TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Omar Mahlapuu

Projektide planeerimise veebirakenduse arendamine tootmisettevõtte näitel

Bakalaureusetöö

Juhendaja: Kristiina Hakk
PhD

Tallinn 2022

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: Omar Mahlapuu

06.01.2022

Tallinn 2022

Annotatsioon

Innovaatiline ja moodsaid klaasvaheseinu ning uksi pakkuvas ettevõttes toimub projektide planeerimine Excelis, mida on kasvaval ettevõttel keeruline hallata, sest sellesse tehakse tahtmatult vigu või kaovad andmed ja väga raske on projekte analüüsida.

Töö eesmärk on luua ettevõttele kaasaegne veebirakendus, mis lihtsustab projektide planeerimist ja võimaldab lihtsamalt projektidest kokkuvõtteid teha.

Bakalaureusetöö esimeses peatükis on esitatud töö sissejuhatus.

Teises peatükis tutvustatakse ettevõtet, kellele veebirakendus luuakse, vaadeldakse probleemi ja seatakse eesmärk probleemi lahendamiseks.

Kolmandas peatükis kirjeldatakse rakenduse nõudeid, võrreldakse erinevaid sobilikke programmeerimiskeeli ja raamistikke ning analüüsitakse andmebaasi valikut ja valitakse välja sobilik lahendus.

Neljas peatükk käsitleb andmebaasimudeli loomist ning primaarvõtme valikut. Kirjeldatakse tagarakenduse struktuuri ja eesrakenduse arendust.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 45 leheküljel, 7 peatükki, 20 joonist ja 2 tabelit.

Abstract

Development of Project Planner Web Application for Production Company

In an innovative and modern production company has done its project planning in Microsoft Excel for years, which can be hard to manage in a rapidly growing company. It is easy to make unwanted and uncontrollable mistakes using Microsoft Excel for a primary software. Unwanted mistakes and loss of data make it hard to do overviews and analyzations.

The purpose of this thesis is to develop a modern web application for a production company. This application is going to make project planning faster and easier. The appliaction consist of a database for data storage, back-end application for database operations and calculations and a easy to use, straight-forward front-end applcation. Using this solution will improve this company's time management and add value to the company itself.

In the first chapter is the intoduction to the thesis.

In the second chapter the author gives an introduction to the the production company and gives an overview of the problem and purpose of this thesis.

In the third chapter the author describes the requirements for the application given by the owner of the company, project – and sales managers. And overview of choosing technologies and frameworks for the application is also given in this chapter.

In the fourth chapter the author describes the development process step-by-step, showing diagrams and code examples.

The thesis is in Estonian and contains 45 pages, 7 chapters, 21 figures and 2 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
BLL	*Bussiness Logic Layer*, äriloogika kiht
CNC	*Computer Number Control*, arvutiprogrammjuhtimine
CSS	*Cascading Style Sheets,* kaskaadlaadistik
DAL	*Data Access Layer*, andmetele juurdepääsu kiht
DTO	*Data Transfer Object,* andmeedastus objekt
HTML	*HyperText Markup Language*, hüperteksti märgistuskeel
IDE	*Integrated Development Environment*, integreeritud arenduskeskkond
JSON	*JavaScript Object Notatation*, JavaScriptil põhinev andmevahetusvorming
MDF	*Medium density fireboard*, keskmise tihedusega puitkiudplaat
MVC	*Model-View-Controller*, mudel-vaade-kontroller-rakendustes kasutatav arhitektuur
REST	*Representational State Transfer*, tarkvara arhitektuuri laad, mis seab veebirakenduse loomisele kindlad piirid
SQL	*Structured Query Language*, struktureeritud päringukeel
UI	*User Interface*, kasutajaliides
UOW	*Unit Of Work*, tööühik
URL	*Uniform Resource Locator*, üldine infoallika asukohamääraja ehk internetiaadress
XML	*Extensible Markup Language*, laiendatav märgistuskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesoleva lõputöö raames luuakse veebirakendus projektide planeerimiseks ettevõttes Structo Group OÜ. Rakenduse eesmärk on kiirendada ja lihtsustada ettevõtte projektide planeerimist. Rakendus peab olema töökindel ja seda peab olema mugav kasutada nii, et see ei tekitaks kasutajale segadust ning sellega saab teha kiiresti ja lihtsalt aasta kokkuvõtteid lõpetatud projektidest.

Probleem, mida töö autor lahendab, seisneb Structo Group OÜ-s kasutusel olnud vana süsteemi asendamises. Ettevõttes oli kasutusel lihtne Microsoft Exceli (edaspidi *Excel*) põhine tabel, mis oli loodud aastaid tagasi ning mis oli tänaseks aegunud ning ka liialt algeline viis projektide planeerimiseks. Excelipõhise süsteemi peamine nõrk koht on see, et sinna on lihtne vigu teha, sisestades info näiteks valesse lahtrisse, mille tõttu võib hiljem hulk andmeid ja vajalikku infot lihtsalt kaotsi minna. Samuti nõuab Excelis kokkuvõtete tegemine rakenduse tundmist ning selle puudlikkus tekitab suurt ajakulu ning käsitööst põhjustatud vigu.

Ettevõte otsustas, et otstarbekas oleks kasutusele võtta spetsiifiline rakendus, mis on loodud spetsiaalselt ettevõttes loodavate projektide planeerimiseks ning mis arvestaks ettevõtte eripäradega. Ettevõte on proovinud eelnevalt valmisrakendusi, kuid nende kasutamine on ettevõtte spetsiifikat arvestades olnud liiga keeruline, üks nendest on Scoro, mille miinuseks on see, et toode on esiteks tasuline ning projektijuhtide sõnul on konfigureerimine lihtsa ülesande puhul ka üleliia keeruline ning kõikidele vajadustele mittevastav.

Lahenduseks sooviti spetsiaalselt ettevõtte vajadustele ehitatud veebirakendust, mille töö autor arendab valmis bakalaureusetööna. Lahendusega muudetakse ettevõttes töötavate projekti– ja müügijuhtide igapäevane projekte puudutav tööprotsess mugavamaks ja kiiremaks ning tõstetakse ettevõtte väärtust, kuivõrd kasutusele võetakse uudne ja praktiline lahendus.

Antud lõputöö raames tutvustatakse esmalt ettevõtet, kirjeldatakse lahendatavat probleemi ja projekti eesmärki.

Lõputöös kirjeldatakse rakenduse nõudeid, analüüsitakse erinevaid programmeerimiskeeli ning raamistikke ning valitakse välja sobilik keel ja raamistik nii tagarakenduse kui ka eesrakenduse arenduseks. Samuti võrreldakse andmebaasi tehnoloogiaid ning tehakse sobilik valik.

Töö käigus kirjeldatakse nii serverirakenduse kui ka kliendiliidese arendust. Serverirakenduse arenduse osas kirjeldatakse rakenduse struktuuri, REST API kontrollereid ning räägitakse rakenduse turvalisusest ning autentimisest. Eesrakenduse arenduse osas kirjeldatakse rakenduse ülesehitust ja toimimist ning analüüsitakse ka kasutajaliidese disainivalikut.

2 Probleemi püstitus ja projekti eesmärk

Käesolevas peatükis kirjeldatakse ettevõtet, vaadeldakse probleemi ehk varasemat olukorda ettevõttes projektide planeerimisel ja seatakse eesmärk probleemi lahendamiseks.

Lõputöös käsitletava tootmisettevõtte Structo Group OÜ üheks tegevuseks on töösse võetavate projektidega tegelemine nende planeerimisest kuni valmimiseni. Ettevõttes töötab kolm projektijuhti ja kaks müügijuhti ning kõikide ühiseks ülesandeks ja missiooniks on klientidelt soovitavate projektide töösse panemise ajaline planeerimine. Iga projekt on individuaalne ning toodete tootmine toimub eritellimuste täitmisel, koostöös ettevõtte partnertehastega - klaaside ja uste valmistajad ning furnituuride pakkujad. Töö tegemine algab kliendi ärakuulamisest, seejärel võtab projektijuht objektil toodete valmistamiseks mõõdud, mille järgi koostatakse joonis ja siis saab tehastelt tellida vajalikud tooted. Töö protsess on peaaegu alati sama, kuid tööde mahud ja lahendused on erinevad. Vahel on vaja paigaldada klaasvaheseinu suure büroohoone mitmele korrusele ning projektijuhi vastutus lõppeb alles siis, kui klient on nõuetele vastava töö vastu võtnud.

2.1 Ettevõtte tutvustus

Ettevõte Structo Group OÜ on loodud *ca* 12 aastat tagasi ja ettevõttes töötab tänaseks 14 inimest. Ettevõtte tegevuseks on eritellimusel MDF (*meedium-density fireboard*) – ja alumiiniumportaalide ning uste tootmine. Viimastel aastatel on ehitusprojektides hakatud kasutama laialdaselt klaasvaheseinu, mida ettevõte samuti projekteerib ja pakub. Structo Group OÜ poolt pakutavate erinevate standardtoodete hulk on üle saja ning ettevõte on alati valmis tootma ka kliendi poolt soovitud muudatustega tooteid.

Ettevõte on väga uuendusmeelne, kasvav ja soovib suunata tooteid ka välismaa turgudele. Juba mitmeid aastaid müüb ettevõte edukalt tooteid Rootsi ning Soome.

2.2 Probleem

Ettevõte pakub palju tooteid, erinevate viimistlustega ning nende tootmise ajakulu on erinev. Tavaliselt sisaldab üks projekt 20 – 50 toodet, suuremas projektis ulatub toodete arv sadadesse, mis võivad koosneda erinevate toodete osadest ja olla erineva viimistlusega. Ettevõtte koostööpartnerid peavad tootmisvõimekuse arvestust nädalates ning erinevates tehastes on erinev tootmisvõimekus ehk maht, mida suudetakse täita. Töötaja võiks seega juba projekti planeerides rakendusest näha, millal on tehastel vaba aeg ning millal on maht täis. Kuna projekte on palju, aga tootmisvõimekus on piiratud, siis peab selle korrektseks planeerimiseks olema sobilik tarkvara, mis ettevõtte meeskonda abistaks.

Kuni 2021. aasta suveni oli ettevõttes kasutusel ühine Exceli fail, kuhu töötaja märkis enda juhitava projekti, selle täitmise eeldatava ajakulu ning kui teine töötaja soovis samasse nädalasse enda projekti lisada, siis pidi ta välja arvutama, kas see sinna mahub. Arvutamine tähendas seda, et töötajal tuli planeerimisel arvestada toodete valmistamise ajaga, toodete paigaldajate graafikuga, kliendi soovide ja tähtajaga. Projekti aja- ja töömahukuse arvutamine oligi kõige suurem käsitöö, mille käigus tehti palju vigu. Juhul kui töötaja tegi valearvestuse, tekitas see koheselt muutused tarnes ning selle tõttu võisid edasi lükkuda ka teised planeeritavad projektid, mis kokkuvõttes on kahjulik ettevõtte tegevusele ja mainele.

Tähtsaks osaks projekti– ja müügijuhtide töös on kuude – ja aastakokkuvõtete tegemine, mis tuleb saata tehastele, et nemad saaks oma tööd planeerida. Eeldades, et graafik oli vigadeta täidetud, kopeeriti kogu info eraldi Exceli tabelisse ning selliselt loodi käsitööna kokkuvõttev tabel projektidest, projektiga seotud toodetest, toodete kogustest ning ajakulust. Kokkuvõtte tegemine võib Exceli mittetundmisel olla suur vigade tegemise koht ning risk, mida suure ja pideva tootmisega ettevõttel pole mõistlik kanda.

Exceli fail ei ole turvaline koht, kus hoida ettevõtte jaoks tähtsat infot, sest info võib kustuda või keegi võib tahtmatult eksida ja tabelis midagi valeks muuta või ära kustutada. Oluliselt turvalisem on hoida infot turvatud andmebaasis. Exceli faile saab küll samuti kaitsta parooliga, kuid nende lahtimurdmine pole keeruline ning peaaegu igaüks leiab vastava info vähese vaevaga Internetist üles. Vanas süsteemis on ka palju loogikat, mis aitas ettevõtte töötajal arvutusi teha, aga probleeme tekkis näiteks siis, kui keegi oli

omatarbeks ajutiselt valemeid muutnud ning unustanud endise olukorra taastada või oli kogemata valemid ära kustutanud ning faili uuesti salvestanud. Selliste vigade vältimiseks tuleks kasutusele võtta süsteem, milles pole võimalik loogikat iseseisvalt kellelgi muuta. Kokkuvõttes oli seega selge, et kiirelt kasvava ettevõtte jaoks oli Excelis igapäevane töötegemine ebamugav ja ei olnud jätkusuutlik. Aastatega kasvanud faili igapäevaselt aktuaalsena hoidmine ja funktsionaalsuse tagamine on üleliia mahukas töö, millega kaasnev ajakulu pole põhjendatud ega mõistlik.

Lõputööna valmiv veebirakendus lahendaks eelmainitud probleemid.

2.3 Eesmärk

Antud diplomitöö eesmärgiks on luua ettevõttele projektide planeerimise veebirakendus, millega saavad ettevõtte projekti– ja müügijuhid mugavalt, aega-säästavalt ja turvaliselt projekte planeerida. Eesmärgi täitmiseks on koostatud analüüs, mis käsitleb taga– ja esirakenduse programmeerimiskeele ja raamistiku valikut ja rakenduse loomist.

Oluline eesmärk veebirakenduse kasutuselevõtuga seoses, on saavutada vigade vähendamine ja andmete kadumine projektidest.

Autor seab eesmärgiks, et loodava veebirakenduse kasutamine peab olema töötajale lihtsalt arusaadav, pakkuma ülevaatlikkust ja olema mugav, kuna veebirakenduse peamine ülesanne on töötaja tööd lihtsustada ja abistada.

Rakendusse sisestatud projektide andmeid on võimalik operatiivselt ja selgelt edastada tehastele, laotöötajatele ning konstruktoritele oma töö tegemiseks.

Selleks, et luua ettevõttele vajalik rakendus, selgitab töö autor veebirakenduse nõuded välja, suheldes ettevõtte juhiga, projekti– ja müügijuhtidega, laotöötajate ja komplekteerijatega.

3 Rakenduse analüüs

Rakenduse analüüsis kirjeldatakse rakenduse nõudeid ning võrreldakse erinevaid sobilikke programmeerimiskeeli ja raamistike. Analüüsi ja võrdluse käigus selgub sobilik programmeerimiskeel ja raamistik nii tagarakendusele kui ka esirakendusele. Analüüsitakse veel andmebaasi valikut ning valitakse välja sobilik lahendus.

3.1 Rakenduse nõuded

Rakenduse nõuded koosnevad ettevõtte vajadustest, kasutajate soovidest ning autoripoolsetest ettepanekutest. Nõuete realiseerimiseks loob autor hajusveebirakenduse, mille teostamiseks luuakse nii taga– kui ka esirakendus.

Sobiliku sisendi saamiseks kasutati agiilset arendusmetoodikat, mis soodustab sagedast projekti kontrollimist ning vähendab teineteisest möödarääkimist.

Rakenduse kasutajad jaotatakse kahte rolli: administraator ja töötaja. Töötaja rolli kuuluvad ettevõtte müügijuhid ja projektijuhid. Töötaja saab rakendusse lisada uusi projekte koos projektiga seotud toodetega ning neid edaspidi hallata. Töötaja saab teha projektidest kokkuvõtteid vastavalt rakenduses tehtud filtreerimisvalikutele. Administraator saab teha kõike, mida töötajagi ning lisaks on temal õigus lisada, kustutada ja muuta tooteid, toodete viimistlusi ja tehaseid. Administraator saab vajadusel kasutajate paroole taastada ning ka kasutajaid kustutada.

Rakendus peab olema lihtsasti kasutatav ning vajaliku info leidmine peab olema kiire. Ettevõttel on aastas töös *ca* 1000 projekti ning kõikide projektide korraga kuvamine pole vajalik. Rakenduse avamisel kuvatakse alati käimasoleva kvartali projektid ning kasutaja saab vajadusel endale sobiva kvartali valida.

Planeerimise tähtsaimaks osaks on aeg. Rakenduses näeb, kui suure ajakuluga on lisatud projektid ning kui palju ajaressurssi on erinevates tehastes veel alles, et ei tekiks olukorda, kus mõni nädal on töödega ajaliselt üle broneeritud.

Rakenduse epikute ja kasutajalugudega saab tutvuda Lisas 2.

3.2 Rakenduse tehnoloogia valik

Rakenduse arenduse alustamiseks on vajalik eelnevalt ära teha mitu tähtsat valikut. Kõige tähtsam on programmeerimiskeele ja raamistiku valik, sest sellest sõltub, kuidas rakendust on võimalik ehitada ning kui kiiresti. Oluliseks valikuks on veel andmebaasi tehnoloogia valik, kus andmeid hoitakse. Arenduse mugavdamiseks tuleb veel valida sobilik integreeritud arendus -, koodihaldus - ja rakenduse majutuskeskkond.

3.2.1 Tagarakenduse programmeerimiskeele ja raamistiku valik

Tagarakendus vastutab selle eest, kuidas rakendus töötab, värskendab ja muudab andmeid. Õige tehnoloogia valik on esimene samm. Tagarakendus on tehnoloogia, mida on vaja sissetulevate päringute töötlemiseks ning väljaminevate vastuste genereerimiseks [1].

Tagarakendus jaotatakse mitmeks osaks: andmebaas, server ja rakendus ise. Tagarakenduse arendamiseks on vaja valida sobilik programmeerimiskeel, millest populaarseimad veebiarenduseks on JavaScript, Python, PHP, Java, Ruby, Golang ja C# [2].

JavaScript

JavaScripti on väga laialdaselt kasutatud veebiarenduses. Aastal 2019 kasutas 67,8% töötavatest arendajatest oma töös JavaScripti. 95,2% veebirakendustest kasutavad JavaScripti, kaasaarvatud lehed nagu Facebook, Youtube, eBay ning Google Maps. 1995. aasta septembris arendas Netscape programmeerija Brandan Eich uue skriptimiskeele kümne päevaga. Selle nimeks sai algselt Mocha, siis LiveScript ning hiljem JavaScript. Tänapäeval on võimalik kasutada JavaScripti ka tagarakenduste loomiseks, kasutades Next.js ja Express tehnoloogiaid. [2], [3]

Python

Pythoni arendas 1995. aastal Guido van Rossum. Python on populaarne eelkõige seetõttu, et seda on lihtne õppida, lugeda ja kasutada. Pythonit kasutatakse mitmel otstarbel, nii objekt-orienteeritult kui ka lühikeste skriptide kirjutamisel. Tagarakenduse arendamiseks Pythoniga kasutatakse näiteks raamistikke: Django, Flask ja Pyramid. [2], [4]

PHP

PHP on serveripoolne skripimiskeel, mis loodud 1994. aastal spetsiifiliseks veebiarenduseks. PHP populaarsuse tagab tema lihtsus ning suur nimekiri kasutatavatest teekidest. PHP toetab paljusid erinevaid andmebaase ning selle rakendused on paigaldatavad kõikidele platvormidele. [2]

Java

Java on klassipõhine objekt-orienteeritud programmeerimiskeel rakenduste arendamiseks. Java on kiire, turvaline ja usaldusväärne, mille tõttu kasutatakse seda paljudes rakendustes arvutites, andmekeskustes, mängukonsoolides ja telefonides. Java arendati 1995. aastal James Goslingu poolt ning esialgu oli keel mõeldud väikeste seadete programmeerimiseks. [5], [6]

Ruby

Ruby arendas Yukihiro Matsumoto Jaapanis 1990. aastate keskel. See loodi programmeerijate jaoks tootlikkuse eesmärgiga, tehes programmeerimise arendajale lõbusaks. See rõhutab vajadust, et tarkavara mõistaksid kõigepealt inimesed ja siis arvutid. Ruby kogub jätkuvalt populaarsust veebirakenduste arendamisel. David Heinemeieri poolt loodud raamistik The Ruby on Rails raamistik tutvustas paljudele inimestele programmeerimise rõõme. Rubyl on elav kogukond, mis toetab algajaid ja on motiveeritud kvaliteetse koodi loomisest. [7]

Golang

Golang on üpriski uus programmeerimiskeel, mis on loodud Google poolt 2009. aastal. Golang`i eeliseks on kiirus ning ta suudab mitut funktsiooni samaaegselt käivitada, mis teeb temast väga hea programmeerimiskeele suurte projektide jaoks. Golangil on võimas ja spetsiifiline veasüntaks, mis kiirendab vigade leidmist arenduse ajal ning lihtsustab nendega tegelemist. Golangil pole nii palju teeke ja raamisitikke kui konkurentidel, arvestades keele vanust. [8]

C#

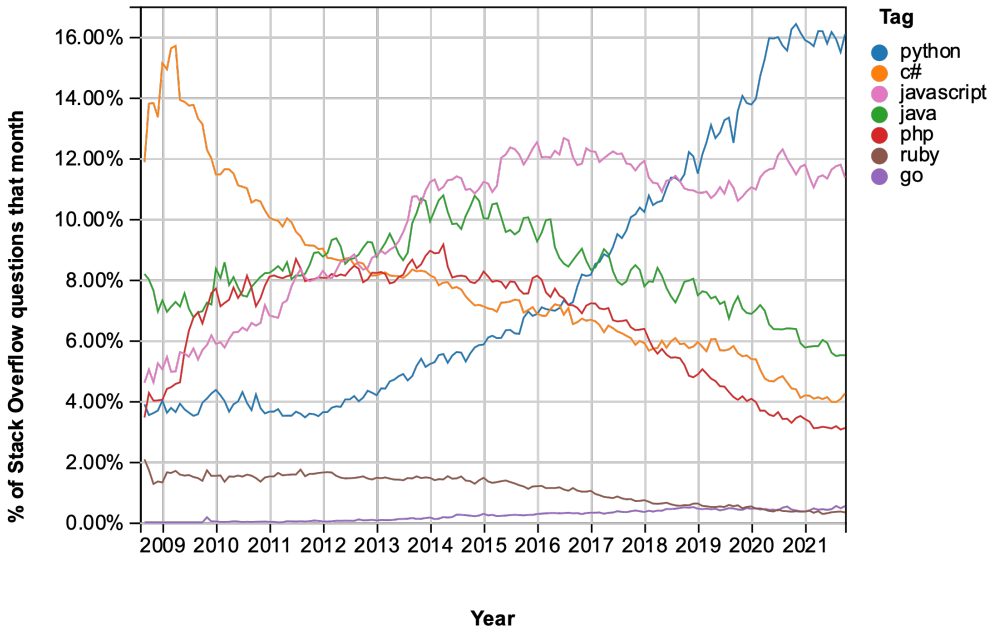
C# on üks neist keeltest, mis tabelites on pidevalt viie parima seas. Keel on objekt-orienteeritud vabavara, mida saab kasutada väga laialdaselt. C# arendati 2001. aastal Microsofti poolt. C# saab kasutada nii veebi-, mobiili– kui ka tarkvaraarendusel. C# eelistatakse sellepärast, et ta töötab igal platvormil, on turvaline, kindel ja väga kiire. C# põhiliseks raamistikuks on .NET, mille on samuti arendanud Microsoft ning mis uueneb ja kasvab konkurentidest kiiremini. [9]

Tabelis 1 võrdleb autor programmeerimiskeeli tabelis serverirakenduse arendamisel.

Tabel 1. Tagarakenduse programmeerimiskeelte võrdlus.

Keel	**Kogemus**	**Õppimise keerukus**	**Otsingu populaarsus lehel StackOverflow** **(Joonis 1)**
Java	Hea	Keskmine [10]	Keskmine
PHP	Nõrk	Lihtne – Keskmine [10]	Keskmine
JavaScript	Nõrk	Lihtne - Keskmine [10]	Kõrge
Python	Nõrk	Lihtne [10]	Kõrge
Ruby	Nõrk	Lihtne [10]	Madal
Golang	Nõrk	Lihtne - Keskmine [10]	Madal
C#	Väga hea	Keskmine [10]	Keskmine

Tagarakenduse programmeerimiskeele valimiseks on autor kaalunud eeltoodud tabelis esitatud aspekte ning lähtunud enda kogemusest programmeerimiskeele kasutamisel tagarakenduse loomisel. Autor kasutab programmeerimiskeele kohta valiku tegemisel ka kättesaadavaid materjale ja statistikat lehelt Stack Overflow, mis näitab, mitu protsenti kogu postituste hulgast on tehtud selle keele kohta (Joonis 1).



Joonis 1. Programmeerimiskeelte otsingute tulemuste võrdlus Stack Overflow lehel.

Jooniselt 1 nähtub, et populaarseim otsing on Pythonil *ca* 16% kogu postitustest. Sellele järgnevad JavaScript ca 11.5%, Java *ca* 6%, C# 4% ja PHP *ca* 3.2%-ga. Ruby ja Go otsingutulemused on alla 1%.

Võttes arvesse lõputöö kirjutamise ja rakenduse kirjutamiseks piiratud aega, on uue keele õppimine aeganõudev ning mõistlikuim lahendus on valida keeled, millega autor on kõige rohkem siiani kokku puutunud. Nendeks on hetkel Java ja C#.

C# omab sarnaselt Javaga pakettide haldurit nimetusega NuGet, mis lubab kiiresti alla laadida vajalikke pakette rakenduse loomisel, kasutades lihtsat kasutajaliidest. Java kasutab pakettide haldamiseks Gradle või Maven tarkvara. Arvestades tehtud analüüsi, Stack Overflow otsingutulemust ja autori kogemust, on C# mõistlikuim tagarakenduse programmeerimiskeele valikuks. Antud juhul on autori jaoks sobilikum C# ka põhjusel, et autor on seda keelt varasemalt töö juures arenduseks kasutanud ning uue keele õppimine piiratud ajaga ei oleks põhjendatud.

Järgmine osa on programmeerimiskeele raamistiku valimine. Raamistik on tarkvaraplatvorm, mis vähendab arendamisaega ning lihtsustab koodikirjutamist. Raamistikel on eelarendatud ja testitud funktsionaalsused, mille kasutamine vähendab vigade tekkimist. Iga kord kui alustatakse uue tarkvara arendust, ei ole ideaalne nullist

alustada, vaid parem on selleks kasutada raamistikke, mis teevad suure osa arendaja tööst juba ära. [11]

Kuna serveripoolseks keeleks valiti C#, siis sobilikeks raamistikeks on ASP.NET Core ja .NET Framework.

.NET Frameworki arendas välja Microsoft 2002. aastal. Raamistikku kasutatakse erinevate rakenduste arenduseks, mille kasutamine on mõeldud Windows operatsioonisüsteemiga masinates [12]. Viimane versioon raamistikust on versioon numbriga 4.8, mis ilmus 2019. aasta 18. aprillil ning uusi versioone enam ei tule. Microsoft otsustas hoida ühte raamistikku ning tuli välja .NET 5-ga, mis ühendas varasema .NET Frameworki ja .NET Core 3. Kuna uuendus oli nii suur, otsustas Microsoft 4. versiooni vahele jätta. [13]

.NET Core on Microsofti poolt arendatud tasuta tarkvaraplatvorm, mis töötab kõikidel platvormidel ning on pidevalt uuenev. Raamistikku saab kasutada laialdaselt: veebi-, mobiili– ja arvutitarkvara arendusel, samuti veel mikroteenuste ja mängude ehitamisel. Kuna autor soovib rakendust hoida Dockeri konteineris ning hiljem majutada pilveteenuses, mille platvormiks on Linux, siis paremaks raamistikuks sobib .NET Core. Lõputöö praktilise osa tegemise ajal oli viimaseks versiooniks .NET Core 5, nii et tehtud rakendus kasutab seda. Kõige uuem versioon on .NET Core 6, mis ilmus 8.11.2021. a. [14], [15]

3.2.2 Eesrakenduse programmeerimiskeele ja raamistiku valik

Eesrakenduse ülesanne on kuvada (renderdada) veebilehte külastavale kasutajale funktsioneeriv kasutajaliides. Eesrakenduse põhilised kolm komponenti on HTML, CSS ja JavaScript. JavaScript tagab veebilehe interaktiivsuse, mis lubab lisada kasutajaliidesesse komponente nagu hüpikaknad, menüüs navigeerimine, vahelehed, andmete valideerimine ja palju muud. [16]

Kuna autor valis tagarakenduse programmeerimiskeeleks C#, siis on võimalik kasutada eesrakenduse arendusel ASP.NET raamistikku, mis asendab JavaScripti osa C# keelega ning kasutaks MVC mustrit. Kuna autor soovib kasutajale luua eraldiseisvat eesrakendust ning serveriga suhtlemiseks REST API tehnoloogiat, siis jääb valikusse ainult JavaScript ja TypeScript. JavaScript on loodud kliendirakenduse arendamiseks, kuid sellel on

mõningad puudused. Nimelt puudub JavaScriptil staatiline tüüpimine ja kompileerimisel esinevate vigade kuvamine. Eelneva probleemi lahendamiseks on loodud TypeScript. TypeScript on põhimõtteliselt täiustatud JavaScript, mis toob esile kompileerimisel tekkinud vead ning muudab arendusprotsessi mugavamaks ja kiiremaks, lisades kohustusliku tüüpimise. Autor soovib, et rakenduse kirjutamine oleks kiire, ning et arenduse käigus kerkiksid vead kindlasti esile. Samuti soovib autor, et kirjutatud kood oleks korrektne ning selle tõttu valis eesrakenduse programmeerimiskeeleks TypeScripti.

Eesrakenduse raamistikke on palju, kuid autor võtab võrdlusesse need, millega tal on olnud isiklik kokkupuude. Valikusse jääb kolm populaarset raamistikku: React.js, Vue.js ja Aurelia.

React.js

React on valikust enimkasutatav JavaScripti raamistik, arendatud Facebooki arendaja Jordan Walke poolt, et luua kiireid ja interaktiivseid kasutajaliideseid veebis ja mobiilirakenduses. Reacti populaarsuse tagab lihtne kasutus ja võimekus. Võrreldes ainult JavaScriptiga, saab Reactiga vähese koodiga luua dünaamilisi veebilehti. Reacti rakendus on ülesehitatud komponentidel, mis on taaskasutatavad ning uuenevad vastavalt vajadusele, muutes rakenduse kiireks. Reactis hoitakse vaadet ja loogikat koos. [17]

Vue.js

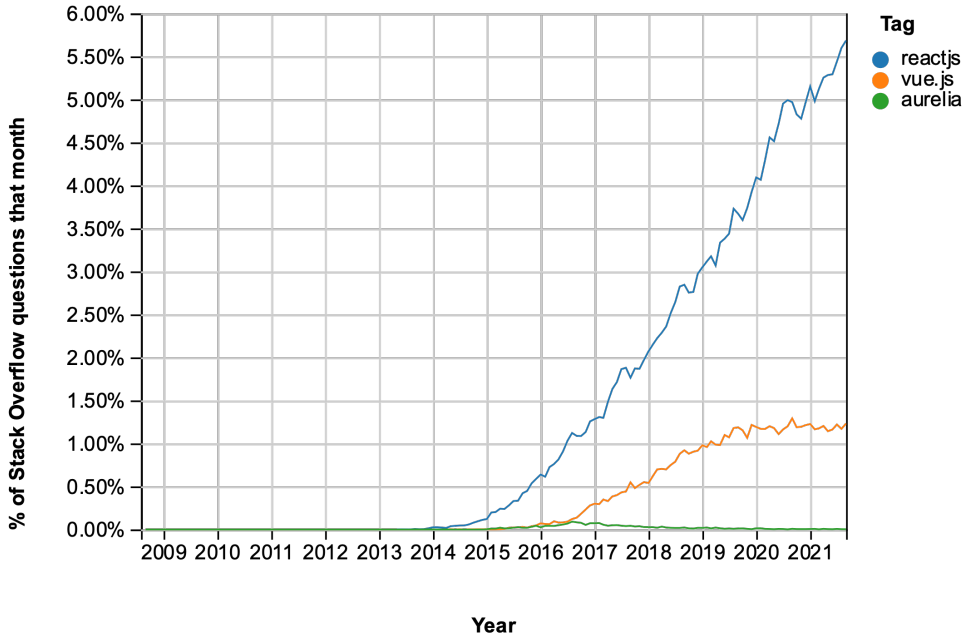
Vue.js on populaarne ja progressiivne JavaScripti raamistik, millega saab luua interaktiivseid ühe-lehe kasutajaliideseid. Raamistiku õppimine on kiire, kui oskad varasemalt JavaScripti, HTML ja CSS-i. Vue on loodud 2014. aastal Evan You poolt ning on algusest peale oma kasutajate kogukonda kasvatanud, tulles välja pidevalt uuendustega. [18]

Aurelia

Aureliat võib pidada „järgmise põlvkonna raamistikuks“, kuna see ehitati ES6 ja ES7 abil. Raamistik on kirjutatud kaasaegses JavaScriptis, mis tähendab, et kasutajatel on saadaval algmoodulid (*native modules),* klassid ja palju muud. Aurelia kasutab HTML – põhist mallisüsteemi ja API-t, mis muudab õppeprotessi sujuvaks kõigile, kes on kaasaegse JavaScriptiga tuttavad. Aurelia on algusest peale palju tuntust kogunud ning

sellega on arendatud tuhandeid veebilehti. Aurelia erineb teistest võrdluse all olevatest raamistikest selle tõttu, et loogikat ja vaateid hoitakse rakenduses eraldi. [19]

Joonisel 2 on näidatud eelnevalt mainitud JavaScript raamistike otsingupopulaarsust veebilehel StackOverflow.



Joonis 2. JavaScripti raamistike otsingute tulemuste võrdlus Stack Overflow lehel.

Tabelis 2 võrreldakse JavaScripti raamistikke React.js, Vue.js ja Angular.js.

Tabel 2. JavaScripti raamistike võrdlus.

Raamistik	**Kogemus**	**Õppimise keerukus**	**Otsingu populaarsus lehel StackOverflow**
React.js	Väga hea	Keskmine [20]	Kõrge
Vue.js	Keskmine	Lihtne [20]	Keskmine
Aurelia	Nõrk	Raske [20]	Madal

Töö autoril on kõige rohkem kogemust JavaScripti raamistikest olnud React.js kasutamisega ning kuna selle raamistiku kohta on lehelt Stack Overflow võimalik saada veel palju abi ja lisainfot, siis tehti valik React.js kasuks.

3.2.3 Andmebaasi tehnoloogia valik

Andmebaasid jagunevad erinevatesse tüüpidesse: relatsiooniline, mitterelatsiooniline, hierarhiline, objekt-orienteeritud, võrgu mudeliga, dokument-orienteeritud ja graaf [21]. Kõige populaarsemad tüübid veebirakenduste arenduses on relatsiooniline, dokument-orienteeritud ning võti-väärtus paaride andmebaasid. Andmebaasi valiku tegemisel võrreldakse kõige populaarsemaid andmebaase ning andmebaase, millega töö autor ülikooliõpingute jooksul kokku puutus. Otsuse teeb töö autor vastavalt rakenduse vajadustele, piiratud aja tõttu ettevõtte soovil rakendus kiiresti valmis saada aga ka varasemale kogemusele.

Andmebaaside valikusse valis töö autor järgmised populaarsed süsteemid: MySQL, MS SQL Server, ja PostgreSQL, mis on kõik relatsioonilised andmebaasid (SQL ehk *Structured Query Language*). SQL loodi 1970. aastate alguses IBM-is, et pääseda ligi IBM System R andmebaasisüsteemile. [22]

MySQL on populaarsuse edetabeli tipus olnud juba mitu aastat, sest see on tasuta ning töötab enamiku rakenduste jaoks ning kõikidel populaarsetel platvormidel. MySQL-i käivitasid 1994. aastal Rootsi Kuningriigis programmeerijad David Axmark ja Michael Widenius. 2008. aastal ostis MySQL-i IT-gigant Sun Microsystems. [23]

MS SQL Server on Microsofti toode, mis loodi aastal 1989. Paljud ettevõtted usaldavad Microsofti ning kasutavad selle firma muid lahendusi ning ei soovi oma IT-ökosüsteemi lisada midagi muud. MS SQL Serverit leiab tihtipeale paljudes ettevõtetes, eriti suuremates. Vaatamata avatud lähtekoodiga lahenduste domineerimisele turul, on SQL Server olnud edukas. [23]

PostgreSQL on tasuta, avatud lähtekoodiga ja töötab kõigis võimalikes olukordades ning kõikidel platvormidel. Sellel on väga dünaamiline kasutajate kogukond, kes aitavad projekti arendada ning kirjutada oma pistikprogramme (*plugins*) ja lahendusi. PostgreSQL-i kasutavad ettevõtted nagu Instagram, Twitch ja Skype, lisaks kinnitab StackOverflow uuring, et PostgreSQL-i lisandus kasutajaid igal aastal. [23]

Lõputöö veebirakenduse andmebaasi tehnoloogiaks valiti SQL Server, mis on töökindel ja stabiiline ning töötab .NET rakendusega. Peatükis 3.5 valitakse veebirakenduse halduskeskkond, mis on hästi kooskõlas SQL Serveri andmebaasiga.

3.3 Integreeritud arenduskeskkonna valik

IDE-sid on kasutusel palju. NET rakenduste arendamiseks on tuntuimad IDE-d Visual Studio, Jetbrains Rider, ja Visual Studio Code. [24]

Visual Studio ja Visual Studio Code (VS Code) on loodud Microsofti poolt. Mõlemad on kasutatavad Windowsil ja macOS operatsioonisüsteemidel ning VS Code lisaks veel Linuxil. Mõlemad on tasuta kasutatavad vabavarad, millega saaks käesoleva projekti valmis kirjutada. Visual Studiole on võimalik lisada ReSharper laiend, millega saab programmeerimisprotsessi mugavamaks muuta. VS Code-l on samuti olemas laiendeid, mida saab kasutada C# koodi kirjutamiseks, kuid negatiivseks asjaoluks on see, et tegu on tekstiredaktoriga, mitte IDE-ga. Kasutades tagarakenduse arendusel .NET raamistikku, teeb programmeerimise tõhusamaks võimas IDE, mis automatiseerib projekti ülesehitamist ja aitab programmeerijat, pakkudes ise korrektseid või alternatiivseid lahendusi koodi kirjutamisel. Eelneval põhjusel jääb VS Code Visual Studiole alla.

Rider on JetBrainsi multiplatvormil töötav IDE, millel on erinevatel platvormidel samad funktsionaalsused. Võrreldes Visual Studioga on see suur boonus, kuna Visual Studios erinevad platvormivahelised rakendused teineteisest. Rider näeb välja ning töötab igal pool samamoodi. JetBrains Rideriga on lihtne luua uut .NET veebirakendust ning projekti luues on võimalik valida rakendusele sobiv mall. Rideri ainukeseks nõrkuseks on see, et tarkvara on litsentsipõhine ning tasuline. [25]

Sobivaks IDE-ks kujunes arendamisel JetBrains Rider, kuna arendusmasinaks kasutati macOS operatsioonisüsteemi ning õpilasetele kehtiva litsentsi tõttu sai võimsat IDE-d kasutada tasuta.

3.4 Koodihalduskeskkonna valik

Lahutamatuks osaks tarkvaraarendusel on versiooni haldus, selleks on vaja valida projektile sobilik koodihalduskeskkond (VCS ehk *version control system*). VCS pakub

standardiseeritud viisi arendatud tarkvara lähtekoodi salvestamiseks. VCS hõlbustab meeskonnakaaslaste vahelist koostööd ning oluline on, et säilitab kogu lähtekoodis tehtud muudatuste ajaloo.

Kolm populaarseimat VCS valikut on: Git, SVN ja Perforce. [26]

SVN

SVN on veteran – VCS – valik, mida 2000. aastal tutvustas CollabNet. SVN sai Apache perekonna osaks 2009. aastal. SVN on tsentraliseeritud versioonihaldussüsteem ning see on tööstusettevõtete ning vanemate ettevõtete lemmik. Kuigi SVN-i väljakujunenud olemus võib olla selle tugevus, kritiseeritakse SVN-i selle vananenud funktsionaalsuste tõttu. [26]

Git

Teiseks versioonikontrolliks on Git, mis on aastast 2005 ja saanud väga populaarseks lahenduseks arendajate seas. Git on hajutatud versioonjuhtimissüsteem ning avatud lähtekoodiga. Giti luues oli eesmärgiks kiirus, lihtne disain ja hajus mudel. Git on ideaalne agiilset meetodit kasutavatele arendustele, kuna arendajad saavad vaadata oma rakenduse hoidla täielikku ajalugu, mis teeb töövoo kiiremaks. [26]

Perforce

Perforce on mängu– ja VR/AR– stuudiote lemmik. Perforce peetakse tööstusharu standardiks ning sellel on palju tugevusi. Perforce peamiseks tugevuseks on kiirus, selle hoidlad hoiavad miljoneid faile ja palju terabaite andmeid. Uuringud on näidanud, et Perfoce on suure hulga failide sünkroonimisel 5-10 korda kiirem kui SVN. Perforce kasuks tasub otsustada, kui lahenduses on palju suuremahulisi faile. [26]

Käesoleva töö koodihalduskeskkonda valides, oli töö autori jaoks tähtsateks kriteeriumiteks kiirus, mugavus ja kogemus. Kuna arendamisel on kasutatud agiilset meetodit, kus lähtekoodi ajaloole juurdepääs on tähtis ning rakendus ei kasuta suuremahulisi faile, siis antud rakenduse jaoks kasutas töö autor Giti.

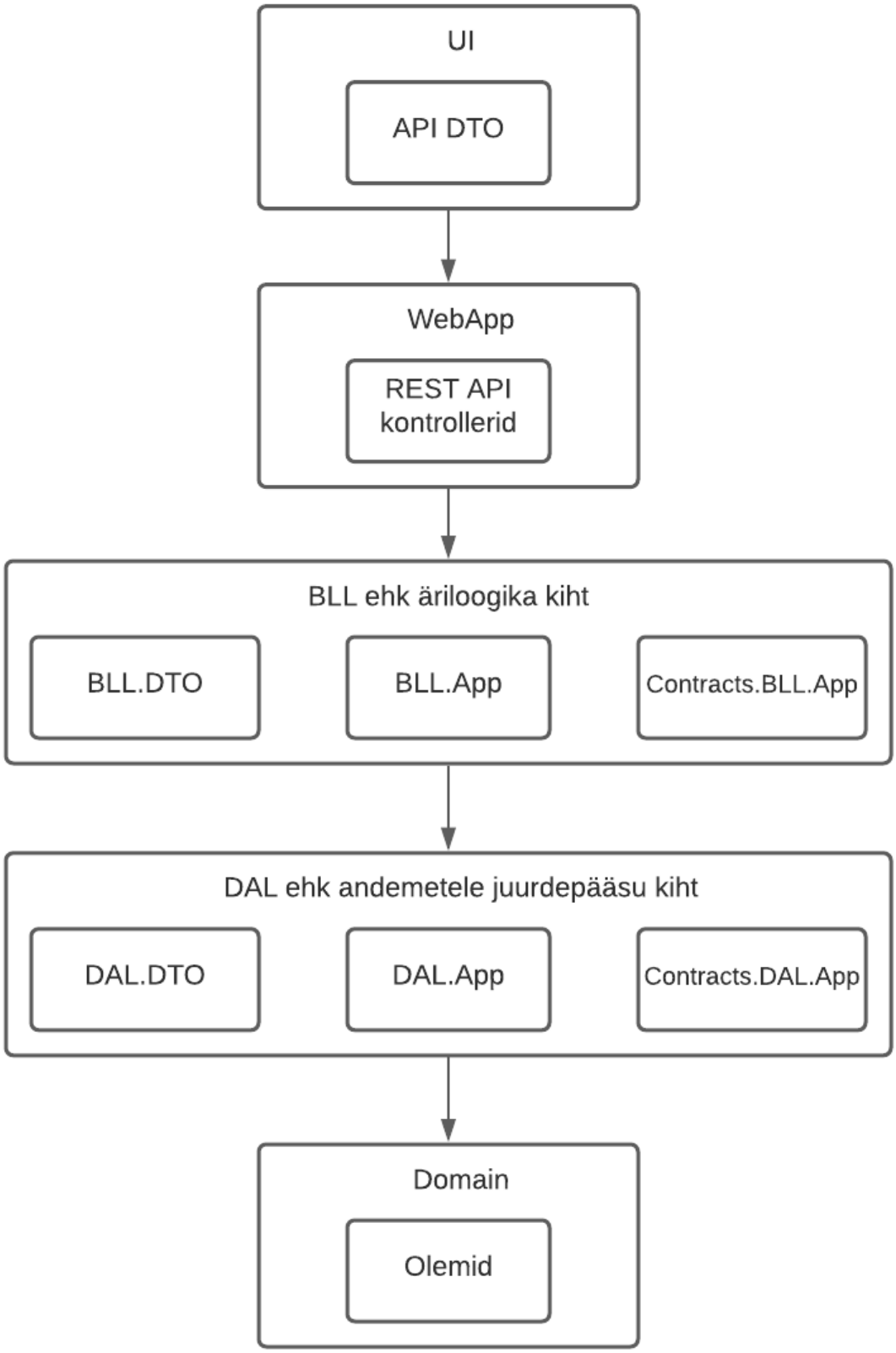
3.5 Veebirakenduse halduskeskkond

Projektide planeerimise veebirakenduse kasutus on mõeldud vaid ettevõtte töötajatele ning laiemal avalikkusel puudub sellele ligipääs, nii et majutamiseks pole vaja võimsat masinat. Tagarakenduse majutamiseks on kasutatud Microsofti Azure keskkonda, kus piisas täiesti tasuta versioonist. Azure keskkonnas saab tagarakendust ka kiiresti ja lihtsalt uuendada, kasutades Dockerit.

3.6 Veebirakenduse arhitektuur

Veebirakenduse arhitektuur on allpool välja toodud joonisena (Joonis 3) ning koosneb järgmistest osadest:

- Andmemudeli kiht (*Domain*): baas-olemid ja olemid.
- Andmetele juurdepääsu kiht (*Data Access Layer*): üldine andmetele juurdepääsu liides, kus asuvad domeeniobjektide hoidlad ja hoidlate liidesed.
- Äriloogika kiht (*Business Logic Layer):* üldine äriloogika liides, teenuste liidesed ja teenused.
- REST API (*representational state transfer application programming interface*): tagarakendusega ühendumise liides
- Veebiliidese ehk kasutajaliidese kiht (User Interface): veebirakendus.



Joonis 3. Veebirakenduse arhitektuur.

4 Projektide planeerimise rakenduse arendus

Rakenduse arenduse peatükis on välja toodud andmebaasimudeli loomine ning primaarvõtme valik. Tagarakenduse arenduse peatükis on kirjeldatud projekti loomine ning tagarakenduse struktuur, kasutatavad teegid ning REST API kontrollerite toimimine ja turvalisus koos koodinäidistega. Eesrakenduse arenduses on välja toodud React raamistikuga rakenduse loomine ning ülesseadistamine ning kasutajaliidese kasutamine.

4.1 Andmebaas

Enne rakenduse arendamist tuleb luua rakenduse andmemudel ja valida sobilik primaarvõtme tüüp. Käesolevas töös võeti andmebaasi primaarvõtmena kasutusele GUID (*Globally Unique Identifier*), mis on väga hea valik tagarakenduse andmebaasil [27]. GUID tagab väga suure tõenäosusega, et iga genereeritud GUID on unikaalne läbi kõikide andmebaasi tabelite. Tõenäosus, et sama GUID genereeritakse kaks korda, on $1/2^{128}$. Andmebaasis on igal tabelil üks primaarvõti, mis genereeritakse automaatselt. GUID-i kasutamisel on miinuseks see, et võtmed nõuavad 16 baiti salvestusruumi ning suuremate andmebaasidega võivad tekkida jõudlusprobleemid, kui antud rakendus nii suureks ei lähe. [28]

Analüüsi käigus valis töö autor kasutatavaks tehnoloogiaks MS SQL Serveri. SQL serveril on GUID-i jaoks oma andmetüüp nimega *uniqueidentifier*.

4.1.1 Olemisuhtediagramm

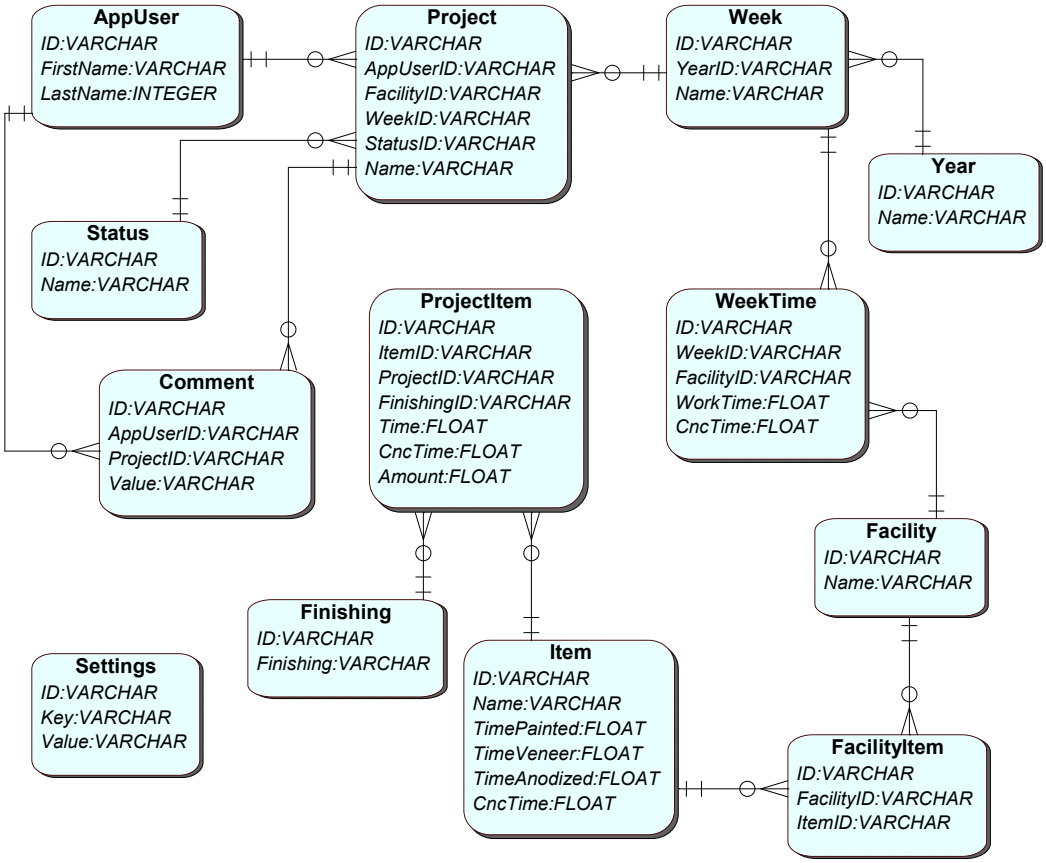
Rakenduse nõuetest ja vajadustest lähtudes on loodud olemisuhte diagramm, mis enne koodi kirjutamist annab hea ülevaate tulevasest domeenikihi struktuurist. Diagrammi tegemiseks on kasutatud QSEE tarkvara.

Rakenduse andmebaasi olemisuhtediagramm on näidatud joonisel 4. Andmebaas koosneb järgnevatest tabelitest:

- Comment – kommentaaride tabel;
- Facility – tehaste tabel;
- Item – toodete tabel;

- FacilityItems – tehaste ja toodete vahetabel;
- Finishing – viimistlustüüpide tabel;
- Project –projektide tabel;
- ProjectItems – projektide ja toodete vahetabel;
- Status – projektide olekute tabel;
- Week – nädalate tabel;
- Year – aastate tabel;
- WeekTime – tehastega seotud nädala töö – ja CNC – võimekus (maksimaalne vaba aeg);
- Settings – muud rakendusega seotud muutujad;
- AppUser – rakenduse kasutajad.

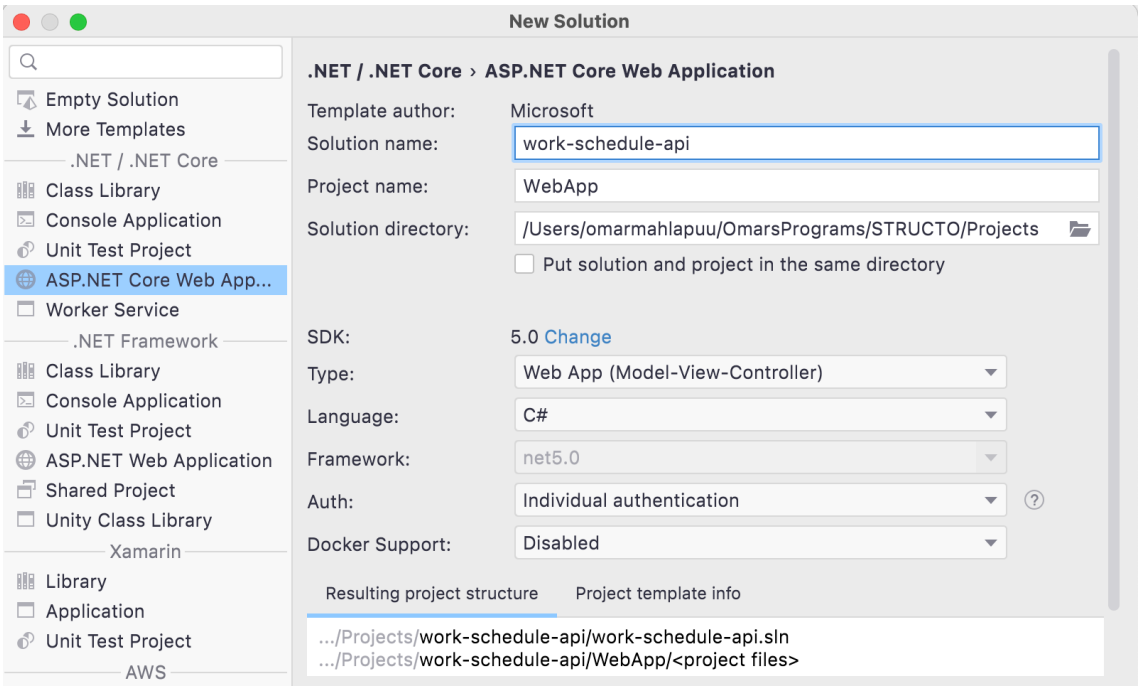
Välja pole toodud *Entity Frameworki* poolt loodud lisatabeleid *Identity* kasutamisel.



Joonis 4. Olemisuhtediagramm.

4.2 Tagarakenduse arendus

Tagarakenduse arendusel on kasutatud C# programmeerimiskeelt, kasutades .NET Core raamistikku. Tagarakendus tegeleb andmebaasiga suhtlemisega, sisaldab äriloogikat ning vastutab andmete pärimise, sisestamise ja salvestamise eest läbi REST API. Rakenduse arendamise alustamiseks tuleb luua uus projekt. JetBrains Rideriga on uue projekti loomine mugav ja arusaadav (Joonis 5).

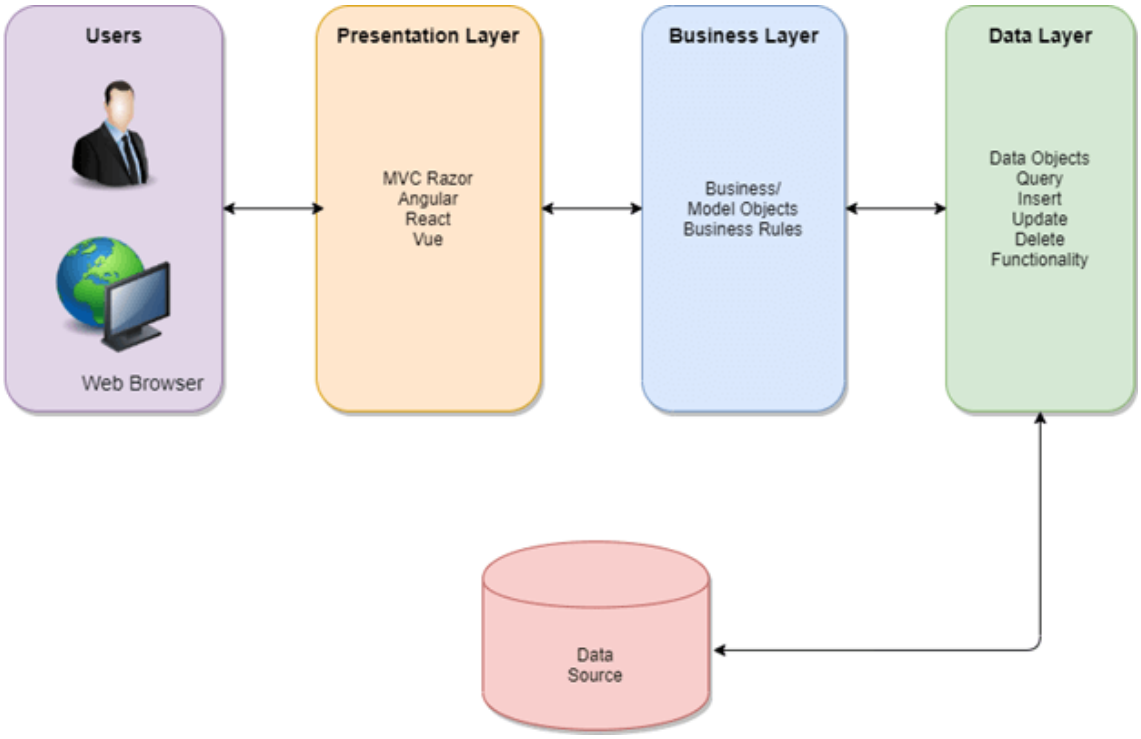


Joonis 5. Uue projekti loomine JetBrains Rideris.

Uue veebirakenduse projekti alustamiseks *JetBrains Rideris* on valitud *.NET Core ASP.NET Core Web Application* mall ning rakenduse tüübiks Web App (*Model-View-Controller*), mis sobib kui luuakse rakendus, kasutades ASP.NET Core MVC Views mustrit kui ka RESTful API tehnoloogiat. Kuna rakendust kasutavad töötajad sisselogimisega, siis autentimise arenduse abistamisel on valitud autentimismalliks *Individual authentication*.

4.2.1 Tagarakenduse struktuur

Tagarakenduse arendamiseks on kasutatud mitmekihilist arendusstruktuuri (Joonis 6), mis teeb koodi haldamise lihtsamaks ning koodi kirjutamise agiilsemaks. Mitmekihiline rakendus koosneb mitmest kihist, millel on oma kindel ülesanne ja vastutus. [29]

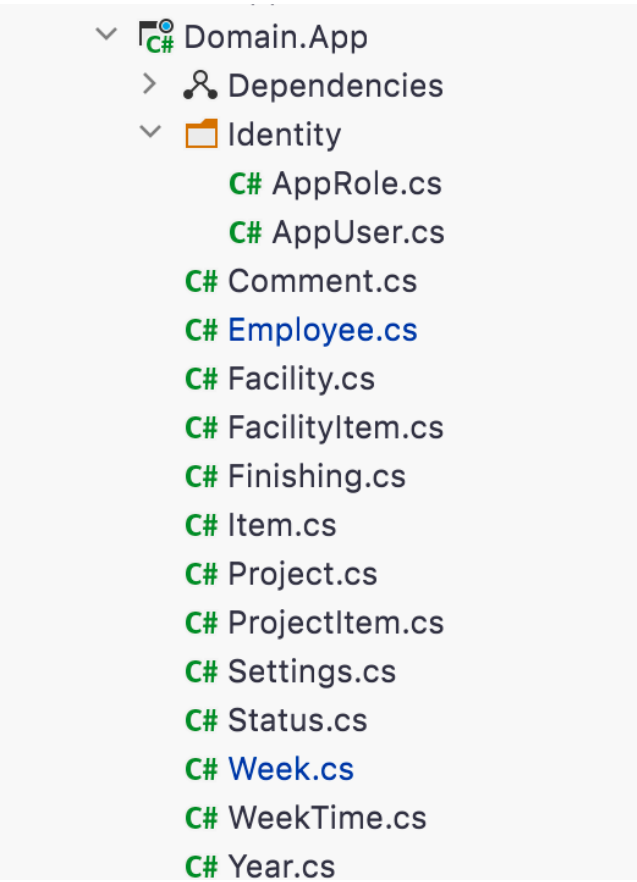


Joonis 6. Mitmekihiline arendusstruktuur. [30]

Mitmekihilise rakenduse kihtideks on andmebaasikiht, andmetele juurdepääsu kiht (*data access layer ehk* DAL), äriloogikakiht (*bussiness logic layer* ehk BLL) ja esitluskiht (*presentation layer*). Mitmekihilises süsteemis sõltub iga kiht selle all olevast kihist ning ühelgi kihil pole aimu, mis toimub selle kohal. Selline arendusmuster lubab arendajal tegeleda ühe asjaga korraga, kirjutades korrektsemat ja hallatavamat koodi. Töötades kindla kihiga, võib teiste kihtide ülesanded ära unustada ning keskenduda ainult sellele kihile, mis jätab arendajale rohkem mõtlemisruumi. [30]

Mitmekihilise mustri kasutamiseks peab aru saama, mis on iga kihi ülesanne. Rakenduse töötamine algab andmebaasikihist. Selles kihis asub rakenduse andmebaas, kus hoitakse rakendusega seotud andmeid. Andmebaasi loob .NET raamistiku pakett *Microsoft.EntityFrameworkCore*, kasutades migratisiooni (*EF Migrations*) [31].

Esimesena luuakse uus *Class Library,* nimega Domain.App, kus hoitakse rakenduse domeenimudelit (Joonis 7). Loodud andmemudelist genereeritakse andmebaas, kasutades käsku *dotnet ef migrations –project DAL.App.EF –startup-project WebApp Add InitalDbCreate.* Käsus on määratud ära *Entity Framework Core* tähtsa klassi *DbContext* asukoht ning rakenduse käivitamismeetodi asukoht. *DbContext* on klass, mida rakendus kasutab, et suhelda andmebaasiga, tagades ühenduvuse ning andmete pärimise ja salvestamise [32].



Joonis 7. Class Libary Domain.App sisu.

Järgmiseks kihiks on DAL. DAL kiht kasutab andmebaasiga suhtlemiseks hoidlamustrit (*repository pattern*), mille eesmärgiks on andmekihi abstraktsioon. Andmebaasiga seotud toimingud: lisamine, kustutamine, uuendamine ja muutmine tehakse läbi otsekoheste meetodite, ilma, et peaks muretsema andmebaasiga ühendumise pärast. Hoidlamustrit kasutades on igal domeeniobjektil oma hoidlaklass, kus on kirjeldatud andmebaasiga suhtlusloogika. [33]

Kihis kasutatakse *Entity Framework Core*, mis eemaldab vajaduse kirjutada madala tasandi andmebaasiga suhtlemise koodi ning võimaldab lihtsasti rakenduse andmebaasi tehnoloogia väljavahetust. [34]

Vältimaks loogika kirjutamist kontrolleritesse, on järgmiseks kihiks äriloogika kiht ehk BLL. BLL ülesanne on luua ühendus esitluskihi ning andmetele juurdepääsukihi vahel. Igale domeeniobjektile on kihis teenuseklass (*service*), mis vastutab vajalike funktsionaalsuste ja arvutuste eest. BLL kasutamine teeb äriloogika koodi loetavamaks ja hallatavamaks [35]. BLL kihis kasutatakse arvutuste tegemiseks ka teiste domeeniobjektide teenusklasse, see on võimalik läbi UOW (*Unit of Work*). UOW klassis

hoitakse kõiki domeeniobjektide hoidlaid, mida saab kasutada erinevates teenuseklassides. Andmete muutmine võib tekitada palju väikseid päringuid andmebaasile, mis teevad rakenduse aeglasemaks. UOW aitab rakendusel meeles hoida tehtud muudatusi, mis võib andmebaasi muuta ning kui kõik vajalikud muudatused on selged, koondab UOW kõik andmebaasipäringud ühte transaktsiooni, vähendades koormust ja suurendades jõudlust. [36]

Viimaseks rakenduse kihiks on esitluskiht, kus asuvad REST API kontrollerid, mis tagastab kasutaja tehtud päringule sobiva vastuse. Iga domeeniobjekti kohta on eraldi REST API kontroller, kus erinevatele tehingutele vastab kindel REST lõpp-punkt. REST lõpp-punktidele päringute tegemine kasutab vastuse saamiseks domeenobjekti teenuseklassi meetodeid.

Andmete liikumine läbi erinevate kihtide käib läbi andme-edastusobjektide ehk DTO (*data transfer object).* Igal domeeniobjektil on igas kihis vastav DTO ainult vajalike parameetritega. DTO-de kasutamisel jagavad esitluskiht ja teeninduskiht omavahel domeeni objektide asemel andmelepinguid (*data contracts*). Andmeleping on sisuliselt neutraalne esitlus andmetest, mida interakteeruvad komponendid vahetavad. Andmeleping kirjeldab andmeid, mida komponent saab, kuid see pole rakendusespetsiifiline klass ehk olem. Andmeleping on abiklass, mis on spetsiaalselt loodud konkreetse teenindusmeetodi jaoks. DTO aitab komplekteerida mitme päringu andmeid ühte objekti, vähendades objektide suurust ning hulka. [37]

Domeeniobjektide konfigureerimiseks kasutab *Entity Framework Fluent* API mustrit (Joonis 8), mis on objekt-orienteeritud API, mille disain tugineb suuresti meetodite aheldamisel. *Fluent* API eesmärk on suurendada koodi loetavust. *Fluent* API termini võtsid kasutusele esimesena Eric Evans ja Martin Fowler 2005. aastal. [38]

```
builder.Entity<Week>()
      .HasMany(m => m.PojectItems)
      .WithOne(o => o.Week!)
      .OnDelete(DeleteBehavior.Cascade)
```

Joonis 8. Fluent API koodinäide.

Andmetele juurdepääsu saamiseks domeeniobjektide hoidlates on kasutatud LINQ (*Language Integrated Query*) avaldisi. LINQ on andmepäringute API, millel on SQL-i sarnane päringu süntaks. LINQ avaldisega saab andmeid hankida, mis tahes objektilt, mis

rakendab *IEnumerable<T>* liidest. LINQ-i eesmärgiks on lisada .NET Frameworki üldotstarbelised päringuvõimalused, mis kehtivad kõikidele andmeallikatele, mitte ainult relatsiooniliste – või XML-andmete jaoks. [39]

LINQ avaliste kasutamise eelisteks on:

- Objektipõhine, keelega integreeritud viisil andmepäringute tegemine, olenemata, kust andmed pärinevad.
- Kompileerimisel süntaksi kontroll.
- Lubab loetavamalt teha päringuid massiividest (*array*), loendatavatest klassidest (*enumerable classes*) jm, sarnaselt päringute tegemisel läbi SQL-i.

Allpool on esitatud näide, kuidas LINQ avaldisega on tehtud klassi *Comment* päring, tagastades kõik kindla projekti kommentaarid.

```
public async Task<IEnumerable<DTO.Comment>>
GetAllAsyncByProject(Guid projectId,
    Guid userId = default, bool noTracking = true)
{
    var query = CreateQuery(userId, noTracking);
    var resQuery = query
        .Include(p => p.AppUser)
        .Where(p=> p!.Project!.Id == projectId)
        .Select(p => _mapper.MapToDal(p));

    return (await resQuery.ToListAsync())!;
}
```

Joonis 9. LINQ avaldise koodinäide.

4.2.2 REST API kontrollerid

Käesoleva tagarakenduse teenustega suhtlemiseks on kasutatud RESTful mustrit. REST ehk *Representational State Transfer* on rakenduse programmeerimisliides, mis lubab kasutajaliidesel suhelda veebiteenusega. REST arhitektuuri lõi arvutiteadlane Roy Fielding, et andmeid oleks võimalik esitluslikult edastada. [40]

Rakenduses kasutatakse REST API-t, et kasutaja saaks läbi kasutajaliidese teha päringuid veebiteenusele. Selleks on igale domeeniobjektile loodud REST API kontroller, mis vastab kindlatele lõpp-punktidele. API kontrollerite genereerimiseks on kasutatud käsurea tööriista *aspnet-codegenerator,* millega saab kiiresti iga domeeniobjekti kohta

genereerida algelise API kontrolleri. Käsu kasutamiseks on vajalik tööriista installimine, milleks on kasutatud käsku *dotnet tool install -g dotnet-aspnet-codegenerator* [41].

Igale kontrollerile saab teha HTTP meetodite: GET, POST, PUT ja DELETE päringuid JSON formaadis. Korrektse päringu tegemiseks on igal kontrolleril oma lõpp-punkt, kontrollerid kasutavad mustrit /api/v<versiooni number>/<kontrolleri nimi>. Allpool on esitatud, kuidas käib C#-s API kontrolleri lõpp-punkti tähistamine annotatsioonidega.

```
[ApiController]
[Route("api/v{version:apiVersion}/[controller]")]
[ApiVersion("1.0")]
```

Joonis 10. API kontrolleri annotatsioonid.

GET

Andmete pärimiseks kasutatakse GET meetodit. Allpool on näide, kuidas on kirjutatud *Item* klassi andmete pärimismeetod.

```
/// <summary>
/// Return collection of Resources from data source.
/// </summary>
/// <returns>Collection of Resources.</returns>
/// <response code="200">The Resources were successfully re-
trieved.</response>
/// <response code="401">Not authorized to see the data.
</response>
/// <response code="404">The Resource object type does not ex-
ist.</response>
[HttpGet]
public async Task<ActionResult<IEnumerable<Item>>> GetItems()
{

    var weeks = await _bll.Items.GetAllAsync();

    return Ok(weeks.Select(ItemMapper.MapGet) );

}
```

Joonis 11. GET päringu koodinäide.

POST

Andmete sisestamiseks kasutatakse POST meetodit. Põhjus, miks andmete sisestamiseks on parem kasutada GET asemel POST päringut, seisneb selles, et veebibrauserid logivad tavaliselt kogu URL-i lihttekstina, mis teeb tundliku teabe saatmise GET päringuga

ebaturvaliseks [42]. Allpool on näide, kuidas käib rakenduses klassi *Item* uue kirje lisamine ja lisatud objekti tagastamine.

```
/// <summary>
/// Create and return new Resource object based on input
Resource DTO.
/// </summary>
/// <param name="createEntity">Resource DTO with properties to
create new Resource object.</param>
/// <returns>Resource object created.</returns>
/// <response code="201">The Resource was successfully
created.</response>
/// <response code="400">Data is not valid to create a new
Resource.</response>
/// <response code="401">Not authorized to see the
data.</response>
/// <response code="404">The Resource object type does not
exist.</response>
[HttpPost]
public async Task<ActionResult<Item>> PostItem(ItemAdd
createEntity)
{
    var bllItem = ItemMapper.MapAdd(createEntity);
    var addedItem = await
_bll.Items.AddItemWithFacilities(bllItem);
    await _bll.SaveChangesAsync();
    var returnItem = ItemMapper.MapGet(addedItem);
    return CreatedAtAction("GetItem", new { id = returnItem!.Id
}, returnItem);
}
```

Joonis 12. POST päringu koodinäide.

PUT

Andmete muutmiseks kasutatakse PUT päringut. POST ja PUT päringu erinevus seisneb selles, et PUT päringud on idempotentsed. See tähendab, et sama PUT päringu väljakutsumine annab alati sama tulemuse. Seevastu POST päringu korduval väljakutsumisel luuakse sama ressurssi mitu korda. [42]

Allpool on toodud näide, kuidas toimub klassi *Item* kirje muutmine.

```
/// <summary>

/// Find Resource by id from data source and update Resource
/// </summary>
/// <param name="id">Resource id - int.</param>
/// <param name="editEntity">Resource DTO with updated
properties.</param>
/// <returns>No content.</returns>
/// <response code="204">The Resource was successfully found and
updated.</response>
/// <response code="400">The Resource with specified id does not
exist.</response>
/// <response code="401">Not authorized to see the
data.</response>
[HttpPut("{id}")]
public async Task<IActionResult> PutItem(Guid id, ItemEdit
editEntity)
{
    if (id != editEntity.Id){ return BadRequest(); }

    var entity = await _bll.Items.FirstOrDefaultAsync(id);
    if (entity == null)
    {
        return BadRequest();
    }
    var editedEntity = ItemMapper.MapEdit(editEntity);
    var facSelected = editEntity.Facilities?.Select(x =>
x.FacilityId).ToList();
    _bll.Items.UpdateItem(editedEntity, facSelected);
    await _bll.SaveChangesAsync();

    try
    {
        await _bll.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!await _bll.Items.ExistsAsync(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }
    return NoContent();
}
```

Joonis 13. PUT päringu koodinäide.

DELETE

Viimaseks tegevuseks on kirjete kustutamine, selleks on kasutatud DELETE päringut. Allpool on näide, kuidas toimub klassi *Item* kirje kustutamine läbi DELETE päringu.

```
/// <summary>
/// Delete and return resource based on id.
/// </summary>
/// <param name="id">Resource id - int.</param>
/// <returns>Deleted Resource object.</returns>
///   <response code="200">The Resource was successfully
deleted.</response>
///   <response code="401">Not authorized to see the
data.</response>
/// <response code="404">The Resource object or Resource with
specified id does not exist.</response>
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteItem(Guid id)
{
    await _bll.Items.RemoveAsync(id);
    await _bll.SaveChangesAsync();

    return NoContent();
}
```

Joonis 14. DELETE päringu koodinäide.

Koodinäidistel on näha, kuidas BLL kihi kasutamine on teinud kontrollerite koodi loetavamaks ja hallatavamaks, jättes loogikaarvutused kontrolleritest välja.

REST API kontrollerite arendamisel on testimiseks kasutatud Postman-i (versioon 3.0). Postman on skaleeritav API testimise tööriist, mis sai alguse 2012. aastal Abhinav Asthana kõrvalprojektina, et lihtsustada API töövoogu testimisel ja arendamisel [43].

4.2.3 REST API kontrollerite turvalisus

Andmete pärimiseks REST API kontrolleritest on tähtis ligipääsu kontrollimine ehk autentimine. Käesolevas veebirakenduses on autentimiseks kasutatud JWT (*JSON Web Token*). JWT on avatud standard, mis võimaldab osapoolte vahel andmeid edastada JSON – objektina turvalisel ja kompaktsel viisil. Andmete edastamisel JWT abiga on andmed digitaalselt allkirjastatud, et neid oleks lihtne kontrollida ja usaldada. [44]

JWT-ga autentimiseks on projekti lisatud vajalik konfiguratsioon, kasutades meetodit „*AddAuthentication()“*.Allpool on näidatud, kuidas on rakenduses *Startup* klassis JWT konfigureeritud.

```
services
    .AddAuthentication()
    .AddJwtBearer(options =>
        {
            options.SaveToken = true;
            options.TokenValidationParameters = new
TokenValidationParameters()
            {
                ValidIssuer = Configuration["JWT:Issuer"],
                ValidAudience =  Configuration["JWT:Issuer"],

                IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(
Configuration["JWT:Key"])),
                ClockSkew = TimeSpan.Zero
            };
        }
    );
```

Joonis 15. JWT konfigureerimise koodinäide.

JWT genereerimiseks on loodud REST API kontroller nimega *AccountController*, kus meetodis Login korrektse kasutajanime ja parooli sisestamisel genereeritakse JWT ning tagastatakse see. Rakenduse REST API kontrollerites on autentimine sisse lülitatud, kasutades vastavat annotatsiooni (vaata joonis 16). Tehes päringu ilma korrektse JWT-ta, tagastatakse HTTP viga koodiga 401 (*UnAuthorizedAccess*).

```
[Authorize(AuthenticationSchemes =
JwtBearerDefaults.AuthenticationScheme)]
```

Joonis 16. REST kontrolleri autentimise annotatsioon.

4.2.4 Kasutatud paketid

Allpool on nimekiri kasutatud pakettidest, mille autor rakenduse arendamiseks NuGet-st alla laadis.

- Microsoft.EntityFrameworkCore – pakett on kaasaegne objekt – andmebaasi kaardistaja (*object – database mapper)* .NET rakenduste jaoks. See toetab LINQ avaldise päringuid, muudatuste jälgimist ja andmebaasi skeemi migreerimist. [45]

- Microsoft.EntityFrameworkCore.Tools – andmebaasi migreerimiseks vajaminevate käsurea käskude pakett. [46]

- Microsoft.EntityFrameworkCore.SqlServer – Microsoft SQL Serveri andmebaasi kasutamise pakett. [47]

- Microsoft.AspNetCore.Authentication.JwtBearer – ASP.NET Core vahevara, mis võimaldab rakendusel vastu võtta OpenID Connect kandja märgi (*bearer token*). [48]

- Microsoft.AspNetCore.Identity.EntityFrameWorkCore – ASP.NET Core Identity pakkuja, mis kasutab Entity Framework Core-i. [49]

- Microsoft.AspNetCore.Diagnostics.EntityFrameWorkCore – pakett, mis sisaldab tüüpe, mis on seotud ASP.NET Core Entity Framework Core diagnostika hõivamise ja aruandlusega. [50]

- Microsoft.Extensions.Identity.Stores – pakkett, mis võimaldab lisada ASP.NET Core rakendusse sisselogimisfunktsioone ja muudab sisselogitud kasutaja andmete kohandamise lihtsaks. [51]

- Microsoft.VisualStudio.Web.Codegeneration.Design – koodi genereerimise tööriist ASP.NET Core jaoks. Sisaldab käsku dotnet-aspnet.codegenerator, mida kasutatakse kontrollerite ja vaadete genereerimiseks. [52]

- AutoMapper – pakett, mis lubab lihtsalt objekt - objekt kaardistamist (*object – object mapping*). [53]

4.3 Esirakenduse arendus

Esirakendus on kasutajaliides, mida ettevõtte Structo Group OÜ töötajad hakkavad kasutama ning mis on loodud tagarakenduse peale. Esirakenduse analüüsi peatükis valis töö autor esirakenduse programmeerimiskeeleks JavaScripti ning raamistikuks Reacti. Analüüsis tõi autor välja ka, et kasutab tavalise JavaScripti asemel TypeScripti parema arendamiskogemuse saamiseks. Kliendiliidese disainimiseks ja kohandamiseks on kasutatud Tailwind raamistikku, mis koosneb klassidest, mis aitavad stiililehte üleküllastamata veebilehe värve, vahekaugusi, tüpograafiat, varje ja kõike muud muuta [54].

4.3.1 Esirakenduse struktuur

Alustades uut React rakendust, tuleb käsireal kasutada käsku npx create-react-app <rakenduse nimi>, mis genereerib rakenduse kausta vajalikud failid ja teegid. Rakenduse kausta tekivad kaustad ja failid:

- node_modules – projekti kuuluvate välisteekide kaust;
- App.tsx – eesrakenduse põhikomponent, kus toimub teiste komponentide kuvamine;
- index.tsx – rakendusse sisenemispunkt;
- package-lock.json – fail, mille eesmärk on jälgida iga installitud paketi täpset versiooni, et projekt oleks 100% samal viisil reprodutseeritav; [55]
- package.json – projekti manifest, mis on hoidlaks tööriistade konfiguratsioonile, projekti nimele ja versioonile ning läbi npm käskluse installitud pakettide nimedele ja versioonidele; [56]
- tsconfig.json – fail võimaldab määrata juurtaseme failid ja kompilaatori suvandid, mida on vaja TypeScripti kompileerimiseks. Faili olemasolu määrab, et nimetatud kataloog on TypeScripti projekti juur. [57]

Põhiosa rakenduse koodist kirjutatakse src kausta, kus asuvad järgmised kaustad:

- assets – CSS failid;

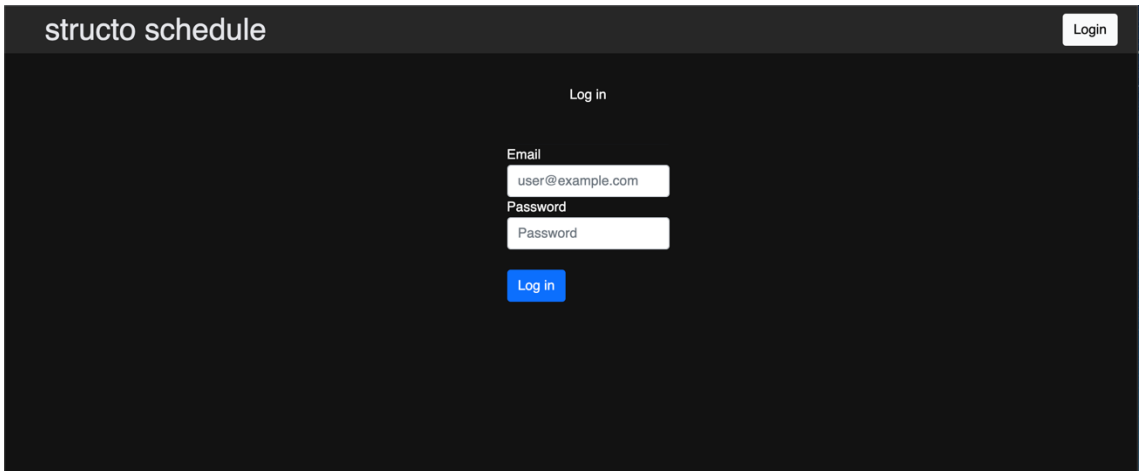
- components – kaustas hoitakse lehtede vahel jagatavaid komponente, näiteks *Modal* ja *Header;*
- containers – sisaldab veebilehe toimimiseks ja kuvamiseks vajaminevaid komponente;
- context – rakenduse olekut sisaldav kaust;
- dto – rakenduse andme-edastusobjekte sisaldav kaust;
- services – sisaldab teenuseid, mida kasutajaliides kasutab tagarakendusega suhtlemiseks;
- types – sisaldab päringute tegemisel vastuse saamiseks vajaminevaid liideseid.

Reactiga koos on kasutatud JSX (JavaScript XML), mis lubab HTML elemente kirjutada ilma *createElement()* meetodita. JSX kasutamiseks on tsconfig.json failis määratud „jsx“ väärtuseks „react-jsx“. Kasutades JSX ja TypeScripti koos, tuleb React komponentide faililaiendiks .tsx. [58]

React rakendus saab alguse index.tsx failist, kus renderdatakse komponent App.tsx. App.tsx komponent on rakenduse põhikomponent, mis on hoidlaks rakenduse olekule kui ka teistele rakenduses kasutavatele komponentidele [59]. App.tsx failis on kasutusel teek React Router, mis võimaldab navigeerimise erinevate komponentide vahel React rakenduses, hoides brauseri URL-i muutmisel kasutajaliidest URL-ga sünkroonis.

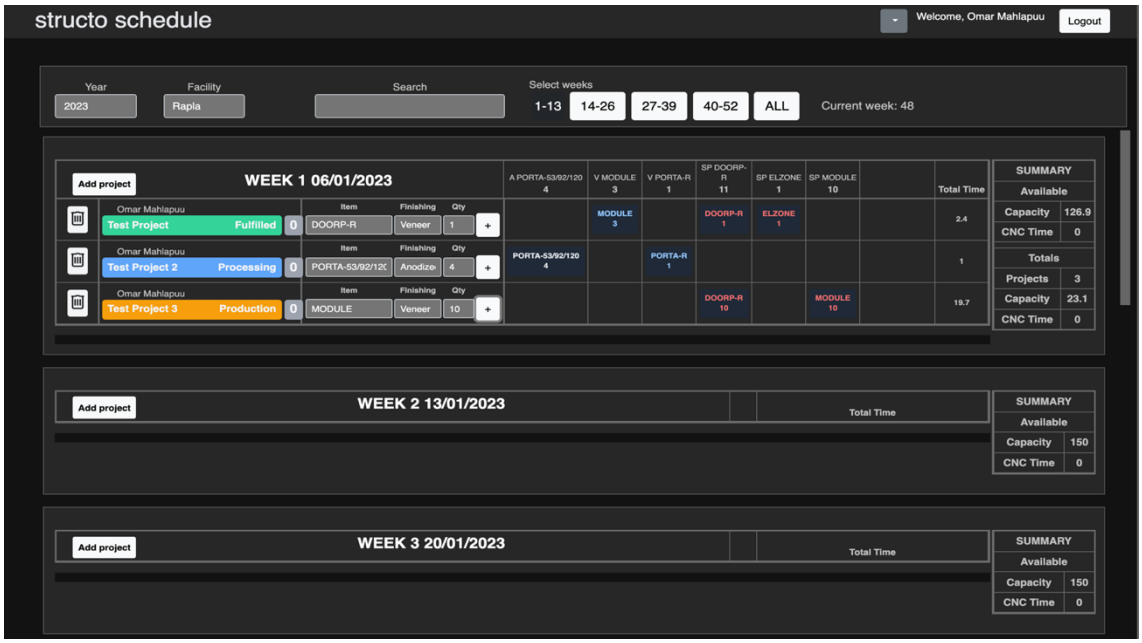
4.3.2 Kasutajaliidese kasutamine

Korrektsele veebilehele pöördumisel kuvatakse rakendusse sisselogimisleht, kus on kasutajanime – ja parooliväli ning sisselogimisnupp (Joonis 17).

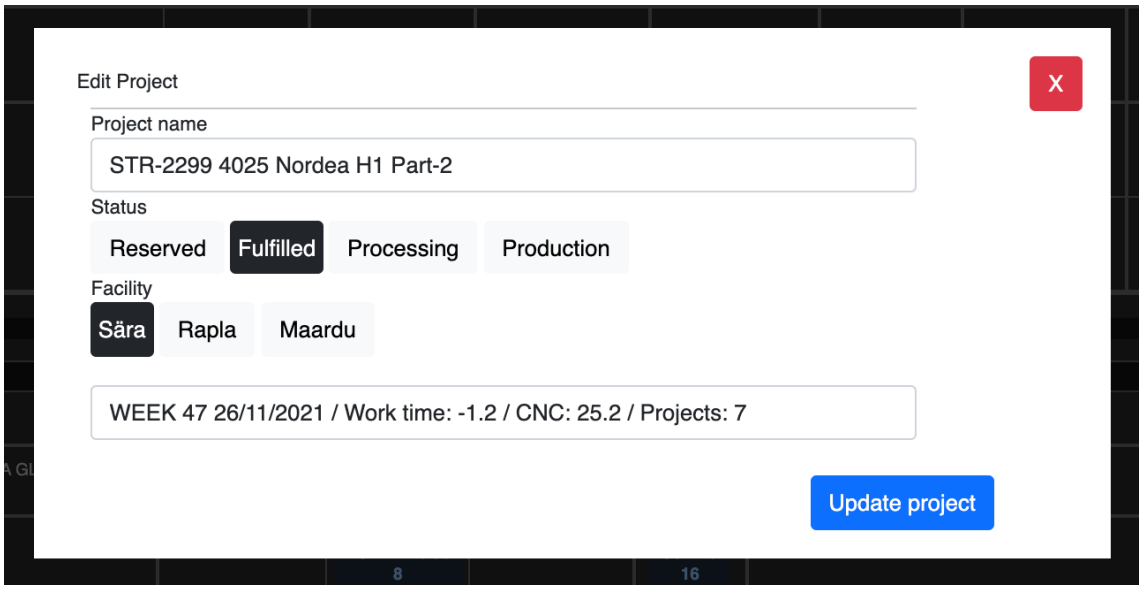


Joonis 17. Sisselogimisleht.

Õnnestunud sisselogimisel suunatakse kasutaja projektide planeerimistabeli lehele, kus on kuvatud käimasoleva kvartali projektid (Joonis 18). Planeerimistabeli lehel on võimalik filtreerida tulemusi muutes aastat, tehast ning kvartalit. Samuti on võimalik ka projekte otsida, kasutades otsinguriba. Lihtsaks kasutamiseks toimub kogu põhitöö ühel lehel. Uue projekti lisamine käib nädala komponendis asuva nupuga „*Add project*“. Uuele projektile saab lisada projektiga seotud tooted projekti kõrval asuvatel väljadel, vajutades nuppu „+“. Projektile toote lisamisel on vaja valida nii toote nimetus, viimistlustüüp kui ka kogus. Pärast korrektset sisestamist ilmub kirje automaatselt uuenevasse tabelisse ning nädalaga ja projektiga seotud info uueneb. Nädala komponendi paremas osas on välja toodud valitud tehase, nädalaga seotud vaba töö maht ja – CNC aeg. Projektide ja toodete info muutmiseks on kasutatud *Modal* komponenti, mis avaneb eraldi aknas (Joonis 19).

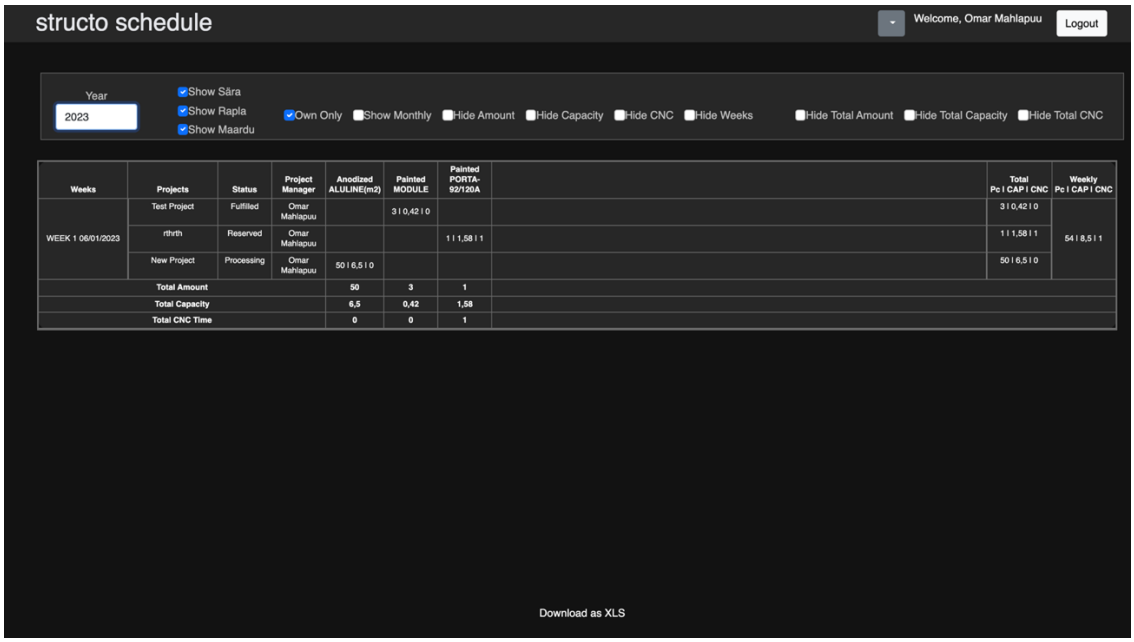


Joonis 18. Projektide planeerimise lehekülg.



Joonis 19. Modal näide.

Veebilehe ülaosa rippmenüü kaudu saab minna rakenduse sätetesse, kus administraatorina on võimalik lisada, muuta või kustutada tooteid, tehaseid, aastaid ning kasutada kasutajatega seotud funktsionaalsusi. Tähtsaks nõudeks rakenduse loomisel oli kokkuvõtete tegemine. Valides rippmenüüst „*Summary*“ suunatakse kasutaja analüüsi lehele (Joonis 20).



Joonis 20. Kokkuvõtete tegemise veebilehekülg.

Analüüsi lehel kuvatakse vaikimisi kõik sisestatud tooted projektide kaupa. Kasutaja saab filtreerida tulemusi, muutes aastat, valides vajalikud tehased ning valides „näita ainult enda projekte“, märkides linnukese „*Own Only*“ lahtrisse. Kasutaja saab tabeli kuju

muuta „projektide kaupa“ olekust „kuude kaupa“ olekuks, märkides linnukese lahtrisse „*Show Monthly*“. Juhul kui kasutaja ei soovi kõiki toodetega seotud väärtusi näha, saab ta need soovi korral peita, kasutades *Hide* lahtreid. Vajadusel saab kasutaja laadida alla tabeli Exceli failina vajutades nupule „*Download as XLS*“, näiteks juhul kui projektijuht soovib saata tehasele eelinfo planeeritavatest projektidest.

5 Tulemused

Lõputöö tulemusena valmis projektide planeerimise veebirakendus, millega saab aegasäästvalt ja lihtsalt projekte planeerida, vältides vigu, mis võivad tekkida andmete tabeli kujul hoidmisel. Rakenduses on kasutajatel oma konto, millega saab sisse logida ning projektide planeerimisega tegeleda. Rakenduses on nähtaval kõik vajalikud parameetrid tulemusliku ja ladusa töö tegemiseks. Rakendust kasutavad kõige enam Structo Group OÜ projekti– ja müügijuhid, kes tegelevad uute sissekannete tegemisega, mis on vajalikuks infoks tehastele, laotöötajatele ning konstruktoritele. Rakenduse nõuded said vastavalt epikutele ja kasutajalugudele edukalt täidetud ning vajalikud funktsionaalsused toimivad.

Valminud rakendus täidab kõik talle seatud eesmärgid, kuid arutades ettevõtte juhi ja töötajatega on juba plaani võetud aastal 2022 rakendusele lisasid ja uuendusi. Üheks uuenduseks on rakendusele lisandumas võimalus automaatselt importida projekte teisest rakendusest.

Autor kavatseb tegeleda veel rakenduse disainiga, muutes rakendust seadmetundlikumaks, et kasutus oleks mugavam ka mobiilis ja tahvelarvutis.

6 Kokkuvõte

Bakalaureusetöö eesmärk oli luua ettevõttele Structo Group OÜ projektide planeerimise veebirakendus, mis asendaks aastaid kasutusel olnud Microsoft Exceli tabelid, kiirendaks tööprotsessi ning oleks turvaline, mugav ja lihtne kasutada. Structo Group OÜ on kiiresti kasvav ja kaasaegne Eesti ettevõte, mis vajas projektide planeerimiseks olemasolevast töökindlamat töövahendit. Suheldes ettevõtte juhiga, projekti – ja müügijuhtidega ning laotöötajate ja komplekteerijatega selgitas töö autor välja loodava veebirakenduse nõuded ja koostas epikud ja kasutajalood.

Töös võrreldi erinevaid programmeerimiskeeli ja raamistikke ning valiti välja sobilik nii tagarakendusele kui ka esirakendusele. Analüüsi käigus valiti andmebaasisüsteemiks MS SQL Server, integreeritud arenduskeskkonnaks JetBrains Rider, koodihalduskeskkonnaks Git ja rakenduse majutuskeskkonnaks Azure. Veebirakendus loodi hajussüsteemina ning mitmekihilise arhitektuuriga. Tagarakendus arendati C# programmeerimiskeelega, kasutades .NET Core raamistikku. Tagarakendus jagati mitmesse kihti: domeenikiht, andmetele juurdepääsu kiht, äriloogika kiht ja REST API. Töös toodi välja mitmeid koodinäiteid tagarakenduse loomisest. Ettevõtte töötajatele loodi tagarakenduse REST API peale kasutajaliides, kasutades TypeScripti koos React.js raamistikuga.

Rakenduse kasutajate tagasisidest on selgunud, et loodud veebirakendus on end Structo Group OÜ igapäevatöös õigustanud - ettevõtte tööprotsess on muutunud märgatavalt kiiremaks, turvaliselt hallatavaks ja selgelt ülevaatlikumaks. Töötajad harjusid rakendust kiiresti kasutama. Töötajatele on enim kasu rakenduses projektide aja- ja töömahukuse arvutamise automatiseerimisest. Uudne ja praktiline rakendus on muutnud ettevõttes töötavate projekti – ja müügijuhtide igapäevase projekte puudutava tööprotsessi mugavamaks ja kiiremaks, millest saavad seega samaaegselt kasu nii töötajad kui ka ettevõte. Uudse töövahendi abil on töötajatel võimalik teha tõhusamalt ja kvaliteetsemat tööd, mis tõstab kindlasti ettevõtte väärtust. Autori hinnangul on loodud rakendus seega täitnud kõik töö sissejuhatuses püstitatud eesmärgid - ettevõtte projektide planeerimine

on nüüd kiirem, mugavam ja turvalisem ning ka kokkuvõtete tegemine lõpetatud projektidest lihtsam.

Rakendus loodi küll kindlale ettevõttele, kuid seda on võimalik vajadusel ja soovi korral ka teiste ettevõtete vajadustele kohandada. Selleks tuleks rakendusele juurde luua võimalus, lisada andmebaasi uusi ettevõtteid ning lisada ettevõtete konfigureerimisvõimalus.

7 Kasutatud kirjandus

[1] C. Academy, „Back-End Web Architecture,“ [Võrgumaterjal]. Loetud aadressil: https://www.codecademy.com/articles/back-end-architecture.

[2] „Top 7 Programming Languages for Backend Web Development,“ 27 04 2021. [Võrgumaterjal]. Loetud aadressil: https://www.geeksforgeeks.org/top-7-programming-languages-for-backend-web-development/. [Kasutatud 03 12 2021].

[3] T. DeGroat, „The History of JavaScript: Everything You Need to Know,“ [Võrgumaterjal]. Loetud aadressil: https://www.springboard.com/blog/data-science/history-of-javascript/. [Kasutatud 03 12 2021].

[4] P. Giamperdraglia, „Why we choose Python as a backend language,“ 23 11 2019. [Võrgumaterjal]. Loetud aadressil: https://www.asapdevelopers.com/python-backend-language/. [Kasutatud 03 12 2021].

[5] J. Hartman, „What is Java? Definition, Meaning & Features of Java Platforms,“ 7 10 2021. [Võrgumaterjal]. Loetud aadressil: https://www.guru99.com/java-platform.html. [Kasutatud 03 12 2021].

[6] JavaTPoint, „History of Java,“ [Võrgumaterjal]. Loetud aadressil: https://www.javatpoint.com/history-of-java. [Kasutatud 03 12 2021].

[7] L. School, „A Brief History of Ruby,“ [Võrgumaterjal]. Loetud aadressil: https://launchschool.com/books/ruby/read/introduction. [Kasutatud 03 12 2021].

[8] A. Osypenko, „When and Why You Should Go with Go(lang) for Backend Development,“ 28 04 2021. [Võrgumaterjal]. Loetud aadressil: https://madappgang.com/blog/backend-development-with-golang/. [Kasutatud 03 12 2021].

[9] M. Chand, „What Is C#,“ 07 03 2020. [Võrgumaterjal]. Loetud aadressil: https://www.c-sharpcorner.com/article/what-is-c-sharp/. [Kasutatud 03 12 2021].

[10] S. Veeraraghavan, „12 Best Programming Languages to Learn in 2022,“ 30 11 2021. [Võrgumaterjal]. Loetud aadressil: https://www.simplilearn.com/best-programming-languages-start-learning-today-article. [Kasutatud 03 12 2021].

[11] V. Singh, „What is a Framework? [Definition] Types of Frameworks,“ 15 05 2021. [Võrgumaterjal]. Loetud aadressil: https://hackr.io/blog/what-is-frameworks. [Kasutatud 03 12 2021].

[12] B. Scout, „.NET FRAMEWORK – HISTORY AND ADVANTAGES,“ [Võrgumaterjal]. Loetud aadressil: https://bytescout.com/blog/2014/06/net-framework.html. [Kasutatud 03 12 2021].

[13] L. C. Gurus, „Microsoft Announces End of .NET Framework & .NET Core: Meet .NET 5,“ 05 07 2019. [Võrgumaterjal]. Loetud aadressil: https://www.iowacomputergurus.com/insights/article/microsoft-announces-end-of-net-framework-net-core-meet-net-5. [Kasutatud 03 12 2021].

[14] Microsoft, „.NET and .NET Core Support Policy,“ [Võrgumaterjal]. Loetud aadressil: https://dotnet.microsoft.com/platform/support/policy/dotnet-core. [Kasutatud 03 12 2021].

[15] T. Teacher, „.NET Core Overview,“ [Võrgumaterjal]. Loetud aadressil: https://www.tutorialsteacher.com/core/dotnet-core. [Kasutatud 03 12 2021].

[16] K. Schroeder ja K. Sugiura, „What Is JavaScript Used for in Front-end Web Development?,“ 07 11 2018. [Võrgumaterjal]. Loetud aadressil: https://www.hilemangroup.com/Thought-Leadership/Hilelights-Blog/JavaScript-and-Front-end-Development. [Kasutatud 03 12 2021].

[17] S. Learn, „What is React: Definition, Why ReactJS, its Features and Installation,“ 15 11 2021. [Võrgumaterjal]. Loetud aadressil: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs. [Kasutatud 03 12 2021].

[18] S. Azam, „What is Vue.js, and Why is it Cool?,“ 2020. [Võrgumaterjal]. Loetud aadressil: https://linuxhint.com/about_vue_js/. [Kasutatud 03 12 2021].

[19] H. Emekoma, „Comparing the best new JavaScript frameworks to React,“ 2 03 2021. [Võrgumaterjal]. Loetud aadressil: https://blog.logrocket.com/comparing-the-best-new-javascript-frameworks-to-react/. [Kasutatud 03 12 2021].

[20] S. Daityari, „Angular vs React vs Vue: Which Framework to Choose in 2021,“ 15 03 2021. [Võrgumaterjal]. Loetud aadressil: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/. [Kasutatud 03 12 2021].

[21] A. Panwar, „ Types of Database Management Systems,“ 09 06 2021. [Võrgumaterjal]. Loetud aadressil: https://www.c-sharpcorner.com/UploadFile/65fc13/types-of-database-management-systems/. [Kasutatud 03 12 2021].

[22] J. Carder, „ What Is SQL Database?,“ 08 07 2020. [Võrgumaterjal]. Loetud aadressil: https://www.openlogic.com/blog/what-sql-database. [Kasutatud 03 12 2021].

[23] J. Romanowski, „The Most Popular Databases in 2020,“ 18 08 2020. [Võrgumaterjal]. Loetud aadressil: https://learnsql.com/blog/most-popular-sql-databases-2020/. [Kasutatud 03 12 2021].

[24] O. Olakunle, „10 Best .NET IDE,“ 21 04 2020. [Võrgumaterjal]. Loetud aadressil: https://www.dunebook.com/best-asp-net-ide/. [Kasutatud 03 12 2021].

[25] R. Peres, „Visual Studi versus Rider,“ 12 03 2018. [Võrgumaterjal]. Loetud aadressil: https://stackify.com/visual-studio-rider/. [Kasutatud 03 12 2021].

[26] C. Richardson, „ How to Choose the Right Version Control Software,“ 11 09 2019. [Võrgumaterjal]. Loetud aadressil: https://cdn2.hubspot.net/hubfs/365/Choosing%20Your%20VCS.pdf. [Kasutatud 03 12 2021].

[27] H. Mehta, „What Is GUID In C#?,“ 26 05 2021. [Võrgumaterjal]. Loetud aadressil: https://www.c-sharpcorner.com/article/what-is-guid-in-c-sharp/. [Kasutatud 03 12 2021].

[28] L. Fernigrini, „How To Choose a Good Primary Key,“ 08 07 2021. [Võrgumaterjal]. Loetud aadressil: https://vertabelo.com/blog/primary-key/. [Kasutatud 03 12 2021].

[29] H. Ak, „A Multi-Layer Back-End Application Architecture in .NET Core,“ 30 08 2019. [Võrgumaterjal]. Loetud aadressil: https://medium.com/@hamzaak/a-multi-layer-back-end-application-architecture-in-net-core-c08898f2427e. [Kasutatud 03 12 2021].

[30] C. S. Blogger, „Three Tier System Architecture For Business Applications,“ Code Authority, 07 05 2020. [Võrgumaterjal]. Loetud aadressil:

https://www.codeauthority.com/Blog/Entry/three-tier-architecture. [Kasutatud 03 12 2021].

[31] Microsoft, 28 10 2021. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli. [Kasutatud 03 12 2021].

[32] Plagimtech, 2020. [Võrgumaterjal]. Loetud aadressil: https://www.pragimtech.com/blog/blazor/asp.net-core-rest-api-dbcontext/. [Kasutatud 03 12 2021].

[33] D. IQ, „Repository Pattern," [Võrgumaterjal]. Loetud aadressil: https://deviq.com/design-patterns/repository-pattern. [Kasutatud 03 12 2021].

[34] Microsoft, 25 05 2021. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/ef/core/. [Kasutatud 03 12 2021].

[35] G. Cuofano, „What Is A Bussiness Logic Layer," [Võrgumaterjal]. Loetud aadressil: https://fourweekmba.com/business-logic-layer/. [Kasutatud 03 12 2021].

[36] M. Fowler, „Unit of Work," [Võrgumaterjal]. Loetud aadressil: https://martinfowler.com/eaaCatalog/unitOfWork.html. [Kasutatud 03 12 2021].

[37] D. Esposito, „Cutting Edge - Pros and Cons of Data Transfer Objects," 08 2021. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/august/pros-and-cons-of-data-transfer-objects. [Kasutatud 03 12 2021].

[38] Wikipedia, „Fluent interface," 11 11 2021. [Võrgumaterjal]. Loetud aadressil: https://en.wikipedia.org/wiki/Fluent_interface#cite_note-fowler2005-1. [Kasutatud 03 12 2021].

[39] D. Dwij, „LINQ In C#," 06 03 2020. [Võrgumaterjal]. Loetud aadressil: https://www.c-sharpcorner.com/UploadFile/72d20e/concept-of-linq-with-C-Sharp/. [Kasutatud 03 12 2021].

[40] R. Hat, „What is a REST API?," 08 05 2020. [Võrgumaterjal]. Loetud aadressil: https://www.redhat.com/en/topics/api/what-is-a-rest-api. [Kasutatud 03 12 2021].

[41] R. Anderson, „dotnet-aspnet-codegenerator," 26 05 2021. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/tools/dotnet-aspnet-codegenerator?view=aspnetcore-6.0. [Kasutatud 03 12 2021].

[42] W3Schools, „HTTP Request Methods," [Võrgumaterjal]. Loetud aadressil: https://www.w3schools.com/tags/ref_httpmethods.asp. [Kasutatud 03 12 2021].

[43] T. Hamilton, „What is Postman?," 29 11 2021. [Võrgumaterjal]. Loetud aadressil: https://www.guru99.com/postman-tutorial.html. [Kasutatud 03 12 2021].

[44] J. Trivedi, „JWT Authentication In ASP.NET Core," 06 05 2020. [Võrgumaterjal]. Loetud aadressil: https://www.c-sharpcorner.com/article/jwt-json-web-token-authentication-in-asp-net-core/. [Kasutatud 03 12 2021].

[45] NuGet, „Microsoft.EntityFramework.Core," [Võrgumaterjal]. Loetud aadressil: https://www.nuget.org/packages/Microsoft.EntityFrameworkCore. [Kasutatud 03 12 2021].

[46] NuGet, „Microsoft.EntityFrameworkCore.Tools," [Võrgumaterjal]. Loetud aadressil: https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Tools/. [Kasutatud 03 12 2021].

[47] NuGet, „Microsoft.EntityFrameworkCore.SqlServer,“ [Võrgumaterjal]. Loetud aadressil: https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.SqlServer/. [Kasutatud 03 12 2021].

[48] NuGet, „Microsfot.AspNetCore.Authentication.JwtBearer,“ [Võrgumaterjal]. Loetud aadressil: https://www.nuget.org/packages/Microsoft.AspNetCore.Authentication.JwtBearer/. [Kasutatud 03 12 2021].

[49] NuGet, „Microsoft.AspNetCore.Identity.EntityFrameworkCore,“ [Võrgumaterjal]. Loetud aadressil: https://www.nuget.org/packages/Microsoft.AspNetCore.Identity.EntityFrameworkCore/. [Kasutatud 03 12 2021].

[50] NuGet, „Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore,“ [Võrgumaterjal]. Loetud aadressil: https://www.nuget.org/packages/Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore/. [Kasutatud 13 12 2021].

[51] NuGet, „Microsoft.Extensions.Identity.Stores,“ [Võrgumaterjal]. Loetud aadressil: https://www.nuget.org/packages/Microsoft.Extensions.Identity.Stores/. [Kasutatud 03 12 2021].

[52] NuGet, „Microsoft.VisualStudio.Web.CodeGeneration.Design,“ [Võrgumaterjal]. Loetud aadressil: https://www.nuget.org/packages/Microsoft.VisualStudio.Web.CodeGeneration.Design/. [Kasutatud 03 12 2021].

[53] NuGet, „AutoMapper,“ [Võrgumaterjal]. Loetud aadressil: https://www.nuget.org/packages/AutoMapper/. [Kasutatud 03 12 2021].

[54] Tailwind, „An API for your design system.,“ [Võrgumaterjal]. Loetud aadressil: https://tailwindcss.com. [Kasutatud 03 12 2021].

[55] Node, „The package-lock.json file,“ [Võrgumaterjal]. Loetud aadressil: https://nodejs.dev/learn/the-package-lock-json-file. [Kasutatud 03 12 2021].

[56] Node, „The package.json guide,“ [Võrgumaterjal]. Loetud aadressil: https://nodejs.dev/learn/the-package-json-guide. [Kasutatud 03 12 2021].

[57] K. Chowdhury, „What Is the tsconfig.json Configuration File?,“ 03 06 2018. [Võrgumaterjal]. Loetud aadressil: https://dzone.com/articles/what-is-the-tsconfigjson-configuration-file. [Kasutatud 03 12 2021].

[58] W3Schools, „React JSX,“ [Võrgumaterjal]. Loetud aadressil: https://www.w3schools.com/react/react_jsx.asp. [Kasutatud 03 12 2021].

[59] A. Sridhar, „A quick guide to help you understand and create ReactJS apps,“ 18 08 2018. [Võrgumaterjal]. Loetud aadressil: https://www.freecodecamp.org/news/quick-guide-to-understanding-and-creating-reactjs-apps-8457ee8f7123/. [Kasutatud 03 12 2021].

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

Mina, Omar Mahapuu

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose Projektide planeerimise veebirakenduse arendamine tootmisettevõtte näitel, mille juhendaja on Kristiina Hakk:
   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.

06.01.2022

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 – Kasutajalood ja epikud

Epik 1: Kasutajana tahan näha projektide planeerimisgraafikut, kus arusaadavalt kirjas projektiga seotud tooted ja muu info.

- K1: Kasutajana tahan näha olemasolevate projektide kogust nädalas koos projektidega seotud toodetega ja ajakuluga, et enda projekte planeerida.
- K2: Kasutajana soovin ma filtreerida tulemusi aasta ja kvartali järgi, et kuvada vaid mulle tähtsad tulemused.
- K3: Kasutajana tahan ma määrata, millise tehase projekte kuvatakse, et saada ülevaadet kindla tehase kohta.
- K4: Kasutajana tahan ma näha nädalas vaba töö – ja CNC - võimekust, et mugavalt arvestust pidada.
- K5: Kasutajana tahan ma näha kõikide projektide olekuid, et saada ülevaadet projektide seisust.
- K6: Kasutajana tahan ma näha projektidele lisatud kommentaare, et olla kursis projekti isepärasustega.

Epik 2: Kasutajana soovin projektide planeerimisgraafikus projekte lisada ja muuta.

- K1: Kasutajana tahan ma lisada valitud nädalasse uut projekti, et alustada planeerimisega.
- K2: Kasutajana tahan ma lisada projekti tooteid, et saada ülevaadet projekti ajakulust.
- K3: Kasutajana soovin ma muuta projekti olekut, et teistele ja endale paremat ülevaadet anda.
- K4: Kasutajana tahan ma tõsta projekti teise nädalasse, kuna võib tekkida parem aeg projekti teostamiseks.
- K5: Kasutajana tahan ma muuta projekti tooteid, et uuendada projekti ajakulu.

- K6: Kasutajana tahan ma täpsustada lisatava toote viimistlust ja kogust, et arvutada täpsem ajakulu.

Epik 3: Kasutajana tahan teha projektidest tabelikujulisi ülevaatlikke kokkuvõtteid.

- K1: Kasutajana tahan ma kokkuvõtet kõikidest aasta projektidest, et saada head ülevaadet.
- K2: Kasutajana tahan ma võimalust näha vaid enda projekte, et saada ülevaadet enda projektidest.
- K3: Kasutajana tahan ma võimalust filtreerida erinevate tehaste projekte, et kindlatele tehastele graafikuid edastada.
- K4: Kasutajana tahan ma võimalust määrata, milliseid toodete parameetreid kokkuvõttes kuvatakse, et saada ülevaadet ainult vajalikust.
- K5: Kasutajana tahan ma võimalust kokkuvõtet alla laadida .xls formaadis, et seda tehastele edasi saata.
- K6: Kasutajana tahan ma võimalust muuta kokkuvõtte projektipõhisest vaatest kuupõhiseks vaateks ja vastupidi, et saada vajalikku ülevaadet.

Epik 4: Administraatorina tahan võimalust hallata tooteid, töötajaid, viimistlustüüpe, tehaseid ja olekuid.

- K1: Administraatorina tahan ma lisada uusi tooteid, et kasutajad saaksid neid projektidesse lisada.
- K2: Administraatorina tahan ma määrata, millises tehases tooteid valmistada saab, et kasutajad saaksid õigeid valikuid teha.
- K3: Administraatorina tahan ma määrata erinevate viimistlustega toodete ajakulu ja CNC tööaega, et projektide planeerimise arvestus oleks korrektne.
- K4: Administraatorina tahan ma lisada, muuta ja kustutada tehaseid, viimistlustüüpe, olekuid ja töötajaid, et rakendus oleks ajakohane.

Epik 5: Administraatorina tahan hallata teisi kasutajaid.

- K1: Administraatorina tahan ma lähtestada kasutajate paroole, et kasutaja saaks parooli unustamisel rakendusse sisse logida.
- K2: Administraatorina tahan ma kustutada kasutajaid, et eemaldada mitteaktiivsed kasutajad.

Epik 6: Kasutajana tahan ma hallata enda kasutajat.

- K1: Kasutajana tahan ma muuta enda parooli, et kasutaja turvalisena hoida.