TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kalle Pilli 206466IABB
Rene Treier 206647IABB

Elektrienergia tarbimise ja tootmise analüüsi veebirakendus

Bakalaureusetöö

Juhendaja: Bahdan Yanovich
BSc

Tallinn 2023

Autorideklaratsioon

Kinnitame, et oleme 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.

Autorid: Kalle Pilli, Rene Treier

16.05.2023

Annotatsioon

Bakalaureusetöö eesmärk oli luua veebirakendus, mis võimaldaks analüüsida elektri tarbimist ja tootmist ning seeläbi tõsta rakenduse kasutaja teadlikkust tarbitavast ja toodetavast elektrist.

Et inimesed saaks teha teadlikumaid valikuid elektrienergia tarbimise kohta, on siiani küll loodud erinevaid vähese funktsionaalsusega kalkulaatoreid ja veebirakendusi, aga puudub lahendus, mis koondaks kokku erinevaid funktsionaalsusi ning annaks kasutajale võimaluse ka enda elektri tarbimise, tootmise ja lepingute infot salvestada.

Töö autorid lõid rakenduse, mis võimaldab ajalooliste andmete põhjal teha kuu- ja tunnipõhist elektritarbimise analüüsi, salvestada ja analüüsida kasutaja elektripakette ning elektri tootmist ja tarbimist. Rakendus erineb olemasolevatest lahendustest oma mitmekülgse funktsionaalsuse poolest ja võimaldab kasutajatel teha informeeritud otsuseid oma elektritarbimise kohta.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 57 leheküljel, 4 peatükki, 29 joonist, 3 tabelit.

Abstract

Web application for the analysis of electricity consumption and production

The aim of this bachelor's thesis was to create a web application that would enable the analysis of electricity consumption and production and increase the user's awareness of consumed and produced electricity.

Various calculators and web applications with limited functionality have been created to help people make more informed choices about electricity consumption. However, there is a lack of solutions that would consolidate different functionalities and provide the user with the ability to save information about their electricity consumption, production and packages.

The authors developed a web application that allows users to conduct monthly and hourly electricity consumption analysis based on historical data, save and analyze their electricity packages, consumption and production. The application stands out from existing solutions with its versatile functionality, enabling users to make informed decisions about their electricity consumption and production.

The thesis is in Estonian language and contains 57 pages of text, 4 chapters, 29 figures, 3 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface* ehk rakendusliides
API *endpoint*	Rakendusliidese otspunkt/ressurss
Backlog	Nimekiri töödest, mida on vaja teha
Blackbox testing	Musta kasti testimismeetod, kus testija kontrollib rakenduse funktsionaalsust teadmata sisemist arhitektuuri ja protsesse
CI/CD	*Continuous integration / Continuous deployment* ehk pidev integratsioon ja automaatne testimine ning juurutamine
CRUD	Lühend andmebaasi toimingutest (*Create, read, update, delete* – sisesta, loe, uuenda, kustuta)
Debug	Programmeerides vigu tuvastama, avastama või kõrvaldama
DTO	*Data Transfer Object* ehk andmeedastusobjekt
Extreme Programming	Eesti keeles „ekstreemne programmeerimine“ – Kent Becki poolt välja töötatud arendusmetoodika
Hard-coded	Realiseeritud tarkvaras mitte dünaamilisena
HTTP	*Hypertext Transfer Protocol* ehk hüperteksti edastusprotokoll
InMemory andmebaas	Lokaalse arvuti mälus jooksev andmebaas
Json Web Token	Digitaalselt allkirjastatud *Json* objekt, mis on kompaktne viis andmete turvaliseks edastamiseks
Pipeline	Järjestus automaatsetest toimingutest
REST	*Representational state transfer* ehk esitusoleku siire. Arhitektuuri stiil, mida kasutatakse veebiteenuste loomiseks kasutades HTTP protokolli
XP	*Extreme Programming* ehk ekstreemne programmeerimine

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

1.1 Üldine taust

Energiahindade kiire tõus ja energiakriis on viimastel aastatel olnud aktuaalne teema nii Eestis kui ka ülejäänud maailmas [1]. Elektri börsihinnad on järsult tõusnud ja Eesti inimeste energiatarbimiskulud on kasvanud. Meedias on üha rohkem infot elektriteenuste ja lepingute kohta ning elektriga seotud uudiste arv on näiteks Eesti Rahvusringhäälingu uudisteportaalis kolme aastaga ligikaudu neljakordistunud [2] [3].

Elektriga seotud teemasid arutatakse riiklikul tasemel ning astutud on samme, et elektrienergia kõrgete hindade mõju vähendada. Näiteks on Eesti Energia kohustatud müüma elektrit universaalteenusena kodutarbijatele, osale äritarbijatest ja kohalikele omavalitsustele alates 2022. aastast [4]. Kuigi 2023. aastal on Nordpool börsi elektrihinnad võrreldes eelmise aastaga langenud, tuleb siiski tarbijatele kasuks analüüsida, milline elektrileping ja tarbimisharjumused oleks kõige optimaalsem.

Tarbimise ja tootmise analüüsimiseks on erinevaid kalkulaatoreid ja tarbimise ajastamist lihtsustavaid rakendusi, kuid nende väheste funktsionaalsuste tõttu on vaja ka alternatiive, et inimesed oleks teadlikumad võimalustest kulusid vähendada ja endale sobivaim elektripakett valida.

1.2 Probleem

Erinevatel elektriteenuste pakkujatel on olemas oma äpid ja kodulehed, kus on võimalik näha oma elektrienergia tarbimis- ja tootmisandmeid, kuid sellega tihti funktsionaalsused piirduvad. Samuti on rakendusi, mis aitavad kaasa igapäevasele elektritarbimise juhtimisele, kuid puudub võimekus põhjalikumaks analüüsiks ning võimalus seda teha ühes kohas.

1.3 Eesmärk

Bakalaureusetöö eesmärgiks on luua veebirakendus, mis tõstab inimese teadlikkust enda tarbitavast ja toodetavast elektrist.

Andmeid analüüsides suudab rakendus aidata inimesel teha otsuseid, mis aitavad optimeerida tema kodust elektritarbimist ning valida tema jaoks sobivaid elektrilepinguid. Rakenduses on võimalik võrrelda erinevaid elektrilepinguid vastavalt oma ajaloolisele tarbimisele, et valida enda jaoks parim lahendus. Samuti saab leida potentsiaalse tootmisseadmete tasuvusaja.

Eesmärk on luua rakendus, millel oleks järgnevad funktsionaalsused:

- elektri tarbimise analüüs ajalooliste andmete põhjal
- ajastatud tarbimissoovituste andmine tuleviku elektrihindade põhjal
- oma elektripakettide ja seadmete lisamine, mille põhjal antakse teada, kui palju oleks olnud võimalik säästa
- elektri tootmis- ja salvestusseadmete tasuvusaja arvutamine
- elektri tulevikuhindade ja ilmateate API andmete (tuulegeneraatori või päikesepaneelide olemasolul) abil potentsiaalse tarbimise ja/või tootmise juhtimine: API *endpoint*, mida saaks potentsiaalselt mõni teine rakendus automatiseerimiseks kasutada

Eesmärgi saavutamiseks kasutasid autorid agiilseid arenduspraktikaid, et luua antud funktsionaalsusi pakkuv rakendus. Sisendandmetena kasutati autorite isiklikke elektritarbimise andmeid, kuid samuti Eleringi pakutavate keskmiste kasutajate infot.

1.4 Töö edasine struktuur

Töö koosneb seitsmest osast. Sissejuhatuses kirjeldatakse bakalaureusetöö probleemi ning eesmärki. Meetodika osa sisaldab töö jaoks valitud tööriistu ning arenduse käigus jälgitud protsesse. Tulemuste all on projekti käigus valminud rakenduse kasutajalood ja erinevate komponentide arhitektuur. Välja on toodud arenduse käigus järgitud koodiarendamise mustrid ning testimise meetodid. Analüüsi osas võrreldakse valminud

rakendust alternatiivsete lahendustega. Analüüsitakse tehtud tööd ning antakse hinnangud projekti erinevatele osadele. Kokkuvõttes võetakse töö tulemused ning järeldused kokku. Kasutatud kirjanduses on loetletud töös kasutatud allikad. Lisad sisaldavad lihtlitsentsi autorite eneseanalüüsi ja koodilõike.

2 Metoodika

Selles peatükis kirjeldatakse, missuguseid tööriistu meeskond bakalaureusetöö raames valminud tarkvara väljatöötamisel kasutas ning milline oli arendusprotsessi metoodika.

2.1 Objekti detailne kirjeldus

Bakalaureusetöö ja lõpuprojekti raames arendasid autorid rakenduse elektri tarbimise ja tootmise optimeerimiseks ning analüüsiks. Rakenduse nimeks sai ElectricityStat. Rakenduse nõuded ning vajalik funktsionaalsus on välja töötatud autorite poolt järgides peamiselt isiklikke vajadusi ning soove, kuid samas mõeldes potentsiaalsete klientide ja kasutajate eelistuste peale.

Loodud lahendus hõlmab endas tagarakendust, kasutajaliidest ning andmebaasi, mis on paigutatud konteineriseeritud keskkonda kasutades Dockerit. Tagarakendus on arendatud C# programmeerimiskeeles ning .NET raamistikus, kasutajaliides aga TypeScript programmeerimiskeeles ja Vue.js raamistikus ning andmebaasina kasutati PostgreSQL andmebaasi. Programmeerimiskeeled ning tarkvaralised tehnoloogiad valiti vastavalt projekti vajadustele ning autorite teadmistele ja kompetentsile.

Valitud tarkvarad ja tööriistad aitasid meeskonnal omada projektist head ülevaadet, võimaldades neil rakendust tõhusalt arendada ja juhtida, teha aktiivselt koostööd ning hallata projekti ajakava ning eesmärke.

2.2 Tööriistad

2.2.1 Raamistikud ja programmeerimiskeeled

Autorid kasutasid tarkvaraarenduseks Microsoft Visual Studio Code'i integreeritud arenduskeskkonda. Tagarakendus ja kasutajaliides loodi samas projektis ja keskkonnas, et lihtsustada rakenduse arendusprotsessi ning käivitamist lokaalses arvutis. Kasutajaliides arendati Vue.js raamistikus ja TypeScript programmeerimiskeeles, samas kui tagarakendus loodi .NET raamistikus ja C# programmeerimiskeeles.

2.2.2 Tehnoloogiad

Tagarakenduse ja kasutajaliidese vaheliseks sõnumivahetuseks kasutati HTTP protokolli ja REST stiili põhiseid liideseid. Liideste dokumentatsiooni loomiseks kasutati OpenApi ja SwaggerUI tööriistu. Algfaasis kasutati andmete hoiustamiseks *InMemory* andmebaasi, kuid rakenduse kasvades võeti kasutusele PostgreSQL andmebaas koos pgAdmin andmebaasihaldussüsteemiga, et lihtsustada rakenduse skaleerimist ja käivitamist erinevates süsteemides.

2.2.3 Versioonihaldus

Versioonihalduseks kasutati GitLab keskkonda, kus hoiti projekti repositooriumit ning kasutati CI/CD funktsionaalsust, et automatiseerida tarkvara testimist ja koodikontrolli enne uue versiooni ühendamist põhiharuga. Selle jaoks loodi Gitlab keskkonda *pipeline*, mis koosnes iga põhiharusse ühendamise puhul *build*, *unit-test* ja *test* etappidest, mis vastavalt ehitasid rakenduse konteinerkeskkonnas, jooksutasid ühikteste ning Postmani integratsiooniteste. *Pipeline*'i ebaõnnestumise korral olid autorid seadistanud GitLabi nii, et põhiharusse ei saanud ühendada planeeritud muudatust, vaid tuli parandada, kuni see õnnestub. GitLab'i projektihalduse abil hallati ka sprinte ja planeeritud ülesandeid.

2.2.4 Testimine ja tulemuste valideerimine

Projekti käigus kasutati Microsoft Office Excel programmi, et teostada esmaseid andmeanalüüse, visualiseerida rakenduse loogikat ja valideerida ning testida rakenduse vastuseid. Lisaks manuaalsele testimisele Exceliga, kasutas meeskond testimiseks programmi Postman, et luua API *endpoint* teste. Projekti loodi ka ühiktestide projekt, et kindlustada arendusprotsessi käigus tehtud muudatuste puhul olemasoleva koodi tööle jäämine.

2.2.5 Ajalogimine

Aja haldamiseks ja projekti panuse hindamiseks kasutas meeskond veebirakendust Toggl Track.

2.2.6 Suhtlus

Suhtlemiseks kasutati peamiselt kahte kanalit: kiireks suhtluseks Facebook Messengeri ning koos programmeerimiseks ja koosolekute pidamiseks, nii omavahel kui ka juhendajaga, kasutati Microsoft Teamsi.

2.3 Tööprotsessi kirjeldus

Projekti arendamine kaheliikmelises meeskonnas oli autorite jaoks väljakutse, kuna tavaliselt on arendusmeeskonnad suuremad. Näiteks Scrumi arendusmetoodikat kasutavad meeskonnad koosnevad tihti 5-10 liikmest [5]. Siiski soovisid autorid rakendada Scrumile iseäralikke printsiipe ja meetodeid, et arendusvoog oleks võimalikult agiilne ja paindlik. Meeskonna väiksuse tõttu tuli muuta ja kohandada Scrumi protsesse nii, et need vastaksid kaheliikmelise meeskonna vajadustele ja võimalustele. Seega otsustati integreerida Kent Becki poolt väljatöötatud arendusmetoodika Extreme Programming'u (edaspidi XP) mõningaid põhimõtteid [6].

2.3.1 Skoop

Scrumi ja XP põhimõtted on suures plaanis sarnased. Väärtustatakse paindlikkust muuta arenduse kurssi ning nõudeid ega kohkuta tagasi uute ideede ees. Tihtipeale ei ole ka lõpuni selge, milline valmis toode välja peab nägema ning missuguseid funktsionaalsuseid omama. XP peab kõige tähtsamaks muutujaks skoopi: mida väiksem skoop, seda parema kvaliteediga saab konkreetses ajaraamistikus tarkvara luua. Autorid lähenesid arendusele algusest peale seda silmas pidades. Otsustati arendada põhimõttega, et tööd oleksid üksteisest võimalikult sõltumatud ning kirjeldatud võimalikult väikeste juppidena [5] [6].

2.3.2 Rollid

Meeskonnaliikmed võtsid vastu otsuse mitte jagada rolle nii nagu need on traditsioonilises Scrumis või XP-s, vaid tegeleda otsustati kõikide rollidega [5] [6]. See otsus võimaldas autoritele suuremat paindlikkust ning kokkupuudet kõikide protsessidega ja aitas meeskonnal paremini mõista kogu arendusprotsessi ning tagada kvaliteetse tarkvara loomine. Meeskond haldas ühiselt projekti *backlog*'i, kus loodi ja kirjeldati pileteid/kasutajalugusid. Taoline lähenemine võimaldas meeskonnaliikmetel olla kursis töödega ja tagada, et kõik pileti üksikasjad ja nõuded oleksid piisaval määral kirjeldatud, et tagada õigeaegne ja edukas lõpuleviimine.

2.3.3 Töövoog

Projekti arenduse töövoog loodi eelkõige Scrumi protsesse [5] järgides ning nägi välja selline:

1. ***Sprint planning***'u ehk sprindi planeerimise käigus tehti korra nädalas Microsoft Teamsi vahendusel koosolek, kus arutati, kuhu järgneva sprindi jooksul soovitakse jõuda ning millised tööd ja mis järjekorras on vaja teha. Loodi piletid, kus oli tööd detailselt kirjeldatud ning seejärel jagati meeskonnaliikmete vahel ülesanded ära. Tööde jagamisel püüti vältida koodi konflikte põhiharus, jagades tööd nii, et need ei segaks teise meeskonnaliikme arendust ja liitmist põhiharuga.

2. **Arenduse** käigus alustasid meeskonnaliikmed ülesannete täitmist ja arendust vastavalt piletitel kirjeldatud nõuetele. Kuna piletite sisu oli piisavalt detailselt kirja pandud ning omavahel selgeks räägitud, ei pidanud autorid igapäevaselt püstijalakoosolekuid (ingl. *Standup*) pidama, mis on üldiselt Scrumi üheks alustalaks. Tavaliselt räägiti enda probleemidest ja saavutustest Facebook Messengeri teel, kuid kord nädalas peeti ka püstijalakoosolekuid.

    a. **Testimine –** arenduse käigus kirjutati ühik- ja API *endpoint*teste, et saadud tulemusi kontrollida ja veenduda rakenduse ning API töös. Testandmeid analüüsiti ka Microsoft Excelis, et visualiseerida rakenduse äriloogikat kui ka valideerida tulemusi.

3. ***Code review*** ehk koodi ülevaatuse osa oli väga tähtis, et tagada usaldusväärne, hallatav ja kvaliteetne kood. Kood vaadati üle igal võimalusel, kui teine meeskonnaliige oli oma pileti valmis saanud. Ka see suhtlus toimus suures osas Facebook Messengeri vahendusel, kuid suuremat ja rohkem seletust nõudvate arenduste puhul tehti Microsoft Teamsi vahendusel koosolek. Kui mõlemad meeskonnaliikmed olid loodud koodiga rahul, liideti haru põhiharuga. Vajadusel tehti muudatusi ning sellele järgnes uuesti koodi ülevaatus.

4. ***Retrospective*** ehk sprindile tagasivaatus on Scrumi arendusmeetodi oluline osa, kus meeskond hindab oma viimase sprindi tulemusi ja koostööd ning arutab läbi, mida saaks järgmises sprindis paremini teha. Protsessi käigus võeti kokku, millised eesmärgid saavutati ning missuguseid takistusi tuli ületada. Lisaks hinnati ka üldist meeskonnatööd ja meeskonna dünaamikat. Tavaliselt järgnes sprindi tagasivaatusele ka järgmise sprindi planeerimine, sest sellel hetkel olid meeskonnaliikmetel kitsaskohad kui ka õnnestumised kõige paremini meeles. See

võimaldas meeskonnal paremini planeerida järgmist sprinti ning muuta kogu protsess veelgi efektiivsemaks.

2.3.4 Strateegia

Autorid otsustasid arenduse strateegiana rakendada XP metoodikat [6], mis keskendub kiirele ja kvaliteetsele tarkvaraarendusele, kasutades järgnevaid põhimõtteid:

1. **Pidev tarkvara integreerimine** - uus kood liideti põhiharuga võimalikult kiiresti, et vältida koodikonfliktide kuhjumist hilisemaks lahendamiseks. Autorid olid motiveeritud üksteise koodi kiirelt üle vaatama, et teha vajalikud parandused ja seejärel liita uus arendus põhiharuga. See võimaldas meeskonnal saavutada paremat tarkvara kvaliteeti ning seejuures vähendada ka arendusele kuluvat aega.

2. **Kood on kollektiivne omand** - meeskonnaliikmed võisid koodi refaktoreerida igal ajal, kui see aitas süsteemi lihtsamaks ning abstraktsemaks muuta. See oli oluline, sest nii õppis meeskond tundma kõiki süsteemi osasid ning koodi dublikatsiooni elimineerimise/vähendamise abil parandati tarkvara kvaliteeti. Selle tulemusena oli loodav projekt paremini hooldatav ja skaleeritav.

3. **Paaris programmeerimine** - koos teise meeskonnaliikmega ülesannete lahendamine muutis kohati arenduse oluliselt kiiremaks ja tõstis tarkvara kvaliteeti. Autorid rakendasid seda olulist XP põhimõtet üsna tihti, sest arenduse käigus tekkis mitmeid tehnilisi probleeme, millele jõuti koos programmeerides ja uurides oluliselt kiiremini jälile.

3 Tulemused

3.1 Alternatiivsed lahendused

Projekti nõuete väljatöötamiseks tutvusid autorid kõigepealt olemasolevate alternatiivsete lahendustega, mille eesmärgiks oli kaardistada funktsioone, millest taolise rakenduse kasutajad võiksid puudust tunda. Autorid valisid funktsionaalsuste võrdlemiseks ja hiljem analüüsimiseks välja viis alternatiivset veebirakendust ning kolm Eesti elektrimüüja mobiilirakendust.

Selleks, et autoritel tekiks laiem pilt turul olemasolevate lahenduste funktsionaalsustest ja võimalustest, tutvuti lisaks muude äppidega, mis aitavad inimestel energiakuludelt raha säästa. Samuti andis rohkemate lahenduste uurimine autoritele ideid enda veebirakenduse edasiarendamiseks.

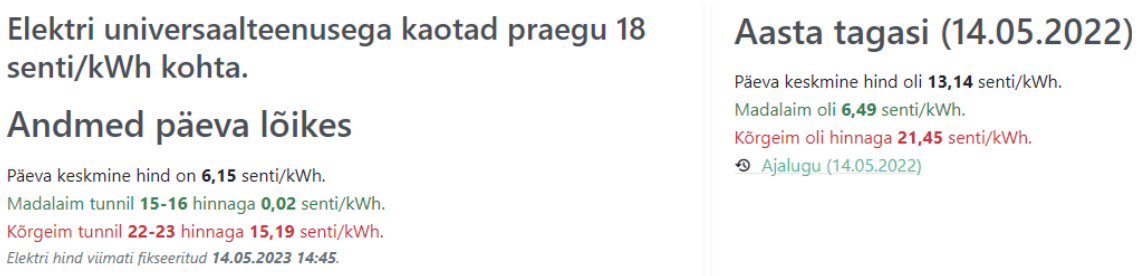
3.1.1 Elektrikell (elektrikell.ee)

Elektrikell võimaldab kasutajatel jälgida Nordpool börsi elektrihindade graafikut. Kasutajale kuvatakse hinnad sentides kilovatt-tunni kohta. Rakendus võimaldab kasutajal valida, mitu tundi soovitakse elektrit tarbida ning süsteem näitab, mis ajal on seda kõige optimaalsem teha. Sama on kasutajal võimalik teha üsna lihtsalt jälgides visuaalselt graafikut. Kiireks ja lihtsalt mõistetavaks indikaatoriks elektrihinna kohta on kuvatav tekst: “Elektri hind hetkel on madal/kõrge.” Rakendus sobib elektrihinna jälgimiseks ning võimalik on läbi viia ka kerge analüüs, millal on optimaalseim tarbida, kuid sellega suures plaanis funktsionaalsused piirduvad [7] (Joonis 1).

Joonis 1. Elektrikell

3.1.2 MacDronic (home.macdronic.com)

Pealtnäha Elektrikellaga üsna sarnane rakendus, kus kuvatakse elektrihind diagrammil sentides kilovatt-tunni kohta. Intuitiivselt kiirelt mõistetavaks indikaatoriks elektrihinna taseme kohta on eri värvi tulbad elektrihinna tulpdiagrammil: roheline - madal hind, oranž - keskmine hind, punane - kõrge hind. Lisaks kuvatakse ka info elektrihinna kohta (päeva keskmine hind, madalaim tunnihind, kõrgeim tunnihind) antud päeval ning ka samal päeval aasta tagasi (Joonis 2). Kasuliku funktsionaalsusena on välja toodud, kui palju kaotaks või võidaks kasutaja, kui ta kasutaks börsihinna asemel universaalteenust [8].



Joonis 2. MacDronic elektrihinna info

3.1.3 Elektrikulu kalkulaator (elekter.pere-eelarve.eu)

Elektrikulu kalkulaatoris on samuti võimalik jälgida börsihinda graafikul sentides kilovatt-tunni kohta. Lisaks on graafik, et jälgida ka viimase 12 kuu keskmisi elektrihindu.

Rakendus võimaldab lisaks elektrihinna jälgimisele viia läbi ka seadmete elektritarbimise analüüsi. Sisestama peab seadme võimsuse, kasutusaja päevas ja hinna parameetrid (Joonis 3). Vajutades nupule “Arvuta”, kuvatakse info selle kohta, kui palju antud seadme tarbimise juures kasutaja elektrile raha kulutab, sealjuures on andmed päeva, kuu kui ka aasta maksumuse kohta. Lisaks tuuakse eraldi välja elektrienergia hind, võrguteenuse hind, taastuvenergia tasu ja tasutav elektriaktsiis [9].

Joonis 3. Elektrikulu kalkulaator avaleht

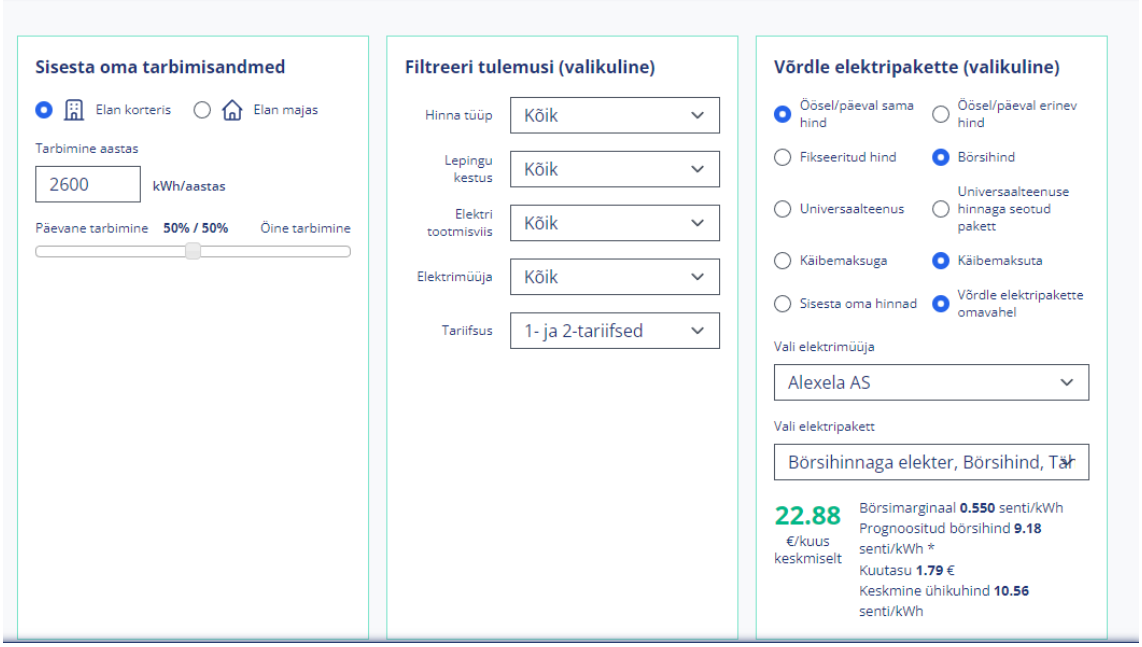
3.1.4 Elektritarbimise kalkulaator (kalkulaatorid.net)

Elektritarbimise kalkulaatori rakendus on mõningal määral sarnane eelnevaga, kuid analüüs on tehtud kasutajasõbralikumaks: lisada saab mitu seadet ning on ka võimalus lisada enim kasutatavad seadmed, nagu näiteks pesumasin, külmkapp, televiisor, kusjuures nende umbkaudsed võimsused on vaikimisi sisestatud (Joonis 4). Kasutaja peab lisama ka ajalise ööpäevase tarbimise ja eeldatava elektrihinna. Pärast parameetrite sisestamist arvutab rakendus elektri tarbimise ja selle maksumuse igal päeval, kuul ja ühes aastas [10].

Joonis 4. Seadme lisamise võimalus

3.1.5 Elektrihind (elektrihind.ee)

Rakendus on mõeldud võrdlemaks erinevaid elektripakette ning mugavalt on võimalik ka samalt lehelt valitud paketti vormistama minna (Joonis 5). Veebirakendus näitab sisestatud parameetrite põhjal kasutaja keskmist elektrikulu kuus. Sisestada saab enda tarbimisandmed: elukoht (korter/maja), tarbimine aastas ning valida saab ühe- ja kahetariifsete pakettide vahel. Võimalik on ka soovitatavaid pakette filtreerida hinnatüübi, lepingu kestuse, elektri tootmisviisi ja elektrimüüja järgi. Lisaks on võimalik elektripakette manuaalselt võrrelda. Sisestama peab elektripaketi hinnastamise tüübi, käibemaksuga/ilma käibemaksuta, elektrihinna ning kuutasu. Rakendus kuvab sisestatud parameetrite põhjal keskmise kasutaja eeldatava keskmise kuu elektrihinna [11].



Joonis 5. Elektrihind pakettide võrdlus

3.1.6 Eesti elektrimüüjate mobiilirakendused

Eesti elektrimüüjatest on mobiilirakendused Eesti Energial, Alexelal ja 220 Energial:

- **Eesti Energia** – rakendus võimaldab vaadata elektri börsihinda graafikul, kasutaja tarbimisajalugu graafikul koos börsihinnaga ja võrrelda graafikul kahe erineva päeva tarbimist tunni kaupa. Lisaks on võimalik näha ka enda lepinguid ja arveid [12].

- **Alexela** – samuti võimalik vaadata börsihinda. Et mitte tarbida kõige kallima elektrihinnaga ajal, on võimalik tellida e-postile teavitused, kui börsihind ületab seadistatud piiri. Tarbimisajalugu ega ka lepinguid Alexela äpis vaadata ei saa, küll aga näeb arveid. Lisaks on rakendusse koondatud ka muud ettevõtte teenused, sealhulgas haagiste rent, kütuse eest maksmine ja kohvikampaaniaga liitumine [13].

- **220 Energia** – samuti saab vaadata börsihinda ning tellida e-postile teavitused, kui börsihind ületab seadistatud hinnapiiri. Saab vaadata oma lepinguid, arveid ning tarbimisajalugu nädala, kuu kui ka aasta kaupa [14].

3.1.7 Muud rakendused energiakuludelt säästmiseks

Kuivõrd on eelpool mainitud rakenduste kasutajate eesmärgiks põhiliselt säästa raha, uurisime edasiarenduse ideede kogumise eesmärgil järgnevaid mobiilirakendusi:

- ***Light Bulb Saver App*** – rakendus arvutab, kui palju on kasutajal võimalik raha säästa, kui ta tavalise lambipirni asendaks LED või CFL pirniga. Sisestada tuleb olemasoleva pirni andmed ja uue pirni maksumus, rakendus arvutab säästetud raha 10 aasta kohta [15].

- ***Meter Readings*** – võimaldab jälgida näiteks elektri, vee ja gaasi tarbimist. Näite saab sisestada enda valitud perioodi järgi manuaalselt või ka csv-faili abil. Manuaalse sisestamise korral on võimalik tellida näitude sisestamiseks meeldetuletused. Mobiilirakendus kuvab statistika lihtsalt loetavatel graafikutel [16].

- ***Energy Tracker*** – paljude funktsionaalsustega mobiilirakendus: andmeid saab laadida üles csv-faili abil, tulemusi kuvada graafiliselt, jälgida tarbimist ja selle trende, võrrelda erinevaid perioode, tulemusi salvestada pdf või csv-failina ja palju muud [17].

3.2 Nõuded

Peatükis antakse ülevaade bakalaureusetöö raames valminud projekti funktsionaalsetest ja mittefunktsionaalsetest nõuetest. Nõuded on välja töötatud projektiliikmete poolt eesmärgiga kirjeldada rakenduse funktsionaalsust ning üldise süsteemi toimimist. Selle jaoks uuriti ning kaardistati sarnaseid olemasolevaid lahendusi ning autorite omavaheliste arutelude käigus töötati välja nõuded, millele projekt vastama peab. Peamiselt lähtuti autorite isiklikest soovidest ja vajadustest, mida rakendus tegema peaks. Lisaks arvestati ka potentsiaalse kliendi vajadusi ning vajalikke funktsionaalsusi, et pakkuda väärtust ka laiemale sihtgrupile. Nõuded pandi kirja ning vormistati piletitena *backlog*'is koos töökirjeldusega.

3.2.1 Funktsionaalsed nõuded

Funktsionaalsed nõuded kirjeldavad rakenduse ärilist vajadust ning annavad rakenduse kasutajale ülevaate rakenduse erinevatest funktsioonidest. Funktsionaalsete nõuete [18] väljatöötamiseks viidi projekti algfaasis projektiliikmete poolt läbi mitmeid arutelusid ning ajurünnakuid. Algselt pandi kirja planeeritud funktsionaalsused. Lisaks vaadati iga sprindi lõpus üle rakenduse nõuded ning jooksvalt lisati uusi funktsionaalsuseid või modifitseeriti eelnevaid.

Algselt kirja pandud funktsionaalsed nõuded:

- Päeva börsi elektrihindade nägemine, nende ühikute muutmine ning käibemaksu lisamine või eemaldamine
- Kasutaja isikliku elektrienergia tarbimisajaloo ning elektripakettide analüüs.
- Kasutaja elektri tootmis- ning salvestusseadmete tasuvusperioodi arvutamine.
- Tuleviku elektrihindade põhjal kasutajale tarbimissoovituste andmine.
- Elektri tuleviku hindade ja ilmateate API andmete (tuulegeneraatori või päikesepaneelide olemasolul) abil potentsiaalse tarbimise ja/või tootmise juhtimine: API endpoint, mida saaks potentsiaalselt mõni teine rakendus automatiseerimiseks kasutada.

Projekti arenduse käigus muudeti jooksvalt funktsionaalseid nõudeid ning tehti need täpsemaks. Lisati uusi nõudeid ja eemaldati või prioritiseeriti ümber erinevaid funktsionaalsusi. Projektiliikmed lähtusid töö käigus isiklikest vajadusest rakenduse kasutamisel ja tihti tükeldati suuremaid funktsionaalsuseid väiksemateks, et oleks võimalik kiiremini lisada rakendusele väiksemaid, kuid lõpuni arendatud komponente.

Funktsionaalsed nõuded, mis valmisid projekti käigus:

- Külalisel on võimalik näha eilse, tänase ja homse päeva börsi elektrihindu ning viimaste kuude keskmiseid börsihindu.
- Külaline ja registreeritud kasutaja saavad võrrelda elektrilepingute maksumusi kuupõhiselt.
- Külaline ja registreeritud kasutaja saavad võrrelda elektrienergia kulu tarbimisharjumuste muutmisel.
- Registreeritud kasutaja saab võrrelda elektrilepingute maksumusi tunnipõhiselt.
- Registreeritud kasutaja saab salvestada elektrilepinguid.
- Registreeritud kasutaja saab salvestada kuupõhiseid elektritarbimisi ja -tootmisi.
- Registreeritud kasutaja saab leida kuupõhiselt toodetud elektrienergia tulu.

3.2.2 Mittefunktsionaalsed nõuded

Mittefunktsionaalsete nõuded on seotud erinevate tarkvarade kasutamise, protsesside järgimise ja projekti käigus tehtud arendustööde põhimõtetega. Mittefunktsionaalsetel nõuetel puudub otsene mõju tarkvara funktsionaalsusele ja nende eesmärk anda projekti õnnestumisele vajalikud eeldused [19].

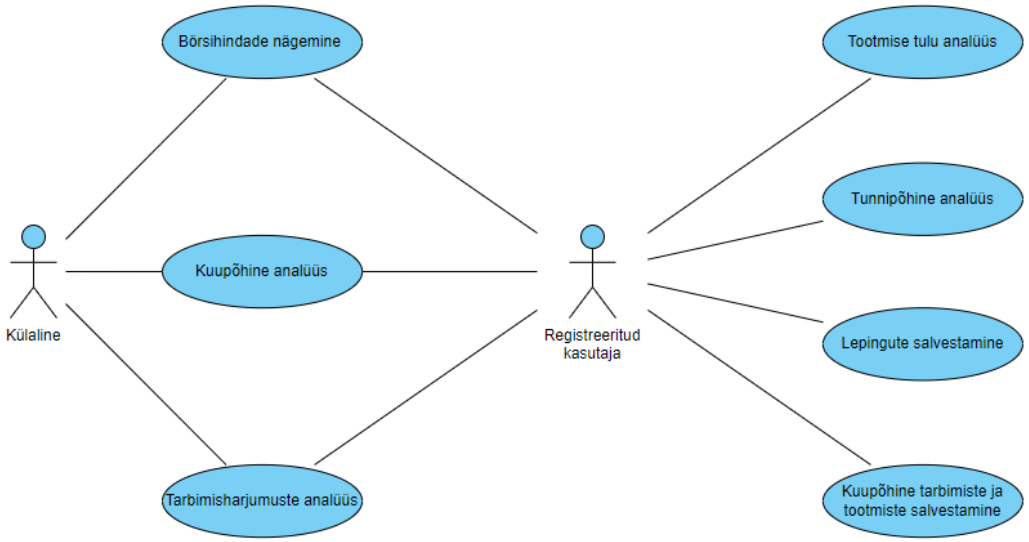
Mittefunktsionaalsed nõuded:

- Kasutaja andmed on turvaliselt hoiustatud.
- Rakenduse kättesaadavus on võimalikult suur.
- Rakenduse funktsioonid on testitud.

- Rakenduse kasutamine on lihtne ja arusaadav.
- Rakendus toetab uute arenduste lisamist.

3.3 Kasutuslood

Koos funktsionaalsete nõuetega koostasid autorid rakendusele kasutuslood. Kasutuslugusid tuli rohkem, kuid järgnevates punktides paneme kirja mõned rakenduses projekti jooksul realiseeritud kasutuslood. Kasutuslood on kujutatud kasutuslugude diagrammil (Joonis 6).



Joonis 6. Kasutuslood

3.3.1 Börsihindade kuvamine

Üldkirjeldus

Rakenduse kasutaja näeb graafiku kujul tänase päeva elektrienergia börsihindu tunnipõhiselt.

Põhivoog

1. Kasutaja navigeerib vahelehele “Börsihinnad”.
2. Süsteem kuvab kasutajale graafikuna tänase päeva börsihinnad (Joonis 7).

3. Kasutajal on võimalik muuta hindade ühikud (EUR/MWh või s/kWh) ning samuti lisada või eemaldada käibemaksu.

Joonis 7. Börsihindade kuvatõmmis

3.3.2 Kuupõhine elektrihindade analüüs

Üldkirjeldus

Kasutaja saab võrrelda elektrikulu hinda, sisestades soovitud kuude tarbimise ja kaks elektriostu lepingut, mida soovib võrrelda.

Järeltingimused

- Kasutajale kuvatakse sisestatud andmete põhjal analüüsi tulemus.

Põhivoog

1. Kasutaja navigeerib vahelehele “Analüüs” ja valib sealt “Kuupõhine analüüs”.
2. Süsteem kuvab kasutajale kaks vormi:

a. Viimase 16 kuu tarbimise sisestamise vormi.

b. Kahe elektriostu lepingu sisestamise vormi.

3. Kasutaja sisestab endale sobivate kuude tarbimise ning kahe elektriostu lepingu andmed ning salvestab mõlemad vormid.

4. Süsteem kuvab korrektsete andmete puhul “Analüüsi” nupu.

5. Kasutaja vajutab “Analüüsi” nuppu.

6. Süsteem arvutab ja kuvab sisestatud andmete põhjal kulunud elektrienergiahinna eurodes mõlema lepingu puhul ning sõltuvalt tulemustest kasumi või kahjumi (Joonis 8).

7. Kasutajal on võimalus vormides andmeid muuta ning sooritada analüüs uuesti.

Tulemused	Ilma KM-ita	KM-iga
Kehtiva paketiga summa	635.12 EUR	762.14 EUR
Alternatiivse paketiga summa	592.67 EUR	711.2 EUR
Kasum(+)/kahjum(-)	**42.45 EUR**	**50.94 EUR**

Joonis 8. Analüüsi tulemused

Alternatiivvoog 1

Käivitamine: Kui kasutaja on sisse logitud, siis jätkub tegevus vastavalt Alternatiivvoole 1.

1. Põhivoo tegevus 1.

2. Süsteem kuvab kasutajale kaks vormi:

   a. Kasutaja salvestatud tarbimiste info kuude kohta.

   b. Kahe elektriostu lepingu sisestamise vormi.

3. Kasutaja saab täiendada kuutarbimiste vormi ning sisestab kahe elektriostu lepingu andmed ning salvestab vormi (Joonis 9).

4. Tegevus jätkub põhivoo tegevusega 4.

Alternatiivvoo lõpp.

3.3.3 Elektrienergia tarbimisharjumuste analüüs

Üldkirjeldus

Kasutajal on võimalik sisestada tunnipõhise elektrienergia tarbimise csv-fail ning muuta parameetreid, mis näitavad kui palju on võimalik erinevate tarbimisharjumuste muutmisega raha kokku hoida (või juurde maksta).

Järeltingimused

- Kasutajale kuvatakse sisestatud andmete põhjal analüüsi tulemus.

Põhivoog

1. Kasutaja navigeerib vahelehele “Analüüs” ja valib sealt “Tarbimisharjumuste analüüs”.
2. Süsteem kuvab kasutajale kaks vormi: faili üleslaadimise vormi ning parameetrite vormi.
3. Kasutaja laadib üles korrektse faili, muudab parameetrid endale sobivaks ning salvestab mõlemad vormid. Süsteem kuvab korrektsete andmete puhul “Analüüsi” nupu.
4. Kasutaja vajutab “Analüüsi” nuppu.
5. Süsteem arvutab ja kuvab sisestatud andmete põhjal elektrienergia algse kulu ning uute parameetritega arvutatud kulu kasumi või kahjumi eurodes.
6. Kasutajal on võimalus andmeid vormides muuta ning uuesti analüüsida.

3.3.4 Isiklike elektrilepingute salvestamine

Üldkirjeldus

Kasutaja saab elektri ostu- ja müügilepinguid salvestada, et neid analüüsides ilma andmeid uuesti sisestamata kasutada.

Eeltingimused

- Kasutaja on sisse logitud.

Järeltingimused

- Andmebaasi on salvestatud kasutaja isiklikud elektrilepingud.

Põhivoog

1. Kasutaja navigeerib vahelehele “Minu Lepingud”.
2. Süsteem kuvab kasutajale tema “Ostulepingud” ning “Müügilepingud”.
3. Kasutajal on seejärel võimalik:
    a. Salvestada uusi lepinguid.
    b. Muuta olemasolevaid lepinguid.
    c. Kustutada olemasolevaid lepinguid.
4. Kasutaja uued lepingud on salvestatud andmebaasi, muudetud lepingud salvestatakse andmebaasi muudetud kujul, kustutatud lepingud eemaldatakse andmebaasist.

3.3.5 Isiklike kuupõhiste tarbimiste ja tootmiste salvestamine

Üldkirjeldus

Kasutajal on võimalus salvestada igakuised elektrienergia tarbimis- ja tootmisandmeid, et hiljem neid erinevates analüüsides kasutada.

Eeltingimused

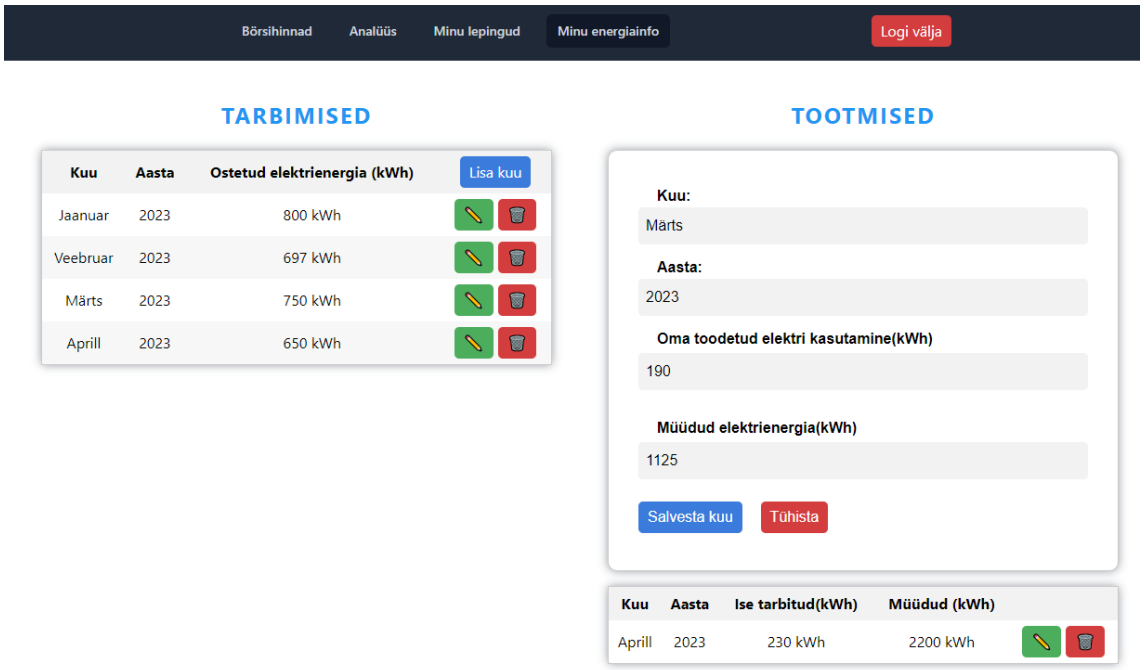
- Kasutaja on sisse logitud.

Järeltingimused

- Kasutaja isiklikud elektrienergia tarbimis- ja tootmisandmed on salvestatud andmebaasi.

Põhivoog

1. Kasutaja navigeerib vahelehele “Minu energiainfo”.
2. Süsteem kuvab kasutajale tema “Tarbimised” ning “Tootmised” (Joonis 9).
3. Kasutajal on seejärel võimalik:
    a. Salvestada uusi tootmiseid ja tarbimisi.
    b. Muuta olemasolevaid tootmiseid ja tarbimisi.
    c. Kustutada olemasolevaid tootmiseid ja tarbimisi.
4. Kasutaja uued andmed on salvestatud andmebaasi, muudetud andmed salvestatakse andmebaasi muudetud kujul, kustutatud andmed eemaldatakse andmebaasist.



Joonis 9. Kasutaja tarbimised ja tootmised

3.3.6 Tunnipõhine elektrihindade analüüs

Üldkirjeldus

Kasutajal on võimalik sisestada tunnipõhine elektrienergia tarbimise csv-fail ning kaks elektriostu lepingut, mida ta võrrelda soovib.

Eeltingimused

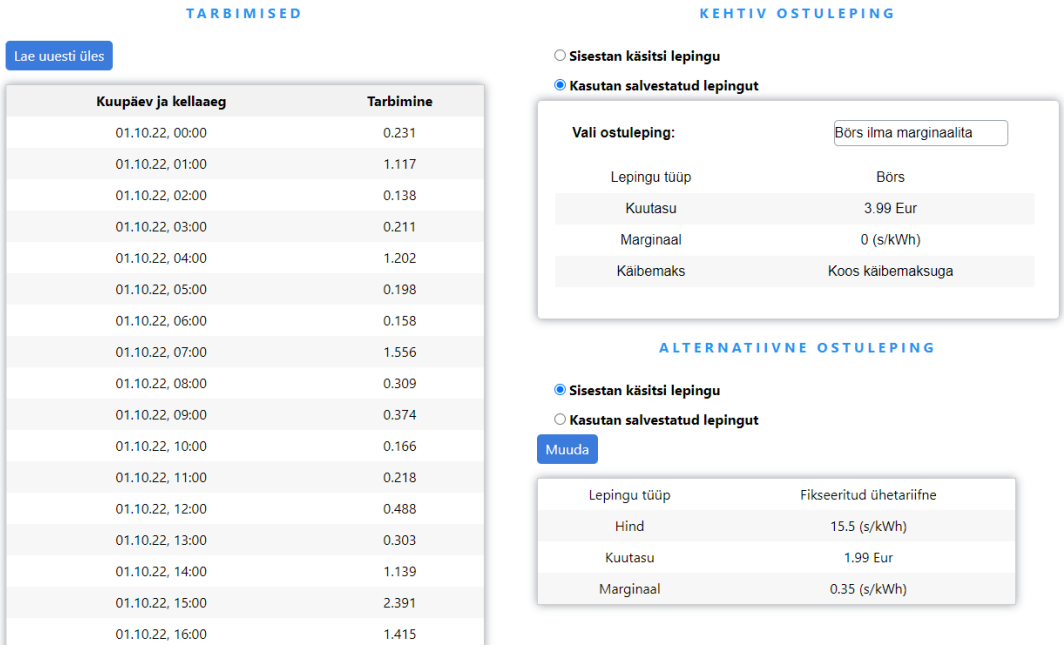
- Kasutaja on sisse logitud.

Järeltingimused

- Kasutajale kuvatakse sisestatud andmete põhjal analüüsi tulemus.

Põhivoog

1. Kasutaja navigeerib vahelehele “Analüüs” ja valib sealt “Tunnipõhine analüüs”.
2. Süsteem kuvab kasutajale kaks vormi:
    a. Faili üleslaadimise vorm.
    b. Kahe elektrilepingu sisestamise vorm.
3. Kasutaja laadib üles tunnipõhise elektritarbimise faili, kahe elektriostu lepingute andmed ning salvestab mõlemad vormid. Elektriostu lepingute andmed on võimalik sisestada käsitsi või kasutada eelnevalt salvestatud lepinguid (Joonis 10).
4. Süsteem kuvab korrektsete andmete puhul “Analüüsi” nupu.
5. Kasutaja vajutab “Analüüsi” nuppu.
6. Süsteem arvutab ja kuvab sisestatud andmete põhjal elektrienergia kuluhinna mõlema lepingu puhul ning sõltuvalt tulemustest kasumi või kahjumi.
7. Kasutajal on võimalus andmeid vormides muuta ning uuesti analüüsida.



Joonis 10. Andmete sisestamine

3.3.7 Elektrienergia tootmise tulu analüüs

Üldkirjeldus

Kasutajal on võimalik sisestada endale sobivate kuude elektrienergia tootmise andmed ning elektri ostu- ning müügileping, mille põhjal arvutatakse sisestatud kuude tootmise tulu eurodes.

Eeltingimused

- Kasutaja on sisse logitud.

Järeltingimused

- Kasutajale kuvatakse sisestatud andmete põhjal analüüsi tulemus.

Põhivoog

1. Kasutaja navigeerib vahelehele “Analüüs” ja valib sealt “Tootmise tulu analüüs”.
2. Süsteem kuvab kasutajale kaks vormi:
    a. Elektri ostu- ja müügilepingu sisestamise vorm.
    b. Kasutaja salvestatud tootmiste info kuude kohta.

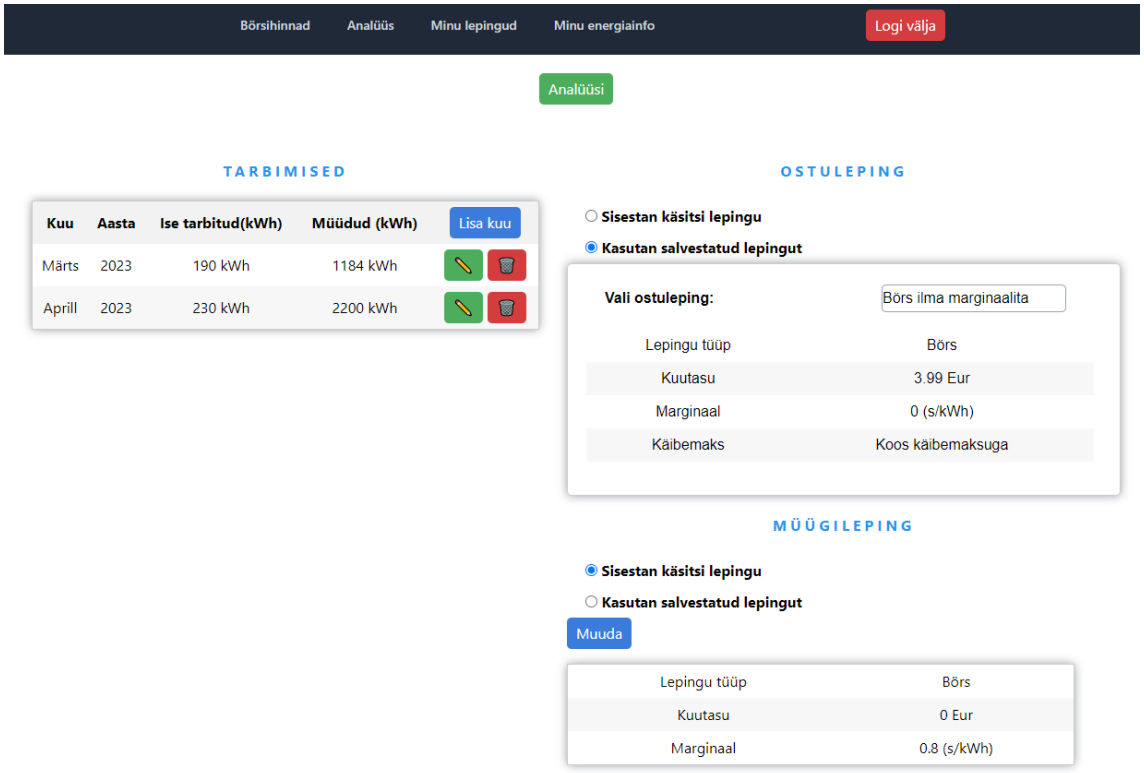
3. Kasutaja sisestab elektri ostu- ja müügilepingu andmed ning täiendab soovi korral tootmiste infot. Lepingute andmed on võimalik sisestada käsitsi või kasutada eelnevalt salvestatud lepinguid (Joonis 11).

4. Süsteem kuvab korrektsete andmete puhul “Analüüsi” nupu.

5. Kasutaja vajutab “Analüüsi” nupule.

6. Süsteem arvutab ja kuvab sisestatud andmete põhjal iga kuu tulu eurodes (Joonis 12).

7. Kasutajal on võimalik andmeid vormides muuta ning uuesti analüüsida.



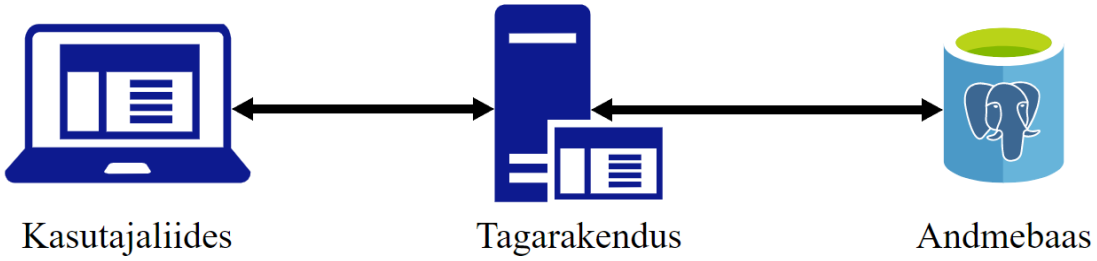
Joonis 11. Tootmise analüüsi vorm

Kuu	Aasta	Ise kasutatud (EUR)	Müüdud (EUR)	Kokku (EUR)	Kokku (EUR +km)
Märts	2023	16.57	93.77	110.34	132.41
Aprill	2023	15.13	127.16	142.29	170.75

Joonis 12. Tootmise analüüsi tulemused

3.4 Üldine arhitektuur

Rakendus koosneb kolmest komponendist. Tagarakenduseks on C# programmeerimiskeeles kirjutatud .NET REST komponent. Kasutajaliidesena on arendatud TypeScript programmeerimiskeeles ja Vue.js raamistikus rakendus. Andmebaasi komponendiks on PostgreSQL tüüpi andmebaas (Joonis 13).



Joonis 13. Üldine arhitektuur

Rakenduse 3 eraldiseisvat komponenti:

- **Tagarakendus** - ülesandeks on suhelda andmebaasiga ning tegeleda kasutajaliidese poolt saadetud või andmebaasist päritud andmete analüüsimisega.
- **Kasutajaliides** - rakenduse kasutaja näeb kasutajaliidese komponenti oma veebibrauseris ning saab selle kaudu rakendust kasutada. Kasutajaliides suhtleb REST API päringute abil tagarakendusega kasutades HTTP protokolli.
- **Andmebaas** - sinna salvestatakse ning sealt päritakse analüüsiks vajaminevaid andmeid kasutajate poolt salvestatud isiklike andmete kui ka elektribörsi ajalooliste hindade kohta.

Kõik komponendid on paigutatud eraldi Docker konteineritesse. Arenduse lihtsustamiseks on kõik komponendid ühes projektis ja GitLabi repositooriumis. Rakenduse kasvades või enne lansseerimist võib kujuneda vajalikuks eraldada komponendid eraldi repositooriumidesse. Rakenduse kvaliteedi tagamiseks kasutasid autorid ka tagarakenduse API *endpoint*- ning ühikteste. API testid ja ühiktestid on samas projektis.

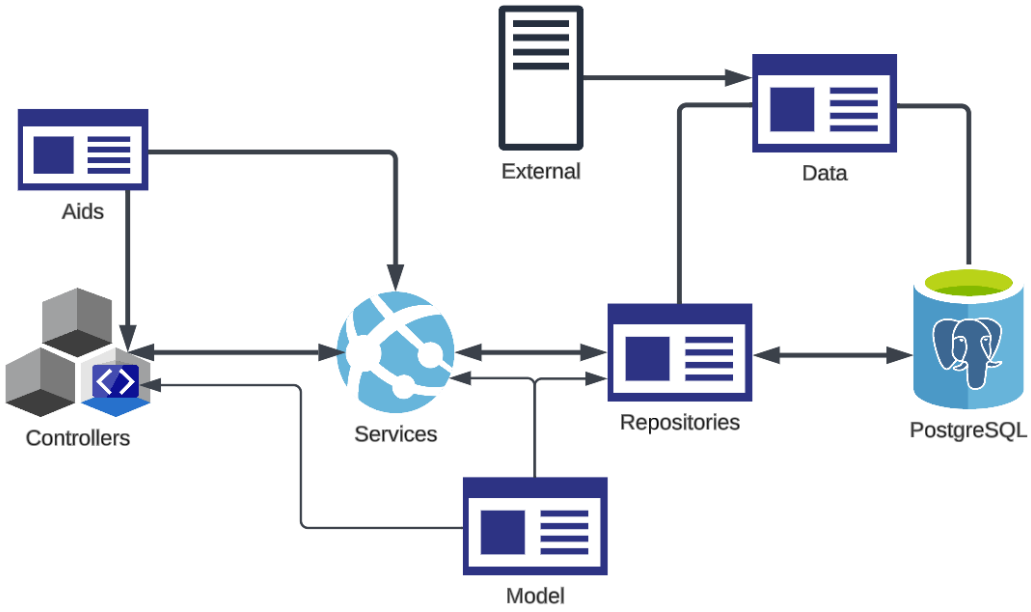
Autorid järgisid tarkvara arhitektuuri mustreid [20], mis eraldavad kasutajaliidese, tagarakenduse ja andmebaasi kihi funktsioone; võimaldavad igat kihti eraldi arendada, testida ning hooldada; teha muudatusi ilma teisi kihte mõjutamata.

3.5 Tagarakenduse arhitektuur

Tagarakenduse arhitektuur on jaotatud erinevateks kihtideks, eesmärgiga hoida lahus üksteisest sõltumatud kihid, järgida puhta koodi ning arhitektuuri printsiipe [21] ja seejuures lihtsustada, kiirendada ning optimeerida rakenduse arendusvoogu ja koodi refaktoreerimist (Joonis 14).

Tagarakenduses loodud kihid:

- **Aids** - sisaldab klasse abistavate meetoditega.
- **Controllers** - kausta on koondatud klassid, mille kaudu on võimalik kasutajaliidesel äriloogikaga suhelda.
- **Services** - vastutab tagarakenduse äriloogika eest.
- **Repositories** - suhtleb andmebaasiga.
- **Data** - vastutab andmebaasi, seal sisalduvate tabelite ning andmete initsialiseerimise eest.
- **Model** - sinna on loodud klassid, mis kirjeldavad rakenduses kasutatavaid objekte.
- **External** - sisaldab väliseid teenuseid.



Joonis 14. Tagarakenduse arhitektuur

3.5.1 Aids

Aids kausta on autorid lisanud vaid ühe klassi: DateTimeHelpers. Rakenduse kasvades võib siiski erinevaid abistavaid klasse juurde lisanduda. DateTimeHelpers klass tegeleb kuupäevade ja kellaaegade konverteerimisega. Kuna rakendus on mõeldud kasutamiseks eelkõige Eesti ja Balti riikide inimestele, on tähtis, et elektrihinnad klapiks kuupäevade ja kellaaegadega. Selleks, et vältida erinevate operatsioonisüsteemide ning arvuti süsteemi kellade erinevusi, loodi meetodid, mis konverteerivad UTC aja Ida Euroopa Standard ajaks ja vastupidi. Lisaks saadakse sealt ka infot tähtpäevade kohta, et arvestada nendega analüüsis.

3.5.2 Controllers

Controllers sisaldab nelja ApiController klassi: AnalyzerController, AuthenticatorController, EnergyDataController, PricesController. Nende klasside kaudu saab kasutajaliides suhelda tagarakendusega.

Kuigi kasutajaliides ja tagarakendus saavad nende klasside kaudu omavahel suhelda, toimub tegelik loogika Services kihi sees. Klassides luuakse Services kihis leiduvate liideste olemid ning nende kaudu saadakse ligi äriloogikale. ApiController klassid võtavad pelgalt vastu kasutajaliidese päringuid, tagastavad vastuseid ning vahendavad andmeid Services kihi ja kasutajaliidese vahel. Samuti vastutavad nad sissetulevate päringute autoriseerimise kontrollimise eest.

3.5.3 Services

Services kiht sisaldab kuut äriloogikat eest vatutavat klassi ning nende liideseid:

- **AnalyzerService** – vastutab analüüsi loogika eest.
- **AuthenticationService** – vastutab autentimise ja kasutajate haldamise loogika eest.
- **ConsumptionService** – vastutab registreeritud kasutajate elektrienergia tarbimise andmete haldamise loogika eest.
- **ContractService -** vastutab registreeritud kasutajate lepingute andmete haldamise loogika eest.

- **PricesService –** vastutab päritud elektri börsihindade tagastamise loogika eest.
- **ProductionService -** vastutab registreeritud kasutajate elektrienergia tootmise andmete haldamise loogika eest.

Seetõttu on Services kiht tagarakenduse keskne komponent, mis võimaldab Controllers kihil saada juurdepääs vajalikele funktsionaalsustele. Services kiht ei suhtle andmebaasiga otse, vaid saab vajalikke andmetega seotud toiminguid teha Repositories kihi abil.

3.5.4 Repositories

Repositories kiht sisaldab viit andmehalduse eest vastutavat klassi ning nende klasside liideseid:

- **AuthenticationRepository** – vastutab autentimise andmehalduse eest.
- **ConsumptionRepository** – vastutab elektrienergia tarbimise andmehalduse eest.
- **ContractRepository -** vastutab lepingute andmehalduse eest
- **PricesRepository –** vastutab elektri börsihindade andmehalduse eest.
- **ProductionRepository -** vastutab elektrienergia tootmise andmehalduse eest.

Repositories kiht vastutab andmebaasiga suhtlemise eest ning võimaldab tagada turvalisuse ja lahususe andmete ning kasutaja vahel. Antud kihi ülesandeks on tagada ligipääs andmetele ning põhilistele andmetega seotud tegevustele: andmete lisamine, lugemine, muutmine ja kustutamine. Kasutaja ei saa Repositories kihile ligi otse, vaid suhtleb Controllers kihiga, mis kutsub välja Services kihis olevaid meetodeid, mis omakorda kutsub välja vajaminevad andmehaldusmeetodid Repositories kihis. Repositories kihi kasutamine aitab vähendada sõltuvusi andmebaasist ja muudest tehnoloogiatest. Seetõttu ei mõjuta tarkvara loogikat näiteks muudatuste tegemine andmebaasi struktuuris või teistes kasutavates tehnoloogiates.

3.5.5 Data

Data kihti on koondatud andmebaasi tabeleid kirjeldavad andmeobjektid, andmete initsialiseerimise klass DbInitializer ja andmebaasi kirjeldav klass ElectricityDatabase.

Data kiht tegeleb andmete haldamisega ning tagab juurdepääsu andmebaasi andmetele ja nendega seotud tegevustele kogu rakenduse ulatuses.

3.5.6 Model

Sisaldab klasse, mis kirjeldavad rakenduses kasutatavaid objekte. Autorid on jaganud klassid nelja erinevasse gruppi: AnalysisModel, ContractModel, InputModel ja DTO. AnalysisModel kausta on koondatud analüüsiga seonduvad klassid, ContractModel kausta lepingutega seotud klassid, InputModel kausta andmete sisestamisega seotud klassid ning DTO kausta on lisatud klassid, mis kirjeldavad olemeid, mida liigutatakse erinevate rakenduse kihtide vahel. Lisaks saadetakse samal kujul tagarakendusest kasutajaliidesesse andmeid.

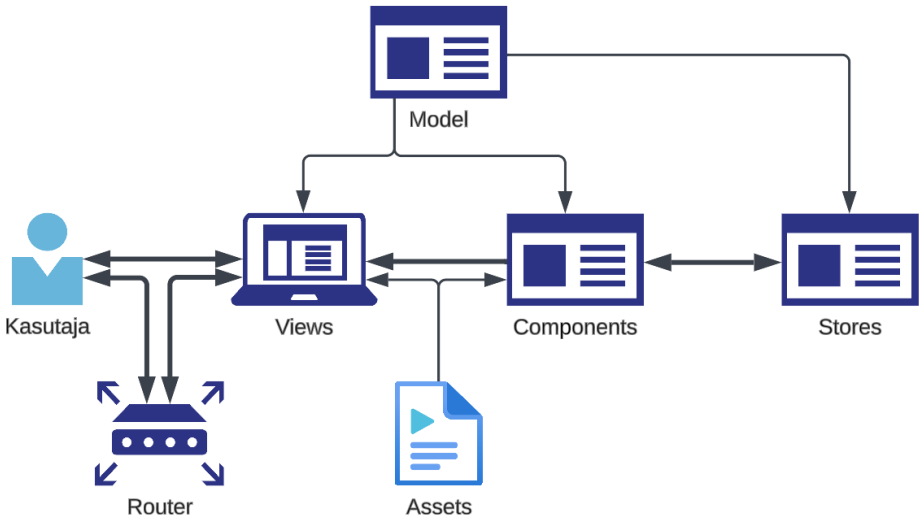
3.5.7 External

External kihti koondatakse liidestused väliste teenustega. Selles projekti faasis on liidestus ainult Eleringi API-ga, kust päritakse Nordpool börsi andmeid [22], kuid vajadusel ja projekti kasvades lisatakse sinna liidestusi juurde.

3.6 Kasutajaliidese arhitektuur

Rakendusega suhtlemiseks lõid autorid kasutajaliidese, mis on kirjutatud TypeScript programmeerimiskeeles [23] ning kasutab Vue.js javascripti raamistikku [24]. Sarnaselt tagarakendusele on kasutajaliideses arhitektuurselt loodud erinevad kihid, mis täidavad erinevaid ülesandeid (Joonis 15):

- **Assets** - sisaldab faile, mida vajatakse kogu rakenduses (pildid, stiil jne).
- **Components** - sinna koondatakse kõik kasutajaliideses vajaminevad komponendid.
- **Model** - andmeobjektide kirjeldused.
- **Router** - vastutab rakenduse kasutajaliideses navigeerimise eest.
- **Stores** - sinna on koondatud funktsioonid, mis suhtlevad tagarakenduse API-ga.
- **Views** - sisaldab erinevaid vaateid kasutaja jaoks.



Joonis 15. Kasutajaliidese arhitektuur

3.6.1 Assets

Assets kaust on loodud rakenduses vajaminevate meediafailide hoiustamiseks - näiteks pildid ja videod, kuid autorid paigutasid antud kausta ka style.css faili, kuhu on koondatud suurem osa CSS koodist, mida on ühtse kasutajaliidese disainiks vaja.

3.6.2 Components

Components kaust sisaldab erinevaid Vue.js komponente. Komponendid on taaskasutatavad ja iseseisvad osad, mida on võimalik kasutada erinevates vaadetes ja ka teiste komponentide sees. See aitab hoida koodi puhtana ning vältida koodi dubleerimist. Näiteks sisaldab projekt komponenti Login, mis võimaldab sisselogimise vaadet näidata kõikjal rakenduses. Veel on korduvalt kasutatud ka näiteks PresetContractInput komponenti, mis võimaldab salvestada elektripakette eri vaadetes vastavalt kontekstile.

Components kaust sisaldab ka Forms kausta, kus asuvad erinevad vormid, mida kasutatakse analüüsi vaadetes. Vormi eesmärgiks on salvestada vajalik info, mis tuleb komponentidest, mis asuvad vormi sees.

3.6.3 Model

Sarnaselt tagarakendusele sisaldab Model klasse, mis kirjeldavad andmeobjekte, mida rakenduses kasutatakse ja mida on vaja, et suhelda tagarakenduse API-ga.

3.6.4 Router

Router sisaldab lehel navigeerimise loogikat ning aitab muuta kasutajakogemust kiiremaks, lihtsamaks ja intuitiivsemaks. Router võimaldab navigeerida erinevate vaadete ja lehtede vahel ilma lehte uuesti laadimata. Sisaldab kümme erinevat URL-i ja nende lehtede komponente. Routeri abil luuakse ka veebiajalugu vastavalt kasutaja navigeerimisele veebirakenduses.

3.6.5 Stores

Stores kausta on autorid koondanud põhiliselt API-ga suhtlemiseks vajaminevad funktsioonid, mis on määratletud Vue.js Pinia raamistiku abil. Lisaks kasutatakse Store'e selleks, et hoida korduvalt kasutatavaid ning *hard-code*'itud andmeid, nagu näiteks kuude nimetused eesti keeles või ka seisundi andmeid, nagu näiteks isAuthenticated authenticatorStore'is, mis annab rakendusele infot, kas kasutaja on sisse logitud või ei.

3.6.6 Views

Views on Vue.js raamistiku komponent, mida kasutatakse kasutajaliideses erinevate vaadete loomiseks ning suures plaanis on see ka ise komponent. View on põhiliselt kogum erinevatest varem loodud korduvkasutatavatest komponentidest ning võib sisaldada HTMLi, JavaScripti ja CSS-i. View'de kasutamine muudab rakenduse haldamise ja arendamise lihtsamaks ja selgemaks, võimaldades selgelt eraldada rakenduse loogikat ja kasutajaliidest.

3.7 Andmebaas

Rakenduses vajalike andmete salvestamiseks ja haldamiseks kasutasid autorid PostgreSQL relatsioonilist andmebaasi. Andmebaas loodi kasutades Dockeri *image*'it ning käivitati Dockeri konteineris.

Andmebaasis on kuus tabelit (Joonis 16):

1. **Prices** tabel sisaldab andmeid Nordpool börsi elektrihindade kohta (s/kWh, EUR/mWH) ja lisaks ka UNIX ajatemplit ning UTC aega.

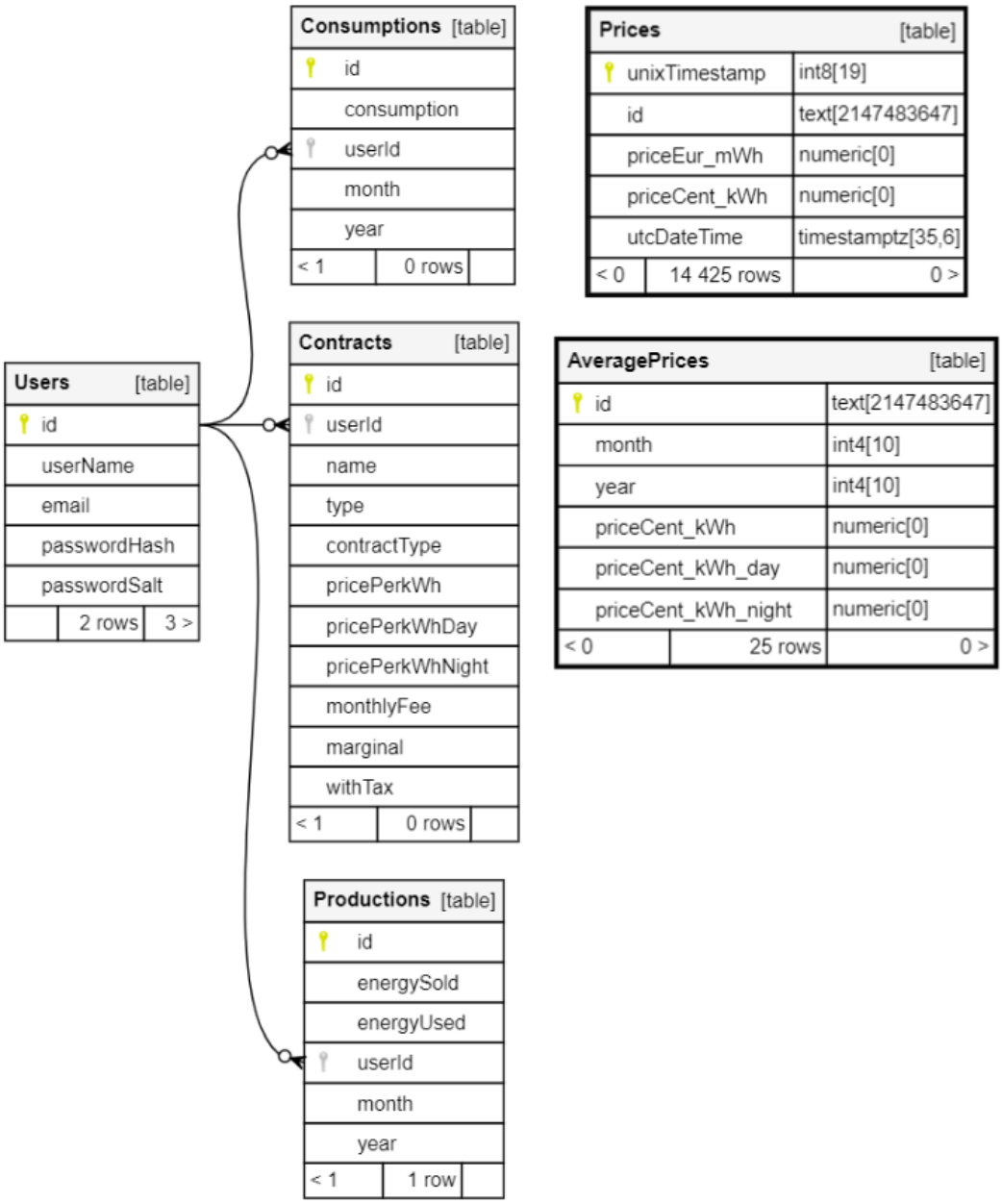
2. **AveragePrices** tabelis on Nordpool börsi kuu keskmised elektrihinnad.

3. **Users** tabelisse on sisestatud kasutajate andmed, sealhulgas kasutaja ID, kasutajanimi, e-posti aadress ning salasõna krüpteerituna SHA512 algoritmi abil.

4. **Consumptions** tabel sisaldab registreeritud kasutajate elektritarbimise andmeid kuu kaupa.

5. **Contracts** tabelisse on salvestatud registreeritud kasutajate lepingud.

6. **Productions** tabel sisaldab registreeritud kasutajate elektritootmiste andmeid kuu kaupa.



Joonis 16. SchemaSpy poolt genereeritud andmebaasi olemi-suhte diagramm

3.8 Disain

Rakenduse arendamisel lähtusid autorid erinevatest objektorienteeritud programmeerimise printsiipidest:

- **SOLID** - koosneb viiest printsiibist, mis aitavad objektorienteeritud programmeerimises luua hallatavaid tarkvarasüsteeme. SOLID on akronüüm järgmisest viiest printsiibist [20] [25]:
    - *Single-responsibility principle* - iga objekt, nagu näiteks klass või meetod, peaks omama ainult ühte ülesannet või vastutusala.
    - *Open-closed principle* - komponendid on avatud laiendamisele, kuid mitte muutmisele (st. muudatused tuleb sisse viia laienduste kaudu).
    - *Liskov substitution principle* - iga alamklassi peab saama kasutada ka ülemklassi asemel, see tähendab, et alamklass ei tohiks muuta ülemklassi ootusi.
    - *Interface segregation principle* - suurte ja üldiste liideste asemel tuleks kasutada spetsiifilisemaid liideseid, et implementeerida konkreetset funktsionaalsust.
    - *Dependency inversion principle* - kõrgemal tasemel olevad tarkvara moodulid peaksid sõltuma abstraktsioonist, mitte madalama taseme moodulist.
- **DRY** - *Don't repeat yourself* (ära korda ennast) [26]

Neid printsiipe kasutati projekti arendamisel, sest need aitavad hoida koodi hallatavana, puhtana ning vältida dubleerimist. Koodi refaktoreeriti pidevalt ning üritati hoida võimalikult abstraktse ja korduvkasutatavana.

Järgnevalt tuuakse näiteid rakenduse disainis kasutatud mustritest ja printsiipidest:

Repositooriumi muster (*Repository pattern*) - igale äriloogika objektile loodi eraldi repositoorium ning liides, kus on defineeritud vajaminevad/saadavalolevad andmehalduse meetodid. Repositooriumi mustri kasutamise eesmärk on eraldada

andmehalduskiht loogikakihist ja ülejäänud rakendusest [27]. Loodi üldine liides IBaseRepository (Joonis 17), mis sisaldab CRUD meetodeid ning mida pärivad IConsumptionRepository, IContractRepository ning IProductionRepository (Joonis 18).

```
namespace BackEnd.Repositories.Interfaces.Base
{
    public interface IBaseRepository<T>
    where T: class
    {
        Task<T> Save(T data);
        Task<T> Get(string id);
        Task<T> Update(string id, T data);
        Task<bool> Delete(string id);
    }
}
```

Joonis 17. IBaseRepository kood

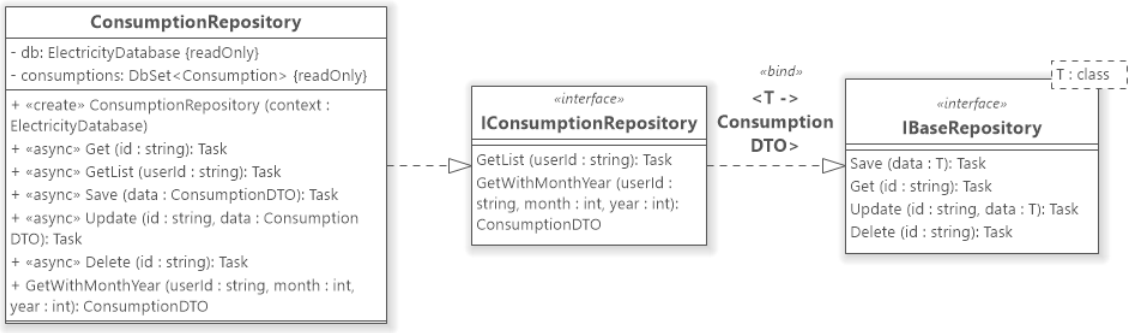
```
using BackEnd.DTO;
using BackEnd.Repositories.Interfaces.Base;

namespace BackEnd.Repositories.Interfaces
{
    public interface IConsumptionRepository:
    IBaseRepository<ConsumptionDTO>
    {
        Task<List<ConsumptionDTO>> GetList(string
userId);
        ConsumptionDTO GetWithMonthYear(string userId,
int month, int year);
    }
}
```

Joonis 18. IConsumptionRepository liides

DRY printsiip - Üldise liidese kasutamine väldib koodi dubleerimist ning muudab koodi hallatavamaks ja loetavamaks [26].

Näiteks ConsumptionRepository implementeerib IConsumptionRepository liidest, mis omakorda pärib IBaseRepository liidest, kus on defineeritud 4 CRUD meetodit. Lisaks IBaseRepository liideses kirjeldatud meetoditele on lisatud IConsumptionRepository meetod GetList(string userId) ja GetWithMonthYear(string userId, int month, int year) (Joonis 19).



Joonis 19. ConsumptionRepository klassidiagramm

DRY printsiipi rakendati ka kasutajaliideses:

- Rakendus sisaldab mitut korduvalt kasutatud komponenti, näiteks: UploadFileForm, MonthProductionForm ja MonthConsumptionForm.
- Kasutajaliidese CSS kood koondati ühte style.css faili, et ühtlustada kasutajaliidese disani ja teha muudatuste sisseviimine lihtsamaks ning kiiremaks.
- Kasutati ContractDetails komponendis props.action muutujat, et anda komponendile teada, missugust funktsionaalsust vajatakse (Joonis 20).

```
<h4 v-if= "props.action==='create'" class="pretty-h4">Uus leping</h4>
<h4 v-if= "props.action==='edit'" class="pretty-h4">Muuda leping</h4>
```

Joonis 20. Näide, kuidas propside abil anti teada, millist funktsionaalsust oodatakse

Sõltuvuste lisamine (*Dependency injection*) – muudab klassi sõltumatuks talle antud sõltuvustest, eraldades objekti initsialiseerimise selle kasutamisest. See lähenemine muudab koodi lihtsamalt testitavaks, laiendatavamaks ning modulaarsemaks [20] [25].

Tagarakenduse AuthenticationService klassi on sõltuvusena lisatud repositooriumi kirjeldav liides IAuthenticationRepository ja rakenduse seadistusi kirjeldav IConfiguration liides (Joonis 21).

```
namespace BackEnd.Service
{
    public class AuthenticationService:
    IAuthenticationService
    {
        private readonly IAuthenticationRepository repo;
        private readonly IConfiguration _configuration;

        public
        AuthenticationService(IAuthenticationRepository
        repository, IConfiguration configuration)
        {
            repo = repository;
            _configuration = configuration;
        }
        // Järgnev kood
    }
}
```

Joonis 21. AuthenticationService ja selle sõltuvused

Sõltuvuste inversiooni printsiip (*Dependency inversion principle*) - Tagarakendus on jaotatud erinevateks kihtideks ega sõltu madalamal tasemel olevatest moodulitest. Teiste kihtidega suhtlemiseks kasutatakse liideseid, kus on määratletud saadaval olevad funktsioonid [20] [25].

Näiteks AuthenticationController kasutab AuthenticationService'iga suhtlemiseks IAuthenticationService objekti ning AuthenticationService omakorda IAuthenticationRepository ja IConfiguration objekti (Joonis 21).

Ülesandepõhine asünkroonse programmeerimise muster (TAP - *Task-based asynchronous pattern*) - *async* ja *await* märksõnade abil saab C# programmeerimiskeeles muuta meetodid ja klassid asünkroonseks [28].

AnalyzerController klassis on kõik meetodid asünkroonsed ning tagastavad Task<ActionResult> tüüpi objekte. Mustrit kasutatakse vältimaks lõimede blokeerimist samal ajal, kui API sooritab äriloogika operatsioone (Joonis 22).

```
// Eelnev kood

[HttpPost("consumptionProduction")]
public async Task<ActionResult>
UploadConsumptionProduction(IFormFile? file, string
fileName) => Ok(await
service.UploadConsumptionProductionCsv(file, fileName));

[HttpPost("consumption")]
public async Task<ActionResult>
UploadConsumption(IFormFile? file, string fileName) =>
Ok(await service.UploadConsumptionCsv(file, fileName));

[HttpPost("postParameters")]
public async Task<ActionResult> PostAnalysisParameters
([FromBody] AnalysisParameters parameters) => Ok(await
service.PostAnalysisParameters(parameters));

[HttpPost("postConsumptionEnergyInput")]
public async Task<ActionResult> PostEnergyInput
([FromBody] List<ConsumptionInput> energyInput) =>
Ok(await service.PostEnergyInput(energyInput));

// Järgnev kood
```

Joonis 22. Näide AnalyzerController meetoditest

Ühe vastutuse printsiip (*Single-responsibility principle*) - samuti üks SOLID printsiipidest, mille kohaselt iga objekt, nagu näiteks klass või meetod, peaks omama ainult ühte ülesannet või vastutusala [20] [25].

Suures plaanis kohtab konkreetset printsiipi rakenduses kõikjal: arhitektuur on jaotatud erinevateks kihtideks, et tegeleda mingisuguse vastutusalaga; meetoditel ja klassidel vastutavad ühe ülesande või vastutusala eest.

3.9 Testimine

Rakenduse kvaliteedi tagamiseks, koodi muutmise ja refaktoreerimise lihtsustamiseks, kasutasid autorid rakenduse testimist kolmel viisil:

1. Manuaalne testimine
2. Ühiktestid
3. API *endpoint*'ide testid

Lisaks sellele anti rakendus kasutada autorite lähikondsetele ja töökaaslastele, et nad saaksid seda testida n-ö musta kasti põhimõttel.

3.9.1 Manuaalne testimine

Rakenduse kasutajaliidese arendamise käigus kasutasid autorid manuaalset testimist katsetades loodud lahendusi, vaateid ning komponente. Põhiliseks tööriistaks oli Google Chrome veebibrauseris sisalduv *Developer Tools* (arendaja tööriistad), mille abil jälgiti võrguliiklust ning rakenduse poolt tehtud päringuid, parameetreid ning vastuseid. Lisaks logiti konsooli parameetreid ja muutujaid, et valideerida tulemusi ning rakenduse loogikat. Kasulikuks osutus Google Chrome'i laiendus *Vue Devtools*, mis võimaldab Vue.js raamistikus loodud rakendusi analüüsida ning inspekteerida igat komponenti, sündmust ja olekut eraldi.

3.9.2 Ühiktestid

Rakenduse repositooriumisse loodi ka MSTest raamistikul põhinev ühiktestide projekt. Selles projektis testiti Aids kihti loodud DateTimeHelpers klassi, rakenduse äriloogikat sisaldavat Services kihti ning andmebaasiga suhtlevat Repositories kihti. Ühiktestide projektis kasutati testandmeid ning lokaalse arvuti mälus olevat *InMemory* andmebaasi. Ühiktestidega kontrolliti tagarakenduse meetodeid isoleeritult ja neid jooksutati lokaalselt arenduse käigus kui ka GitLabis pideva integreerimise abil.

3.9.3 API *endpoint* testid

Tagarakenduse API testimiseks kasutati JavaScriptis loodud API *endpoint* teste, mis testivad rakenduse äriloogika ning Controller kihis olevate meetodite toimimist musta kasti põhimõttel. Testid loodi põhiliselt arenduse käigus, et valideerida arendatud meetodeid ning testida API toimimist. Teste jooksutati lokaalselt Postman rakenduse abil, kollektsioonid lisati GitLabi *pipeline*'i ning sarnaselt ühiktestidele jooksutati neid pideva integreerimise abil.

3.9.4 *Blackbox* testimine

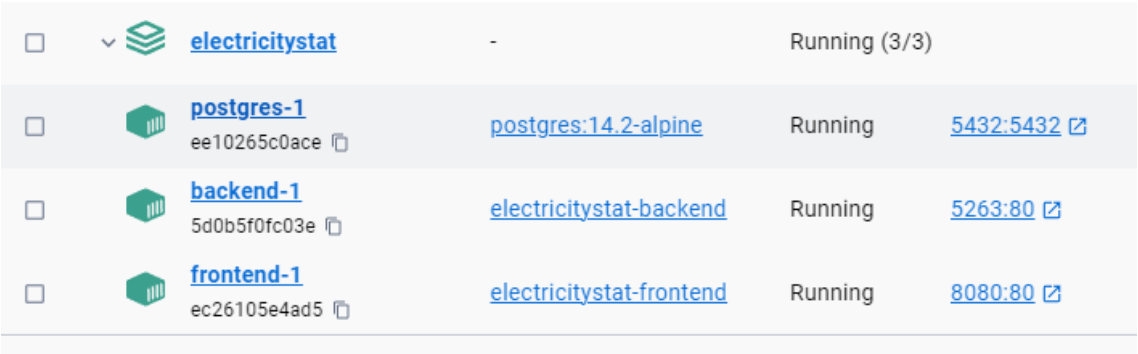
Autorid jagasid rakendust lähikondsetega, kes proovisid rakendust kasutada ja andsid oma arvamuse üldise funktsionaalsuse kohta ning raporteerisid leitud vigu. Lisaks saadi indikatsiooni kasutajatelt, millised funktsionaalsused võiks olla teistsugused või milliseid

võiks tulevikus lisada. Edasist arendust tehes võtsid autorid arvesse kasutajatelt saadud tagasisidet.

3.10 Docker

Rakenduse lihtsamaks testimiseks ning tulevikus lansseerimiseks sai kogu projekt pandud Docker konteinerisse. Tagarakendusele ja kasutajaliidesele koostati eraldi *Dockerfile*'id, kus kirjeldatakse, mida rakendus tööle minnes tegema peab. Projektile sai koostatud docker-compose.yml fail (Lisa 4), mille käivitamisel käsuga „*docker-compose up*" käsureal läheb rakendus konteineris tööle. Failis kirjeldatakse ka porte, mille kaudu teenused omavahel suhtlevad ning missuguste portide kaudu on võimalik tagarakendusele ja kasutajaliidesele ligi pääseda [29].

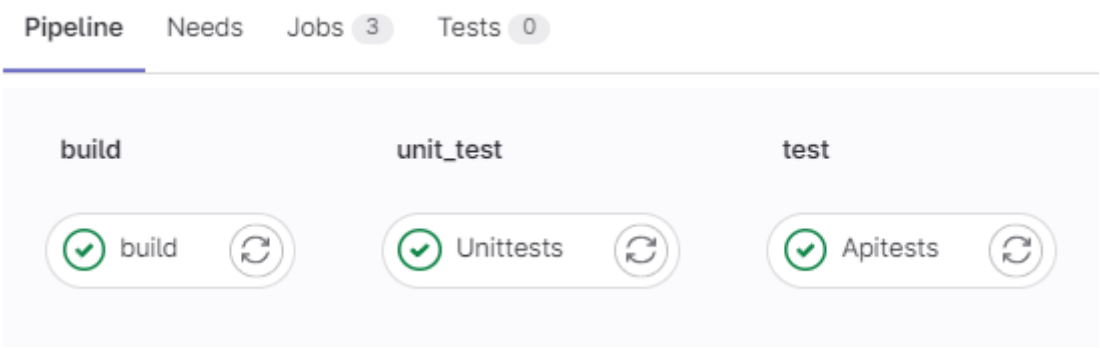
Tööle läheb üks konteiner, milles on kolme teenusena andmebaas, tagarakendus ning kasutajaliides (Joonis 23).



Joonis 23. Dockeri konteiner koos kolme teenusega

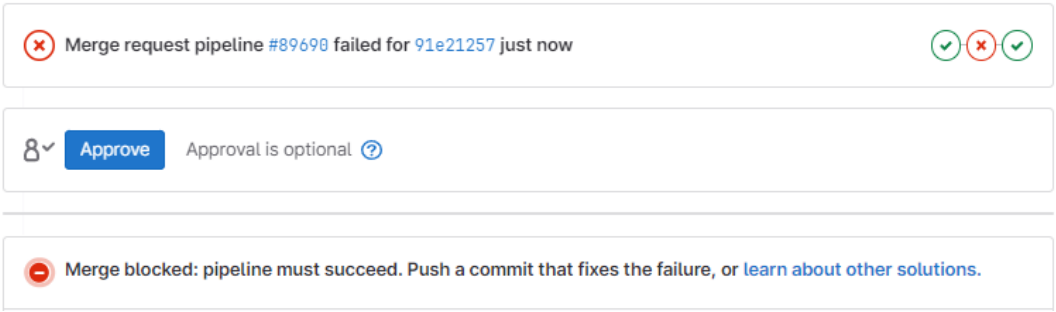
3.11 GitLab pipeline

Projekt paikneb GitLabi repositooriumis, kus kasutati pideva juurutamise kindluse jaoks *pipeline*'i [30]. Selle jaoks loodi projekti .gitlab-ci.yml (Lisa 5). Failis on kirjeldatud kolme etappi, mis *pipeline*'is tööle lähevad. GitLabis saab *pipeline*'i etappide tööd jälgida konsoolis kui ka graafiliselt (Joonis 24).



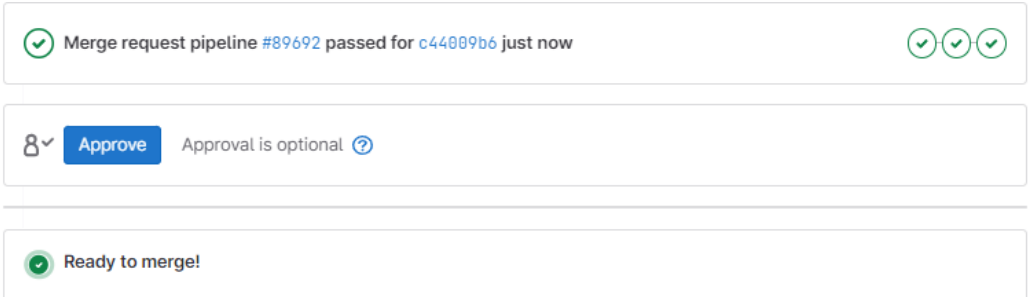
Joonis 24. *Pipeline*'i graafiline kujutamine GitLabis

Build etapp ehitab rakenduse ja kontrollib, et kood kompileeruks. *Unit_test* etapis jooksutatakse tagarakenduse ühikteste ning *test* etapis pannakse rakendus konteineris tööle ning jooksutatakse projekti lisatud API integratsiooniteste. Kõikidele etappidele on lisatud tingimus, et nad lähevad tööle ainult siis, kui tehakse *merge request* peaharusse. Autorid seadistasid GitLabi nii, et vähemalt ühe etapi ebaõnnestumisel on konkreetse haru peaharusse ühendamine keelatud (Joonis 25).



Joonis 25. *Pipeline* ebaõnnestus, peaharusse ühendamine on keelatud

Kui kõik etapid on edukalt läbitud, oli võimalus uus arendus peaharusse ühendada (Joonis 26).



Joonis 26. *Pipeline* õnnestus, peaharusse ühendamine on lubatud

4 Analüüs

4.1 Tehnoloogilised tööriistad

Autorid jäävad üldjoontes valitud tööriistadega rahule. GitLab suutis ühendada mitmed funktsionaalsused, mida autorid projekti arenduseks vajasid. Seal oli mugav hoida projekti repositooriumit ning samal platvormil hallata ka projekti *backlog*'i, sprinte ja pileteid. Töö autorid olid nii ülikooliõpingute käigus kui ka töökohtadel kasutanud rakendust Postman. Seetõttu oli Postmani kasutamine selge, mis lõi eelduse edukaks integratsioonitestide kirjutamiseks ning rakenduse testimiseks. Ajalogimiseks ja suhtlemiseks valiti autorite jaoks harjumuspärased ja vajalikke funktsioone sisaldavad rakendused, mistõttu ei tekkinud ka nendega probleeme.

Tagarakenduse programmeerimiskeeleks valiti C#, sest selles keeles omasid autorid kõige rohkem kogemust. Valik õigustas end, sest arendus toimus kiirelt ja suuremate probleemideta. Kasutajaliidese puhul langes samuti valik varasemalt kasutatud Vue.js raamistikule ja TypeScript programmeerimiskeelele.

Docker platvormi kasutamine võimaldas terve rakenduse käivitada vaid ühe käsuga. Arenduse lihtsustamiseks koostati ka teine docker-compose.yaml fail, mis käivitas konteineris vaid andmebaasi. See lähenemine võimaldas käivitada ülejäänud projektiosi lokaalselt ning andis võimaluse rakendust kerge vaevaga *debug*'ida. GitLab *pipeline*'i kasutusele võtmine võimaldas pärast arendustüki lõpetamist teha automaatne kontroll, isegi siis, kui arenduse käigus oli testide jooksutamine unustatud.

Ainsa ebaõnnestunud valikuna võib välja tuua arenduskeskkonna Visual Studio Code (VS Code). Olles kiire ning väga modulaarne, on VS Code võrreldes Visual Studioga oluliselt madalama võimekusega. Näiteks oli keeruline leida moodulit, mis suudaks genereerida klassidiagramme. Samuti olid koodimeetrika ja ühiktestide pistikprogrammid oluliselt madalama võimekusega, kui Visual Studiosse sisse ehitatud analoogid. Autorid said sellegipoolest VS Code'is oma projekti edukalt lõpule viidud ning positiivne on see, et VS Code'i puhul on tegemist vabavaralise tarkvaraga, milletõttu on võimalik samas keskkonnas rakendust jätkata ka pärast ülikooli lõpetamist.

4.2 Lahenduse vastavus nõuetele

4.2.1 Funktsionaalsetele nõuetele vastavus

Autorid töötasid projekti alguses välja rakenduse esialgsed funktsionaalsed nõuded. Projekti käigus kasutati agiilseid arendusvõtteid ning lisaks saadi kasutajatelt rakenduse kohta tagasisidet. Seetõttu vaadati nõuded iga sprindi lõpus üle ning vajadusel ka muudeti. Projekti vältel tekkisid rakendusele uued funktsionaalsed nõuded, mis olid kasutajatele tähtsamad kui autorite esialgsed ideed. Seetõttu võib öelda, et lahendus ei vasta esialgsetele funktsionaalsetele nõuetele. Suuremad muudatused funktsionaalsetes nõuetes on seotud kasutajate loomisega. Algselt ei planeeritud kasutajaid luua, kuid tagasisidest saime teada, et soovitakse korduvalt vajaminevaid andmeid salvestada (tarbimine, tootmine, lepingud). See arendus lõi eelduse võimalikuks äriliseks kasuks, kui tulevikus registreerimine tasuliseks muuta. Teise suurema muudatusena otsustati edasi lükata planeeritud API *endpoint*'i arendus, mis tagastaks analüüsi tulemusi mõnele tarbimist automatiseerivale seadmele. Antud funktsionaalsuse puhul oli kasutajatel huvi väike ning autorite eesmärk oli luua rakendus võimalikult suurele sihtgrupile.

Nimekriri realiseeritud funktsionaalsustest (kasutuslood ning kuvatõmmised peatükis 3.3 Kasutuslood):

1. Nordpool börsi elektrihindade kuvamine, ühikute muutmine, käibemaksu lisamine või eemaldamine.
2. Kuu keskmiste Nordpool börsihindade kuvamine.
3. Kuupõhine elektri tarbimise analüüs ning pakettide võrdlus.
4. Tunnipõhine tarbimisharjumuste analüüs csv-failist loetud andmetega.
5. Kasutaja registreerimise võimalus.
6. Registreeritud kasutaja lepingute, kuise elektri tarbimise ja tootmise salvestamine andmebaasi.
7. Registreeritud kasutaja tunnipõhine elektri tarbimise analüüs ja pakettide võrdlus csv-failist loetud andmetega.

8. Registreeritud kasutaja tootmise tulu analüüs.

4.2.2 Mittefunktsionaalsetele nõuetele vastavus

Projekti lõpetamise hetkel pole autorid rakendust veel *live* keskkonda pannud. Seega paljudele mittefunktsionaalsetele nõuetele vastavust ei ole võimalik täielikult üle kontrollida.

Selleks, et oleks võimalik kiiremini ja lihtsamini uusi arendusi luua ning suurema eneskindlusega koodi refaktoreerida, on tagarakenduse funktsioonid testitud nii ühik- kui ka API integratsioonitestidega. Lisaks on seadistatud GitLabi *pipeline*, mis takistab uute arenduste ühendamist olemasolevaga, kui rakendus ei kompileeru või testid ebaõnnestuvad. See lisab autoritele kindlust, et rakenduse kättesaadavus on heal tasemel. Kasutajate paroolid on räsifunktsiooni abil krüpteeritud ning autentimiseks kasutatakse *Json Web Token*'it. Selle lahenduse kasutamine aitab kaasa andmete turvalisele hoiustamisele, mis oli samuti üks mittefunktsionaalne nõue. Rakenduse jõudluse tagamiseks suuremate andmemahtudega, tuleb andmebaasi tabelitele lisada indekseerimine. Seni kuvatakse kasutajale analüüsi toimumise ajal info, et andmeid töödeldakse. Tagasisidest saime teada, et rakenduse kasutamine on lihtne, kuid peaks lisama informatsiooni, kuidas analüüsid töötavad, milliseid andmeid suudab rakendus töödelda ning kuidas analüüside tulemusi tõlgendada.

4.3 Võrdlus alternatiivsete lahendustega

Autorid uurisid enne rakenduse funktsionaalsete nõuete väljatöötamist olemasolevaid alternatiivseid lahendusi. Selle eesmärgiks oli kaardistada funktsionaalsused, mida taolise rakenduse kasutajad võiksid soovida kasutada. Uuriti viit alternatiivset veebirakendust ja kolme Eesti elektrimüüja mobiilirakendust ning nende funktsionaalsusi. Järgnevas tabelis (Tabel 1) on toodud välja erinevad põhilised funktsionaalsused, mida alternatiivsed lahendused kui ka autorite loodud veebirakendus (ElectricityStat) praeguses etapis sisaldas. Kui rakendus sisaldab kirjeldatud funktsionaalsust on tabelisse märgitud "✓".

Tabel 1. Võrdlus alternatiivsete lahendustega

	ElectricityStat	Elektrikell	MacDronic	Elektrikulu kalkulaator	Elektritarbimise kalkulaator	Elektrihind	Eesti Energia	Alexela	220 Energia
Börsihinna jälgimine	✓	✓	✓	✓		✓	✓	✓	✓
Ühikute muutmine börsihinna graafikul	✓								
Kuu keskmised börsihinnad	✓			✓					
Tarbimise ajastamise soovitamine		✓	✓			✓			
Erinevad arvutused päevalõikes		✓	✓	✓		✓			
Info börsi ja elektritarbimise ja arvutuste kohta			✓			✓			
Kasutajate loomise võimalus	✓								
Kasutaja tarbimise ja tootmise salvestamine	✓								
Kasutaja lepingute salvestamine või kuvamine	✓						✓		✓
Arvete kuvamine							✓	✓	✓
Pakettide võrdlus kuu- või aastapõhise tarbimise järgi	✓			✓	✓	✓			
Pakettide võrdlus tunnipõhise tarbimise järgi	✓								

	ElectricityStat	Elektrikell	MacDronic	Elektrikulu kalkulaator	Elektritarbimise kalkulaator	Elektrihind	Eesti Energia	Alexela	220 Energia
Tunnipõhine tarbimisharjumuste analüüs	✓								
Seadme(te) lisamine ja tarbimise analüüs				✓	✓				
Tootmise tulu analüüs	✓								
Tarbimise sisestamine csv-failiga	✓								
Kahe erineva päeva elektritarbimise võrdlemine							✓		
Tarbimisajaloo kuvamine	✓						✓		✓
Teavitused e-postile seatud hinna ületamisest								✓	✓
Ettevõtte muud teenused								✓	

Välja on toodud 20 funktsionaalsust. Autorite loodud rakendus sisaldab neist 12 ja on võrreldavate alternatiivsete lahenduste seas kõige rohkemate funktsioonidega. Järgmisena on enim funktsioone Elektrikulu kalkulaatoril, Elektrihinnal, Eesti Energial ja 220 Energial, mis tulevad toime viie välja toodud funktsionaalsusega.

Kuigi rakendused Elektrikulu kalkulaator, Elektritarbimise kalkulaator ning Elektrihind pakuvad võimalust analüüsida tarbitava elektri kogust ja hinda, on need kasutajakogemuselt üsna ebamugavad ning ebatäpsed, sest tarbimist ei saa sisestada tunnipõhiselt, vaid saab lisada seadme eeldatava tarbimise või üldise aastase tarbimise.

Autorite loodud rakendus võimaldab sisestada nii tarbimist kui ka tootmist csv-faili abil tunnipõhiselt, mis muudab analüüsimise ning pakettide võrdlemise oluliselt täpsemaks, seda eriti, kui tegemist on börsipaketiga.

Alternatiivsetest lahendustest saab registreeritud kasutajana kasutada ainult elektrimüüjate mobiilirakendusi. Siiski ei ole nende äppide eesmärgiks analüüsida kasutaja tarbimist või tootmist. Autorite loodud lahendus võimaldab luua kasutaja ja enda kasutajale salvestada tarbimise ja tootmise info ning elektri ostu- ja müügilepingud. Taoline lahendus teeb analüüsimise kasutajakogemuse ja rakenduse kasutamise voo mugavamaks ning kiiremaks, sest võimaldab korduvat sisestamist nõudvad andmed sisestada vaid ühe korra.

Autorite püstitatud probleem kirjeldab rakenduse puudumist, mis koondaks erinevaid funktsionaalsusi ning millel oleks võimekus põhjalikuks analüüsiks. Suuremas osas täidab autorite loodud rakendus probleemi lahendamisel eesmärki. ElectricityStat omab alternatiivsete lahendustega võrreldes oluliselt rohkem funktsionaalsusi – näiteks kasutajate registreerimine ja enda andmete andmebaasi salvestamine, börsihinna graafikul ühikute muutmine ja kuu keskmiste börsihindade kuvamine. Lisaks võimaldab rakendus mitmeid analüüsimise võimalusi: kuupõhine tarbimise analüüs/pakettide võrdlemine, tunnipõhine tarbimise analüüs/pakettide võrdlemine, tarbimisharjumuste analüüs, tootmise tulu analüüs/pakettide võrdlemine.

4.4 Disain

Rakendust arendades lähtusid autorid SOLID printsiipidest ning DRY printsiibist. Projekti mahu kasvades ning nõuete ümberkujunemise tõttu tekkis arendusfaasis mitmeid olukordi, kus autorid märkasid refaktoreerimise ja arhitektuuri abstraktsemaks muutmise võimalusi. Refaktoreerimist püüti mitte edasi lükata, sest autorid teadsid eelneva tarkvaraarenduse kogemuse põhjal, et mahu kasvades muutub rakenduse ümberkujundamine oluliselt keerukamaks ning ajamahukamaks.

4.4.1 Tagarakenduse arhitektuur

Autorite valitud tagarakenduse arhitektuur, kus mitmesugused vastutused on jagatud erinevate kihtide vahel, osutus põhjendatuks. Arenduse projekt oli valitud arhitektuuri tõttu arusaadavalt struktureeritud, mis muutis arendusvoo võrdlemisi kiireks. Suudeti

vältida ka koodi dubleerimist. Loodud kihid ei sõltu üksteisest otseselt, vaid suhtlevad omavahel liideste kaudu. Sellegipoolest kujunes arhitektuur sellisena välja arenduse käigus, sest autorid panid tähele erinevaid kitsaskohti. Koodi refaktoreerides ning arhitektuuri parendades tugineti suures osas TalTechi äriinfotehnoloogia õppekavas õpitule. Abiks olid ka töötades omandatud kogemused ning praktikad. Autorid on tagarakenduse disainiga rahul, sest kood järgib valitud tarkvaraarenduse printsiipe, väldib duplikatsioone ning on hallatav ja loetav.

4.4.2 Kasutajaliidese arhitektuur

Kasutajaliidese arhitektuur on sarnaselt tagarakendusele jaotatud erinevateks osadeks, et luua selge arhitektuur, vältida koodi duplikatsiooni ning lihtsustada ja kiirendada arendusvoogu. Kasutajaliides koosneb suures osas komponentidest ning nendest kokku pandud vaadetest. Ainuüksi taoline lähenemine rakendab DRY printsiipi ning soodustab korduvkasutatava koodi kirjutamist. Autorid jäid loodud komponentidega rahule ning suutsid neid modifitseerida nii, et neid oleks võimalik kasutada erinevates olukordades. Ka CSS koodi koondamine ühte style.css faili vältis koodi dubleerimist ning lõi eelduse kasutajaliidese ühtse disaini loomisel.

Kasutajaliidese koodis näevad autorid siiski refaktoreerimise võimalusi:

1) **Axios** - Store kausta on koondatud API-ga suhtlemiseks vajaminevad meetodid, kus kohtab hetkel koodi dubleerimist (Joonis 27). Võimalik oleks kasutusele võtta Axiose Http klient, mis võimaldaks salvestada parameetreid, nagu näiteks *headers*, *content-type* ja *authorization token* ning hiljem seda konfiguratsiooni kasutada [31].

```
// Eelnev kood
const getContract = async (contractId: string) => {
  const res = await fetch(apiUrl +
'energyData/contracts/' + contractId, {
    method: 'GET',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
      Authorization: 'Bearer ' +
localStorage.getItem('token'),
    }
  })
  const jsonData = await res.json();
  return jsonData;
}

const getProductions = async () => {
  const res = await fetch(apiUrl +
'energyData/productions', {
    method: 'GET',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
      Authorization: 'Bearer ' +
localStorage.getItem('token'),
    }
  })
  const jsonData = await res.json();
  return jsonData;
}
// Järgnev kood
```

Joonis 27. Koodi duplikatsiooni näide energyDataStore.ts-is

2) ***Composables*** - võimaldab loogikat mitmesuguste ülesannete jaoks korduvkasutada. Näiteks MonthlyAnalysis.vue ja HourlyAnalysis.vue sisaldavad mõningaid identseid meetodeid, mida oleks võimalik *composables*'i kasutades korduvkasutada (Joonis 28) [32].

```
// Eelnev kood
const updateContractsData = (data: any) => {
  contractsData.value = data;
  contractsDataSaved.value = true;
  isDataSaved();
}

const changeContractsData = () => {
  contractsDataSaved.value = false;
  isDataSaved();
}

const isDataSaved = () => {
    if(manualDataSaved.value===true &&
contractsDataSaved.value===true){
        dataSaved.value= true;
    }else{
        dataSaved.value= false;
    }
    showResponse.value= false;
}
// Järgnev kood
```

Joonis 28. Identsed meetodid kahes komponendis

Autorid jäid loodud rakenduse kasutajaliidese disaini ja arhitektuuriga rahule ning teadvustavad esinevaid kitsaskohti, mida kavatsetakse tulevikus lahendada.

4.4.3 Testimine

Üheks mittefunktsionaalseks nõudeks olid autorid seadnud asjaolu, et rakenduse funktsioonid on testitud. Selle jaoks lõid autorid ühiktestide projekti, mille eesmärgiks oli testida tagarakenduse meetodeid. Integratsioonitestide näol loodi Postman rakenduse abil kollektsioon API *endpoint* testidest, mis kontrollisid API ja tagarakenduse kontrollerite toimimist ning andmebaasi ühendust. Kasutajaliidest testisid autorid manuaalselt ning samuti anti rakendus kasutada projektivälistele isikutele *blackbox* stiilis testimise näol. Projekti algusfaasis kasutati vaid Postman integratsiooniteste, kuid üpris kiirelt kujunes vajadus ühiktestide järele, sest integratsioonitestidega ei tulnud väiksemate meetodite vead esile - refaktoreerimise käigus võttis koodist vigade leidmine oluliselt rohkem aega kui planeeritud. Selle jaoks loodi projekti keskfaasis ühiktestide projekt ning kirjutati testid ka juba valmis arendusosadele. See aeglustas mõneks päevaks arenduse tempot, sest tagantjärele testide kirjutamine võttis aega. Siiski muutus edasine arendus selle töö tulemusena oluliselt lihtsamaks ning vead leiti kiiremini.

Manuaalse testimise abil vaadati üle arenduse osade valmimisel erinevad vaated ja analüüside tulemused. Vigade leidmisel need parandati. Juhul, kui viga ei olnud kasutajaliideses, kirjutati antud vea edaspidise tekkimise välistamiseks ühik- või integratsioonitestid.

Projekti lõpufaasis anti rakendus kasutada potentsiaalsetele kasutajatele ning juhendajale. Tagasisidena toodi välja mitmed puudujäägid. Oli raskusi arusaamisega, milliseid andmeid peab rakenduse vormidesse sisestama ja kuidas tõlgendada tulemusi. Lisaks jäi arusaamatuks, millises formaadis csv-faili peaks kasutaja üles laadima. Samuti anti tagasisidet visuaalse poole ja kasutajakogemuse kohta. Kogu tagasisidet arvesse võtta ei jõutud, kuid parandati rakenduse üldist visuaalselt poolt, kasutajakogemust ja erinevates vormides olevat sõnastust. Ühe lisana soovisid kasutajad, et oleks võimalik kahe elektrilepingu sisestamisel valida üks olemasolevatest ning teine sisestada käsitsi. Algselt oli võimalik valida mõlemad olemasolevatest või sisestada mõlemad käsitsi. Antud idee realiseeriti.

Rakendus ei ole automaattestidega täielikult testitud, kuid autorid on testitud osa mahuga rahul, sest kaetud on kõik suuremad äriloogika meetodid ja projekti omavaheliste komponentide suhtlus. Algselt oli planeeritud äriloogikat katta vaid integratsioonitestidega, seega ühiktestide lisamine oli juba rohkem, kui algselt planeeritud.

Autorid on ka rahul valitud testimise meetoditega. Ühik- ja integratsioonitestid andsid enesekindlust arendamiseks ning refaktoreerimiseks, muutes sealjuures arenduse kiiremaks. Potentsiaalsete kasutajate testimine tõi välja ärilised kitsaskohad ning andis võimaluse dünaamiliselt muuta projekti teekaarti ning prioriteete.

4.5 Hinnang projekti teostamisele

Projekti arenduseks kulus autoritel kolm kuud: veebruari algusest mai keskpaigani. Projekti juhendajaks oli Tallinna Tehnikaülikooli külalisõppejõud Bahdan Yanovich. Juhendajaga suheldi peamiselt läbi MS Teamsi kõnede. Kui autorid olid töö käigus jõudnud etappi, mida sooviti juhendajale näidata või saada mõtteid ja abi, võeti omavahel ühendust ning lepiti kokku koosolekud.

Autorid töötasid välja projekti idee veebruari alguses. Esimese kuu aja jooksul uuriti valdkonna kohta iseseisvalt. Selle aja jooksul uuriti erinevaid olemasolevaid lahendusi. Lisaks pidasid autorid mitmeid koosolekuid, kus arutati täpsemalt, mida soovitakse saavutada ning milliseid tehnoloogiaid selle realiseerimiseks plaanitakse kasutada. Koosolekute käigus kujunes välja üldine teekaart projekti jaoks. Autorid panid kirja esimesed üldised kasutuslood ning need prioriseeriti olulisuse järgi. Märtsi alguses alustati esimest arenduse sprinti. Sprindi pikkuseks lepiti kokku üks nädal ning kokku lõpetati üheksa sprinti. Autorid kasutasid modifitseeritud agiilset *Scrum*'i, kus otsene rollide jaotus puudus. Mõlemad liikmed olid nii tootejuhi, arendaja kui ka testija rollis. Seetõttu tuli omavahel oluliselt rohkem suhelda ja otsuseid koos vastu võtta. Arenduse tempo hoidmiseks peeti oluliseks võimalikult kiiresti üksteise tehtud töö üle vaadata. Enda loodud koodi testimisega tegeles iga liige üldjuhul iseseisvalt, kuid potentsiaalselt ohtlikud kohad vaadati siiski koos üle. Autorid on varasemates projektides kasutanud sarnast protsessi ning seetõttu olid võimalikud kitsaskohad sellise lähenemise puhul teada. Põhiliseks ohuks antud lähenemise puhul oli omavaheline möödarääkimine nõuete osas, kuid seda suudeti vältida.

Sprintide alguses koostasid autorid piletid GitLab keskkonda ja arutasid, mida iga pileti raames tegema peab. Seejärel jaotati ülesanded vajalike tegevuste järjekorras ning jagati piletid. Kui töö käigus tekkis olukordi, kus mõne ülesande täitmisega läks kauem aega, prooviti teisele meeskonnaliikmele leida ülesanne, mis ei segaks pikale läinud töö lõpetamist. Antud lähenemisega jäid autorid rahule, sest omavaheline suhtlus oli tihe ning ülesannete järjekorda suudeti kiirelt muuta. Samuti lubas iga sprindi alguses läbiviidav *sprint planning* dünaamiliselt muuta rakenduse teekaarti ning vajadusel nõudeid ümber kujundada.

Projekti õnnestumiseks andis eelduse hea ajaplaneerimine. Autorid panid projekti alguses paika plaani, mis kuupäevaks mingi osa valmima peab. Projekti liikmed pidasid ajakavast kinni ja seetõttu ei tekkinud kiirustamist. Kõik sprindid olid ühesuguse ülesehitusega ja oli teada kui palju aega mõlemad liikmed igal sprindil panustama peavad.

Kõik projekti alguses välja töötatud nõuded ei saanud täidetud. Autorid olid varasemalt teadvustanud, et projekti vältel võivad nõuded muutuda ning sellega arvestati. Kokkuvõttes realiseeriti rakenduses mõned planeeritud funktsionaalsused teistmoodi. Mõned funktsionaalsused lisati ning mõned lükati edasi tulevikku.

Loodud rakendus suudab mineviku andmete põhjal analüüsida kasutajate elektritarbimist ning võrrelda erinevaid elektripakette. Selle abil on võimalik valida kõige otstarbekam pakett. Hetke projektifaasis ei ole võimalik arvutada seadmete tasuvusaega, kuid on võimalik leida igakuine kasum toodetud elektrienergia pealt. Samuti on võimalik kasutajal jälgida elektri börsihindu ning kuude keskmiseid börsihindu. Mitmed erinevad funktsionaalsused on koondatud ühte rakendusse, seetõttu on autorid arvamusel, et töö eesmärgiks püstitatud probleem sai suures osas lahendatud ja jäävad realiseeritud funktsionaalsusega rahule.

4.6 Probleemid ja katsumused

Projekti vältel suudeti arendusprotsessis suuremaid probleeme vältida. Kuna autorid olid ka varem korduvalt koos töötanud, võis eeldada, kuidas omavaheline koostöö toimib ning millised võivad olla kitsaskohad. Arenduse kõige suuremad katsumused tekkisid erinevate projektide ja teenuste seadistamisega. Oodatust rohkem aega kulus GitLabi *pipeline*'i, Dockeri konteinerite ja ühiktestide projekti ülesseadmisele ja konfigureerimisele. Põhjuseks vähene varasem kokkupuude antud projektide ja teenuste seadistamisel.

4.7 Võimalused edasiarenduseks

Mõningad algselt planeeritud funktsionaalsed nõuded jäid täitmata ning edaspidi tuleks panna rõhku nende arendusele.

Realiseerimata nõuded:

- Luua liidestus mõne ilmajaama API-ga, mille abil prognoosida võimalikku tootmist päikesepaneelide või tuulegeneraatoriga.
- API *endpointi*'i loomine elektri tarbimise ja/või tootmise automatiseerimiseks.
- Tootmis- või salvestusseadmete tasuvusaja arvutamine.
- Registreeritud kasutajate võimalus salvestada erinevaid tarbimis- ja tootmisandmeid asukohapõhiselt.

Selleks, et luua liidestus ilmajaamaga ning selle abil anda kasutajale prognoos võimaliku tootmise kohta, oleks vaja arendada üldist tulevikku vaatavat analüüsi lahendust. Praeguses projekti faasis keskenduti kasutaja tarbimisharjumuste analüüsile ning tegeleti minevikus kogutud andmetega.

API *endpoint*'i loomine automatiseerimise lahenduseks tuleks arendada koostöös mõne seadme tootjaga, kes sellist teenust vajaks. Sellisel juhul oleks konkreetsed nõuded teada, et seade saaks automaatselt tööd teha.

Ühe funktsionaalsusena on võimalik lisada kasutajatele asukohapõhise tarbimise salvestamine. See annaks kasutajale võimaluse analüüsida iga elektrivõrku ühendatud kohta eraldi. Hetkel on võimalik rakendust kasutada vaid ühe tarbimiskoha põhiselt.

Tootmis- ja/või salvestusseadmete tasuvusaja arvutamise funktsiooni loomine on kindlasti üks prioriteetsemaid, sest realiseerimata nõuetest suudab meeskond selle tõenäoliselt kõige kiiremini implementeerida.

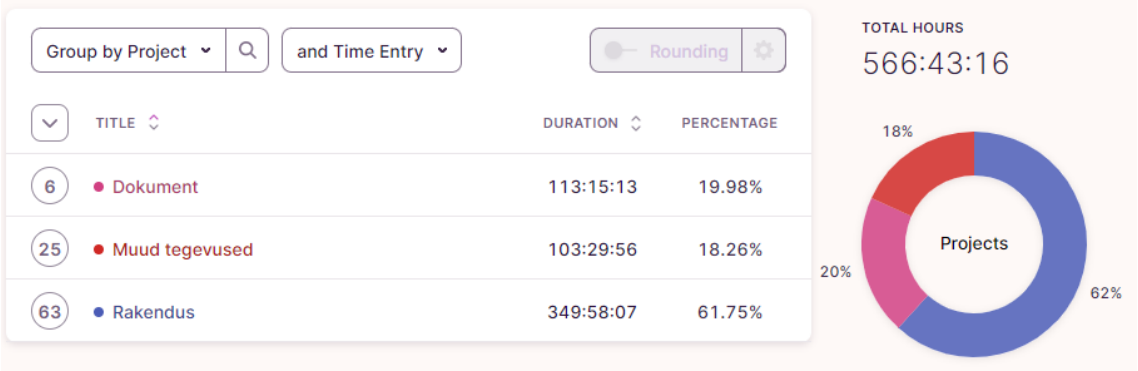
Tulevikus tuleks luua sobiv kasutajaliides kõikidele seadmetele (nutitelefon, tahvelarvuti). Praeguses projekti etapis on kasutajal rakendust mugav kasutada vaid arvuti veebibrauseris, kusjuures rakendus käitub samamoodi kõikides laiemalt kasutatud brauserites ning operatsioonisüsteemides. Sellegipoolest näevad autorid veel mitmeid kitsaskohti ka praeguses kasutajaliideses. Kasutaja vajab juhendeid ning selgitusi, mida erinevad analüüsivõimalused pakuvad ning kuidas andmeid sisestada ja tulemusi tõlgendada. Lisaks on vaja luua juhend, kuidas ja kust laadida alla csv-fail, mida analüüs toetab (hetke faasis toetab rakendus vaid Eleringi iseteenindusest alla laaditud faili).

Rakendus sobiks kasutamiseks ka töölauarakenduse või äpina. Töölauarakendusena kasutades võiks rakendus kiiremini reageerida, sest see sõltuks interneti ühendusest ainult hinnaandmete pärimisel.

Praeguses faasis on rakendus sobiv minevikus toimunud tarbimise ja tootmise analüüsimiseks. Edaspidi tuleks arendada seda, et rakendus suudaks kasutajale anda adekvaatseid soovitusi tuleviku tarbimise ja tootmise kohta ning kasutajakogemus oleks üldiselt mugavam ja lihtsama vooga.

4.8 Projekti logid ja meeskondlik hinnang

Autorid hindavad üksteise panust võrdselt. Mõlemad liikmed tegelesid nii nõuete väljatöötamise, tarkvaraarenduse kui ka testimisega. Omavaheline koostöö oli väga hea ja suhtlus toimis probleemideta. Autorid said olla kindlad kokkulepetest kinnipidamises. Lõputööle ja -projektile pühendatud aega logiti Toggl Track veebirakenduses. Kokku logisid autorid 566 tundi (Joonis 29). Erinevate osadena logiti aega kolme erineva ülesande alla. Rakenduse alla logiti kogu arendusele kulunud aeg, dokumendi alla projekti dokumentatsiooni kirjutamise aeg ning muud tegevused hõlmasid projekti jaoks teostatud analüüsi, koosolekuid, kõnesid ning jooksvaid arutelusid.



Joonis 29. Ajalogid

4.8.1 Kalle Pilli ajalogide sisuline kokkuvõte

Kalle Pilli tegevused kalendrinädala kaupa on kirjeldatud järgnevas tabelis (Tabel 2).

Tabel 2. Kalle Pilli ajalogide sisuline kokkuvõte

Nädal	**Tegevused**
6	• Analüüs ja valdkonnaga tutvumine
7	• Arutelud autorite vahel • Nõuete väljatöötamine
8	• Arutelud autorite vahel • Nõuete väljatöötamine • Tehnoloogiliste tööriistade valimine
9	• Lõpuprojekti dokumentatsiooni ülesseadmine • Rakenduse projektide ülesseadmine

10	• Kasutajaliidese põhja loomine • csv-faili üleslaadmine
11	• csv-faili töötlemine tagarakenduses • Parameetrite sisestamine tarbimisharjumuste analüüsiks
12	• Elektri tarbimise ja tootmise manuaalne lisamine • Analüüsi tulemuste loogika väljatöötamine • Nordpool börsihindade laadimine rakenduse käivitamisel • Andmebaasi loomine
13	• Refaktoreerimine: äriloogika eraldamine kontrollerist, Repositories kihi loomine • Andmebaasi loomine ning rakenduse käivitamisel Nordpool hindade salvestamine
14	• Dockeri seadistamine • Lõpuprojekti dokumentatsioon
15	• Ühiktestide projekti loomine ja seadistamine • Manuaalselt sisestatud tarbimise analüüs • Testide kirjutamine varem loodud lahendustele • Lõpuprojekti dokumentatsioon
16	• DateTimeHelpers: UTC ja EET aja sünkroniseerimine • PostgreSQL andmebaas • Lõpuprojekti dokumentatsioon
17	• Kasutajate süsteemi loomine • Testide kirjutamine • Kasutajate tarbimise CRUD • Lõpuprojekti dokumentatsioon
18	• Refaktoreerimine ja vigade parandamine • Kasutajaliidese arendamine • Lõpuprojekti dokumentatsioon
19	• Lõpuprojekti dokumentatsioon • Testide kirjutamine

20	• Lõpuprojekti dokumentatsioon

4.8.2 Rene Treieri ajalogide sisuline kokkuvõte

Rene Treieri tegevused kalendrinädala kaupa on kirjeldatud järgnevas tabelis (Tabel 3).

Tabel 3. Rene Treieri ajalogide sisuline kokkuvõte

Nädal	**Tegevused**
6	• Analüüs ja valdkonnaga tutvumine
7	• Arutelud autorite vahel • Nõuete väljatöötamine
8	• Arutelud autorite vahel • Nõuete väljatöötamine • Tehnoloogiliste tööriistade valimine
9	• Lõpuprojekti dokumentatsiooni ülesseadmine • Rakenduse projektide ülesseadmine • Börsihindade diagrammi arendus
10	• Börsihindade diagrammi täiustamine • Ajavööndite arendus
11	• Börsihindade diagrammi täiustamine • Elektrilepingute sisestamise arendus
12	• Kasutajaliidese .css refaktoreerimine • csv-üles laadimise muutmine
13	• GitLab *pipeline*'i seadistamine
14	• Dockeri seadistamine • GitLab *pipeline*'i seadistamine • Lõpuprojekti dokumentatsioon
15	• Ühiktestide lisamine *pipeline*'i • Eleringi API ühildamise muutmine • Lõpuprojekti dokumentatsioon

16	• Tootmise analüüsi arendamine • Lõpuprojekti dokumentatsioon
17	• Tootmise analüüsi arendamine • Elektripakettide võrdluse lisaarendused • Lõpuprojekti dokumentatsioon
18	• Lõpuprojekti dokumentatsioon • Kasutajaliidese täiustamine • *Service*'ite loomine
19	• Testide kirjutamine • Lõpuprojekti dokumentatsioon
20	• Lõpuprojekti dokumentatsioon

Kokkuvõte

Bakalaureusetöö eesmärgiks oli luua rakendus, mis koondaks erinevaid elektri tarbimise ja tootmise analüüsimise võimalusi ning seeläbi tõstaks kasutaja teadlikkust enda tarbitavast ja toodetavast elektrist.

Selleks loodi kolmest komponendist koosnev veebirakendus, kuhu kuulusid C# programmeerimiskeeles ning .NET raamistikus loodud tagarakendus, TypeScript programmeerimiskeeles ja Vue.js raamistikul põhinev kasutajaliides ning PostgreSQL andmebaas. Lahendus paigutati Docker konteinerisse. Kasutati agiilset arendusmetoodikat, kus kombineeriti Scrumi ja *Extreme Programming*'u protsesse, põhimõtteid ja printsiipe.

Loodud rakendus võimaldab jälgida Nordpool börsihindu, analüüsida nii elektrienergia kuu- ja tunnipõhist tarbimist, tarbimisharjumusi kui ka tootmise tulu. Samuti on võimalik registreeruda kasutajaks ning salvestada enda elektri ostu- ja müügilepinguid ning elektrienergia tarbimist ja tootmist, et neid hiljem analüüsides kasutada.

Loodud veebirakendus koondab võrreldes alternatiivsete lahendustega oluliselt rohkem funktsionaalsusi ning analüüsimise võimalusi. Seetõttu loeme eesmärgi täidetuks. Kuna mitmed algselt planeeritud funktsionaalsused jäid nõuete ümber kujundamise tõttu realiseerimata, annab see võimaluse rakendust tulevikus edasi arendada.

Kasutatud kirjandus

[1] „Majandus,“ AS Postimees Grupp, 2022. [Võrgumaterjal]. Available: https://majandus.postimees.ee/7663083/uuring-inimeste-teadlikkus-energiaturust-on-kasvanud. [Kasutatud 1 Aprill 2023].

[2] „ERR Uudised,“ Eesti Rahvusringhääling, [Võrgumaterjal]. Available: https://www.err.ee/search?phrase=elekter&from=01.04.2022&to=01.04.2023&page=1. [Kasutatud 1 Aprill 2023].

[3] „ERR Uudised,“ Eesti Rahvusringhääling, [Võrgumaterjal]. Available: https://www.err.ee/search?phrase=elekter&from=01.04.2020&to=01.04.2021&page=1. [Kasutatud 1 Aprill 2023].

[4] „MKM,“ Majandus- ja Kommunikatsiooniministeerium, 15 Detsember 2022. [Võrgumaterjal]. Available: https://mkm.ee/universaalteenus. [Kasutatud 1 Aprill 2023].

[5] „Atlassian,“ Atlassian, [Võrgumaterjal]. Available: https://www.atlassian.com/agile/scrum. [Kasutatud 10 Aprill 2023].

[6] K. Beck ja C. Andres, Extreme Programming Explained Second Edition, Addison-Wesley Professional, 2004, pp. 13-55.

[7] „Elektrikell.ee,“ [Võrgumaterjal]. Available: https://www.elektrikell.ee/. [Kasutatud 25 Märts 2023].

[8] „MacDronic,“ Macdronic OÜ, [Võrgumaterjal]. Available: https://home.macdronic.com/et/elekter/live. [Kasutatud 2 Aprill 2023].

[9] „Elektrikulu kalkulaator,“ [Võrgumaterjal]. Available: https://elekter.pere-eelarve.eu/. [Kasutatud 2 Aprill 2023].

[10] „Elektitarbimise kalkulaator,“ [Võrgumaterjal]. Available: https://kalkulaatorid.net/elektritarbimise-kalkulaator. [Kasutatud 2 Aprill 2023].

[11] „Elektrihind,“ Go OÜ, [Võrgumaterjal]. Available: https://elektrihind.ee/paketid. [Kasutatud 2 Aprill 2023].

[12] „Eesti Energia,“ App Store, [Võrgumaterjal]. Available: https://apps.apple.com/ee/app/eesti-energia/id898742562. [Kasutatud 25 Aprill 2023].

[13] „Alexela,“ App Store, [Võrgumaterjal]. Available: https://apps.apple.com/us/app/alexela/id1549876996?ign-mpt=uo%3D2. [Kasutatud 25 Aprill 2023].

[14] „220 Energia,“ App Store, [Võrgumaterjal]. Available: https://apps.apple.com/ee/app/220-energia/id1559042675#?platform=iphone. [Kasutatud 25 Aprill 2023].

[15] „Light Bulb Saver,“ App Store, [Võrgumaterjal]. Available: https://apps.apple.com/us/app/light-bulb-saver/id1095939669. [Kasutatud 20 Aprill 2023].

[16] „Meter Readings,“ App Store, [Võrgumaterjal]. Available: https://apps.apple.com/us/app/meter-readings/id320551309?platform=iphone. [Kasutatud 20 Aprill 2023].

[17] „Energy Tracker,“ App Store, [Võrgumaterjal]. Available: https://apps.apple.com/us/app/energy-tracker/id1193010972. [Kasutatud 20 Aprill 2023].

[18] M. Martin, „What is a Functional Requirement in Software Engineering?,“ Guru99, 8 Aprill 2023. [Võrgumaterjal]. Available: https://www.guru99.com/functional-requirement-specification-example.html#:~:text=A%20Functional%20Requirement%20(FR)%20is,%2C%20its%20behavior%2C%20and%20outputs. [Kasutatud 24 Aprill 2023].

[19] „Jama Software,“ Jama Software, [Võrgumaterjal]. Available: https://www.jamasoftware.com/requirements-management-guide/writing-requirements/how-non-functional-requirements-impact-product-development. [Kasutatud 20 Aprill 2023].

[20] R. C. Martin ja M. Martin, Agile Principles, Patterns, and Practices in C#, Pearson, 2006.

[21] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Massachusetts: Pearson Education, Inc, 2008.

[22] Elering, „Elering dashboard API documentation,“ Elering, [Võrgumaterjal]. Available: https://dashboard.elering.ee/assets/api-doc.html. [Kasutatud Märts 15 2023].

[23] „TypeScript Documentation,“ [Võrgumaterjal]. Available: https://www.typescriptlang.org/docs/. [Kasutatud 20 Märts 2023].

[24] „Vue.js,“ [Võrgumaterjal]. Available: https://vuejs.org/guide/introduction.html. [Kasutatud 20 Märts 2023].

[25] „DigitalOcean,“ 21 September 2021. [Võrgumaterjal]. Available: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design. [Kasutatud 16 Aprill 2023].

[26] A. Hunt ja D. Thomas, The Pragmatic Programmer, Addison Wesley Longman, Inc., 1999.

[27] E. Evans, Domain-Driven Design, Addison-Wesley Professional, Inc. , 2003.

[28] „.NET,“ Microsoft, 2 Veebruar 2023. [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/. [Kasutatud 1 Aprill 2023].

[29] „Docker overview,“ Docker, [Võrgumaterjal]. Available: https://docs.docker.com/get-started/overview/. [Kasutatud 3 Aprill 2023].

[30] „CI/CD pipelines,“ GitLab, [Võrgumaterjal]. Available: https://docs.gitlab.com/ee/ci/pipelines/. [Kasutatud 10 Aprill 2023].

[31] „Axios,“ [Võrgumaterjal]. Available: https://axios-http.com/docs/instance. [Kasutatud 20 Aprill 2023].

[32] „Composables,“ Vue.js, [Võrgumaterjal]. Available: https://vuejs.org/guide/reusability/composables.html. [Kasutatud 1 Mai 2023].

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

Autorid Kalle Pilli ja Rene Treier

1. Annavad Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Elektrienergia tarbimise ja tootmise analüüsi veebirakendus“, mille juhendaja on Bahdan Yanovich
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Oleme teadlikud, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

16.05.2023

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 – Kalle Pilli eneseanalüüs

Lõpuprojekti arendasime Rene Treieriga kahekesi ning projekti arenduse vältel panustasime mõlemad kõikidesse projekti etappidesse ja osadesse. Siiski tooksin välja minupoolse suurima panuse projekti:

- Nõuete väljatöötamine
- Tagarakenduse, kasutajaliidese ja ühiktestide projekti ülesseadmine
- Kood kõikides rakenduse osades
- Kasutajate ja autentimise süsteemi loomine
- Dokumentatsioon

Projekti arendus ja selle kallal töötamine oli huvitav, sest saime nõuded ise välja töötada ning luua rakenduse, mida ise vajalikuks peame. Alternatiivseid lahendusi uurides jõudsin arusaamale, et sellist rakendust on ka päriselt vaja ning see võib inimestele tõelist väärtust pakkuda.

Olen rahul enda panusega lõpuprojekti ja ka valitud meeskonnakaaslasega. Oleme Renega teinud koos mitmeid koolitöid ning -projekte alates esimesest semestrist ning seeläbi tundma õppinud nii üksteise tugevusi kui ka nõrkusi. Selle aja jooksul oleme moodustanud hästi kokku töötava meeskonna, mis lõi eeldused projekti edukaks lõpuleviimiseks.

Projekt oli võrdlemisi keerukas, sest välja tuli töötada äriloogika, mis pakuks kasutajale mingisugust väärtuslikku analüüsi ja annaks kasulikku tagasisidet tema elektri tarbimise ja/või tootmise kohta. Seetõttu oli nõuete väljatöötamine ja projekti skoobi defineerimine paras väljakutse. Projekti käigus skoop ning nõuded ka muutusid: lisandusid mõned planeerimata funktsionaalsused, kuid mõned plaanitud funktsionaalsused lükati edasi tulevikku.

Lõpuprojekti arendamine sobib hästi ka tuleviku karjäärieesmärkidega, sest soovin lähiaastatel töötada tarkvaraarendajana. Enda nõrkuseks olen siiani pidanud kasutajaliidese arendamist, kuid nüüd tunnen end tänu projekti jooksul õpitule oluliselt

kindlamalt ka TypeScriptis ja Vue.js raamistikku kasutades. Tagarakenduse arenduse poole pealt oli arendav tegeleda autentimise ja kasutajate süsteemi üles ehitamisega, mis andis väärtuslikke teadmisi JWT *token*i kasutamise ja privaatsuse ning turvalisuse kohta. Lisaks sain teadmisi ka CI/CD protsessidest ning ülesseadmisest. Kuigi sellega tegeles projekti käigus põhiliselt Rene, jagas ta minuga saadud õppetunde, väljakutseid ning teadmisi.

Kokkuvõttes arvan, et projekt kujunes edukaks ning minu panus sellesse oli võrdväärne teise meeskonnaliikme Rene Treieriga. Motivatsioon oli kõrge terve projekti vältel ning nähtavad tulemused innustasid veelgi arendusse ja tarkvara parendamisse panustama.

Lisa 3 – Rene Treieri eneseanalüüs

Lõpuprojekti teostasime kahekesi Kalle Pilliga, kellega olime projekti vältel kõikides arendusmeeskonna rollides. Suurim panus projekti oli:

- Nõuete väljatöötamine
- GitLab *pipeline* üles seadmine koos ühik- ja integratsioonitestidega
- Rakenduse Docker konteinerisse paigutamine
- Kood kõikides rakenduse osades
- Dokumentatsioon

Projekti teemavalik oli mulle väga südamelähedane, sest jälgin igapäevaselt oma elektritarbimist. Samuti on mul päikesepaneelid ja jälgin ka neid tootmisandmeid igapäevaselt. Elektriinfoga kursis olemine aitas meil projekti vältel oluliselt vähema vaevaga välja mõelda arvutuskäike ja ärilisi nõudeid, milleni soovisime jõuda. Projekti keskpaigas oli rakendus jõudnud arendusjärku, kus osad funktsionaalsused olid juba realiseeritud. See aitas mind juba oma elektriinfo analüüsimisel, mida muidu tegin Exceli abil.

Projektikaaslaseks sai valitud Kalle Pilli põhjusel, et me oleme varem korduvalt koostööd teinud ja teame üksteise oskustest ning motivatsioonist ning saame kindlad olla üksteise peale. Seetõttu ei tekkinud meil kordagi omavahel probleeme. Projekti jooksul tekkis mitmeid kordi mõlemal olukordi, kus olime kinni jooksnud oma ülesande täitmisel, kuid väga lihtne oli üksteiselt abi küsida ja paarisprogrammeerimise raames leida lahendusi. Olen väga rahul meeskonnakaaslase valikuga.

Projekti kallal töötamine oli väga huvitav, sest selle käigus saime teha nullist uue rakenduse. Kuna olime valinud lähenemiseks väga agiilse arenemise, siis iganädalaselt oli põnev töötada projekti kallal, kus iga nädala lõpus võis tulla uusi ideid ja lähenemisi, et mida järgmisena ette võtta. Kuigi algselt olid välja mõeldud funktsionaalsused, mis soovisime projekti käigus valmis saada, siis lõpuks realiseeritud tulemusega olen väga rahul. Minu jaoks isiklikult leiab rakendus palju kasutust.

Tehnilistest aspektidest rääkides sain väga palju teadmiseid CI/CD ülesseadmisest ning Docker konteinerite seadistamisest. Lisaks sellele sain väga palju kindlamaks kasutajaliidese arendamisega. Kindlasti tunnen ennast palju tugevamini just Vue.js arendamisel ja TypeScripti kasutamisel.

Jään kindlasti rahule oma panusega. Sain erinevaid rolle täites tegeleda nii äriliste nõuete välja mõtlemisega kui ka nende arendamise ja testimisega. Samuti aitas projekti käigus tehtud töö kinnistada juba varasemaid tarkvara arendamise teadmiseid, sest igapäevaselt ma tarkvara arendajana ei tööta. Kindlasti aitasid projekti õnnestumisele kaasa teadmised ja kogemused töötades tarkvaratestija ja tootejuhina.

Lisa 4 – docker-compose.yml

```
version: '3.9'

networks:
  pg_net:

services:
  postgres:
    image: postgres:14.2-alpine
    environment:
      - POSTGRES_PASSWORD=password
      - TZ="UTC"
    ports:
      - "5432:5432"
    networks:
      - pg_net
  backend:
    build:
      context: ./BackEnd
      dockerfile: Dockerfile
    environment:
      - ASPNETCORE_URLS=http://+:80
      - TZ="UTC"
    ports:
      - "5263:80"
    depends_on:
      - postgres
    networks:
      - pg_net
    restart: on-failure
    command: ["dotnet", "run", "--urls",
"http://0.0.0.0:80"]

  frontend:
    build:
      context: ./FrontEnd
      dockerfile: Dockerfile
    ports:
      - "8080:80"
    depends_on:
      - backend
    networks:
      - pg_net
```

Lisa 5 – .gitlab-ci.yml

```
stages:
  - build
  - unit_test
  - test
build:
  stage: build
  image: docker/compose:1.29.2
  script:
    - docker-compose up --build -d
    - docker-compose down
  services:
    - postgres
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"
&& $CI_MERGE_REQUEST_TARGET_BRANCH_NAME == "main"'
      when: always
Unittests:
  stage: unit_test
  image: mcr.microsoft.com/dotnet/sdk:6.0
  script:
    - cd UnitTests/ElectricityStat.Tests
    - dotnet test
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"
&& $CI_MERGE_REQUEST_TARGET_BRANCH_NAME == "main"'
      when: always
Apitests:
  stage: test
  image: docker:latest
  script:
    - apk add --no-cache nodejs npm
    - npm install -g newman
    - docker-compose up -d
    - sleep 10 #postgres needs time for initial data
    - cd ApiTests
    - newman run ElectricityTests.postman_collection.json
--env-var backendUrl=http://172.17.0.1:5263
    - docker-compose down
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"
&& $CI_MERGE_REQUEST_TARGET_BRANCH_NAME == "main"'
      when: always
```