TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Katre-Helena Käppa 134104IABB

TALLINNA BÖRSI ETTEVÕTETE FINANTSANDMETE VEEBIRAKENDUS

Bakalaureusetöö

Juhendaja: Tõnn Talpsepp
PhD

Tallinn 2021

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: Katre-Helena Käppa

22.12.2020

Annotatsioon

Bakalaureusetöö eesmärgiks on luua finantsandmete veebirakendus, mis keskendub Tallinna börsi ettevõtetele ja mis aitaks kohalikel investoritel teha läbimõeldud investeerimisotsuseid. Antud töö on edasiarendus eksisteeriva demorakenduse esmaversioonist.

Tallinna börsil tegutseval investoril ei ole hetkel mugavat veebirakendust, mis võimaldaks kuvada kasutajale ettevõtte andmeid reaalajas ning pakuks lihtsat võimalust ajalooliste andmete analüüsiks. Olemasolevad rakendused ei kata neid vajadusi Tallinna börsi väiksuse tõttu.

Töö tulemusena valmis rakenduse demoversioon uute funktsionaalsustega. Olulisemaks uuenduseks on aktsiahindade perioodiline uuendamine ja mõne olulise finantssuhtarvu automaatne arvutamine ning kuvamine. Lisandus veebirakenduse kasutajakonto loomise võimalus, millega on võimalik kasutajal lisada aktsiasümboleid jälgimisnimekirja ning teha üks-ühele börsifirmade finantsandmete võrdlust.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 38 leheküljel, 5 peatükki, 23 joonist.

Abstract

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

The aim of the current thesis is to create a web application of the companies listed on the Tallinn Stock Exchange. The application data should be trusted enough to be used as a tool for making investment decisions. This thesis is a further development of an already existing demo application which was built by a former student for his thesis.

Investors who are active in Tallinn Stock Exchange do not have a comfortable solution for both viewing a company's current financial status and offering straightforward way to analyze the historical data. There are various solutions existing in the world, but due to the low volume and small amount of investors on Tallinn Stock Exchange, the data displayed there is incomplete.

As a result of the work completed for the current paper, a demo version with new functionality was created. One of the most important improvements is the shift to start using the real-life data instead of mock data. Stock prices are now updated regularly and calculations of some of the financial ratios are done automatically. In addition, the user can now create an account and add specific stocks to their watch list. Another relevant feature added is the opportunity to compare companies financial data side-by-side.

The thesis is in Estonian and contains 38 pages of text, 5 chapters, 23 figures.

Lühendite ja mõistete sõnastik

Aktsia *screener*	Instrument, mis võimaldab aktsiaid välja filtreerida valitud parameetrite järgi
AMQP	*Advanced Message Queuing Protocol*, sõnumijärjekorraprotokoll
API	*Application Program Interface*, rakendustarkvara liides
CSRF	*Cross-Site Request Forgery*, saidivaheliste päringute võltsimine
CSS	*Cascading Style Sheets*, veebilehe kujundusel kasutatav keel
CSV	*Comma Separated Values*, tekstifail, mis kasutab komasid andmete eraldamiseks
DDoS	*Distributed Denial-of-Service*, hajutatud teenusetõkestamise rünne
DOM	*Document Object Model*, dokumendiobjektide mudel
HTML	*Hypertext Markup Language*, veebilehtede märgenduseks kasutatav keel
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastamise protokoll
HTTPS	*HyperText Transfer Protocol Secure*, turvaline hüperteksti edastamise protokoll
JSON	*JavaScript Object Notation*, andmevahetusformaat, mis põhineb võtme-väärtus paaridel
JSX	*JavaScript XML*, JavaScripti laiendatav märgistuskeel
JWT	*JSON Web Token*, JSONi veebimärk
REST	*Representational State Transfer*, arhitektuuri stiil
Scraper	Tööriist info eraldamiseks lähtekoodist
SOAP	*Simple Object Access Protocol*, lihtne objektipöördusprotokoll
SQL	*Structured Query Language*, struktureeritud päringukeel
URI	*Uniform Resource Identifier*, ühtne ressursiindikaator
XML	*Extensible Markup Language*, laiendatav märgistuskeel
XSS	*Cross-Site Scripting*, ristskriptimine

Sisukord

Jooniste loetelu

1 Sissejuhatus

Käesolev bakalaureusetöö käsitleb Tallinna börsil tegutsevate ettevõtete finantsandmete veebirakenduse arendust. Antud veebirakenduse loomisega alustati teise tudengi lõputöö raames ning praegune töö keskendub selle eesmärgipärasele edasiarendusele.

Investor kasutab läbimõeldud investeerimisotsuste tegemiseks erinevaid allikaid. Reaalajas pakuvad olulisemate finantsnäitajate kohta infot erinevad *screener* tööriistad, kuid enamasti on need mõeldud suurtele välismaistele börsidele ja pole Eesti turule keskendunud investori jaoks piisavad. Pikemaajaliste investeeringute tegemiseks jääb ka sellest infost väheseks ning peab lisaks analüüsima ettevõtte majandusaasta aruandeid ning muid dokumente, et saada võimalikult head pilti ettevõtte plaanidest ning majanduslikust käekäigust.

Tallinna börsil tegutseval investoril ei ole hetkel mugavat veebirakendust, mis võimaldaks kuvada kasutajale aktsiate andmeid reaalajas ning samas pakuks lihtsat võimalust ettevõtte ajaloolise tervikpildi analüüsiks.

Eelnevalt kirjeldatud probleemi lahendamiseks on loodud veebirakenduse demoversioon fookusega Tallinna börsile. Käesoleva töö eesmärgiks on tuua rakendusse esimesi reaalandmeid ja täiendada rakendust oluliste funktsionaalsustega, mis eristaksid seda olemasolevatest toodetest ning toetaksid lõppeesmärgi täitmist. Lõppeesmärgiks on luua tööriist, mida investor saaks kasutada igapäevaste investeerimisotsuste tegemiseks. Veebirakendus kajastaks nii kaubeldavate ettevõtete reaalandmeid kui ka esitaks terviklikku, päevakohast ja usaldusväärset infot aktsiatega seotud ajalooliste finantsandmete kohta ilma, et kasutaja peaks hakkama seda infot otsima majandusaasta aruannetest.

Rakenduse eelmises versioonis loodi rakendus, mis kuvas kasutajale staatilisi andmeid Tallinna börsi põhinimekirjas kaubeldavatest aktsiatest. Andmed olid esitatud tabelitena, lisaks võimaldas rakendus üksikaktsiate otsingut ning nende info kuvamist detailvaates.

Praeguse töö tulemusena valmis demoversioon uute funktsionaalsustega. Olulisemaks uuenduseks on reaalandmete osaline kasutuselevõtt - aktsiahindade uuendamine rakenduses teatud intervalli tagant ja mõne olulise finantssuhtarvu automaatne arvutamine ning kuvamine. Lisandus kasutajakonto loomise võimalus, millega on võimalik kasutajal lisada jälgimisnimekirja aktsiasümboleid ning teha börsifirmade finantsandmete üks-ühele võrdlust.

Tagarakendus on üles ehitatud kasutades Java programmeerimiskeelt ning Spring Boot raamistikku. *Frontend* poole pealt kasutatakse JavaScripti React raamistikku. Kasutajaliidese elementide arenduseks on kasutatud PrimeReact-i komponentide kogu ja disainiküsimuste lahendamiseks CSS-i (*Cascading Style Sheets*) võimalusi. Andmete kraapimise ehk *scraper* rakendus on üles ehitatud kasutades Pythoni programmeerimiskeelt ning suhtlus *scraper*-i ning tagarakenduse vahel toimub läbi RabbitMQ sõnumivahendaja. Andmebaasi valikuna on kasutatud PostgreSQL-i. Projekti arenduses jälgitakse *clean code* põhimõtteid.

Töö järgnevates osades antakse ülevaade olemasolevatest lahendustest, selgitatakse kasutatud tehnoloogiaid ning kirjeldatakse tööprotsessi. Tuuakse välja defineeritud funktsionaalsed ja mittefunktsionaalsed nõuded ning töö tulemused. Samuti kirjeldatakse kasutatud arhitektuuri ning disaini, antakse ülevaade koodist ja testimisest. Lõpetuseks analüüsitakse tehtud tööd ning tehakse ettepanekuid edasiarenduseks.

2 Projekti ülevaade

Investeerimishuvi on tavainimeste seas hakanud viimastel aastatel populaarsust koguma. Inimesed on muutunud oma rahaasjades ja säästmisharjumustes tähelepanelikumaks, eesti keelse investeerimisteemalise kirjanduse hulk on suurenenud ning väikeste summadega investeerimisega alustamine on muutunud odavamaks ja kättesaadavamaks.

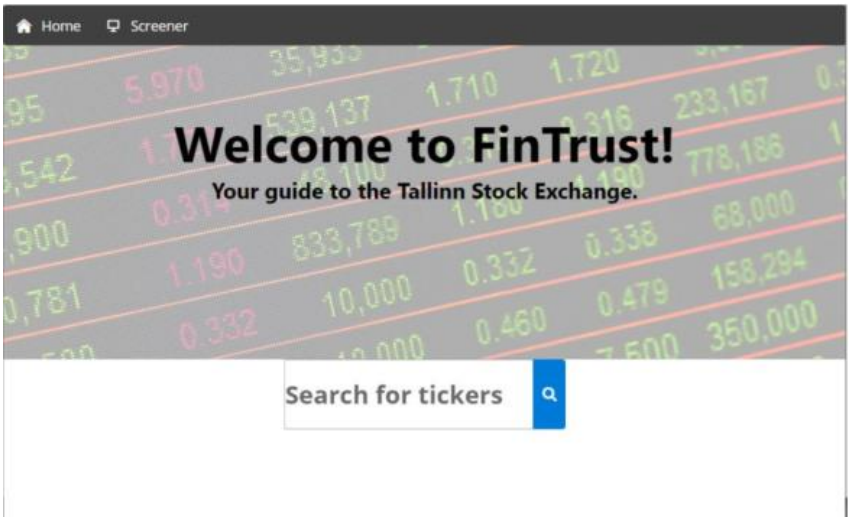
Töö autor tegutseb Tallinna börsil ning on omal käel teinud investeerimisotsuseid lähtudes erinevatest finantsnäitajatest. Autor on pidanud oma otsuste tegemiseks uurima ettevõtete majandusaasta aruandeid, saanud aru protsessi ebamugavusest ning hakanud otsuste lihtsustamiseks kasutama ka välisallikaid. Tallinna börsile suunatud veebirakendus, mis kajastaks nii hetkelist finantsseisu kui ka annaks lihtsa viisi tutvuda ka usaldusväärsete ajalooliste andmetega oleks hea täiendus juba olemasolevatele lahendustele.

2.1 Valitud eksisteerivate lahenduste ülevaade

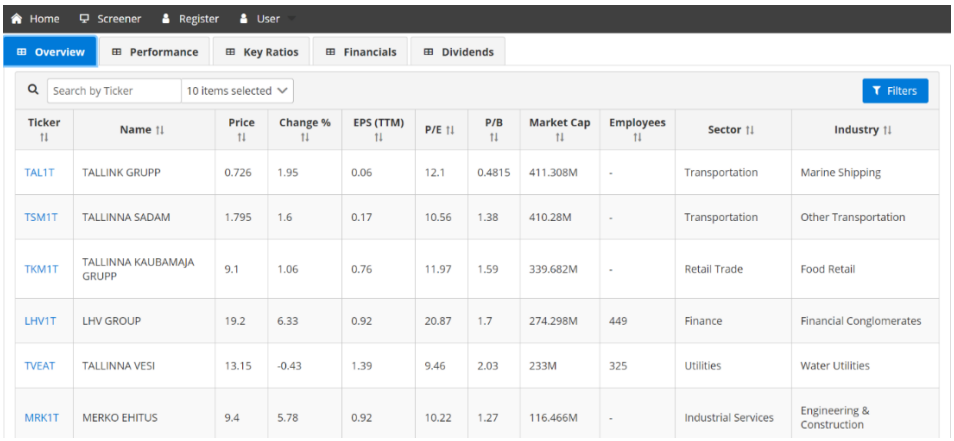
Järgnevalt kirjeldatakse lühidalt veebirakenduse esimest demoversiooni ning tuuakse välja ka mõned juba investorite poolt kasutusel olevad lahendused.

2.1.1 Käesoleva veebirakenduse esimene demoversioon

Esmane demoversioon loodi 2020.aasta kevadsemestril teise tudengi bakalaureusetöö raames [1]. Veebirakendus kasutas *mock*-andmestikku, mis laeti üles tagarakenduse andmebaasi CSV (*Comma Separated Values*) failide kujul. Tagarakendus edastas kasutajaliidesele REST (*Representational State Transfer*) API (*Application Program Interface*) kaudu infot ettevõtte finantsandmete kohta. Lõppkasutaja jaoks olulised funktsionaalsused kasutajaliidese rakenduses olid pealehel aktsiasümboli järgi otsingu teostamine, börsiettevõtetega seotud andmete kuvamine mitme vahelehega tabelivaates ning üksiku börsiettevõte detailvaade.



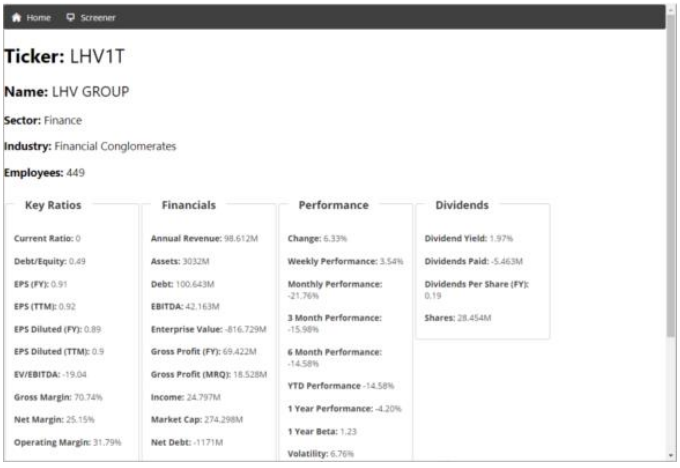
Joonis 1. Kuvatõmmis rakenduse esmaversiooni pealehest.



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. Tallina börsiettevõtete *mock*-andmestiku kuvamine tabelivaates.

Tabelivaates on *mock*-andmestik kategoriseeritud viieks teemaks, mille vahel on võimalik navigeerida üle vahelehtede. Tabeli päis võimaldab andmeid sobiva karakteristiku järgi sorteerida. „*Filters*“ nupp pakub võimalust ettevõtteid erinevate parameetrite järgi üldtabelist välja filtreerida.

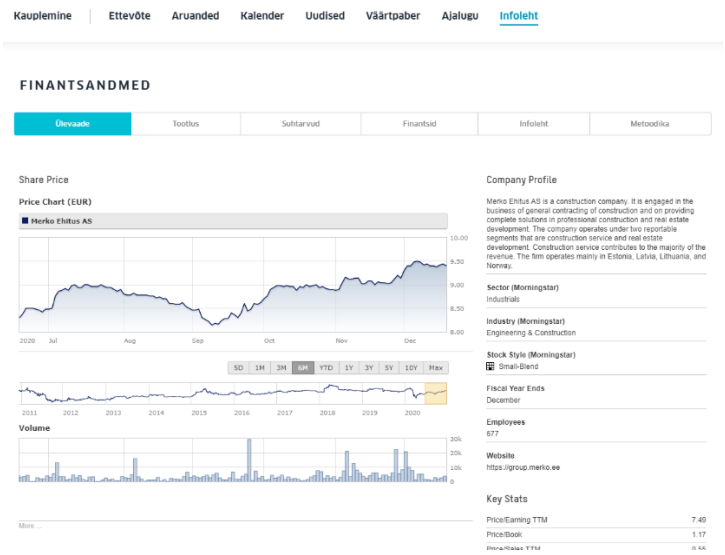


Joonis 3. Üksiku börsiettevõte detailvaade.

Esmane demoversioon oli vaheetapp pikemas arendusprotsessis, kust praeguse töö autor on jätkanud veebirakendusele oluliste funktsionaalsuste lisamist.

2.1.2 Nasdaq Balti börsi veebileht

Tallinna, Riia ja Vilniuse börsid on kokku koondatud Nasdaq Balti väärtpaberibörsi alla. Börs võimaldab kauplemist erinevate turuinstrumentidega – Baltikumis noteeritud aktsiate, võlakirjade ja fondidega [2]. Veebirakenduse tugevusena on investori jaoks tegemist usaldusväärse allikaga, kuna sealse info korrektsust kontrollib iga riigi kohalik finantsjärelvalveamet. Leheküljel on võimalik leida ettevõtte aktsia kohta hetkeinfot ja lisaks ka ajaloolisi andmeid aastate lõikes. Samuti on võimalik kasutajal lugeda börsiteateid ja luua jälgimisnimekirja.



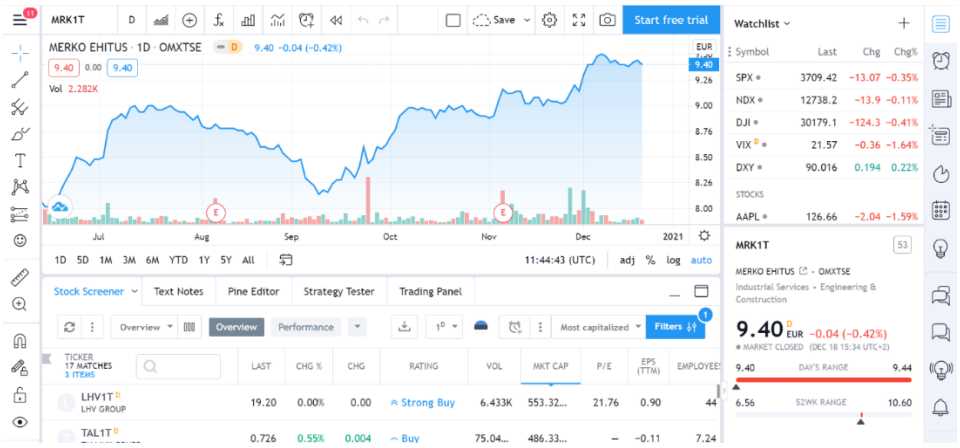
Joonis 4. Kuvatõmmis Merko Ehitus AS aktsia finantsandmete pealehest.

Lehekülje puudusteks on info rohkus, kuna börs teenindab peale investorite ka teisi osapooli milletõttu pole investorite kasutajamugavus esiplaanil. Ettevõtete ajaloolise info on leidmine on raskendatud – infot tuleb otsida uurides pikki majandusaasta aruandeid. Samuti pole võimalik luua eri ettevõtete detailsemate finantsandmete võrdlustabeleid.

2.1.3 TradingView rakendus

TradingView on tarkvara, mille abiga saab koostada ning analüüsida erinevaid finantsandmete graafikuid ja jälgimisnimekirju. Veebirakendus võimaldab luua tabeleid vastavalt kasutaja soovitud parameetritele ning välja filtreerida mittehuvitavad väärtused. Lisaks on võimalik seadistada teavitusi erinevatest sündmustest ja lugeda ettevõttega seotud uudiseid. Peale aktsiate on võimalus kuvada infot ka võlakirjade, indeksite,

futuuride, eri valuutade ning krüptoraha turgude kohta. TradingView katab üle 50 börsi maailmas, mille sees on ka Tallinna börs [3]. Tallinna börsil kaubeldavaid aktsiaid on võimalik leida otsinguga ning nende detailvaates näha aktsiahinna liikumist, erinevate investorite ideid aktsia suhtes ning olulisi finantssuhtarve.



Joonis 5. TradingView graafikute koostamise tööriist.

Lisaks sellele on rakenduses suhtlusvõrgustik nii algajatele kui kogenud investoritele, kus saab teha koostööd ja koguda investeerimisideid. Platvormil on oma skriptimiskeel Pine, millega on võimalik luua kohaldatud tabeleid ja indikaatoreid.

TradingView tugevaimaks küljeks on rakenduse paindlikkus luua analüüsi jaoks vajaminevaid andmestikke vastavalt kasutaja soovile. Lisaks saab kasutaja detaillehel kiire ülevaate kaubeldavast aktsiast ning rakendus on mõeldud eelkõige investoritele kasutamiseks.

Nõrgem pool on andmete puudulikkus väiksemate börside korral – Tallinna börsi aktsiate kõik olulised finantsnäitajad ei ole saadaval. Samuti ei ole rakenduses ajaloolisi tulemusi, mille saamiseks peaks jällegi otsima infot aastaaruannetest.

2.2 Kasutatud tehnoloogiad

Käesolev peatükk annab lühiülevaate töös kasutatud tehnoloogiatest.

2.2.1 Projekti arenduskeskkonnad ja koodi hoidla

Peamiseks arenduskeskkonnaks oli selle töö raames IntelliJ IDEA versioon 2019.2. Mainitud keskkonnal on peale Java virtuaalmasina peal jooksvate keelte tugi ka teistele

projektis kasutatavatele keeltele nagu JavaScript, SQL (*Structured Query Language)* ja HTML (*HyperText Markup Language*). Lisaks saab arenduskeskkonda integreerida Git versioonikontrolli süsteemi ja levinud Maven paki ehitamise tööriistaga [4]. Valikut lihtsustas ka töö autori varasem kokkupuude sama keskkonnaga.

Pythoni *scraper* tööriista kirjutamiseks kasutati Visual Studio Code versiooni 1.52.1. Vaatamata sellele, et IntelliJ IDEA-l on Python-i toe jaoks olemas oma *plugin*, siis autor otsustas kasutada Visual Studio Code-i nimelt ülesseadmise lihtsuse tõttu.

Antud rakendus on eelmise tudengi töö jätk. Lisaks käesoleva töö autorile jätkub tarkvara arendus ka järgmistel semestritel. Mitme tudengi osaluse tõttu on projekti lähtekood üles laetud privaatsesse Bitbucketi koodirepositooriumi. Lisaks Bitbucketile laetakse kood ajutiselt kaitsmise tarbeks üles ka avalikku GitHub-i repositooriumi (Lisa 2).

2.2.3 *Backend* komponendid ja andmebaas

Projekti *backend* ehk tagarakendus koosneb kolmest komponendist. Pythoni versioonis 3.8.6 on kirjutatud kood, mis korjab regulaarselt aktsiahindu rakendusevälisest allikast. Pythoni töövoo automatiseerimiseks Microsoft Windows platvormil kasutati PowerShelli skriptimise keelt.

Ühenduslüliks aktsiahindade korjamise töövoo ja Java tagarakenduse vahel on RabbitMQ sõnumiedastaja. Projektis on kasutusel RabbitMQ versioon 3.8.9, mille kasutamiseks on vajalik Erlang 23.1 tugi.

Projekti tagarakendus, mis suhtleb kasutajaliidesega, on kirjutatud Java keeles. Rakenduse ehitamisel on kasutatud Spring Boot raamistikku, mis võimaldab veebirakendusi kiiremini ja lihtsamalt üles seadistada, konfigureerida ja testida. Andmebaasisüsteemina on kasutusel PostgreSQL, mis kasutab päringute tegemiseks SQL keelt.

2.2.4 *Front-end* komponent

Front-end rakenduse loomiseks on kasutatud JavaScripti programmeerimiskeelt ja selle kasutajaliideste loomiseks mõeldud React teeki. React võimaldab luua keerukaid kasutajaliideseid väikestest ja isoleeritud koodiosadest nimega komponendid.

Komponendid tagastavad JSX-i (*JavaScript XML*), mis võimaldavad luua korduvkasutatavaid HTML elemente ja millest saab üles ehitada veebilehti [5].

Lehekülgede ülesehitamisel on rakendatud PrimeReact-i komponentide teeki, millega on võimalik kiiresti luua erinevaid kasutajaliidese elemente ja mis ühildub automaatselt Reacti tehnoloogiatega [6]. Lisaks on osade disainiküsimuste lahendamiseks kasutatud CSS märgistuskeelt.

2.3 Töö teostamise protsess

Projekti tegemise esimeses etapis analüüsiti ja pandi kirja funktsionaalsed ja mittefunktsionaalsed nõuded. Nõuete defineerimisel lähtuti funktsionaalsustest, mis aitaksid tõsta veebirakenduse väärtust ja unikaalsust võrreldes teiste samataoliste rakendustega.

Rakenduse arendamisel ei kasutatud konkreetset arendusmetoodikat. Töö autor kasutas tööülesannete märkimisel ning ülevaate tegemiseks isiklikku *backlog*-i.

Juhendajaga suhtlus toimus enamasti kirja teel, kus töö autor andis ülevaate ajaperioodi jooksul tehtud ning järgnevatest planeeritud sammudest. Lisaks toimusid koosolekud läbi Zoomi, kus autor esitles juhendajale tehtud funktsionaalsusi ning arutati järgmiste sammude ja prioriteetide üle.

3 Töö tulemused

Peatükis tuuakse välja projekti funktsionaalsed ja mittefunktsionaalsed nõuded. Seejärel kirjutatakse täpsemalt lahti rakenduse arhitektuur ja disain. Lõpetuseks tehakse ülevaade kirjutatud koodist ning testidest.

3.1 Nõuded

Analüüsides rakenduse esmases versioonis valminud projekti ning hinnates millised funktsionaalsused annaksid veebirakendusele lisandväärtust võrreldes olemasolevate lahendustega, on praeguses arendusetapis kirjeldatud järgnevad nõuded.

Funktsionaalsed nõuded:

- Investoril on võimalik rakenduses näha aktsia hinnainfo uuenemist kindla aja tagant.
- Investoril on võimalik näha finantssuhtarvude uuenemist kindla aja tagant
- Investoril on võimalik näha ettevõtte kohta ajaloolisi andmeid, et aidata teha paremaid investeerimisotsuseid.
- Investoril on võimalik korraga võrrelda kahte ettevõtet omavahel, et paremini otsustada investeerimisobjekti üle.
- Investoril on võimalik luua endale kasutajanime ja parooli nõudev kasutajakonto.
- Investoril on võimalik koostada ja hallata personaalset aktsiate jälgimisnimekirja.

Mittefunktsionaalsed nõuded:

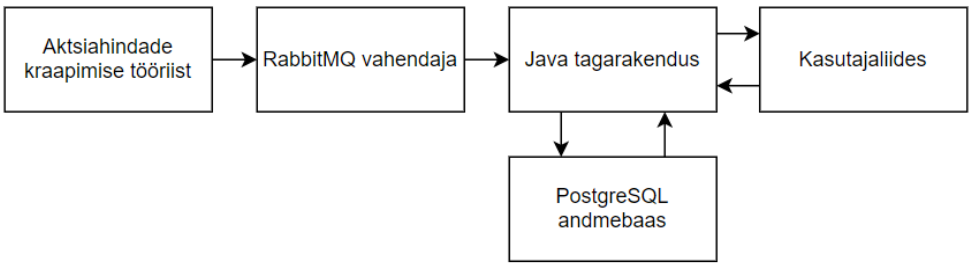
- Rakenduse olulised funktsionaalsused töötavad probleemideta Google Chrome ja Mozilla Firefox uuemates brauserites.
- Rakendus peab vastama andmepäringule ning laadima kasutajale lehekülje kolme sekundi jooksul.
- Rakendus peab olema baastasemel turvaline ning kasutaja andmed on enamlevinud rünnakute eest kaitstud.

- Rakenduse funktsioonid peavad olema kasutaja jaoks esitletud võimalikult arusaadavalt ja olema intuitiivselt kasutatavad.
- Rakenduse arhitektuur peab võimaldama selle lihtsat laiendamist ja edasiarendamist.

3.2 Arhitektuur

Veebirakenduse ehitamisel on kasutatud kihilist arhitektuuri, mille komponentide omavahelist sõltuvust on proovitud teha minimaalseks. Arhitektuurimudel jaotab tarkvarakomponendid loogilisteks horisontaalseteks kihtideks, kus igal kihil on projekti sees oma roll [7].

Projektis on presentatsioonikihiks React-i kasutajaliides, mis teostab andmete saamiseks päringuid tagarakendusele ja kuvab kasutajale liidest ning seal sees olevaid elemente. Java tagarakendus tegeleb põhilise äriloogikaga. Äriloogika alla kuulub ka aktsiahindade kraapimise töörist, mis korjab väliselt veebilehelt reaalajas aktsiahindu ning seejärel saadab sõnumina RabbitMQ vahenduslülisse. RabbitMQ on andmevahetuse kanaliks aktsiahindade tööriista ja Java tagarakenduse vahel. Lisaks tegeleb Java komponent andmebaasipäringute, finantssuhtarvude kalkuleerimise ja kasutajate autentimise ning autoriseerimisega. PostgreSQL on andmebaasi serveriks.



Joonis 6. Komponentide omavaheline seotus.

3.2.1 Aktsiahindade kraapimise tööriista arhitektuur

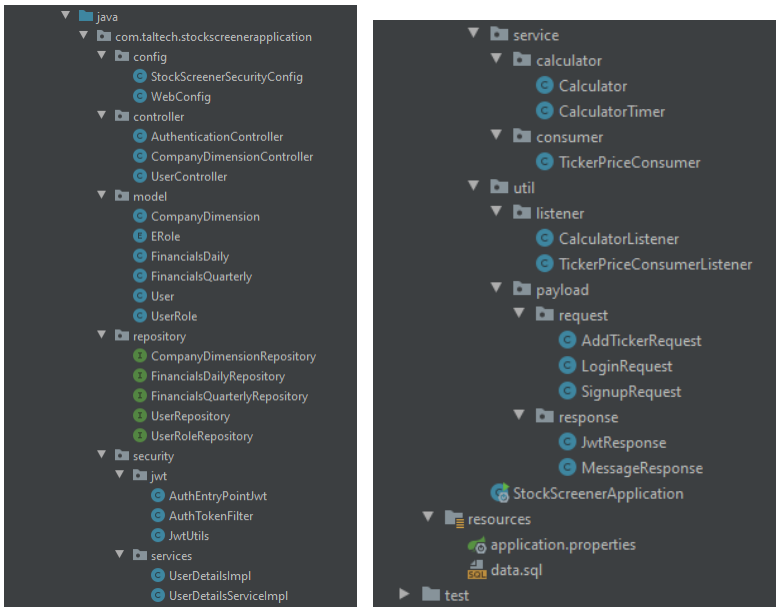
Nasdaq Data on Demand pakub tasulise teenusena API-t, millega on võimalik saada ligipääs ajaloolistele andmetele ja muule finantsinfole [8]. Selle teenuse klientideks on enamasti professionaalsed finantsinstitutsioonid ning käesoleva töö tarbeks see ei sobinud.

Sel põhjusel otsustati luua eraldi tööriist, mis korjab kindla intervalli tagant aktsiahinna reaalväärtusi Nasdaq Baltic kodulehelt. Manuaalselt andmeid kopeerida on aeganõudev ning vigaderohke. Selleks, et veebirakenduses kajastuksid asjakohased andmed, eraldatakse veebilehtedel olev info automaatselt. Pythoni tööriist keskendub info eraldamisele lähtekoodist ja vastavat tegevust nimetatakse kraapimiseks (inglise keeles *scraping*) [9].

Aktsiahindade kraapimise tööriist on oma mahult väga väike ning täidab vaid ühte ülesannet – börsil kaubeldavate ettevõtete aktsiahindade korjamise ja saatmisega sõnumijärjekorda. Arhitektuurilise stiili poole pealt sarnaneb lahendus üksiku mikroteenusega. Rakendus on eraldiseisev, täidab konkreetset ärilist eesmärki ning suhtleb komponentidega läbi kolmanda osapoole (RabbitMQ *broker*). Teenuse käivitamise protsess on Windows masinate peal täielikult automatiseeritud kasutades selle tarbeks kirjutatud PowerShelli skripti.

3.2.2 Java *backend* komponendi arhitektuur

Java tagarakenduse edasiarenduse käigus on üritatud jätkata kihilise arhitektuuristiiliga, millega alustati töö esimeses faasis. Tagarakenduse kood üritatakse samuti jaotada pakettide järgi eri funktsioone täitvateks osadeks. Kihiline arhitektuur aitab arendajal keskenduda rohkem konkreetse kihiga töötamisele ja on lihtsasti arusaadav nii koodi hooldajale kui ka edasiarendajale.



(a) (b)

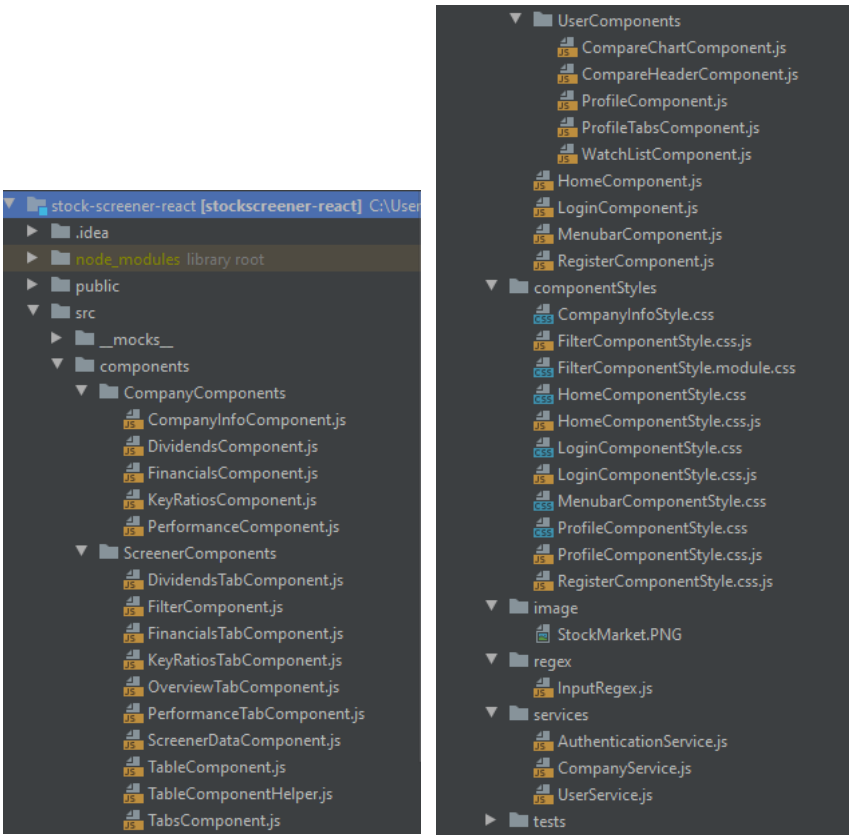
Joonis 7. Projekti tagarakenduse arhitektuur (a) ja (b).

Järgnevalt on toodud selgitused iga paketi sees olevate klasside ülesannetest:

- *Config* paketis olevad klassid konfigureerivad teistes domeenides olevatele ressurssidele päringute tegemist ning kasutaja autentimisega seotud protseduure.
- *Controller* paketis olevad klassid kontrollivad andmete liikumist päringute kaudu.
- *Model* paketis olevad klassid kirjeldavad andmetabeleid, tabelite vahelisi ühendusi ja valideerimist.
- *Repository* paketis olevad klassid võimaldavad teostada erinevad päringuid andmebaasis olevate andmete vastu.
- *Security.jwt* paketis on JWT (*JSON Web Token*) võtmega seotud tegevused nagu võtme genereerimine, valideerimine, filtri defineerimine.
- *Security.services* paketis on abiklassid kasutaja autentimise teostamiseks.
- *Service.calculator* paketis olevad klasse kasutatakse finantssuhtarvude arvutamiseks ning perioodiliselt jooksva kalkulaatori töövoo defineerimiseks.
- *Service.consumer* paketis olev klass teostab RabbitMQ's olevate sõnumite korje ning salvestamise.
- *Util.listener* paketis olevad klassid defineerivad *listener* meetodid, mis algatavad perioodilised töövood tagarakenduse käivitamisel.
- *Util.payload.request* klassid kirjeldavad *contoller* päringute sisendiks olevaid objekte.
- *Util.payload.response* klassid kirjeldavad *controller* päringute väljundiks olevaid objekte.
- *Resources* kaust sisaldab rakenduse parameetreid sisaldavat faili ning sql skripti alusandmestiku loomiseks.
- *Test* kaust sisaldab rakenduse kontrolliks koostatud *unit*-teste.

3.2.3 *Front-end* komponendi arhitektuur

Kasutajaliidese sisemine arhitektuur on loodud monoliitse üksusena, kus eri komponendid on üksteisest sõltuvuses. Sarnast arhitektuuri jätkatakse lihtsuse mõttes ka edasiarenduses, kuna töö autoril on minimaalne React-i arenduskogemus ja keerukamate lahenduste välja mõtlemine oleks selle töö raames liiga ajakulukas.



(a) (b)

Joonis 8. Projekti kasutajaliidese arhitektuur (a) ja (b).

Järgnevalt on toodud selgitused tähtsaimatest kasutajaliidese sees olevate komponentide ülesannetest:

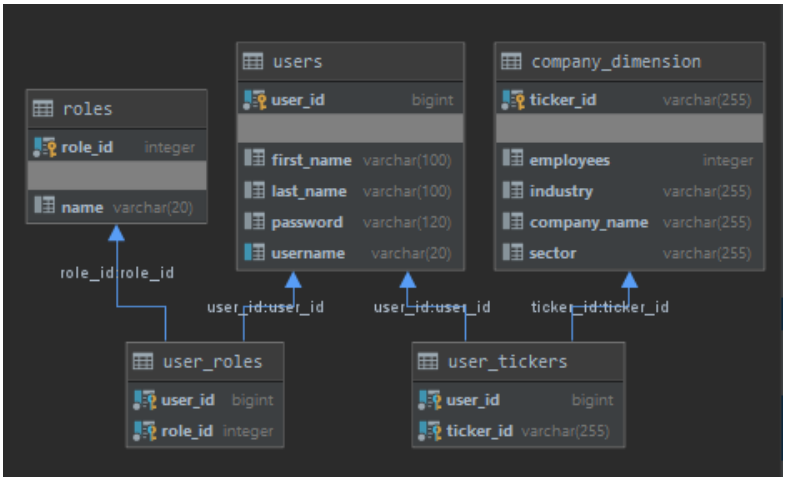
- *ScreenerComponents* kaustas on tabelivaate kokkupanekuga seotud komponendid.
- *UserComponents* kaustas on komponendid, mis vajavad ligipääsuks kasutaja sisselogimist.
- *CompanyInfoComponent* käsitleb börsiettevõtte detailvaadet.
- *CompanyComponents* kaust sisaldab ettevõtte finantsandmete grupeerimiseks kasutatud komponente.
- *HomeComponent* viitab esilehele.
- *LoginComponent* on mõeldud kasutaja sisselogimiseks.
- *MenuBarComponent* on iga lehe ülaservas asuva menüürea jaoks.
- *RegisterComponent* ülesandeks on teostada kasutaja registreerimist.
- *Services* pakk sisaldab tagarakendusele andmete küsimiseks koostatud päringuid ja autentimisega seotud toiminguid.

Lisaks on disaini parandamiseks mõeldud paketid *componentStyles* ja *images*. *Regex* valideerib tekstiväljade sisu ning *tests* klass sisaldab kasutajaliidese teste.

3.2.4 Andmebaasi arhitektuur

Rakenduse andmebaas koosneb seitsmest tabelist. Esmaversioonist on rakenduses juba olemas *company_dimension*, *financials_daily* ja *financials_quarterly* tabelid. *Company_dimension* tabel sisaldab börsil kaupleva ettevõtte üldinfot. Ülejäänud kahe tabeli andmed on jaotatud vastavalt ajahorisondile - *financials_daily* tabel sisaldab börsiettevõtte finantsnäitajaid, mis võivad muutuda päeva lõikes ning *financials_quarterly* esitab kord kvartalis uuenevaid näitajaid. Tabelid on omavahel seotud läbi *@PrimaryKeyJoinColumn* annotatsiooni.

Käesoleva töö raames lisati uued *users*, *roles*, *user_roles* ja *user_tickers* andmetabelid. *Roles* tabel defineerib kõik võimalikud eksisteerivad kasutajatasemed, mis erinevad nendele antud õiguste osas. Antud rakendus toetab hetkel ühte kasutajatüüpi - *ROLE_USER*. *Users* tabel salvestab registreeritud kasutajate andmed. *Roles* ja *users* tabelid on omavahel seotud *@ManyToMany* seosega ning *user_roles* on tabel, mis kirjeldab millised rollid on seotud milliste kasutajatega. Rakendusse lisati aktsiate jälgimisnimekirja paneku funktsionaalsus ning seost millisel kasutajal on mis aktsiad nimekirjas, kajastab *user_tickers* andmetabel. *Company_dimension* ja *user* tabelid on omavahel seotud *@ManyToMany* annotatsiooniga. Juhul kui tehakse andmepäring kasutaja jälgimisnimekirjas olevate aktsiate kohta, siis tagastatakse lisaks *company_dimension* infole ka aktsiaga seotud *financials_daily* ja *financials_quaterly* info.



Joonis 9. Projekti uued andmetabelid ning nende omavahelised seosed.

3.3 Disain

Käesolevas peatükis antakse ülevaade projekti tarkvarakomponentide disainimustrite eripäradest.

3.3.1 Andmete kraapimise tööriist ja RabbitMQ

Python-i kraapimise tööriist on oma mahult väike ning disainitud töötama kui skript. Tegemist on rea käskudega, mille täitmine on tehtud kasutaja jaoks võimalikult automaatseks. Tööriista automatiseerimiseks Microsoft Windows platvormil on loodud PowerShelli skript.

Kuna saadetakse vaid ühte tüüpi andmeid, siis on RabbitMQ sõnumite vahendajas loodud vaid üks ühendus, kus Python-i skript on saatjaks ja Java tagarakendus vastuvõtjaks. Sõnumite saatmiseks kasutatakse AMQP-d (*Advanced Message Queuing Protocol*), mis võimaldab pakkuda asünkroonset ühendust.

3.3.2 Java *backend* komponendi disain

Projekti edasiarenduses on jätkatud sama REST API disainimustri kasutamist nagu tarkvara esmases versioonis. REST arhitektuuriline stiil on väga levinud ning rakendatud modernsete veebiteenuste API disainis. Disaini aluseks võeti Mark Masse raamatu põhjal kirjeldatud REST API disainimise reeglid [10].

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. Lisaks kasutati töös URI-de nimetamiseks dokumendiressursi kontseptsiooni, kus viidatakse konkreetsele eksemplarile või andmebaasikirjele. Dokumendi olekuesitus sisaldab tavaliselt ka seotust alamväljade ja muude oluliste ressurssidega.

Käesoleva arenduse käigus tulid GET päringutele lisaks POST ja DELETE päringutüübid. DELETE-i kasutatakse ressursi eemaldamiseks kollektsioonist ning POST-i uue ressursi lisamiseks kollektsiooni ja kontrollerite töö täitmiseks.

HTTP (*Hypertext Transfer Protocol*) päringute vastused sisaldavad staatuskoode, mis peaksid olema vastavuses disainireeglitega. 200 peab olema vastuseks mittespetsiifilisele edukale päringule, 201 kasutatakse kui on loodud kollektsiooni uus ressurss. Näiteks sobib see kood aktsiasümboli lisamisel kasutajaga seotud aktsianimekirja. Rakendusele lisatud kasutaja autoriseerimise funktsioon peaks tagastama 401 juhul kui kasutajal ei ole ligipääsu konkreetsele ressursile.

Lisaks on soovitus, et REST API peaks kasutama eelistatult JSON-i formaati, et struktureerida sõnumis olevat infot. JSON peab olema korrektselt vormindatud võtme-väärtus paaride kogum.

Tagarakenduse edasiarenduses jätkatakse Spring raamistiku alla kuuluva *dependency injection* tehnika kasutamist, mille põhimõttel saab objekt kohale toimetada teise objekti sõltuvused. Selle jaoks kasutatakse *@Autowired* annotatsiooni [11].

3.3.3 *Front-end* komponendi disain

Projekti esimene versioon kasutas erinevaid React raamistikuga seotud disainimustreid, mille kasutamist on edasiarenduses jätkatud. Lisaks on toodud mõned näited käesolevas projektis kasutatud mustritest.

React-i rakendus koosneb eri komponentidest, mis peavad olema kergesti edasi arendatavad ja töötama koos tervikuna. Komponendid võimaldavad sama struktuuri taaskasutada eri andmestike korral ning kasutajaliidese arendusel peaks otsima võimalusi kuidas jaotada elemendid uuesti kasutatavateks osadeks [12]. Jätkatud on ka *Handling Events* tehnika kasutamisega [13], kus kasutaja tegevusega (näiteks nupuvajutusega) liideses seotakse väljakutsutav funktsioon.

Mõnikord on vajalik kuvada komponenti vaid teatud tingimuste täitumisel ehk kasutada *Conditional Rendering* tehnikat [14]. Näiteks börsiettevõtete võrdlustabeli komponent tuleb nähtavale alles siis kui on kasutaja poolt sisestatud sobivad aktsiasümbolid.

Render meetod kuvab React elemendid DOM-i (*Document Object Model*). Kui element on loodud, siis ei ole see enam muudetav. See tähendab, et kasutajaliidese visuaalseks muutmiseks on vajalik luua uus element ning saata see *render* meetodisse. Vahepeal on vajalik saada ligipääsu elementidele, mis on loodud *render* meetodi sees – näiteks

erinevate valideerimissõnumite kuvamiseks. Seda võimalust pakub *ref*-ide kasutamine [15].

3.4 Projekti kood

Suure tõenäosusega jätkatakse projekti arendamist ka teiste tudengite poolt ning seepärast on oluline, et eelnevalt kirjutatud kood oleks kindlat eesmärki täitev, hästi loetav ning efektiivne. Seetõttu on proovitud projekti koodi kirjutamisel võimalikult palju lähtuda *clean code* põhimõtetest.

Siinkohal on toodud mõned näited, mida on proovitud järgida projekti kirjutamisel:

- sisukad nimed funktsioonidele, muutujatele, klassidele jne [16, lk 17-30].
- funktsioonid on võimalikult väiksed ja teevad ühte asja [16, lk 34-35].
- vähendada korduvat koodi [16, lk 48].
- koodi visuaalne vormindamine [16, lk 78-85].

Lisaks kasutati koodi kvaliteedi parandamiseks PMD *Source Code Analyzer* liidest, millel on eelnevalt valmis koostatud reeglistikud ja võimalus teha kohandatud valideerimisreegleid [17]. Olemasolevaid reeglistikke on tööriistal palju ja mõnikord ei ole vajalik iga paranduse tegemine, seetõttu on valinud töö autor kasutamiseks praeguses kontekstis vaid mõistlikumad (näiteks *bestpractices* reeglistik). PMD-l ei ole JavaScripti tuge ning seetõttu kasutati kasutajaliidese ülevaatuseks SonarLint liidest [18].

3.4.1 Python *scraper* tööriista ja PowerShelli skripti kood

Tööriista eesmärgiks on koguda kokku Tallinna börsil tegutsevate aktsiate viimased hinnad (korjamise hetke ajal) ning edastada need väärtused RabbitMQ sõnumijärjekorda.

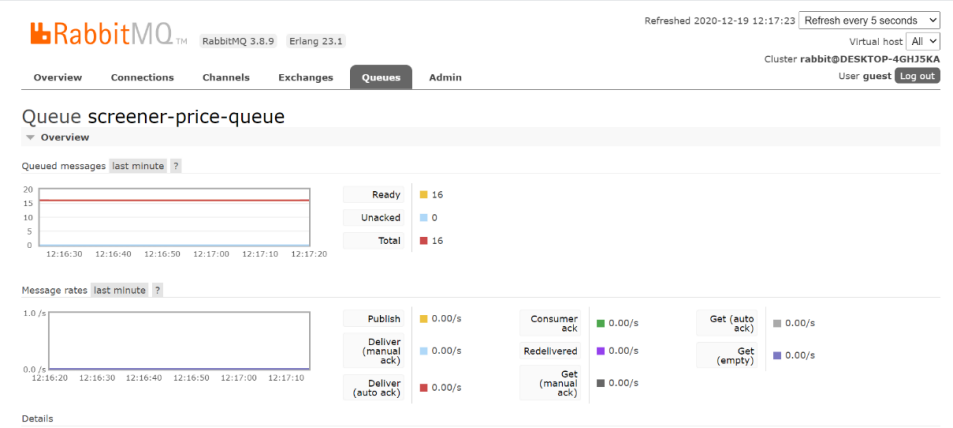
Kõigepealt teostab tööriist HTTP päringu veebilehele, kust tahetakse korjata andmeid. Antud rakenduses korjatakse andmeid Nasdaq Baltic kodulehelt, aktsiate alamlehelt: https://nasdaqbaltic.com/statistics/et/shares. Tööriist kasutab HTTP päringu tegemiseks httplib2 teeki [19], millega tagastatakse päringu vastuse päis ning sisu HTML kujul.

BeautifulSoup on Python-i teek, mida kasutatakse andmete kättesaamiseks HTML ja XML failidest ning mis suudab lisaks käsitleda ka vigase struktuuriga faile (näiteks puuduvad mõned HTML *tag*-id). Päringu vastuse sisu viiakse BeautifulSoup objekti ning

seejärel BeautifulSoup puhastab ja kasutab lxml *parser*-it viimaks selle *parse tree* kujule. Seejärel on võimalik vastavale HTML puule käivitada andmete sees navigeerimiseks ja otsimiseks erinevaid meetodeid [20].

Peale HTML puu analüüsi otsitakse *select* meetodiga üles aktsiatabeli rea märgendid ning seejärel aktsia lühinime ja viimasele hinnale vastavad lahtri väärtused. Väärtused puhastatakse ning koostatakse võtme-väärtus paaridega sõnum, mis saadetakse edasi RabbitMQ vahendajasse.

Sõnumite saatmiseks RabbitMQ-sse kasutatakse Python-i pika teeki [21]. Ühenduse loomiseks kõigepealt defineeritakse AMQP URL. Seejärel luuakse ühendus ja täpsustatakse järjekorra nimi kuhu sõnum saata. RabbitMQ on sõnumite vahendaja – selle eesmärk on aktsepteerida sõnumi saatjalt ning ja edastada need tarbijale läbi AMQP protokolli [22].



Joonis 10. Kuvatõmmis aktsiahindade järjekorra üldvaatest.

Joonis 11. Kuvatõmmis järjekorras olevate sõnumite kuvamisest.

Töö autori lokaalne arenduskeskkond kasutab Microsoft Windows 10 operatsioonisüsteemi. Python-i skripti automaatseks jooksutamiseks loodi täiendav PowerShelli skript, mille ühekordsel käivitamisel luuakse *Windows Task Scheduler*-i korduv töö [23]. Enne Python-i skripti käivitamist viiakse see üle *.py* kujult käivitatava faili (*.exe*) kujule, kasutades selleks PyInstaller paketti. PowerShell on Microsofti poolt ehitatud eri ülesannete automatiseerimise ja konfiguratsiooni haldamise raamistik ning see koosneb käsurea kestast ja skriptikeelest [24].

Selle tulemusel käivitatakse Python-i skript aktsiahindade korjamiseks iga päev kindla ajaintervalli tagant. Veebirakenduse arendustöö järgevates faasides luuakse komponentidele *live*-keskkonnad ning sel hetkel saab kiiresti ja lihtsalt üles seada korduvat töövoogu PowerShelli skriptiga. Hetkel on skript konfigureeritud nii, et aktsiahindade kraapimine toimub esmaspäevast reedeni üheminutiliste intervallide tagant vahemikus kell 10-16 (Lisa 3). Üheminutiline intervall on pandud rakendusele demo tegemise eesmärgil.

3.4.2 Java *backend* komponendi kood

Peatükis käsitletakse eraldi iga uue rakenduse koodi. Tuuakse ülevaade RabbitMQ sõnumite tarbija klassist, kalkulaatori tööprotsessist, *listener*-ide klassidest, Spring Security JWT rakendusest, uutest REST API päringutest ning kirjeldatakse ka andmebaasi andmesisestuse loogikat.

Consumer.java klassis on kirjeldatud ühenduse loomine RabbitMQ vahendajaga. Ühenduse loomiseks serveriga luuakse *ConnectionFactory* objekt, mis vaikimisi on seatud *localhost*-i peale ning *Connection* objekt, mis on aluseks *Channel*-i loomisele. Ühendusi ei panda *try-with-resources* bloki sisse, kuna ühendust on vajalik pidevalt lahti hoida aktsiahindade regulaarseks uuendamiseks. *Channel*-i objekti *queueDeclare* meetod defineerib järjekorra, kust hakatakse tarbima sõnumeid. Järjekord tuleb alati deklareerida, et vältida olukordi, kus sõnumite genereerija (praegusel juhul Pythoni skript) ei ole veel üleval, kuid tarbija soovib juba sõnumeid korjata mitteeksisteerivast järjekorrast. Juhul kui sama nimega järjekord on juba loodud, siis jäetakse operatsioon lihtsalt vahele. *BasicConsume* meetod tarbib sõnumid ning teostab operatsioonid kuidas sõnumeid järgnevalt töödeldakse [25]. Peale sõnumi kättesaamist konverteeritakse see *bytearray* andmetüübist stringiks ning viiakse üle *ObjectMapper* klassi abil *FinancialsDaily*

objektiks. Jackson *ObjectMapper* klass on mõeldud JSON stringide lihtsaks konverteerimiseks Java objektiks [26]. Seejärel uuendatakse *updatePrice* meetodi abil andmebaasi *financials_daily* tabelit uuendatud aktsiahinna infoga. Meetod *updatePrice* on kohandatud SQL UPDATE päring, mis on loodud kasutades Spring Data *@Query* ja *@Modifying* annotatsioone.

Alustamaks järk-järgult reaalsete andmete toomist veebirakendusse, on loodud kalkulaator erinevate finantssuhtarvude arvutamiseks. Arvutused on hetkel teostatud kahele suhtarvule, mis mõlemad kasutavad ühe sisendparameetrina aktsia hetkehinda. Need on seotud taustal jooksva tööprotsessiga, mis teostab arvutusi mitmel korral päevas. Veebirakenduse järgmistes faasides saab laiendada arvutatavate finantssuhtarvude hulka.

Kalkulatsioonid teostatakse klassis *Calculator.java* kahele suhtarvule – P/E ja dividendimäär. P/E (*Price/Earnings*) ehk hinna ja kasumi suhe näitab kas aktsia on üle- või alahinnatud [27]. Dividendimäär on protsent kui suure osa moodustab dividend aktsia hinnast [28].

Vastavad valemid mõlema arvutamiseks on järgmised:

- P/E = aktsia hind / puhaskasum aktsia kohta
- Dividendimäär = dividendi suurus / aktsiahind

Arvutuste teostamiseks korjatakse kõigepealt andmebaasipäringuga arvutusteks vajalikud väärtused *Map* objekti. *Map* objekt valiti kuna see võimaldab salvestada arvud võtme-väärtus paaridena, kus võtmeks on aktsiasümbol ja väärtuseks arvutuseks vajalik arv (näiteks aktsia hind).

Meetod *doCalculations* võtab sisendiks kaks *Map* objekti, kust leitakse ühise võtme väärtuse järgi iga aktsia kohta kalkulatsioonideks vajalikud arvud. Pärast kalkulatsioone tagastatakse uus *Map* objekt, mis sisaldab võtmena aktsiasümbolit ja väärtusena kalkulatsiooni tulemust. Seejärel uuendatakse *calculatePE* ja *calculateDividendYield* meetodites andmebaas arvutatud suhtarvudega.

Kalkulatsioonide teostamise ajaline intervall ja täidesaadetavad meetodid on defineeritud *CalculatorTimer.java* klassis. Kõigepealt luuakse *Timer* objekt, millele kutsutakse välja *schedule* meetod. Meetod võtab sisendiks *TimerTask* objekti kus kirjeldatakse ära kavas olevate tegevuste sisu. Kalkuleeritakse P/E ning dividenditootluse suhtarvud ja

lõpetamisel logitakse vastav rida koos ajatempliga. Lisaks võtab *schedule* meetod sisse *Date* parameetri, millega määratakse ära viivitus enne esimest kalkulatsiooni. Antud veebirakenduses on selleks väärtuseks 10 sekundit. Lisaks viivitusele on määratud ka periood, mis hetkest täidetakse töövoog uuesti. Rakenduses on hetkel selleks määratud 15 sekundit.

Andmete kvaliteedi tagamiseks jooksevad taustal andmete uuendamisprotsessid. Konkreetsemalt on nendeks protsessideks RabbitMQ sõnumite korjamise ja finatssuhtarvude arvutamise ning uuendamise töövood. Selle jaoks on kasutusele võetud Java *ApplicationListener* klass, millega abiga aktiveeritaks nõutud töövood Spring Boot rakenduse käivitamise järgselt. *ApplicationReadyEvent* luuakse hetkel kui rakendus on üleval ja valmis protsessima erinevaid päringuid. See sündmus võetakse parameetriks sisse *onApplicationEvent* meetodisse, mis käivitab vajaliku koodi. *@Order* annotatsiooniga pannakse paika töövoogude käivitamise järjestus. Antud rakenduses on *listener*-ide loogika välja toodud *TickerPriceConsumerListener.java* klassis, mis käivitab aktsia reaalhindade muutmise tööprotsessi ning *CalculatorListener.java* klassis, mis käivitab finantssuhtarvude arvutamise tööprotsessi.

Tagarakenduses on võetud on kasutusse ka autentimise ja autoriseerimise protsess kasutades selleks Spring Security raamistikku ja JWT teeki. Lahenduse eripära on selles, et peale kasutaja edukat autentimist väljastatakse talle serveri poolt genereeritud ja salavõtmega allkirjastatud *token*. Seda *token*-it kasutab hiljem server, et tuvastada kas päringut tegev kasutaja on seesama isik, kes on sisse logitud. Server ei salvesta eraldi infot kasutaja sessioonide kohta.

Kui kasutaja on edukalt tuvastatud, siis server genereerib JWT, allkirjastab selle oma salajase võtmega kasutades HS256 algorütmi ning saadab tagasi brauserile päringu vastusena. *Token*-i aegumise parameeter on seatud tagarakenduse konfiguratsioonis ning hetkel on selle väärtuseks 12 tundi. Pärast 12 tunni möödumist JWT aegub ning kasutajal on vajalik teenuste kasutamiseks end uuesti sisse logida. Serverisse ei salvestata infot kasutajate sessioonide kohta ja kui kasutajaliidesest saadetakse uus päring, siis see peab sisaldama päises JWT *token*-it. Seejärel server verifitseerib kas *token* on peale selle allkirjastamist vahepeal muutunud. Juhul kui ei ole, siis *token* deserialiseeritakse ning antakse kasutajale lubatud ligipääsuõigused ressursile [29].

Spring Security arhitektuuri loomisel kasutati GitHubi repositooriumist koodi [30] ning kohandati vastavalt käesoleva rakenduse eripäradele. Lisandunud on kaks uut mudel objekti klassi – *User.java*, *Role.java* ja loend *ERole.java*. *ERole* klassis defineeritakse ära võimalikud veebirakenduse kasutaja rollid, milleks on antud projektis ainult *ROLE_USER*. *Role* klassis kirjeldatakse kasutaja rolli objekti ning *User* klassis veebirakenduse kasutaja objekti. *User*-iga on seotud väljad id, kasutajanimi, eesnimi, perekonnanimi, parool. Lisaks võib ühe kasutajaga olla seotud üks või rohkem rolli ja aktsiat. Igale mudel objektile on loodud ka *repository* klassid, mis võimaldavad andmebaasiga suhtlust läbi Spring Data JPA. Vastavad klassid on *UserRepository.java* ja *UserRoleRepository.java*.

StockScreenerSecurityConfig.java klassis asub konfiguratsioon ja seal määratakse ära millised leheküljed vajavad ligipääsuks kasutaja autentimist. Kasutaja registreerimisel kontrollitakse kõigepealt kasutajanime unikaalsust ning seejärel salvestatakse kasutaja andmed koos krüpteeritud parooliga andmebaasi. Kasutaja sisselogimise päringus saadetakse kasutajanimi ja parool *UsernamePasswordAuthenticationToken* objekti, seejärel Spring Security *AuthenticationManager* teostab *UserDetailsService* ja *PasswordEncoder* klasside abiga kasutaja autentimise. Autentimise õnnestumisel saadetakse päringu vastuses kasutaja detailid ning *Authentication* objekt.

Authentication objekti kasutatakse sisendiks JWT *token*-i loomiseks *createJwtToken* meetodiga *JwtUtils.java* klassis. Meetod defineerib ära milliseid andmeid soovitakse JWT-s hoida. Antud rakenduse korral koostatakse *token* juba edukalt tuvastatud kasutaja kasutajanimest ja *token*-i aegumise ajatemplist.

AuthTokenFilter.java klass kirjeldab filter meetodit, mida viiakse läbi kord päringu tegemisel. Päringu saabumisel serverisse eraldatakse autoriseerimise päis ja valideeritakse selles sisalduv JWT. Kui valideerimine on edukas, siis eraldatakse *token*-ist kasutaja detailid ja *Authentication* objekt. Valideerimist teostab *JwtUtils.java* klassi meetod *validateJwtToken*.

Kasutajaliides teeb päringuid tagarakendusele kasutades REST API-t. Veebirakenduse esimeses versioonis oli kirjeldatud kahte börsiettevõtete andmeid küsivat GET päringut:

- GET */companies* päring tagastab kõik *company_dimension* tabelis määratletud ettevõtted aktsiasümboli järgi.

- GET */companies/{tickerId}* päring tagastab üksiku aktsia koos kõigi seotud finantsandmetega. *TickerId*'ks on aktsiasümboli väärtus, näiteks TKM1T.

Edasiarenduses lisandus viis päringut, meetoditena lisandusid POST ja DELETE. Uued päringud on kättesaadavad *AuthenticationController.java* ja *UserController.java* klassides.

AuthenticationController.java:

- POST */auth/signup* päring võtab sisendiks *SignupRequest* objekti, mis koosneb kasutajanimest ja paroolist.
    - Andmete õigsusel luuakse uus kasutaja ning tagastatakse sõnum „*User registered successfully*!"
    - Korduva kasutajanime korral kuvatakse vastuseks teade „*Username is already taken!*"
- POST */auth/login* päring võtab sisendiks *LoginRequest* objekti, mis koosneb eesnimest, perekonnanimest, kasutajanimest ja paroolist.
    - Andmete õigsusel tagastatakse *JwtResponse*, mis sisaldab loodud *bearer token*'t, kasutaja detaile, rolli ja seotud aktsiaid.
    - Kasutajanime ja parooli vale kombinatsiooni sisestamisel tagastatakse veateade „*Error: Username and password combination was incorrect!*".

UserContoller.java klassi päringud on kõik *@PreAuthorize* annotatsiooniga, mis nõuab päises JWT-i ja kasutaja sobiva rolli olemasolu.

- GET */users/{userId}* päring tagastab id järgi kasutaja detailandmed.
- POST */users/{userId}/tickers* päring lisab üksiku aktsia kasutajaga seotud aktsiate hulka. Sisendiks võetakse *AddTickerRequest* objekt, mis sisaldab aktsiasümbolit.
    - Päringut kasutatakse kasutajaliideses aktsiate jälgimisnimekirja lisamiseks.
    - Edu korral seotakse aktsiasümbol kasutajaga ning tagastatakse teade „*Ticker added successfully!*".
- DELETE */users/{userId}/tickers/{tickerId}* päring kustutab aktsiasümboli ja kasutaja vahelise seose. Päringut kasutatakse aktsia kasutaja jälgimisnimekirjast eemaldamiseks.

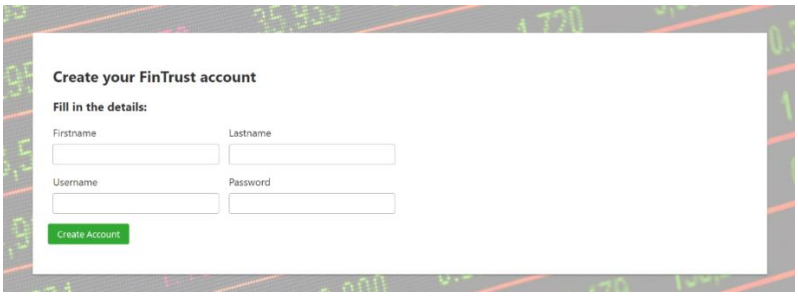
- Edu korral kustutatakse seos ja tagastatakse sõnum „*Ticker deleted successfully!*“.
- Juhul kui soovitakse eemalda aktsiasümbolit, mis ei ole kasutajaga seotud, kuvatakse veateade: „*Error: cannot delete ticker that has not been added!*“.

Veebirakenduse esmases versioonis toimus *mock*-andmestiku sisestamine baasi CSV failide sisse lugemise kaudu. Juhendajaga tehtud arutelu käigus otsustati senisest lahendusest loobuda ning kirjutada ümber andmete loomine rakenduse käivitamisel.

Praegune rakendus kasutab andmetabelite täitmiseks /*resources/data.sql* faili, mis sisaldab endas INSERT INTO lauseid täitmaks *roles*, *company_dimension*, *financials_daily* ja *financials_quarterly* tabeleid *mock*-andmetega. Spring Boot rakenduse käivitamisel kustutatakse vanad ja luuakse kõik tabelid uuesti *spring.jpa.hibernate.ddl-auto=create* parameetriga ja seejärel lisatakse andmestik *spring.datasource.initialization-mode=always* parameetriga.

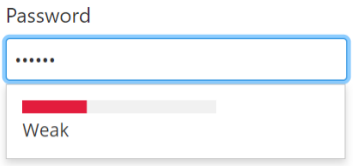
3.4.3 *Front-end* komponendi kood

Projekti kasutajaliides oli kirjutatud kasutades JavaScripti ning sellel baseeruvat React-i raamistikku. Lisaks kasutati PrimeReact elementide loomise võimalust ning CSS märgistuskeelt. Esimeses versioonis valmisid *Home*, *Screener* ja *CompanyInfo* vaated. *Home* vaatele vastab rakenduses *HomeComponent.js*. Vaate URI tee on / ning see võimaldab peamise funktsionaalsusena ettevõtte otsingut aktsiasümboli järgi. *Screener* vaade on kuvab Tallinna börsiettevõtete andmetabelit koos sorteerimis- ja filtreerimisvõimalustega. *CompanyInfo* vaade esitab kasutajale ühe ettevõttega seotud grupeeritud info. URI tee *Screener* vaatele on */screener* ja *CompanyInfo* korral */screener/{tickerId}*.
Järgnevalt antakse ülevaade kasutajaliidesele juurde tulnud vaadetest. Uusi vaateid on kolm – *Login*, *Register* ja *Profile* ning lisaks sellele muutus osaliselt ka menüüriba element.
Register vaatele viitab *RegisterComponent.js* kasutajaliideses ning *Home* vaatest saab sellele navigeerida valides menüüribalt *Register*. Lehekülje eesmärgiks on teostada kasutaja registreerimine, mille jaoks on vaja sisestada eesnime, perekonnanime, kasutajanime ning parooli väljad. Kõik väljad on kohustuslikud ning sisendite kontrollimiseks kasutatakse HTML5 vormi valideerimise võimalusi.



Joonis 12. *Register* vaate sisselogimisvorm.

Vormi väljade ehitamiseks kasutati PrimeReact *InputText* [31] ning paroolivälja jaoks *Password* [32] komponenti. Paroolivälja sisestatud sümbolid on peidetud ning lisaks näidatakse edenemisriba parooli tugevuse kohta.



Joonis 13. Paroolisisestusvälja edenemisriba.

Vormi esitamisel kutsutakse välja *AuthenticationService.js* all kirjeldatud *registerUser* meetod andmete lisamiseks serverisse. Meetod teostab *axios* [33] teegi abil POST */register* päringu ning tagastab vastuse JSON objekti kujul. Õnnestumisel peidetakse registreerimisvorm ning selle asemel kuvatakse liideses POST päringu vastus. *„Proceed to login?“* nupuvajutusel viiakse kasutaja *Login* vaatesse.

Joonis 14. Vaade peale registreerimise õnnestumist.

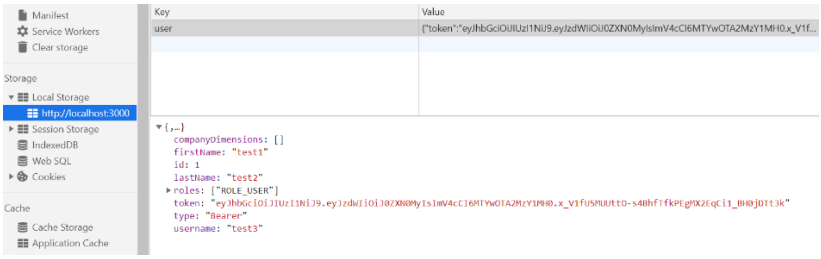
Juba kasutusel oleva kasutajanime sisestamisel kuvab rakendus veateate. Registreerimisvormi peitmist sel juhul ei toimu.

Joonis 15. Korduva kasutajanime sisestamisel kuvatav veateade.

Login vaatele vastab */login* URI ning rakenduses vastab sellele *LoginComponent.js*. Kasutajale kuvatakse vorm kasutajanime *InputText* ja *Password* sisendväljadega. Väljade täitmisel teostatakse serverile *axios* teegi abil POST */login* päring.

Joonis 16. Kasutajaliidese *Login* vaade.

Juhul kui päring tagastab *token*'i, siis salvestatakse see koos ülejäänud vastuse sisuga brauseri lokaalsesse hoidlasse (inglise keeles *local storage*). Seejärel viiakse kasutaja */user/{usename}* lehele, mis vastab *Profile* vaate URI-le.



Joonis 17. Kasutaja andmed brauseri lokaalses hoidlas.

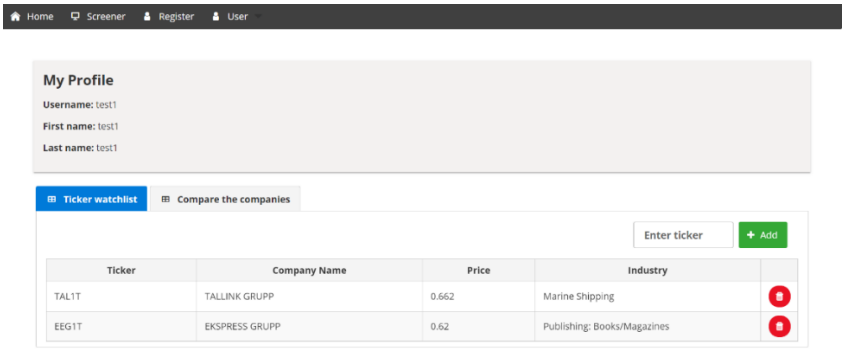
Vormi esitamiseks peavad mõlemad väljad olema täietud. Juhul kui kasutajanime ja parooli kombinatsioon on vale, siis kasutajale kuvatakse vastav veateade.

Joonis 18. Veateade vale kasutajanime ja parooli kombinatsiooni sisestamise korral.

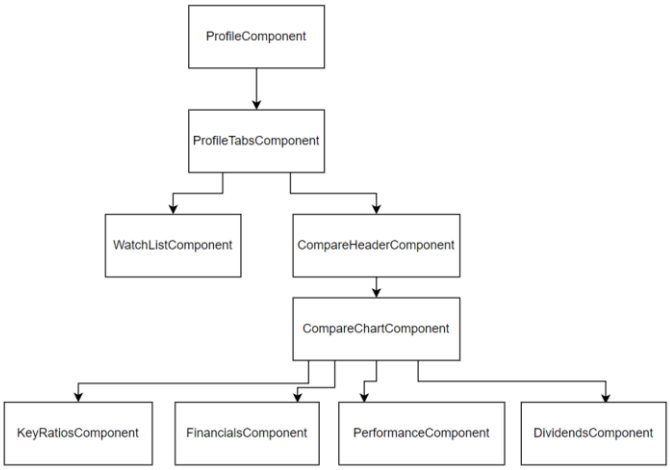
Kasutaja *Login* vaatesse saab navigeerida kahel juhul – *Register* vaatest peale vormi edukat täitmist või kasutades menüüriba *User* alammenüü *Dashboard* valikut. Juhul kui kasutaja andmeid ei ole brauseri lokaalses hoidlas, siis *Dashboard* alammenüü avab *Login* vaate. Andmete olemasolu korral viiakse kasutaja *Profile* vaatesse. Menüü *Logout* valik käivitab *logOut* funktsiooni mille tulemusel eemaldatakse kasutaja andmed lokaalsest hoidlast ning seejärel suunatakse *Home* vaatesse.

Joonis 19. Menüüriba ülesehitus.

Profile vaate URI tee on */user/{username}* ning funktsionaalsusteks on kasutaja info kuvamine, isikliku aktsiate jälgimisnimekirja koostamine ja kahe börsiettevõtte finantsandmete võrdlemine. See on üles ehitatud viiest komponendist, kus *CompareChartComponent.js* koosneb omakorda neljast väiksemast komponendist (Joonis 21). Vaatele ligipääsemiseks peab olema rakenduse kasutaja sisse logitud ehk tema andmed peavad olema kajastatud brauseri lokaalses hoidlas.



Joonis 20. Kasutajaliidese *Profile* vaade



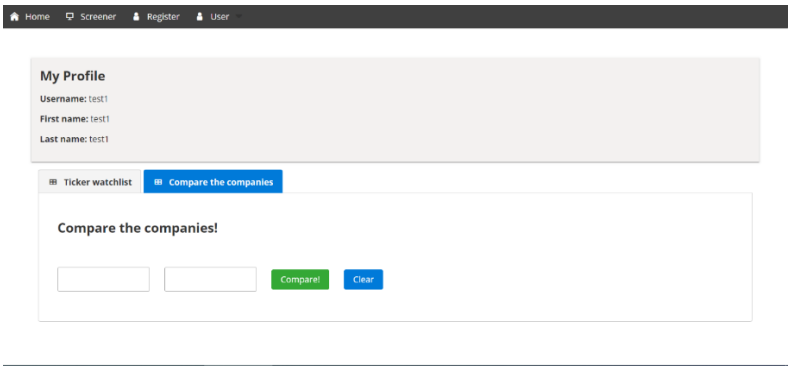
Joonis 21. *Profile* vaate komponentide hierarhia.

ProfileComponent eesmärgiks on registreeritud kasutaja andmete kuvamine. Selleks on kasutatud *Card* komponenti [34], milles esitatakse lokaalsesse hoidlasse salvestatud kasutajanimi, eesnimi ja perenimi. Kasutaja andmete kättesaamiseks kutsutakse välja *getCurrentUser* meetod.

ProfileTabsComponent kasutab PrimeReacti *TabView* elementi [35], et jagada vaade kahe funktsionaalsuse vahel. „*Ticker watchlist*“ *tab* kuvab kasutajaga seotud aktsiad ehk käitub jälgimisnimekirjana ja „*Compare the companies*“ *tab* võimaldab teostada börsifirmade võrdlust. *Tab*-ide vahel navigeerides lehte ei uuendata, seega objektide väärtused eri vaadete liikumise vahel ei muutu.

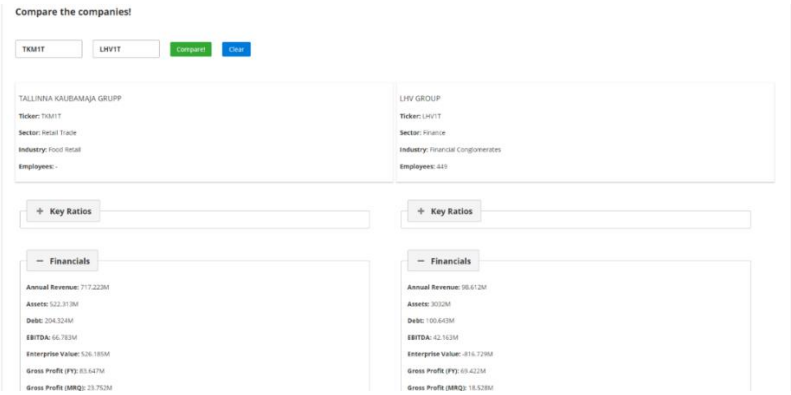
WatchListComponent täidab ettevõtete aktsiate jälgimisnimekirja kuvamise ja haldamise eesmärki. Vaate DOM-i liidese sisse lugemisel kutsutakse välja *componentDidMount* meetod, mis omakorda käivitab *getPublicContent* ja *getUserContent* meetodid. *GetUserContent* meetod teostab GET */users/{user_id}* päringu, mis tagastab kasutajaga seotud aktsiasümbolid, mida seejärel kasutatakse *DataTable* komponendi abil jälgimisnimekirja koostamiseks [36]. Aktsia jälgimisnimekirja lisamiseks sisestatakse *AutoComplete* [37] välja korrektne sümbol ning „*Add*“ nupu vajutamisel käivitatakse *updateTickerList* meetod, mis lisab aktsia läbi POST */users/{userId}/tickers* päringu ja uuendab lehekülje. *AutoComplete* võrdleb sisendit *getPublicContent* meetodi vastusena saadud aktsiate nimekirjaga ning pakub seejärel kasutajale võimalikke variante. Vale sisendi korral kuvatakse kasutajale veateade. Nimekirjast kustutamiseks on *DataTable* tabeli viimases reas prügikasti ikoon, millele vajutades teostatakse läbi *deleteTicker* meetodi DELETE */users/{userId}/tickers/{tickerId}* päring.

CompareHeaderComponent'i eesmärk on võimaldada sisselogitud kasutajale kahe erineva ettevõtte võrdlemist aktsiasümboli järgi. Vaate laadimisel loetakse *componentDidMount* meetodi abil sisse kõik olemasolevad aktsiasümbolid. Võrdluse jaoks on kasutajale kuvatud kaks *AutoComplete* välja, kuhu sisestada võrreldavad aktsia sümbolid.



Joonis 22. Börsiettevõtete võrdlust teostav *tab*.

Compare nupule vajutamisel käivitub *onCompare* meetod, mis teostab GET */companies/{tickerId}* päringu mõlema aktsiasümboli info leidmiseks. Seejärel kuvatakse *CompareChartComponent*, mis esitab valitud aktsiasümbolite finantsandmete sisu kõrvuti võrdlusena. Andmete grupeerimiseks on kasutatud PrimeReact-i *Fieldset* komponenti [38] ning samu andmegruppe, mis esinesid juba *CompanyInfo* vaates. Andmegruppide paremaks kuvamiseks ning koodi duplitseerimise vähendamiseks on loodud *KeyRatiosComponent*, *FinancialsComponent*, *PerformanceComponent* ja *DividendComponent*, mida kasutavad *CompareChartComponent* ja *CompanyInfo* vaade. Valitud aktsiasümbolite info saadetakse hierarhiliselt ülemast komponendist alamkomponenti kasutades *props* parameetrit. Juhul kui sisestatud sümbolid pole erinevad või ei eksisteeri, siis kuvatakse veateade.



Joonis 23. Aktsiate võrdlustabel Tallinna Kaubamaja ja LHV näitel.

Lisaks on võimalik ka osa andmegruppe peita kasutades *Fieldset*-i „*toggleable collapsed*“ võimekust. *Clear* nupp käivitab *onClear* meetodi, millega tagastatakse algseis.

3.5 Projekti komponentidele kirjutatud testid

Testid koodi kontrollimiseks on loodud Java tagarakendusele ning React-i kasutajaliidesele. Tagarakenduse testid katavad *controller* klasside ning kalkulaatori funktsionaalsuse tööd. Selleks on kasutatud *@SpringBootTest* annotatsiooni ning JUnit5 funktsionaalsusi. Arenduse esimeses osas lisatud *CompanyDimensionControllerTest* klass katab *CompanyDimensionController* klassi funktsionaalsusi.

Edasiarenduses lisati *AuthenticationControllerTest*, *UserControllerTest* ja *CalculatorTests* testklassid ning eemaldati *CompanyDimensionSpringBatchConfigTest*.

Eemalduse põhjus oli algandmete loomise muudatus, kus CSV failidest lugemise asemel toimub andmete sisestamine SQL INSERT INTO lausete abil.

AuthenticationControllerTest sisaldab *unit* teste, mis katavad kasutaja registreerimise ning sisselogimisega seotud funktsionaalsusi. *UserControllerTest* katab kasutaja andmete pärimise ning aktsiate lisamise ning eemaldamisega seotud toimingud. *CalculatorTests* sisaldab *unit* teste, mis katavad suhtarvude arvutamise ja uuendamisega seotud stsenaariumid.

Projekti kasutajaliidese testimisel on kasutatud Jest raamistikku ning Enzyme abitööriista. Lisaks rakenduse esmases versioonis valminud testidele on praeguses versioonis lisatud *unit* teste *Register* ja *Login* vaadetele. Samuti lisati teste *CompanyInfo* ja *CompareChart*-le ning nende alamkomponentidele. Edasiarendusega muutusid ka mõned CSS-i disainielemendid ning seetõttu tuli uuendada eelmise versiooni *snapshot* testi.

4 Analüüs ja järeldused

Käesolevas peatükis analüüsitakse kasutatud tehnoloogiaid ning selgitatakse valiku põhjuseid. Lisaks vaadatakse üle töö vastavus kirja pandud funktsionaalsetele ning mittefunktsionaalsetele nõutele ja analüüsitakse rakenduse arhitektuuri ning disaini. Lõpetuseks antakse mõtteid kuidas võiks rakendust järgnevates etappides edasi arendada.

4.1 Kasutatud tehnoloogiate analüüs

Rakenduse esimeses versioonis otsustati kirjutada tagarakendus kasutades Java Spring Boot raamistikku ja kasutajaliides JavaScripti Reacti võimalusi. Sellele lisaks kasutati kasutajaliidese elementide kiireks loomiseks PrimeReact-i kogu. Käesoleva töö autor omas mõnevõrra kokkupuudet Java programmeerimiskeelega ning seetõttu oli lihtsam tagarakenduse võimalusi edasi arendada. JavaScripti ja eelkõige React raamistikuga oli eelnev kogemus minimaalne ja seetõttu oli arendustöö läbiviimine ka keerulisem. Samas leiab töö autor, et kasutatud tehnoloogiad on oma võimekuselt projekti tegemiseks sobivad ning nendega võiks jätkata ka järgnevatel arendusetappidel.

Projekti praeguses arendusfaasis loodi *scraper* tööriist, mis kasutab andmete edastamiseks RabbitMQ-d ja AMQP protokolli. Andmete korjamist veebilehtedelt võimaldavad eri tehnoloogiad, näiteks Node.js ja Ruby. *Scraper* tööriista skript on kirjutatud kasutades Python-i programmeerimiskeelt. Valiku kasuks otsustas juhendaja soovitus, Python-i lihtsus ja kraapimiseks sobivate tehnoloogiate olemasolu. Andmete kraapimiseks on olemas sobivad teegid (BeautifulSoup) ning samal ajal on sellel erinevate platvormide tugi (Windows, Linux). Kuna andmete korje on päevas mitmeid kordi teostatav tegevus, siis loodi Python-i skripti regulaarselt käivitav PowerShelli skript. PowerShell osutuks valituks, kuna töö autori arendusmasin kasutab Microsoft Windowsi operatsioonisüsteemi. Vajadusel saab automatiseerimise skripti üle kirjutada ka Linux platvormile kasutades *cron* teenust [39].

RabbitMQ vahendaja lisati ühenduslüliks Python-i skripti ja Java programmi vahele eesmärgiga minimaliseerida tarkvarakomponentide omavahelist sõltuvust.

Sõnumivahendaja kasutamine teeb lihtsamaks komponentide ühekaupa uuendamise, ühe komponendi jõudlus ei mõjuta teist komponenti ja sõnumite saatmise ebaõnnestumisel ei ole teine komponent mõjutatud. Sõnumite vahendaja pakub ajutist kohta saadetavate andmete hoidmiseks, võimaldades rakendustel neid saata ja vastu võtta kui see on neile vajalik. RabbitMQ valiku kasuks võrreldes teise vahendajatega otsustas selle lihtne ülesseadistus, võimalus integreerida enimkasutatavate programmeerimiskeeltega, nii mälus kui püsivate sõnumite loomise tugi ja üleüldine laiendatavus. Arenduse käigus uuriti ka Kafkat ja Redist. Kafka on tugevama jõudlusega, kuid mõeldud eelkõige suurtesse süsteemidesse, kus sõnumite hulk ulatub päevas miljonitesse [40]. Redis on mälupõhine ning seetõttu on suurem oht sõnumite ära kaotamiseks [41]. Rakenduse praegune versioon on loodud kasutades üks-ühele ühenduse loomise võimalust, kuid RabbitMQ võimaldab luua ka ühendusi, kus ühele sõnumile on mitu tarbijat. AMQP protokolli valiku kasuks võrreldes HTTP protokolliga on võimekus toetada ühepoolset asünkroonset sõnumite saatmist, mis ei eelda töö jätkamiseks vastuse saamist [42]. Vastavad tehnoloogiad valiti eelkõige mõeldes järgnevatele arendusetappidele, kus töödeldavate andmete hulk on suurem ning vajadus komponentide vahel sõnumeid saata muutub olulisemaks.

Kuna tagarakendus on kirjutatud Javas kasutades Spring-i raamistikku, siis oli edasiarenduses loogiline samm võtta kasutusele kasutajate autentimiseks ja autoriseerimiseks Spring Security. Kasutajat saab autentida erinevalt – üheks levinud meetodiks on identiteedi salvestamine sessiooni küpsiste sisse, kus serveri poole peal salvestatakse sessiooni identifikaatorid. Teine levinud viis on kasutada JWT *token*-it, kus kasutaja info salvestatakse kasutajaliidese poolel ning server vaid dekodeerib võtme. Rakenduses otsustati kasutada JWT meetodit, kuna see lähenemine toetab veebirakenduse lihtsamat laiendamist ning parandab jõudlust. Näiteks soovib kasutaja tulevikus sisse logida mitmest seadmest või kasutab teist domeeni, siis JWT-ga autentimiseks piisab vaid korrektse *token*-i olemasolust päringu päises. Samuti on mobiilsetes platvormides ja mitme serveriga rakendustes sessioonide kasutamine raskendatud. JWT-sse saab salvestada erinevad andmeid, näiteks kasutaja rolli mille läbi on võimalik vähendada andmebaasipäringute hulka ja parandada jõudlust [43] [44].

4.2 Töö tulemuste analüüs nõuete baasil

„Nõuded“ peatüki all loetleti edasiarenduse aluseks olevad funktsionaalsed ja mittefunktsionaalsed nõuded. Järgnevalt selgitatakse lahti kui suures ulatuses said need täidetud.

4.2.1 Funktsionaalsed nõuded

Esmane nõue oli, et loodud veebirakenduses oleks võimalik näha aktsia hinnainfo uuenemist kindla aja tagant. See nõue sai täidetud – hindade korjamiseks loodi Python *scraper* tööriist, mis kogub aktsiate hindu iga minuti tagant esmaspäevast reedeni kuni kauplemispäeva lõpuni. Üheminutilised intervallid on hetkel seatud lihtsama esitamise eesmärgil ning valmis rakenduses saab perioode kohandada. Reaalhindade info salvestatakse *financials_daily* tabelisse ning kuvatakse REST API kaudu kasutajale.

Veebirakenduse usaldusväärsuste tõstmiseks on vajalik tagada ka teiste andmete perioodiline uuenemine. Selle täitmiseks oli defineeritud finantssuhtarvude regulaarne uuendamise nõue. See sai enamjaolt täidetud kalkulaatori funktsiooni loomisega Java rakenduses. Teostati kahe suhtarvu, P/E ja dividendimäära arvutamine, kus mõlema arvutusvalemi üheks sisendiks olid aktsia reaalväärtused. Juhendajaga arutelu käigus tehti otsus, et esialgu piisab kahest arvutusest ning seda funktsionaalsust saab vajaduse korral järgnevate tudengite poolt lisada. Rakenduse demo eesmärgil toimuvad arvutused iga 15 sekundi tagant, reaalses elus võiks arvutuste intervalliks olla umbes üks tund.

Üheks nõudeks oli veebirakenduses investorile ajalooliste andmete kuvamine. See on oluline nõue, mis hetkel ei saanud täidetud selle mahukuse ning ajaliste piirangute tõttu. Usaldusväärtsete andmete saamiseks on vaja teha põhjalikku eeltööd, kuhu oleks vaja kaasata peale informaatikatudengite ka majandustudengeid. Lisaks tuleb mõelda kuidas lahendada funktsionaalsus andmebaasi tasandil ning kuidas oleks seda kõige mugavam kasutajale liideses kuvada. Järgmistes arendusetappides saaks kasutada *mock*-andmestikku ja luua funktsionaalsus, mida on hiljem võimalik lihtsa vaevaga reaalandmete peale üle viia.

Järgmiseks nõudeks oli luua investorile funktsionaalsus, mis võimaldaks kahte ettevõtet omavahel kõrvuti võrrelda, et teha selle info alusel paremaid investeerimisotsuseid. See oli üks olulisemaid nõudeid, kuna olemasolevad finantsrakendused kõrvuti võrdlemist ei

paku. Nõue sai täidetud *ProfileComponent*-i alla loodud *tab*-ga, kus sai valida kaks ettevõtet aktsiasümboli järgi ning seejärel nende finantsandmeid kõrvutada. Finantsandmed olid omavahel grupeeritud parema arusaamise nimel.

Kasutuskogemuse personaalsemaks muutmise eesmärgil oli kirja pandud nõue, et investor saaks luua endale konto. Funktsionaalsus sai täidetud – investor saab luua endale konto, millele on võimalik sisse logida kasutades kasutajanime ja parooli. Rakenduse loomisel kasutati Spring Security-t ja autentimiseks JWT tehnoloogiat. Sisselogimine annab ligipääsu ainult registreeritud kasutajale mõeldud funktsionaalsustele nagu isikliku aktsiate jälgimisnimekirja koostamine ning börsifirmade kõrvuti võrdlemine.

Viimaseks kirja pandud funktsionaalseks nõudeks oli investori võimalus luua endale isiklik aktsiate jälgimisnimekiri. See nõue sai ka täidetud. *ProfileComponent*-i alla loodi *tab*, mille sees on võimalik koostada jälgimisnimekirja. Praegune rakendus võimaldab nimekirja kuvamist, sinna uute lisamist ja olemasolevate kirjete kustutamist.

4.2.2 Mittefunktsionaalsed nõuded

Üheks esmaseks mittefunktsionaalseks nõudeks on veebirakenduse kasutatavus tähtsaimates veebibrauserites. Google Chrome ja Mozilla Firefox brauserid on Eestis kasutatavuselt vastavalt esimesel ja kolmandal kohal [45]. Rakenduse korrektset tööd kontrolliti kirjutamise hetkel brauserite uuemates versioonides milleks olid Google Chrome versioon 87.0.4280.88 ja Mozilla Firefox versioon 84.0.1.

Rakendus peaks olema piisavalt kiire, et investoril oleks seda mugav kasutada. Andmepäringu kiiruse kohta oli esitatud nõue, et vastus peab tulema kolme sekundiga. Kasutajaliidese laadimiskiiruse mõõtmiseks kasutati *Page load time plugin*-it [46], mis mõõtis lehe laadimiskiirust ja esitas seda brauseri tööriistaribal mugavalt arendajale. Mõõtmistulemused jäid alla sekundi ning selle alusel sai nõue täidetud.

Käesoleva arendusetapi käigus lisandus kasutajate registreerimise funktsionaalsus, millega muutus oluliseks nõue tagada rakenduse turvalisus. See nõue ei ole aja puudusel piisavalt täidetud. JWT-s sisalduvat kasutaja infot on kerge dekodeerida, siis kasutaja andmete kaitseks on *token*-isse salvestatud vaid minimaalne info – kasutajanimi ja *token*-i aegumistähtaeg. Lisaks on *token*-i genereerimisel kasutatav serveri võti piisavalt keerukas, et seda pole võimalik ära arvata.

Turvalisuse poole pealt on suurimaks puudujäägiks JWT salvestamine lokaalsesse hoidlasse, kuna see teeb rakenduse haavatavaks XSS-rünnakutele (*Cross-Site Scripting*). Võimalikuks lahenduseks järgnevas arendustöös on *token*-i saatmine serveri poolt *httpOnly* küpsisena, mis blokeerib andmete lugemise JavaScriptiga. Küpsiste kasutamine võrreldes lokaalse hoidlaga muudab rakenduse rohkem haavatavaks CSRF-rünnakutele (*Cross-Site Request Forgery*) ning seetõttu peaks mõtlema ka CSRF-i lisakaitsele. Teine variant on salvestada kasutajaliidese poolt *token* brauseri mällu ning seejärel kasutada *refresh token*-it sessiooni uuendamiseks [47] [48].

Lisaks sellele kasutab praegune rakendus päringute tegemisel HTTP ühendust, kuid turvalisuse tõstmiseks tuleks hakata kasutama krüpteeritud HTTPS (*HyperText Transfer Protocol Secure*) päringuid.

Kasutatavuse seisukohalt oli nõudeks, et rakenduse funktsionaalsused ei oleks lõppkasutaja jaoks liiga keerulised. Töö autor on andnud endast parima, et kasutajaliidese elemendid oleksid võimalikult konkreetsed ja arusaadavad. Lisaks on paremaks arusaamiseks loodud erinevaid kasutajale kuvatavaid veateateid, mis annaksid piisava ettekujutuse oodatavast sisendist. Näiteks kasutaja registreerimisel kasutatakse sisendite valideerimist nii sõne pikkuse kui sobivate sümbolite osas. Kasutajaliidese keerukuste tõusmisel hilisemas arendusetappides tuleks seda nõuet sügavamalt valideerida.

Rakenduse arendamine toimub mitme tundegi poolt ning iga etapiga läheb süsteem keerulisemaks. Selle tõttu on arenduse poole pealt nõudeks, et süsteem oleks lihtsasti laiendatav uute funktsionaalsustega. Nõude täitmiseks üritab töö autor kasutada *clean code* põhimõtteid ning teha komponente väiksemateks osadeks, mis kokkuvõttes aitaks kaasa koodi hulga vähenemisele ning võimaldaks elemente taaskasutada.

4.3 Arhitektuuri analüüs

Töö esmaversioon jaotas arhitektuuri tagarakenduseks ja kasutajaliideseks. Selline eraldamine soodustab projektide omavahelist nõrka seotust, mis kokkuvõttes muudab rakenduse kergemini hallatavaks ja testitavaks. Edasiarendus jätkas valitud arhitektuuriga ning lisatud komponendid (Pythoni *scraper* ja RabbitMQ sõnumiedastaja) aitavad edaspidist eraldatust toetada. *Scraper* tööriist tegeleb andmete korje ja puhastamisega, mis erineb äriloogika poolest teistest projekti osadest. Seetõttu tundus mõistlik arendada

scraper kui eraldiseisev projekt. Praeguses rakenduse versioonis on kalkulaatori funktsionaalsus üks osa Java tagarakendusest, kuid kalkulatsioonide mahu suurenemise korral võiks ka selle funktsionaalsuse tõsta eraldi projekti või ühendada andmete kraapimise projektiga. Sel juhul oleks kõik andmete töötlusega seonduv loogika koos ühes arhitektuurilises üksuses.

Rakenduse järgnevates arendusetappides reaalandmestiku osatähtsus kasvab. Selle tõttu on oluline, et projekti osad oleks suureneva andmehulgaga tegelemiseks kergemini laiendatavad. RabbitMQ sõnumiedastaja lisati projekti sellel eesmärgil, et töödeldava andmehulga tõusul oleks lihtne viis edastada infot ühest komponendist teise. Samuti vähendab edastaja kasutuselevõtt andmete kaotsiminekut ning kiirendab info üle andmist asünkroonse ühenduse kaudu. RabbitMQ võimaldab sõnumijärjekordade hulka ja struktuuri laiendada, mis kataks ära hilisemad suurema andmehulga töötlemisega tulevad tarkvara nõuded.

4.4 Disaini analüüs

Rakenduse API disainis jätkati REST arhitektuurilist stiili kasutamist. Võrreldes REST-i mõne teise veebiteenuse API-ga, näiteks SOAP-iga (*Simple Object Access Protocol*), siis on teda lihtsam rakendada, formaate andmete saatmiseks on rohkem (SOAP toetab vaid XML formaati) ning REST kasutab vähem ressursse ja on kiirem [49]. Arendusega tulid tagarakendusele juurde kaks *controller* klassi, samuti lisandusid POST ja DELETE päringutüübid. Päringute hulga suurenemisel ja keerukuse kasvamisel peaks järgnevatel arendusetappidel mõtlema kuidas neid paremini grupeerida ja milline oleks optimaalseim URI tee. Töö autor andis parima, et URI-sid korrektselt nimetada, samas võib päringu põhjus rakenduse vähemkogenud kasutajale tunduda arusaamatuna.

Kasutajate autoriseerimise ning autentimise funktsionaalsus muutis olulisemaks rakenduse turvalisuse. Hilisem rakenduse analüüs tõi välja kasutajaliidese disaini kitsaskohad, millele töö autor ei osanud algul tähelepanu pöörata. Üheks näiteks on JWT salvestamine lokaalsesse hoidlasse, mis on ühelt poolt lihtne, aga teisalt ebaturvaline lahendus. XSS-i rünnakutele on haavatavad ka erinevad kasutajaliidese *input* väljad, mida tuleks põhjalikult turvalisuse osas testida. Samuti ei ole sisendväljad piisavad turvalised ka näiteks DDoS (*Distributed Denial-of-Service*) rünnakute tõrjumiseks. Turvalisuse teema on pikk ning nõuaks põhjalikumat analüüsi.

Kasutajaliidese disaini osa pealt said mõned suuremad komponendid tükeldatud väiksemateks osadeks ning samuti üritati koodi võimalikult lihtsustada. Lisaks proovis töö autor koodi võimalikult palju taaskasutada ja struktureerida, näiteks toodi *axios* teeki kasutavad meetodid eraldi klassi. Kuna varasem kogemus React-iga puudus, siis kindlasti ei ole lahendus lõplik ja kohti parandamiseks leidub.

4.5 Töö edasiarenduse võimalused

Töö tulemusena valmis rakenduse demoversioon uute funktsionaalsustega, kus lisaks on *mock*-andmed osaliselt asendatud reaalandmetega. Rakenduse kasutajakogemus on muudetud personaalsemaks registreerimise ja kasutajakonto lisamisega. Konto võimaldab teha mitteavalikke operatsioone nagu aktsiate jälgimisnimekirja koostamine ning kahe ettevõtte üks-ühele võrdlemine.

Veebirakenduse väärtuse tõstmiseks on edasiarenduse seisukohalt kõige olulisem kvaliteetsete andmete osakaalu suurendamine. Hetkel kraabitakse Nasdaq Baltic kodulehelt aktsiate reaalhindu ning arvutatakse nende baasilt kahe finantssuhtarvu väärtused. Võimalusi rakenduse edasiarendamiseks on siinkohal mitmeid.

Üheks võimaluseks on Python-i *scraper* tööriista funktsionaalsuse laiendamine. Kõiki andmeid, mis on veebilehel avalikult saadaval, on võimalik kraapida. Selle alusel võiks peale aktsiahinna koguda ka hinna liikumisgraafikuid päeva, nädala või pikema ajaperioodi vältel. Vastavate andmetega saaks laiendada kasutajaliidese *CompanyInfo* vaadet, kus kuvatakse börsiettevõtte detailandmeid. Samuti võiks kaaluda kraapimist ka lahendusena aktsiate ajaloolise info korjamiseks nii Nasdaq Baltic lehelt kui ka sinna üles laetud *pdf* failidest (näiteks majandusaasta aruanded). Milliseid ajaloolisi andmeid on kõige mõistlikum kraapida võiks olla kooskõlastatud juhendajaga, kuna andmete töötlemisega tegelevad ka majandusteaduskonna tudengid.

Teine võimalus kvaliteetsete andmete kasutuselevõtu tõstmiseks on kalkulaatori funktsionaalsuse edasiarendus. Praegu on arvutuste tarbeks loodud Java tagarakenduses eraldi klassid, mis tegelevad kahe suhtarvu arvutamisega. Finantssuhtarve on aga palju rohkem. Lisaks teostavad arvutused palju andmebaasioperatsioone ning jõudluse parandamiseks ja sõltuvuse vähendamiseks võib kaaluda kalkulaatori tõstmist

eraldiseisvasse rakendusse. Eraldi rakendus viiks läbi kõik vajalikud kalkulatsioonid ning saadaks valmis andmed RabbitMQ sõnumiedastaja kaudu Java rakendusse.

Praeguses rakenduses hoitakse finantsandmeid kolmes andmebaasitabelis. Ajalooliste andmete kasutuselevõtuga tabelite arv kindlasti suureneb ja seetõttu tuleks mõelda andmebaasistruktuuri peale, mis võimaldaks hiljem andmeid kerge vaevaga uuendada.

Rakenduse kasutajakonto võimaldab sisselogimist kasutajanime ja parooli kaudu. Käesolevas arenduses ei ole kaetud parooli uuendamise stsenaariumi ehk juhul kui kasutaja unustab oma parooli, siis ei ole võimalik kontole enam ligipääsu saada. Edasiarenduses võib kaaluda emaili aadressi lisamist registreerimisprotsessi ning konto taastamise võimalust selle kaudu. Lisaks tuleks rakendus muuta turvalisemaks erinevate veebirünnakute vastu ja kasutaja andmete kaitseks peaks kommunikatsiooni üle viima HTTP-lt krüpteeritud HTTPS ühendusele.

Veebirakendust võib edasi arendada eri külje pealt ning töö autor tooks välja ka mõned madalama prioriteediga võimalikud funktsionaalsused. Praegune kasutajaliides on inglise keeles, kuna esmane sihtgrupp on eesti investorid, siis võib mõelda ka eestikeelse kasutajaliidese peale. Lisaks on kõik loodud kasutajakontod „*USER*“ rolliga, tulevikus võib kaaluda näiteks laiendatud õigustega administraatori rolli lisamist. *Mock*-andmete osas on hetkel kaetud vaid Tallinna börsil tegutsevad ettevõtted. Kui tulevikus on andmete kvaliteet saavutanud piisavalt hea taseme, siis võib hakata lisama ettevõtteid ka ülejäänud Balti börsilt.

5 Kokkuvõte

Töö käigus arendati edasi Tallinna börsi aktsiate finantsandmete veebirakenduse esmast versiooni. Lõputöö põhieesmärk oli lisada rakendusele olulisi funktsionaalsusi ja alustada *mock*-andmete asendamist reaalsete andmetega.

Edasiarendus toetab kaugemat eesmärki, milleks on luua Tallinna börsil tegutsevatele investoritele veebirakendus, kus on võimalik saada kiire ülevaade ettevõtte finantsseisust ning samas oleks ka hea ligipääs kvaliteetsetele ajaloolistele andmetele. Aktsiate *screener* tööriistu on Internetis mitmeid, kuid nendes käsitletav andmestik on kohaliku investori jaoks ebapiisav.

Arenduse tulemusena valmis lisandunud funktsionaalsustega demoversioon, mis ei ole veel produktsioonivalmis kuna kasutab *mock*-andmestikku. Uute arenduste tulemusel on hakatud rakendusse järk-järgult sisse tooma reaalandmeid. Lisaks on kasutajakogemus muutunud personaalsemaks kasutajakonto loomise võimalusega. Registreeritud kasutajal on võimalus luua isiklikke aktsiate jälgimisnimekirju ning teostada finantsandmete kõrvuti võrdlemist.

Töö järgnevates etappides oleks vaja hakata koguma suuremal hulgal reaalandmeid, parandada rakenduse turvalisust ning leida viis kasutajale ajalooliste andmete kuvamiseks. Suure tõenäosusega jätkub veebirakenduse arendus veel pikemat aega erinevate arendajate osavõtul.

Kasutatud kirjandus

[1] Jagor, M. Veebirakendus Tallinna börsiettevõtete finantsandmete kajastamiseks : bakalaureusetöö. Tallinna Tehnikaülikool, Tallinn 2020.

[2] Nasdaq Balti börsid [WWW] https://nasdaqbaltic.com/et/meist/nasdaq-balti-borsid/ (10.12.2020).

[3] TradingView Features. [WWW] https://www.tradingview.com/features/ (10.12.2020).

[4] IntelliJ IDEA Features. [WWW] https://www.jetbrains.com/idea/features/ (10.12.2020).

[5] React. [WWW] https://reactjs.org/ (11.12.2020).

[6] PrimeReact. [WWW] https://www.primefaces.org/primereact/showcase/#/ (11.12.2020).

[7] Fundamentals of Software Architecture. M. Richards, N. Ford. O'Reilly Media Inc., 2020. [Online] (12.12.2021).

[8] Nasdaq Data on Demand. [WWW] https://dataondemand.nasdaq.com/docs/index.html#introduction (12.12.2020).

[9] Python Web Scraping Cookbook. M. Heydt. Packt Publishing, 2018. [Online] Oreilly (13.12.2020).

[10] REST API Design Rulebook. M. Masse. O'Reilly Media, Inc., 2011. [Online] Oreilly (16.12.2020).

[11] Spring Quick Reference Guide: A Pocket Handbook for Spring Framework, Spring Boot, and More. A.L. Davis. Apress, 2020. [Online] Oreilly (16.12.2020).

[12] Learning React, 2nd Edition. E. Porcello, A.Banks. O'Reilly Media, Inc., 2020. [Online] Oreilly (16.12.2020).

[13] Handling Events. [WWW] https://reactjs.org/docs/handling-events.html (16.12.2020).

[14] Conditional Rendering. [WWW] https://reactjs.org/docs/conditional-rendering.html (16.12.2020).

[15] Refs and the DOM. [WWW] https://reactjs.org/docs/refs-and-the-dom.html (16.12.2020).

[16] Clean Code, A Handbook of Agile Software Craftsmanship. C. R. Martin. USA : Pearson Education, Inc., 2009. [Online] https://enos.itcollege.ee/~jpoial/oop/naited/Clean%20Code.pdf (16.12.2020).

[17] PMD. [WWW] https://pmd.github.io/latest/index.html (20.12.2020).

[18] SonarLint. [WWW] https://www.sonarlint.org/ (20.12.2020).

[19] Httplib2. [WWW] https://pypi.org/project/httplib2/ (05.12.2020).

[20] Beautiful Soup Documentation. [WWW] https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (05.12.2020).

[21] Introduction to Pika. [WWW] https://pika.readthedocs.io/en/stable/intro.html (05.12.2020).

[22] RabbitMQ Features. [WWW] https://www.rabbitmq.com/#features (16.12.2020).

[23] Microsoft Docs New-ScheduledTaskTrigger. [WWW] https://docs.microsoft.com/en-us/powershell/module/scheduledtasks/new-scheduledtasktrigger?view=win10-ps (17.12.2020).

[24] What is PowerShell? [WWW] https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.1 (17.12.2020).

[25] RabbitMQ Java introduction. [WWW] https://www.rabbitmq.com/tutorials/tutorial-one-java.html (17.12.2020).

[26] Class ObjectMapper. [WWW] https://fasterxml.github.io/jackson-databind/javadoc/2.7/com/fasterxml/jackson/databind/ObjectMapper.html (17.12.2020).

[27] Price-to-Earnings Ratio – P/E Ratio. [WWW] https://www.investopedia.com/terms/p/price-earningsratio.asp (18.12.2020).

[28] Dividend Yield. [WWW] https://www.investopedia.com/terms/d/dividendyield.asp (19.12.2020).

[29] Introduction to JSON Web Tokens. [WWW] https://jwt.io/introduction/ (10.12.2020).

[30] Spring Boot JWT Authentication example with Spring Security & Spring Data JPA. [WWW] https://github.com/bezkoder/spring-boot-spring-security-jwt-authentication (10.12.2020).

[31] InputText. [WWW] https://www.primefaces.org/primereact/showcase/#/inputtext (19.12.2020).

[32] Password. [WWW] https://www.primefaces.org/primereact/showcase/#/password (19.12.2020).

[33] Axios. [WWW] https://www.npmjs.com/package/axios (19.12.2020).

[34] Card. [WWW] https://www.primefaces.org/primereact/showcase/#/card (20.12.2020).

[35] TabView. [WWW] https://www.primefaces.org/primereact/showcase/#/tabview (20.12.2020).

[36] DataTable. [WWW] https://www.primefaces.org/primereact/showcase/#/datatable/basic (20.12.2020).

[37] AutoComplete. [WWW] https://www.primefaces.org/primereact/showcase/#/autocomplete (20.12.2020).

[38] Fieldset. [WWW] https://www.primefaces.org/primereact/showcase/#/fieldset (20.12.2020).

[39] How Linux Works. B. Ward. No Starch Press, 2014. [Online] Oreilly (16.12.2020).

[40] Kafka Introduction. [WWW] https://kafka.apache.org/intro (21.12.2020).

[41] Introduction to Redis. [WWW] https://redis.io/topics/introduction (20.12.2020).

[42] RabbitMQ Essentials – Second Edition. D. Dossot, L. Johansson. Packt Publishing, 2020. [Online] Oreilly (21.12.2020).

[43] Hands-On Spring Security 5 for Reactive Applications. J. Tomcy. Packt Publishing, 2018. [Online] Oreilly (21.12.2020).

[44] Cookies vs Tokens: The Definitive Guide. A.Kukic. [WWW] https://dzone.com/articles/cookies-vs-tokens-the-definitive-guide (21.12.2020).

[45] Browser Market Share Estonia. [WWW] https://gs.statcounter.com/browser-market-share/all/estonia (21.12.2020).

[46] Page load time. [WWW] https://chrome.google.com/webstore/detail/page-load-time/fploionmjgeclbkemipmkogoaohcdbig?hl=en (21.12.2020).

[47] Web Application Security. A. Hoffman. O'Reilly Media, Inc., 2020. [Online] Oreilly (22.12.2020).

[48] React Authentication: How to Store JWT in a Cookie. R. Chenkie. [WWW] https://medium.com/@ryanchenkie_40935/react-authentication-how-to-store-jwt-in-a-cookie-346519310e81 (22.12.2020).

[49] Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. R. Daigneau. Addison-Wesley Professional, 2011. [Online] Oreilly (22.12.2020).

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

Mina, Katre-Helena Käppa

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Tallinna Börsi Ettevõtete Finantsandmete Veebirakendus“, 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. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

04.01.2020

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 – Projekti koodi ligipääsulingid

Projekti *scraper* tööriist: https://github.com/katrehel/web-scraper

Projekti tagarakendus: https://github.com/katrehel/stock-screener-application

Projekti kasutajaliides: https://github.com/katrehel/stock-screener-react-application

Lisa 3 – PowerShelli-i skript

```
$action = New-ScheduledTaskAction -Execute "PATH\scraperTool.exe"
$triggers = @( $(New-ScheduledTaskTrigger
      -Weekly -DaysOfWeek Monday, Tuesday, Wednesday, Thursday, Friday
      -At 10:00:00am))
$task = Register-ScheduledTask  -Action
$action -Trigger $triggers -TaskName "Scraper"
      -Description "Automated execution of ticker price scraper tool."
foreach ($t in $task.Triggers) {
    $t.Repetition.Duration = "PT6H"
    $t.Repetition.Interval = "PT1M"
}
$task | Set-ScheduledTask
```