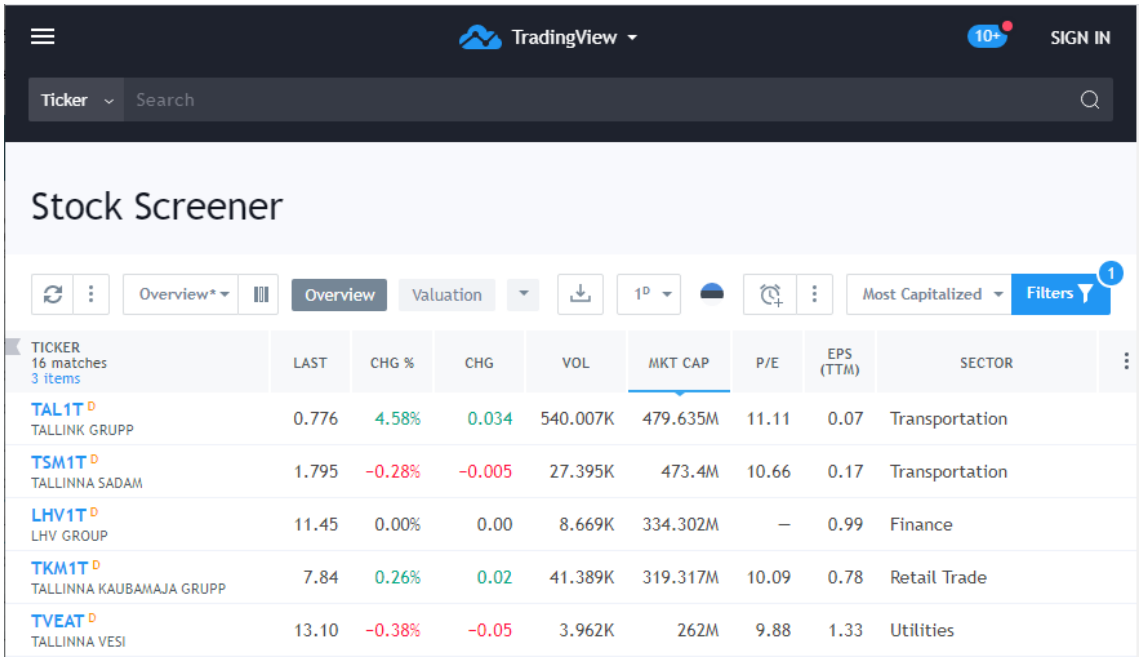
Joonis 2. Kuvatõmmis Nasdaq Baltic veebilehelt Balti börsi üldnimekirjast.

### 2.1.2 TradingView

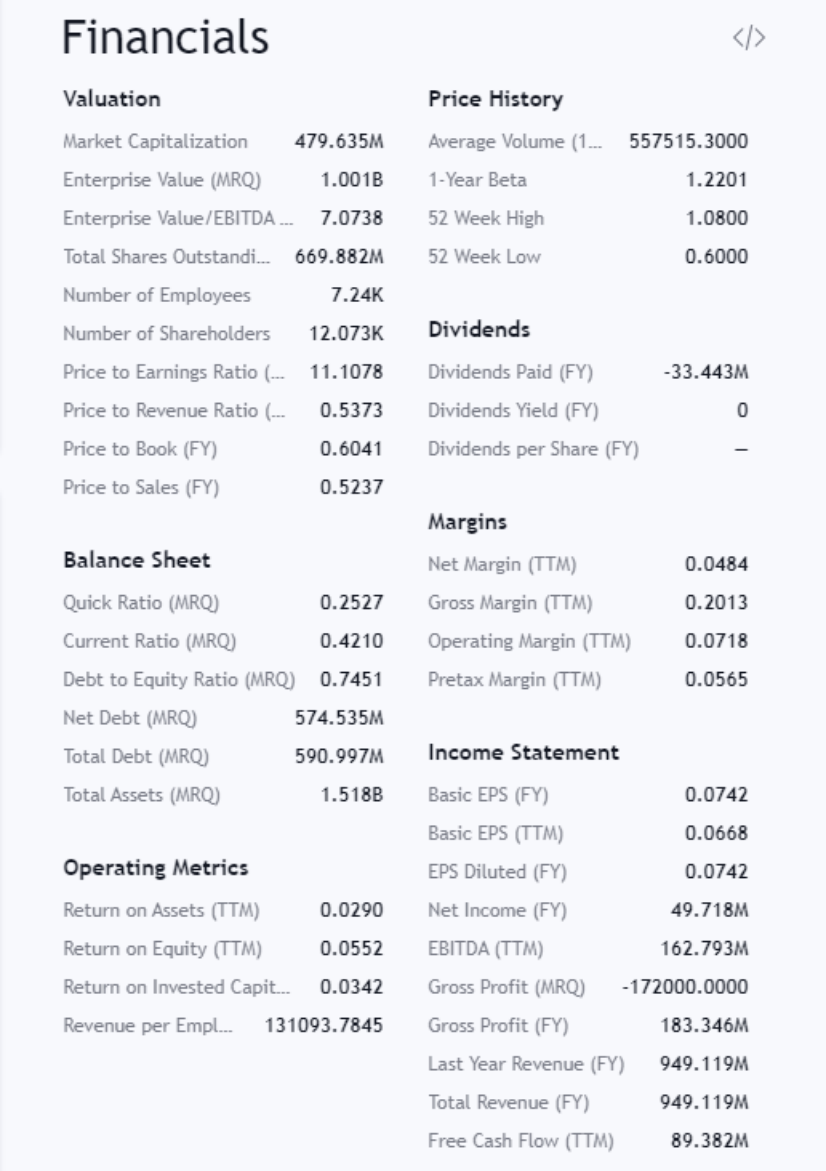
TradingView reklaamib end kui võimsat tööriista nii algajatele kui ka kogenud investoritele. Rakendus sobib kõige lihtsamateks tegevusteks nagu aktsia viimase hinna vaatamiseks reaalajas kui ka keerulisteks tehnilisteks analüüsideks [2].

TradingView on kõige aktiivsem sotsiaalvõrgustik kauplejatele ning investoritele. Selle abil saab suhelda investoritega üle maailma, arutada investeerimisideid ja nii edasi. See katab üle 50 erineva börsi üle maailma, sealhulgas ka Tallinna börsi. Informatsiooni leiab nii aktsiate, futuuride, suuremate indeksite, valuutade ning krüptovaluutade kohta [2].

Rakenduse suurimaks eeliseks on selle väga lihtne kasutamine ning võimalus saada kiire ülevaade börsil kaubeldavatest aktsiatest (Joonis 3). Kasutajal on võimalus filtreerida ettevõtteid enda poolt paika pandud kriteeriumite järgi, välistades enda jaoks mittesobivad ettevõtted. Rakendus annab kasutajale võimaluse ka täielikult enda soovidele vastav tabel luua, ehk korraga näha kõikide ettevõtete kohta just neid andmeid, mida parasjagu näha tahetakse. Sobiva ettevõtte leidmisel on võimalik tema praeguste finantsnäitajatega lähemalt tutvuda (Joonis 4).




Joonis 3. Kuvatõmmis TradingView veebilehelt Tallinna börsi üldnimekirjast.



Financials

Valuation

| | |
|---|---|
| Market Capitalization | 479.635M |
| Enterprise Value (MRQ) | 1.001B |
| Enterprise Value/EBITDA ... | 7.0738 |
| Total Shares Outstandi... | 669.882M |
| Number of Employees | 7.24K |
| Number of Shareholders | 12.073K |
| Price to Earnings Ratio (... | 11.1078 |
| Price to Revenue Ratio (... | 0.5373 |
| Price to Book (FY) | 0.6041 |
| Price to Sales (FY) | 0.5237 |

Balance Sheet

| | |
|---|---|
| Quick Ratio (MRQ) | 0.2527 |
| Current Ratio (MRQ) | 0.4210 |
| Debt to Equity Ratio (MRQ) | 0.7451 |
| Net Debt (MRQ) | 574.535M |
| Total Debt (MRQ) | 590.997M |
| Total Assets (MRQ) | 1.518B |

Operating Metrics

| | |
|---|---|
| Return on Assets (TTM) | 0.0290 |
| Return on Equity (TTM) | 0.0552 |
| Return on Invested Capit... | 0.0342 |
| Revenue per Empl... | 131093.7845 |

Price History

| | |
|---|---|
| Average Volume (1... | 557515.3000 |
| 1-Year Beta | 1.2201 |
| 52 Week High | 1.0800 |
| 52 Week Low | 0.6000 |

Dividends

| | |
|---|---|
| Dividends Paid (FY) | -33.443M |
| Dividends Yield (FY) | 0 |
| Dividends per Share (FY) | — |

Margins

| | |
|---|---|
| Net Margin (TTM) | 0.0484 |
| Gross Margin (TTM) | 0.2013 |
| Operating Margin (TTM) | 0.0718 |
| Pretax Margin (TTM) | 0.0565 |

Income Statement

| | |
|---|---|
| Basic EPS (FY) | 0.0742 |
| Basic EPS (TTM) | 0.0668 |
| EPS Diluted (FY) | 0.0742 |
| Net Income (FY) | 49.718M |
| EBITDA (TTM) | 162.793M |
| Gross Profit (MRQ) | -172000.0000 |
| Gross Profit (FY) | 183.346M |
| Last Year Revenue (FY) | 949.119M |
| Total Revenue (FY) | 949.119M |
| Free Cash Flow (TTM) | 89.382M |

Joonis 4. Kuvatõmmis TradingView veebilehelt ettevõtte finantsandmetest Tallink Grupp näitel.

Rakenduse suurimaks miinuseks on see, et kohati ei ole andmed terviklikud. See tähendab, et näiteks väiksemate börside puhul, nagu seda on Tallinna börs, võivad puududa andmed mõne finantsnäitaja kohta. Seega ei pruugi investor saada adekvaatset võrdlusmomenti mõne teise ettevõttega. Samuti puudub rakenduses informatsioon ettevõtte ajalooliste finantstulemuste kohta, sest kajastatakse ainult viimaseid tulemusi. See ei anna tervikpilti ettevõttest ning läbimõeldud investeerimisotsuse tegemiseks tuleks otsida järgmiselt veebilehelt välja ettevõtte finantsaruanded.

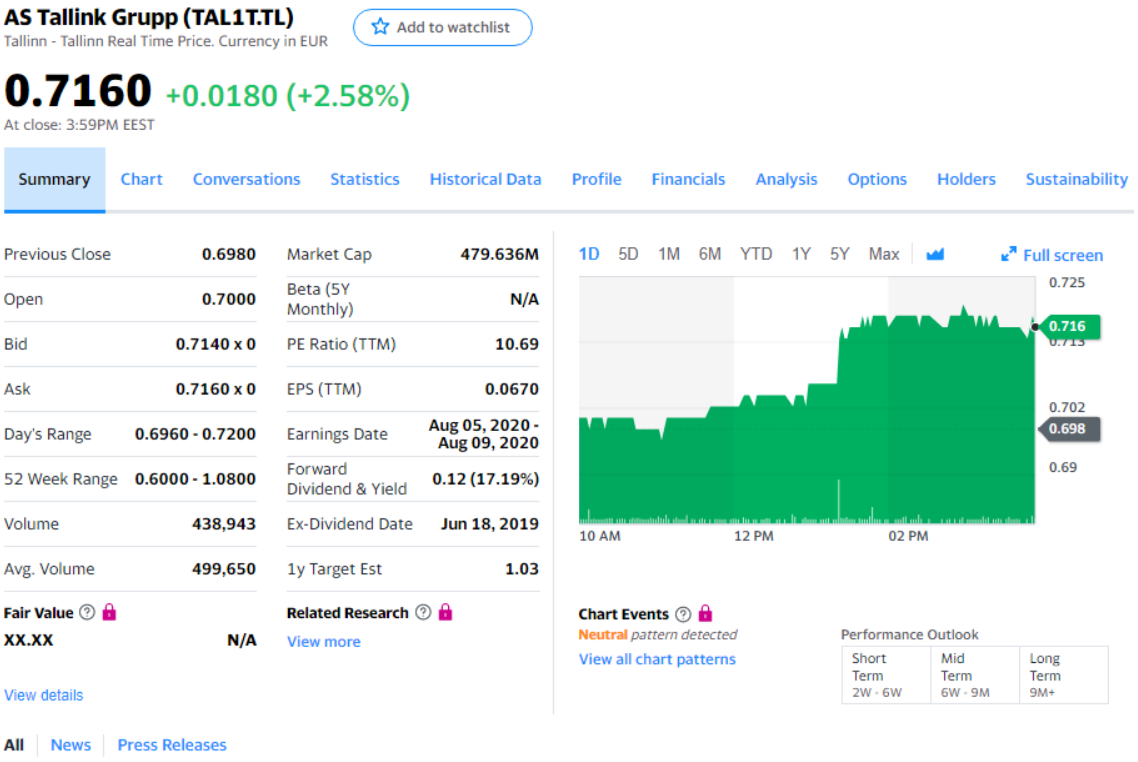
### 2.1.3 Yahoo Finance

Yahoo Finance veebilehel on kättesaadav suur hulk finantsinformatsiooni, erinevaid tööriistu ning uudisallikaid. Kasutajal on võimalik otsida ettevõtete finantsandmeid,

analüüsida erinevate aktsiate hinnagraafikuid ja filtreerida ettevõtteid tema finantsnäitajate järgi [3].

Veebilehelt on võimalik leida finantsuudiseid, süvaanalüüse ning kommentaare erinevate ettevõtete kohta. Samuti võimaldab lehekülg kursis olla börsiteadetega, mis puudutavad kasumiaruandeid, IPO-sid (*Initial Public Offering*) ja nii edasi. Lisaks on võimalik luua enda virtuaalne portfell, kuhu koondada endale huvi pakkuvate ettevõtete aktsiad [3].

Yahoo Finance puhul on jällegi tegemist väga võimsa tööriistaga, mille abil on võimalik uurida erinevaid ettevõtteid (Joonis 5). Informatsiooni on palju ning lehekülg kajastab ka ettevõtte ajaloolisi andmeid.




Joonis 5. Kuvatõmmis Yahoo Finance veebilehelt ettevõtte finantsandmetest Tallink Grupp näitel.

Yahoo Finance miinuseks on jällegi kohatine informatsiooni puudulikkus, kui rääkida Tallinna börsi ettevõtetest. Kõik andmed ei ole kättesaadavad. Samuti on suur osa ajaloolisest infost tasuline ning arvestades, et andmed on niigi osaliselt puudulikud, siis autori hinnangul ei õigustaks see vähemalt Tallinna börsi puhul sellise kulutuse tegemist.

Yahoo Finance veebileheküljel on üsna keeruline ülesehitus ning erineva informatsiooni või tööriistade leidmine võtab üksjagu aega. Mõnes mõttes on see loogiline, sest tegemist on väga mahuka lehekülgega, kus on väga palju erinevaid funktsionaalsusi.

## 2.2 Kasutatud tehnoloogiad

Antud peatükis antakse lühike ülevaade töös kasutatud tehnoloogiatest.

### 2.2.1 Arenduskeskkond

Projekti arendus toimub IntelliJ IDEA programmiga. IntelliJ IDEA on arenduskeskkond, mis on mõeldud JVM (*Java Virtual Machine*) programmeerimiskeeltes arenduseks. Tänu erinevatele *plugin*-itele toetab IntelliJ IDEA ka JavaScript-i, SQL-i (*Structured Query Language*) ja CSS-i kasutamist, mida projektis vaja läheb [4].

### 2.2.2 Projekti hoidla

Projekt on salvestatud Bitbucket-i hoidlasse. Bitbucket on Git-il baseeruv koodi hoidmiseks ning tiimisiseseks koostööks mõeldud tööriist [5].

Kuna projektiga alustamisel arvestati, et hilisema edasiarenduse puhul võib mõni arendaja lisaks tulla, siis kasutatakse koodi hoidmiseks just seda hoidlat. Lisaks kindlustab selline koodi hoidmine selle, et kui arendaja riistvaras tekib mõni tõsisem viga, ei hävine kogu tehtud töö.

Projekt on ajutiselt kättesaadav ka läbi GitHub-i linkide (Lisa 1 – GitHub-i lingid).

### 2.2.3 Projekti tagarakendus

Projekti tagarakenduse, ehk *backend* pooles on kasutatud Java programmeerimiskeelt ning koodi paremat haldamist toetab Spring Boot raamistik. Spring Boot lubab kasutajal lihtsa vaevaga luua Spring-i baasil aplikatsioone. Enamik Spring Boot aplikatsioone vajavad minimaalset konfigureerimist, et projekt käivitada [6].

Spring Boot-i kasutamine hoiab projekti arendamisel väga palju aega kokku. Suurem osa seadistustest on kasutaja jaoks varem ära tehtud ning seega saab üsna kiirelt alustada oma projekti arendamisega.

### 2.2.4 Projekti kasutajaliides

Projekti kasutajaliidese, ehk *frontend* pooles on kasutatud JavaScript-i programmeerimiskeelt ning JavaScript-ile mõeldud React-i raamistikku. React on mõeldud kasutajaliideste arendamiseks. Selle abil on võimalik luua keerukaid kasutajaliideseid, jagades need väiksemateks koodijuppideks, mida nimetatakse komponentideks [7].

Kiiremaks kasutajaliidese arendamiseks on kasutatud PrimeReact-i komponentide kogu, mis on mõeldud kasutamaks koos React-i raamistikuga. PrimeReact sisaldab üle 70 lihtsasti kasutatava komponendi, et vastata kõikidele võimalikele kasutajaliideste nõuetele [8].

Lisaks eelnevale on kasutajaliidese arenduses kasutatud ka CSS märgistuskeelt, et disainida veebilehel olevaid elemente vastavalt vajadusele.

### 2.2.5 Andmebaas

Projekti andmebaasina kasutatakse PostgreSQL-i andmebaasi. PostrgreSQL on võimas, avatud lähtekoodiga objekt relatsiooniline andmebaasisüsteem, mis laiendab SQL-i päringukeelt. See võimaldab turvaliselt säilitada ka kõige keerulisemaid andmekogusid [9].

## 2.3 Tööprotsess

Projekti arendusmetoodikana otsustati kasutada *scrum* metoodikat. *Scrum* on agiilne metoodika arendamaks innovatiivseid tooteid ning teenuseid [10, lk 1].

Agiilse lähenemise puhul on vaja luua tootele võlgnevus, ehk *backlog* – nimekiri nõudmistest ning võimekusest, mida on vaja, et arendada valmis edukas toode. Neid nõudmisi kirjeldatakse töö edasises peatükis „Nõuded rakendusele“. Juhituna toote võlgnevusest, töötatakse alati esmalt kõrgema prioriteediga nõudmiste kallal [10, lk 1].

Töö viiakse läbi lühikestes iteratsioonides, mis tavaliselt varieeruvad oma pikkuselt nädala kuni kuuni. Iga iteratsiooni puhul tehakse ära kogu töö, sealhulgas kodeerimine, disainimine ning testimine, mida on vaja, et muuta mingi osa lõpptoodangu kõlbulikuks [10, lk 1]. Selle projekti puhul lepiti kokku, et ühe iteratsiooni pikkuseks võiks olla umbes kaks nädalat.

Juhendajaga suhtlemiseks loodi suhtluskanalisse Slack grupivestlus. Selle kanali kaudu sai kiirelt lahendada erinevad küsimused ja probleemid, kui neid tekkis. Samuti liikus selle kanali kaudu vajalik info projektiarengust.

Lisaks olid planeeritud ka aegajalt toimuvad koosolekud, mille eesmärgiks oleks olnud koostöös juhendajaga projekti detailsem läbi vaatamine ning keerulisemate küsimuste lahendamine. Seoses viimastel kuudel riigis kehtinud eriolukorraga, ei tekkinud paraku üldse võimalusi näost näkku kohtumiseks. Seega lahendati kõik olulisemad küsimused ning probleemid ära Slack keskkonnas.

Tänu rakenduses paika pandud nõudmistele oli võimalik projekt väiksemateks osadeks jaotada. See omakorda aitas pidevalt jälgida, kuidas projekti tervikuna areneb ning millised ülesanded vajavad järgmisena tegemist. Projekti iteratsioonideks jaotamine aitas autoril paremini aega planeerida, et seatud eesmärgid saaksid võimalikult suures ulatuses õigeks ajaks täidetud.

# 3 Töö tulemused

Antud peatükis tuuakse välja töö peamised tulemused. Esmalt loetletakse rakendusele seatud nõuded, mida arendamisel jälgiti. Seejärel kirjeldatakse töös järgitud arhitektuuri ning disainimustreid. Viimaks selgitatakse töös kirjutatud koodi ning testimistulemusi.

## 3.1 Nõuded rakendusele

Analüüsides peatükis „Olemasolevad lahendused“ välja toodud rakendusi, on arendataval finantsrakendusel järgnevad nõuded:

### 3.1.1 Funktsionaalsed nõuded

- Investoril on ettevõtete tabelis nähtavad kõik Tallinna börsil kaubeldavad aktsiad, et tekiks üldpilt võimalikest valikutest.
- Investoril on võimalik saada ettevõtte kohta täpsemat informatsiooni, sealhulgas ajaloolisi andmeid, et otsustada, kas tegemist võiks olla sobiva investeeringuga.
- Investoril on võimalik filtreerida ettevõtteid enda valitud kriteeriumite järgi, et välistada mittesobivad ettevõtted.
- Investoril on võimalik ettevõtteid omavahel võrrelda, et paremini aru saada, milline võiks olla parim investeering.
- Investoril on ettevõtete tabelis võimalus valida, milliseid finantsandmeid talle parasjagu kuvatakse.
- Investoril on võimalik kiiresti üles otsida konkreetne ettevõte, et sellega lähemalt tutvuda.
- Investoril on võimalik luua endale jälgimisnimekiri talle huvi pakkuvatest aktsiatest.

### 3.1.2 Mittefunktsionaalsed nõuded

- Rakendus peab olema kasutatav Google Chrome ja Mozilla Firefox brauseri värskeimas versioonis.
- Rakendus peab kiirelt vastama kasutaja andmepäringutele. Andmepäringu kiirus peaks olema alla kolme sekundi.

- Rakendus peab olema arendatud nii, et sinna on võimalus lihtsa vaevaga juurde lisada uusi funktsionaalsusi.
- Rakenduses kasutatavad andmed peavad uuenema vähemalt korra ööpäeva jooksul.
- Rakenduse ettevõtete tabelis olevad andmed peavad olema loogiliselt esitatud/kategooriatesse jaotatud, et lihtsasti üles leida kõik soovitud finantsnäitajad.
- Rakenduse funktsioonid ei tohi olla kasutaja jaoks liiga keerulised.

## 3.2 Arhitektuur

Arendatav rakendus on jaotatud kaheks projektifailiks. Esimene projekt tegeleb peamiselt CSV (*Comma-Separated Values*) failide lugemisega ja kirjutamisega andmebaasi ning lisaks andmebaasist andmepäringute tegemisega. Teine projekt küsib esimesest projektist andmed, vajadusel töötleb neid ning presenteerib läbi kasutajaliidese kliendile. Sisuliselt on projekt tervikuna jaotatud tagarakenduseks ja kasutajaliideseks. Samuti on projektil olemas andmebaas.

### 3.2.1 Tagarakenduse arhitektuur

Projekti tagarakenduse osa puhul on kasutatud kihilist arhitektuuri (Joonis 6), mille olemust on kirjeldanud Martin Fowler [11]. Selline arhitektuur annab võimaluse igat projekti osa vaadelda eraldi, mis tähendab, et erinevad koodi osad ei ole omavahel nii tihedalt seotud. Seega saab iga koodijupiga tegeleda selliselt, et tehtud muudatustel on minimaalne mõju teistele klassidele ning meetoditele. Samuti lubab selline arhitektuur hoida projekti sisu organiseerituna ning lihtsustab kogu projekti edasiarendamist.

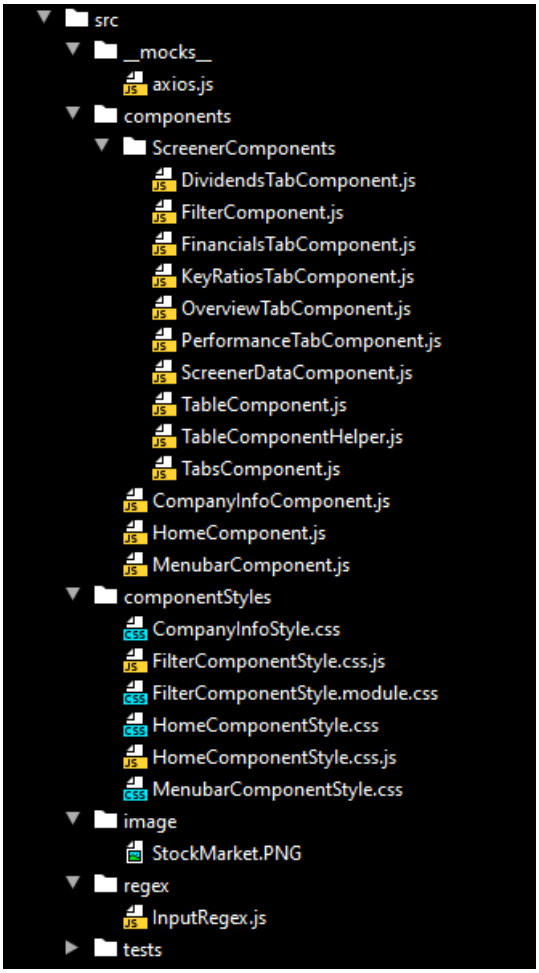

Joonis 6. Projekti tagarakenduse arhitektuur.

Kaustas *src\main\java\com\taltech\stockscreenerapplication\model* olevad klassid esindavad erinevaid andmetabeleid ning läbi nende lisatakse tabelitele omavahelised ühendused. Kaustas *src\main\java\com\taltech\stockscreenerapplication\repository* olevad klassid annavad võimaluse andmebaasis olevatele objektidele erinevaid andmeoperatsioone teostada. Kaustas *src\main\java\com\taltech\stockscreenerapplication\controller* olev klass tagastab päringu peale erinevaid andmeid. Kaustades *src\main\java\com\taltech\stockscreenerapplication\batch* ja *src\main\java\com\taltech\stockscreenerapplication\config* olevate klasside peamisteks ülesanneteks on CSV failides olevate andmete lugemine ja kirjutamine andmebaasi tabelitesse ning andmevahetuse võimaldamine kasutajaliidese projektiga. Kaustas *src\main\resources\data* on erinevad CSV failid, millest loetakse andmed andmebaasi

tabelitesse. Lisaks on olemas *src\test* kaust, milles on erinevate klasside meetodite testid.

### 3.2.2 Kasutajaliidese arhitektuur

Projekti kasutajaliidese osa puhul on kasutatud pigem monoliitset arhitektuuri (Joonis 7). See tähendab, et projekt on disainitud töötama iseseisva üksusena. Rakenduse komponendid on üksteisest sõltuvad ning selle tulemusena on koodi osad omavahel tihedalt seotud. Sellist arhitektuuri on lihtne teostada ning väiksem komponentide arv muudab rakenduse kergemini testitavaks. Negatiivse poole pealt muudab sellise arhitektuuri kasutamine rakenduse haldamise tulevikus raskemaks. Ühes rakenduse osas muudatuse tegemine mõjutab suure tõenäosusega ka teisi rakenduse osasid [12, lk 260-262]. Monoliitne arhitektuur sai valitud põhjusel, et autor ei ole varasemalt React-i raamistikuga kokku puutunud.

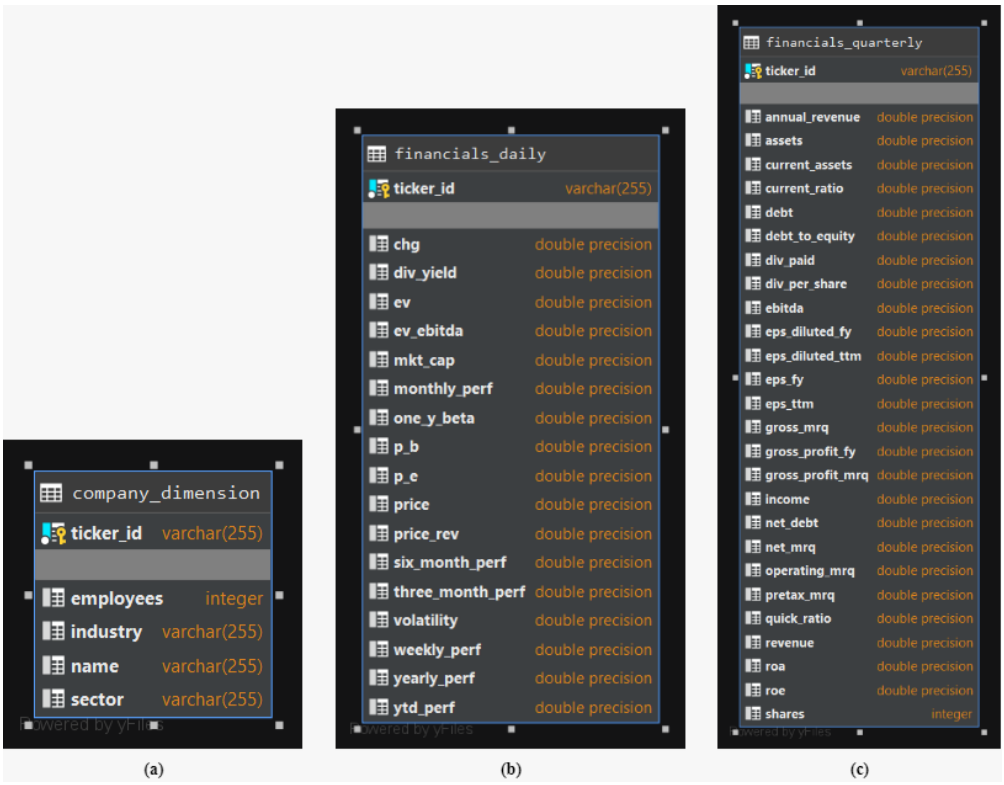



Joonis 7. Projekti kasutajaliidese arhitektuur.

Kaustas *src\components* on erinevate komponentide klassid, mis tegelevad projekti tagarakendusest andmete küsimisega, vajadusel andmete töötlemisega ning samuti kliendile kasutajaliidese kuvamisega. Mõni komponent on tema keerukuse tõttu jaotatud väiksemateks komponentideks. Kaustades *src\componentStyles* ja *src\image* on klassid ning pildifailid kasutajaliidese disaini parandamiseks. Kaustas *src\regex* olevad klassid tegelevad kasutaja poolt tekstiväljadesse lisatava teksti valideerimisega. Kaustades *src__mocks__* ning *src\tests* on erinevad testimisega seotud klassid.

### 3.2.3 Andmebaasi arhitektuur

Andmebaas on jaotatud kolmeks tabeliks: *company_dimension*, *financials_daily* ning *financials_quarterly* (Joonis 8). Peamiselt hoitakse tabelites rakenduses kajastatavate ettevõtete finantsandmeid. Tabelis *company_dimension* on ettevõtte üldised andmed, mis ei tohiks ajas muutuda, näiteks nagu ettevõtte nimi. Tabelis *financials_daily* on andmed, mis muutuvad igapäevaselt, näiteks nagu aktsia hind. Tabelis *financials_quarterly* on andmed, mis muutuvad kvartaalselt, nagu näiteks ettevõtte kasum. Andmebaas on jaotatud kolmeks väiksemaks tabeliks sellel põhjusel, et andmete suure hulga tõttu on neid kergem hallata väiksemate tabelite kaupa. Lisaks tuleb mõningaid andmeid tihedamini uuendada.




Joonis 8. Projekti andmetabelid: (a) *company_dimension*, (b) *financials_daily*, (c) *financials_quarterly*.

Projekti tagarakenduses on tabelite vahel ühendus loodud läbi *@PrimaryKeyJoinColumn* annotatsiooni. Ühendus on loodud selliselt, et kui klient teeb päringu ettevõtte üldistele andmetele (*company_dimension*), siis tagastatakse ka ettevõtte kohta tema igapäevased ja kvartaalsed andmed. Rakenduses eeldatakse, et tagastama peab korraga kõik ettevõtte andmed, seega eraldi tabelitele päringut teha ei ole võimalik ning kogu andmete küsimine käib läbi *company_dimension* tabeli.

Andmed loetakse andmebaasi kolmest erinevast CSV failist, ehk igale tabelile vastab üks CSV fail. Andmed CSV failidest loetakse andmetabelitesse kasutades Spring Batch raamistikku. Protsessi kirjeldatakse täpsemalt peatükis „Tagarakenduse kood“.

## 3.3 Disain

Projekti tagarakendus ning kasutajaliides kasutavad teineteisest erinevaid programmeerimiskeeli ja raamistikke. Sellest tulenevalt erinevad ka nendes kasutatavad disainimustrid. Neid mustreid kirjeldatakse järgnevates peatükkides.

### 3.3.1 Tagarakenduse disain

Projekti tagarakendus on disainitud töötama REST (*Representational State Transfer*) API-na (*Application Program Interface*), mis vastab kasutajaliidese rakenduses tehtud päringutele. REST API arendamisel on lähtutud teatud reeglitest, mida tuleks sellise rakenduse puhul järgida.

URI (*Uniform Resource Identifier*) tee puhul on järgitud, et igal tee segmendil oleks tähenduslik nimi. Nii on selge, mida päring peab tagastama. Kuna päringud tagastavad erinevate ettevõtetega seotud andmeid, ehk andmekogu, siis päringu tee on mitmuse vormis [13, lk 16-17]. Võimalikud päringud on välja toodud peatükis „Tagarakenduse kood“.

REST API puhul kasutatakse erinevaid HTTP (*Hypertext Transfer Protocol*) päringumeetodeid ning staatuskoode. Käesoleva rakenduse puhul kasutatakse ainult GET meetodit, mille ülesandeks on tagastada ettevõtte või ettevõtetega seotud andmed. GET meetod ei tohi täita mitte ühtegi teist ülesannet. Eduka päringu puhul tagastatakse staatuskood 200 ning vigase päringu puhul staatuskood 404 [13, lk 23-24, 28, 31].

Lisaks kasutatakse projektis veel *Dependency Injection* disainimustrit, mis paneb ühe objekti sõltuma teisest objektist [14]. Antud rakenduse puhul kasutatakse klassidevahelist *dependency injection*-it. Spring raamistikuga loodud projektides kasutatakse sõltuvuse tekitamist läbi *@Autowired* annotatsiooni.

### 3.3.2 Kasutajaliidese disain

Projekti kasutajaliidese osas on kasutatud React raamistikuga seotud disainimustreid. Järgnevalt tuuakse välja mõned levinumad disainimustrid, mida antud projektis kasutatud on.

Üks React raamistiku võtmetunnuseid on komponentide loomine. Komponendid peavad suutma koos edukalt töötada, sõltumata sellest, kas arendajaid on üks või mitu. Komponentidesse peab saama funktsioone lisada nii, et sellel oleks võimalikult väike mõju ülejäänud koodile [15].

Kasutajaliidese puhul tuleb arvestada, et see peab täitma kasutaja poolt antud käsklusi. Selle jaoks on kasutuses *Handling Events* tehnika [16]. Iga kord, kui kasutaja vajutab mõnele nupule, sisestab teksti ja nii edasi, kutsutakse rakenduses välja mõni meetod, mis sellega tegelema peab ning kasutajale soovitud tulemuse tagastama.

Keerulisema ülesehitusega komponentide puhul tekkis olukordi, kus mitu komponenti peavad samaaegselt kajastama ühesuguseid tulemusi. *Lifting State Up* tehnika seisneb selles, et kui erinevad komponendid peavad kasutama sama objekti olekut, siis tuleb seda objekti hoida hierarhiliselt kõrgemas komponendis. Kui mõni hierarhiliselt madalam komponent muudab objekti olekut, siis saadetakse see tagasi kõrgemasse komponenti, mis jagab selle oleku ka teistele komponentidele [17].

## 3.4 Kood ja testimine

Koodi kirjutamisel on töö autor andnud endast parima, et järgida *clean code* põhimõtteid. Näiteks on koodi kirjutamisel kasutatud põhimõtet, et väärtustel, klassidel ja meetoditel oleksid tähenduslikud nimed [18, lk 18]. Funktsioonid on võimalikult lühikesed, et ka teised arendajad saaksid vajadusel koodist aru [18, lk 34]. Kood peab olema loetav ning korrektselt formaaditud [18, lk 76]. Kood on kaetud testidega ning iga *unit* test testib ainult ühte meetodit või meetodi osa korraga [18, lk 124, 131].

Koodis ei ole duplikaate ning aegajalt refaktoriseeritakse koodi, et parandada selle disaini ning loetavust [18, lk 172-173]. Need on mõned näited põhimõtetest, mida on proovitud järgida.

Koodi kvaliteedi paremaks muutmist toetab IntelliJ-sse lisatud SonarLint *plugin*. See tuvastab koodi kirjutamise ajal erinevaid vigu ning annab soovitusi vea parandamiseks. Lisaks aitab *plugin* vältida koodi kirjutamisel halbu tavasid [19].

### 3.4.1 Tagarakenduse kood

Nagu eelnevalt mainitud, on projekti tagarakendus kirjutatud Java programmeerimiskeeles, projekt on loodud kasutades Spring Boot raamistikku ning toimib REST API-na, mis tagastab kasutajale päringu peale ettevõttega seotud andmeid (Lisa 2 – Näide ettevõtte andmetest JSON objektina).

Järgnevas tabelis (Tabel 1) on võimalikud andmepäringud:

Tabel 1. Võimalikud andmepäringud

| Päringu meetod | Päring | Päringu selgitus |
|---|---|---|
| GET | /companies | Tagastab kõik andmebaasis olevad ettevõtted |
| GET | /companies/id | Tagastab ettevõtte tema id järgi, nt /companies/LHV1T |

Projekti tagarakenduse üks tähtsamaid komponente on andmete lugemine CSV failidest ning sealt saadava informatsiooni kirjutamine vastavatesse andmebaasi tabelitesse ning tabeliveergudesse. CSV failid on kasutusel sellel põhjusel, et demoversiooni arenduses testiti erinevaid tabelite ülesehitusi ning CSV failidega töötamine ja nendes muudatuste sisse viimine oli mugavam võrreldes andmetabelite pideva ümberseadistamisega.

Eelmises lõigus kirjeldatud eesmärgi saavutamiseks on kasutatud Spring Batch raamistikku. Spring Batch on kerge ning laiaulatuslik pakkraamistik, mis on mõeldud arendamaks robustseid pakkrakendusi, mida läheb vaja ettevõtete süsteemide igapäevaseks toimimiseks. Spring Batch pakub taaskasutatavaid funktsioone, mis on vajalikud suure mahuga kirjete töötlemiseks [20].

Tüüpilise pakkprogrammi kasutusjuhud on tavaliselt järgmised:

- Suure koguse andmete lugemine andmebaasist, failist või järjekorrast

- Andmete töötlemine mingil moel
- Andmete kirjutamine andmebaasi, faili või järjekorda [21]

Iga CSV faili jaoks on loodud eraldi samm, mis loeb vastavast failist andmed, loob nende jaoks vastava andmetabeli ning kirjutab need andmed tabelisse. Kogu protsessi eest vastab üks suur pakktööde, mis tehakse läbi iga rakenduse käivitamise korral (Lisa 3 – Tagarakenduse pakktööde).

Iga kord kui pakktööde käivitatakse, kirjutatakse andmed tabelites uuesti üle. Juhul, kui andmetabelid on mingil põhjusel ära kustutatud, luuakse automaatselt uued tabelid. See muudab toimingu täielikult automatiseerituks ning arendaja poolt on vaja lihtsalt andmebaas üles seada, kuhu tabelid saaks luua.

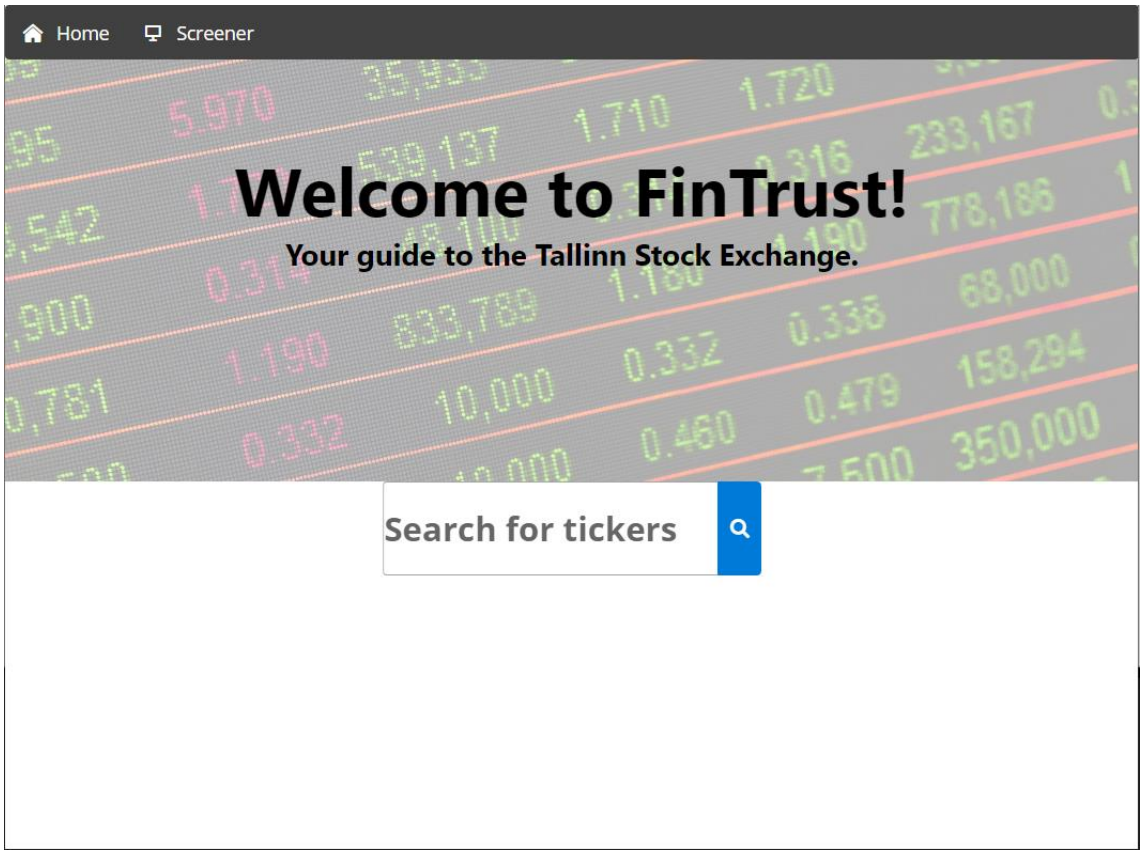
Lisaks on seadistatud projektile CORS (*Cross-Origin Resource Sharing*) (Lisa 4 – Tagarakenduse CORS seadistus), mis lubab rakendusele teha andmepäringuid teistest domeenidest [22]. See on vajalik, et rakenduse kasutajaliidese projekt saaks teha andmepäringuid.

### 3.4.2 Kasutajaliidese kood

Nagu eelnevalt mainitud, on koodi kasutajaliides kirjutatud JavaScript programmeerimiskeeles, kasutades React raamistikku. Lisaks on kasutajaliidese disainimisel kasutatud PrimeReact komponentide kogu ning CSS märgistuskeelt.

Rakenduse kasutajaliides on jaotatud kolmeks suuremaks vaateks: *Home*, *Screener* ja *CompanyInfo*. Lisaks on iga vaate küljes *Menubar*, mille abil saab erinevate vaadete vahel navigeerida.

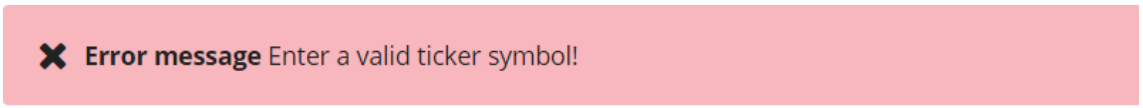
*Home* vaate puhul on tegemist esimese vaatega, mida kasutaja näeb, kui rakenduse avab (Joonis 9). Rakenduse funktsionaalsuse seisukohast ei ole tegemist rakenduse kõige tähtsama vaatega. Sellel põhjusel on vaatele arenduse mõttes pigem vähem tähelepanu pööratud. *Home* vaate URI tee on /. Rakenduses vastab vaatele *HomeComponent*.




Joonis 9. Rakenduse *Home* vaade.

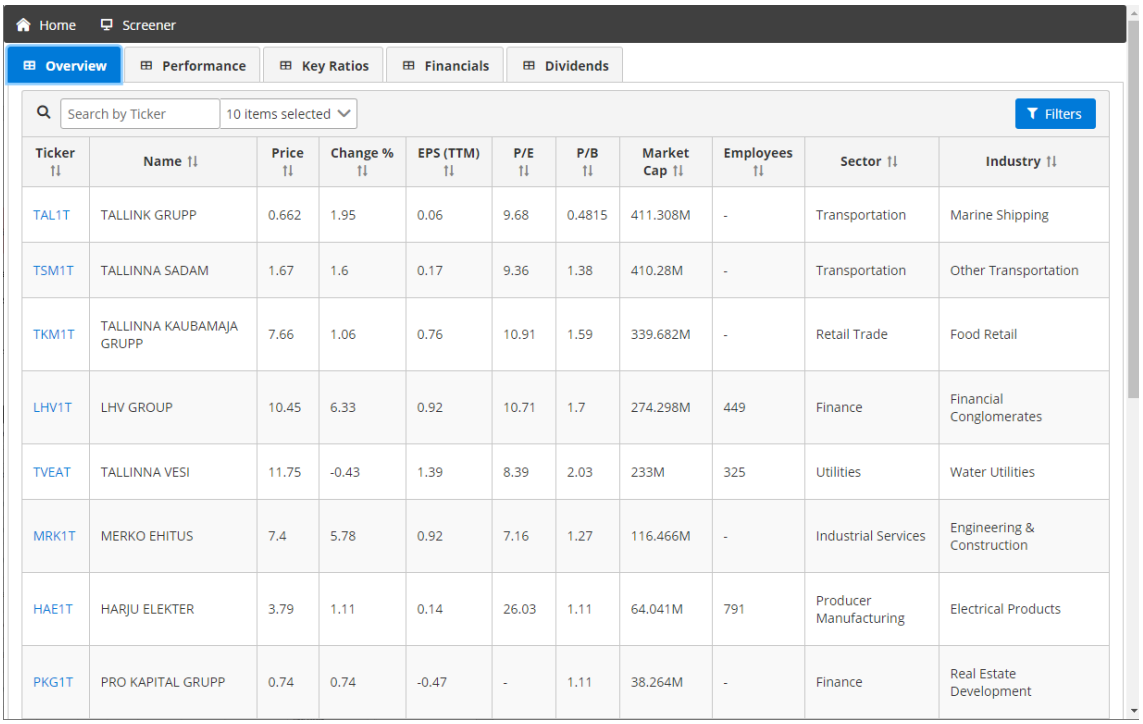
Kui vaade loetakse DOM (*Document Object Model*) liidesesse sisse, kutsutakse koheselt välja *componentDidMount* meetod. Selle meetodi sees on sobiv teha andmete päringuid tagarakendusest [23]. Andmete pärimine serverist toimub läbi axios teegi, mis muudab päringu automaatselt JSON (*JavaScript Object Notation*) objektiks [24]. *Home* vaate puhul tehakse tagarakendusse */companies* päring ning sealt võetakse *tickers* objekti sisse kõikide ettevõtete id-d (Lisa 5 – Andmepäring tagarakendusse axios teegi abil).

Vaate peamiseks funktsionaalsuseks on anda kasutajale võimalus koheselt mõnda ettevõtet otsida selle aktsiasümboli järgi. Otsingukastina kasutatakse PrimeReact *AutoComplete* komponenti, mis annab reaalajas soovitusi sümboli otsimisel [25]. Eelnevalt *tickers* objekti loetud andmeid filtreeritakse pidevalt vastavalt kasutaja poolt sisestatule ning seeläbi soovitab otsingumootor, mida otsida. Sisestades otsingumootorisse korrektse sümboli ning seejärel otsingunuppu vajutades, avaneb uues aknas *CompanyInfo* vaade. Vale sümboli sisestades kuvab rakendus kasutajale veateate (Joonis 10).




Joonis 10. Vale sümboli sisestamisel kuvatav veateade.

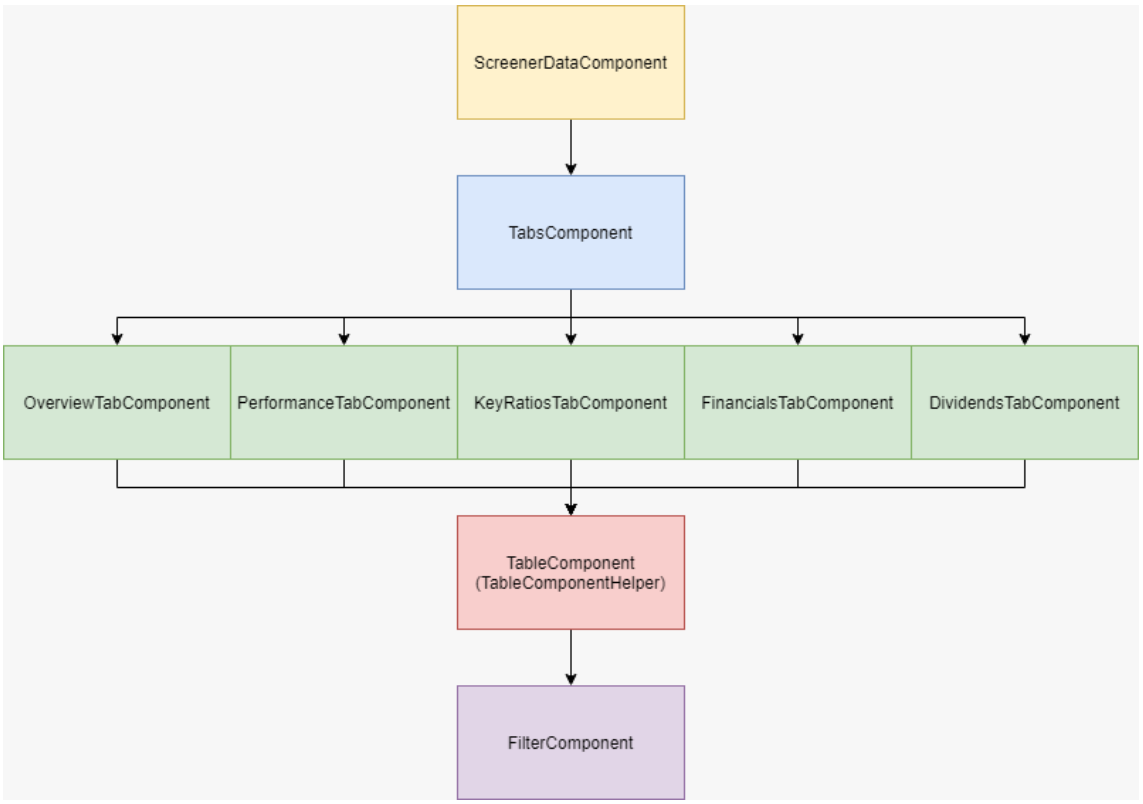
*Screener* vaade on praeguse rakenduse versiooni kõige olulisem vaade (Joonis 11). Lisaks on antud vaatel kõige keerulisem implementatsioon. Vaate peamiseks ülesandeks on anda kasutajale tabeli kujul ülevaade Tallinna börsi ettevõtetest. *Screener* vaate URI tee on */screener*.




| Ticker | Name | Price | Change % | EPS (TTM) | P/E | P/B | Market Cap | Employees | Sector | Industry |
|---|---|---|---|---|---|---|---|---|---|---|
| TAL1T | TALLINK GRUPP | 0.662 | 1.95 | 0.06 | 9.68 | 0.4815 | 411.308M | - | Transportation | Marine Shipping |
| TSM1T | TALLINNA SADAM | 1.67 | 1.6 | 0.17 | 9.36 | 1.38 | 410.28M | - | Transportation | Other Transportation |
| TKM1T | TALLINNA KAUBAMAJA GRUPP | 7.66 | 1.06 | 0.76 | 10.91 | 1.59 | 339.682M | - | Retail Trade | Food Retail |
| LHV1T | LHV GROUP | 10.45 | 6.33 | 0.92 | 10.71 | 1.7 | 274.298M | 449 | Finance | Financial Conglomerates |
| TVEAT | TALLINNA VESI | 11.75 | -0.43 | 1.39 | 8.39 | 2.03 | 233M | 325 | Utilities | Water Utilities |
| MRK1T | MERKO EHITUS | 7.4 | 5.78 | 0.92 | 7.16 | 1.27 | 116.466M | - | Industrial Services | Engineering & Construction |
| HAE1T | HARJU ELEKTER | 3.79 | 1.11 | 0.14 | 26.03 | 1.11 | 64.041M | 791 | Producer Manufacturing | Electrical Products |
| PKG1T | PRO KAPITAL GRUPP | 0.74 | 0.74 | -0.47 | - | 1.11 | 38.264M | - | Finance | Real Estate Development |

Joonis 11. Rakenduse *Screener* vaade.

*Screener* vaade on jaotatud kümneks väiksemaks komponendiks. Komponentide hierarhia järgi kõrgeimast madalamani on need jaotatud järgmiselt (Joonis 12):




Joonis 12. *Screener* vaate komponentide hierarhia.

*ScreenerDataComponent* on *Screener* vaate kõrgeim komponent ning sarnaselt *Home* vaatele tehakse tagarakendusele andmepäring, et saada kõikide ettevõtete finantsandmed, mida tabelis kasutajale kuvada. Lisaks töötleb see komponent mõningaid protsendilisi andmeid, mis tuleb 100-ga läbi korrutada, et need õiges formaadis oleks. Töödeldud andmed saadetakse madalamasse komponenti kasutades *props* parameetrit.

*TabsComponent* jaotab *Screener* vaate viieks erinevaks tabeliks, kasutades PrimeReact *TabView* komponenti [26]. Komponendi eeliseks on, et selle abil saab eri *tab*-ide vahel navigeerida lehekülge uuesti laadimata, mis muudab *tab*-ide vahel objektide oleku jagamise probleemivabaks. Samuti ei kaota objekti olek *tab*-e vahetades oma väärtust. Näiteks, kui filtreerida tabelis olevaid ettevõtteid, siis ühe *tab*-i alt filtreerides mõjub sama filter ka teistele *tab*-idele.

Komponendi üheks suuremaks eesmärgiks ongi vaate haldamine selliselt, et kõik kasutaja poolt ühele tabelile tehtavad muudatused, mis puudutavad filtreerimist ja tabelist otsimist, mõjuksid ka teistele tabelitele.

Komponent saadab kõik vajalikud objektid ja funktsioonid madalamatesse komponentidesse kasutades *props* parameetrit.

*OverviewTabComponent*, *PerformanceTabComponent*, *KeyRatiosTabComponent*, *FinancialsTabComponent* ja *DividendsTabComponent* on hierarhiliselt samal tasemel asetsevad komponendid. Nende komponentide kaudu määratakse ära kõik tabelite veerud, välja arvatud aktsiasümboli (*Ticker*) veerg, mis on ühine ning määratakse hierarhiliselt madalamas *TableComponent* komponendis.

Kasutades PrimeReact *DataTable* komponendi omadusi [27], on igale veerule lisatud sorteerimine ning eemaldatud võimalus globaalse otsingu tegemiseks, ehk tabelist vastava veeru järgi otsimine. Globaalne otsing käib ainult läbi aktsiasümboli (*Ticker)* veeru.

Komponendid saadavad kõik vajalikud objektid ja funktsioonid madalamasse komponenti kasutades *props* parameetrit.

*TableComponent* ja *TableComponentHelper* komponendid vastutavad peamiselt selle eest, et kõik andmed oleksid korrektsel kujul tabelisse paigutatud.

*TableComponent* põhineb peamiselt PrimeReact *DataTable* komponendil. Nagu eelnevalt kirjeldatud, siis igal tabelil on erinevad veerud ning selleks, et ei peaks igale tabelile eraldi komponenti looma, on kasutatud PrimeReact *DataTable – Dynamic Columns* funktsionaalsust [28], mille jaoks on loodud konstant *dynamicColumns*. Lisaks tuleb siin abiks *TableComponentHelper* klass, milles olev meetod *setCellBody* aitab iga tabeli lahtri sisu vajadusel õigesse formaati panna. Näiteks mõnel objekti väljal võib olla väärtuseks *null*, kuid tabelis oleks vaja tühja lahtri asemel see märgistada sidekriipsuga (-).

Tabelites on võimalik peita erinevaid tabeliveerge. Selleks on kasutatud PrimeReact *DataTable – Column Toggler* funktsionaalsust [29]. Funktsionaalsus on loodud komponendis oleva meetodi *onColumnToggle* ning konstandi *toggleColumns* abil.

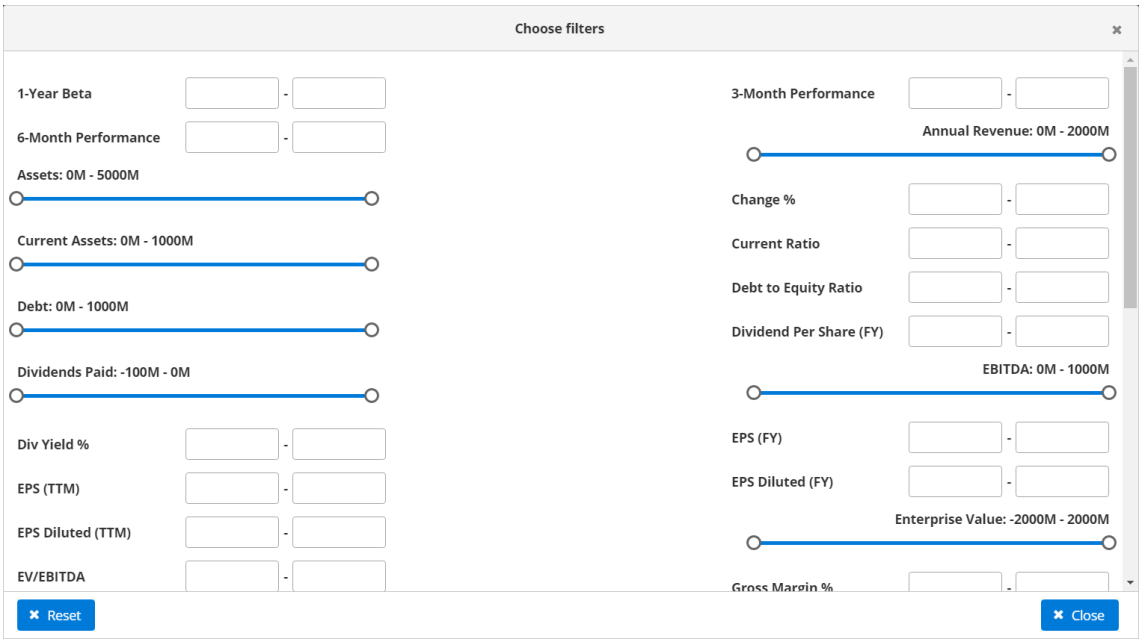
Tabelist on võimalik ettevõtteid otsida ettevõtte aktsiasümboli (*Ticker)* veeru järgi. Aktsiasümboli järgi otsimisel on kasutatud PrimeReact *DataTable – Filter* funktsionaalsust [30]. Funktsionaalsus on loodud komponendis oleva meetodi *onTickerSearch* ning konstandi *searchByTicker* abil. Antud funktsionaalsus nõuab

rakenduse arendamisel esimest korda *Lifting State Up* tehnikat, ehk oleku kõrgema hierarhiaga komponendile saatmist. Antud juhul kasutatakse seda selleks, et tabeli *tab*-ide vahel navigeerides oleks otsingulahtris pidevalt sama väärtus ning tabelites kuvatavate ettevõtete arv oleks sünkroonis.

Igal alamtabelil on ühine aktsiasümboli (*Ticker)* veerg ning vastava veeru lahtrites olevad väärtused toimivad lingina. Sellele vajutades avaneb uues brauseri aknas vastava ettevõtte *CompanyInfo* vaade. See toimub läbi *openCompanyInfoPage* meetodi.

Komponent saadab kõik vajalikud objektid ja funktsioonid madalamasse komponenti kasutades *props* parameetrit.

*FilterComponent* komponent on hierarhiliselt *Screener* komponentidest kõige madalamal. Selle põhjaks on võetud PrimeReact *Dialog* komponent [31]. Komponendi eesmärgiks on anda kasutajale võimalus ettevõtteid tema finantsnäitajate järgi tabelist välja filtreerida (Joonis 13).



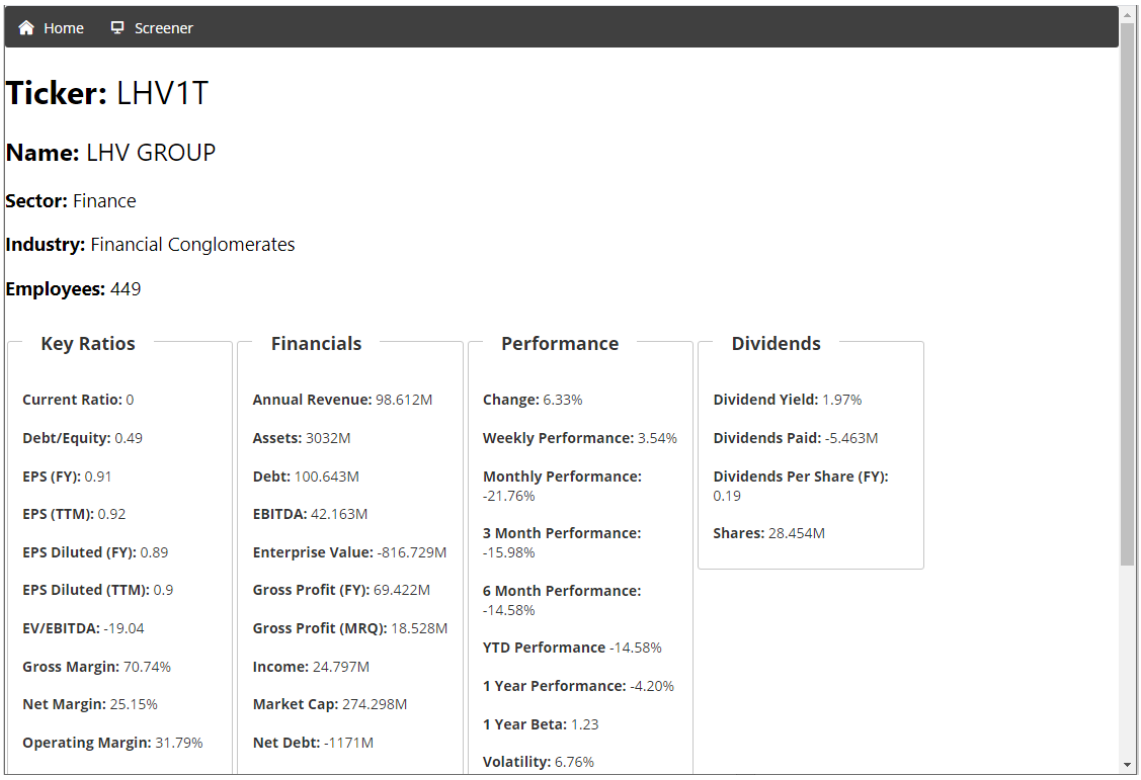

Joonis 13. Rakenduse hüpikvaade *Screener* vaate tabelis olevate ettevõtete filtreerimiseks.

Kasutaja saab väärtusi sisestada erinevatesse tekstiväljadesse või kasutades liugureid, mis on mõeldud suuremate väärtustega finantsnäitajate jaoks. Tekstiväljade jaoks on kasutatud PrimeReact *InputText* komponenti [32] ning liugurite jaoks PrimeReact *Slider* komponenti [33]. Tekstiväljades lubatakse sisestada ainult numbrilisi väärtusi ning seda valideeritakse *regex*-i abil [34]. Liuguritele on seatud alam- ning ülempiirang, mille järgi kasutaja saab oma valiku teha.

Filtreerimine toimib peamiselt meetodite *positiveValuesFilter*, *positiveAndNegativeValeusFilter*, *sliderValuesFilter* ning *filterResults* abil. Lisaks on igale filtreeritavale lahtrile/finantsnäitajale eraldi abimeetod, mis kasutab vastavalt vajadusele ühte kolmest selles lõigus esimesena mainitud meetodist. Meetodi *filterResults* kaudu tagastatakse vastavalt kasutaja valikutele ning meetodisisesele filtreerimistulemusele õiged ettevõtted.

Antud komponendis kasutatakse *Lifting State Up* tehnikat, et filtreerimine erinevate alamtabelite vahel oleks sünkroonis ning *tab*-ide vahel navigeerides oleks tabelis kuvatavate ettevõtete arv sama. Seda tehnikat kasutavad peamiselt meetodid *handleInputChange*, *handleSliderChange* ning *resetFilterInputs*. Esimesed kaks tegelevad tekstiväljades ning liugurites väärtuse muutmisega ning seeläbi filtreerimismeetodite välja kutsumisega. Meetod *resetFilterInputs* tühistab kõik kasutaja poolt sisestatud filtrid ning tabel kuvab uuesti kõiki ettevõtteid.

*CompanyInfo* vaate eesmärgiks on kasutajale kompaktselt kuvada kõik ettevõttega seonduv informatsioon (Joonis 14). Rakenduses vastab vaatele *CompanyInfoComponent*.



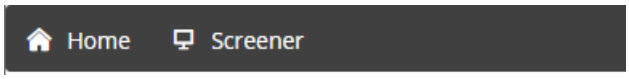

Joonis 14. Rakenduse *CompanyInfo* vaade.

Vaatesse on võimalik navigeerida läbi *Home* vaate otsingumootori või valides *Screener* vaate tabeli aktsiasümboli (*Ticker)* veerust soovitud ettevõte.

Komponendi URI tee on */screener/id* ning kasutades *id* parameetrit tehakse päring tagarakendusse, mis tagastab komponendile vastava ettevõtte finantsandmed. Näiteks kui URI tee on */screener/LHV1T*, siis rakendus teeb päringu aadressil */companies/LHV1T*.

*CompanyInfoComponent* komponent kasutab PrimeReact *Fieldset* grupeerimiskomponenti [35], mille abil jaotatakse ettevõtte finantsandmed erinevateks gruppideks ning kuvatakse kompaktselt kasutajale.

*MenubarComponent* komponent toimib menüüribana ning on mõeldud navigeerimiseks *Home* ja *Screener* vaate vahel (Joonis 15). Komponent on iga rakenduses oleva vaate ülemises osas.




Joonis 15. Rakenduse menüüriba.

Menüüribana kasutatakse PrimeReact *Menubar* komponenti [36].

### 3.4.3 Testid

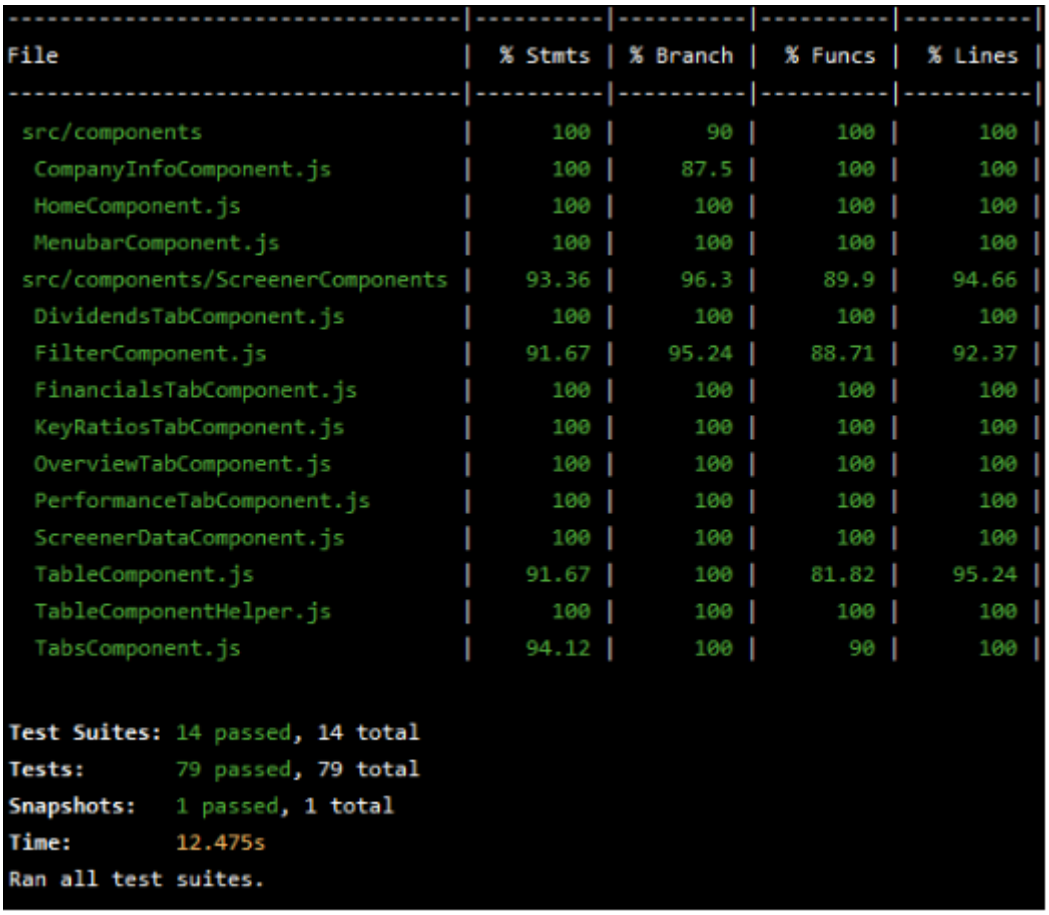
Projekti tagarakenduses on testitud rakenduse ainsat *controller* klassi ning CSV failide lugemiseks ja andmebaasi kirjutamiseks mõeldud pakktöödet.

*CompanyDimensionControllerTest* testklassil on kasutatud *@SpringBootTest* annotatsiooni ning selles on kolm *unit* testi. Testitud on, et rakendus tagastaks tulemuse päringule */companies* ning samuti testitakse, et rakendus tagastaks tulemuse päringule *company/id*. Lisaks testitakse, et vigane *companies/id* päring tagastaks *null* väärtuse.

*CompanyDimensionSpringBatchConfigTest* testklassil on kasutatud *@SpringBootTest* ja *@SpringBatchTest* annotatsiooni. Klassis on üks *companyObjectJobTest* testmeetod, mille puhul on tegemist *end-to-end* testiga, ehk tehakse läbi kogu pakktööde tervikuna. Esmalt tühjendab test kõik *repository*-d, seejärel kontrollib, et *repository*-d oleks tühjad. Järgmisena käivitatakse pakktööde ning kontrollitakse, et selle läbi viimine oli edukas. Viimaks kontrollitakse, et *repository*-sse lisati pakktööte tulemusena õige arv objekte.

Projekti kasutajaliideses on *unit* testidega kaetud kõik komponendid ja meetodid. Testimisel kasutatakse JavaScripti jaoks mõeldud Jest testimisraamistikku, mis töötab edukalt React projektidega [37]. Samuti on kasutuses React-ile mõeldud Enzyme testimisutiliit [38].

Teste on 79 ning autor on andnud endast parima, et testidega kaetud meetodite hulk oleks võimalikult suur (Joonis 16). Kõik meetodid on testitud *unit* testimise tasandil.



| File | % Stmts | % Branch | % Funcs | % Lines |
|---|---|---|---|---|
| src/components | 100 | 90 | 100 | 100 |
| CompanyInfoComponent.js | 100 | 87.5 | 100 | 100 |
| HomeComponent.js | 100 | 100 | 100 | 100 |
| MenubarComponent.js | 100 | 100 | 100 | 100 |
| src/components/ScreenerComponents | 93.36 | 96.3 | 89.9 | 94.66 |
| DividendsTabComponent.js | 100 | 100 | 100 | 100 |
| FilterComponent.js | 91.67 | 95.24 | 88.71 | 92.37 |
| FinancialsTabComponent.js | 100 | 100 | 100 | 100 |
| KeyRatiosTabComponent.js | 100 | 100 | 100 | 100 |
| OverviewTabComponent.js | 100 | 100 | 100 | 100 |
| PerformanceTabComponent.js | 100 | 100 | 100 | 100 |
| ScreenerDataComponent.js | 100 | 100 | 100 | 100 |
| TableComponent.js | 91.67 | 100 | 81.82 | 95.24 |
| TableComponentHelper.js | 100 | 100 | 100 | 100 |
| TabsComponent.js | 94.12 | 100 | 90 | 100 |

```
Test Suites: 14 passed, 14 total
Tests:       79 passed, 79 total
Snapshots:   1 passed, 1 total
Time:        12.475s
Ran all test suites.
```

Joonis 16. Komponentide testidega kaetuse aruanne.

# 4 Analüüs ja järeldused

Antud peatükis analüüsitakse projektis tehtud olulisemaid valikuid ning tehakse nende kohta vastavad järeldused. Peamiselt põhjendatakse projektis kasutatavate tehnoloogiate valikut, selgitatakse projektile seatud nõuetele vastavust ning analüüsitakse rakenduse arhitektuuri ja disaini. Lisaks antakse lühike ülevaade tehtud tööst ja võimalikest edasiarenduse ideedest.

## 4.1 Kasutatud tehnoloogiate valik

Tehnoloogiate valikul sai suures osas lähtutud varasematest kasutuskogemustest ning juhendaja soovitustest. Järgnevalt selgitatakse nende tehnoloogiate valikut, millel oli suurem mõju projekti arendamisel.

Projekti tagarakenduse puhul sai määravaks varasem arenduskogemus, mis antud töö autori puhul tähendas Java programmeerimiskeelt ning Spring Boot raamistikku või C# programmeerimiskeelt ning ASP.NET raamistikku. Mõlemad programmeerimiskeeled ning raamistikud on olnud kasutuses mitmetes ülikooli jooksul läbitud õppeainetes. Samuti on Java ning C# programmeerimiskeeled ühed populaarsemad professionaalsete arendajate hulgas. Veebilehe Stack Overflow poolt 2019. aastal läbi viidud arendajatele suunatud küsitluses, populaarseimate programmeerimiskeelte kategoorias, kasutas Java ning C# programmeerimiskeelt vastavalt 39.2% ja 31.9% küsitletutest [39]. Määravaks sai see, et viimase aasta jooksul on oskusi arendatud rohkem Java programmeerimiskeeles ning isiklik huvi kasutatud raamistiku suhtes on suurem.

Projekti kasutajaliidese puhul puudus autoril isiklik eelistus, millist programmeerimiskeelt või raamistikku kasutada, sest varasemalt puudus tõsisem kokkupuude kasutajaliidese arendamisega. Juhendaja soovitus oli kasutada React või Vue raamistikku, mille puhul kasutatakse peamiselt JavaScript programmeerimiskeelt. Eelnevalt mainitud Stack Overflow küsitluse järgi, veebiraamistike kategoorias, kasutas React või Vue raamistikku vastavalt 32.3% ning 15.5% professionaalsetest arendajatest [39]. Lisaks võib välja tuua, et küsitluse järgi oli React raamistik professionaalsete

arendajate hulgas kõrgel kolmandal kohal ning kõikide vastanute hulgas koguni teisel kohal. Sellega seoses saigi valitud just React raamistik.

Seoses React raamistiku kasutamisega oli vaja leida lahendus, kuidas kasutajaliidese arendamine kiiremaks muuta. Seda toetas fakt, et autoril puudus varasem kogemus React raamistikuga töötamisel, mis tähendas, et arendusprotsess võib võtta kauem aega. Seega oli vaja otsida mõni komponentide kogu, mida rakenduses kasutada. Internetis leidub väga palju erinevaid komponentide kogusid ning valiku tegemist mõjutas peamiselt asjakohane dokumentatsioon, komponentide kasutatavuse lihtsus ning spetsiaalse foorumi olemasolu. Sellest tulenevalt jäi valikusse React Bootstrap, Material-UI ning PrimeReact komponentide kogu, millest valituks osutus just viimane.

PrimeReact pakub laia komponentide valikut ning dokumentatsioon on väga põhjalik. Iga komponendi kohta on toodud konkreetsed kasutusnäited ning selgitused. PrimeReact-il on olemas ka vastav foorum, kust võib leida vastuseid levinud probleemidele. Lisaks tekitas autoris usaldust asjaolu, et sama veebileht pakub arvestatavale hulgale kasutajatele sarnaseid lahendusi ka raamistikudes nagu JavaServer Faces, Angular, Vue ning jQuery.

## 4.2 Töö nõuetele vastavus

Peatükis „Nõuded rakendusele“ määrati rakendusele peamised nõuded, mida peaks suutma täita, et rakendus oleks kasutatav. Järgnevalt selgitatakse, kui suures osas õnnestus nõudeid täita.

### 4.2.1 Funktsionaalsetele nõuetele vastavus

Üks esmaseid nõudeid oli, et rakendus peab ettevõtete tabelis kuvama kõiki Tallinna börsil kaubeldavaid aktsiaid. Suures osas on see nõudmine täidetud. Rakendus kuvab andmeid kõigi kuueteistkümne Tallinna börsi põhinimekirjas kaubeldavate ettevõtete aktsiate kohta. Välja on jäänud üksikud lisanimekirjas või alternatiivturul First North kaubeldavad aktsiad.

Iga ettevõtte kohta oli seatud nõue, et investoril peab olema võimalik saada täpsemat informatsiooni, sealhulgas ajaloolisi andmeid, et otsustada, kas tegemist võiks olla sobiva investeeringuga. Tegemist on ühe olulisema nõudega, mis jäi täitmata. Investoril

on võimalik eraldi vaates näha korraga kõiki üksiku ettevõttega seotud andmeid, kuid puudub võimalus näha ajaloolisi andmeid, mis on investeerimisotsuse tegemise seisukohast oluline puudujääk. Selle nõude täitmine vajab suuremahulist andmete kogumist, milleks praeguse töö raames aega ei jätkunud ning millega tuleks tegeleda tulevikus.

Tabelis kuvatavatele ettevõtetele oli seatud nõue, et neid on võimalik filtreerida nende finantsnäitajate järgi. Kasutajale on loodud võimalus filtreerida ettevõtteid, kasutades erinevaid tekstivälju ning liugureid, kuhu sisestada piirväärtused. Tabelis kuvatakse vaid neid ettevõtteid, mis kriteeriumitele vastavad.

Ettevõtete kohta oli seatud nõue, et neid oleks võimalik võrrelda. Antud nõue on täidetud küllaltki minimaalses ulatuses. Tabel kuvab kasutajale kõiki ettevõtteid ning erinevaid veerge on võimalik sorteerida, et tekitada võrdlusmoment. See lahendus ei ole väga täpne. Võrdlusmoment on küll olemas, kuid oleks vaja luua lahendus, kus on näiteks võimalik kõrvutada kaks või enam üksikut ettevõtet nende finantsnäitajate järgi.

Ettevõtete tabelitele oli seatud nõue, et kasutajal on võimalus valida, milliseid finantsandmeid talle parasjagu kuvatakse. Nõue on täidetud mõne üksiku puudujäägiga. Kõik andmed on kategooriate järgi jaotatud viite erinevasse tabelisse: *Overview*, *Performance*, *Key Ratios*, *Financials* ning *Dividends*. Kõiki tabelites olevaid veerge, peale ettevõtte aktsiasümboli (*Ticker*), on võimalik peita, mis tähendab, et soovi korral saab tabelid kompaktsemaks muuta ning eemaldada tabelist mittevajalikke finantsandmeid kuvavad veerud. Hiljem on need võimalik jälle tagasi lisada. Puudujäägina võib välja tuua selle, et erinevatesse tabelitesse ei saa lisada veerge teistest tabelitest, et anda kasutajale võimalus tabel täielikult enda nägemise järgi üles ehitada.

Ettevõtete otsimise jaoks oli seatud nõue, et neid oleks võimalik kiiresti eraldi üles otsida, kui on koheselt teada, millise ettevõtte kohta andmeid soovitakse näha. Selleks on loodud kaks erinevat võimalust. Esimene ning kõige kiirem variant on läbi rakenduse *Home* vaate sisestada otsingumootorisse soovitud ettevõtte aktsiasümbol, mis suunab kasutaja koheselt *CompanyInfo* vaatesse, milles on kõik ettevõttega seotud andmed. Teine võimalus sama tulemuse saavutamiseks on kasutada *Screener* vaates olevat ettevõtte aktsiasümboli otsingulahtrit.

Rakenduse kasutajal võib tekkida enne ettevõttesse investeerimist soov mõnda aega selle käekäiku kõrvalt jälgida. Selleks peaks ta saama luua endale eraldi nimekirja nendest ettevõtetest, millel ta silma peal hoiab. Selle nõude täitmiseni töö jooksul ei jõutud. Tegemist ei ole rakenduse seisukohast kõige tähtsama nõudega, kuid pikas perspektiivis oleks selle täitmine kasulik.

### 4.2.2 Mittefunktsionaalsetele nõuetele vastavus

Rakendusele on seatud nõue, et see töötaks Google Chrome ning Mozilla Firefox brauserite värskeimas versioonis. Veebilehe Statcounter andmete kohaselt on Google Chrome ning Mozilla Firefox kaks kolmest kõige populaarsemast *desktop* brauserist viimase kaheteist kuu jooksul [40]. Töö kirjutamise hetkel on mainitud brauserite viimased versioonid vastavalt 81.0.4044.138 ning 76.0.1. Rakendus töötab edukalt mõlemas brauseris.

Rakendus peab vastama andmepäringutele kiiremini, kui kolm sekundit. Visuaalselt on näha, et andmed jõuavad kasutajani kiiremini, kui kolm sekundit, kuid täpne keskmine mõõtmistulemus puudub. Seega tuleks tulevikus adekvaatsema tulemuse saavutamiseks luua erinevad andmepäringu kiirustestid.

Rakenduse arendamisel on seatud nõue, et rakenduse laienemisel peab funktsionaalsuste lisamine olema endiselt lihtne. Töö autor on andnud endast parima, et järgida erinevaid *clean code* põhimõtteid, mis toetaksid sellise nõude täitmist.

Selleks, et rakendus oleks reaalselt kasutatav, peaksid selles olevad andmed muutuma vähemalt korra ööpäeva jooksul. Praegune demoversioon kasutab *mock* andmeid ning seega ei ole võimalik nõuet täita.

Lisaks oli tabelitele seatud nõue, et nendes kuvatavad andmed peavad olema loogiliselt jaotatud, et muuta otsitavate andmete leidmine kergemaks. Nagu eelnevalt mainitud, on erinevaid tabeleid viis. *Overview* tabel kuvab andmeid ettevõtte üldisemate näitajate kohta, näiteks ettevõtte nimi, viimane kauplemishind ja nii edasi. Mõned selles tabelis kuvatavad andmed on nähtavad ka teistes tabelites. *Performance* tabel kuvab andmeid ettevõtte aktsiahinna muutuste ning riski kohta. *Key Ratios* tabel kuvab andmeid ettevõtte suhtarvude kohta, nagu näiteks P/E, P/B ja nii edasi. *Financials* tabel kuvab

andmeid, mis puudutavad ettevõtte väärtust, vara ja kasumlikkust. *Dividends* tabelis on andmed ettevõtte dividendidega seonduva kohta.

Rakenduse kasutatavuse seisukohalt on oluline, et selle kasutamine oleks lihtne ning sellest tulenevalt ei tohi olla rakenduses olevad funktsioonid keerulised. Töö autor on andnud endast parima, et olemasolevad funktsionaalsused oleksid kergelt kasutatavad. Rakenduse edasistes arendusetappides tuleks kindlasti kasutajaliidese kasutatavust valideerida.

## 4.3 Arhitektuuri analüüs

Projektiga alustades tuli läbi mõelda, millist arhitektuuri arendamisel kasutada. Üsna kiirelt sai selgeks, et mõistlik oleks luua kaks eraldi projekti. See tähendas, et rakendus jaotati tagarakenduseks ning kasutajaliideseks.

Selline eraldamine muudab projektid nõrgalt seotuks, lubades nende eraldi haldamist ning testimist. Väiksemad projektid on paremini skaleeritavad ning suure tõenäosusega tekib vähem erinevaid vigu. Tänu sellele, et projekti tagarakendus toimib eraldi REST API-na, saab seda taaskasutada, näiteks kui luua ka mobiilirakenduse võimalus.

Alternatiivina oleks võinud kogu rakenduse arendada ühe suurema projektifaili raames. See oleks tekitanud ühe suure monoliitse rakenduse. Sellisel juhul puuduks vajadus kahe projekti vahel CORS mehhanismi kasutada ning üleüldine juurutus oleks tõenäoliselt kergem. Siiski, pikemas perspektiivis ning rakenduse kasvades tundus kahe eraldi projekti loomine mõistlikum.

## 4.4 Disaini analüüs

Projekti tagarakenduse disaini osas on autoril keeruline välja tuua erinevaid aspekte, mida oleks võinud arendamisel teisiti teha. Võimalik, et põhjuseks on REST API küllaltki lihtne üles ehitus. Praeguses rakenduse versioonis on vaid kaks erinevat andmepäringut, millele vastata tuleb. Erinevad disainiprobleemid võivad tekkida, kui rakendus peab hakkama vastama suuremale kogusele erinevatele päringutele. Lisaks on arendamisel võimalikult suures osas lähtutud reeglitest, mis puudutavad REST API loomist. Need reeglid ja põhimõtted seletati pikemalt lahti peatükis „Tagarakenduse disain“.

Projekti kasutajaliidese poolel leidus nüansse, millega tulevikus arvestada ning millest õppida. Disainivead tekkisid peamiselt eelneva arenduskogemuse puudumisest React raamistikuga.

Töös on mitmeid komponente, mida saaks veel väiksemaks muuta, et nende sisu oleks loetavam. Üks võimalus on jaotada komponent oma olemuselt kaheks – üks pool komponendist tegeleks ainult meetodite loogikaga, ehk vastutaks selle eest, kuidas komponent peaks käituma mõne meetodi välja kutsumisel. Teine pool komponendist vastutaks komponendi väljanägemise eest, ehk mida kasutajale kuvatakse [41]. Kirjeldatud mustrit on töös kasutatud, kuid kindlasti oleks saanud seda veel rohkem teha.

Üks muster, mida töö jooksul kasutati, kannab nime *Prop Drilling*. Sellega puututakse kokku siis, kui on vaja *props*-e hierarhiliselt kõrgemast komponendist madalamasse saata. Tegemist on React-i puhul täiesti tavalise praktikaga ning selle kasutamist toetab asjaolu, et nii on kerge jälgida, kus mõni objekt või meetod, mida *prop*-ina madalamale saadetakse, deklareeritud on. Probleemid tekivad, kui rakendus muutub väga suureks ning mõni *prop* on vaja läbi mitme komponendi madalamale saata. See on tüütu ning koodi seisukohast ei näe alati kõige puhtam välja [43].

Esimene võimalus selle vältimiseks on leida hierarhiliselt madalaim võimalik komponent, kust soovitud *prop*-i veel madalamale saata [43]. Sellist lähenemisviisi on kasutatud antud töös.

Teine võimalus on kasutada React Context API-t, mis seda kõike vältida aitab [42][43]. Tehtud töös tekkis olukordi, kus mõni *prop* oli vaja hierarhiliselt väga madalasse komponenti saata. Seega on React Context API kindlasti võimalus, mida tulevikus sarnaste olukordade lahendamiseks kasutada.

## 4.5 Rakendus ja edasiarendused

Töö tulemusena valmis esmane demoversioon finantsrakendusest, mis annab kasutajale ülevaate Tallinna börsil tegutsevatest ettevõtetest. Rakendus koosneb kolmest erinevast vaatest.

Rakendust käivitades avaneb *Home* vaade, milles oleva otsingumootori kaudu on võimalik aktsiasümboli järgi üksikut ettevõtet otsida. Rakenduse peamine vaade on *Screener* vaade, mis kuvab kasutajatele andmeid ettevõtete kohta. Andmed on jaotatud *tab*-ide abil viieks tabeliks. Tabelis olevaid andmeveerge on võimalik peita ning sorteerida. Tabeli kohal olevat otsingumootorit on võimalik kasutada filtreerimiseks ettevõtte aktsiasümboli järgi. Vaates oleva *Filters* nupu abil avaneb uus aken, mida kasutades saab tabelis olevaid ettevõtteid filtreerida erinevate finantsnäitajate järgi. Iga tabeli aktsiasümboli (*Ticker*) veeru lahtris olev väärtus toimib lingina, mis avab vaate vastava ettevõtte andmete kohta. Üksiku ettevõtte andmeid kajastab *CompanyInfo* vaade, kus on kuvatud kõik vastava ettevõtte kohta olemasolevad andmed.

Rakenduse praeguse versiooni suurimaks puuduseks on *mock* andmete kasutamine. Reaalne rakendus nõuaks andmeid nii ettevõtte praeguse olukorra, kui ka ajalooliste andmete kohta. See tähendaks suurt andmete kogumist ja töötlemist, mida praeguse töö mahu juures oleks olnud keeruline teostada. Kindlasti tuleks mainitud tegevuse jaoks kaasata ka uusi meeskonnaliikmeid.

Juhul, kui rakendus kasutaks reaalandmeid, tuleks mõelda, kuidas neid andmebaasides säilitada ning ajas juurde lisada. Kindlasti tekiks praeguse kolme tabeli asemel juurde väga palju uusi tabeleid, seda just ajalooliste andmete kogumise tõttu. See tekitab juurde erinevaid andmebaasi arhitektuuri ning disainiga seotud küsimusi.

Rakendusele tuleks luua võimalus kasutajakonto tegemiseks. See lubaks muuta rakenduse kasutamise personaalsemaks ning annaks võimaluse kontopõhise jälgimisnimekirja funktsionaalsuse arendamiseks. Konto loomine tõstataks ka erinevad turvalisusega seotud küsimused, mille lahendamisega tegeleda.

Investeerimisotsuse tegemise lihtsamaks muutmiseks tuleks rakendusele luua eraldi vaade ettevõtete võrdlemiseks. Selle kaudu saaks kõrvutada kaks või enam ettevõtet ning muuta valiku tegemine lihtsamaks, kui peaks näiteks tekkima soov investeerida vaid ühte võrreldavatest ettevõtetest.

Rakenduse praegune kasutajaliides on inglise keeles. Seda põhjusel, et börsidel investeerimine on rahvusvaheliste mastaapidega ning teoorias võiksid rakendust kasutada ka välismaalased. Tulevikus on plaanis lisada ka eestikeelne kasutajaliides.

Rakendus võiks lisaks ettevõtete finantsandmetele kuvada ka aktsiagraafikuid, mis näitaksid ajaloolist aktsiahinna liikumist. See on üsna tavaline funktsionaalsus, mida pakuvad ilmselt kõik sarnased finantsrakendused. Selle funktsionaalsuse lisamine nõuaks jällegi reaalandmeid, mis hetkel puuduvad.

Kaugemas tulevikus võib mõelda ka selle peale, et lisada rakendusse veel erinevaid börsil kaubeldavaid ettevõtteid. Näiteks liikuda selle suunas, et rakendus kataks ära kogu Balti börsi. See oleks pigem üks viimaseid arendusetappe, sest esmalt peaks rakendus näitama korrektseid reaalandmeid praeguste ettevõtete kohta ning üleüldiselt ilma vigadeta töötama.

# 5 Kokkuvõte

Lõputöö raames alustati finantsveebirakenduse arendamisega, mis kajastaks Tallinna börsiettevõtete finantsandmeid. Täna puudub Tallinna börsi jaoks rakendus, mis töötaks *screener*-ina ning samas pakuks ka informatsiooni ettevõtte ajalooliste tulemuste kohta. Pikas perspektiivis on eesmärgiks luua toimiv rakendus, mis oleks piisavalt usaldusväärne, et selle abil investeerimisotsuseid vastu võtta.

Töö autor alustas rakenduse arendamist seoses isikliku huviga investeerimise vastu. Rakenduse arendamisel on saadud inspiratsiooni erinevate sarnaste rakenduste kasutamisest ning analüüsimisest.

Töö on jaotatud kaheks väiksemaks projektiks. Projekti tagarakendus vastab andmepäringutele ning projekti kasutajaliides presenteerib tulemused kasutajale. Arendamise käigus on lähtutud kirjanduses ja internetist leitud arhitektuuri, disaini ning koodi kirjutamise headest tavadest ning reeglitest.

Töö tulemusena valmis esmane demoversioon rakendusest. Demoversiooni peamine funktsionaalsus on ettevõtete finantsandmete kuvamine tabelites. Seoses *mock* andmete kasutamisega ei ole rakendust kasutades võimalik veel investeerimisotsuseid vastu võtta.

Töö järgmisteks etappideks oleks kokku koguda ettevõtete finantsandmed ning lisada juurde erinevaid funktsionaalsusi kasutajakogemuse parandamiseks. Suure tõenäosusega võtab rakenduse arendamine veel palju aega ning vajab juurde ka teisi arendajaid. Seoses huviga teema vastu üldiselt on töö autoril plaanis projektiga tulevikus jätkata.

# Kasutatud kirjandus


[1] Nasdaq Balti börsid. [WWW] https://nasdaqbaltic.com/et/meist/nasdaq-balti-borsid/ (03.05.2020)

[2] TradingView Features. [WWW] https://www.tradingview.com/features/ (03.05.2020)

[3] Getting started with Yahoo Finance. [WWW] https://help.yahoo.com/kb/finance-for-web/started-yahoo-finance-sln3642.html (03.05.2020)

[4] IntelliJ IDEA overview. [WWW] https://www.jetbrains.com/help/idea/discover-intellij-idea.html (04.05.2020)

[5] A brief overview of Bitbucket. [WWW] https://bitbucket.org/product/guides/getting-started/overview#a-brief-overview-of-bitbucket (04.05.2020)

[6] Spring Boot. [WWW] https://spring.io/projects/spring-boot (04.05.2020)

[7] Tutorial: Intro to React. [WWW] https://reactjs.org/tutorial/tutorial.html (04.05.2020)

[8] Why PrimeReact? [WWW] https://primefaces.org/primereact/showcase/#/ (04.05.2020)

[9] What is PostgreSQL? [WWW] https://www.postgresql.org/about/ (04.05.2020)

[10] Rubin, K. S. Essential Scrum: A Practical Guide to the Most Popular Agile Process. Upper Saddler River, NJ: Addison-Wesley Professional, 2012. (04.05.2020)

[11] Fowler, M. PresentationDomainDataLayering. 2015. [WWW] https://martinfowler.com/bliki/PresentationDomainDataLayering.html (05.05.2020)

[12] Ingeno, J. Software Architect's Handbook: Become a Successful Software Architect by Implementing Effective Architecture Concepts. Birmingham: Packt Publishing Ltd., 2018. (05.05.2020)

[13] Masse, M. REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. Beijing: O'Reilly Media, Inc., 2011. (05.05.2020)

[14] Schuchert, B. L. DIP in the Wild. 2013. [WWW] https://martinfowler.com/articles/dipInTheWild.html

[15] Design Principles. [WWW] https://reactjs.org/docs/design-principles.html (05.05.2020)

[16] Handling Events. [WWW] https://reactjs.org/docs/handling-events.html (05.05.2020)

[17] Lifting State Up. [WWW] https://reactjs.org/docs/lifting-state-up.html (05.05.2020)

[18] Martin, R. C. Clean Code: A Handbook of Agile Software Craftsmanship. Upper Saddle River, NJ: Prentice Hall, 2008. (06.05.2020)

[19] SonarLint Features. [WWW] https://www.sonarlint.org/features/ (06.05.2020)

[20] Spring Batch Introduction. [WWW] https://docs.spring.io/spring-batch/docs/current/reference/html/spring-batch-intro.html#spring-batch-intro (06.05.2020)

[21] Spring Batch Usage Scenarios. [WWW] https://docs.spring.io/spring-batch/docs/current/reference/html/spring-batch-intro.html#springBatchUsageScenarios (06.05.2020)

[22] Cross-Origin Resource Sharing (CORS). [WWW] https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (06.05.2020)

[23] React.Component. [WWW] https://reactjs.org/docs/react-component.html#componentdidmount (06.05.2020)

[24] Axios. [WWW] https://github.com/axios/axios (06.05.2020)

[25] PrimeReact AutoComplete. [WWW] https://www.primefaces.org/primereact/showcase/#/autocomplete (06.05.2020)

[26] PrimeReact TabView. [WWW] https://www.primefaces.org/primereact/showcase/#/tabview (06.05.2020)

[27] PrimeReact DataTable. [WWW] https://www.primefaces.org/primereact/showcase/#/datatable (06.05.2020)

[28] PrimeReact DataTable – Dynamic Columns. [WWW] https://www.primefaces.org/primereact/showcase/#/datatable/dynamiccolumns (06.05.2020)

[29] PrimeReact DataTable – Column Toggler. [WWW] https://www.primefaces.org/primereact/showcase/#/datatable/coltoggle (06.05.2020)

[30] PrimeReact DataTable – Filter. [WWW] https://www.primefaces.org/primereact/showcase/#/datatable/filter (06.05.2020)

[31] PrimeReact Dialog. [WWW] https://primefaces.org/primereact/showcase/#/dialog (06.05.2020)

[32] PrimeReact InputText. [WWW] https://primefaces.org/primereact/showcase/#/inputtext (06.05.2020)

[33] PrimeReact Slider. [WWW] https://primefaces.org/primereact/showcase/#/slider (06.05.2020)

[34] Regular expressions. [WWW] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions (06.05.2020)

[35] PrimeReact Fieldset. [WWW] https://www.primefaces.org/primereact/showcase/#/fieldset (06.05.2020)

[36] PrimeReact Menubar. [WWW] https://www.primefaces.org/primereact/showcase/#/menubar (06.05.2020)

[37] Jest. [WWW] https://jestjs.io/ (07.05.2020)

[38] Enzyme. [WWW] https://enzymejs.github.io/enzyme/ (07.05.2020)

[39] Stack Overflow Developer Survey Results 2019. [WWW] https://insights.stackoverflow.com/survey/2019#most-popular-technologies (09.05.2020)

[40] Desktop Browser Market Share Worldwide. [WWW] https://gs.statcounter.com/browser-market-share/desktop/worldwide (09.05.2020)

[41] Abramov, D. Presentational and Container Components. 2015. [WWW] https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0 (10.05.2020)

[42] Context. [WWW] https://reactjs.org/docs/context.html (10.05.2020)

[43] Dodds, K. C. Prop Drilling. 2018. [WWW] https://kentcdodds.com/blog/prop-drilling (10.05.2020)

# Lisa 1 – GitHub-i lingid

Projekti tagarakendus: https://github.com/MarkoJagor/StockAppBackend

Projekti kasutajaliides: https://github.com/MarkoJagor/StockAppFrontend

# Lisa 2 – Näide ettevõtte finantsandmetest JSON objektina


```
{
    "ticker_id": "LHV1T",
    "name": "LHV GROUP",
    "employees": 449,
    "sector": "Finance",
    "industry": "Financial Conglomerates",
    "financialsDaily": {
        "ticker_id": "LHV1T",
        "price": 10.45,
        "div_yield": 0.0197,
        "mkt_cap": 274298.0,
        "p_e": 10.71,
        "price_rev": 2.84,
        "p_b": 1.7,
        "ev_ebitda": -19.04,
        "ev": -816729.0,
        "chg": 0.0633,
        "weekly_perf": 0.0354,
        "monthly_perf": -0.2176,
        "three_month_perf": -0.1598,
        "six_month_perf": -0.1458,
        "ytd_perf": -0.1458,
        "yearly_perf": -0.042,
        "one_year_beta": 1.23,
        "volatility": 0.0676
    },
    "financialsQuarterly": {
        "ticker_id": "LHV1T",
        "current_ratio": 0.0,
        "debt_to_equity": 0.49,
        "net_debt": -1171000.0,
        "quick_ratio": null,
        "assets": 3032000.0,
        "debt": 100643.0,
        "current_assets": null,
        "eps_fy": 0.91,
        "eps_ttm": 0.92,
        "eps_diluted_ttm": 0.9,
        "ebitda": 42163.0,
        "gross_profit_mrq": 18528.0,
        "gross_profit_fy": 69422.0,
        "revenue": 98612.0,
        "eps_diluted_fy": 0.89,
        "annual_revenue": 98612.0,
```

```
        "income": 24797.0,
        "gross_mrq": 0.7074,
        "operating_mrq": 0.3179,
        "pretax_mrq": 0.3179,
        "net_mrq": 0.2515,
        "div_paid": -5463.0,
        "div_per_share": 0.19,
        "roa": 0.0105,
        "roe": 0.1399,
        "shares": 28454
    }
}
```

# Lisa 3 – Tagarakenduse pakktööde

```
@Configuration
@EnableBatchProcessing
public class CompanyDimensionSpringBatchConfig {

    @Bean
    public Job companyObjectJob(JobBuilderFactory jobBuilderFactory,
                StepBuilderFactory stepBuilderFactory,
                ItemReader<CompanyDimension> companyDimensionItemReader,
                ItemWriter<CompanyDimension> companyDimensionItemWriter,
                ItemReader<FinancialsDaily> financialsDailyItemReader,
                ItemWriter<FinancialsDaily> financialsDailyItemWriter,
                ItemReader<FinancialsQuarterly> financialsQuarterlyItemReader,
                ItemWriter<FinancialsQuarterly> financialsQuarterlyItemWriter
    ) {

        Step companyDimensionStep = stepBuilderFactory.get("Company
Dimension")
                .<CompanyDimension, CompanyDimension>chunk(100)
                .reader(companyDimensionItemReader)
                .writer(companyDimensionItemWriter)
                .build();

        Step financialsDailyStep = stepBuilderFactory.get("Financials Daily")
                .<FinancialsDaily, FinancialsDaily>chunk(100)
                .reader(financialsDailyItemReader)
                .writer(financialsDailyItemWriter)
                .build();

        Step financialsQuarterlyStep = stepBuilderFactory.get("Financials
Quarterly")
                .<FinancialsQuarterly, FinancialsQuarterly>chunk(100)
                .reader(financialsQuarterlyItemReader)
                .writer(financialsQuarterlyItemWriter)
                .build();

        return jobBuilderFactory.get("ETL-Load")
                .incrementer(new RunIdIncrementer())
                .start(companyDimensionStep)
                .next(financialsDailyStep)
                .next(financialsQuarterlyStep)
                .build();
    }
```

```
    @Bean
    public FlatFileItemReader<CompanyDimension> companyDimensionItemReader()
{

        return createItemReader(CompanyDimension.class,
"src/main/resources/data/CompanyDimension.csv");

    }

    @Bean
    public FlatFileItemReader<FinancialsDaily> financialsDailyItemReader() {

        return createItemReader(FinancialsDaily.class,
"src/main/resources/data/FinancialsDaily.csv");
    }

    @Bean
    public FlatFileItemReader<FinancialsQuarterly>
financialsQuarterlyItemReader() {

        return createItemReader(FinancialsQuarterly.class,
"src/main/resources/data/FinancialsQuarterly.csv");
    }

    public <T> FlatFileItemReader<T> createItemReader(Class<T> type, String
filePath) {

        FlatFileItemReader<T> flatFileItemReader = new
FlatFileItemReader<>();
        flatFileItemReader.setResource(new FileSystemResource(filePath));
        flatFileItemReader.setName("CSV-Reader");
        flatFileItemReader.setLinesToSkip(1);
        flatFileItemReader.setLineMapper(createLineMapper(type));
        return flatFileItemReader;
    }

    public <T> LineMapper<T> createLineMapper(Class<T> type) {
        DefaultLineMapper<T> defaultLineMapper = new DefaultLineMapper<>();
        DelimitedLineTokenizer lineTokenizer = new DelimitedLineTokenizer();

        lineTokenizer.setDelimiter(";");
        lineTokenizer.setStrict(false);

        if (type == CompanyDimension.class) {
            lineTokenizer.setNames("ticker_id", "name", "employees",
"sector", "industry");
        } else if (type == FinancialsDaily.class) {
            lineTokenizer.setNames("ticker_id", "price", "div_yield",
"mkt_cap", "p_e", "price_rev", "p_b", "ev_ebitda", "ev", "chg",
"weekly_perf", "monthly_perf", "three_month_perf", "six_month_perf",
"ytd_perf", "yearly_perf", "one_y_beta", "volatility");
        } else if (type == FinancialsQuarterly.class) {
            lineTokenizer.setNames("ticker_id", "current_ratio",
"debt_to_equity", "net_debt", "quick_ratio", "assets", "debt",
"current_assets", "eps_fy", "eps_ttm", "eps_diluted_ttm", "ebitda",
"gross_profit_mrq", "gross_profit_fy", "revenue", "eps_diluted_fy",
"annual_revenue", "income", "gross_mrq","operating_mrq", "pretax_mrq",
```

```
"net_mrq", "div_paid", "div_per_share", "roa", "roe", "shares");
        }

        BeanWrapperFieldSetMapper<T> fieldSetMapper = new
BeanWrapperFieldSetMapper<>();
        fieldSetMapper.setTargetType(type);

        defaultLineMapper.setLineTokenizer(lineTokenizer);
        defaultLineMapper.setFieldSetMapper(fieldSetMapper);

        return defaultLineMapper;
    }
}

@Component
public class CompanyDimensionDBWriter implements ItemWriter<CompanyDimension>
{

    @Autowired
    private CompanyDimensionRepository companyDimensionRepository;

    @Override
    public void write(List<? extends CompanyDimension> companyDimensions) {
        companyDimensionRepository.saveAll(companyDimensions);
    }
}

@Component
public class FinancialsDailyDBWriter implements ItemWriter<FinancialsDaily> {

    @Autowired
    private FinancialsDailyRepository financialsDailyRepository;

    @Override
    public void write(List<? extends FinancialsDaily> financialsDailies) {
        financialsDailyRepository.saveAll(financialsDailies);
    }
}

@Component
public class FinancialsQuarterlyDBWriter implements
ItemWriter<FinancialsQuarterly> {

    @Autowired
    private FinancialsQuarterlyRepository financialsQuarterlyRepository;

    @Override
    public void write(List<? extends FinancialsQuarterly>
financialsQuarterlies) {
        financialsQuarterlyRepository.saveAll(financialsQuarterlies);
    }
}
```

# Lisa 4 – Tagarakenduse CORS seadistus

```
@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**").allowedOrigins("http://localhost:3000");
    }

    //Lühiduse huvides on üleliigne kood välja jäetud.
}
```

# Lisa 5 – Andmepäring tagarakendusse axios teegi abil

```
class HomeComponent extends React.Component {

    constructor() {
        super();
        this.state = {
            tickers: [],
            //Lühiduse huvides on üleliigne kood välja jäetud.
        }
    }

    componentDidMount() {
        //Lühiduse huvides on üleliigne kood välja jäetud.
        axios.get("http://localhost:8080/companies")
            .then(response => {
                this.setState({
                    tickers: response.data.map(obj => obj.ticker_id)
                })
            })
    }

    //Lühiduse huvides on üleliigne kood välja jäetud.
}
```