TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Hendrik Laretei 178956IABB

Oskar Neemre 179394IABB

Risto-Raul Pajula 179161IABB

Mark Rõõmussaar 179661IABB

ETS NORD AS TOOTEKONFIGURAATORI ARENDUS JA ANALÜÜS

Bakalaureusetöö

Juhendajad: Gunnar Piho
PhD,
Viljam Puusep
MSc

Tallinn 2021

Autorideklaratsioon

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

Autor: Hendrik Laretei, Oskar Neemre, Risto-Raul Pajula, Mark Rõõmussaar

17.05.2021

Annotatsioon

ETS NORD AS tellimuse põhjal koostati katuseotsikute tootekonfiguraator, mille eesmärgiks oli filtreerida ETS NORD katuseotsikuid kindlate parameetrite põhjal ning seejärel valitud toodet kasutaja nägemise järgi edasi konfigureerida. Rakendus on ettevõttele vajalik, sest see aitab ettevõtte klientidel lihtsustada katuseotsikute väljavalimise ning konfigureerimise protsessi. Arendusmeeskond valis projekti läbiviimiseks vajaminevad tehnoloogiad, mustrid ning metoodikad iseseisvalt.

Käesolev bakalaureusetöö sisaldab projekti läbiviimiseks kasutatud meetodite ja tööriistade kirjeldust. Lisaks sellele annab töö ülevaate projekti tulemustest ja analüüsist, mis sisaldab endas projekti tehnilise teostuse põhjendust, hinnanguid ning järeldusi.

Projekti tulemusena saab rakenduse kasutaja leida endale kindlate parameetrite põhjal sobiliku katuseotsiku, konfigureerida selle omadusi ning lisada katuseotsikule juurde erinevaid kõrvalisi tooteid, mis on vajalikud katuseotsiku paigalduseks. Pärast toote konfigureerimist on kasutajal võimalik toote tellimuse dokument alla laadida. Tuginedes asjakohasele kirjandusele, analüüsitakse ning põhjendatakse, kuidas autorid püstitatud eesmärgini jõudsid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 61 leheküljel, 5 peatükki, 38 joonist, 7 tabelit.

Abstract

Development and Analysis of Product Configurator for ETS NORD AS

This bachelor thesis is based on the development and analysis of a product configurator which is a contract project ordered by ETS NORD AS. ETS NORD AS is an Estonian company that specializes in manufacturing and selling ventilation components. The goal of the project was to develop an information system which could filter and configure the product group of roof hoods based on industry specific parameters. This application is important to the company because it simplifies and accelerates the process of choosing and ordering their products. The project team had to individually choose corresponding architecture, design, technologies and methodologies to build the application without IT-guidance from the company. The information system had to be developed from ground up since the company had no usable software or database for this project.

This thesis includes description of tools and methodologies used for the project. Furthermore, it provides an overview of the results of the project and analysis of the technical implementation regarding how the application was designed and how the project was managed. Based on relevant literature the thesis analyses and answers why such architecture and design of the resulting application was developed. In addition, it includes a review of the management of this project and information on what could have been done better.

As a result of this project the user can filter the list of products based on given criteria and then configure a selected product by choosing customizable features of it. In addition, it has a functionality to add supplementary products for the selected roof hood which are necessary for the installation of the roof hood. After configuring a specific product the user can then download a PDF-document which includes details and specifications about the configured product and forward it as a purchasing order.

The thesis is in Estonian and contains 61 pages of text, 5 chapters, 38 figures, 7 tables.

Lühendite ja mõistete sõnastik

.resx	Võtme ja väärtuse paare sisaldav failitüüp
API	*Application Programming Interface*
Azure	*Microsoft Azure cloud computing service*
CI/CD	*Continuous Integration and Continuous Delivery*
Dünaamilised andmed	Muutuv informatsioon
Entity Framework Core	Teek objekt relatsiooniliseks kaardistamiseks
Hooks	React teegi funktsioonid hetkeseisu muutmiseks
IDE	*Integrated Development Environment*
JSON	*JavaScript Object Notation*
MVP	*Minimum Viable Product*
Netlify	Pilvandmetöötluse platvorm
Objekt	Abstraktne andmetüüp kindlate omadustega
Postman	Rakendus API testimiseks
Props	*Properties* – kasutatakse Reactis, et andmeid ühelt komponendilt teisele saata
Query	(HTTP) query
Redux variables	Globaalsed muutujad Reduxi jaoks
Staatilised andmed	Muutumatu informatsioon
State variables	Hetkeseisu muutujad Reactis
UI	*User Interface*
Webshop	Töövõtja infosüsteem toodete tellimiseks

Sisukord

Jooniste loetelu

Tabelite loetelu

1. Sissejuhatus

Meeskonnaprojekti tellijaks oli ettevõte ETS NORD AS, mis on suurim ventilatsiooniseadmete ja -tarvikute tootja ning müüja Eestis aastast 1998. Peale Tallinna tehase on ettevõte laiendanud oma tootmist ka Soome, kus tegutseb alates 2009. aastast. Tulenevalt ettevõtte vajadusest muuta vajamineva ventilatsiooni toote leidmine ja tellimine kiireks ja mugavaks, oli tellimuseks katuseotsikute tootegrupi konfiguraator. Projekt käivitati Tallinna Tehnikaülikooli tudengite Risto-Raul Pajula, Hendrik Laretei, Mark Rõõmussaare, Oskar Neemre ning ETS NORDi koostöös kahes etapis – meeskonnaprojekti ja lõputööna.

1.1 Probleem

Hoonete projekteerimisel on projekteerijal vaja kanda hoolt kogu hoone ventilatsioonisüsteemi toimimise eest, kuid projekti nõuetele vastavaid tooteid võib olla palju ning nende sobivuse manuaalne kontrollimine tootedokumentidelt on aeganõudev.

Ventilatsioonitoodete valikusüsteem on lahendatud manuaalselt, navigeerides läbi tootekataloogi vajaliku tootegrupini ning valides sealt konkreetse toote. Seejärel peab projekteerija laadima alla selle tootedokumendi, kus on kirjeldatud seadme standardmõõdud, materjalid ning õhuvoolu, rõhukao ja mürasumbuvuse graafikud. Võttes arvesse kõiki parameetreid, peab projekteerija endale valima sobiliku toote. Käesoleva projekti raames valmis katuseotsikute valikuprogramm tootekonfiguraatorina, mis lihtsustab kasutajal sobiliku toote leidmise ja konfigureerimise protsessi.

1.2 Eesmärk

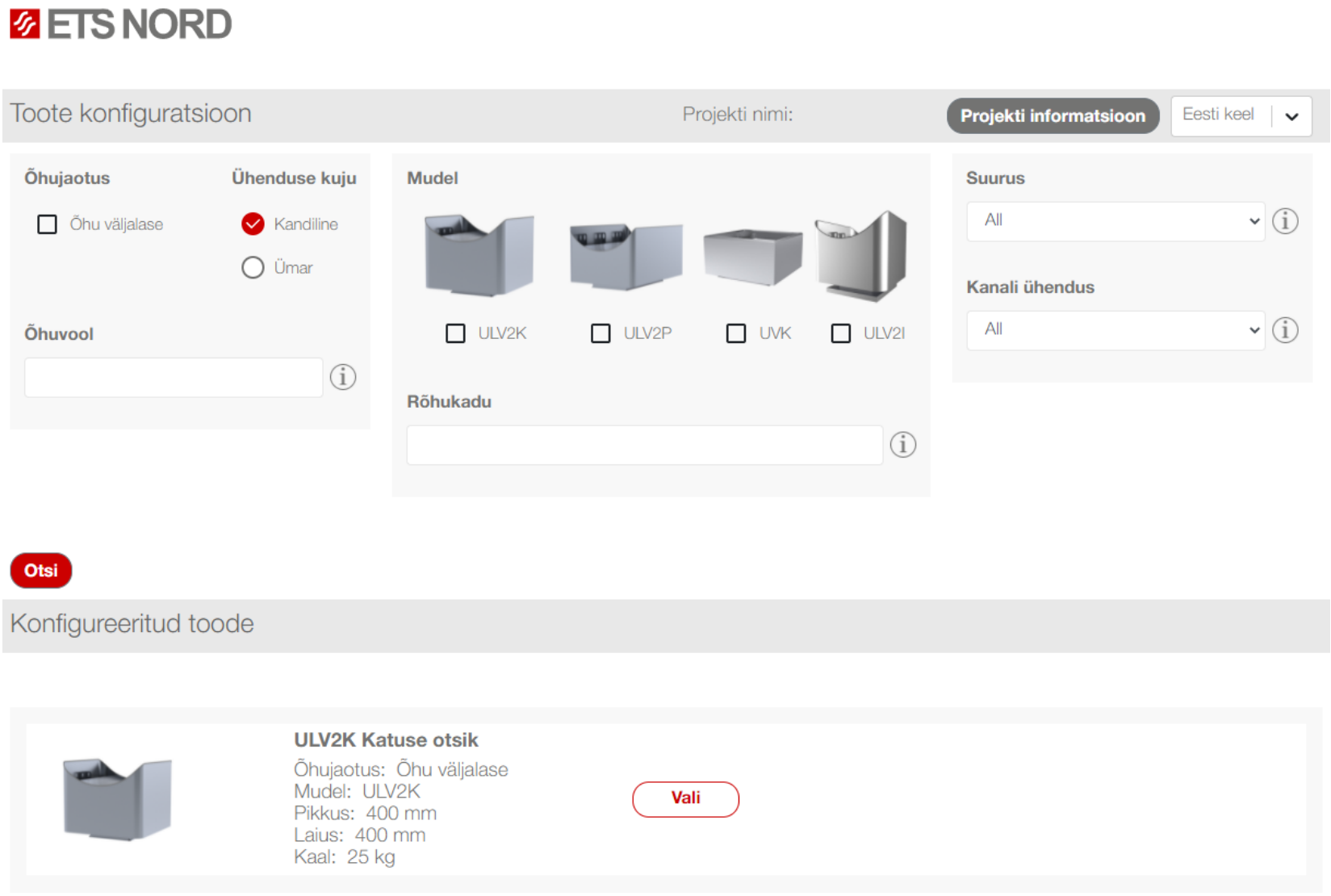
Tuleneval probleemist sai projekti peamiseks eesmärgiks katuseotsikute tootekonfiguraatori konstrueerimine ja kliendile kättesaadavaks tegemine, ühildades selle vajalike süsteemidega, pakkudes kliendile kaasaegset ja hallatavat tarkvara. Lõputöö laiendatud eesmärk on protsessi kaardistada ja analüüsida selle käigus saadud kogemusi, et pakkuda ülevaadet ja soovitusi teistele tudengitele sarnases olukorras, kui ka ülikoolile õppekava muutmiseks või konkreetsete ainete võimalikuks tegemiseks.

1.3 Teostatud funktsionaalsus

Projektil oli mitmeid funktsionaalseid ja mittefunktsionaalseid nõudeid. Meeskonnaprojekti raames teostati katuseotsikute filtreerimine erinevate parameetrite järgi, sealhulgas vastava toote leidmine sisestatud õhuvoolu või rõhukao järgi. Lisaks projekti informatsiooni lisamine tootele ning tellimusdokumendi allalaadimine PDF-dokumendina (Joonis 1). Projekti teises etapis lisandus toote edasine konfigureerimine, mis võimaldab erinevate tooteomaduste muutmist. Tootele saab lisada paigaldamiseks vajaminevaid tooteid ja ka nende omadusi konfigureerida. Rakendust saab kasutada kahes erinevas keeles. Tähtsamateks mittefunktsionaalseteks nõueteks olid eelkõige ajakohaste raamistike ja pakettide kasutamine ning universaalse arhitektuuri loomine, mida saaks potentsiaalselt rakendada hiljem ka teistel tootegruppidel.

1.4 Töö edasine struktuur

Käesolevas lõputöös antakse ülevaade teostatud projekti tulemustest ja selle analüüsist. Järgnevad peatükid millest juttu tuleb on: metoodika, projekti tulemused, analüüs ja järeldused ning kokkuvõte. Metoodika osas on räägitud täpsemalt objektist, kasutatud tööriistadest ning lahti seletatud tööprotsessi olemus. Tulemuste osas presenteeritakse projekti lõpptulem. Analüüsi ja järelduste peatükis on juureldud töö tulemuste üle, kus on lahti seletatud projekti tehniline teostus. Peale seda esitatakse ülevaade kirjandusest ning logi tehtud töödest, millele järgneb hinnang projekti tööprotsessile ja resultaadile.



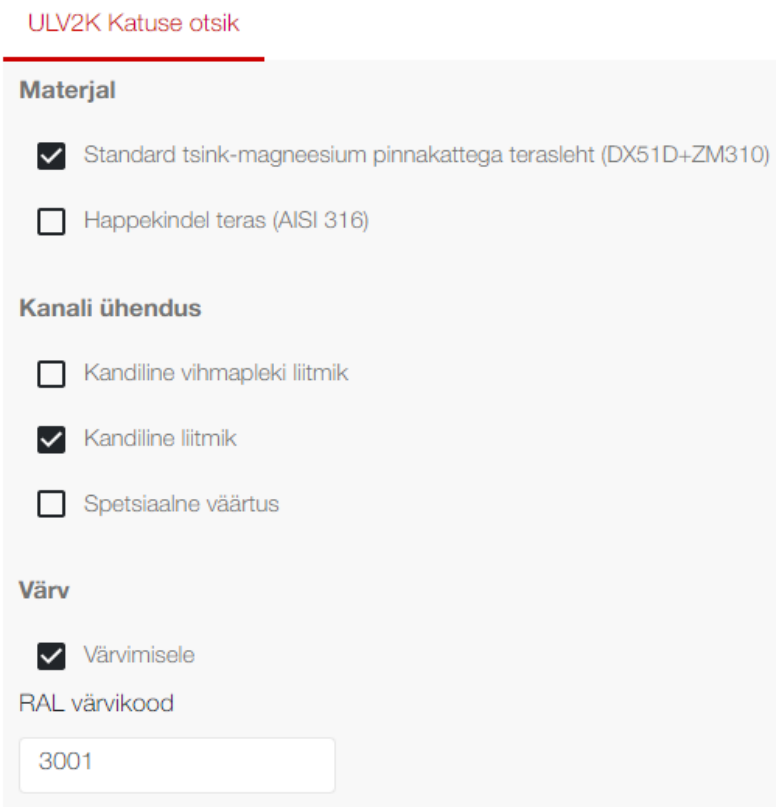
Joonis 1. Osaline avalehe vaade

2. Metoodika

2.1 Objekti detailne kirjeldus

Objektiks on toodete filtreerimise ja konfigureerimise infosüsteem. Iga toode jaguneb ETS NORDis erinevasse tootegruppi ning antud infosüsteem tegeleb ühe tootegrupiga, milleks on katuseotsikud. Tooted võivad olla erineva laiuse, pikkuse, kõrguse, funktsiooni (õhu sissevõtt või väljavise) ning võimaliku õhuvoolu ja rõhukao vahemikuga. See tähendab, et igal tootel on kindlad omadused ning toodete filtreerimine toimib nimetatud omaduste põhjal, mis võimaldab leida sobiva toote(d) (Joonis 1). Kõik võimalikud tooted on defineeritud andmebaasis. Rakendust on võimalik kasutada nii eesti kui ka inglise keeles.

Konkreetse toote muutmist võimaldab tootekonfiguraator. Lisaks filtreeritavatele omadustele on tootel ka konfigureeritavad omadused, mida saab lisada või muuta (Joonis 2). Lisaks on võimalik konkreetsele katuseotsikule juurde valida teised tooted, mis on vajalikud katuseotsiku paigaldamiseks ning juurde lisatud toodete omaduste konfigureerimine.



Joonis 2. Katuseotsiku konfigureeritavad väljad.

Meeskonna poolt arendatava katuseotsikute konfiguraatori jaoks rangeid tehnoloogilisi nõudeid ette ei antud, sest ainuke olemasolev konfiguraator oli ettevõttel varasemalt sisse ostetud ning selle kood kuulub selle loonud ettevõttele. Ligipääs varasemale koodile puudus, mistõttu said autorid vabaduse valida arendusplatvormide, tehnoloogiate ning praktikate vahel. Ettevõtte poolt saadi sisendiks ETS NORDi kodulehel olev katuseotsikute tootekataloog koos iga toodet kirjeldava dokumendiga. Konfiguraatori väljanägemise ja funktsionaalsuse eeskujuks pakuti välja teise tarkvaraettevõtte poolt loodud tootekonfiguraator ning spetsiifilisemad stiilinõuded sisaldusid ETS NORD *brand book*'is ning Adobe XD prototüübis. Ettevõttel puudus kõikide toodete informatsiooni sisaldav andmebaas ning ka ettevõtte siseselt kasutatavad Exceli failid ning raamatupidamistarkvara polnud projekti arenduse vajadustele lihtsasti rakendatavad. Andmebaasi loomine ja toodete sisestamine jäi seega samuti arendusmeeskonna ülesandeks.

2.2 Tööriistade kirjeldus

Projekti alguses edastas klient meeskonnale ekraanipildid konkureeriva ettevõtte veebilehest koos selgitavate tekstilõikudega. Ekraanipildid tõlgendati tudengite poolt kasutajalugudeks ja nõueteks. Projekti arendamiseks kasutati erinevaid raamistikke ja arenduskeskkondi. Meeskond otsustas, et serveripoolne rakendus valmib .NET CORE 3.0 API-na (*Application Programming Interface*) kasutades C# programmeerimiskeelt Visual Studio 2019 IDE-s (*Integrated Development Environment*). Andmeid säilitatakse Azure SQL andmebaasis, mille loomiseks kasutati Entity Framework Core-i. Serveri rakendus on paigaldatud ja käivitatud Azure App Service'is. See suhtleb omakorda käivitamisel Google Sheets API'ga, kust laetakse andmebaasi tabelitesse tooteandmed.

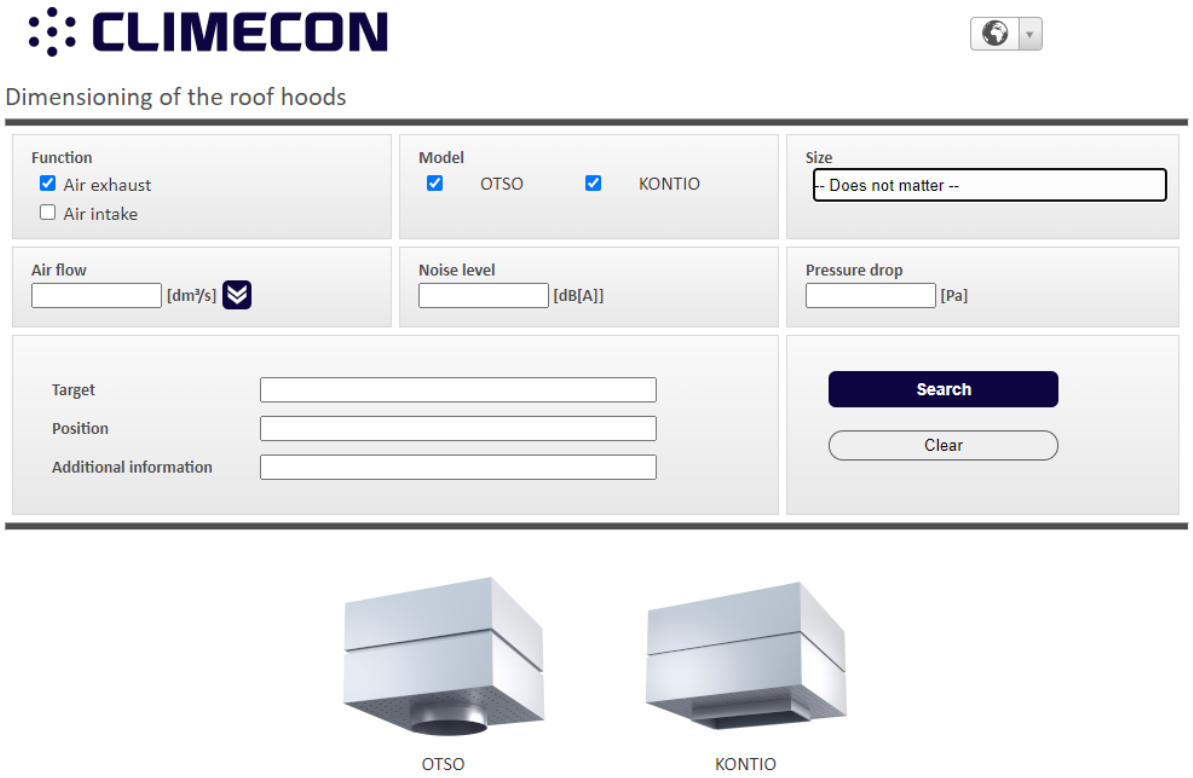
Projekti kasutajaliides realiseeriti React teegis TypeScript keeles kasutades Visual Studio Code IDE-d. Kasutajaliidese baaskomponentideks kasutati React Bootstrap 4 kasutajaliidese teeki, mille kujundust muudeti CSS'i abil. Kasutajaliidese rakendust hoiustatakse Netlify veebiteenuse abil.

Versioonikontrolliks nii koodi üleslaadimisel kui projekti planeerimisel kasutati GitLab'i ja Azure DevOps'i. Tööaja kaardistamiseks kasutati Toggl'it. Arendamisel järgiti agiilse arenduse põhimõtteid tagades kliendi rahulolu, olles valmis muudatusteks ja tarnides kliendile tarkvara võimalikult sageli. Kasutajaliidese graafilise disaini prototüübi järgimiseks kasutati Adobe XD'd ja ETS NORD *brand book*'i. Koodi refaktoorimisel kasutati ReSharperit. Projekti

testimiseks kirjutati vastuvõtutestid Postman'is, kus kontrolliti päringute õigsust. Samuti kirjutati integratsiooniteste ja ühikteste MSTest testimisteegis, et testida andmebaasi kui ka rakendust.

2.3 Tööprotsessi kirjeldus

Tööprotsess algas ETS NORD ventilatsiooniseadmete ja -tarvikute tutvumisega. Järgnevalt selgitas ettevõtte esindaja meeskonnale lühidalt probleemi ning tutvustas meeskonnale konkureeriva ettevõtte katuseotsikute tootekonfiguraatorit. Algselt pidi lõpptulemus olema funktsionaalselt väga sarnane sellele, milline oli konkurendi katuseotsikute tootekonfiguraator. Klient andis meeskonnale ekraanipildid konkureeriva ettevõtte tootekonfiguraatorist (Joonis 3), mis kirjutati autorite poolt kasutajalugudeks. Antud materjalide põhjal arutati meeskonna ning juhendajaga, kuidas peaks välja nägema projekti arhitektuur, disain ning meeskonna töökorraldus. Ettevõte ei esitanud projektile tehnoloogilisi nõudeid, mistõttu pidid tudengid ise otsustama millistes raamistikes, keeltes ja keskkondades projekti arendama hakata. Projekti disaini aluseks võeti J. Arlow ja I. Neustadt *Product* arhetüüp [1].



Joonis 3. Konkureeriva ettevõtte tootekonfiguraatori avaleht [2].

Meeskond pani ise paika edasise töökäigu ning milliseid tööriistu ja metoodikaid arendamisel rakendada. Protsessi juhtimiseks otsustati kasutada agiilse arenduse võtteid ning jagada arendusetapid sprintideks. Iga sprindi lõpus tehti retrospektiiv, kus võeti kokku üks arendusetapp ja arutati, mida oleks saanud teha teisiti. Lisaks otsustati arendamisel kasutada paarisprogrammeerimise metoodikat. Meeskond kasutas ka SCRUM metoodikast pärinevat *Daily Scrum* koosoleku võtet, mille mõte seisneb selles, et enne tööpäeva algust arutatakse kiirelt läbi, mida igaüks tegi eelneval päeval, millised on takistused ning millele tuleb hetkel keskenduda. Sprindi eesmärgid ja jooksvad ülesanded pandi GitLab'is kirja *issue*'dena, mis andsid hea ülevaate pooleliolevatest ülesannedest. Kuna ettevõtte konfiguraatorite infosüsteem on arendusfaasis ning uusi mõtteid ja lahendusi tuli ettevõtte poolt kogu aeg jooksvalt juurde, pidi meeskond rõhuma paindlikkusele. Kliendiga kohtuti keskmiselt iga kahe nädala tagant, et saada tagasisidet lisandunud funktsionaalsusele või leida lahendusi esile kerkinud küsimustele. Töötunde märgiti meeskonnaga Toggle'is, kust sai hea ülevaate meeskonnaliikmete panuse kohta.

Projekti alguses saadi rakenduse arendamisel meeskonnaga füüsiliselt kokku ning eraldi programmeerimist oli vähem. Kokku saadi meeskonnaga 2-3 korda nädalas ning suuremate probleemide lahendamiseks toimus täiendavaid kokkusaamisi juhendajatega. Kui meeskonna töökorraldus oli ühtlustunud ja projekti esialgne põhi valmis saanud, sai edasise töö kasutajalugudena ära jaotada – võeti kasutusele paarisprogrammeerimine ning tükid jaotati paaride vahel ära. Projekti arendamise teises pooles viidi pandeemia tõttu kokkusaamised ja suhtlus üle veebikeskkonda. Hilisema arendusjärgu jooksul kliendi esialgsed nõuded täpsustusid ning muutusid olulisel määral, millest tulenevalt viidi läbi muudatusi rakenduse arhitektuuris ja disainis.

Klient soovis arendusprotsessi kõrvalt jälgida, et uuenenud rakendusega enne kokkusaamist tutvuda. Seega pani tiim kogu rakenduse veebiserverisse ülesse. Rakendus automatiseeriti pideva integratsiooni (CI) ja pideva tarne (CD) vahenditega. Projekti kogu kood laeti ülesse GitLab'i repositooriumisse, mille kaudu toimus versioonihaldus. Serveri rakendus laeti ülesse Azure'i ning enne juurutamist käivitatakse seal automaatselt testid.

3. Tulemused

Katuseotsikute tootegrupi konfiguraatori loomine algas esialgu konkurendil konfiguraatori ekraanipiltidest koos juurde lisatud selgitustega, mida nad oma programmis sooviksid jätta samaks ning mida teisiti teha. Projekti eesmärgiks oli lihtsustada ja kiirendada vajaliku toote leidmise protsessi projekteerija, lõppkliendi ning ka ettevõtte töötajate jaoks.

3.1 Korralikult väljatöötatud nõuded

Käesolevas peatükis on antud ülevaade projekti nõuetest, kasutajalugudest ja kasutusjuhtudest. Kasutajalugude kirja panemisel lähtuti narratiivide tehnikast, mis seletab rakenduse funktsionaalsust. Tehnikat tutvustab Dines Bjørner raamatus “Software Engineering 1” [3]. Kasutusjuhud leiab lisast 2.

Funktsionaalsed nõuded:

- Keele valimine
- Katuseotsiku filtreerimine
    - funktsiooni järgi;
    - mudeli järgi;
    - mõõtude järgi;
- Vastava(te) katuseotsiku(te) leidmine sisestatud
    - õhuvoolu järgi;
    - rõhukao järgi;
- Projekti informatsiooni täitmine
- Katuseotsiku edasine konfigureerimine detailvaate alt
- Toote tellimusdokumendi allalaadimine PDF-formaadis

Mittefunktsionaalsed nõuded:

- Universaalse andmemudeli loomine
- Puhta koodi kirjutamine
- Efektiivse andmestruktuuri leidmine
- Korrektsete andmetüüpide kasutamine

- Funktsioonide ja algoritmide keerukuse minimeerimine
- Ajakohaste raamistike ja pakettide kasutamine
- Olemasolevate komponentide kasutamine
- Kasutajaliidese ja serveripoolse rakenduse ühildamine
- Kasutajaliideses pole ebavajalike detaile ja on sarnane originaalse konfiguraatoriga
- Vigade käsitlus (*Error handling)*

3.1.2 Kasutajalood

1. Kasutajana tahan avada katuseotsiku konfiguraatori, et saaksin valida oma vajadustele vastava katuseotsiku.
2. Kasutajana tahan valida katuseotsiku funktsiooni, et saaksin valida kas otsik võtab õhku sisse või puhub välja.
3. Kasutajana tahan valida katuseotsiku mudeli, et saaksin otsida toodet kõikide tootekataloogis olevate mudelite põhjal.
4. Kasutajana tahan valida rippmenüüst katuseotsiku suurused nii, et need oleks tõusvas järjekorras.
5. Kasutajana tahan valida ümarate ja kandiliste katuseotsikute vahel, et saaksin valida oma vajadustele vastava ülemineku kujuga katuseotsiku.
6. Kasutajana tahan sisestada õhuvoolu, et valida katuseotsik mis viib endast läbi korrektses mahus õhku.
7. Kasutajana tahan sisestada korrektse rõhukao, et valida katuseotsik mis vastab mu vajadustele.
8. Kasutajana tahan projekti informatsiooni sisestada viite välja.
    8.1. Väljad on järgmised: Kliendi Nimi, Projekti Nimi, Disainer, Kuupäev, Viitenumber, Kommentaarid.
    8.2. Sisestatud informatsioon peab olema alfanumeeriline.
    8.3. Kasutajana tahan vajutada “Tühista” et väljad tühistada ja kustutada.
9. Kasutajana tahan vajutada otsingunupule pärast oma parameetrite sisestamist, et näha millised otsikud minu valikutele vastavad.
    9.1. Kõik väljad ei pea olema täidetud, et otsingutulemused tagastada.
10. Kasutajana tahan vajutada nupule “Vali”, et toodet konfigureerida.
    10.1. Kasutajana tahan valida katuseotsiku materjali.

10.2. Kasutajana tahan valida kanali ühendust, sealhulgas võimalust lisada oma valitud suurustega liitmik.

10.3. Kasutajana tahan sisestada katuseotsiku värvi RAL värvikoodi alusel.

11. Kasutajana tahan linnukesega valida otsikule vajadusel katuseläbiviiku.

11.1. Kasutajana tahan valida katuseläbiviigu materjal.

11.2. Kasutajana tahan valida katuseläbiviigu kõrgust.

11.3. Kasutajana tahan valida katusläbiviigu sobiva ühenduse.

11.4. Kasutajana tahan valida katuseläbiviigu värvi.

12. Kasutajana tahan võimalust alla laadida minu konfigureeritud tooteversiooni PDF-dokumenti tema kõikide omadustega, et konfiguratsioon üle kontrollida.

13. Kasutajana tahan vahetada rakenduses keelt, et lugeda teksti oma emakeeles.

14. Kasutajana tahan hiirega õhuvoolugraafikul hõljudes, et see graafikut suurendaks.

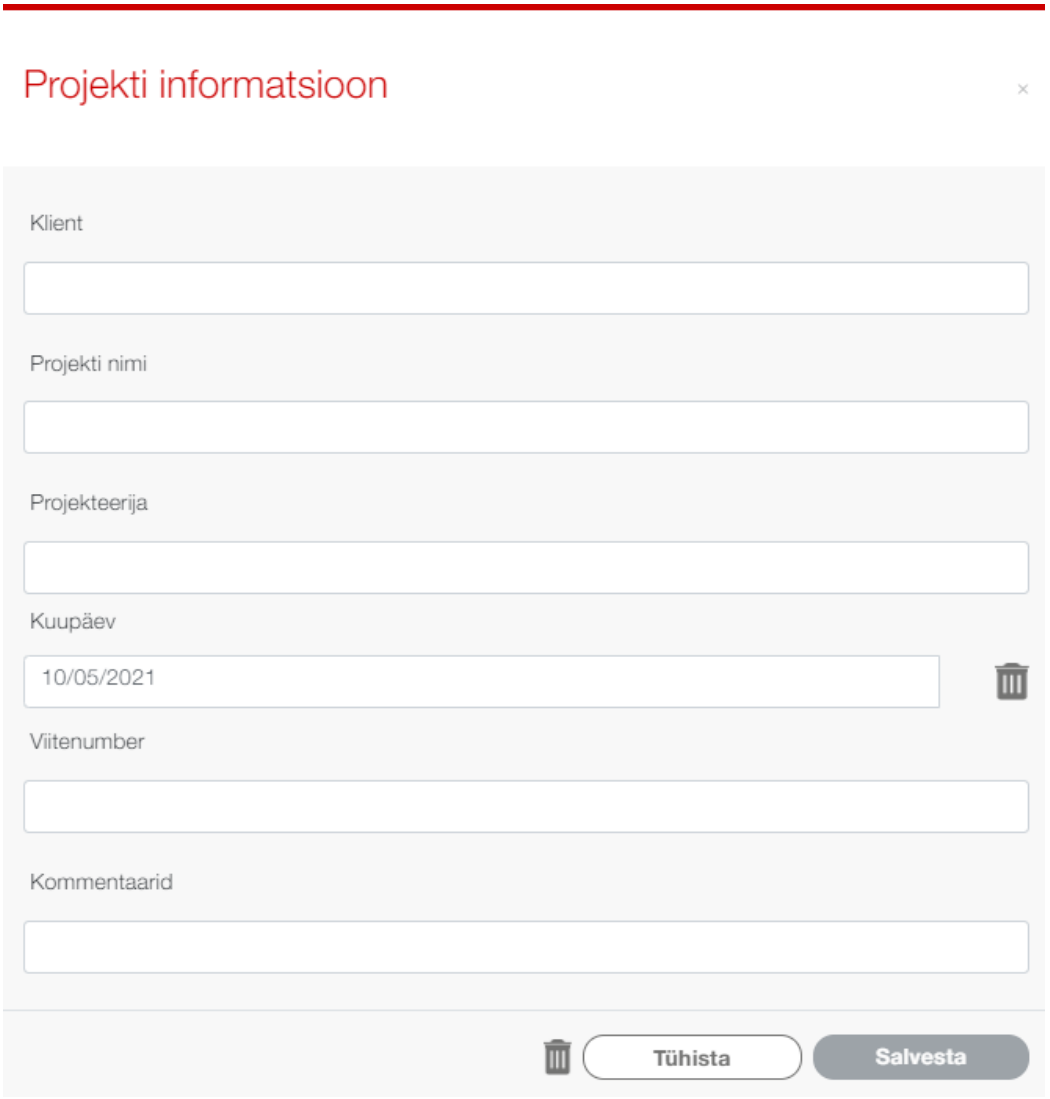
3.2 Rakendus

Valminud projekt on täiesti eraldiseisev osa, mis hakkab tulevikus suhtlema ettevõtte infosüsteemis paikneva *Webshop*'iga, kus toodete andmed liiguvad nii *Webshop*'ist rakendusse kui ka pärast konfigureerimist tagasi *Webshop*'i.

Avalehel avaneb kasutajale esialgu 7 erinevat valikut, mille järgi on võimalik kasutajal filtreerida välja endale sobilik toode. Valikud toote filtreerimiseks on järgmised: õhujaotus, õhuvool, mudel, rõhukadu, suurus, kanali ühendus. Vahetades katuseotsiku ühenduse kuju, muutuvad selle järgi vastavalt ka kõik rippmenüüdes olevad väärtused, sealhulgas mudelite ja õhujaotuse valik. Õhuvoolu sisestamisel arvutatakse välja iga toote kohta rõhukadu. Rõhukao sisestamisel vastupidiselt arvutatakse välja iga toote kohta õhuvool. (Joonis 1)

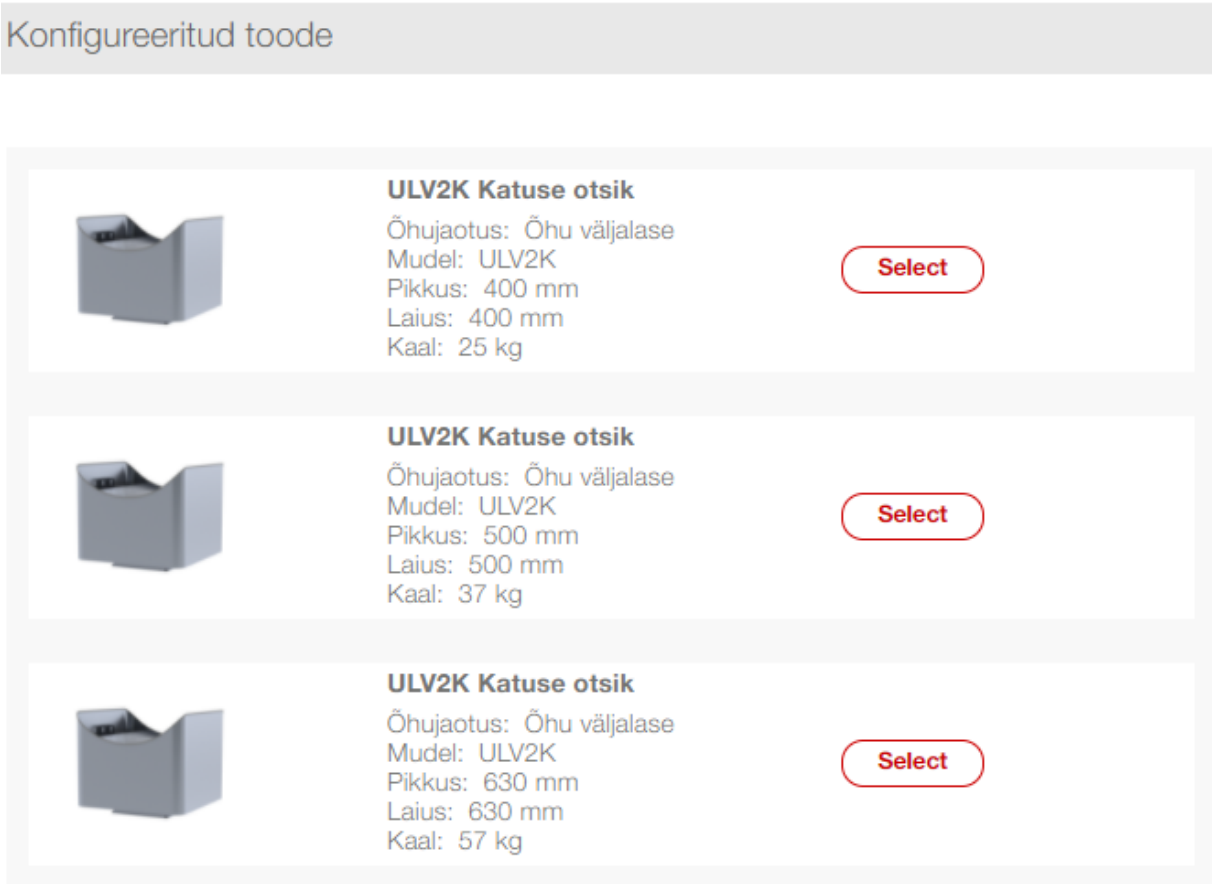
Avalehel on võimalus kasutajal keelt vahetada, kus on valikus hetkel eesti keel ja inglise keel. Kasutaja saab avalehel ära määrata enda projekti üldise info klikkides nupule "Projekti informatsioon". Pärast seda avaneb modaalaken, kus on kasutajal võimalik täita järgmised väljad projekti kohta: klient, projekti nimi, projekteerija, kuupäev, viitenumber ja kommentaarid (Joonis 4). Kui kasutaja soovib täidetud väljad mingil põhjusel kõik korraga tühjendada ja täitmist algusest peale alustada, saab ta vajutada nupule "Tühista". Vajutades nupule "Salvesta", salvestatakse kasutaja poolt täidetud väljad ning need kuvatakse hiljem koostatud

PDF-dokumendis koos konfigureeritud tootega. Erandina ei saa salvestada juhul, kui mõni väli on täitmata, kõik väljad on kohustuslikud ning sisestada tohib vaid alfanumeerilisi tähemärke.



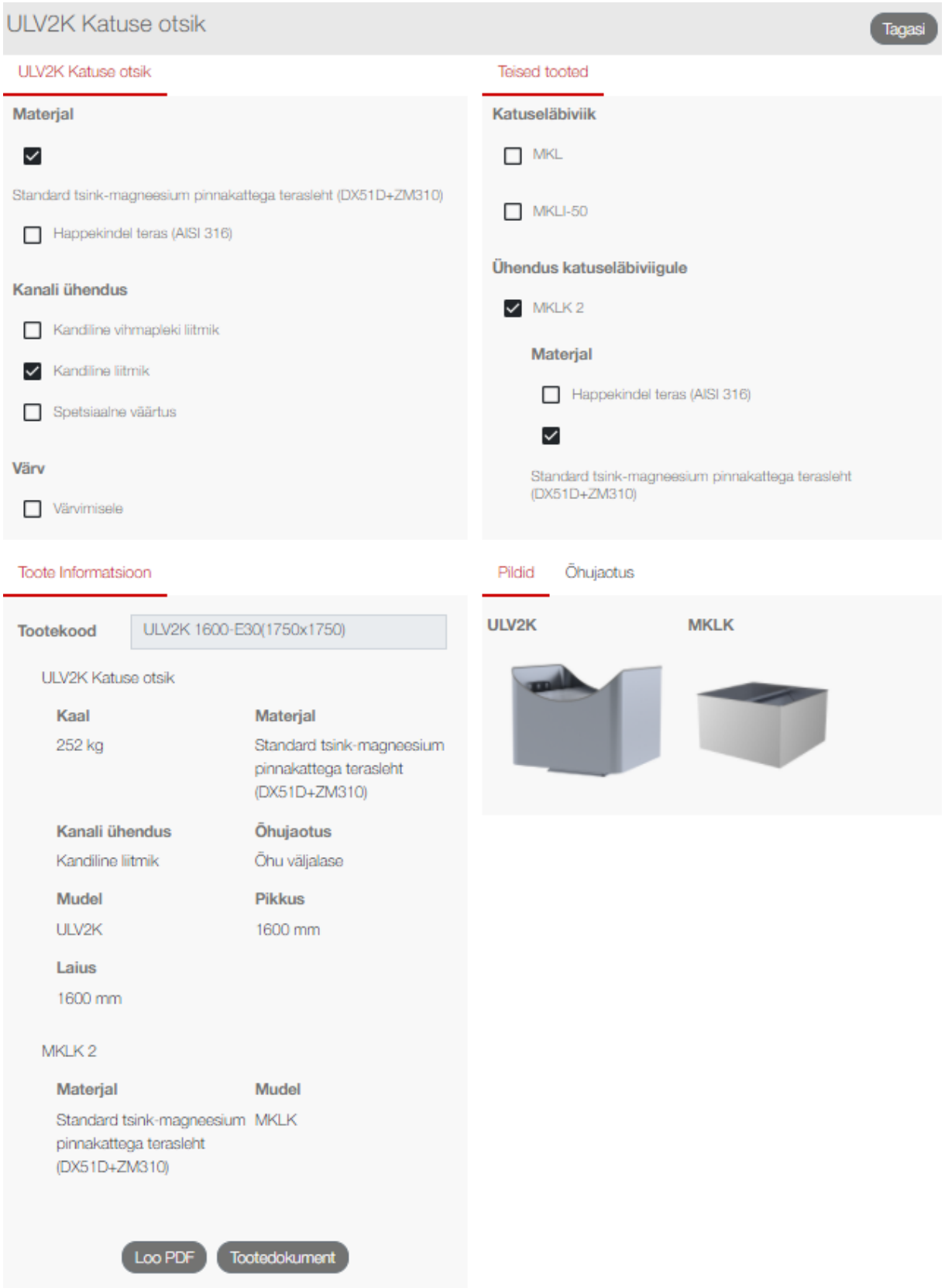
Joonis 4. Projekti informatsiooni modaalaken.

Pärast soovitud parameetrite valimist ning sisestamist saab kasutaja vajutada nupule “Otsi”, pärast mida kuvatakse alampäises “Konfigureeritud toode” valitud parameetritele vastavate toodete loetelu (Joonis 5). Korraga kuvatakse kasutajale kuus toodet ning kui tooteid on rohkem, saab kasutaja klikkida nupule “Kuva rohkem”, millega kuvatakse lisaks veel neli toodet. Iga toote juures on eraldi välja toodud selle toote täpsed andmed ja mudeli pilt. Kui kasutaja on valinud kataloogist toote, vajutades nupule “Otsi”, suunatakse kasutaja uuele vaatele. Uuel vaatel saab toodet konfigureerida ning vastavate andmetega PDF-dokument alla laadida.



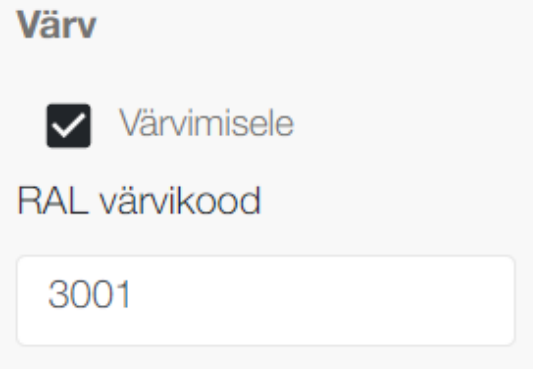
Joonis 5. Osaline toodete valiku vaade.

Detailvaates avanevad kasutajale valikud toote konfigureerimiseks ning lehe alampäises “Toote Informatsioon” all on kuvatud kogu informatsioon konfigureeritava toote kohta (Joonis 6).

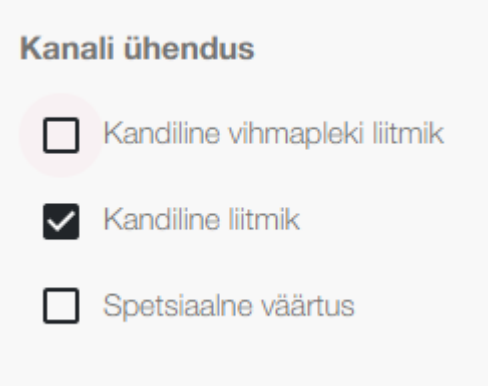


Joonis 6. Osaline detaili vaade.

Vasakpoolses märkeruutude tulbas saab kasutaja konfigureerida eelnevalt valitud toodet, valides sobiliku materjali ja kanali ühenduse. Kui kasutaja soovib, et tema tellimus läheks tootmisel värvimisele, avaneb tekstikast, kuhu peab sisestama RAL-värvikoodi (Joonis 7). Materjali valikuid on kaks, standard tsink-magneesium pinnakattega terasleht ja happekindel teras. Katuse läbiviigu ühenduse valikuid on kolm. Esimene valik on kandiline vihmapleki liitmik ning teine valik on kandiline liitmik. Viimaseks valikuks on spetsiaalne väärtus ning kui kasutaja valib selle, avaneb selle alla kaks teksikasti, kuhu ta saab ise sisestada liitmiku pikkuse ja laiuse (Joonis 8).



Joonis 7. Värvikoodi tekstikast.

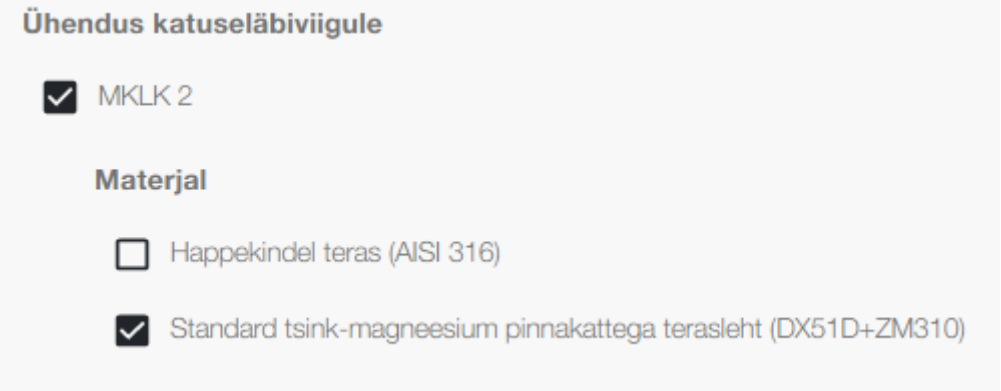


Joonis 8. Kanaliühenduse konfigureerimise väljad.

Parempoolses tulbas saab kasutaja oma valitud tootele külge lisada katuseläbiviigu ning selle ühenduse. Kui klient soovib ja klikib katuseläbiviigu valikule “MKL” avanevad uued märkeruudud selle valiku konfigureerimiseks (Joonis 9). Katuseläbiviigu konfigureerimiseks on järgmised valikud: materjal, isolatsioon, kõrgus, sisekest.

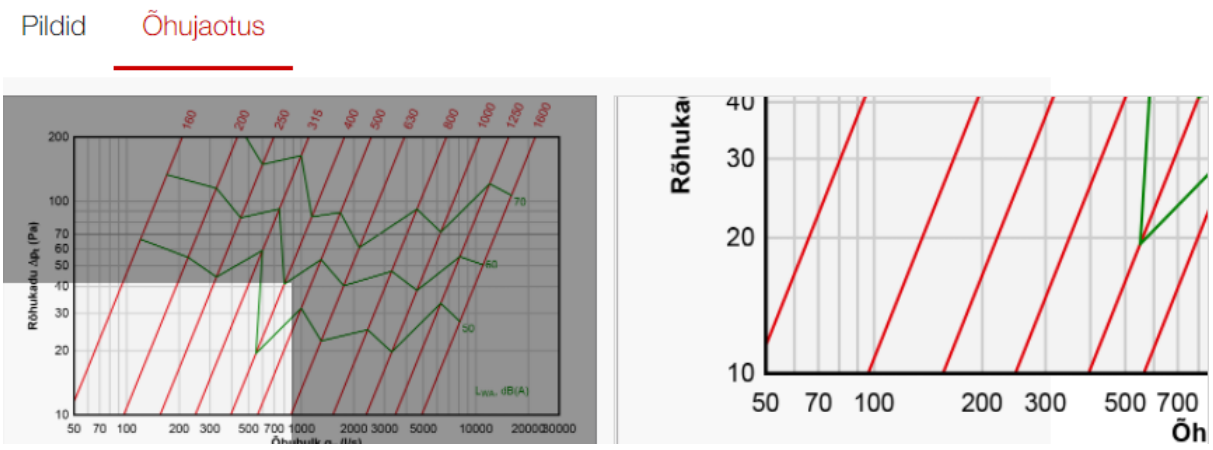
Joonis 9. Katuseläbiviigu konfigureerimise valikud.

Kui kasutaja soovib toote juurde lisaks katuseläbiviigu ühendust, siis antul juhul klikib ta valikule “MKLK 2”. Järgnevalt lisandub selle alla samuti uus valik, kus saab ära märkida sobiva materjali valitud katuseläbiviigu ühendusele (Joonis 10)



Joonis 10. Katuseläbiviigu ühenduse konfigureerimise valik.

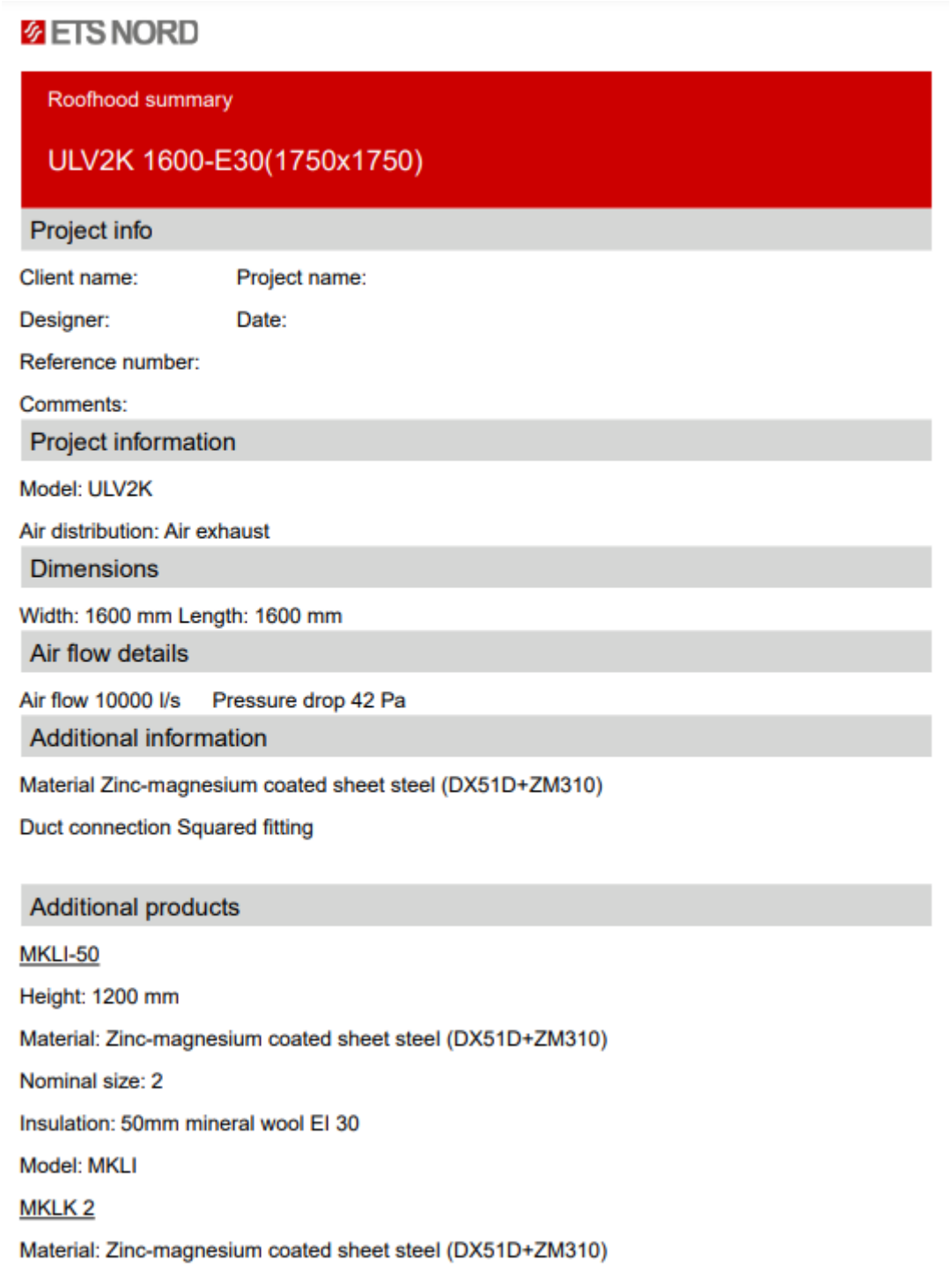
Pärast toote ning sellega kaasnevate juppide konfigureerimist saab klient täieliku ülevaate koos kõigi toodete parameetritega alampäisest "Toote Informatsioon" (Joonis 12). Samuti genereeritakse kasutajale jooksvalt tootekood, mis kirjeldab kogu toote konfiguratsiooni koos vajadusel kasutaja poolt juurde lisatud katuseläbiviigu ja selle ühendusega. Samuti on välja toodud paremas küljes valitud toote pilt ning klikkides sõnale "Õhujaotus" kuvatakse selle asemel toote õhuvoolu-rõhukao graafik, mida on kasutajal võimalik suurendada, kursoriga pildile liikudes (Joonis 11).



Joonis 11. Katuseotsiku õhuvoolu ja rõhukao graafiku suurendamine.

Joonis 12. Osaline Product Info vaade.

Kui kasutaja on konfigureerimise lõpule viinud on tal võimalik kogu eelnevalt sisestatud informatsioon koos toodete parameetritega salvestada PDF-dokumendina ning see alla laadida vajutades nupule “Loo PDF” (Joonis 13). Kui kasutaja soovib näha täpsustavat informatsiooni oma valitud toote kohta, on tal võimalik avada tootekataloog vajutades nupule “Tootedokument”.



ETS NORD

Roofhood summary

ULV2K 1600-E30(1750x1750)

Project info

Client name: Project name:

Designer: Date:

Reference number:

Comments:

Project information

Model: ULV2K

Air distribution: Air exhaust

Dimensions

Width: 1600 mm Length: 1600 mm

Air flow details

Air flow 10000 l/s Pressure drop 42 Pa

Additional information

Material Zinc-magnesium coated sheet steel (DX51D+ZM310)

Duct connection Squared fitting

Additional products

MKLI-50

Height: 1200 mm

Material: Zinc-magnesium coated sheet steel (DX51D+ZM310)

Nominal size: 2

Insulation: 50mm mineral wool EI 30

Model: MKLI

MKLK 2

Material: Zinc-magnesium coated sheet steel (DX51D+ZM310)

Joonis 13. Tellimuse PDF-dokument.

3.3 Arhitektuur

Tarkvara arhitektuuris on tavaks eraldada rakenduses üksteisest kolm kihti, mida nimetatakse kolmekihiliseks arhitektuuriks. Kihid on järgmised: andmetele juurdepääsu kiht, loogikakiht ja presentatsioonikiht. Esimene vastutab suhtluse eest andmebaasiga, teine esindab rakenduse loogikat ja rakenduse funktsionaalsust ning kolmas konverteerib andmed kasutajaliidese jaoks sobivale kujule. [4]

Model-View-ViewModel (edaspidi MVVM) on tarkvara arhitektuuri muster, kus *Model* esindab andmebaasi klasse, *View* esindab kasutajaliidese vaateid ning *ViewModel* esindab andmeid sobival kujul kasutajaliidese jaoks. *Model* paikneb peamiselt andmetele juurdepääsu kihis, *ViewModel* paikneb nii presentatsioonikihis kui ka loogikakihis, vahendades suhtlust *Modeli* ja *View* vahel. MVVM arhitektuuris toimib suhtlus lineaarselt (Joonis 14) – *View* on teadlik *ViewModelist*, aga mitte vastupidi ning *ViewModel* on teadlik *Modelist*, kuid mitte vastupidi. [4]

Joonis 14. MVVM muster [4].

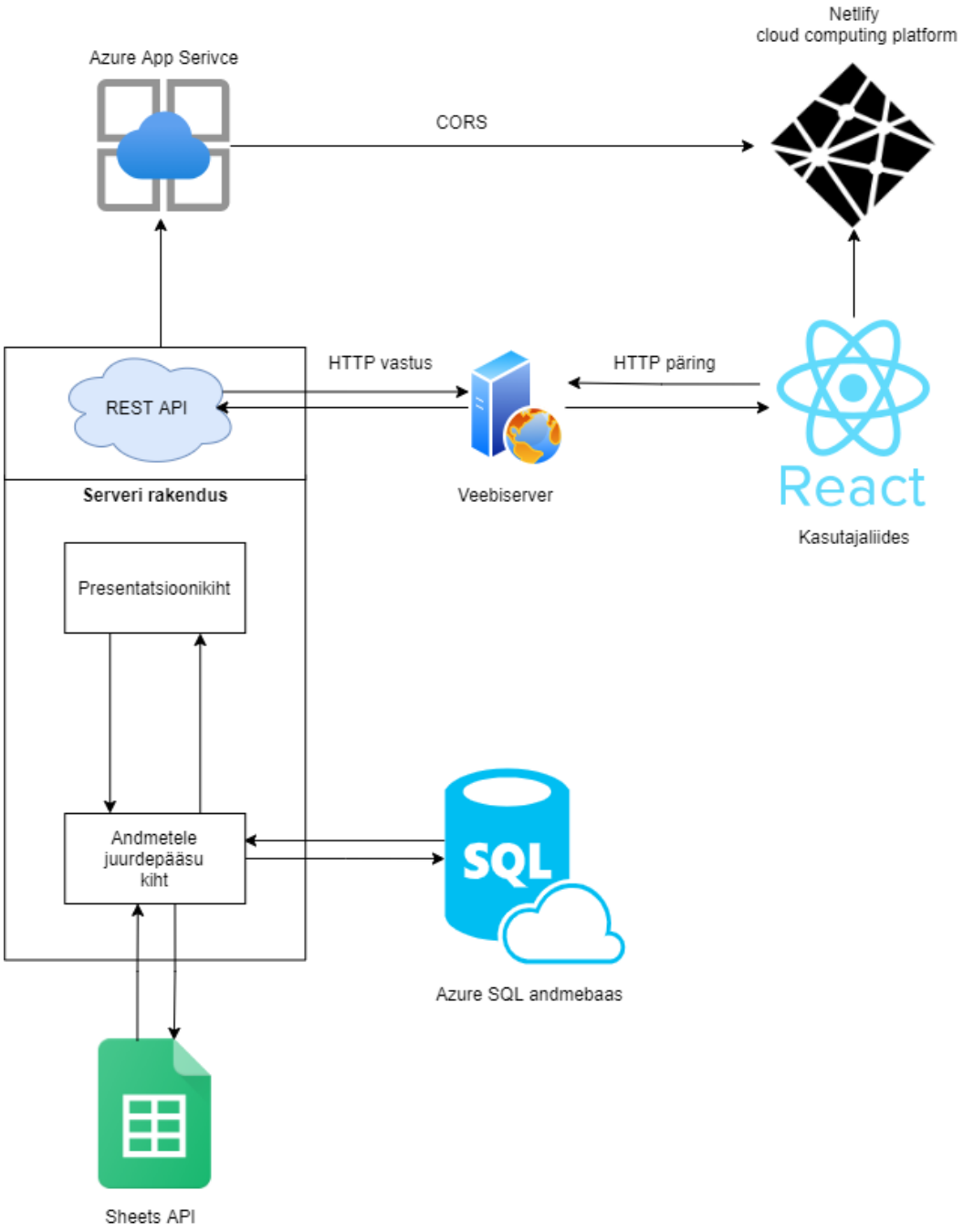
Antud rakenduses on realiseeritud presentatsioonikiht ning andmetele juurdepääsu kiht, st loogikakiht puudub. Presentatsioonikiht küsib andmetele juurdepääsu kihilt andmed (*Model* klassid), konverteerib need sobivale kujule (*ViewModel* klassid) ning koostöös REST API teenusega (täpsemalt järgmises peatükis) küsib kasutajaliides sobivad andmed presentatsioonikihilt. Kasutajaliides, mis esindab *View*'sid, käitub eraldiseisva rakendusena. Viimane on *Single-page application*, millele on iseloomulik see, et *View*'d paiknevad kliendipoolses rakenduses, mitte serveri rakenduses, kus asuvad presentatsioonikiht ja andmetele juurdepääsu kiht [5].

Andmetele juurdepääsu kihi (ing. *data access layer*) eesmärgiks on suhtlemine andmebaasiga ning edastada andmed presentatsioonikihile. Andmeid sisestatakse andmebaasi läbi Sheets API, mis on Google'i poolt loodud tasuta tarkvara, millega võimaldatakse laiaulatuslikku ligipääsu Google Sheets tüüpi dokumentidele läbi REST teenuse. Sheets API andmete uuendamine toimub reaalajas ning on pidevas suhtluses andmetele juurdepääsu kihiga, mis võimaldab andmete uuendamist andmebaasis peale igat *pushi* (laadib lokaalsed muudatused serverisse üles).

Andmetele juurdepääsu kiht:

1. saab andmed Sheets API-lt;
2. loob nendest andmebaasi tabelid Entity Framework Core abil;
3. ladustab andmed Azure SQL serveri andmebaasi;
4. presentatsioonikiht küsib ladustatud andmeid läbi andmetele juurdepääsu kihi (Joonis 15).

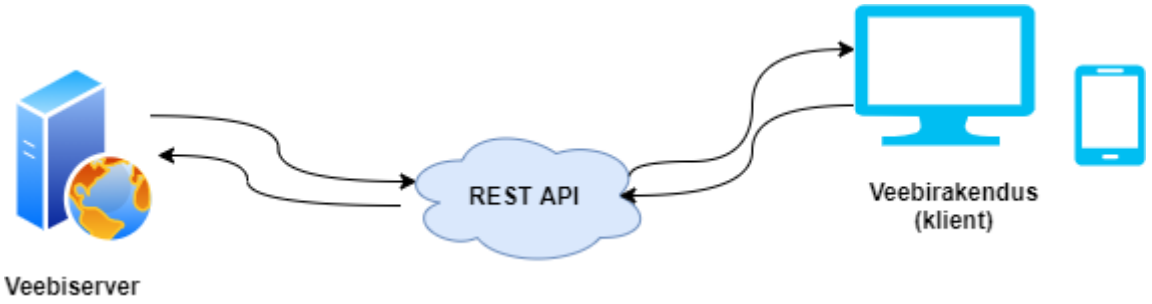
Serveri rakendust võõrustab pilvandmetöötluse platvorm Azure App Service ning kasutajaliides on pilvandmetöötluse platvormil Netlify. Kaks platvormi peavad omavahel suhtlema, et viimane saaks serveri rakenduselt andmeid küsida. Turvalisuse huvides on suhtlus kahe erineva domeeni vahel piiratud SOP (*Same-Origin Policy*) eeskirja järgi ning selle lubamiseks kasutatakse CORS-i (*Cross-Origin Resource Sharing*). Lõplik skeem on kujutatud alloleval joonisel (Joonis 15).



Joonis 15. Projekti arhitektuuri kirjeldus.

3.3.1 REST API

Sõna REST võeti esimest korda kasutusele 2000. aastal Roy Fieldingu poolt, kes avaldas enda doktoritöö veebi arhitektuuri disainist, mille nimeks sai "*Representational State Transfer*" (REST). API (*application programming interfaces*) on liides, mis hõlpsustab funktsioonide ja andmete kogumi abil integratsioone luua programmide vahel ning võimaldab neil omavahel kommunikeerida (Joonis 16). Seega on REST veebi arhitektuur ning API on veebiteenuse liides, mis suhtleb kliendiga ning REST API on API, mis järgib REST-i printsiipe. [6]

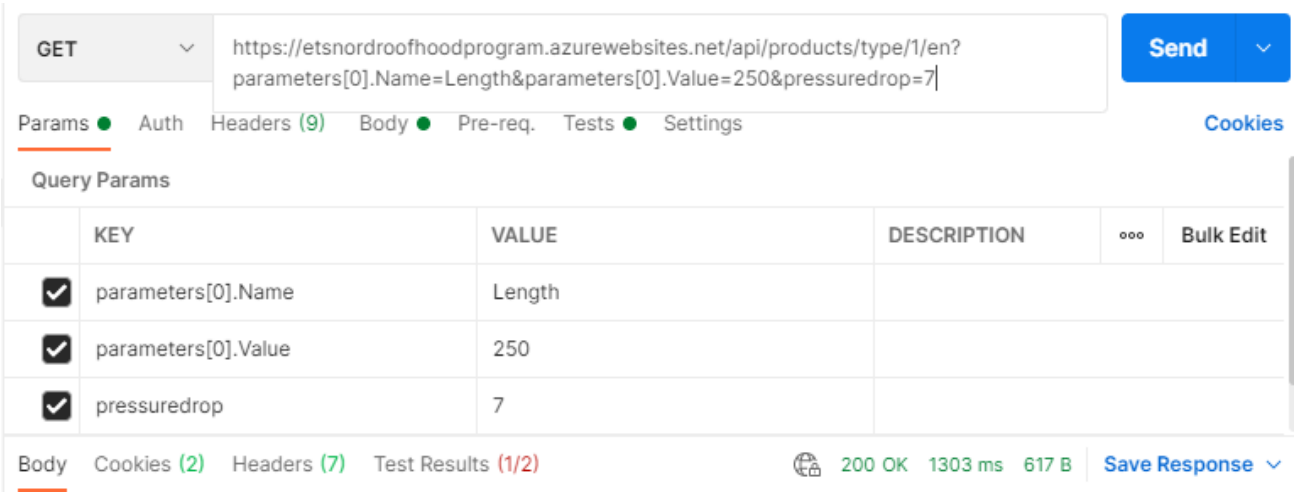


Joonis 16. REST-API andmevoog [6].

Selles projektis on kasutatud REST API-t kahel erineval juhul. Suhtlemiseks kahe veebirakenduse vahel, kus:

1. saatjaks serveri rakendus ja vastuvõtjaks kliendipoolne rakendus (kasutajaliides);
2. saatjaks *Sheets* API ja vastuvõtjaks serveri rakendus.

REST API-d kasutavad ressursside suunamiseks *Uniform Resource Identifiers*'e (URI-s), mida käsitletakse läbi HTTP protokolli, saates selleks GET, POST, PUT, DELETE päringuid [6]. Antud projektis on läbivaks päringuks GET erinevate toodete ja omaduste pärimiseks (Joonis 17). URL-is on võimalik defineerida *query*sid. Joonisel 17 on *query* parameetriks näiteks toote omaduse nimi "*Length*" ja väärtus "250", mis filtreerivad tooteid tema pikkuse alusel ja tagastavad vastava(d) toote(d).



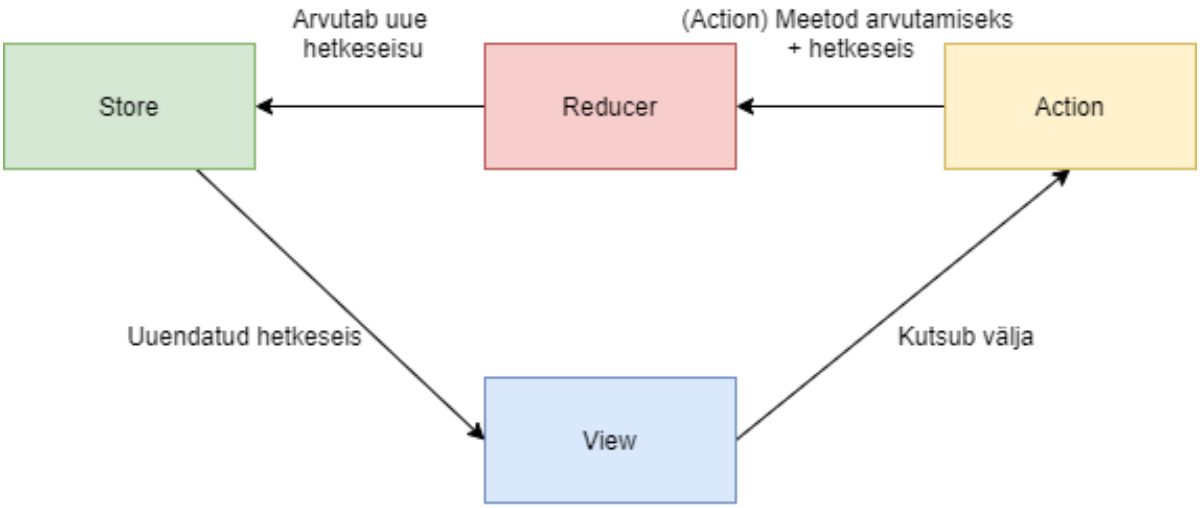
Joonis 17. GET päring toodete ja nende omaduste pärimiseks.

3.3.2 React Redux

Lisaks REST API-le on rakenduse loomiseks vaja kasutajale visuaali, et andmeid näha ja kasutada. Paljud suuremahulised aplikatsioonid kasutavad MVC (*Model-View-Controller*) arhitektuuri hetkeseisu haldamiseks (*state management*). Erinevalt MVC-st on kliendipoolsete raamistikega seda keeruline sarnaselt teha, sest hetkeseis (*state)* on laiali erinevate lehtede vahel, mitte koondatud kokku ühte kohta – rakenduse tasemele (*application level*). Kliendipoolsetes raamistikes on hetkeseisu võimalik hallata Redux arhitektuuri abil, mida antud projektis on rakendatud. [7]

Store hoiab endas kogu rakenduse hetkeseisu ning hetkeseisu muutmiseks on Reduxis Reducer meetodid. Reducer funktsioonile tuleb anda hetkeseis ja tegevus (*Action*) ning Reducer otsustab nende põhjal, mida muuta *Store*'is. Ring lõpeb sellega, et *Store* saadab uuendatud informatsiooni *View*'le tagasi. Seega *View* kutsub välja *Action*'i, *Action* ja hetkeseis saadetakse Reducerile, mis uuendab *Store*'i ning uuendatud informatsioon saadetakse *View*'le tagasi (Joonis 18). [7]

Sellist Reacti arhitektuuri kasutatakse projektis, kui tegu on väliste ja suuremahuliste andmetega, mida ei kasutata ainult ühes vaates (globaalsed andmed). Kui tegu on aga iseseisva elemendiga, mille hetkeseisu soovitakse muuta, siis Redux'i asemel saab kasutada lisaks Reacti komponenti "useState()" või sündmusi (*event*), milles tuleb täpsemalt juttu peatükis 3.4.2.

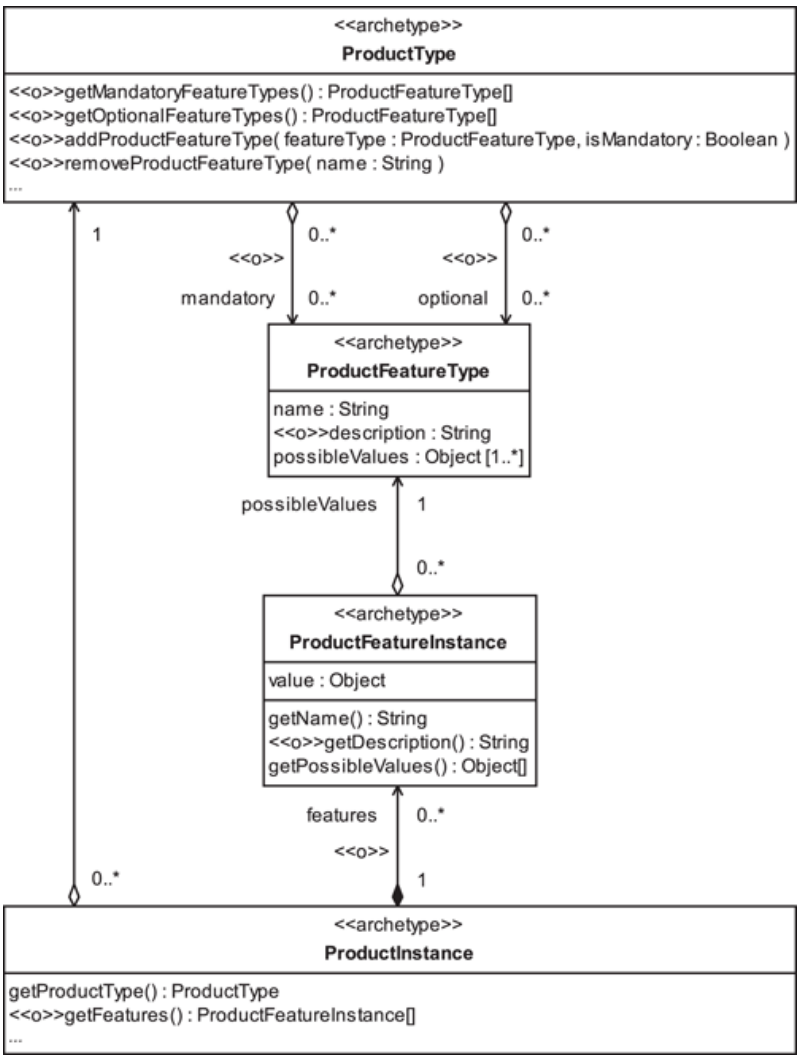


Joonis 18. Redux'i andmevoog [7].

3.4 Disain

3.4.1 Andmemudel

Andmemudeli tegemisel lähtuti J. Arlowi ja I. Neustadti *Product* arhetüübist. Sellest mudelist (Joonis 19) implementeeriti neli arhetüüpi: *ProductInstance, ProductType, ProductFeatureType, ProductFeatureInstance.* Arhetüübid nimetati ümber järgmisteks klassideks: *Product, ProductType, ProductFeature, FeatureValue*. Lisaks Arlowi mudelist tulnud arhetüüpidele implementeeriti juurde klassid *AdditionalFeatureValue, MainFeatureValue, ProductRelation* (Joonis 20). [1]

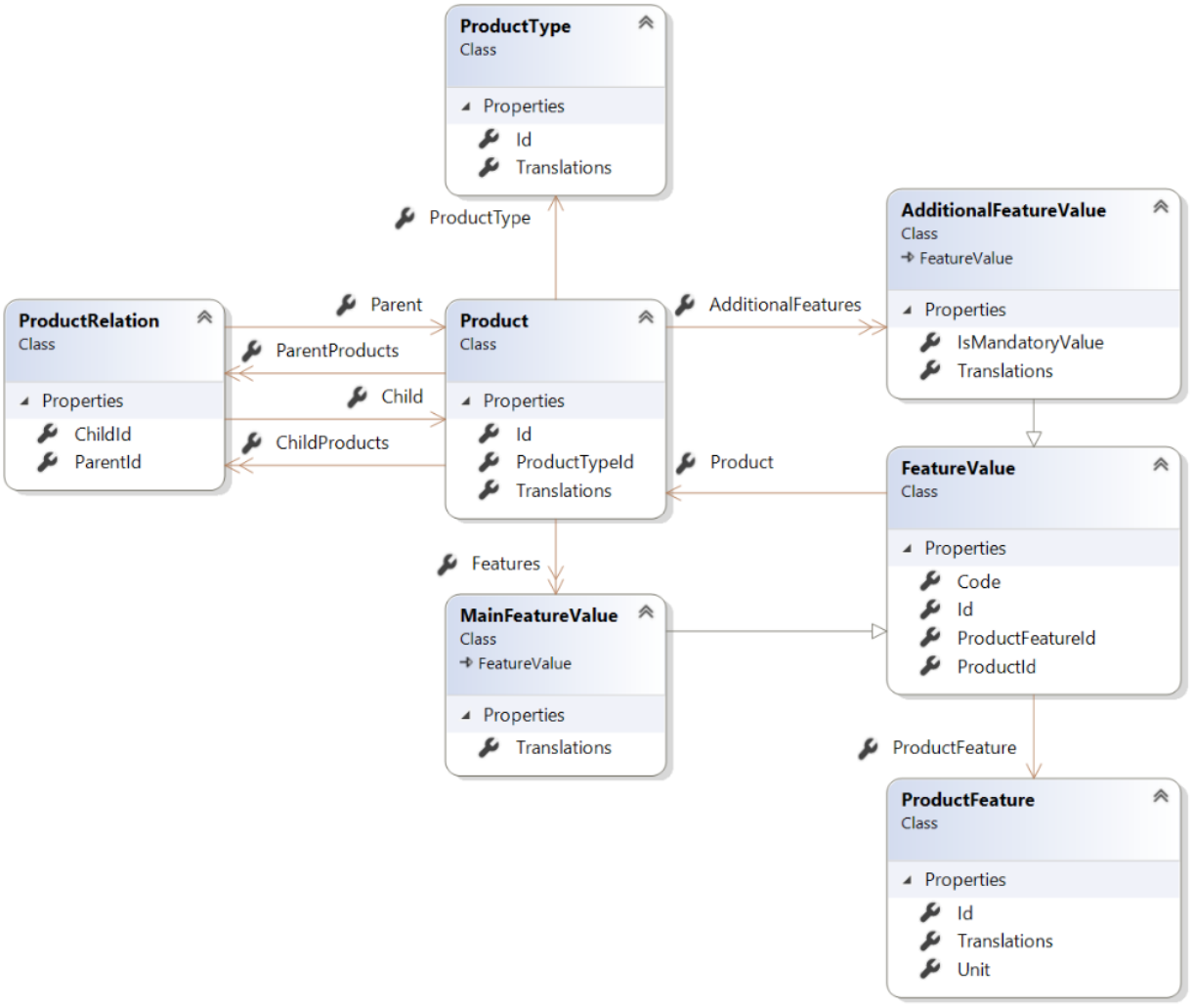


Joonis 19. *Product* arhetüüp [1].

Igal andmebaasi tabelil on oma identifikaator ning objektide vahelised seosed on realiseeritud väliste võtmete abil nagu on kirjeldanud M. Fowler mustritega *Identity Field* ja *Foreign Key Mapping* [8]. Klassil *Product* on tema identifikaator (Id), toote tüübi identifikaator (*ProductTypeId*) ning kogum erinevatest omadustest (*FeaturesValue*), mis tema külge kuuluvad. Omadused jaotuvad *MainFeatureValue'*deks ehk omadused, mis on toote küljes vaikimisi ning *AdditionalFeatureValue*'deks ehk omadused, mida on võimalik kasutajaliideses toote külge konfigureerida või vahetada vaikimisi väärtus *AdditionalFeatureValue* väärtuse vastu. *Product* klass on modelleeritud endale viitava mitu-mitmele seosega ehk tootega võivad olla seotud tema lapsed (*ChildProducts*) ja vanemad (*ParentProducts*). Selle teostamiseks loodi vahetabel nimega *ProductRelation*.

Klassil *ProductType* on küljes identifikaator (Id). Sellega määratakse ära, mis tüüpi tootega on tegu. Klassil *ProductFeature* on küljes identifikaator (Id) ning ühik *Unit*, mis määrab ära mis tüüpi ühikuga on objekt, näiteks millimeeter (mm).

Klass *FeatureValue* ühildab klassi *ProductFeature* ja *Product*. Tema atribuutideks on *ProductId* (võõrvõti), *ProductFeatureId* (võõrvõti) ja *Id* (primaarvõti). *IsMandatoryValue* atribuut määrab, kas omadust on võimalik konfigureerimise käigus eemaldada (*false)* või mitte (*true)*. *Code* atribuut on abiks lõpliku tootekoodi loomisel. Kui näiteks omaduse väärtus (*Value)* on "Tsink-magneesium", siis sellele vastav *Code* on "ZM". (Joonis 20)



Joonis 20. *Product* klassidiagramm.

Jooniselt 20 on näha, et enamikel klassidel on *Translations* väli, kus defineeritud atribuudid kirjutatakse erinevates kultuuriruumides vastavalt. *Culture* klass sisaldab endas identifikaatorit

ning *Language* atribuuti, mis määrab kasutatava keele. Jooniselt 21 paistab, et *FeatureValue* väärtus (*Value* atribuut) paikneb hoopis temale kuuluva *FeatureValueTranslation* klassis. Andmete sisestamiseks tuleb määrata keel ja luua väärtus vastavas keeles. Näiteks tuleks inglise keeles *Value* atribuudi väärtuseks "Stainless steel" ja temaga seotud *Culture*'i *Language* atribuudiks "en". Lisas 3 on ülevaade kõikidest andmemudeli atribuutidest.

Joonis 21. *FeatureValueTranslation* klassidiagramm.

3.4.2 Andmete esitlus

Andmestruktuur on andmete organisatsioon, mis sisaldab enda mälus informatsiooni, et luua efektiivseid algoritme. Andmestruktuuriks võib olla järjekord, pinu, lingitud list, kuhi, *dictionary* või puu. *Dictionary* on abstraktne andmetüüp võtme ja väärtuse paaride hoiustamiseks. Ligi pääseb väärtusega seotud võtmega ning põhilised operatsioonid on *new, insert, find* ja *delete.* Massiiv (ing. keeles *Array*) on järjestatud kollektsioon üksustest, millele saadakse ligi indeksiga. Antud projektis käsitletakse peamiselt *dictionary* ja massiiv tüüpi andmestruktuure. [9]

Sobiv andmestruktuur moodustatakse rakenduse presentatsioonikihis. Järgnev näide (Joonis 22) on meetodist, kus luuakse *ProductViewModel* tüübiga objekte. Andmed küsitakse SQL vaatelt (Lisa 4) ning parameetriteks on toote ja tema tüübi identifikaatorid ning keel (*language*). Andes meetodile sisendiks “1”, “1”, “en”, filtreeritakse SQL vaatest read, kus veergude *ProductId*, *ProductTypeId* ja *Language* väärtused vastavad sisendile. Kui *Language* sisendiks on “et”, tagastatakse kirjed eesti keeles. Leitud read käiakse *foreach* tsükliga läbi, luuakse *ProductViewModel*, kus esinevad omadused (*MainFeatureViewModel*) lisatakse *dictionary* tüüpi atribuuti *Features*.

```
public async Task<ActionResult<ProductViewModel>> CreateProduct(string productTypeId, string productId,
    string language)
{
    var product :List<ProductViewDataEn> = await _context.ProductsViewEn // DbSet<ProductViewDataEn>
        .Where(x :ProductViewDataEn => x.ProductTypeId == productTypeId)
        .Where(x :ProductViewDataEn => x.ProductId == productId) // IQueryable<ProductViewDataEn>
        .Where(x :ProductViewDataEn =>x.Language == language).ToListAsync(); // Task<List<…>>
    var productViews = new List<ProductViewModel>();
    foreach (var r :ProductViewDataEn in product)
    {
        var productView = productViews.FirstOrDefault();
        if (productView == null)
        {
            var productViewModel = new ProductViewModel
            {
                Id = r.ProductId, Name = r.ProductName, ProductType = r.ProductType,
                ProductTypeId = r.ProductTypeId
            };
            var feature = new MainFeatureViewModel
            {
                Name = r.Feature, Value = r.Value, Unit = r.Unit, Code = r.Code, ProductFeatureId = r.FeatureId
            };
            productViewModel.Features[r.FeatureId] = feature;
            productViews.Add(productViewModel);
        }
        else
        {
            var feature = new MainFeatureViewModel
            {
                Name = r.Feature, Value = r.Value, Unit = r.Unit, Code = r.Code, ProductFeatureId = r.FeatureId
            };
            productView.Features[r.FeatureId] = feature;
        }
    }
    return productViews.FirstOrDefault();
}
```

Joonis 22. Meetod toote loomiseks.

Meetodi välja kutsumisel kontrolleris, tagastatakse veebibrauseris *ProductViewModel* JSON (*JavaScript Object Notation*) formaadis (Joonis 23). Antud objekti küljes on kõik olulised atribuudid andmemudelist, mis on kasutajaliideses vajalikud toodete ja tema atribuutide küsimiseks, kuvamiseks ning filtreerimiseks. Sealhulgas *dictionary* nimega *features*, mis sisaldab endas objekte ning objektile vastavaid võtmeid, näiteks "*material*". *Dictionary* atribuudid on tuttavad andmemudelist, nagu näiteks *name* väärtusega "Materjal", *value* väärtusega "Tsink-magneesium..." ja *code* väärtusega "ZM".

```
{
    "id": "12",
    "name": "ULV2K Katuse otsik",
    "productType": "Katuse otsik",
    "productTypeId": "1",
    "features": {
        "weight": {
            "unit": "kg",
            "productFeatureId": "weight",
            "name": "Kaal",
            "value": "252",
            "code": null
        },
        "material": {
            "unit": null,
            "productFeatureId": "material",
            "name": "Materjal",
            "value": "Standard tsink-magneesium pinnakattega terasleht (DX51D+ZM310)",
            "code": "ZM"
        },
        "ductConnection": {
            "unit": null,
            "productFeatureId": "ductConnection",
            "name": "Kanali ühendus",
            "value": "Kandiline liitmik",
            "code": "E30(1750x1750)"
        },
        "airDistribution": {
            "unit": null,
            "productFeatureId": "airDistribution",
            "name": "Õhujaotus",
            "value": "Õhu väljalase",
            "code": null
        },
        "model": {
            "unit": null,
            "productFeatureId": "model",
            "name": "Mudel",
            "value": "ULV2K",
            "code": null
        },
        "length": {
            "unit": "mm",
            "productFeatureId": "length",
            "name": "Pikkus",
            "value": "1600",
            "code": null
        },
        "width": {
            "unit": "mm",
            "productFeatureId": "width",
            "name": "Laius",
            "value": "1600",
            "code": null
        }
    }
}
```


Joonis 23. Ühe toote GET päringu tulemus JSON formaadis.

Joonis 24 on näide päringust ühe toote võimalike omaduste (*AdditionalFeatures*) kohta, mida saab tootel muuta, kustutada ja tootele lisada. Tegu on *dictionary*'ga, kuid see sisaldab endas massiive erinevatest objektidest. Seekord vastab võti massiivile, mitte objektile. Meetodi antud andmestruktuuri loomiseks leiab lisast 6.


```
{
  "material": [
    {
      "isMandatoryValue": true,
      "productFeatureId": "material",
      "name": "Materjal",
      "value": "Standard tsink-magneesium pinnakattega terasleht (DX51D+ZM310)",
      "code": "",
      "unit": null
    },
    {
      "isMandatoryValue": true,
      "productFeatureId": "material",
      "name": "Materjal",
      "value": "Happekindel teras (AISI 316)",
      "code": "H",
      "unit": null
    }
  ],
  "ductConnection": [
    {
      "isMandatoryValue": true,
      "productFeatureId": "ductConnection",
      "name": "Kanali ühendus",
      "value": "Kandiline vihmapleki liitmik",
      "code": "KL(1750x1750)",
      "unit": null
    },
    {
      "isMandatoryValue": true,
      "productFeatureId": "ductConnection",
      "name": "Kanali ühendus",
      "value": "Kandiline liitmik",
      "code": "E30(1750x1750)",
      "unit": null
    },
    {
      "isMandatoryValue": true,
      "productFeatureId": "ductConnection",
      "name": "Kanali ühendus",
      "value": "Spetsiaalne väärtus",
      "code": "KL(AxB)",
      "unit": null
    }
  ]
}
```

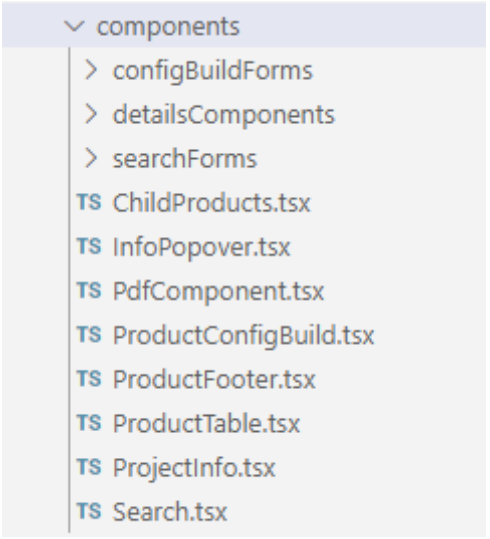

Joonis 24. Ühe toote võimalike omaduste GET päringu tulemus JSON formaadis.

3.4.3 ReactJs

ReactJs (edaspidi React) on Facebooki poolt arendatud lähtekoodiga teek, mis võimaldab luua kasutajaliideseid ning taaskasutatavaid komponente [7]. Reactis on erinevaid viise, kuidas andmeid komponentidele saata või neilt pärida – seda saab teha *props (properties),* hetkeseisu või Redux'i abil. Peatükis 3.3.2 "React Redux" toodi välja, kuidas toimib serveripoolne renderdamine Redux'i abil. Käesolevas peatükk kajastab kliendipoolset renderdamist.

"Reacti saab kasutada ja defineerida kui V osana MVC raamistikus, lüües vaate lahku erinevateks komponentideks" [10]. Komponentide loomiseks kasutatakse rakenduses JSX laiendust. "React toimib ka ilma JSX laienduseta, kuid see võimaldab komponente luua ja kasutada lihtsamini ning neid struktureerida nagu tavalisi HTML elemente" [10].

Rakendus kompilleerub TypeScriptis (*superset of* JavaScript) ning jooniselt 25 on näha TypeScript laiendusega JSX komponente (.tsx). "Lihtsaim viis suhtluse moodustamiseks ülem- ja alamkomponendi (*child* ja *parent*) vahel on läbi *props*'ide" [10]. Komponendi defineerimisel saab komponendile lisada atribuute, mida kutsutakse *props*'ideks. Antud juhul (Joonis 26) on ülemkomponendiks *Product* ning alamkomponendiks *ProductTable* ning koodiread, mis jäävad "<*ProductTable*" ja "/>" (koodiread 239-242) vahele on *propsid*.



Joonis 25. Kasutajaliidese rakenduse *components* kaust.

```
236        )}
237        {!productsFetching ? (
238          <ProductTable
239            data={products}
240            productsToShow={productsToShow}
241            airFlow={airFlow}
242            pressureDrop={pressureDrop}
243          />
244        ) : (
245          <Loading/>
246        )}
```

Joonis 26. Kasutajapoolse rakenduse komponendi *ProductTable* initsialiseerimine.

Teine viis andmete käsitlemiseks on hetkeseis. Kui *propse* saadetakse komponendilt komponendile, siis hetkeseis representeerib komponendi sisest informatsiooni. Olgu muutujaks *productsToShow* ning tema vaikimisi väärtus "6" (Joonis 27). Muutuja omab endas hetkeseisu ning hetkeseisu muutmiseks on funktsioon *setProductsToShow*.

```
const [productsToShow, setProductsToShow] = useState<number>(6);

const loadMore = () => {
  setProductsToShow(productsToShow + 4);
};
```

Joonis 27. Hetkeseisu muutmine kasutajapoolses rakenduses.

Sündmused (*events*) on tegevused, mis käivitatakse pärast kasutaja interaktsiooni HTML elemendiga. Sündmusele saab defineerida *handler*-meetodi, mida konkreetse sündmuse korral käivitada [11]. Interaktsiooniks võib olla näiteks klikkimine mõne HTML elemendi peal. Joonisel 28 on *handler* meetod, mis võtab sisendiks sündmuse ning muudab muutuja *productModel* hetkeseisu (*setProductModel*).

```
const handleModelChange = useCallback(
  (e: React.ChangeEvent<HTMLInputElement>) => {
    const options = modelOptions;
    let index;
    if (e.target.checked) {
      options.push(e.target.value);
    } else {
      index = options.indexOf(e.target.value);
      options.splice(index, 1);
    }
    var multipleproductModelsToString = options.join(",").toString();
    setProductModel(multipleproductModelsToString);
    sessionStorage.setItem("ModelValue", multipleproductModelsToString);
  },
  [productModel]
);
```

Joonis 28. *Handler*-meetod, mis muudab hetkeseisu.

Reacti elutsükli viimaseks meetodiks on *render*, mida kasutatakse informatsiooni uuendamiseks. Kui kuskil tuvastatakse muudatus, siis rakendus renderdab ning informatsiooni uuendatakse. *Render* meetodit saab esile kutsuda läbi sündmuste ja meetodi *useEffect* abil (Joonis 29). *UseEffect* on kombinatsioon Reactis varasemalt tuntud meetoditest *componentDidMount* ja *componentDidUpdate* [12]. Meetod võib osutuda vajalikuks näiteks väliste andmete pärimiseks, sealhulgas uuendada informatsiooni ilma, et kasutaja peaks ise esile kutsuma sündmuse. Kandilistes sulgudes on võimalik defineerida sõltuvad muutujad, mille väärtuse muutumisel kutsutakse meetodit uuesti välja (Joonis 29). Kui sulud tühjaks jätta, kutsutakse meetod välja vaid esimesel renderdamisel.

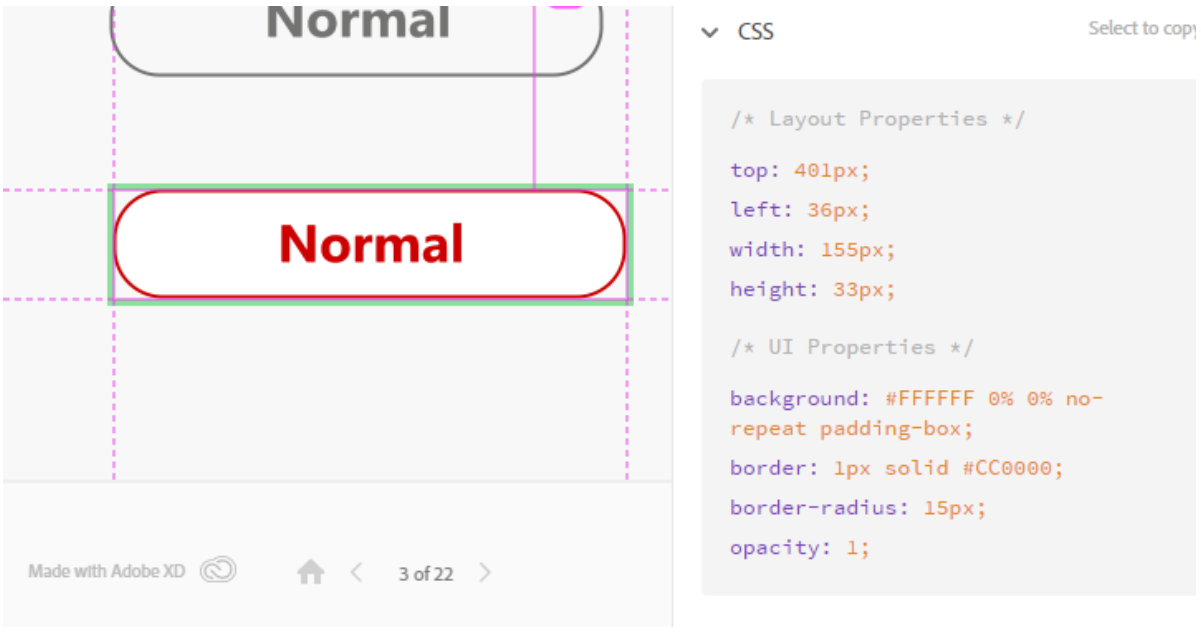
```
useEffect(() => {
  console.log("useeffect");
  getProductsForSearchForm();

}, [shape, i18n.language]);
```

Joonis 29. *UseEffect* meetodi näide.

UI (*User Interface*) komponentide teegid on kogumikud, kus on eelnevalt valmis väikesed komponendid nagu näiteks nupud, sisestusväljad jne. Antud komponendid on modulaarsed ja neid on lihtne sättida vastavalt oma vajadustele. Samuti on neil tavaliselt vaikimisi CSS stiil ette kirjutatud, mis teeb arenduskäigu mugavamaks. Sisuliselt erinevad UI komponendid varasemalt

mainitud komponentidest vaid selle võrra, et need on kellegi teise poolt loodud. Suure osa kasutajaliidese lihtsamate komponentide jaoks kasutati *Reactstrap* teeki, mis sisaldab hõlpsasti taaskasutatavaid ja kontrollitavaid React Bootstrap 4 komponente (*Input*, *Label*, *Table*) ning samuti *Material-UI* komponente (*Text-Field, Checkbox*).

Kasutajaliidese visuaalne disain ei olnud projekti alguses kindlalt ära määratud, vaid testimise eesmärgil paluti arendusmeeskonnal kasutada samu värviskeeme ja proovida taasluua ETS NORD köögikubude konfiguraatori väljanägemist. Esialgsed stiilinõuded teostati kasutades Chrome veebibrauseri Inspect funktsiooni, mille abil sai köögikubude konfiguraatorilt kätte värvid, teksti suurused ja muu vajalikud stiilielemendid. Selline lahendus oli kliendile arenduskäigu alguses piisav. Hilisema arenduse käigus edastati meeskonnale ETS NORD *brand book* ning graafilise disaineri poolt köögikubude veebilehe Adobe XD prototüübid ja litsentseeritud veebifondid (Joonis 30). Adobe XD tegi kogu stiili implementeerimist oluliselt lihtsamaks, sest graafilise disaineri loodud ekraanipildid ja prototüübid konverteeritakse elemendi kaupa CSS reegliteks. Iga graafilise elemendi nagu näiteks ikoonid saab vajadusel salvestada ka eraldi pildi- või vektorfailina (SVG).



Joonis 30. ETS NORD konfiguraatori disainielemendid programmis Adobe XD.

4. Analüüs ja järeldused

Käesolev peatükk käsitleb projekti nõuetele vastavust, projektis kasutatavate tehnoloogiate valikut, analüüsitakse projekti arhitektuuri ja disaini ning tehakse selle põhjal vastavad järeldused. Lisaks antakse ülevaade projekti protsessist ja kitsaskohtadest.

4.1 Tehnilise teostuse põhjendus

Järgnevalt esitatakse projekti tehnilise teostuse põhjendus, kus selgitatakse nõuete sisu, põhjendatakse arhitektuur ja disain, tuuakse välja milliseid reegleid järgiti koodi kirjutamisel ja kuidas rakendus on testitud.

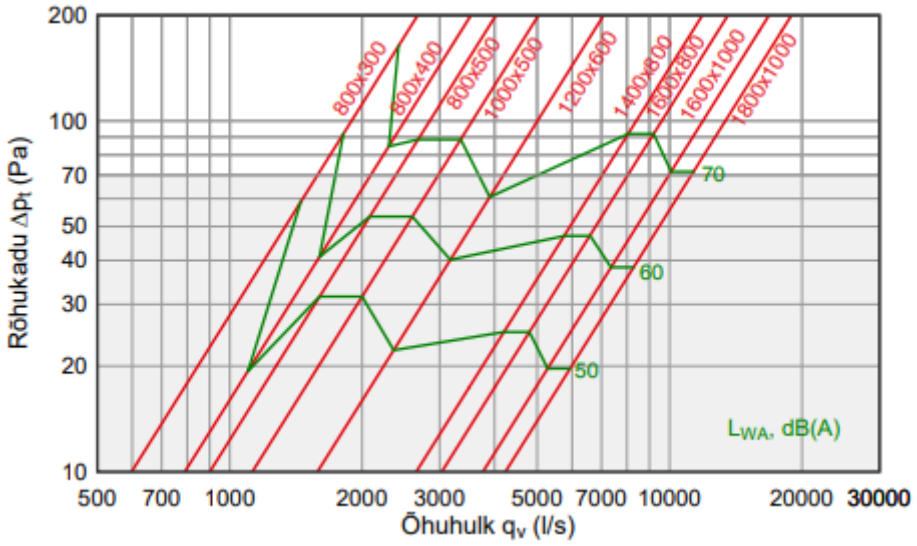
4.1.1 Nõuded

Nõuded kirjeldati eelkõige lähtuvalt ideest manuaalset toote konfigureerimist kiirendada, et kasutajale vajamineva toote valimine ei oleks nii aeganõudev ja ebamugav. Rangeid tehnoloogilisi nõudeid ettevõtte poolt ei esitatud, ainukese tehnoloogilise nõudena oli teada, et rakendus peab tulevikus olema ühildatav ettevõtte arenduses oleva *Webshop* rakendusega. Projektis kasutatavad tehnoloogiad ja platvormid on valitud arendusmeeskonna poolt tuginedes õppejõudude soovitustele ning varasemale kasutuskogemusele. Visuaalsed disainiõuded kinnitati ETS NORD *brand book*'i ja Adobe XD prototüübi failidega.

Üks ettevõte nõuetest sisaldas võimalust filtreerida katuseotsikuid õhuvoolu või rõhukao järgi. Selle väljaarvutamiseks oli vaja kasutusele võtta spetsiaalne valem. Ettevõte esitas meeskonnale nõuded ekraanipiltidena, mis kirjeldasid funktsiooni ja selle tulemust, kuid mitte seda kuidas konkreetselt tulemuseni jõuda. Probleemi lahendamiseks oli vaja logartimilistelt graafikutelt kättesaadav informatsioon välja lugeda ning seejärel valemisse rakendada.

Esialgu prooviti graafikutelt visuaalne info kätte saada masinlugemise abil, kasutades selleks vabavara rakendust WebPlotDigitizer. WebPlotDigitizer võimaldab mistahes graafiku pildina üles laadida ning määrata teljestiku nurgapunktid. Rakendus ehitab punktide järgi mällu logaritmilise koordinaatteljestiku, mille abil saab määrata valitud punkti konkreetsed koordinaadid. Antud lahendus tõi toimivaid, kuid osaliselt ebatäpseid tulemusi tingitud mõõtemääramatusest.

Hiljem sai meeskond ettevõttelt iga toote suuruse kohta graafikul ühe täpse punkti koordinaadid (Joonis 31). Lisaks tuli meeskonnal leida vastav valem, mida ettevõte ei teadnud ning arendajad pidid selle leidmiseks küsima abi nii juhendajatelt kui ka teiste teaduskondade spetsialistidelt.



Joonis 31. Mudeli ULV2P õhuhulga-rõhukao graafik logaritmilisel skaalal.

Valem õhuvooluhulga arvutamiseks:

$$L = k * \sqrt{p}$$

kus

L – õhuvooluhulk (m3/s),

p – rõhulangus üle kohttakistuse (Pa)

k – numbriline konstant ilma ühikuta, mis on erinev igal tootegrupil ja igal toote suurusel.

Tuletades valemist konstandi “k”, ning asendades valemisse ühe etteantud punkti koordinaadid, kus x-koordinaat on õhuhulk ja y-koordinaat rõhulangus, on võimalik arvutada konstant. Kui on teada ühe mudeli suuruse kindel konstant, siis on võimalik mis tahes õhuvoolu või rõhukao sisestamisel kätte saada selle mudeli eeldatav õhuvool või rõhukadu. Konstandid defineeriti andmebaasis ning neid kasutatakse katuseotsikute õhuvoolu või rõhukao arvutamisel.

4.1.2 Arhitektuur

Tarkvara loomisel peetakse silmas kahte osa: tarkvara funktsionaalsus ning tarkvara kvaliteet. Funktsionaalsus tähendab, kui palju tarkvara teeb ning mitut erinevat funktsiooni see täidab, kuid kvaliteet rõhub tarkvara korrektsusele, efektiivsusele, abstraktsusele ning võimalusele tarkvara taaskasutada ja laiendada. “Kvaliteet esimesena” mudel väidab, et ainuke osa, mis tarkvaras kasvab on funktsionaalsus ning kvaliteet peaks olema konstantselt täiuslik. See võib olla põhjuseks, miks antud projektis jäi implementeerimata äriloogika kiht. Põhifookuses oli tarkvara funktsionaalsuse kasvamine, kuid seda raskemaks muutus kvaliteedi parandamine tagantjärgi. Kvaliteet peab olema alati korras ning kui ei ole, siis tuleb see korda teha ja seda kõike enne uue funktsiooni loomist. Äriloogika kihi puudumine antud rakenduse funktsionaalsust hetkel ei mõjuta, sest andmeid ainult küsitakse kasutaja poolt mitte ei lisata ning valideerida tuleb ainult üksikuid sisestusvälju. Sellegipoolest abstraktse ja taaskasutatava koodi loomine võimaldaks tulevikus lihtsamini uut funktsionaalsust lisada. [13]

Projekti alguses pidi meeskond otsustama, kas projekt teostatakse traditsioonilise mitmeleherakendusena (MPA) või üheleherakendusena (SPA), sest projektile eelnevatel semestritel oli käsitletud mõlemaid. Oluline erinevus seisneb selles, et SPA korral toimuvad uuendused kasutajaliideses dünaamiliselt ühel lehel. SPA kommunikeerib serveriga JSON formaadis asünkroonselt, seega muudetakse vaid kuvatavaid andmeid, mitte ei värskendata tervet lehte. Mitmeleherakendustes esitatakse uute andmete korral päring serveripoolsele rakendusele, mis kombineerib andmed ning vaate, mille järel tagastatakse lehitsejale uuenenud andmetega HTML leht. Kuna konfiguraator toetub kasutaja sisendile ja interaktsioonile, tagab lehevärskenduste puudumine mugava ja voolava kasutajakogemuse. SPA nõrkusteks on halb SEO ehk kättesaadavus otsingumootoritele neis sisalduva JavaScript’i tõttu, mis käesolevas projektis oli ebaoluline puudus. Nõuete poolt kirjeldatud konfiguraator teostati üheleherakendusena eelkõige sellepärast, et toote konfigureerimise protsess oleks kiirem. MPA puhul oleks kogu lehe taaslaadimine olnud aeglustav faktor, mis ei oleks sobinud ettevõte nõuetega. [5]

Eelnevatele ülikooli õppeainetele tuginedes otsustas meeskond projekti jagada kaheks rakenduseks, millest üks on kliendipoolne (React) ja teine on serveripoolne (.NET CORE 3.0 API) rakendus. Alternatiivselt oleks saanud projekti teostada ühe monoliitse rakendusena, kuid meeskond soovis et projektid oleksid eraldi testitavad ja hallatavad. .NET CORE raamistiku

eeliseks on Entity Framework Core tööriist, mis võimaldab disainida keerulisi andmebaase ilma SQL-i kirjutamata. EF Core võimaldab kasutada LINQ (*Language-Integrated Query*) päringukeelt, mille abil on võimalik andmeid andmebaasist küsida ja neid manipuleerida C# keeles. LINQ päringus liiguvad andmed loogilises järjekorras vasakult paremale, SQL puhul on järjekord juhuslikum ning mahuka päringu puhul võib kood muutuda pikaks ja mitteloetavaks. React teegi valiku põhjuseks oli varasem kogemus TalTechi õppeainest "Infosüsteemide arendus IV". Alternatiivseteks valikuteks olid Vue, Blazor ja Angular teegid. Blazorit kaaluti, sest seda on võimalik kirjutada meeskonnale tuttavas C# keeles, kuid tegu on uue tehnoloogiaga ja abimaterjalide kättesaadavus oli piiratud. [14, 15]

Rakenduses on kasutatud REST API-t, kuid REST-ile leidub lisaks alternatiive. GraphQL on päringukeel, millega on võimalik teha HTTP päringuid sarnaselt REST API-le. Kuigi tegu on päringu keelega, ei ole see otseselt ühenduses andmebaasiga nagu SQL. GraphQL-i eeliseks on võimalus küsida täpselt selliseid andmeid nagu vaja – ei rohkem ega vähem. Peatükis 3.2 jooniselt 5 on näha, et toodete tabelis on esindatud kõik toote omadused, sest REST API päring ei võimalda täpsustada, milliseid omadusi vaates on vaja. Kui mõned omadused on kasutaja jaoks väiksema tähtsusega, siis on neid oluline kuvada alles toote detailvaates. Serverile spetsiifilisi päringuid võimaldab teha GraphQL. Hasanuddini ülikoolis viidi läbi jõudluse uuring GraphQL-i ja REST API vahel. Uuringu eesmärgiks oli välja selgitada, kumma reaktsiooniaeg on kiirem kasutajale andmete saatmiseks. GraphQL-i keskmine reaktsiooniaeg jäi REST API reaktsiooniajale alla ligikaudu 1000 millisekundiga. Seega REST API on jõudluse poolest kiirem, kuid päringutega võib esineda andmete üle pärimist (informatsiooni pakutakse kliendile rohkem kui vaja) või ala pärimist (informatsiooni ei ole ühes päringus piisavalt, mille tõttu peab tegema mitmekordseid päringuid). Kuna meeskond oli varasemalt tuttavam REST API-ga ning võttes arvesse nende plusse ja miinuseid, otsustati kasutada REST teenust. [16, 17]

Pilvandmetöötluse platvormide Azure App Service ja Netlify funktsioon on rakenduse võõrustamine veebiserveris, et rakendus oleks kasutajatele kättesaadav kindlal veebiaadressil. Projekti alguses prooviti rakendus juurutada kasutades Dockerit. Dockeri serveripoolne seadistamine oli keeruline ning võttis ebaproportsionaalselt palju aega. Azure ja Netlify kasutavad taustal samuti Dockeri konteinereid, kuid serveripoolne seadistus tehakse kasutaja eest ära. Visual Studio IDE ja Azure App Service on mõlemad Microsofti omandid, seega rakenduse juurutamine Azure'i oli väga mugavaks tehtud ning selle kasuks otsustatigi. Netlify's

on iga kuu 300 minutit tasuta *build time*'i ning 100GB tasuta andmemahtu, mis oli arenduse tempo jaoks piisav ning selle tõttu juurutati kliendipoolne rakendus Netlify keskkonda.

Esialgu sisestati kõik tooteandmed andmebaasi läbi klassi *ProductContext*, kuid suure hulga erinevate toodete kirjeldamiseks kulus juba tuhandeid koodiridu, mistõttu ei olnud rakendus enam efektiivselt hallatav ega kooskõlas puhta koodi põhimõtetega. Probleemi parandamiseks võeti kasutusele tasuta veebiteenus Google Sheets API, mis võimaldab internetis olevast .xls failist andmeid lugeda ja neid sinna kirja panna. Google Sheets API kasutamiseks peab looma Google kontoga uue Cloud Platform projekti, kus saab seejärel Sheets API käivitada. Samuti tuleb serveripoolsesse rakendusse alla laadida Google.Apis.Sheets.v4 NuGet pakett ning anda rakendusele ka API autoriseerimisvõti.

4.1.3 Disain

Projekti disaini planeerimisel konsulteeriti tihedalt juhendajaga kindlustamaks jätkusuutliku ning efektiivse andmemudeli valikut. Juhendaja soovitusel järgiti arhitektuuri paikapanemisel eelkõige J. Arlow ja I. Neustadt poolt kirjeldatud *Product* arhetüüpi, mida rakendati ka andmebaasi mudeli loomisel [1]. Konfiguraatori idee seisneb selles, et tootele saaks lisada omadusi juurde. Seda ei võimalda vaid ühe klassi realiseerimine, kus on defineeritud kõik tooteomadused atribuutidena. Selliselt läheneti rakendusele meeskonnaprojekti alguses, et oleksid olemas andmed, mida kasutajaliideses kuvada. Kui aga toote klassi defineerida kollektsioon erinevatest omadustest, siis on võimalik omaduste lisamine, muutmine ja kustutamine – selliselt loodi esimene üks-mitmele seos klassi *Product* ja *Feature* vahel. Selleks, et vältida korduvaid kirjeid andmebaasis, moodustati *Feature*'i asemel kaks klassi: *FeatureValue* ja *ProductFeature*. Esimene kirjeldab omaduse väärtust ning teine omaduse nime. Loodi üks mitmele seos kahe klassi vahel, kus üks *ProductFeature* saab olla mitmel *FeatureValue*'l. Toode võib olla näiteks särk või püksid ning samamoodi on erinevat tüüpi tooteid ventilatsiooni seadmetes. Realiseeriti klass *ProductType*, mida saab igal tootel olla üks, kuid üks tüüp saab olla mitmel erineval tootel, sarnaselt *ProductFeature*'i ja *FeatureValue*'iga. Sellise loogikaga saab ära defineerida kõik standardtooted andmebaasis.

Listi erinevatest omadustest, mida on võimalik eelnimetatud omaduste kollektsiooni lisada nimetatakse *AdditionalFeatureValue* klassiks. Seega on vaja toote külge lisada ka kollektsioon erinevatest lisatavatest, muudetavatest omadustest. *FeatureValue* eraldati kaheks klassiks: *MainFeatureValue* ja *AdditionalFeatureValue*, mis pärivad klassi *FeatureValue*.

Projekti käigus selgus, et teatud toote omadused on hoopis eraldiseisavad (teise tüübiga) tooted, kuid arendusmeeskond pidas neid tooteid alguses lisatavateks, muudetavateks omadusteks (*AdditionalFeaturevalue*). Näiteks tootele A saab kuuluda toode B ja C ning toode B ja C võivad kuuluda veel tootele D. Toote klassile tuli moodustada endale viitav mitu-mitmele seos ning nüüd oli toote küljes kaks uut kollektsiooni: *ChildProducts* ja *ParentProducts*. Sellist seost ei saa muud moodi luua kui vaheklassi realiseerimisega, mille nimeks sai *ProductRelation*. Uued kollektsioonid on *Product* tüüpi (kollektsioon toodetest), mis tähendab, et neid pidi saama konfigureerida vastavalt nendele kuuluva *AdditionalFeatureValue* kollektsiooni järgi.

Projekti alguses ei arvestatud, et andmebaasis olevate kirjete talletamine erinevates keeltes esineb väljakutseks. Staatiliste andmete tõlkimine on kergem, sest .resx failides saab kõik kirjed erinevates keeltes ära defineerida ja koodis viidata vaid sõna võtmele. Dünaamiliste andmete kriteerium on järgmine: kui klassi atribuut on tekstitüüpi väli ning selle väärtus kirjutatakse vastavalt kultuurile erinevalt, tuleb see ka andmebaasi kirja panna erinevates keeltes. Otsides erinevaid lahendusi veebisaidilt Stack Overflow ning küsides sisendit ettevõttes Helmes töötavalt vanemarendajalt, võeti kasutusele *Translations* klassid. Klassid, kus defineeritakse atribuudid, mis vastavad eelmisele kriteeriumile, st identifikaatorid ja muud kultuuri neutraalsed atribuudid sinna ei kuulu. Igal klassil (v.a *ProductRelation*) on individuaalne *Translations* kollektsioon ning kuna tegu on kollektsiooniga saab keeli vastavalt vajadusele juurde lisada, millega loodi üks-mitmele seos. Sellist protsessi, kus tõlgitav tekst kohandatakse lõppkasutaja kultuuriruumi, nimetatakse lokaliseerimiseks.

Objektorienteeritud keeled nagu C#, Java, C++ ei mõista SQL keelt ning aastakümneid põimiti SQL lauseid koodi sisse, kasutades selleks ADO.NET raamistikku. Looma pidi mitmeid komponente selleks, et andmed andmebaasi jõuaks ning kirjutatud kood oli habras. Kui muuta näiteks tabelis veeru nime, siis kompilaator veast ei teavita, aga koodi käivitades esineb tõrge (*exception*). Paljud probleemid lahenesid tänu objekt-relatsioonilise kaardistaja (ORM) tööriistadele. Üheks selleks tööriistaks on antud projektis Entity Framework Core, mis võimaldab andmebaasi tabeleid luua *entity* tüüpi klasside (andmemudeli klassid) järgi. See tähendab, et igale tabelile andmebaasis vastab rakenduses konkreetne klass mis vastab M. Fowler'i kirjeldatud mustrile *Concrete Table Inheritance* [8.1]. Andmebaasi loomine vastavalt klassidele viitab sellele, et rakendati "kood esimesena" lähenemisviisi, vastupidiselt sellele, et kood kirjutatakse eksisteeriva andmebaasi põhjal (andmebaas esimesena). Lisas 5 on andmebaasi disainimine, kasutades ORM-i. [18, 19]

Kindla elemendi otsimist suuremahulistest andmekogumitest võib osutuda ajakulukaks. Kui tegu on dünaamilise massiiviga, siis ühele objektile juurdepääs on O(n) keerukusega, kus n on objektide koguarv massiivis. Projekti alguses moodustati andmestruktuur just selliselt, kus iga toote küljes on massiiv erinevatest toote omadustest. Selgus aga, et kasutajaliideses oli tarvis kindlatele toote omadustele (nt toote mudel) ligi pääseda, kuid selleks peab läbi käima terve massiivi, et leida sealt soovitud element, mis tähendab, et otsene ligipääs puudus. Lahenduseks sai *hash* funktsiooni kasutamine, kus element otsitakse andmekogumist üles elemendi võtme järgi, millega saavutatakse konstantne keskmine keerukus O(1). Toote omadused paigutati *dictionary* tüüpi andmestruktuuri (Joonis 23) ning kindlale omadusele pääseb ligi järgnevalt: *Product.Features["model"]*, kus võtmeks on *model*. Sarnaselt on paigutatud *dictionary* tüüpi andmestruktuuri toote konfigureeritavad omadused (Joonis 24), mis võimaldab võtme järgi kasutajaliideses elemente kaardistada märkeruutudena (Joonis 6). [20]

SQL vaade on oma olemuselt päring, mis loob virtuaalse tabeli etteantud reaalsete baastabelite põhjal, mis andmebaasis eksisteerivad. Üldjoones ei ole see kriitiliselt vajalik, kuid see lihtsustab mõnevõrra arendust ning ei nõua palju lisamälu. Juhul kui kasutajal on vaja mingisuguseid andmeid erinevatest tabelitest, saab talle ette anda ühe tabelina just need konkreetsed andmed, mis on talle aktuaalsed. See toimib mõneti ka turvamehhanismina, sest kasutajale ei pea andma ligipääsu andmebaasis olevatele reaalsetele tabelitele. Vaated võimaldavad end dünaamiliselt muuta kui mõnes baastabelis peaks toimuma muudatus. [21]

Rakenduse arendamise käigus võeti eeskuju GRASP (*General Responsibility Assignment Software Patterns*) mustritest. See on kogum üheksast fundamentaalsest printsiibist, mis leiavad objekt-orienteeritud programmeerimises kasutust objekti disainimise ja vastutuse näol. Mustrid juhendavad looma puhast disaini, kus igal klassil on kindel ülesanne ning vastutus. [22]

Controller on klass, mis vahendab liiklust kasutajaliidese ning loogikakihi vahel. Kui kasutaja vajutab kasutajaliideses nuppu, käivitades päringu, saab kontroller sellest esimesena teada ja juhib vastavalt sisendile vajalikke komponente, et täita etteantud operatsioon. Käesoleva projekti näitel järgib *Controller* mustrit .NET CORE API klass *ProductsController*, mis saab kasutajaliidesest päringu ning käivitab seejärel vastavad operatsioonid.

Creator (looja) on muster, mis kehtestab kes peaks vastutama objektide loomise eest. Hea kandidaat *Creator* klassile võiks olla näiteks selline klass, mis kasutab loodavat klassi või omab

selle initsialiseerimiseks andmeid. Projekti serveripoolses rakenduses on klass *ProductRepository*, mis vastutab vajaminevate *ViewModel*'ite valmistamise eest.

Indirection (vahemees) on muster, mis toob kahe või rohkema komponendi vahele uue vastutava vahemehe, et hoida ära klasside omavahelist otsest sidumist. Vahemehe kasutamine tagab koodi kõrgema taaskasutatavuse ja madalama sidususe, sest osapooled ei suhtle omavahel otse. Projekti näitel ei küsita kasutajaliides andmeid otse andmebaasist, vaid selleks on loodud vahemees *ProductRepository*, mis pöördub andmebaasi poole ning tagastab vajaminevad *ViewModel*'id.

Gang of Four disainimustrid, mida on kokku 24 tükki, on samuti objekt orienteeritud programmeerimise disainimustrid. Need jagunevad *Creational* -, *Structural* - ning *Behavioral* liiki mustriteks [23]. *Gang of Four* disainimustreid rakenduse arendamisel ei kasutatud, sest esmakordse tellimusprojekti loomisel oli väga raske lähtuda mustritest, mida varem implementeeritud ei oldud.

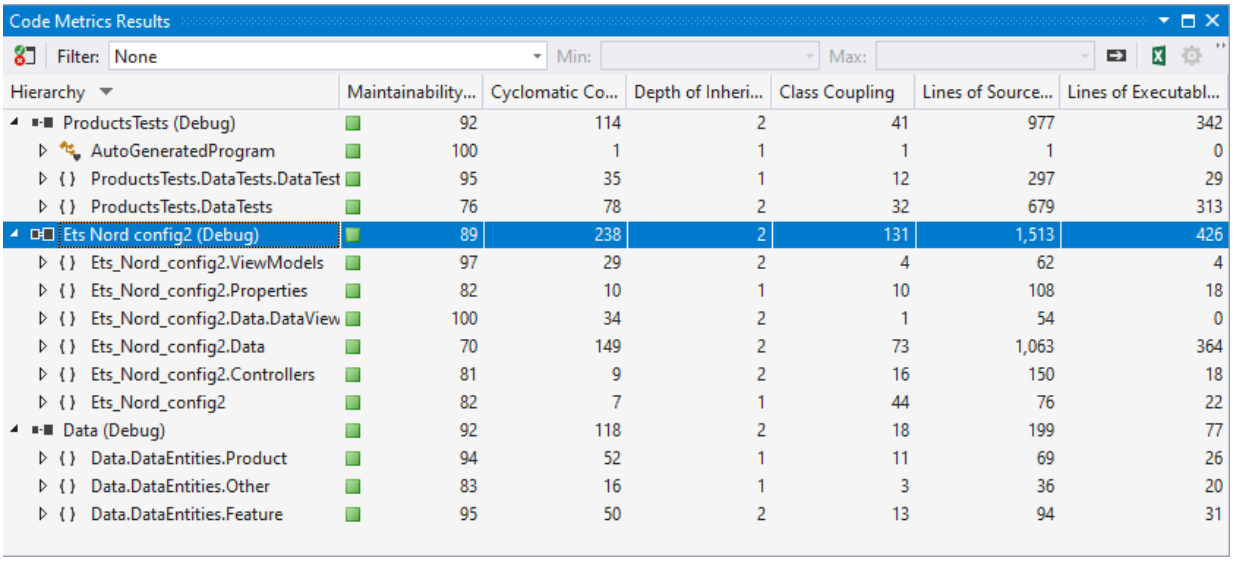
4.1.4 Kood

Rakendus on kirjutatud kahes osas: serveri rakenduse osa on valminud C# keeles ning .NET raamistikus. Kliendipoolse rakenduse osa on valminud React teegis ning kasutatud keelteks on TypeScript, CSS, HTML, JavaScript.

GitLab'i analüüsikohaselt on kogu rakenduse peale 42,14% koodist C# programmerimiskeeles, 48,41% on TypeScript ning 3,68% on CSS ja 1,29% HTML. Sellest saab järeldada, et kogu rakenduse peale on serveri rakendus ja kliendipoolne rakendus koodiridade poolest ligikaudu võrdsed. Serveri rakenduses on kolme erineva projekti peale 2689 rida koodi, millest põhiprojektis on 1513 rida, testimist teostav projekt koosneb 977 koodireast. *Data* projekt, mis koosneb 199 reast, on põhiprojektist eraldi lahku tõmmatud osa, mis tagab seosed andmebaasi tabelite vahel (Joonis 32).

Cyclomatic Complexity, mis mõõdab arvuliselt ja lineaarselt eraldiseisvaid kooditeid ehk *path*-e läbi rakenduse lähtekoodi, on üks mõõdikutest Visual Studio 2019 programmis koodi meetrika analüüsimiseks [24]. Tsüklomaatilise keerukuse näitaja lähtekoodil on 238. Jagades selle näitaja projekti meetodite arvuga 120, tuleb projekti keskmiseks näitajaks ühele meetodile ligikaudu kaks. Näiteks meetodil, mis sisaldab vaid ühte *if-else* tsüklit on tsüklomaatiline keerukuse

näitaja alati kaks, sest kood saab kompileeruda vaid kahte erinevat teed pidi. Sellest saab järeldada, et tsüklomaatilise keerukuse näitaja jääb lähtekoodil normaalsuse piiridesse. Projekti *Maintainability index* ehk hoolduse indeks, mis näitab kui lihtne on koodi hooldada ja muuta on kolme projekti keskmisena 91/100. Selle põhjal saab järeldada, et projekt on hoolduse indeksi järgi pigem lihtsasti hooldatav.



Hierarchy	Maintainability...	Cyclomatic Co...	Depth of Inheri...	Class Coupling	Lines of Source...	Lines of Executabl...
ProductsTests (Debug)	92	114	2	41	977	342
AutoGeneratedProgram	100	1	1	1	1	0
ProductsTests.DataTests.DataTest	95	35	1	12	297	29
ProductsTests.DataTests	76	78	2	32	679	313
Ets Nord config2 (Debug)	89	238	2	131	1,513	426
Ets_Nord_config2.ViewModels	97	29	2	4	62	4
Ets_Nord_config2.Properties	82	10	1	10	108	18
Ets_Nord_config2.Data.DataView	100	34	2	1	54	0
Ets_Nord_config2.Data	70	149	2	73	1,063	364
Ets_Nord_config2.Controllers	81	9	2	16	150	18
Ets_Nord_config2	82	7	1	44	76	22
Data (Debug)	92	118	2	18	199	77
Data.DataEntities.Product	94	52	1	11	69	26
Data.DataEntities.Other	83	16	1	3	36	20
Data.DataEntities.Feature	95	50	2	13	94	31

Joonis 32. Visual Studio 2019 Code Metrics.

Tiimiga järgiti koodi kirjutamisel eelnevalt kokkulepitud ühtset struktuuri. Serveri rakenduse arendamisel järgiti Robert C. Martini "Clean Code" raamatu põhimõtteid. Nimetamisel kasutati sisuka tähendusega nimesi klassidele ning muutujatele. Funktsioonide juures järgiti DRY-printsiipi, mis tähendab otsetõlkes, et ära korda ennast. DRY-printsiibi põhimõte seisneb duplikatsioonide vältimises. Lisaks järgiti funktsioonide juures, et funktsioon täidaks ühte konkreetset ülesannet, mis aitab hoida puhast struktuuri ja vältida kõrvalmõjusid. Koodi organiseerimisel järgiti, et kood oleks grupeeritud funktsionaalsuse kaupa. Meetodid järjestati klassides ülevalt alla väljakutsumise järjekorra alusel. Koodi kirjutamisel järgiti, et klassid oleksid võimalikult kõrge kokkukuuluvusega ja täidaksid vaid oma vastutusalale kohaseid operatsioone. [25]

SOLID printsiipidest järgiti klasside ja liideste puhul *The Single Responsibility Principle*'i, mis deklareerib seda, et igal klassil peaks olema konkreetne vastutusala. Teine SOLID printsiip, mida järgiti oli *Open-Closed Principle*, mis tähendab seda, et rakenduse klassid ja funktsioonalsus peaksid olema avatud laiendustele, kuid suletud muutusteks. [25]

Kliendipoolse rakenduse arendamisel lähtuti samuti mitmest puhta koodi põhimõttest, mis kehtivad Reactis TypeScript'ile, CSS'le ja HTML'le. Puhta koodi nõuete järgimise eesmärgiks oli eelkõige loetava ja arusaadava koodi kirjutamine, mis garanteerib, et projekti edasiarendamine tulevikus on lihtsam. Puhas kood aitab paremini mõista klasse ja meetodeid inimestel, kes liituvad arendusmeeskonnaga hiljem ning näevad koodi esimest korda. Üks elementaarsemaid reegleid, mida järgiti Reactis TypeScripti kirjutamisel oli see, et üks fail peaks hõlmama ainult ühte Reacti komponenti, kuid erandina esineb olukord, kus mitu *stateless* (komponent, mis ei kasuta hetkeseisu) või *pure* komponenti (komponent, mis renderdab vaid siis, kui tema enda hetkeseis või *propsid* muutuvad) võivad olla ühes failis. [26]

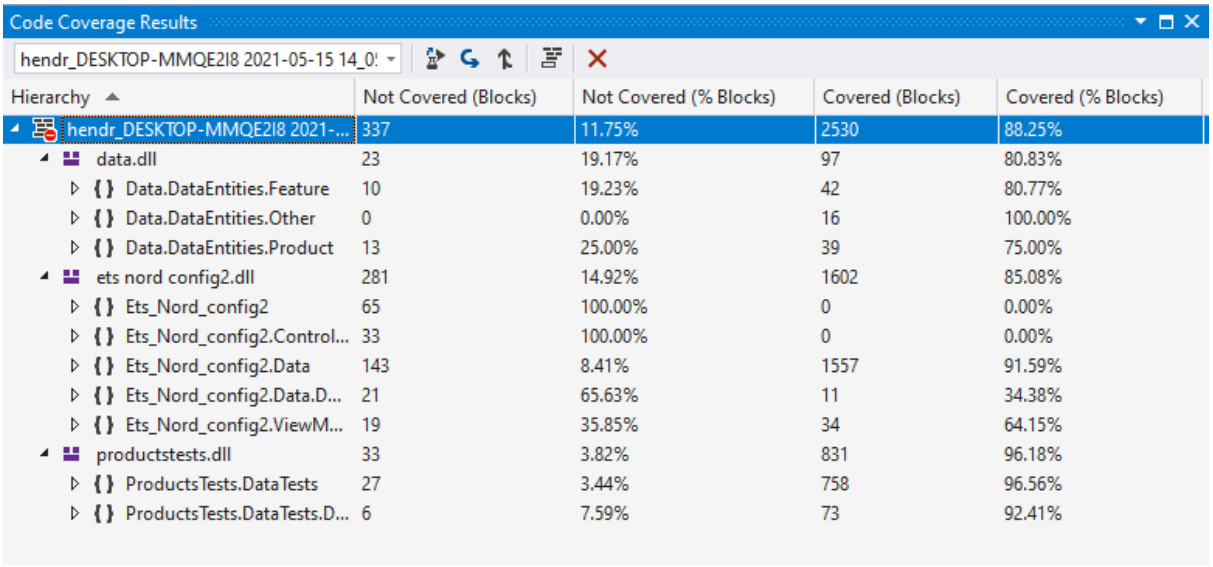
Nimetamisel lähtuti mitmest reeglist: puhtas TypeScript keeles kirjutatud failid peaksid lõppema .ts liitega ning failid, mis sisaldavad JSX-elemente peaksid lõppema .tsx liitega. Reacti failide ja komponentide nimetamisel kasutati *PascalCase-i*. Komponentide *instance*'ite ning samuti *props*'ide nimetamisel kasutati *camelCase*-i. Failide, komponentide, meetodite ja muutujate nimed loodi baseerudes nende funktsionaalsusele või rollile rakenduses ehk iga üksuse nimi kirjeldab tema ülesannet. Rakenduses järgiti muutujate ja meetodite deklareerimisel kindlalt järjekorda. [26]

Kliendipoolse rakenduse komponentide struktuur failis on ülevalt alla järjekorras:

- *Hooks* - funktsioonid, mis kasutavad hetkeseisu;
- Redux *variables* - Redux'i globaalsed muutujad;
- *State variables* - hetkeseisu omastavad muutujad;
- *ClickHandlers* ja *EventHandlers* - funktsioonid, mis käivitakse mingi sündmuse järel;
- *UseEffect* - käsklus, mis sõltuvuste muutumisel käivitab uue renderi;
- *Return* - tagastab HTML-i, mis määrab lehe visuaalse struktuuri;

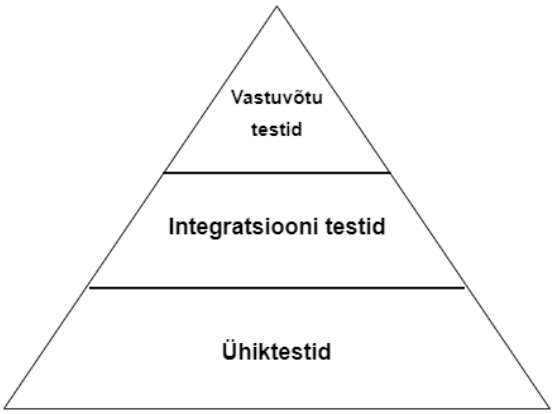
4.1.5 Testid

Rakenduse testimiseks on kasutatud MSTest testimisteeki. Testprojekt katab kogu rakenduse 88.25% ulatuses (Joonis 33), see hõlmab endast nii ühikteste kui ka integratsiooni teste. Rakenduse testid on automatiseeritud sellisel viisil, et kui uus versioon rakendusest laetakse pilve ülesse, siis enne juurutamist kontrollib versioonihaldus tarkvara automaatselt, kas kõik testid läbitakse.



Joonis 33. Visual Studio 2019 Test Analytics.

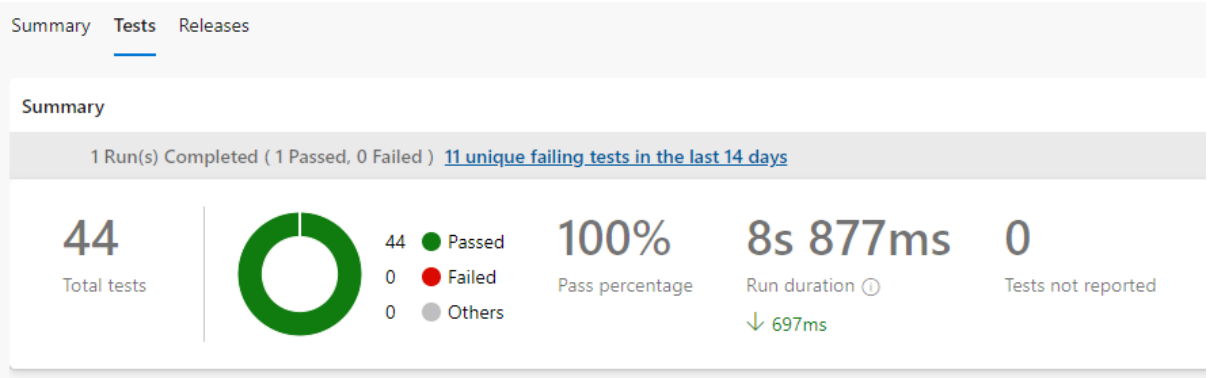
Testimise struktuur tugines testimise kolmnurga põhimõttele (Joonis 34). Mike Cohn'i poolt välja mõeldud testimise kolmnurk koosneb kolmest kihist: need on ühiktestid, integratsiooni testid ning kasutajaliidese testid, mida nimetatakse ka vastuvõtu testideks. Ühiktestid, mis on kõige kiiremad, kuid ka kõige lihtsamad olemuselt, peaksid moodustama kõige suurema osakaalu kogu testide osakaalust. Püramiidi keskmises osas asuvad integratsiooni testid, mis kasutavad rakenduse andmebaasi ning on suurema keerukusega. Integratsiooni testid kontrollivad rakenduse erinevate osade koostoimimist ning nende osakaal peaks olema 20-30% kogu testidest. Kõige väiksema osakaaluga on vastuvõtu testid, mis on ka olemuselt kõige keerukamad ja aeglasemad, kuid see eest kasutavad rohkem rakenduse kihte ja on põhjalikumad. Vastuvõtu testide osakaal testimise kolmnurga põhimõtte järgi on ligikaudu 10-15%. [27]



Joonis 34. Testimise kolmnurk. [27]

Serveri rakenduse vastuvõtutestide jaoks kasutati Postman platvormi, kust saab hea ja kiire ülevaate API erinevat tüüpi HTTP päringutest ning millega saab neid päringuid testida ja automatiseerida integreerides need CI/CD (*Continuous Integration/Continuous Delivery) pipeline*'i. Kuigi rakenduses loogikakiht otseselt puudub, siis antud testid kontrollivad, et kõikide komponentide omavaheline suhtlus toimiks korrektselt ning presentatsioonikihist liiguksid vastavad andmed edasi soovitud kujul. Kui ühiktestidega kontrollitakse komponente klasside kaupa ning testid ei ole moodulite osas omavahel sõltuvuses, siis API testid on seotud kogu serveripoolse rakenduse lõpliku väljundiga ning neid saab kontrollida kui rakendus töötab tervikuna. [28]

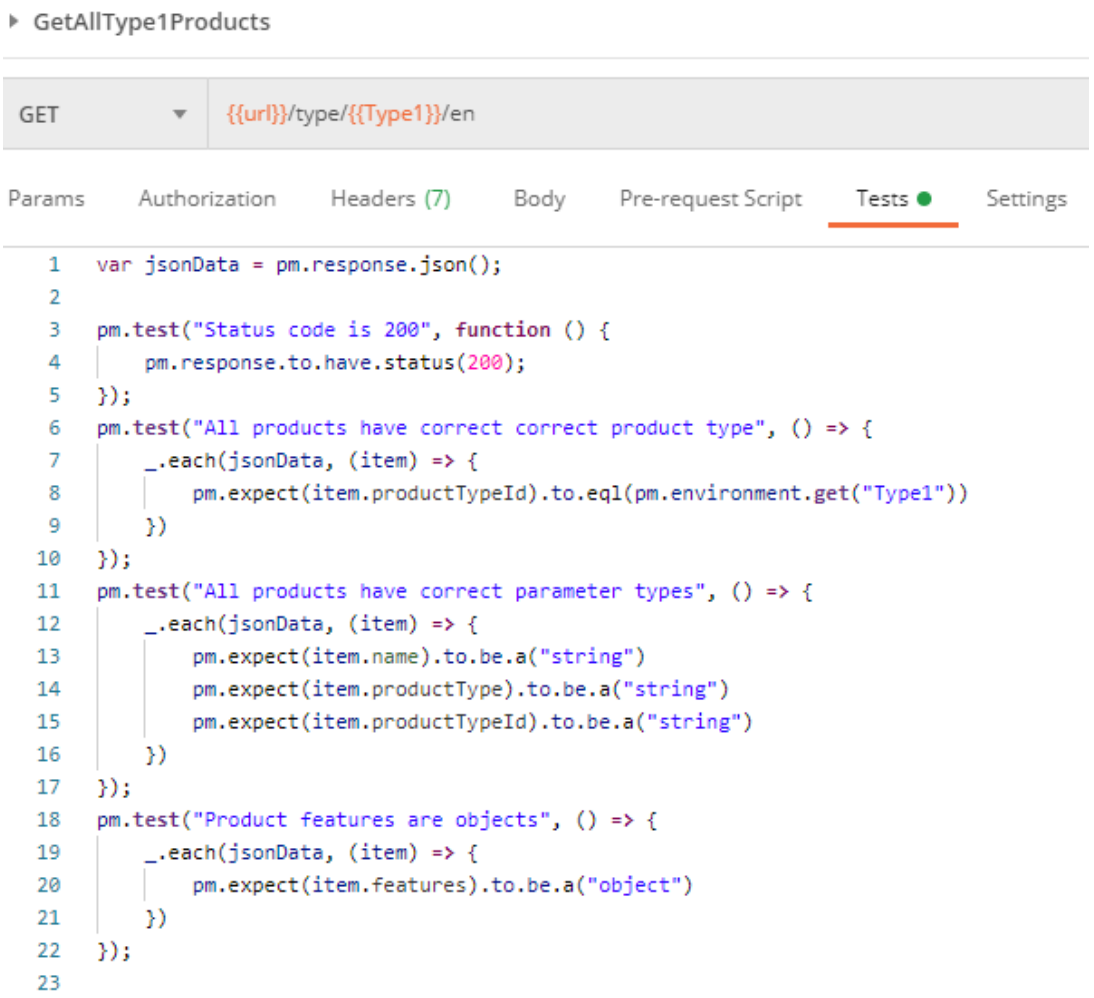
Postmani platvormil on kokku 44 vastuvõtu testi (Joonis 35). Antud rakenduse puhul testiti ainult GET päringuid. Teist tüüpi päringud nagu näiteks PUT, POST või DELETE rakenduses ei ole (Joonis 36).



Joonis 35. Postmani testide aruanne Azure pipeline'is.

Joonis 36. Postmani päringud.

Näiteks päring GetAllType1Products tagastab kõik tooted, mille tüübi Id on “1”, ning selle juures testitakse, et kõik tagastatud tooted oleksid katuseotsikud ning objekti parameetrite tüübid oleksid vastavad. Lisaks kontrollitakse, et toote omadused oleksid objekti kujul tootele külge lisatud (Joonis 37). Päringuga GetType1ProductsWithConstant testitakse, kas tootele on andmebaasis lisatud konstant, millega arvutatakse vastavalt õhuvool ning rõhukadu. Seega selles päringus kontrollitakse, et andmebaasis olevate katuseotsikute arv vastaks päringus tagastatud katuseotsikute arvule. Samuti on testitud päringud toodete erinevate parameetrite sisestuse korral, et tagastatud väärtused oleksid vastavuses andmebaasis olevate objektidega. Kõikide päringute juures on ühiselt testitud, et kõik päringud õnnestuksid ja leitaks vaste.



Joonis 37. GetAllType1Product päringu testid.

Üks näide integratsioonitestidest on GetProductsByParameterAllModelsTest (Joonis 38), mis testib serveripoolse rakenduse GetProductsByParemeter meetodit. GetProductsByParameter meetod võtab parameetriteks erinevad omadused, mille järgi saab tooteid filtreerida. Nendeks on toote mudel, mõõtmed, läbiviigu kuju, funktsioon (õhu sissevõtt ja -väljavise), rõhukadu ning läbitav õhuvool. Testmeetod testib, kas tooteid saab filtreerida kõikide mudelite järgi. Selleks võetakse andmebaasist kõik võimalikud mudeli väärtused ning antakse see parameetriks. Seejärel käiakse *foreach* tsükliga kõik vasted läbi ja kontrollitakse, et vastete eeldatav toote mudel vastaks tegelikule toote mudelile.

```
[TestMethod]
0 references | Risto-Raul Pajula, 33 days ago | 2 authors, 3 changes
public void GetProductsByParameterAllModelsTest()
{
    products = repository.GetAllProducts("1", "et").Result.Value;
    foreach (var x in products)
    {
        var expectedModel = x.Features["model"].Value;
        List<MainFeature> productParameters = new List<MainFeature>();
        var actualProducts = repository.GetProductsByParameters(productParameters,
            expectedModel, null, null, "1", null, null, "en").Result;
        foreach (var y in actualProducts)
        {
            var actualModel = y.Features["model"].Value;
            Assert.AreEqual(expectedModel, actualModel);
        }
    }
}
```

Joonis 38. GetProductsByParameterAllModelsTest.

4.2 Kirjanduse ülevaade

Projekti koostamisel ning arendamisel võeti aluseks erinevad teaduslikud allikad ning ettevõtte poolt etteantud näidisprogrammid. Soovitud raamatud saadi O'Reilly e-raamatukogust. Teadusartikleid leidis meeskond eelkõige IEEE Xplore e-raamatukogust. Näidisprogrammideks oli ettevõtte teise tootegrupi konfiguraator (ETS NORD köögikubu konfiguraator) ning konkurendi (Climecon) katuseotsikute konfiguraator [2].

Tootekonfiguraatorite empiirilise uuringu põhjal saavad autorid välja tuua, et konfiguraatorite keerukus seisneb eelkõige toote heas mõistmises ning selle disainis. Konfiguraatori arendus ja disain on kaks täiesti eraldiseisvat tegevust, sest tarkvarainsener ei ole tingimata hea konfiguraatori disainer. Hea konfiguraatori arendamiseks peab olema selle funktsioon ja vajalikkus disaineri poolt täpselt defineeritud, et selle otstarve ja eesmärk oleks arusaadav kõigile osapooltele. Peaaegu pooltel juhtudest konfiguraatorite arendamisel on probleemiks, et äripoole ja arendajate vaheline kommunikatsioon ei ole kõige parem. Samuti on konfiguraatorite arenduse ning haldamise kitsaskohaks toodete keerukuse tase ehk mida spetsiifilisem on toode, seda keerulisem on ka konfiguraatori disain. [29]

Uuringust sai paralleele tõmmata ka antud projekti arendamisega. Ventilatsiooniseadmete valdkond oli meeskonnale projektieelselt täiesti võõras, oli sellega täpsemalt tutvumine

omakorda aeganõudev protsess arendamise kõrvalt. Kui esialgsed nõuded toodete osas olid meeskonnale selgeks saanud, tuli poole arendusperioodi pealt juurde mitmeid nõudeid toodete keerukuse osas. Võimalike lisanõudmiste tekkega ei osatud alguses arvestada, mistõttu tuli suur osa arhitektuurist ümber muuta vastavalt uutele nõuetele. Ilmnenud lisatöö oli tingitud ebaselgest kommunikatsioonist, kus arendusmeeskond ei saanud kohe aru oodatavatest tulemustest.

Ettevõte ETS NORD kasutas soovitud kasutajaliidese selgitamiseks konkurendi Climecon rakendust ja ETS NORD köögikubude konfiguraatorit [2]. Näidised andsid visuaalse ülevaate, millist funktsionaalsust tulevaselt rakenduselt oodatakse. Peamiselt võeti eeskuju toote otsingu parameetritest ja lehekülje paigutusest.

Projekti andmemudeli koostamisel kasutati J. Arlowi ja I. Neustadti raamatus käsitletud *Product* arhetüüpi. See muster annab võimaluse tooteid tõhusalt modelleerida. Arhetüübil põhinev rakenduse arhitektuur moodustab paindliku ärisüsteemi, mida saab vastavalt ärivajadustele laiendada. *Product* valdkonnamustrit ei implementeeritud täielikult, vaid võeti kasutusele osa sellest. Antud arhetüübile realiseeriti juurde täiendavad klassid vastavalt projekti vajadusele. [1]

Tootekonfiguraatoreid on võimalik disainida erinevatel viisidel. Konfiguraatorid jagunevad automaatseteks ja manuaalseteks ning nende peamiseks erinevuseks on, et automaatsed konfiguraatorid võtavad arvesse ka valikute tõenäosuse esinemist konfigureerimise protsessis. 2007. aastal viidi läbi uuring, mis keskendus konfiguraatorite efektiivsuse parandamisele. Uuringu tulemusena disainiti toote konfiguraator, kus kasutaja sisestab oma eelistused ning toote esinemise tõenäosusega vastavas konfiguratsioonis kuvatakse kasutajale kindel toode. Selleks kasutati informatsiooni teooria algoritmi, mis arvutas välja vajaliku toote konfiguratsiooni efektiivsemalt ja vähema sisendiga. Mõte seisneb selles, et valitakse välja kõige rohkem informatsiooni andev komponent, et elimineerida ülejäänud valikud ja sealhulgas ka määramatus. Sellise lähenemise eesmärk on anda konfiguraatorile kindel konfigureerimise järjekord, mis võimaldab kasutajal täpsustada oma vajadused, mille tagajärjel konfigureeritakse toode, mis kirjeldab tema vajadusi kõige paremini. Algoritmi disainimiseks kasutati *Bayesian* võrgustikku, mis on graafiline mudel muutujatest ja nende tingimuslikest sõltuvustest läbi suunatud atsüklilise graafi. Võrreldes sellist toote konfiguraatorit käesoleva lõputöö projekti omaga, seisneb peamine erinevus selles, et toote konfigureerimiseks ei ole kasutatud graafil põhinevat algoritmi, vaid lähteandmete sisestamine käib manuaalselt. [30]

4.3 Teostatud tööde detailne kirjeldus logi vormis

Tabel 1. Nädalad 1-4 (31.08.2020 - 21.09.2020).

Nädal	**Risto-Raul**	**Hendrik**	**Oskar**	**Mark**
1. 31.08-06.09	- Projekti kaardistamine ning ülesehituse planeerimine. -Koosolek ettevõtte esindajaga. -Materjalide lugemine ja läbi töötlemine	- Projekti kaardistamine ning ülesehituse planeerimine. -Koosolek ettevõtte esindajaga. -Materjalide lugemine ja läbi töötlemine	- Projekti kaardistamine ning ülesehituse planeerimine. -Koosolek ettevõtte esindajaga. -Materjalide lugemine ja läbi töötlemine	- Projekti kaardistamine ning ülesehituse planeerimine. -Koosolek ettevõtte esindajaga. -Materjalide lugemine ja läbi töötlemine
2. 07.09-13.09	-Juhendajaga arhitetektuuri paika panek. -Kasutajalood -API loomine esialgsete andmetega	-Juhendajaga arhitetektuuri paika panek. -Kasutajalood -API loomine esialgsete andmetega	-Juhendajaga arhitetektuuri paika panek. -Kasutajalood -API loomine esialgsete andmetega	-Juhendajaga arhitetektuuri paika panek. -Kasutajalood -API loomine esialgsete andmetega
3. 14.09 20.09	-API *Repository* ja controller -Esialgse kasutajaliidese loomine -Otsing nime ja funktsiooni järgi API’s	-API *Repository* ja controller -Esialgse kasutajaliidese loomine -Otsing nime ja funktsiooni järgi API’s	-API *Repository* ja controller -Esialgse kasutajaliidese loomine -Otsing nime ja funktsiooni järgi API’s	-API *Repository* ja controller -Esialgse kasutajaliidese loomine -Otsing nime ja funktsiooni järgi API’s
4. 21.09-27.09	-Otsing mitme mudeli järgi API’s -Mudelite otsingu implementeerimine kasutajaliidesse	-Otsing mitme mudeli järgi API’s -Mudelite otsingu implementeerimine kasutajaliidesse	-Otsing mitme mudeli järgi API’s -Kasutajaliidese disain ja korrastamine	-Otsing mitme mudeli järgi API’s -Kasutajaliidese disain ja korrastamine

Tabel 2. Nädalad 5-9 (28.09.2020 - 01.11.2020).

Nädal	**Risto-Raul**	**Hendrik**	**Oskar**	**Mark**
5. 28.09-04.10	-Ettevõttele esimene demo -Andmebaasi normaliseerimine	-Ettevõttele esimene demo -Andmebaasi normaliseerimine	-Ettevõttele esimene demo -*Project info* salvestamine ja clearimine	-Ettevõttele esimene demo -*Project info* salvestamine ja clearimine
6. 05.10-11.10	-*Details* vaade API *fetch*'iga -Uue arhitektuuri toote *search* probleemi lahendamine juhendajaga	-*Details* vaade API *fetch*'iga -Uue arhitektuuri toote *search* probleemi lahendamine juhendajaga	-*Details* vaate esialgne vorm koos vajalike väljadega -Uue arhitektuuri toote *search* probleemi lahendamine juhendajaga	-*Details* vaate esialgne vorm koos vajalike väljadega -Uue arhitektuuri toote *search* probleemi lahendamine juhendajaga
7. 12.10-18.10	-Juhendajaga API arhitektuuri korrigeerimine -*2nd branch* API	-Juhendajaga API arhitektuuri korrigeerimine -*2nd branch* API	-Juhendajaga API arhitektuuri korrigeerimine -React CSS(*search*)	-Juhendajaga API arhitektuuri korrigeerimine -React CSS(*search*)
8. 19.10-25.10	-Ettevõttele teine demoesitlus -PDFi lisamine -Kasutajaliidese toote *details, fetching* kindla toote kohta	-Ettevõttele teine demoesitlus -PDFi lisamine -Kasutajaliidese toote *details, fetching* kindla toote kohta	-Ettevõttele teine demoesitlus -PDFi lisamine -*Details* metalli lisamine -Tootepildid + Graafik	-Ettevõttele teine demoesitlus -PDFi lisamine -*Details* metalli lisamine -Tootepildid + Graafik
9. 26.10-01.11	-Reacti arhitektuuri muutmine(meetodid klassist -> consti) -Graafikute lugemine	-Reacti arhitektuuri muutmine(meetodid klassist -> consti) -Graafikute lugemine	-*Details* vaate sisene korrastamine -Graafikute lugemine	-*Details* vaate sisene korrastamine -Graafikute lugemine

Tabel 3. Nädalad 10-15 (02.11.2020 - 13.12.2020).

Nädal	**Risto-Raul**	**Hendrik**	**Oskar**	**Mark**
10. 02.11-08.11	-APIs andmebaasi *view*'de loomine/muutmine	-APIs andmebaasi *view*'de loomine/muutmine	-Ettevõtte *brand book*'i järgi *front-endi* disain -Data klassi testimine	-Ettevõtte *brand book*'i järgi *front-endi* disain -Data klassi testimine
11. 09.11-15.11	-Demo esitlus ettevõttele -Muudatuste analüüsimine -Täieliku andmebaasi loomine täpsete andmetega	-Demo esitlus ettevõttele -Muudatuste analüüsimine -Täieliku andmebaasi loomine täpsete andmetega	-Demo esitlus ettevõttele -Muudatuste analüüsimine -Täieliku andmebaasi loomine täpsete andmetega	-Demo esitlus ettevõttele -Muudatuste analüüsimine -Täieliku andmebaasi loomine täpsete andmetega
12. 16.11-22.11	-Konfigureeritavate väljade loomine -Väljade sidumine PDFiga	-Konfigureeritavate väljade loomine -Väljade sidumine PDFiga	-*ProductCode* loomine -*Details* vaate disain	-*ProductCode* loomine -*Details* vaate disain
13. 23.11-29.11	-Projekti *back-end* ja *front-endi* üles panek -APIsse lisa *search* meetod	-Projekti *back-end* ja *front-end*i üles panek -APIsse lisa *search* meetod	-Dockeri ja andmebaasi ühendamine ja sellega lähemalt tutvumine	-Dockeri ja andmebaasi ühendamine ja selle lähemalt tutvumine
14. 30.11-06.12	-Demo esitlus ettevõttele -Muudatuste üle vaatamine ja analüüs -*Back-end*'i arhitektuuri muutmine	-Demo esitlus ettevõttele -Muudatuste üle vaatamine ja analüüs -*Back-end*'i arhitektuuri muutmine	-Demo esitlus ettevõttele -Muudatuste üle vaatamine ja analüüs -Docker uurimine ja viewd loomine	-Demo esitlus ettevõttele -Muudatuste üle vaatamine ja analüüs -Dockeri uurimine ja viewde loomine
15. 07.12-13.12	-Refaktoorimine reactis (*common functions*)	-API refaktoorimine -Toote andmete PDF aadresside tabeli loomine -Õhuvool ja rõhukadu Reacti	-Adobe XD uurimine	-Adobe XD uurimine

Tabel 4. Nädalad 16-21 (14.12.2020 - 21.02.2021).

Nädal	**Risto-Raul**	**Hendrik**	**Oskar**	**Mark**
16. 14.12-20.12	-"*Load more*" nupp -Projekti Azure'i ja Netlifysse üles panek	-Muudatuste sisse viimine -Projekti live panemine	-Muudatuste sisse viimine -Uue *brand book*'i ja Adobe XD järgi disainimine	-Muudatuste sisse viimine -Uue *brand book*'i järgi disainimine
17. 18.01-24.01	-Ümara ja kandilise ühenduse valik juures raadionupuga -Refaktoorimine ja vigade parandused -Uus *web deploy* (vahetasin *subscriptionit*)	-Ettevõttele demo + koosolek -Kaalu atribuudi lisamine tootele -*ProductType* tüüp tegemine API-sse -Suuruse järgi otsingu parandamine	-Ettevõttele demo + koosolek +*ProjectInfo* + *ProjectDyn* + *Search* CSS refactor	-Ettevõttele demo + koosolek -Pisivigade parandamine/muutmine
18. 25.01-31.01	- Lokaliseerimine (keele valik) kasutajaliideses algus -Staatiliste ridade tõlkimine	-Toote graafikute *zoom* funktsionaalsus -Toodete ja nende omaduste andmebaasi lisamine	-*Popover* komponent -*ProductTable* muutmine -ETS NORD footer	-Toote värvi kood validatsioon -*On hover* pilt
19. 01.02 07.02	-Resource tabel tuleb excelist, mis kirjutab read jsoniks ja siis resource failideks -Peale igat buildi loeb excelit ja värskendab faile	-*LocalStorage* ja *SessionStorage* Reacti väljade jaoks	-*Dynamic Footer* -*ProjectInfo Form Validation testing with Material UI*	-Mudelite search lahtri lahendused -Tõlkimine + ettevõtte *input*'i küsimine tõlkimisel
20. 08.02-14.02	-Uus andmemudel lokaliseerimiseks -Andmebaasi andmete tõlkimine -Andmete pärimine keele valiku põhjal	-*LocalStorage* ja *SessionStorage* Reacti väljade jaoks	-*Formik* + *Yup react-hook-form validation testing*	-Postmani keskkond ning esialgsed testid
21. 15.02-21.02	-Ühe toote pärimine detaili vaate jaoks API-lt -Õhuvoolu ja rõhukadu on URL-is parameetrid (Reactis) -Omaduste lisamine ja uuendamine viidud Redux *store*'i	-*ProductDetails* ja avalehe CSS kujundamine ja ümbermuutmine	-*ProductDetails Tabs* komponendi demo	-Üksiku toote pärimine API-lt Reactis

Tabel 5. Nädalad 22-26 (22.02.2021 - 28.03.2021).

Nädal	**Risto-Raul**	**Hendrik**	**Oskar**	**Mark**
22. 22.02-28.02	-Ettevõttele demo + koosolek -Kasutajaliideses sisestusväljad viidud üle märkeruutudeks	-Ettevõttele demo + koosolek -Andmebaasi andmete sisestamine uue loogika järgi -Tootekood uue loogika järgi	-Ettevõttele demo + koosolek -*Details tabs* komponent refactor	-Ettevõttele demo + koosolek -Kasutajaliidesse sisestus *checkbox*'id
23. 01.03-07.03	-Laadimine tehtud filtreerimise väljadele päringu ajal	-Testimine, *mock-repository* loomine	-Ühiktestid	-Esialgsed ühiktestid
24. 08.03-14.03	-Ettevõttele demo + koosolek	-Ettevõttele demo + koosolek	-Ettevõttele demo + koosolek	-Ettevõttele demo + koosolek -Ühiktestid
25. 15.03-21.03	-Uue loogika välja mõtlemine, kuidas lisada toote külge featureid, mida saab konfigureerida -Kuidas eristada vaikimisi väärtuseid ja konfigureeritavaid väärtuseid -*Feauture*'i mudeli viimine uude andmestruktuuri (dictionary)	-Integratsiooni testid -Testid repositooriumi meetodite kohta	-Ühiktestid -Integratsioonitestid -*ImgPopover fix* -*Search* muudetud.	-Azurega ühendamine/ülespanek -Integratsioonitest
26. 22.03-28.03	-Ettevõttele demo + koosolek -Konfigureeritavate omaduste mudeli loomine -*AdditionalFeatures* lokalisatsioon -Reacti loogika ümber tegemine, sest omadused on uues andmestruktuuris -Ühenduse kuju küsitakse läbi võtme väärtuse	-Ettevõttele demo + koosolek -Andmestruktuuride muutmine *Dictionary*-ks koos Risto-Rauliga -Ettevõtte Azure kasutajasse andmebaasi viimine -*AdditionalFeatures* struktuuri muutmine, -*FeatureValue* id muutmine	-Ettevõttele demo + koosolek -Toodete Excelist andmebaasi lugemine -*FakeData* loomine	-Ettevõttele demo + koosolek -Ettevõtte Azurega ühendamine/ülespanek

Tabel 6. Nädalad 26-28 (22.03.2021 - 11.04.2021).

Nädal	**Risto-Raul**	**Hendrik**	**Oskar**	**Mark**
27. 29.03-04.04	-*AdditionalFeatureViewModel* ei ole *ProductViewModeli* küljes, vaid nende vahel on sõltuvus (küsitakse eraldi meetodiga toote id põhjal) -Sheet API loomine: kõik andmed mida lisatakse on nüüd Google Sheedis	-Integratsiooni testid -Testid repositooriumi meetodite kohta	-*ProjectInfo refactor* -*Repository* tests	-Postman testid + Snapshot test
28. 05.04-11.04	-Tootele endale viitava mitu-mitmele seose loomine -*Cart* loogika reactis: tooteid saab lisada nö ostukorvi ja lõplik ostukorv on hulk tooteid, mida saab API-le tagasi saata -Omaduste lisamine erinevatele toodetele ja nende muutmine ostukorvis	-Koosolek juhendajaga -Testide katsetus *pipeline*'is -*Cart* loogika arutamine -Kanaliühenduste sõltuvus katuseläbiviigu isolatsiooni paksusest	-Koosolek juhendajaga	-Koosolek juhendajaga -Postman testid + Snapshot test
29. 12.04-18.04	-*Childproducts.tsx* klass kõik dünaamiliselt kaardistatud märkeruutudena -Ostukorvis saavad olla ainult unikaalse tüübi id-ga tooted -Ettevõttele demo + koosolek	-Ettevõttele demo + koosolek -Test andmebaasi URL aadressi turvaliseks tegemine -Repositooriumi meetodite refaktoorimine	-Ettevõttele demo + koosolek	-Ettevõttele demo + koosolek -Ühiktestid (vana arhitektuur)
30. 19.04-25.04	-Koosolek juhendajaga -*MatchProduct* meetod vastava artikli leidmiseks -Postman testid pipeline'i -Ettevõttele demo + koosolek	-Koosolek juhendajaga -Integratsiooni testide refaktoorimine -Õhuvoolu ja rõhukao valemi refaktoorimine	-Koosolek juhendajaga -Nimede muutmine -Ebavajalike failide eemaldamine	-Koosolek juhendajaga -Postman testide muutmine uuele branchile

4.4 Hinnang projekti teostamise protsessi kohta

4.4.1 Projekti teostamise protsess

Projekti arendamine kestis meeskonnal kokku 30 nädalat. Ettevõtte esindajaga saadi esimest korda kokku 28. augustil 2020. aastal, kus tutvustati meeskonnale pealiskaudselt projekti nõudeid ning soovitud tulemust. Arendustöödega lõpetati 25. aprill 2021. aastal. Kindlaid tööpäevi tiimiga paika ei pandud, kuid igale meeskonnaliikmele oli ette nähtud ettevõtte poolt 16 töötundi nädalas. Selle sisse arvestati projekti arendamine kui ka erinevate vajalike tööriistade uurimine ja õppimine antud projekti tarbeks.

Projekti ettevõttepoolne esindaja oli ETS NORD AS tootearendusjuht. Ettevõtte poolsete juhendajatega saadi kokku enamasti iga kahe nädala tagant. Lisandunud funktsionaalsusest tehti demo, et saada sellele ettevõtte tagasisidet. Samuti arutati, mida võiks muuta või teisiti realiseerida, ning prooviti leida mõlemale osapoolele võimalikult sobilik lahendus. Meeskonnaga tehti iga nädala alguses koosolek, kus pandi paika nädala töö eesmärk, räägiti läbi murekohad ning jagati ära ülesanded. Projekti arendamisel kasutati paarisprogrammeerimise metoodikat, millest tulenevalt jagati suuremad ülesande ära paari peale ning väiksemad ülesanded teostati individuaalselt. Samuti tehti iga iteratsiooni lõpus koosolek, kus võeti kokku eelnev arendusetapp.

4.4.2 Hinnang, millised olid projekti protsessi puhul kitsaskohad

Projekti käigus esines kitsaskohti ning elemente, mida ei jõutud teostada. Esinemise põhjusteks olid peamiselt teadmiste nappus, kommunikatsiooniprobleemid ning ettevõtte poolse IT-toe puudumine. Üheks probleemiks oli katuseotsikute õhuvoolu ja rõhulanguse arvutamine. Ettevõte andis esialgu ainult pildi formaadis toodete logaritmilised graafikud, mille põhjal pidi meeskond välja mõtlema meetodi, kuidas oleks võimalik graafikute pealt tuletada kindel valem, mille järgi saab arvutada vastavalt õhuvoolu ja rõhukao. Alles pärast kontakteerumist TalTechi spetsialistidega, jõudis meeskond selle valemini. Teiseks raskuskohaks oli üldine ettevõtte poolne IT-toe puudumine, mis tähendab seda, et kogu infosüsteemi arendamiseks, pidi meeskond lähtuma ainult enda teadmistest ning IT-küsimuste korral ettevõtte seest abi ei saanud. Lisaks sellele, kui esialgse funktsionaalsusega rakendus valmis sai oli ettevõttel probleeme uute nõuete välja töötamisega ja oma nõudmiste väljendamisega. Kuna ventilatsioon on võrdlemisi spetsiifiline valdkond ja meeskonnal puudusid selles valdkonnas teadmised,

tekkisid raskused sellega, kuidas mingisugused tooted ja nende abitooted omavahel sobituvad, mis tegi konfiguraatori arendamise protsessi raskemaks. Üheks väljakutseks oli projekti ja meeskonnatöö juhtimine. Kuna tiimis konkreetset juhti ei olnud siis vahepeal esines raskusi töö ja töörollide jaotamisega.

Viimaseks eesmärgiks oli rakenduse ühildamine ettevõttepoolse *Webshop* infosüsteemiga. See tähendab lõppkasutaja jaoks konfiguraatorist valitud toodetega edasi liikumist teisele domeeni aadressile, et seal ost vormistada. Andmete vahetamiseks kahe infosüsteemi vahel, peavad nii *Webshop* kui ka katuseotsikute konfiguraator aktsepteerima kindlat andmestruktuuri. Osapooled leppisid kokku sobiva andmestruktuuri ning arendus sai jätkuda. Ettevõtte poolt on kindlad tooted ära defineeritud artiklitega (tootekoodid), mis tähendab antud rakenduse jaoks seda, et peale toote konfigureerimist, peab rakendus otsima kõikide toodete kombinatsioonide seast sobiva artikli. Kui vastavat artiklit ei leitud, siis see tähendab, et sellist olemasolevat toodet ei eksisteeri ning tootele jääb märge, et tegu on konfiguratsiooniga. Selline funktsionaalsus ka realiseeriti, kuid lõpliku integratsiooni kahe infosüsteemi vahel teha ei jõutud. Põhjuseks oli ettevõtte hõivatus, tingitud muudest kriitilistest tööülesannetest.

4.4.3 Hinnang üldisele projekti teostamise protsessile

Ettevõtte esindajaks antud projekti puhul oli ETS NORD AS tootearendusjuht, kes andis projektitiimile vahetut tagasisidet rakenduse kohta ning suunas meeskonda nõuete osas. Tallinna Tehnikaülikooli poolt olid juhendajateks dr Gunnar Piho, kes andis meeskonnale nõuandeid rakenduse arhitektuuri ja disaini osas ning Viljam Puusep, kes aitas meeskonda pisemate tehniliste probleemide korral.

Töökorraldus kinnitati lepinguga, arvestades et iga töönädal peaks iga meeskonnaliige nädalas panustama fikseeritud hulga töötunde. Töötundide hulk pandi paika arvestades tudengite ülikooli tunniplaani ning õppemahtu. Kindlaid tööpäevi paika ei pandud, et tudengid saaksid olla paindlikud projekti arendamisel õpingute kõrvalt. Kuna ettevõte ei seadnud projektile tehnoloogilisi nõudeid, olid tööriistade valik ning projekti tarkvaraline juhtimine tudengite vastutuses.

Meeskond toimis efektiivselt, suuremaid konflikte ei esinenud ning alati leiti probleemidele lahendus. Töötegevus toimus esialgu eelkõige ülikooli rühmatöö ruumides ning eriolukorra süvenedes liikus suhtlus ja töö veebikeskkonda. Töö käigus said meeskonnaliikmed harjutada

tegevust erinevates vastutusalades, mida sellise projekti tegemine hõlmab. Meeskond sai vajalikke kogemusi täites rolle nagu analüütik, *front-end* arendaja, *back-end* arendaja, UI/UX disainer, infosüsteemi arhitekt, DevOps insener, andmebaasi insener.

4.5 Meeskondlik hinnang

Meeskondlik konsensuslik hinnang kõikide meeskonnaliikmete panuse kohta meeskonnas kujul: +2 panustas oluliselt rohkem kui teised; +1 panustas rohkem; 0 - panustas samaväärselt; -1 panustas vähem; -2 panustas oluliselt vähem.

Tabel 7. Meeskondlikud hinnangud tiimiliikmetele.

	Risto-Raul	**Hendrik**	**Oskar**	**Mark**
Hinnang	+2	0	0	0

Kõik meeskonnaliikmed täitsid oma ülesandeid korralikult ning suuremaid konflikte ega probleeme omavahel ei tekkinud. Risto-Raul Pajulale otsustas meeskond anda +2 lisapunkti, sest ta sai oma ülesannetega kiiremini hakkama kui teised, on suure osa rakenduse funktsionaalsuse autoriks, aitas teistel vajadusel nende ülesanded tehtud saada ning näitas välja suurt pühendumist projekti osas. Lisades 7, 8, 9 ja 10 on toodud kaasautorite hinnangud enda tehtud tööle ja panusele meeskonnas.

5. Kokkuvõte

Käesoleva bakalaureusetöö raames teostati ettevõtte ETS NORD AS tellimustööna projekt, mille eesmärgiks oli arendada infosüsteem. Peamiseks probleemiks oli toote valimise ja konfigureerimise protsess, mis oli kliendile ajakulukas ning süvenemist nõudev. Infosüsteemi peamine eesmärk oli katuseotsikute tootegrupi filtreerimine ja konfigureerimine.

Meeskond alustas infosüsteemi loomist täielikult algusest, sest tellimuse esitanud ettevõttel puudus tootekonfiguraatorite infosüsteem ja kasutatav andmebaas. Projekti arendus algas meeskonnaprojektina ning jätkus lõputöö raames vajalike edasiarenduste ja täiendustega. Nõuded kirjeldati ETS NORD AS konkurendi konfiguraatori põhjal ning vormistati kasutajalugudeks. Projekti juhtimiseks kasutati agiilse arenduse metoodikaid, töötades korduvates tsüklites, tarnides kliendile võimalikult tihti uuenenud tarkvara.

Lõputöö raames valmis rakendus, mis kiirendab tootevaliku protsessi. Rakendus võimaldab katuseotsiku tootegruppi filtreerida ning seejärel teatud omadusi muuta ning konfigureerida. Katuseotsikutele saab konfigureerida ka füüsiliseks paigalduseks vajaminevaid tooteid. Tellimuse saab salvestada PDF-dokumendina, mis kajastab kasutaja tehtud konfiguratsiooni. Rakendus ühendatakse tulevikus veebipoega, mis on ETS NORD-il veel arendamisel.

Serveripoolne rakendus tehti .NET Core raamistikus ja C# programmeerimiskeeles. Kasutajaliides arendati React teegis, milleks kasutati TypeScript programmeerimiskeelt. Lõputöö andis ülevaate kuidas kolmanda kursuse äriinfotehnoloogia tudengid said tellimustööna valminud projektiga hakkama, luues infosüsteemi, mis leiab päris maailmas kasutust ja teeb lõppklientide elu mugavamaks. Meeskonnaliikmed said kogemust erinevate infosüsteemide arendamise valdkonnas kasutust leiduvate rollide näol.

Kasutatud kirjandus

[1] J. Arlow, I. Neustadt, “Product archetype pattern,” in Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML. Addison-Wesley Professional, 2003. [Online]. Loetud aadressil: https://learning.oreilly.com/library/view/enterprise-patterns-and/032111230X/
Kasutatud: 27.04.2021

[2] Climecon, Dimensioning of the roof hoods, 2021. [Online]. https://katosx.climecon.fi/search.xhtml
Kasutatud: 10.05.2021

[3] D. Björner, Software Engineering 1: Abstraction and Modelling. Berlin, Heidelberg : Springer-Verlag, 2006.
Kasutatud: 30.04.2021

[4] J. Kouraklis, MVVM in Delphi: Architecting and Building Model View ViewModel Applications. Apress, 2016. [Online]. Loetud aadressil:
https://learning.oreilly.com/library/view/mvvm-in-delphi/9781484222140/
Kasutatud: 05.05.2021

[5] E. Scott, “What is a single page application,” in SPA Design and Architecture. Manning Publications, 2015. [Online]. Loetud aadressil:
https://learning.oreilly.com/library/view/spa-design-and/9781617292439/
Kasutatud: 06.05.2021

[6] M. Masse, “Introduction,” in REST API Design Rulebook, O'Reilly Media, 2011. [E-book]. Loetud aadressil: https://learning.oreilly.com/library/view/rest-api-design/9781449317904/
Kasutatud: 06.05.2021

[7] M. Thakkar, Building React Apps with Server-Side Rendering. Apress, 2020. [Online]. Loetud aadressil: https://learning.oreilly.com/library/view/building-react-apps/9781484258699/
Kasutatud: 08.05.2021

[8] M. Fowler, “Object-Relational Structural Patterns,” in Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. [Online]. Loetud aadressil: 09.05.2021
https://learning.oreilly.com/library/view/patterns-of-enterprise/0321127420/
Kasutatud: 13.05.2021

[9] A. Javeed, "Performance Optimization Techniques for ReactJS," 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), 2019, pp. 1-5, [Online].
https://ieeexplore.ieee.org/document/8869134
Kasutatud: 13.05.2021

[10] Vipul A M, P. Sonpatki, ReactJS by Example - Building Modern Web Applications with React, Pact Publishing, 2016. [E-book]. Loetud aadressil:
https://learning.oreilly.com/library/view/reactjs-by-example/9781785289644/
Kasutatud: 10.05.2021

[11] React, Handling Events, 2021. [Online]. https://reactjs.org/docs/handling-events.html
Kasutatud: 08.05.2021

[12.] Digital Ocean, Replacing Component Lifecycles with the useEffect Hook, 2020. [Online].
https://www.digitalocean.com/community/tutorials/react-replacing-component-lifecycles-with-useeffect
Kasutatud: 08.05.2021

[13] B. Meyer, “Practice to Perfect: The Quality Model,” IEEE, vol. 30, no. 5, pp. 103-105, 1997. [Online]. Loetud aadressil: https://ieeexplore.ieee.org/document/589917
Kasutatud: 13.05.2021

[14] J. Albahari, B. Albahari, “Comprehension queries,” in LINQ Pocket Reference. O'Reilly Media, 2015. [Online]. Loetud aadressil:
https://learning.oreilly.com/library/view/linq-pocket-reference/9780596519247/
Kasutatud: 12.05.2021

[15] LINQPad, Why LINQ beats SQL, 2020. [Online].
https://www.linqpad.net/WhyLINQBeatsSQL.aspx
Kasutatud: 12.05.2021

[16] D. A. Hartina, A. Lawi and B. L. E. Panggabean, "Performance Analysis of GraphQL and RESTful in SIM LP2M of the Hasanuddin University," 2018 2nd East Indonesia Conference on Computer and Information Technology (EIConCIT), 2018, pp. 237-240, https://ieeexplore.ieee.org/document/8878524
Kasutatud: 10.05.2021

[17] DEV Community, 4 reasons why you should use GraphQL over REST APIs, 2021. [Online]
https://dev.to/blessingartcreator/stop-using-rest-for-apis-53n
Kasutatud: 10.05.2021

[18] Microsoft Docs, Overview of Entity Framework Core, 2020. [Online].
https://docs.microsoft.com/en-us/ef/core/
Kasutatud: 09.05.2021

[19] S. Barskiy, “Introducing Entity Framework,” in Code-First Development with Entity Framework. Packt Publishing, 2015. [Online].
https://learning.oreilly.com/library/view/code-first-development-with/9781784396275/
Kasutatud: 11.05.2021

[20] P. E. Black, "data structure", in Dictionary of Algorithms and Data Structures [Online], Paul E. Black, ed. 15 December 2004. https://www.nist.gov/dads/HTML/dataStructure.html
Kasutatud: 09.05.2021

[21] Microsoft Docs, CREATE VIEW (Transact-SQL) - SQL Server, 2020. [Online].
https://docs.microsoft.com/en-us/sql/t-sql/statements/create-view-transact-sql?view=sql-server-ver15
Kasutatud: 11.05.2021

[22] C. Larman,"GRASP: Designing Objects with Responsibilities", in Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Pearson, 2004. [Online]. Loetud aadressil: https://learning.oreilly.com/library/view/applying-uml-and/0131489062/
Kasutatud: 04.05.2021

[23] E. Gamma, J. Vlissides, R. Helm, R. Johnson, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994. [E-book].
https://learning.oreilly.com/library/view/design-patterns-elements/0201633612/
Kasutatud: 05.05.2021

[24] Microsoft Docs, Calculate Code Metrics - Visual Studio, 2020. [Online].
https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2019
Kasutatud: 03.05.2021

[25] R. C. Martin, Clean Code, Pearson, 2008. [E-book]. Loetud aadressil:
https://learning.oreilly.com/library/view/clean-code-a/9780136083238/
Kasutatud: 05.05.2021

[26] Airbnb, Airbnb React/JSX Style Guide, 2021. [Online]. https://airbnb.io/javascript/react/
Kasutatud: 05.05.2021

[27] martinfowler.com, The Practical Test Pyramid, 2018. [Online]
https://martinfowler.com/articles/practical-test-pyramid.html
Kasutatud: 03.05.2021

[28] ToolsQA, API Testing With Postman, 2021
https://www.toolsqa.com/postman/api-testing-with-postman/
Kasutatud: 03.05.2021

[29] L. L. Zhang, P. T. Helo, A. Kumar and X. You, "Implications of product configurator applications: An empirical study," 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), 2015, pp. 57-61, Loetud Aadressil: https://ieeexplore.ieee.org/document/7385608
Kasutatud: 28.04.2021

[30] Y. Wang, M. M. Tseng, "An approach to improve the efficiency of configurators," 2007 IEEE International Conference on Industrial Engineering and Engineering Management, 2007, pp. 1332-1336, Loetud Aadressil:
https://ieeexplore.ieee.org/document/4419409
Kasutatud: 28.04.2021

Lisa 1 – Lõputöö lihtlitsents

Meie, Hendrik Laretei, Risto-Raul Pajula, Oskar Neemre, Mark Rõõmussaar

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "ETS NORD AS tootekonfiguraatori arendus ja analüüs", mille juhendajad on Gunnar Piho ja Viljam Puusep

    1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

    1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

12.05.2021

---

[1] Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Kasutusjuhtude kirjeldus

Konfiguraatori esimene vaade (Toote otsimine ja valimine)

- Katuseotsiku filtreerimine funktsiooni järgi:
  Kasutaja valib rippmenüüst katuseotsiku funktsiooni → Kasutaja vajutab nuppu "Otsi" → tagastatakse valitud funktsioonile vastavad tooted.

- Katuseotsiku filtreerimine kanali ühenduse järgi:
  Kasutaja valib rippmenüüst katuseotsiku kanali ühenduse → Kasutaja vajutab nuppu "Otsi" → tagastatakse valitud kanali ühendusele vastavad tooted.

- Katuseotsiku filtreerimine mitme mudeli järgi:
  Kasutaja valib rippmenüüst katuseotsiku mudeli (1-12) → Kasutaja vajutab nuppu "Otsi" → tagastatakse valitud mudelitele vastavad tooted.

- Katuseotsiku filtreerimine mõõtude järgi:
  Kasutaja valib rippmenüüst katuseotsiku standardmõõdu → Kasutaja vajutab nuppu "Otsi" → tagastatakse valitud standardmõõdule vastavad tooted.

- Katuseotsiku rõhukao arvutamine õhuvoolu ning mõõtude järgi:
  Kasutaja sisestab õhuvoolu → Kasutaja vajutab nuppu "Otsi" → tagastatakse rõhukadu sõltuvalt ristlõike pindalast ja õhuvoolust.

- Katuseotsiku õhuvoolu arvutamine rõhukao ning mõõtude järgi:
  Kasutaja sisestab rõhukao → Kasutaja vajutab nuppu "Otsi" → tagastatakse õhuvool sõltuvalt ristlõike pindalast ja rõhukaost ja vastavast arvulisest koefitsendist.

- Projekti informatsiooni lisamine viite välja:
  Kasutaja vajutab nuppu "Projekti Informatsioon" → Kasutaja sisestab viiele väljale projektiga seonduva informatsiooni

Konfiguraatori teine vaade (Toote konfigureerimine ja muutmine)

- Katuseotsiku edasine konfigureerimine:
  Kasutaja valib välja standardtoote, mis vastab tema nõutele ning mida ta soovib konfigureerida → Kasutaja läheb toote detailvaatele vajutades tootetabelis nupule “Vali” → Kasutaja valib tootele omadused, mida ta tahab väljavalitud katuseotsikule → Kogu tooteinfot on näha tabelist, mis muutub uute valikute tegemisel.

- Katuseotsiku materjali konfigureerimine:
  Kasutaja valib linnukestega katuseotsiku materjali, kusjuures igal otsikul on vaikimisi materjal juba määratud.

- Katuseotsiku kanali ühenduse konfigureerimine:
  Kasutaja valib linnukesega enda kanali ühenduseks sobiva liitmiku, sealjuures on kasutajal märkekastike “Spetsiaalne väärtus”, mis võimaldab sisestada väärtused ise.

- Katuseotsiku värvi konfigureerimine:
  Kasutaja sisestab neljast numbrist koosneva RAL-värvikoodi ettenähtud välja.

- Katuseläbiviigu konfigureerimine:
  Kasutaja valib linnukesega katuseläbiviigu → Kasutaja saab linnukestega konfigureerida katuseläbiviigu materjali ning kõrgust → Uue valitud toote informatsioon on kasutajale tabelist näha.

- Katuseläbiviigu ühenduse konfigureerimine:
  Kasutaja valib vajadusel linnukesega katuseläbiviigu ühenduse → Kasutaja saab linnukestega konfigureerida katuseläbiviigu ühenduse materjali → Uue valitud toote informatsioon on kasutajale tabelist näha.

- Toote tellimuslehe vaatamine:
  Kasutaja valib välja toote, mis vastab tema nõutele → Kasutaja konfigureerib toote omadustega, mida ta soovib → Kasutaja valib “Ekspordi PDF” → Kasutaja näeb PDF-dokumenti, kus on kirjas tootekood, too(de)te omadused, projekti informatsioon.

Lisa 3 – Andmemudeli atribuutide tabel

Klass	Atribuut	Kohustuslik?	Andmetüüp	Max pikkus	Näide
Product	Id	Jah	string	400	“1”; “10”; “167”;
	ProductTypeId	Jah	string	400	“1”; “10”; “167”;
	ProductType		**ProductType**		
	Features		ICollection<**MainFeatureValue**>		
	AdditionalFeatures		ICollection<**AdditionalFeatureValue**>		
	Translations		ICollection<**ProductTranslation**>		
	ChildProducts		ICollection<**ProductRelation**>		
	ParentProducts		ICollection<**ProductRelation**>		
ProductTranslations	Id	Jah	string	400	“1”; “10”; “167”;
	ProductId	Jah	string	400	“1”; “10”; “167”;
	Name	Jah	string	256	“Materjal”, “Mudel"
	CultureId	Jah	string	400	“1”; “10”; “167”;
	Product		**Product**		
	Culture		**Culture**		
ProductType	Id	Jah	string	400	“1”; “10”; “167”;
	Translations		ICollection<**ProductTypeTranslation**>		

Klass	Atribuut	Kohustuslik?	Andmetüüp	Max pikkus	Näide
ProductRelation	ParentId	Jah	string	400	“1”; “10”; “167”;
	ChildId	Jah	string	400	“1”; “10”; “167”;
	Parent		**Product**		
	Child		**Product**		
ProductTypeTranslation	Id	Jah	string	400	“1”; “10”; “167”;
	ProductTypeId	Jah	string	400	“1”; “10”; “167”;
	Type	Jah	string	256	“Katuseotsik”; “Katuseläbiviik”;
	CultureId	Jah	string	400	
	ProductType		**ProductType**		
	Culture		**Culture**		
FeatureValue	Id	Jah	string	400	“1”; “10”; “167”;
	ProductId	Jah	string	400	“1”; “10”; “167”;
	ProductFeatureId	Jah	string	400	“1”; “10”; “167”;
	Code		string	256	“ZM”; “H”;
	Product		**Product**		
	ProductFeature		**ProductFeature**		
MainFeatureValue	Translations		ICollection<**MainFeatureValueTranslation**>		

Klass	Atribuut	Kohustuslik?	Andmetüüp	Max pikkus	Näide
AdditionalFeatureValue	Translations		ICollection<**AdditionalFeatureValueTranslation**>		
	IsMandatoryValue		bool		false; true;
ProductFeature	Id	Jah	string	400	“1”; “10”; “167”;
	Unit	Jah	string	256	“kg”; “mm”;
	Translations		ICollection<**ProductFeatureTranslation**>		
FeatureValueTranslation	Id	Jah	string	400	“1”; “10”; “167”;
	FeatureValueId	Jah	string	400	“1”; “10”; “167”;
	CultureId	Jah	string	400	“1”; “10”; “167”;
	Value	Ei	string	256	“1600”; “Roostevabateras”;
	Culture		**Culture**		
MainFeatureValueTranslation	MainFeatureValue		**MainFeatureValue**		
AdditionalFeatureValueTranslation	AdditionalFeatureValue		**AdditionalFeatureValue**		
ProductFeatureTranslation	Id	Jah	string	400	“1”; “10”; “167”;
	ProductFeatureId	Jah	string	400	“1”; “10”; “167”;

Klass	Atribuut	Kohustuslik?	Andmetüüp	Max pikkus	Näide
ProductFeatureTranslation	Name	Jah	string	256	“Materjal”; “Pikkus”; “Mudel”
	CultureId	Jah	string	400	“1”; “10”; “167”;
	ProductFeature		**ProductFeature**		
	Culture		**Culture**		
Culture	Id	Jah	string	400	“1”; “10”; “167”;
	Language	Jah	string	256	“et”; “en”

Lisa 4 – Osaline toodete SQL vaade

ProductId	ProductTypeId	ProductName	Feature	Value	Unit	Code	FeatureId	ProductType	Language
77	1	ULV2I Roof hood	Width	200	mm	2	width	Roof hood	en
77	1	ULV2I Roof hood	Weight	7	kg	*NULL*	weight	Roof hood	en
77	1	ULV2I Roof hood	Material	Zinc-magnesiu...	*NULL*	*NULL*	material	Roof hood	en
77	1	ULV2I Roof hood	Duct connection	Squared rainfitt...	*NULL*	KL-400x400	ductConnection	Roof hood	en
8	1	ULV2K Roof hood	Air distribution	Air exhaust	*NULL*	*NULL*	airDistribution	Roof hood	en
8	1	ULV2K Roof hood	Model	ULV2K	*NULL*	*NULL*	model	Roof hood	en
8	1	ULV2K Roof hood	Length	630	mm	*NULL*	length	Roof hood	en
8	1	ULV2K Roof hood	Width	630	mm	*NULL*	width	Roof hood	en
8	1	ULV2K Roof hood	Weight	57	kg	*NULL*	weight	Roof hood	en
80	1	ULV2I Roof hood	Air distribution	Air exhaust	*NULL*	*NULL*	airDistribution	Roof hood	en
80	1	ULV2I Roof hood	Model	ULV2I	*NULL*	*NULL*	model	Roof hood	en
80	1	ULV2I Roof hood	Length	200	mm	2	length	Roof hood	en
80	1	ULV2I Roof hood	Width	200	mm	2	width	Roof hood	en
80	1	ULV2I Roof hood	Weight	7	kg	*NULL*	weight	Roof hood	en
80	1	ULV2I Roof hood	Material	Zinc-magnesiu...	*NULL*	*NULL*	material	Roof hood	en
80	1	ULV2I Roof hood	Duct connection	Squared rainfitt...	*NULL*	KL-500x500	ductConnection	Roof hood	en
9	1	ULV2K Roof hood	Air distribution	Air exhaust	*NULL*	*NULL*	airDistribution	Roof hood	en
9	1	ULV2K Roof hood	Model	ULV2K	*NULL*	*NULL*	model	Roof hood	en
9	1	ULV2K Roof hood	Length	800	mm	*NULL*	length	Roof hood	en
9	1	ULV2K Roof hood	Width	800	mm	*NULL*	width	Roof hood	en
9	1	ULV2K Roof hood	Weight	92	kg	*NULL*	weight	Roof hood	en
75	2	MKL	Length	1750	mm	*NULL*	length	Roof transition	en
75	2	MKL	Width	1750	mm	*NULL*	width	Roof transition	en
75	2	MKL	Material	Zinc-magnesiu...	*NULL*	*NULL*	material	Roof transition	en
75	2	MKL	Insulation	50mm mineral ...	*NULL*	50	insulation	Roof transition	en
78	2	MKLI-50	Height	1200	mm	*NULL*	height	Roof transition	en
78	2	MKLI-50	Material	Zinc-magnesiu...	*NULL*	*NULL*	material	Roof transition	en

Lisa 5 – Andmebaasi disain kasutades EF Core'i

```
protected override void OnModelCreating(ModelBuilder b)
{
    base.OnModelCreating(b);
    SheetData.SheetConfiguration(b);

    b.Entity<Product>(entity =>
    {
        entity.HasOne( navigationExpression: p :Product => p.ProductType);
        entity.ToTable("Products");
    });
    b.Entity<ProductRelation>(entity =>
    {
        entity.HasKey(x :ProductRelation => new {x.ChildId, x.ParentId});
        entity // EntityTypeBuilder<ProductRelation>
            .HasOne( navigationExpression: x :ProductRelation => x.Parent) // ReferenceNavigationBuild
            .WithMany( navigationExpression: x :Product => x.ChildProducts)
            .HasForeignKey(x :ProductRelation => x.ParentId)
            .OnDelete(DeleteBehavior.Restrict);
        entity // EntityTypeBuilder<ProductRelation>
            .HasOne( navigationExpression: x :ProductRelation => x.Child) // ReferenceNavigationBuilder
            .WithMany( navigationExpression: x :Product => x.ParentProducts)
            .HasForeignKey(x :ProductRelation => x.ChildId)
            .OnDelete(DeleteBehavior.Restrict);
        entity.ToTable("ProductRelations");
    });
    b.Entity<AdditionalFeatureValue>(entity =>
    {
        entity.HasOne( navigationExpression: p :AdditionalFeatureValue => p.Product) // ReferenceNavi
            .WithMany( navigationExpression: x :Product => x.AdditionalFeatures)
            .HasForeignKey(x :AdditionalFeatureValue => x.ProductId);
        entity.HasOne( navigationExpression: p :AdditionalFeatureValue => p.ProductFeature);
        entity.ToTable("AdditionalFeatureValues");
    });
    b.Entity<MainFeatureValue>(entity =>
    {
        entity.HasOne( navigationExpression: p :MainFeatureValue => p.Product) // ReferenceNavigationBuil
            .WithMany( navigationExpression: x :Product => x.Features)
            .HasForeignKey(x :MainFeatureValue => x.ProductId);
        entity.HasOne( navigationExpression: p :MainFeatureValue => p.ProductFeature);
        entity.ToTable("MainFeatureValues");
    });
    b.Entity<ProductTranslation>(entity =>
    {
        entity.HasOne( navigationExpression: x :ProductTranslation => x.Culture);
        entity.HasOne( navigationExpression: p :ProductTranslation => p.Product) // ReferenceNavigationBu
            .WithMany( navigationExpression: x :Product => x.Translations)
            .HasForeignKey(x :ProductTranslation => x.ProductId);
        entity.ToTable("ProductTranslations");
    });
```

```
b.Entity<ProductFeatureTranslation>(entity =>
{
    entity.HasOne(navigationExpression: x :ProductFeatureTranslation => x.Culture);
    entity.HasOne(navigationExpression: p :ProductFeatureTranslation => p.ProductFeature) // ReferenceNavigatic
        .WithMany(navigationExpression: x :ProductFeature => x.Translations)
        .HasForeignKey(x :ProductFeatureTranslation => x.ProductFeatureId);
    entity.ToTable("ProductFeatureTranslations");
});
b.Entity<ProductTypeTranslation>(entity =>
{
    entity.HasOne(navigationExpression: x :ProductTypeTranslation => x.Culture);
    entity.HasOne(navigationExpression: p :ProductTypeTranslation => p.ProductType) // ReferenceNavigationBuilde
        .WithMany(navigationExpression: x :ProductType => x.Translations)
        .HasForeignKey(x :ProductTypeTranslation => x.ProductTypeId);
    entity.ToTable("ProductTypeTranslations");
});
b.Entity<MainFeatureValueTranslation>(entity =>
{
    entity.HasOne(navigationExpression: x :MainFeatureValueTranslation => x.Culture);
    entity.HasOne(navigationExpression: p :MainFeatureValueTranslation => p.MainFeatureValue) // ReferenceNavi
        .WithMany(navigationExpression: x :MainFeatureValue => x.Translations)
        .HasForeignKey(x :MainFeatureValueTranslation => x.FeatureValueId);
    entity.ToTable("MainFeatureValueTranslations");
});
b.Entity<AdditionalFeatureValueTranslation>(entity =>
{
    entity.HasOne(navigationExpression: x :AdditionalFeatureValueTranslation => x.Culture);
    entity.HasOne(navigationExpression: p :AdditionalFeatureValueTranslation => p.AdditionalFeatureValue) /
        .WithMany(navigationExpression: x :AdditionalFeatureValue => x.Translations)
        .HasForeignKey(x :AdditionalFeatureValueTranslation => x.FeatureValueId);
    entity.ToTable("AdditionalFeatureValueTranslations");
});
b.Entity<ProductFeature>().ToTable("ProductFeatures");
b.Entity<ProductType>().ToTable("ProductTypes");
b.Entity<Culture>().ToTable("Cultures");
b.Entity<RoofHoodConstant>().ToTable("RoofHoodConstant") // EntityTypeBuilder<RoofHoodConstant>
    .HasKey(key :RoofHoodConstant => new { key.ProductId });

//Views
b.Entity<ProductViewDataEn>().ToView("ProductsEnView").HasNoKey();
b.Entity<ProductViewDataEt>().ToView("ProductsEtView").HasNoKey();
b.Entity<ProductViewDataWithAdditionalFeatures>()
    .ToView("ProductsWithAdditionalFeaturesView").HasNoKey();
b.Entity<ChildProductsViewData>().ToView("ChildProductsView").HasNoKey();
```

Lisa 6 – Meetod *AdditionalFeature dictionary* loomiseks

```
public async Task<ActionResult<Dictionary<string, List<AdditionalFeatureViewModel>>>> CreateAdditionalFeaturesByProduct(
    string productTypeId, string productId, string language)
    {
        var productWithAdditionalFeatures :List<ProductViewDataWithAdditionalFeatures> = await _context.ProductsWithAdditionalFeatures // Db
            .Where(x :ProductViewDataWithAdditionalFeatures => x.ProductId == productId)
            .Where(x :ProductViewDataWithAdditionalFeatures => x.ProductTypeId == productTypeId) // IQueryable<ProductViewDataWithAdditionalFeatures>
            .Where(x :ProductViewDataWithAdditionalFeatures => x.Language == language).ToListAsync(); // Task<List<..>>
        if (productWithAdditionalFeatures == null || !productWithAdditionalFeatures.Any()) return null;
        var featuresList = new Dictionary<string, List<AdditionalFeatureViewModel>>>();
        foreach (var r :ProductViewDataWithAdditionalFeatures in productWithAdditionalFeatures)
            if (productId == r.ProductId)
            {
                var feature = new AdditionalFeatureViewModel
                {
                    Name = r.Feature,
                    Value = r.Value,
                    IsMandatoryValue = r.IsMandatoryValue,
                    Code = r.Code,
                    ProductFeatureId = r.FeatureId,
                    Unit = r.Unit
                };
                if (featuresList.ContainsKey(feature.ProductFeatureId))
                {
                    featuresList[r.FeatureId].Add(feature);
                }
                else
                {
                    var additionalFeaturesOnProduct = new List<AdditionalFeatureViewModel> {feature};
                    featuresList.Add(r.FeatureId, additionalFeaturesOnProduct);
                }
            }
        return featuresList;
    }
```

Lisa 7 – Risto-Raul Pajula eneseanalüüs

Saime enne projektiga alustamist kokku ettevõtte esindajaga ETS NORD-i peakontoris. Meile esitati nõuded, mida meilt oodatakse järgmise 16 nädala jooksul ning rääkisime ka pisut pikemast plaanist. Esimesel koolinädalal kaardistasime koos teiste liikmetega enda jaoks ära, mida ettevõtte esindaja meilt nõuab. Õppisin selle käigus, kui oluline on saada võimalikult detailne ülevaade, sellest mida meilt nõutakse. Esimesed kaks nädalat läks puhtalt mõtlemisele ja koodi prügikasti kirjutamisele. Ma järeldasin sellest, et see on tavaline programmeerimise osa.

Kolmandal nädalal kasutasime eelnevalt õpituid tehnikaid, kuidas algeline API valmis teha ning alustati praktiseerimisega Reactis. Õppisin, kuidas käib andmete pärimine serverilt, mida tähendab render, kuidas teha kasutaja jaoks vorme (Forms). See osa oli siiani kõige põnevam, sest kasutajaliidese tegemisel on tulemust kohe oma silmaga näha. Neljandal nädalal saime praktiseerida C# keeles andmeahelate (list) arhitektuuri ja loogikat, millega võimaldas rakendus nüüd mitu mudelit korraga valida.

Viiendal nädalal tuli esimest korda demonstreerida ettevõtte esindajale progressi. Tema jaoks oli kõige tähtsam, mis ta oma silmaga nägi. Saime küll kiita, aga teadsin, et andmemudelisse tuleb teha suured muudatused. Lõime andmebaasi lahku kolmeks erinevaks tabeliks ning proovisin implementeerida Arlowi ja Neustadti mudelit. Siit maalt liiguti edasi kahe erineva branchiga, teised harjutasid kätt TypeScripti kirjutamisel, aga ma alustasin nullist uue andmemudeliga. Tutvusin, kuidas kirjutada SQL View'sid, kus panin kokku ühe tabeli 3st teisest andmetabelist. Andmete küsimiseks tuli luua uus otsingufunktsiooni loogika controlleris, mille tegemisele kulus mul mitu päeva.

Kaheksas-üheksas nädal kulus peamiselt kasutajaliidese korrigeerimisele ja muutmisele. Õppisin, miks ülerenderdamine on halb ning kuidas seda vältida. Tutvusin uue metoodikaga - klassi komponentide asemel saab luua funktsionaalseid komponente, mille kirjutamine sobis mulle rohkem. Üleliigset koodi on vähem ja üks meetod võib teha kahe meetodi töö ära. Reactiga töötades tutvusin mitmete tehnoloogiatega: kuidas luua PDF dokumente ning kuidas küsida väliseid andmeid Axiosega.

16. nädala pühendasin sellele, et rakendus läheks serverisse üles, et ka ettevõtte esindaja saaks rakendusele ligi. Me proovisime meeskonnaga koos 13. nädalal üles seada CI/CD'd (pidev integratsioon/pidev juurutamine) koos Dockeriga, esimese poolega saime hakkama, aga rakendust juurutatud ei saanud. Seega viimasel nädalal ma võtsin kasutusele hoopis teise suuna. Youtubest leidsin, et Reacti saab lihtsalt üles panna Netlify abiga. Selle sain ka tehtud - meil oli nüüd rakendus üleval, aga ilma API-ta. API panin üles läbi Azure App Service'i ja andmebaasi Azure SQL-i. Mõned muudatused veel Reactis ja ligipääsu lubamine Azure'is ning rakendus oligi üleval täielikult. Selle nädala jooksul omandasin rohkelt teadmisi DevOps-ist.

Meeskonnaprojekt oli selleks hetkeks läbi ning edasi liiguti lõputöö raames. Suur hulk koodi vajas refaktoorimist ja vigade parandamist. Lisandus suur hulk nõudeid ning asusin neid kaardistama ja välja arendama. Kõige aeganõudvam töö oli edaspidi rakenduse viimine mitmesse kultuuriruumi, millele kulus järgmised kolm nädalat. Tutvusin selle käigus uute tehnoloogiatega nagu "i18n". Mitut kultuuriruumi aktsepteeriva andmemudeli loomine nõudis palju analüütimist mõtlemist.

Järgnevalt esines palju tööd React teegiga. Tuli tutvuda põhjalikult Redux arhitektuuriga ning sellest hetkest hakkas React mulle meeldima. Tundsin, et front end arendaja on positsioon, kus tulevikus töötada. Realiseeritud funktsionaalsuse märksõnad: laadimine, navigeerimine ühelt lehelt teisele, märkeruudud ning Redux arhitektuur.

Toodete filtreerimine oli teostatud ning nüüd tuli realiseerida toote konfigureerimine, mis osutus väljakutseks. Tutvusin uute andmestruktuuride ning nende implementeerimisega. Andmemudeli keerukus kasvas veelgi ning realiseerida oli vaja uued klassid. Õppisin väga põhjalikult kasutama Entity Framework Core'i ning LINQ keelt, et luua mitmeid keerulisi seoseid klasside vahel. Andmebaasi loomise kirjeid oli tuhandeid ning selleks tutvusin Sheets API-ga ning õppisin integratsioone looma kahe rakenduse vahel. Konfiguraatori loomisele kulus umbes 3 nädalat ning selle lõpus olid oskused .NET raamistikus head.

Järgnevalt tuli luua integratsioon meie rakenduse ning ettevõtte poolse veebipoe vahel. Nüüdsest teadsin, et kaks rakendust peavad aktsepteerima samasugust andmestruktuuri, et omavahel informatsiooni vahetada. Edaspidi järgnes kasutajaliideses palju tööd, et konfigureerimise lõppedes, oleksid andmed õigel kujul veebipoe jaoks. Selleks lõin niiöelda tootekorvi loogika, et kõik lisatud elemendid jõuaksid ühte kohta, mida veebipoodi edastada.

Kokkuvõttes oli projekt väga arendav ning praktilisi kogemusi sai rohkelt. Meil puudus ettevõttepoolne juhendaja, mille tõttu oli väga palju oli guugeldamist ja tühjuses kompimist. Selle miinuseks oli küll see, et väga palju tunde kulutasime otsimisele, kui koodi kirjutamisele, kuid selle käigus jäävad asjad paremini meelde. Projekti käigus tundsin, et tööde jaotamine ei olnud kõige paremini reguleeritud ning selle tõttu ei jätkunud kõigile võrdselt tööülesandeid. Soovin, et meeskonnas oleks olnud veel projektijuht, kes ülesandeid võrdselt liikmete vahel laiali jagaks. Kõige olulisem osa on see, et klient ja meeskond saaksid asjadest samamoodi aru - kaheti mõistmisel esineb palju lisatööd ning pahameelt.

Lisa 8 – Hendrik Laretei eneseanalüüs

Septembrikuus alustasime projekti tutvumise ja planeerimisega. Aitasin välja mõelda kasutajalugusid ning arutasime, kuidas peaks välja nägema projekti arhitektuur. Seadsime alguses eesmärgiks luua lihtne toodete filtreerimissüsteem, mis toimiks kolme parameetriga ning oleks seotud andmebaasiga nii nagu ettevõtte esindaja soovis. Otsustasime, et teeme alguses lihtsa arhitektuuri, et asjad jooksma saada, nii soovitas ka meie juhendaja. Keskendusime lihtsa loogikaga serveripoolse rakenduse välja mõtlemisele, mis oleks seotud kliendipoolse rakendusega ning suudaks päringuid tagastada andmete näol. Oktoobrikuus hakkasime paralleelselt arendame teist haru, kus oli keerukam arhitektuur mitme andmetabeliga. Kasutasime Arlow'i ja Neustadt'i toote arhetüübi mustrit nii nagu meile juhendaja poolt soovitati. Esines keerukusi arhetüüpi kohandamisega selliseks nagu projektile vaja oli, sest klassid ei olnud siiski identsed ning mustri integreerimine nõudis toote arhetüübi väga head mõistmist. Võtsime kasutusse View'd ning edastasime andmed vaadetena, et kliendipoolne rakendus ühilduks uue serveripoolse rakenduse arhitektuuriga. Peamised minu panused projekti septembri- ja oktoobirkuus olid serveripoolse rakenduse ülesse ehitamine ning arhitektuuri väljamõtlemine. Serveripoolse rakenduse repositooriumisse lõin toodete filtreerimise loogika. Mõtlesin välja viisi, kuidas kätte saada konstandid iga katuseotsiku erineva suuruse juures, et arvutada vastavalt kindla mudeli õhuvool ja rõhulangus. Kogu arvutuse loogika nõudis palju aega, sest väljastpoolt abi ei saanud. Lisaks olin abiks serveripoolse ja kasutajaliidese rakenduse ühendamisel. Filtreerimise loogikale lisasin juurde funktsionaalsuse, et ühele otsitavale parameetrile saaks anda mitu valikut. Kliendipoolse rakenduse arendamise käigus tegin paarisprogammeerimist Risto-Rauliga. Tegime ära filtreerimise osa kliendipoolses rakenduses ning lihvisime loogikat serveripoolse- ja kliendipoolse rakenduse vahel. Novembrikuus alustasime toote detailvaatega ning suurim murekoht oli React, kus esines raskusi andmete edasi saatmisel. Alustasin toote tellimuse PDF-dokumendi loomisega, kuhu salvestuvad kõik ühe toote andmed, konfigureeritud omadused, tootekood ja projekti informatsioon. Novembrikuu teises pooles arendasin koos Oskariga Reactis tootekoodi loogikat ning katuseotsikute edasist konfigureerimist detailvaates. Detsembrikuus täiustasime arhitektuuri ja refaktoorisime koodi. Panustasin palju aega projekti serverisse saamisega, ning lisasin andmebaasi kõik tooted koos nende vajalike seostega. Sellega lõppes meeskonnaprojekti aine, kuid ettevõtte soovis, et projekti tiim arendaks rakendust edasi. Tekkisid uued nõuded, mis nõudsid veel spetsiifilisemaid teadmisi ventilatsiooni toimimise kohta. Ettevõte soovis katuseotsikute lisakonfigureerimise

funktsionaalust, mis võimaldaks valitud tootele külge lisada ventilatsiooni toimimiseks vajaminevaid kõrvaltooteid. Lisaks sellele oli vaja ühendada rakendus ettevõttepoolse arenduskäigus oleva veebipoega. Suur osa ajast kulus nõuete kaardistamisele ja ettevõttega suhtlemisele, sest ettevõte ei väljendanud täpselt, milliseid abitooted ja nende eriversioonid sobivad kindlale katuseotsiku mudelile. Lähtuvalt uutest nõutest pidime laiendama ja ümber tegema rakenduse arhitektuuri. Panustasin aega rakenduse avalehe ja detailvaate kujundamisele, kuhu lisandusid juurde uued väljad ja rohkem informatsiooni toodete parameetrite kohta. Tegin Reactis juurde funktsionaalsuse, et kliendipoolne rakendus kasutaks SessionStorage't, mis tähendab, et kasutaja valitud väljad ja konfigureeritud toode ei kao ära navigeerimisel ja veebilehe värskendamisel. Peale selle kujunes ajamahukaks serveripoolse rakenduse testimine, mis hõlmas endas testkeskkonna seadistamist ja integratsiooni- ning ühiktestide kirjutamisest.

Suurimad väljakutsed olid seotud ettevõttepoolse juhendaja puudumisega, kes oleks tarkvaraarendamise taustaga. Tugiisiku puudumise tõttu kulus rohkelt aega, et leida vastused arendamise käigus tekkinud spetsiifilistele küsimustele. Teiseks väljakutseks kujunes infosüsteemi arhitektuuri arendamine ilma suurema juhendaja poolse abita. Leian, et kogu protsess oli küll ajakulukam, kuid kokkuvõttes arendavam meeskonnale. Katuseosikute tootekonfiguraatorit arendamisel pidi meeskond omama palju spetsiifilisi teadmisi ventilatsiooni valdkonnas, kuid antud teadmised meeskonnal tegelikult puudusid, mistõttu kulus rohkelt aega ventilatsiooni toodete uurimisele ja mõistmisele. Palju vaeva ja aega kulus katuseotsikute õhuvoolu ja rõhukao valemi tuletamisele, sest ettevõtte ei osanud valemi leidmisel aidata. Meeskonnasiseselt oli aeg-ajalt keeruline teha koostööd, sest töö rollide jagunemine ei olnud alati selge, mis tekitas omakorda väiksemaid lahkhelisid. Kogu protsessi jooksul arenes meeskond just seetõttu, et tiim pidi olema võimeline ise kõikidele probleemidele lahendusi leidma. Tootekonfiguraatori süsteem tuli nullist üles ehitada ning veebirakendusse üles seada. Oluliseks pidepunktideks kujunesid koosolekud, kus üheskoos tiimiga analüüsiti tehtud tööd, toodi välja, mida ja kuidas oleks saanud teha efektiivsemalt ning seati uued eesmärgid. Kuigi minu hinded olid eelmistel semestritel väga head või head, mis võiks viidata sellele, et mul peaks olema piisav ettevalmistus ja teadmised selle projekti jaoks, osutus projekt tegelikkuses palju keerukamaks. Kogu projekti vältel pidin väga palju aega panustama arenduskeelte ja raamistike õppimisele. Sain kogemuse arendamisega seotud kannatlikkusest, mis tähendab, et 80% ajast kulub lahenduse otsimisele, mis on tingitud tehniliste teadmiste puudusest ning ainult 20% ajast sain arendada projekti. Sellest järeldan, et arendamine kujutab endast tihtipeale

olukordi, kus suurem osa ajast kulub kõrvalistest allikatest lahenduse otsimisele. Sain selle projektiga palju kogemust väga paljudes valdkondades: back-end ja front-end arendus, süsteemi arhitektuur, rakenduse veebisaidile ülesse panemine, lisaks sellele ettevõttega suhtlemine ja kliendi nõuete kaardistamine. Selline laiapõhjaline kogemus ja päriselt kasutusele minev rakendus andis mulle tugeva baasi, et saaksin edaspidi tarkvaraarenduse valdkonnas töötamist jätkata.

Lisa 9 – Mark Rõõmussaar eneseanalüüs

Antud projekti alustasime ettevõtte poolt ette antud nõuete analüüsimisega ja nende kaardistamisega. Arutasime läbi oma juhendajaga ning saime juhiseid, kust ja kuidas alustada. Alustada oli suhteliselt keeruline, kuna ettevõtte poolt oli ette antud ainult näidispildid, milline võiks välja näha projekti lõplik kasutajaliides, ning seletavad laused iga funktsiooni kohta. Samuti puudus ettevõtte poole IT tugi, mis tegi meie jaoks olukorra veelgi raskemaks, kuna kõigi tööriistade ning projekti arhitektuuri paika panemine oli meie välja mõelda.

Esimesel kuul tegelesime projekti nõuete uurimisega ning esialgse arhitektuuri paika panekuga. Meeskond alustas ka lihtsa API loomisega, kus liiguksid esialgu testandmed ning alustati Reactis minimaalse kasutajaliidese loomisega. Seal saadi valmis kolme parameetriga otsingumootor.

Järgmisel kuu algas ettevõttele demo esitlusega, millega olid nad väga rahul. Alustasime Oskariga front-end'is modaalakna loomist projekti informatsiooni jaoks. Samuti arendasime eraldi detaili vaate toodetele. Kuu teises pooles saime kokku juhendajaga, et leida uus arhitektuuriline lahendus. Kuu lõpu poole tegime algust front-endi disainimisega.

November algas taaskord ettevõttele demo esitlusega, millega oldi väga rahul. Kuu alguses lisasime toodete detaili vaatesse PDF-dokumendi vormi ning selle allalaadimise võimaluse. Sel kuul oli suureks murekohaks toodete õhuvoolu graafikute lugemine ning automatiseerimine. Kuu lõpu poole tegime muudatusi kasutajaliidese disainis, võttes aluseks ettevõtte poolt antud brand book.

Järgneval kuul lisasime detaili vaatesse toote konfigureerimiseks soovitud väljad, mis kajastuvad ka PDF-is. Lõime ka tootekoodi lahtri, kus kajastab toote omadusi. Proovisime CI/CD üles seada, et klient saaks jooksvalt projekti kasutajaliidest uurida. Seega tegime Oskariga tutvust Dockeriga, millega olime eelmine aasta juba natuke kokku puutunud. Kuu lõpus tuli üllatuseks, et ettevõte on oma kodulehe disaini ning brand book'i mõne võrra muutnud. Sealt tulenevalt saime esmakordselt uurida Adobe XD tarkvara ning seal graafilise disaineri poolt saadetud köögikubude prototüüpi.

Jaanuaris saime suurem tagasiside ettevõttelt, kus oli täpsemalt välja toodud, mis muudatusi oleks vaja teha. Toimus ettevõttega koosolek, kus vaadati üle muudatust vajavad kohad ning

pakkusime välja omalt poolt leitud lahendusi. Sel kuul tegelesimegi enam jaolt soovitud muudatuste sisse viimisega nii disaini kui ka loogika koha pealt. Samuti proovisime ettevõttele erinevaid lahendusi leida toodete piltide ning graafikute suurendamisele.

Veebruar alguses proovisime leida sobivat lahendust mudelite valiku juurde. Selle saime paika pandud jooksvat ettevõttega arutades ning demonstreerides. Tegelesime ka rakenduse keelte valikuga, saades ettevõttelt vastavad tõlked kõigile sõnalistele väärtustele. Tegelesime ka loogikaga, et API-lt saaks pärida ühte toodet, mida detaili vaates kuvada.

Järgneval kuul toimus demo esitlus ettevõttele. Samuti alustasime projekti testimisega. Esialgu tegime algelised ühiktestid ning uurisime juhendajaga ka integratsiooniteste ning selle võimalikke lahendusi. Kuu teises pooles alustasin vastuvõtu testidega Postman-i keskkonnas ning proovisin seal kahte erinevat võimalust. Alustasin varasemalt õpitule tuginedes ning tegin tutvust ka läbi mock-serveri testimist. Tutvusime Azure App Serviceiga ning proovisime oma rakendust sinna üles panna.

Aprillis toimus juhendajaga koosolek, kus arutati nii projekti kui ka lõputöö osas. Jätkasin Postman-i keskkonnas testimist. Täiendasin samuti ühikteste, kuid kuna me muutsime projekti arhitektuuri suuresti ümber siis need kasutust lõpuks ei leidnud. Kuu lõpus uuendasin vastuvõtuteste vastavalt arhitektuuri muudatustele.

Kokkuvõtvalt oli sellisel kujul projekti tegemine väga kasulik ning õpetlik. Suurimaks väljakutseks oli kindlasti meeskonnaga iseseisev arendamine ning sobilike lahenduste leidmine. Kuna meeskonnal ettevõtte poolne IT tugi kui ka tiimi juhendav isik puudus, oli väga keeruline efektiivselt tööd teha. Tihtipeale tuli teha kannapööre erinevate lahenduste osas, mis tähendas tühja tehtud tööd ning ajakulu. Samuti oleksime võinud meeskonnana paremini toime tulla ülesannete jagamisel ning arendamisel hoida kindlat tööstruktuuri. Kogemuse kohapealt tõi projekti arendamine meeskonnaga väga hästi välja, kui oluline on kliendi soovitu õigesti mõistmine. Vastasel juhul võib see arendajatele kaasa tuua suure lisatöö. Ettevalmistuse osas tundsin ennast üsnagi kindlalt projekti alguses, kuid reaalsus oli teine. Väga palju tööd tuli ise tööd tööriistade uurimisel ning lahenduste leidmiseks koodi kirjutamisel. Lõpetuseks jõudsime projektiga korraliku tulemuseni, millega klient samuti väga rahul oli.

Lisa 10 – Oskar Neemre eneseanalüüs

Projekti esimene kuu algas User Story'de kirjutamisega ja arhitektuuri paika panemisega. Valmis esimene testrakendus, milles olime ettevõttega kokku leppinud. Teisel kuul valmis modaalaken kuhu klient saaks sisestada infot projekti kohta, koos nuppudega mis sisendit salvestavad või kustutavad. Samuti aitasin lisada Details vaate kus kuvatakse kõik tooteomadused. Kolmandal kuul lisasime Details vaatesse tootepildid ja graafikud ning nupu millega saab alla laadida konfigureeritud toote PDF-dokumendi. Kasutajaliidese disainielemendid nagu fondid ja värvid viidi vastavusse ETS NORD brandbook'iga. Neljandal kuul lisandusid detailvaatesse esimesed konfigureeritavad väljad mis seoti ka PDF-dokumendiga ning alustasime CI/CD ülesseadmisega. Tuginedes varasemalt õpitule proovisime teha seda Docker'i abil, kuid kuna meil ei olnud serveripoolse seadistusega kogemusi, hakkas see kulutama ebaproportsionaalselt palju aega. Neljanda kuu lõpus edastati meeskonnale ka Adobe XD prototüübifailid mille järgi sai parandada CSS reeglid kasutajaliidese elementidele. Viies kuu jätkus disaini refaktoorimisega, mille käigus anti CSS reeglitele oluliselt paremad nimed ning kustutati ebavajalik ProjectInfo, Search, ProductTable ja ProjectDyn komponentides. Samuti lisasin ETS NORD footeri iga lehekülje allosasse ning alustasin kasutajasisendi validatsiooni testimisega. Kuuendal kuul sai ProjectInfo validatsioon esialgselt valmis ning selle jaoks ei kasutatud välist teeki, sest valideerimiseks ettenähtud moodulid tekitasid rakenduses sügavaid type-error'eid. Details vaatesse lisandus tabs komponent, tehti algust testimisega. Seitsmendal kuul jätkati ühiktestide kirjutamisega ning koos Viljam Puusepa abiga lisandus repository meetodite testimiseks FakeData, mille järel alustati andmete üleviimist Google Sheets APIsse. Kaheksandal kuul jätkasin testimisega kui ka ebavajaliku kustutamisega kasutajaliidesest, peamine rõhk läks kaheksandal kuul lõputöö kirjutamisse.

Suurimaks väljakutseks projekti teostamisel oli kindlasti React Typescript, tundsin end projekti käigus kõige rohkem arenevat just sellega tegeledes. Üsna keerukas oli minu jaoks ka CI/CD ülesseadmine, sest serveripoolne seadistamine jäi jätkuvalt ebaselgeks ning ilmselt tuleb sellega veel tulevikus tööd teha. Kõige lihtsam oli minu jaoks tegeleda disainiga, millega olen põgusalt varem kokku puutunud. Rohkem oleks võinud keskenduda meeskonnatöö juhtimisele ning kõigi paralleelsele arengule. Pidime suure osa ajast olema iseenda peremehed ning sellest õppisin, miks on kasulik kui projektil on juht. Töö efektiivsus on tugevalt seotud sellega kuidas

meeskonnaliikmed omavahel suhtlevad, tiim saab olla vaid nii tugev kui on tema nõrgem lüli. Samuti ilmnes kui oluline on kindlate tööprotsesside paikapanemine ja nende täpne järgimine. Mõistsin ka kui oluline on kliendi soove täpselt kaardistada, sest klient ei oska kunagi mõelda rakendusest nii nagu analüütik või rakenduse arendaja. Ärinõuete mõistmine on omamoodi kunst, kus tuleb osata väga palju õigeid küsimusi küsida. Projekti käigus ei olnud ühtegi asja millega edaspidi tegeleda ei tahaks, kõik tegevused toetasid üksteist ning olid minu jaoks võrdse tähtsusega.

Sellise projekti jaoks on väga keeruline end ette valmistada, ülikool on suutnud teha selleks üsna hea töö ning peamiseks puuduseks võibki lugeda CI/CD kaasneva keerukuse. Ettevalmistus individuaalsel tasemel sõltub kindlasti ka tugevalt sellest, kui palju rõhku on tudeng oma õppetööle pannud. Sellise projekti teostamisel soovitaksin jääda kindlaks oma oskustele ja teadmistele, olles valmis probleemideks mis esialgu ahastust tekitavad. Tegelikult peabki väga palju ise uurima, lugema ja läbi töötama, et toimuks progress.