TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Ruben Galoyan 185316
Daniel Antonov 179117
Vladislav Jekimtsev 185619

# COLUMBUS EESTI AS KASSAVOO JUHTIMISE LAHENDUSE EDASIARENDAMINE

Bakalaurusetöö

Juhendaja: Tõnn Talpsepp
Doktorikraad

Tallinn 2021

# Autorideklaratsioon

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

Autorid: Ruben Galoyan, Daniel Antonov, Vladislav Jekimtsev

18.05.2021

# Annotatsioon

Columbus Eesti AS on pakkunud luua lahenduse rahavoogude jälgimiseks ning saldo prognoosimiseks. Oleme alustanud kassavoo lahenduse arendamisega ITB1706 aine raames ning jätkasime sellega meeskondliku bakalaureuse projektina. Meeskonnaprojekti raames valmis selle lahenduse prototüüp. Eesmärgiks oli saada valmis produkti, mida ettevõtte saaks pakkuda klientidele kasutamiseks.

Lõpuprojekti tulemuseks on majandustarkvara, mis on integreeritud ettevõtte keskkonda. Lahenduses saab laadida üles andmed ettevõtte rahavoogude kohta ning vaadata saldoprognoosi, kas tabelist või graafikult. Kuna meie lahenduse mõned osad soovis ettevõtte kasutada oma teistes lahendustes, oli üheks nõuetest nende osade universaalsus.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 59 leheküljel, 5 peatükki, 22 joonist ja 3 tabelit.

# Abstract

## Proceeding with development of the Cash Flow Management Solution for Columbus Eesti AS

Columbus Eesti AS has offered us to create a Cash Flow Management solution for monitoring cash flows and forecasting the balance. We have started developing a cash flow solution within the subject ITB1706 and continued it as our team bachelor's thesis. A prototype of solution was completed as a part of university team project. The goal of the current project was to deliver a finished product that the company could offer to their customers for real use.

The result of the final project is business software, that is integrated into the company's environment. The solution allows you to upload data of the company's cash flows and view the balance forecast, either from a table or a graph. As the company wished to use some parts of our solution in its other projects, one of the requirements was the flexibility of developed parts.

The thesis is in Estonian and contains 59 pages of text, 5 chapters, 22 figures and 3 tables.

# Lühendite ja mõistete sõnastik

| | |
|---|---|
| AAA | *Arrange Act Assert* |
| API | *Application Programming Interface* |
| ASP.NET Core | *Microsoft*'i veebirakenduste arendusraamistik. |
| Azure | Pilveplatvorm *Microsoft*'i poolt tehtud pilevplatvorm, mis võimaldab hoida ja arendada projekti andmed hajutatud andmekeskustes asuvates serverites. |
| BI | *Business Intelligence* |
| Covid-19 | Nakkushaigus |
| CRUD | Andmebaasidega töötamisel neli põhifunktsiooni: loomine, lugemine, muutmine, kustutamine. |
| DAL | *Data Access Layer* |
| DAO | *Data Access object* |
| ERP | *Enterprise Resource Planning* |
| HTTP | Andmeedastusprotokoll |
| IDE | *Integrated Development Environment* |
| jQuery | *JavaScripti* funktsioonide komplekt |
| Konfiguratsioon | Columbuse andmebaas, kus saab seadistada tabelit ja visuaali |
| Mock | Võltsobjekt, mis asendab reaalset objekti. |
| MsSQL | *Microsoft SQL Server* |
| MVC | *Model-View-Controller* |
| NuGet | Teekide haldussüsteem *Microsoft*'i arendusplatvormidele, peamiselt *.NET Frameworki* jaoks. |
| ORM | *Object-Relational Mapping* |
| POST | Andmete taotlemise meetod |
| ReportÄpp | Columbuses kasutuses olev majandustarkvara |
| SQL | Andmebaasikeel ehk struktuurpäringukeel |
| SUT | *System Under Test* |
| Tablix | Columbuse poolt arendatud tabel andmete kuvamiseks |
| UX | User Experience |

# Sisukord

# Jooniste loetelu

# Tabelite loetelu

# 1 Sissejuhatus

Antud peatükis antakse lugejale lühiülevaade teostatud lahendusest, selle eesmärgist ning probleemist. Samuti kirjeldatakse üldsõnaliselt loodud funktsionaalsust ning meeskondliku lõputöö edasist ülesehitust.

## 1.1 Üldine taust ja projekti lühikirjeldus

Columbus Eesti AS (edaspidi Columbus) on globaalne digitaliseerimise lahendusi ja ärikonsultatsiooni pakkuv ettevõte. Columbuse visiooniks on olla ülemaailmselt eelistatuim teenusepakkuja digitaalsete ärilahenduste vallas ja missiooniks tuua digitaalse muutuse, kasvu ja tulevikukindluse Sinu ettevõttesse. Columbuse peamiseks eesmärgiks on aidata ettevõttel jõuda oma eesmärkideni ning saada jätkusuutlikuks, kasvavaks ning tulusaks ettevõtteks [10].

Teostatud projektiks on ITB1706 aine raames alustatud ettevõtte rahavoo juhtimise lahenduse edasi arendamine. Kassavoo lahendus on lisa olemasolevale Columbuse poolt kasutatavale rakendusele, mis on mõeldud eelkõige ettevõtte finants- ja projektijuhtidele. Antud lahendus aitab saada ülevaate hetke finantsseisundist kui ka tulevikuprognoosidest, tuvastada ja ennetada probleemsed kohad. Sobib nii suurele kui ka väiksemale ettevõttele.

## 1.2 Probleem ja projekti eesmärk

Põhilisteks probleemideks on:

- Palgapäeval ei ole pangaarvel piisavalt raha kõikidele töötajatele palga maksmiseks.
- Hankijad helistavad ja nõuavad tähtajaks tasumata arvete tasumist.
- Klientidele välja kirjutatud arvete laekumise tähtajad on pärast palgapäeva. Osad kliendid on maksetega hilinemas.
- Raha ei ole õigel ajal õiges kohas.

- Ettevõttel puudub reserv.
- Ebaratsionaalsed kulud mingis kategoorias.
- Ebarealistlik finantsprognoos tulevikuks.
- Hooajalised nõuded.
- Vale hinnapoliitika.

[8], [11]

Antud lahendus tagaks rahavoogude ehk likviidsuse igapäevast juhtimist. Firmal jätkuks raha kohustuste õigeaegseks katmiseks ja ülejääk teeniks juurde. Samas kassavoo lahendus aitab tuvastada vale hinnapoliitika, kus juurdehindlus ei katta kõiki kulusid ja ennetada finantsilisi raskusi, tekitades õigeaegselt reservi.

Lahendust võtavad ettevõtted kasutusse reeglina siis, kui on halvad ajad ja raha puudu. Samas ka siis, kui on head ajad ja raha on üle, siis saavad firmad prognoosida kui palju ja kui kauaks saavad investeerida. Põhiliseks kasutajaks ja sihtgruppiks on ettevõtte finantsjuht, kes tagab, et palgapäev oleks õigeaegne ja projektijuht, kes vastutab oma projekti laekumiste ja maksete tasakaalu eest.

Projekti eesmärgiks on saavutada kassavoo juhtimise lahendus, kuhu saab üleslaadida andmeid oma sissetulekutest ja väljaminekutest ning nende põhjal saada perioodi saldo tasakaalu ning prognoose tulevikuks.

## 1.3 Lühiülevaade teostatud funktsionaalsusest

Projekti raames funktsionaalsetest nõuetest sai valmis võimalus saada andmed arvete kohta Excel failist ning nende andmete põhjal koostatakse tabel, kust saab näha konto saldo prognoosi vastavalt seatud filtritele. Valides konkreetse kategooria saab näha eraldi detailtabelis selle arveid. Exceli faili üleslaadimisel läbib see valideerimist. Andmeid saab sisestada tabelisse ka manuaalselt. Ülevaate finantsolukorrast saab vaadata graafikult, kus on kuude lõikes kajastatud lõppsaldo. Funktsionaalsetest nõuetest ei saanud valmis andmete hankimine ERP (Enterprise Resource Planning) süsteemist ning

valideerimisaken. Mittefunktsionaalsetest nõuetest on kasutajasõbralikus ja sujuv kasutuskogemus.

Peamised kasutusjuhud:

1. Kasutaja saab üles laadida Excel formaadis faili, kust loetakse andmed arvete kohta andmebaasi.
2. Finantsjuht saab prognoosida konto saldot nädalate/kuu lõikes võttes arvesse esitatud ja saadud arved.
3. Finantsjuht saab tuginedes prognoosile tagada, et palgapäev oleks õigeaegne.
4. Projektijuht saab jälgida oma projektiga seotud sissetulekuid ning väljaminekuid.
5. Projektijuht saab koostada õiglase hinnapoliitika toodetele või teenustele
6. Finantsjuht saab teha otsuseid, kuidas rahavoog tasakaalu saada, mitte alus infot kokku koguda iga päev otsast peale.
7. Kasutaja saab palgad ja maksud õigel ajal makstud.
8. Kasutaja saab raha ülejäägi targalt investeeritud ning tekitada tulevikuks reservi.
9. Kasutaja saab hoida oma maksehäireregister rohelises.
10. Kasutaja saab raskel ajal teha teadlikud otsused.

## 1.4 Töö edasine struktuur

Järgmiseks osaks on „Metoodika“, kus tuleb projekti detailne kirjeldus, loetletakse tööriistu ja tehnoloogiaid ning antakse ülevaade tööprotsessist. Kolmandaks osaks on „Töö tulemused“, kus kirjeldatakse valminud produkti. „Analüüsi ja järelduste“ osas antakse hinnang valminud lahendusele. „Teostatud tööde detailsete logide“ all on kirjeldatud töömaht. „Meeskondliku hinnangu“ osas on kirja pandud hinnang iga meeskonnaliikme panuse kohta. Lõpuks on kokkuvõtte, kus lühidalt kirjeldatakse tehtud tööd. Lisades on iga meeskonnaliige eneseanalüüs.

# 2 Metoodika

Antud peatükis antakse lugejale ülevaade meeskondliku lõputööna tehtud kassavoo lahendusest, teostamiseks kasutatavatest tööriistadest ning tööprotsessist.

## 2.1 Objekti detailne kirjeldus

Meie poolt arendatav lahendus hakkab olema lisaks ReportÄpp'ile. ReportÄpp on Columbuses kasutuses olev majandustarkvara, mis pakub kliendile erinevaid võimalusi finantside ja inventuuri jälgimiseks.

Kassavoo lahenduse projektiga alustasime veel ITB1706 aine raames, mille me lõpuks ei jõudnud valmis ning oli tehtud otsus jätkata seda meeskondlikuna bakalaureuse projektina. ITB1706 aine raames jõudsime projektiga punkti, kus saime aru, et edasi jätkata meie tabeliga ei olnud võimalik ning ettevõttepoolse juhendajaga otsustasime kasutada nende tabelit "Tablix" ning integreerida see oma lahendusse. Selle tulemusel saime tagada ühtlast disaini terves projektis. Kuna esialgu ei olnud meil ligipääsu Columbuse andmebaasile, siis kasutasime enda oma (lokaalset), milleks oli MsSQL. Kaitsmisel tagasisidena paluti meil ülevaadata andmebaasi struktuuri, siis seda tegime muutes osad tabelid ning viies seda Columbuse andmebaasi. Kuna esialgse projekti kogu front-end kood oli kirjutatud React-Redux'iga, siis sellest ei saanud integreerides midagi kasutada ning pidime enamus koodi ülekirjutama. Kuna ReportÄppi APIs on kasutatud repositooriumi arhitektuur, siis pidime ka seal kogu koodi ülekirjutama. Eelnevalt sai ka valmis lugemine excel failist, mille saime muutmata üle viia hetkesse lahendusse.

Teostatud lahenduse oleme täielikult integreerinud ReportÄppi, sealhulgas üleviinud andmebaasi Azure`i, ümberkirjutanud API (Application Programming Interface) vastavusse üldise arhitektuuriga, kirjutanud API`le integratsiooni- ja ühikteste ja enamus visuaalist ülesseadnud läbi andmebaasi konfiguratsiooni kooskõlasse kogu disainiga.

Kassavoo lahendus pakub kasutajale, kelleks on üldjuhul finants- või projektijuht üleslaadida andmeid Exceli failist. Selle puhul on olemas failimall, mida süsteem oskab valideerida. Üheks nõueks oli ERP süsteemist andmete hankimise võimalus, kuid

hetkeseisuga pole jõutud seda realiseerida. Üleslaetud andmed laetakse Tablix`isse, kus kuvatakse saldo vastavalt seatud filtritele. Saldosummad arvutatakse kokku vastavalt kreditoorsele ja debitoorsele võlgnevusele. Vajutades konkreetsele väärtusele või kategooriale saab kasutaja vaadata detailtabelist selle andmeid arvete kaupa ning soovi korral muuta oodatava laekumise kuupäeva. Kasutajal on võimalik lisada uue arve või kategooria manuaalselt. Soovi korral saab ülevaate graafikult, kus kuvatakse saldo vastavalt seatud filtrile ning kus on märgitud arvelduskrediidi limit. Vastavalt seatud liimidile ning saldole kuvatakse tulbad, kas rohelise või punase värvina, andes ülevaate, kas saldo on üle või alla seatud limiidi. Kassavoo lahenduse kasutajal on võimalik kustutada laekumiste ja väljaminekute kategooriaid.

Kogu projekti visuaal on seadistatud vastavalt ReportÄppile läbi andmebaasi konfiguratsiooni. Vajadusel oli lisatud enda kood, mida sai seadistada läbi konfiguratsiooni, näiteks failide üleslaadimise komponent ning mitmesugused spetsiifilised väljad andmete sisestamiseks, millest lähemalt räägitakse peatükis 3.5.1.

## 2.2 Tööriistade kirjeldus

Projekti back-end on kirjutatud C# programmeerimiskeeles kasutades ASP.NET Core raamistiku ning Entity Framework ORM'i (Object-Relational Mapping) ja kogu arendamine toimus IDE-s Rider, mis on pakutud JetBrains poolt ning millele on tudengitele olemas litsens.

Esialgu kasutasime front-end'i loomiseks Typescript ja React raamistiku ja olekute kontrollimiseks Redux töörista. Esialgselt oleme proovinud integreerida meie Reacti rakendust ReportÄppi keskkonda, mis meil ebaõnnestus. Arutades antud probleemi ettevõttepoolse juhiga oleme otsustanud, et me loobume meie esialgsest visuaalist ning võtame kasutusele ReportÄppis olevat tabelit ning ülejäänud visuaali loome läbi andmebaasi konfiguratsiooni.

Front-end'i parema variandi loomiseks oleme teinud disainiprototüübi Figmas. Figma on veebiteenus, mis on suunatud liidese arendamiseks ja prototüüpide loomiseks. Meeskonnaliikmete muudatuste jälgimiseks kasutasime versioonikontrollisüsteemi Azure poolt, kuigi endise projekti jaoks oleme kasutanud GitHub`i. Samuti oleme kasutanud Azure`is projektihaldussüsteemi selleks, et saaks jooksvalt jälgida ülesandeid

ning nende tähtaegasid. Andmebaasi struktuuri ülesehitamiseks ja kõikide skeemide joonistamiseks kasutasime dbdiagram.io. Projekti andmebaasi loomiseks kasutasime Azure SQL Database andmebaasi ning DataGrip IDE'd (Integrated Development Environment).

## 2.3 Tööprotsessi kirjeldus

Tööprotsess jagunes kolmeks põhiliseks osaks ettevalmistamine, arendus ning vahetulemuste valideerimine. Nendest lähemalt on kirjas järgnevates kolmes alampeatükis.

### 2.3.1 Ettevalmistus

Peale meeskonnaprojekti kaitsmist oleme mitmekordselt kohtunud ettevõttepoolse meeskonnaga, et analüüsida eelnevalt tehtud tööd, arutada edasised sammud ning koostada disainiprototüüp ja tööplaan. Meeskonnaprojekti oleme arendanud kolmekesi ning planeerimisfaasi jooksul on meie meeskonnaga liitunud Vladislav Jekimtsev. Selle faasi jookusl oleme otsustanud, et me tõstame projekti ReportÄppi keskkonda ning teeme arendusi edasi seal. Pannes paika nõudeid meie projektile oleme koostanud disainiprototüübi, mille eelnevalt valideerisime ettevõttepoolse juhendajaga. Kui plaan oli koostatud ja ettevalmistus sai lõpuni viidud oleme alustanud projekti edasiarendamisega.

### 2.3.2 Arendus

Tööprotsess oli kogu tööaja vältel jagatud enam-vähem võrdselt. Planeerimisfaasi jooksul oleme töötanud enemasti kolmekesi (Ruben, Daniel ja Paul Opmann) ning Vladislav võttis seda aega, et tutvuda meie esialgse projektiga, nõuetega ja dokumentatsiooniga. Tööprotsessi käigus meeskond oli otsustanud, et Paul ei jätka meiega tööd, kuna tema panus polnud küllaltki piisav. Edasi oleme arendanud projekti juba kolmekesi. Oleme töötanud kontoris, kui olukord riigis seda soodustas, kuid enamik ajast töötasime kodus ja kontakteerisime Microsoft Teams vahendusel. Oleme alati aidanud teineteist, ning vajadusel küsinud abi ettevõtte arendajatelt. Meeskonnaprojekti kaitsmisel saime tagasisideks, et oleme liiga vähe suhelnud ettevõtte meeskonnaga. Lõpuprojekti tegemisel oleme arvestanud sellega ning püüdsime võimalikult tihti valideerida oma vahetulemusi ning vajadusel julgelt abi küsida.

Töö algas sellest, et me püüdsime integreerida meie lahendust ReportÄppi. Sellega kaasnesid probleemid, kuna meie API arhitektuur erines ReportÄppis kasutatust ning pidime API üle vaatama ning mõned asjad üle kirjutama. Samal ajal oleme teinud korrigeerimisi andmebaasi osas. Front-endi integreerimisel tekkis problem, sest see oli kirjutatud kasutades React raamistiku, aga ReportÄpp seda ei toetanud. Oleme proovinud integreerida React`i, kuid see ebaõnnestus. Selliste olukordade tekkimisel oleme alati suhelnud ettevõttepoolsete juhendajatega. Analüüsides probleemi oleme otsustanud, et võtame täielikult kasutusele ReportÄppis olevad visuaali alused, milleks on tabelid ja modaalid. Visuaali seadistatakse läbi andmebaasi konfiguratsiooni ning algul pole julgenud ise katsetada sellega ning palusime juhendamist arendajate poolt. Saime sellest kiiresti aru ning saime ise teha muudatusi. Vahepeal mingit funktsionaalsust konfiguratsioonil polnud ning oleme palunud ettevõtte arendajatelt teha vastavad muudatused.

Siis, kui endine lahendus sai integreeritud ReportÄppi oleme alustanud koodi korrigeerimistega ning uue funktsionaalsuse lisamisega. Koodi refaktoorimist oleme teostanud viimastel nädalatel ning oleme investeerinud aega testide kirjutamisse, selleks oleme läbi vaadanud kursused internetis.

Tööülesanded olid pandud kirja Azure`s. Jagasime tööülesanded nii, et igaüks tegeleks sellega, mida ta oskas kõige paremini ning me saaksime töötada maksimaalselt effektiivselt. Oleme töötanud paaris ning üldiselt pidasime kinni agiilse tarkvaraarenduse põhimõtetest.

Ruben Galoyan: Enamasti tegeles back-endi ja andmebaasiga. On teinud muudatusi konfiguraatsioonis. Danieliga koos oli pidevas kontaktis ettevõtte arendajatega.

Daniel Antonov: Enamasti tegeles back-endi ja front-endiga. Tegi muudatusi konfiguratsioonis. Rubeniga koos oli pidevas kontaktis ettevõtte arendajatega.

Vladislav Jekimtsev: Enamasti tegeles andmebaasi ja back-endiga. Kirjutas unit teste ning proovis integreerida meie rakendust ReportÄppi.

Paul Opmann: Daneiliga koos proovis integreerida meie rakendust ReportÄppi ning tegi ühe modaali (valideerimisakna) baasi, kuid seda me ei saanud kasutada meie rakenduses.

### 2.3.3 Vahetulemuste valideerimine

Iga nädal kolmapäeviti oleme saanud kokku ettevõttepoolse juhendajaga ning arendajatega, kes aitasid meid selle projekti vältel. Valideerisime tehtud tööd ning arutasime meie edasised sammud.

# 3 Töö tulemused

Töö tulemuseks hetkel on majandustarkvara lisa olemasolevale ReportÄppile, millel on olemas põhifunktsionaalsus, mida ettevõttepoolne juhendaja soovis saada esimese etappi lõpuks.

Kassavoo lahenduse kasutajal on hetkel võimalik jälgida ettevõtte rahavoogusid tabelist, mis kajastab kõiki sissetulekuid ja väljaminekuid vastavalt seatud filtrile. Kasutaja saab muuta algsaldot ning arvelduslikku krediidilimiidi, muuta oodatavat laekumise kuupäeva konkreetsel arvel, lisada tulu ja kulu kategooriad. Andmeid tabeli jaoks saab kasutaja laadida Excel failimalliga, mis valideerimise õnnestumisel laetakse üles andmebaasi ning kuvatakse tabelis. Alternatiivina saab kasutaja vaadata ülevaate graafikult, mis kajastab vastavalt filtrile erivärvides perioodi lõppsaldo.

## 3.1 Rakendus

Alguses meie lahendus oli ReportÄppi osa, kuid hiljem oli otsustatud, et tuleb projekti viia eraldi plokki. Selleks oleme kasutanud ettevõtte arendaja poolt ettevalmistatud malli. Rakendus koosneb 6 peamisest osast: põhitabel, detailvaade, menüü, andmete import aknad ning graafik.

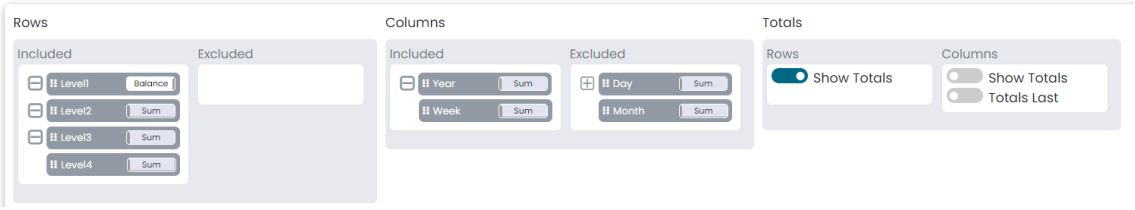
### 3.1.1 Põhitabel

Lõppsaldo prognoosi kuvamiseks kasutame Tablixi põhitabelit. See mida ja kuidas peaks põhitabel kuvama on seadistatud konfiguratsioonis, samamoodi määratakse ka API url, kuhu tabel teeb päringu andmete saamiseks. Andmed on jaotatud nii, et tulud ja kulud on gruppeeritud kategooriate kaupa. Põhitabelis näidatakse ainult kategooriate summasid. Rahavoogude alla käivad kõik tulud ja kulud. Algsaldo ja rahavoogude summa on omakorda lõppsaldo osad. Eelmise perioodi lõppsaldo on järgmise periodi algsaldo. Vajutades mingile lahtrile saab vaadata selles lahtris oleva summa taga olevaid arveid, mida kuvatakse detailvaates.

| | 2021 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Level1 > Level2 > Level3 > Level4 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| **Final Balance** | **100** | **100** | **-113 900** | **-124 222** | **-148 722** | **-183 122** | **-206 022** | **-226 122** | **-226 122** | **-226 122** | **-226 122** |
| **Cashflow** | | | **-114 000** | **-10 322** | **-24 500** | **-34 400** | **-22 900** | **-20 100** | | | |
| **Income** | | | **114 000** | **10 322** | **24 500** | **34 400** | **22 900** | **20 100** | | | |
| Potential income | | | | | | | | | | | |
| Software development | | | 114 000 | 10 322 | 24 500 | 34 400 | 22 900 | 20 100 | | | |
| **Outcome** | | | **-228 000** | **-20 644** | **-49 000** | **-68 800** | **-45 800** | **-40 200** | | | |
| Potential outcome | | | -114 000 | -10 322 | -24 500 | -34 400 | -22 900 | -20 100 | | | |
| Tax | | | -114 000 | -10 322 | -24 500 | -34 400 | -22 900 | -20 100 | | | |
| Initial Balance | 100 | | | | | | | | | | |

Joonis 1. Tablixi põhitabel

Tablix pakub erinevaid võimalusi selle seadistamiseks. Näiteks kasutaja võib määrata, kas soovib näha andmeid kuude või nädala lõikes.




Joonis 2. Põhitabeli seadistamine

### 3.1.2 Detailvaade

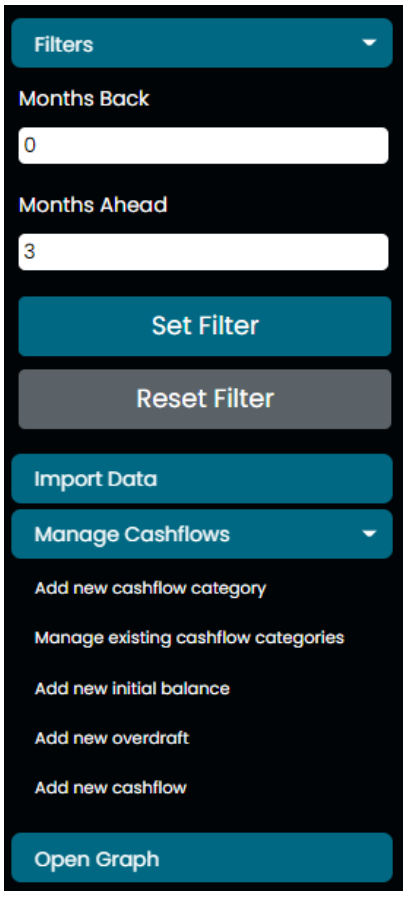
Selleks, et kasutajal oleks võimalik vaadata andmeid arvete kaupa on olemas detailvaade, mis on Tablixi osa. Valides põhitabelis, kas kategooria või konkreetse kuupäeva summa tehakse päring APIsse, mis tagastab vastavad andmed detailvaatesse. See, mis väljad peaksid olema näidatud ja kas nad on redigeeritavad seadistatakse konfiguratsioonis.

| Invoice number | Amount | Amount Due | Partner Id | Transaction Date | Payment Due Date | Project Id |
|---|---|---|---|---|---|---|
| 2103 | -2100 | -1600.00 | 15 | 03.04.2021 | 17.05.2021 | Projekt 2 |
| 2104 | -200000000 | -1000.00 | 13 | 04.04.2021 | 18.05.2021 | Projekt 2 |
| 2105 | -2100 | -1400.00 | 14 | 05.04.2021 | 19.05.2021 | Projekt 2 |
| 2106 | -2100 | -1600.00 | 15 | 06.04.2021 | 20.05.2021 | Projekt 2 |
| 2107 | -3300 | -122.00 | 13 | 07.04.2021 | 21.05.2021 | Projekt 2 |
| 2108 | -3700 | -3000.00 | 14 | 08.04.2021 | 22.05.2021 | Projekt 2 |
| 2109 | -2100 | -1600.00 | 15 | 09.04.2021 | 23.05.2021 | Projekt 2 |
| **TOTAL** | | **-10322.00** | | | | |

Joonis 3. Tablixi detailvaade

### 3.1.3 Menüü

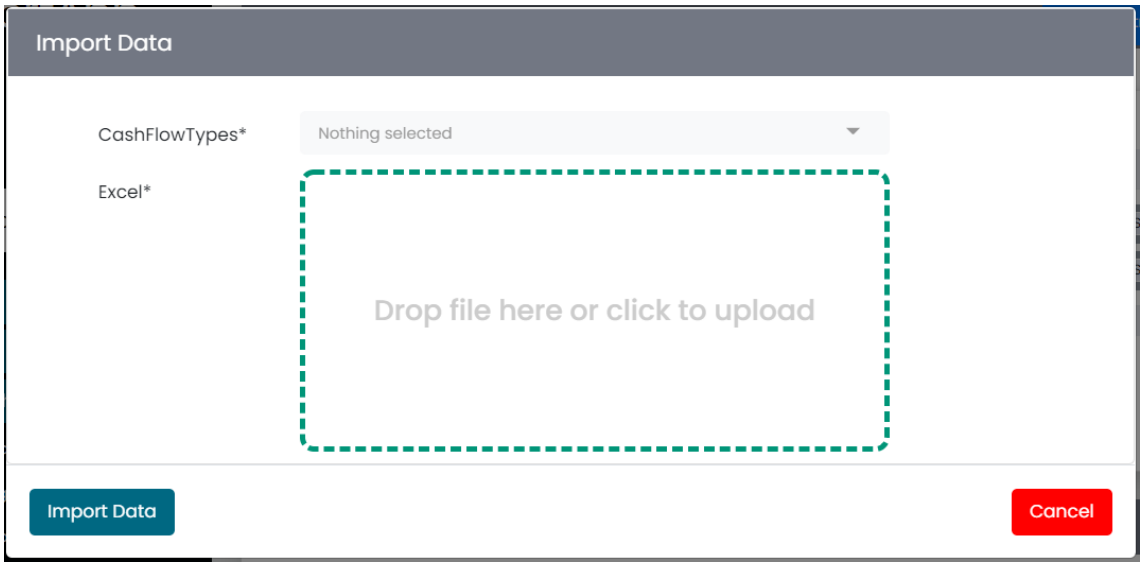
ReportÄppis on olemas navigatsiooni menüü, kus saab seadistada filtrid, laadida üles andmed Excel faili kujul, vaadata lõppsaldo prognoosi graafikul ning avada modaalid andmete käsitsi haldusega. Saab lisada uut rahavoo kategooriat, redigeerida olemasolevaid, lisada uut algsaldot, arvelduslikku krediidilimiidi ning käsitsi sisestada uus arve.



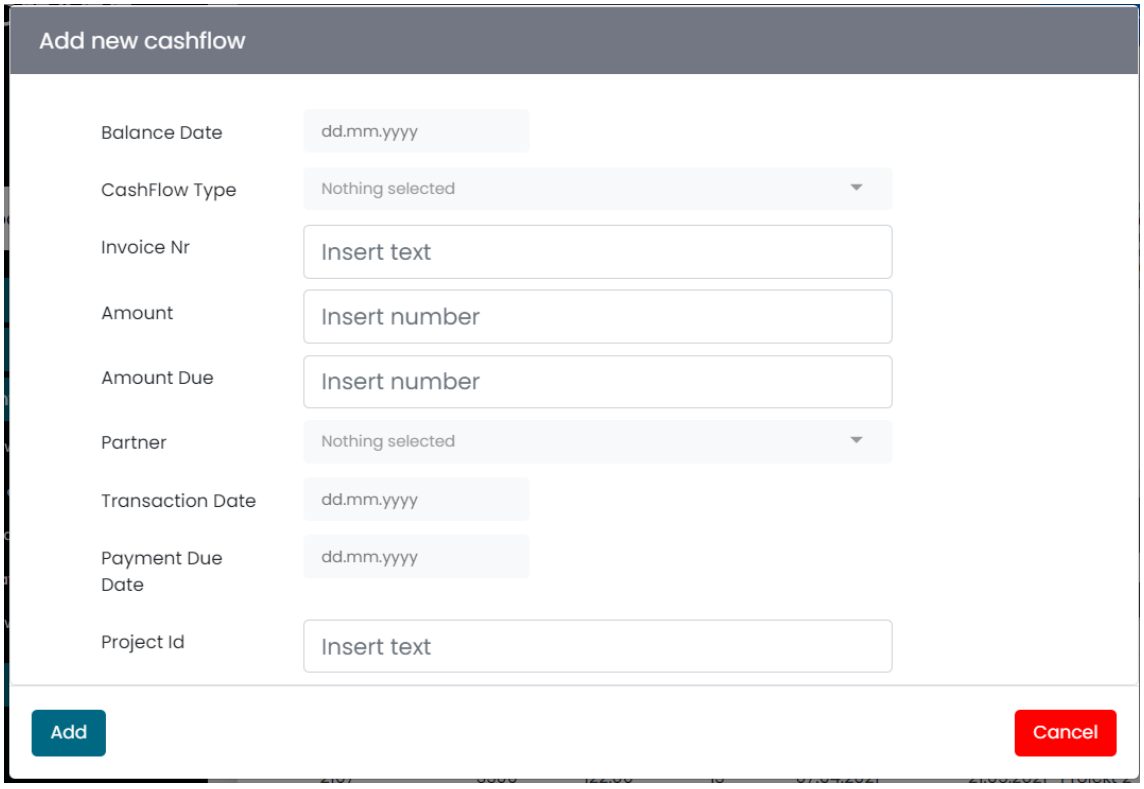

Joonis 4. Navigatsiooni menüü

### 3.1.4 Andmete import

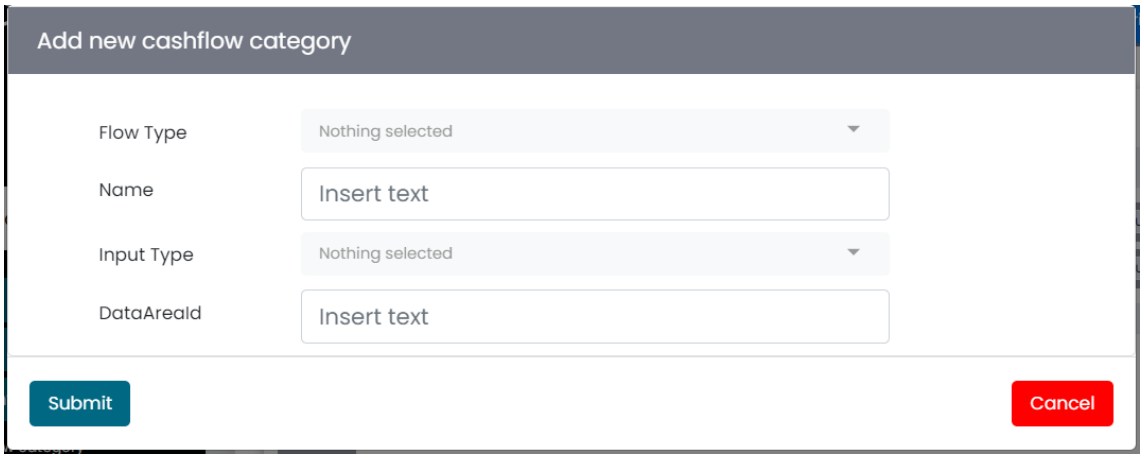
Andmed saab üles laadida mitmel erineval viisil: Excel failiga või sisestada käsitsi. Excel faili üles laadimisel tuleb valida, mis kategooria kohta laaditakse andmed üles (vt Joonis 5). Peale faili üleslaadimist selle sisu valideeritakse. Erinevad andmed saab üles laadida käsitsi sisestades vastavas modaalaknas (nt. Joonis 6 ja Joonis 7)




Joonis 5. Exceli üleslaadimine




Joonis 6. Uue arve käsitsi sisestamine




Joonis 7. Uue rahavoo kategooria lisamine

### 3.1.5 Imporditava Exceli mall

Exceli kaudu andmete üleslaadimiseks on loodud kindel Exceli mall.

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| Partner Name* | Partner Code* | Transaction Date* | Payment Due Date* | Invoice Nr* | Project | Amount* | Amount De* |
| Columbus | 555555555 | 01.02.2021 | 15.02.2020 | 2101 | Projekt 1 | 5000.00 | 4500.00 |
| Tafutik | 777777777 | 02.02.2021 | 16.02.2020 | 2102 | Projekt 2 | 110000.00 | 109500.00 |
| Al Fredo | 888888888 | 03.02.2021 | 17.02.2020 | 2103 | Projekt 2 | 2100.00 | 1600.00 |
| Columbus | 555555555 | 04.02.2021 | 18.02.2020 | 2104 | Projekt 2 | 200000000.00 | 1000.00 |
| Tafutik | 777777777 | 05.02.2021 | 19.02.2020 | 2105 | Projekt 2 | 2100.00 | 1400.00 |
| Al Fredo | 888888888 | 06.02.2021 | 20.02.2020 | 2106 | Projekt 2 | 2100.00 | 1600.00 |
| Columbus | 555555555 | 07.02.2021 | 21.02.2020 | 2107 | Projekt 2 | 3300.00 | 122.00 |

Joonis 8. Imporditava Exceli mall

### 3.1.6 Graafik

Lõppsaldo prognoosi saab vaadata kasutajasõbralikumal moel – graafikul. Lisaks lõppsaldodele on graafikul kajastatud ka arvelduslik krediidilimiit (vt Joonis 9).


Joonis 9. Lõppsaldo graafiku kujul

## 3.2 Korralikult väljatöötatud nõuded

Projekti arendamise alguses saime üldise ülevaate tuleviku projektist. Vastavalt sellele ülevaatele [9] planeerisime koos BI (Business Intelligence) ärisuunajuhiga kohakaasluses meie projekti mentori Kristen Pugi'ga kogu tööprotsessi ja vajalikke nõudeid. Eraldi nõudeid meile polnud antud Columbuse poolt, vaid põhinesid tööprotsessi käigus suulistel kokkulepedel. Nõute kirjeldamise eelduseks oli põhimõte, et kogu lahendus oleks kasutajasõbralik ning kiire.

Meie projekti jaoks olid järgmised nõuded:

- Kliendil on võimalik laadida oma andmed Exceli kujul või ERP süsteemist.
- Kasutaja andmete põhjal saab koostada põhitabel lõppsaldo prognoosiga.
- Excel faili üleslaadimisel peaksid andmed läbima valideerimise, mille tulemus kuvatakse kasutajale.
- Andmete valideerimise etappil võib tekkida olukord, kui veeru nimele pole õiget paari leitud. Sellisel juhul kasutaja peaks ise koostama paari oodatavast

veerunimest ja Excelis olevast nimest. Excelist saadud veerunimed, millel pole paari asuvad „Paarita veerunimed“ plokis.

- Tabelis saab sisestatud andmete alusel visualiseerida konto saldo prognoosi nädalate või kuude kaupa lisaks saab kasutada filtrid (näiteks määrata uus vaadeldav periood).

- Kasutajal peab olema võimalus muuta olemasolevad väärtused (näiteks lisada eeldatav arve laekumise kuupäev, mis erineb maksetähtajast).

- Kasutajal peab olema võimalus vaadata andmeid arvete lõikes. Selleks kasutaja peaks valima põhitabelis lahtri või kategooria, mille kohta soovib täpsemat infot. Andmed kuvatakse detailvaates.

- Lisaks andmete muudatuste võimalusele peab olema võimalik sisestada kõik andmed käsitsi ehk manuaalselt.

- Andmete selgemaks visualiseerimiseks peab olema võimalik vaadata lõppsaldo prognoosi graafikult, kus joonega on märgitud ettevõtte arvelduslik krediidilimiit. Kui lõppsaldo on krediidilimiidist väiksem, siis on see märgitud punase värviga. Kui lõppsaldo krediidilimiidist suurem, siis on see märgitud rohelisega.

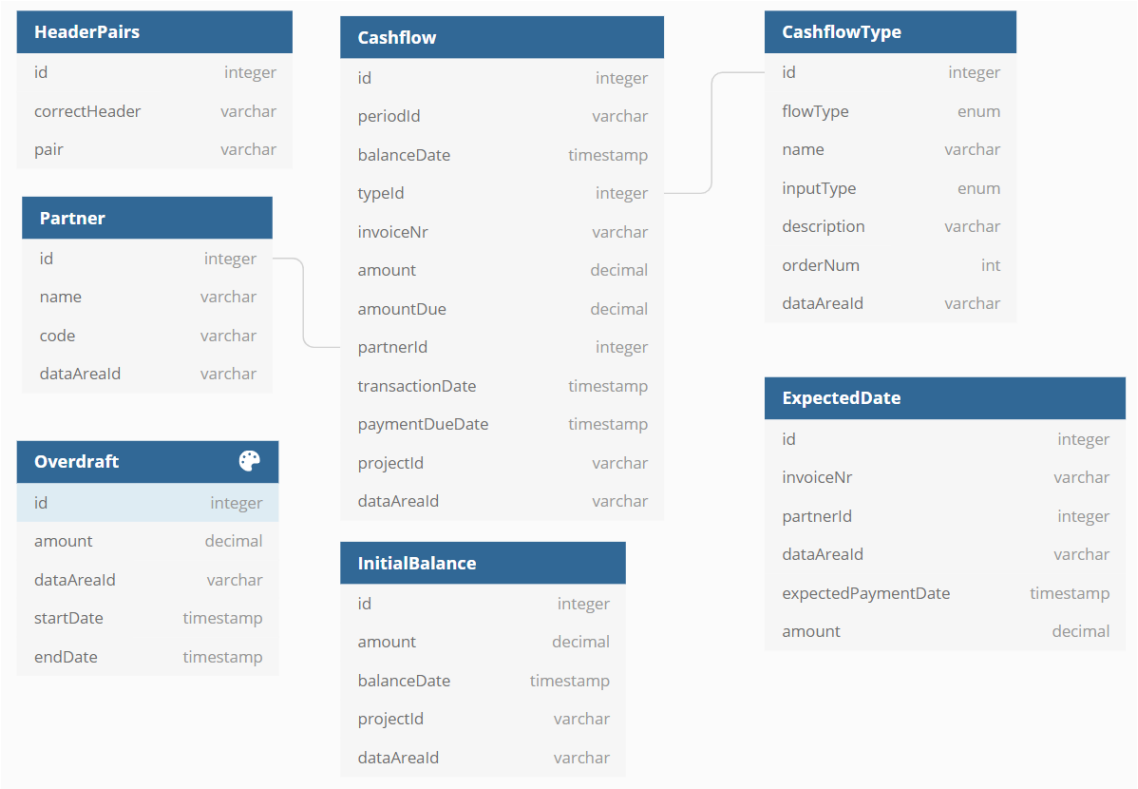
## 3.3 Arhitektuur

Front-endi osas me pole eriti palju uut koodi lisanud. Enamik front-end`iga seotud koodi on ReportAppCore.MVC projektis, kõik modaalid, tabel ja detailvaade on seadistatavad läbi konfiguratsiooni. Konfiguratsiooniks on andmebaas, kus vastavas tabelis peab lisama uue rea, mis mõjutab front-endi soovitud osa.

Kogu back-end kood asub eraldi ColCashFlow repositooriumis ning vastavalt Columbuses ReportÄppis kasutatavale back-endi ülesehitusele koosneb ta kolmest eraldiseisvast projektifailist v.a. testid. Back-end`i projekt on tehtud ettevõtte arendaja poolt saadud malli põhjal. Meie API on eraldiseisev plokk ning on ülesehitatud vastavalt repositooriumi mustrile.

Testid asuvad CashFlow solutioni Tests kaustas. Nii ühik- kui ka integratsiooni testid on kirjutatud eraldi projektidesse ning Clean Code põhimõtteid järgides jagatud kõik osad eraldi kaustadesse vastavalt MVC (Model-View-Controller) mustrile.

Andmebaasi struktuur on järgmine:




Joonis 9. Andmebaasi struktuur

CashFlow tabel on ettevõtte olemasolevatest tuludest ja kuludes info säilitamiseks. See tabel on ühendatud CashFlowType ja Partner tabelitega üks-mitu seosega. CashFlowType tabelis asub info tulu ja kulu kategooriatest ja sellest millised on nende sisestuse tüübid. Partner tabelis on ettevõtte partnerite informatsioon. InitialBalance tabelis asuvad sisestatud algsaldot ja Overdraft tabelis on kogu info ettevõtte arvelduskrediidi limiidi kohta. ExpectedDate tabelis hoitakse eeldatavad arvete laekumiste kuupäevad. HeaderPairs tabel lisati selleks, et oleks võimalik mugavam teha toimingud Exceli kujul tulenevaid andmetega, täpsemalt veerude nimetuste kontrollimiseks ja valideerimiseks.

## 3.4 Disain

Kassavoo lahenduse disaini nõueteks oli üldine kokkusobivus ReportÄppi stiiliga ning võimalusel ära kasutatud Columbuse poolt eelnevalt tehtud elemendid.

Üldiselt olid tavapärastes modaalaknates asetatud kõik sisendid keskele ritta ning selle nimetused paigutatud nendest vasakule.

Meie poolt oli ka teostatud kasutajasõbralik graafik. Graafikult saab kasutaja visuaalse ülevaate arvelduslikku krediidi limiidist ja lõppsaldost vastavalt filtrile, kas kuude või nädalate kaupa. Hiirega suunates konkreetsele saldole saab väikses aknas ülevaate selle summast, sama kehtib ka arvelduslikku krediidi joone kohta graafikul. Vastavalt sellele, kas lõppsaldo ületab või mitte arvelduslikku kreedidi liimidi, kuvatakse lõppsaldo tulp, kas rohelise või punase värvina.

## 3.5 Kood

Kassavoo lahenduse kood jaguneb front-end'iks ning back-end'iks. Kõik front-end elemendid olid tehtud meie poolt enamasti kasutades JavaScripti ning selle jQuery ja Bootstrap raamistiku. Back-end oli kirjutatud kasutades Asp.Net Core ehk C# programmeerimiskeelt.

### 3.5.1 Front-end

Kõik front-end'iga seotud lahendused asuvad ReportAppCore.Mvc projektis, mida me omalt poolt väga ei teinud ning ei muutnud, kuna kõik seadistused käisid läbi andmebaasi konfiguratsiooni. Omalt poolt lisatud failid oleme alati paigutanud eraldi kausta CashFlow. Back-end'ilt andmete saamiseks ning töötlemiseks kasutasime JavaScripti Ajax päringuid.

Meie põhitabel ehk Tablix on loodud läbi Grid tabeli andmebaasis, milles on loodud meie tabeli unikaalne id, antud nimi, tehtud algseadistus, kas on kuvatav kasutajale ning määratud DbTenantId. Tabelisse andmete saamiseks on kasutatud tabelit GridMatrix, milles on seadistatud url, kust tabel pärib endale väärtusi ja antud väärtus, mida otsitakse back-endilt saadud JSON failist ning kuvatakse maatriksis. Samas tabelis on tehtud teised põhimaatriksi üldised seadistused, mida ja kus kuvada ning millises formaadis. Põhigraafiku tegemiseks oleme palunud abi ettevõttepoolsetelt arendajatelt.

Kõik meie lahenduses kasutatavad modaalaknad on seadistatud läbi andmebassi tabeli DataInputConfiguration, milles on kidlaks määratud meie tabeli:

- GridId – Meie põhitabeli Id ehk millises tabelis kuvatakse modaalaknas teostatud muudatused
- Name – Nuppuna kasutajale kuvatav aknanimi
- DbTenantId – Unikaalne andmebaasi id
- GroupId – Määrab, kuhu gruppi panna nupp, mis avab seda modaali
- SubmitApiUrl – Url, millele teostatakse päring muudatuste aktsepteerimisel
- SubmitBtnName – Kasutajale aktsepteerimisenuppu kuvatav nimi
- RefreshAfterSubmit – Boolean väärtus vastavalt millele tabelit uuendatakse või mitte
- SubmitApiUrlType – Ajax päringutüübi kindlaks määramine

Modaalaknates kasutatud sisendid on seadistatud läbi DataInputConfigurationLine tabeli, milles tuleb määrata kindlaks, mis id'ga modaalaknale soovid sisendi lisada ning annad sellele vajalikud seadistused. DataInputConfigurationLine tabeli olulisemad väljad on:

- Name – Sisendi juures olev nimi
- Type – Sisendi tüüp (nt dropdown)
- GridDimension – Dimensiooni id, kust saab andmed nt dropdown'i jaoks
- Order – Järjekorranumber
- DefaultValue – Vaikimisi väärtus (ei pea lisama)
- DbValue – Submit nuppu vajutamisel päringu objektis kasutatav nimi

Kui Type määrad dropdown'iks, siis on vajalik GridDimensionId lisamine ning omaette selle dimensiooni loomine GridDimension tabelis. Antud tabeli ülesandeks on teha API päring õigele urlile ning tagastada nimekiri soovitavatest elementidest.

Kuna Columbusel endal puudusid osad vajalikud sisendid, siis pidime neid ise implementeerima ning seadistama läbi konfiguratsiooni. Nendeks olid näiteks faili

lohistamine üleslaadimiseks, kahe kuupäevaga sisend, arvuline ja tekstiline sisend ning graafik. Graafiku teostamiseks kasutasime JavaScripti Chart.js teegi.

Excel faili importimine on tehtud läbi konfiguratsiooni sarnasel moel, kuid tabeliteks olid DataImportConfiguration ja DataImportConfigurationLine.

### 3.5.2 Back-end

Kogu back-end kood on kirjutatud vastavalt repositooriumi mustrile. Repositooriumi muster üldsõnaliselt tähendab seda, et kõik andmebaasi käsklused hoitakse ühes kohas, mis meie lahenduse puhul asub Repositories kaustas ja kannab nime EfRepository ning pärib IRepository<TEntity>  interface, kus TEntity on klassiks.

ColCashFlow.Entities projektis on kokku kolm kausta Database, Helpers ja Models ning nagu nimi ütlebki vastutab kõikide loogikakihis kasutatavate objektide eest. Database kaustas asuvad mudelid vastavalt andmebaasi tabelitele, mis on kasutatud läbi UnitOfWorki andmete pärimiseks. Helpers kaustas asuvad objektid, mis on kasutatud abiobjektidena. Models kaustas on objektid, mida on kasutatud erinevates lahenduse osades andmete erikujul tagastamiseks, sealhulgas ka andmetekogumiseks ning -töötlemiseks loodud SQL vaatest.

ColCashFlow.DAL projekt on Data Access Layer ehk kiht andmetele juurdepääsuks. DAL on vahekihiks Apile ja Entities projektidele. DAL projektis asub ApiDbContext ja vastavalt repositooriumi mustrile ka ApiEfUnitOfWork, milles luuakse kõik projektiks vajalikud repositooriumid. ApiDbContext on kasutatud entity`te loomiseks, millele saab rakendada CRUD operatsioone. Samas klassis on ka funktsioon OnModelCreating, mis määrab kindlaks andmebaasi objektide piiranguid. Abstract kaustas asuvad kõik interface`id. Repositories kaustas hetkel asub ainult EfRepository, mis on baasrepositooriumiks. Selles on kõik baasfunktsioonid, mis on kasutusel loogikakihis. EfRepositoryProvider on klass, milles määratakse, mis tüüpi repositooriumi objektile luuakse, kas kasutades baas interface või erilist, kus tuleb luua eraldi repositoorium vajalike funktsioonidega.

ColCashFlow.Api projektis asub kogu meie kassavoo lahenduse loogika ehk kontrollerid ning on seadistatud Startup ja appsettings.json failid. Controllers kaustas asuvad kõik meie lahenduses kasutatavad kontrollerid ehk kogu meie projekti loogika.

Helpers kaustas asub abikiht, milles on erinevad klassid, mis on kasutatud andmetelugemiseks Excel failist. Väärtuste lugemine failist toimub NuGet package'i DocumentFormat.OpenXml abil. OpenXml eeliseks on see, et kasutajal ei pea olema arvutis installeeritud Excel, kuna antud package loeb sisse faili metaandmed. Seetõttu saadud väärtused tuleb osadel juhtudel teisendama meile vajalikku tüüpi.

Middleware kaustas asuvad Columbuse arendjate poolt tehtud komponendid. ExceptionHandlerMiddleware on kasutatud debug`imisel vigade leidmiseks. MiddlewareExtension klass on kasutatud Startup Configure meetodis, mis vastutab selle eest, kuidas rakendus vastab HTTP päringutele.

Extension kaustas on IdentityServiceExtensions klass, mis on kirjutatud samuti Columbuse arendajate poolt ning vastutab kasutajate autentimise eest.

Controllers kaustas asuvad kontrollerid, milles on kirjutatud kogu rakenduse loogika. Hetkel kõik meie kontrolleriklassid ei nõua kasutajate sisselogimist ning on märgitud [AllowAnonymous] atribuutidega. Kõikide kontrollerite url on malliga api/[controller]. Andmebaasist andmete pärimiseks on kasutatud nii ColCashFlow.Entities kui ka ColCashFlow.DAL projekte.

Kõik teostatavad HTTP päringud on väga konkreetsete nimetustega, mis peegeldavad nende käitumist. Näiteks GET päringu puhul, mis peab tagaststama massivi Partneritest saab urlilt api/Partners/GetAllPartners. Andmete saamiseks võetakse päring`ust dataAreaId ning kasutades _uow (Unit of work) asünkroonselt leiab Partner tabelist kõiki partnereid, kelle dataAreaId on võrdne päritatavaga (vt Joonis 10).

```
[HttpGet("GetAllPartners")]
public async Task<ActionResult<IEnumerable<Partner>>>
GetAllPartners([FromQuery] string dataAreaId)
{
       if (!ModelState.IsValid) return NotFound();
      var partners = await _uow.Partner.Where(s => s.DataAreaId ==
      dataAreaId).ToListAsync();
       return partners;
}
```

Joonis 10. GetAllPartners funktsioon

Antud projektis on eraldi Temp kaust, kuhu salvestatakse ajutine Excel fail, mida kasutatkse väljade valideerimiseks ning seejärel kustutakse.

### 3.5.3 SQL

Tabelite loomiseks kirjutasime SQL laused kasutades „create table“ käsklust, mille sees oli võimalik määratleda kõik väljad, nende tüübid ja atribuudid.

```
create table CashFlow
(
    Id              int identity
        constraint PK_CashFlow
            primary key,
    PeriodId        varchar(16),
    BalanceDate     datetime2      not null,
    TypeId          int            not null,
    InvoiceNr       nvarchar(16)   not null,
    Amount          decimal(18, 4) not null,
    AmountDue       decimal(18, 4) not null,
    PartnerId       int            not null,
    TransactionDate datetime2      not null,
    PaymentDueDate  datetime2      not null,
    ProjectId       nvarchar(16),
    DataAreaId      nvarchar(16)   not null,
    constraint AK_CashFlow_InvoiceNr_PartnerId_TypeId_BalanceDate_DataAreaId
        unique (InvoiceNr, BalanceDate, TypeId, PartnerId, DataAreaId)
)
go
```

Joonis 11. CashFlow andmebaasitabel

Meeskonnaprojekti raames kogu info kogumiseks proovisime luua SQL Join, et oleks võimalik põhitabelis kuvada vajalik info mitmest andmebaasi tabelist. Töökäigus selgus, et Entity Framework’is polnud võimalik teha tabelitest join meie soovide järgi ning suheldes ettevõtte arendjaga otsustasime, et see lahendus ei sobi meie jaoks ja lõpuprojekti jaoks oleme lisanud SQL View ’GetGridData’. See View on kasutusel GridDataController’is.

```
CREATE VIEW GetGridData
    AS SELECT CashFlow.Id, CashFlow.periodid, CashFlow.balancedate,
COALESCE(ExpectedDate.ExpectedPaymentDate, CashFlow.PaymentDueDate) as
PaymentDueDate, CashFlow.TransactionDate,CashFlow.invoicenr, CashFlow.amount,
CashFlow.amountdue, CashFlow.typeid, CashFlowType.FlowType, CashFlowType.Name
as FlowTypeName, CashFlowType.InputType,CashFlow.partnerid, Partner.Name as
PartnerName, CashFlow.projectid, CashFlow.dataareaid
    FROM CashFlow LEFT JOIN CashFlowType ON CashFlow.TypeId = CashFlowType.Id
    LEFT JOIN Partner ON CashFlow.PartnerId = Partner.Id
    LEFT JOIN ExpectedDate ON CashFlow.InvoiceNr = ExpectedDate.InvoiceNr AND
CashFlow.PartnerId = ExpectedDate.PartnerId
        AND CashFlow.DataAreaId = ExpectedDate.DataAreaId
go
```

Joonis 12. GetGridData SQL View

## 3.6 Testid

Kassavoo lahenduse raames kirjutasime back-end koodi integratsioone ja ühikteste. Testid katavad 76% kogu API koodist. Mõlemad testprojektid viitavad ColCashFlow.Api, ColCashFlow.DAL ja ColCashFlow.Entities projekte. Testide kirjutamiseks kasutasime XUnit testimise raamistiku ning kõik testid on märgitud [Fact], kui on üks sisend või [Theory] attribuutidega, kui testitakse korraga mitu juhtu erinevate väärtusega. Testide kirjutamisel oleme jälginud AAA (Arrange Act Assert) mustrit.

### 3.6.1 Integratsiooni testid

Integratsiooni testid on mõeldud tarkvara mitme erineva koodikomponendi koostöö testimiseks. Hetkel on testpäringute tegemiseks ja testimiseks kasutatud sama andmebaas, mida kasutab päris API päringuteks. Failis testappsettings.json on kindlaks tehtud andmebaas, kuhu testprojekt teeb päringuid.

Integratsiooni testides on kontrollitud kõikide kontrollerite loogikat ja saadud katteprotsendiks 76%. Testide jaoks on tehtud eraldi CustomColCashFlowFactory ja IntegrationTest fail, milles on testide käivitamiseks vajalikud konfiguratsioonid. CustomCashFlowFactory klass pärib WebApplicationFactory<TStartup> klassi, millega luuakse testserver ning kus TStartup klassiks on meie ColCashFlow.Api.Startup. IntegrationTest klassis on kasutatud Respawn utiliiti, mis aitab puhastada andmebaasi andmetest ning viia selle Checkpoint'is seatud olekusse igal käivatamisel. IntegrationTest klassis luuakse HttpClient vajaliku konfiguratsiooniga, mida kasutab iga kontroller päringute tegemiseks. Helpers kaustas on Utilities klass, milles on loodud testandmed, millega täidetakse andmebaas igal testide käivitamisel. Serialization kaustas on JsonSerializerHelper abiklass, mida kasutatakse kontrollerite loogika testimisel, et teisendada C# object JSON formaati ja vastupidi [14].

Models kaustas on kontrollerite loogika testimiseks vajalikud päritud mudelid ColCashFlow.Entities projektist. Näiteks TestInitialBalance klass (Joonis 1) pärib InitialBalance kõiki Property`sid ning lisaks omab funktsiooni, millega loob uue InitialBalance objekti uuendatud väärtustega, mis tulevad kaasa parameetrina changes. Antud abifunktsiooni on kasutatud igas kontrolleri loogikat testitavas klassis, mis aitab jälgida Clean Code põhimõtteid ning mitte luua igakord uue objekti koodis, vaid kutsudes

antud funktsiooni seada uuendatud väärtuse ja saada tagasi uus InitialBalance objekt, milles oleks DataAreaId null (vt Joonis 13).

```
public class TestInitialBalance : InitialBalance
{
      public TestInitialBalance CloneWith(Action<TestInitialBalance>[]
      changes)
      {
           var clone = (TestInitialBalance)MemberwiseClone();
           foreach (var change in changes)
           {
               change(clone);
           }
           return clone;
      }
}

var initialBalance = GetValidInitialBalanceModel().CloneWith(new
Action<TestInitialBalance>[]{ a => a.DataAreaId = null});
```

Joonis 13. TestInitialBalance mudel

[Fact] attribuudiga Post_ValidInitialBalance_ReturnsCreatedResult testis, vastavalt AAA mustrile esialgu saab content muutuja väärtuse, milleks saab TestInitialBalance objekt, mille saab funktsioonist GetValidInitialBalanceModel. Teise sammuna läbi _client muutuja, mis on üles seatud IntegrationTest klassis vajaliku konfiguratsiooniga, teostatakse testandmebaasi asünkroonse POST päringu, millega kaasa antakse esimeses sammus loodud objekti ning abiklassi, mis määrab kindlaks deserialiseerimise nõuded saadetud objektile. Kolmanda sammuna testitakse Assert klassi abil, et teises sammus tehtud päringu tagastatud kood on võrdne oodatava koodiga 201, mis tähendab, et objekt oli edukalt loodud (vt Joonis 14).

```
[Fact]
public async Task Post_ValidInitialBalance_ReturnsCreatedResult()
{
       var content = GetValidInitialBalanceModel();

       var response = await
       _client.PostAsJsonAsync("api/initialbalance/createinitialbalance",
       content, JsonSerializerHelper.DefaultSerialisationOptions);

       Assert.Equal(HttpStatusCode.Created, response.StatusCode);
}
```

Joonis 14. Post_ValidInitialBalance_ReturnsCreatedResult

[Theory] attribuudiga Post_WithInvalidName_ReturnsExpectedProblemDetails testis, kontrollitakse saadud tulemusi mitme erineva sisendiga, mida saadakse klassist GetInvalidInputAndProblemDetailsErrorsValidator. Antud klassis luuakse list test objektidest, millel on ebakorrektne väärtus Property`l ning Assert klassi abil määratakse, mis Property`l ja milline viga peaks tulema (vt Joonis 15). Näiteks Joonis 16 on näha, et DataAreaId'le on pandud enam, kui 16 kirjamärki, mis ei ole lubatud ja mille tulemusel saaks viga. Edasi testklassis saadud response abil saadud viga võrreldakse oodatava veaga.

```
[Theory]
[MemberData(nameof(GetInvalidInputsAndProblemDetailsErrorsValidator))]
public async Task
Post_WithInvalidName_ReturnsExpectedProblemDetails(TestInitialBalance
initialBalance, Action<KeyValuePair<string,string[]>> validator)
{
	var response = await
	_client.PostAsJsonAsync("api/initialbalance/createinitialbalance",
	initialBalance, JsonSerializerHelper.DefaultSerialisationOptions);

	var problemDetails = await
	response.Content.ReadFromJsonAsync<ValidationProblemDetails>();

	Assert.Collection(problemDetails.Errors, validator);
}
```

Joonis 15. Test Post_WithInvalidName_ReturnsExpectedProblemDetails

```
new object[]
       {
       GetValidInitialBalanceModel().CloneWith(new
       Action<TestInitialBalance>[]{ x => x.DataAreaId =
       "ABB211245456464226"}),
              new Action<KeyValuePair<string, string[]>>(kvp =>
                     {
                           Assert.Equal("DataAreaId", kvp.Key);
                           var error = Assert.Single(kvp.Value);
       Assert.Equal("The field DataAreaId must be a string with a maximum
       length of 16.", error);
                     })
       }
```

Joonis 16. Testi sisend

Testprojekti käivitamisel täidetakse kõik selles projektis olevad testid ning seejärel kuvatakse tulemust Unit Tests aknas. Kui mõni test kukkub läbi kuvatakse ebaõnnestumise staatusemärk ning valides seda saab ülevaate, mida oodati ja mida tegelikult saadi. Testi õnnestumise puhul kuvatakse roheline linnuke.


Joonis 17. Testide ülevaade

### 3.6.2 Ühiktestid

Ühiktestid on mõeldud väikse koodiosa testimiseks ning tõestamaks, et funktsioon teostab õiget operatsiooni. Ühiktestid testivad kontrollerite meetodite päringute saatmist mockitud andmetega kasutades Moq package`i. Näiteks CashFlowController saab järgmised sisendid:

```
public CashFlowController( IOptions<ApiSettings>  appSettings,
ILogger<CashFlowController> logger, IApiUnitOfWork uow)
{
      _appSettings = appSettings.Value;
      _logger = logger;
      _uow = uow;
      var optionsBuilder = new DbContextOptionsBuilder<ApiDbContext>();
}
```

Joonis 18. CashFlowController sisendid.

Iga testklaasi jaoks oli loodud konstruktor, milles asuvad algseadistused (vt Joonis #02). Need seadistused on vajalikud baasfunktsioonidega CashFlow mocki loomiseks, mida kasutab validator. Iga kontrolleri sisendi jaoks on loodud mock ja vastav seadistus. Kõik seadistatud parameetrid on kasutatud SUT (System Under Test) lomiseks. SUT on objekt, mille abil testitakse vastav klass ja selle meetodid. Lisaks konstruktoris asuvad abimuutujad erinevate päringu staatustega.

```
private Mock<IRepository<CashFlow>> validator;
private ApiSettings api;
private Mock<IOptions<ApiSettings>> optionsMock;
private CashFlow[] cashFlows;
private Mock<ILogger<CashFlowController>> logger;
private ILogger<CashFlowController> loggerMock;
private Mock<IApiUnitOfWork> unitOfWorkMock;
private CashFlowController sut;
private int statusBadRequest, statusNotFound, statusCreated, statusOk;

public CashFlowControllerTests()
{
    statusBadRequest = 400;
    statusNotFound  = 404;
    statusCreated = 201;
    statusOk = 200;

    validator = new Mock<IRepository<CashFlow>>();

    api = new ApiSettings() { Test = "" };
    optionsMock = new Mock<IOptions<ApiSettings>>();
    optionsMock.Setup(ap => ap.Value).Returns(api);

    logger = new Mock<ILogger<CashFlowController>>();
    loggerMock = logger.Object;
    loggerMock = Mock.Of<ILogger<CashFlowController>>();

    unitOfWorkMock = new Mock<IApiUnitOfWork>();
    unitOfWorkMock.Setup(m => m.CashFlows).Returns(validator.Object);

    sut = new CashFlowController(optionsMock.Object,
        loggerMock, unitOfWorkMock.Object);

}
```

Joonis 19. Testklassi konstruktor

POST meetod CreateCashFlow saab sisendiks CashFlow objektide massiivi (Joonis #03). Kindaltel juhtumitel võib päringu tulemuseks olla veakood NotFound või BadRequest. Juhul kui massiivis olid korrektsed andmed siis andmebaasis luuakse uued CashFlow objektid ning tagastatakse staatus CREATED (201).

Eeltoodud testis kontrollitakse järgmist meetodit:

```
[HttpPost("CreateCashFlow")]
public async Task<IActionResult> CreateCashFlow([FromBody] CashFlow[]
cashFlows)
{
     if (!ModelState.IsValid) return NotFound();
     if (cashFlows == null || !cashFlows.Any()) return BadRequest();

     await _uow.CashFlows.AddRangeAsync(cashFlows);
     await _uow.SaveChangesAsync();

     var ids = new int[cashFlows.Length];

     foreach (var cf in cashFlows)
     {
         ids = new int[]{cf.Id};
     }

     return CreatedAtAction(nameof(GetCashFlow), ids, cashFlows);
}
```

Joonis 20. CreateCashFlow meetod

[Fact] attribuudiga asünkroonses testis Create_CheckOkResult testitakse kas õigete sisendite puhul oodatav ja saadud staatus on samad (Joonis #04).

Esialgu luuakse CashFlow massiiv, mille sees on korrektsed objektid. Kontrolleri kasutamiseks luuakse muutuja cashFlowRedirect, milles salvestub päringu tulemus. Assert.Equal meetodi abil kontrollitakse kas saadud staatuse kood on võrdne oodatud koodiga.

```
[Fact]
public async void Create_CheckOkResult()
{
    cashFlows = new CashFlow[]
     {
        new CashFlow
        {
            Id = 1,
            TypeId = 4,
            BalanceDate = DateTime.Parse("30-03-2021"),
            TransactionDate = DateTime.Parse("15-01-2021"),
            PaymentDueDate = DateTime.Parse("20-04-2021"),
            Amount = 20000m,
            AmountDue = 20000m,
            PartnerId = 3,
            InvoiceNr = "123AAS",
            DataAreaId = "ABB1"
        }

     };
    var cashFlowRedirect = await sut.CreateCashFlow(cashFlows);
    var result = cashFlowRedirect as CreatedAtActionResult;
    Assert.Equal(statusCreated, result.StatusCode);
}
```

Joonis 21. Create_CheckOkResult testmeetod

# 4 Analüüs ja järeldused

Antud osas tuleb analüüs tehtud tööle, meeskonnatööle, juhendamisele, ülevaade kasutatud kirjandusest ning kokkuvõtlikud järeldused.

## 4.1 Tehnilise teostuse analüüs

Rakenduse üldine struktuur ja arhitektuur oli kooskõlastatud Columbuse BI ärisuunajuhiga, arendajatega ja konsultandiga, kes määrasid töö alguses tuleviku rakenduse tehnoloogiad, raamistikud ning ka andmemudeli ehituse struktuuri. Iganädalate koosolekutel tehnilise teostamise plaani korrigeeriti vastavalt muudetavatele nõutele ja meie oskustele.

### 4.1.1 Nõuded

Nõuded olid kirjeldatud põhimõttega, et kasutaja jaoks tehtud rakendus oleks mitte ainult kasulik, vaid ka mugav kasutamises. Mugavuse tagamiseks oli otsustatud, et lõppsaldo prognoosi peaks olema võimalik vaadata mitte ainult tabelis, vaid ka visuaalselt graafikul. Lahenduse kasutamise voog peaks olema võimalikult intuitiivne. Selleks, et kasutaja ei peaks tegema oma andmetest teisendusi lahendus peaks aktsepteerima andmed mitmel erineval viisil (Excel fail, ERP süsteem või käsitsi sisestades).

Pakutav lahendus peaks olema võimalikult informatiivne pakkudes infot nii üldistatumalt kui ka detailselt. Mõned lahenduse osad ettevõttel puudusid, aga soovisid neid kasutada teistes lahendustes ka. Sellega kaasnes nõue, et need lisatavad osad oleksid võimalikult konfigureeritavad. Näiteks, ettevõttel puudus faili üleslaadimise võimalus ning ka selle andmete valideerimine.

### 4.1.2 Arhitektuur

Meeskonnaprojekti raames oleme kasutanud lokaalset MsSQL andmebaasi. Pärast edukat kaitsmist ja otsust jätkata selle projekti arendamist lõputööna oleme otsustanud andmebaasi täiustada ja kasutada mitte kohalikku versiooni, vaid ettevõtte pakutavat lahendust. Hakkasime kasutama Microsofti poolt pakutava pilveteenust andmetöötluseks Azure SQL Database. Andmebaasi serveri muutmist ka mõjutas kogu projekti integreerimine ReportÄpp'i keskkonnasse.

Oma tegevuste kavandamiseks, sealhulgas struktuuri planeerimiseks, kasutasime dbdiagram, kus lõime andmebaasi diagrammi.

Andmebaasi tabelite koostamiseks ei ole kasutatud Entity Frameworki migratsioone, vaid ettevõtte arendaja soovitusel on kasutatud traditsioonilist DAO ehk Data Access Object meetodit. See tähendab, et kõik andmebaasi tabelid luuakse manuaalselt kirjutades ise SQL käsklusi.

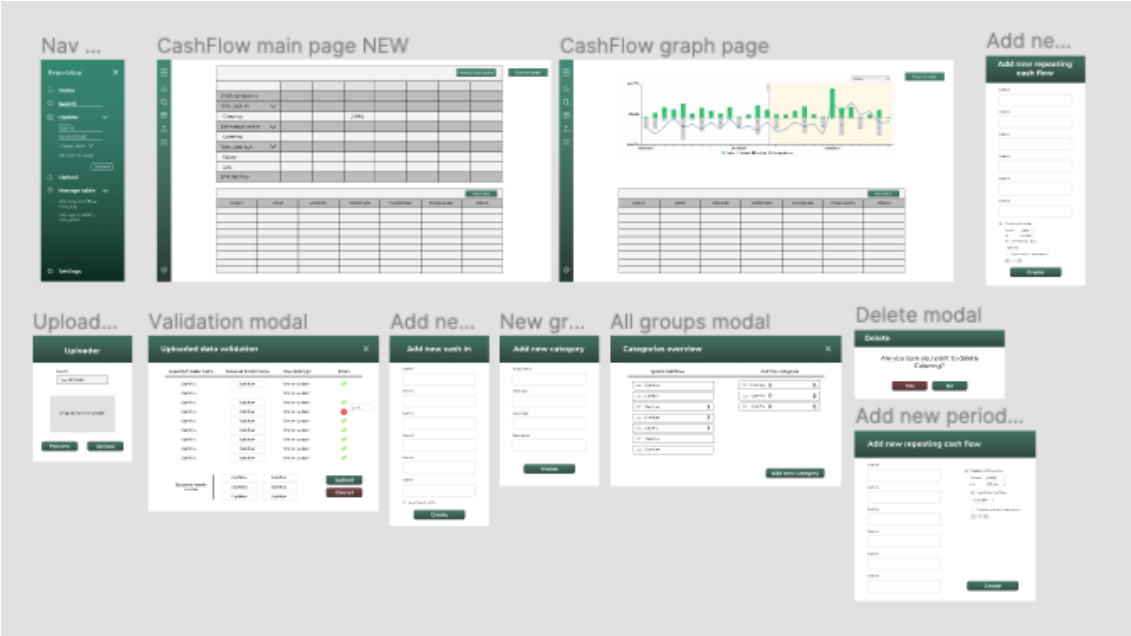
Tabelite muudatused puutusid mitte ainult tabelite ja väljade ümbernimetamist, vaid ka mõne andmetüübi muutmist ja tabelisuhete parandamist. Muutunud on ka andmebaasis olevate tabelite arv, lõpuprojekti jaoks on lisatud tabel HeaderPairs ja hetkel on see selle projekti jaoks kõige optimaalsem tabelite hulk.

Oma tegevuste kavandamiseks, sealhulgas struktuuri planeerimiseks, kasutasime dbdiagram, kus lõime andmebaasi diagrammi (vt Joonis 10).

Andmebaasi struktuuri loomiseks ja muutmiseks kasutasime DataGrip 2021.1.1 tarkvara.

### 4.1.3 Disain

Kuna meie rakendus on lisaks Columbuses kasutuses olevale ReportAppile, siis pidime sobitama oma lahenduse olemasoleva üldstiiliga. Esialgu tegime valmis Figmas lahenduse prototüübi, millest me lõpuks saime visuaalset inspiratsiooni UX'i (User Experience) kohalt ja üldistest vajalikkest nõuetest.




Joonis 22. Disaini prototüüp Figmas

Kuna meie esialgne lahendus oli täiesti sobimatu ReportÄppi stiiliga, siis otsustasime oma esimesest lahendusest loobuda. Loobumiseks oli ka mitmeid põhjuseid. Esiteks soovis meie ettevõttepoolne juht kasutada võimalikult igal pool nende loodud Tablixit ehk meie põhitabelit andmetega. Ise vaadates esialgse pilguga Tablixile mõtlesime, et saame hakkama parema ja kiirema tabeliga, kuid lõpuks oli väga hea õppekoht. Teiseks, oleme meeskonnaprojekt raames proovinud kahte Reacti raamistiku tabeli loomiseks. Alguses oli kasutuses raamistik (react-data-grid). Kuna see lõpuks ei olnud piisavat paindlik, oleme otsustanud sellest loobuda ning kasutada teist raamistiku (primereact). Õnneks lõpuks saime aru, et kolmandaosapoole tabeliga siin ei pääse ning tuleb kirjutada täitsa enda tabel, mida Columbus ka eelnevalt tegi ja pakus meile kasutamiseks. Meeskonnaprojekti kirjutamise ajal see tabel oli veel arengufaasis, mis oli üks põhjustest miks me pole kohe sellega tööd alustanud. Kolmandaks, Reacti integreerimine ReportÄppi oleks olnud mõttetu, kuna kogu rakendus on ülesehitatud läbi konfiguratsiooni ja meie Reacti osa lihtsalt ei sobiks sinna.

Valmis saadud lahendus on täiesti identne üldise ReportÄppi stiiliga.

### 4.1.4 Kood

Koodikirjutamisel oleme kasutanud agiilse arendamise meetodikaid ja jälginud Clean Code põhimõtteid. Paljud projektis tehtud lahendused on argumenteeritud üldise ReportÄppi koodikirjutamise stiiliga.

Muutujate nimetused oleme kasutanud vastavalt Martin Fowler õpetusele raamatus Clean Code [5]. Kõik nimed on konstruktiivsed ning annavad ülevaate, kus ja milleks see on kasutatud. Klasside ja funktsioonide nimetustest kasutasime *UpperCamelCase* reegleid (nt *ExcelController*) ja muutujate nimetustes *LowerCamelCase* reegleid (nt *dataAreaId*). Projektis on rangelt välditud lühikesi ja ebaarusaadavate nimetuste kasutamist.

Meie esialgses projektis oli kasutusel tavaline MVC muster, kus ühes projektis olid nii mudelid, front-end kui ka kontrollerid oma loogikaga. Kuna ReoprtÄppi back-endis on kasutatud repositooriumi mustrit, siis pidime ka enda vana API peaaegu täielikult ümber kirjutama vastavaks. Kuna varasemalt pole selle mustriga kokku puutunud, siis esmalt oli raske arusaada, kuhu mis koodiosa läheb. Lõpuks saime ilusti mustrist aru ning muutuseid saime kiiresti viia sisse. Samal ajal töötades repositooriumi mustriga hakkasime aru saama selle eelistest ning tajuma, kuidas seal on rakendatud Clean Code põhimõtted. Repositooriumi mustri kasutamise põhieelisteks on see, et see vähendab koodiduplikatsioone ning üldiste muutmiste sisse viimisel ei pea sama probleemi lahendama eraldi igas kontrolleris, vaid saab muuta Unit of Worki. Selle abil on kõik koodiosad isoleeritud üksteiselt ning see lihtsustab ühiktestimise [13]. Esialgu asusid nii back-end kui ka front-end ühes solutionis, kuid ühel hetkel otsustas Columbuse arenduse meeskond tõsta kogu back-end kood ReportÄppist eraldi asuvatesse plokidesse, mille tulemusel pidime uuesti vastavalt saadetud API mallile seda ümberkirjutama.

Kuna ReportÄppi kogu front-end on tehtud läbi konfigurstsiooni, siis esialgu abistasid meil teha algseadistused Columbuse arendajad. Saime nendelt ülevaate, millisel kujul tahab põhitabel saada andmeid ning tegime vastavad muudatused oma apis. Kuna meie esialgne tabel tegi ühe päringu ja sai kohe vastavalt filtrile summeeritud andmed, siis Tablixisse me tagastame andmed päevade kaupa ja see summerib neid ise vastavalt filtrile. Kuna Tablix ise loeb kõiki summasid, tagastame talle summad kategooriate kaupa ning ta ise teostab vajalikud tehingud nendega. Detailvaates saab ülevaate kindla

kategooria arvetest. Meie lahenduse integreerimiseks ReportÄppi täiustasid Columbuse arendajad Tablixi  struktuuri ning tegid selle veelgi paindlikumaks.

Meie esialgses lahenduses olid kõik front-endi komponendid kirjutatud kasutades TypeScripti ja Reacti. Kuna ReportApp ei kasuta neid, siis me oma vanast koodist ei kasutanud mitte midagi. Esialgu üritasime integreerida React`i ReportAppi, aga see meil ei õnnestunud. Kuna kogu visuaal seadistatakse läbi konfiguratsiooni, siis üldise stiili hoidmiseks otsustasime minna kaasa Columbuses kasutatavaga JavaScriptiga ning jQuery ja Bootstrap raamistikega. Meie poolt loodud komponendid on tehtud samuti paindlikuks ning seadistatavaks läbi konfiguratsiooni.

### 4.1.5 Testid

Kuna Columbus enamasti ei nõua arendajatelt teste tehtud lahendustele, siis nad ei nõudnud neid ka meie poolt. Kirjutatud testid ei kuulunud projekti nõuetesse ning olid tehtud, kuna lõputöö näeb ette, et tehtud lahendusel on olemas testid.

Kuna NUnit ja MSTest raamistikud on juba vananenud ja ei ole nii laialdaselt kasutuses oli valitud testide kirjutamiseks xUnit .NET testimise raamistik. Selle põhieeliseks teiste raamistike ees on selle paindlikkus. xUnit võimaldab lisada ühele ja samale testile mitu erinevat sisendid, laiem Assert klasside valik ning antud raamistik edendab puhtade ning hooldatavate testide kirjutamise [4].

Kuna oma eelnevates projektides ja kassavoo lahenduse esialgses variandi oli väga vähe teste, aga samas on väga tähtis, mida ja kuidas süsteem arvutab ja tagastab, siis otsustasime teha laiapõhjalist testimist kirjutades nii integratsiooni kui ka ühikteste. Kuna ühiktestid testivad väikest koodiosa, nagu näiteks funktsiooni õige operatsiooni teostamist ja seegi tihti mock’itud andmetega, siis otsustasime kirjutada integratsiooni teste, mis pakuvad laiapõhjalisemat testimist reaalsete objektidega ning teostades reaalse päringu test andmebaasi.

Integratsiooni testide kirjutamiseks kasutasime eraldi seadistatud HttpClient’i, millele on antud eraldi ühendus testandmebaasiga, kuhu teostatakse kõik päringud. Testide kirjutamiseks oli võimalus kasutada reaalse andmebaasi asemel in-memory-database’i, aga loobusime sellest lähtumisest. Kuna Entity Framework’i sisemise mälu andmebaas

tihti käitub vastuoluliselt relatsioonilise andmebaasi käitumisega, otsustasime kasutada päris test andmebaasi.

Ühiktestide kirjutamiseks olid kasutatud mitte reaalsed andmed andmebaasist, vaid mockid. Selline lähenemine aitab testida funktsionaalsust kergemal kujul ning turvalisemalt, sest kasutusel on võltsandmed. Mockide kasutamine võimaldab mitte kirjutada testitavad meetodid üle, vaid testida muudatusteta funktsionaalsust edastades sellele võltsandmed [12].

Kuna me kasutame API's asünkroonseid päringuid andmebaasi, tuli meil seadistada algsisendid iga testklassi jaoks. Esialgne lahendus, kus oleme proovinud luua ühine algseadistus, mida saaks kasutada igas testklassis pole sobinud meie projektisse.

Testide kirjutamisel lähtusime sammuti kogu projektis kasutatavatest nimetamise üldstiilides. Testinimi alati peegeldab, mida antud test teeb. Üldjuhul algas testinimi päringutüübiga, seejärgi tuli objektitüüp ning lõpus, mida peab test tagastama. Näiteks Delete_NoExistingId_ReturnsBadRequest() test teostab Delete päringu ning saadetav objekt on Idga, mida andmebaasis pole ja tulemuseks peab saama veakoodi 400. Antud lahendus aitab arendajatel edaspidi saada kiiresti ülevaate, mis probleem on tekkinud.

Kokku annavad testid 76% katmist kogu koodile.

## 4.2 Kirjanduse ülevaade

Projekti loomisel kasutas meeskond erinevat tehnilist kirjandus ja teadusartikle. Vajaliku kirjandust leiti O'Reilly õppekeskkonnast, samas kasutati ka Google Scholar'it. Lisaks kirjandusliketele materjalile oli läbi vaadatud erinevad õpetlikud videokursused läbi Pluralsight platvormi ning kasutatud meeskonnaliikmete praktilisi teadmisi. Lisakirjanduse kasutamise põhjusteks oli soov kirjutada kvaliteetne kood ning teha projekt professionaalsemaks ja usaldusväärsemaks, kuna seda hakkab kasutama ettevõtte pärisklientidel.

Andmebaasi arhitektuuri loomisel olid kasutatud relatsioonilise andmemudeli näiteid Erki Eessaar raamaatust "Andmebaaside projekteerimine" [1] selleks, et paremini mõista, kuidas luua antud lahendusele asjakohast andmebaasi. Mõistmiseks, et meie skoopi sobib kõikidest mudelitest kõige paremini relatsiooniline andmemudel uurisime ning

analüüsisime ka teisi raamatuid nagu Jan L. Harrington “Relational Database Design and Implementation, 4th Edition” [2].

Kuna seda tüüpi rakenduse testide kirjutamise kogemust oli vähe, vaadati PluralSight veebiplatvormil Steve Gordoni õppekursust “Integration Testing ASP.NET Core Applications: Best Practices”. Ühiktestide kirjutamiseks oli läbi vaadatud kaks kursust Jason Robertsi poolt “Mocking with Moq and xUnit” ja “ Testing .NET Code with xUnit.net: Getting Started” [6], [7], [8]. Antud õppekursused olid üldiselt väga kasuliku, aga veelgi põhjalikumaks arusaamiseks oli kasutatud Stephen D. Ritchie raamatud “Pro .NET Best Practices” [3]. Toetudes antud raamatule ja videokursuse materjalile, otsustasime kasutada xUnit testimise raamistiku ning Moq package.

Projekti käigus meeskonnaga lähtusime Robert C. Martini “Clean Code” raamatu põhireeglitest [5]. Koodi kirjutamisel oleme järginud, et koodi struktuur oleks puhas ning arusaadav, oleme minimiseerinud koodi duplitseerimist ning oleme kirjutanud sellist koodi, millest võiks kiiresti aru saada ka inimene, kus varasemalt pole selle projektiga kokku puutunud. Kõik muutujate, meetodite jms nimed on valitud niimodi, et need peegeldaks nende sisu. Näiteks meetodite nimed selgitavad mida teeb see konkreetne meetod, nimed pole eksitavad. Iga funktsioon teeb ainult ühe asja ning need on võimalikult väiksed.

## 4.3 Teostatud tööde detailne logi

Alljärgnevates peatükkides on detailne ülevaade iga projekti panustanud üliõpilasest. Ajamõõtmiseks oli kasutatud Toggl Track tarkvara ning ülevaade logidest on seisuga 18.05.2021.

Kokku oli projekti panustatud kõikide liikmete poolt 658 tundi ja 40 minutit.

- Daniel Antonov – 235 tundi ja 5 minutit.
- Ruben Galoyan – 237 tundi ja 47 minutit.
- Vladislav Jekimtsev – 174 tundi ja 58 minutit.
- Paul Opmann – ligikaudu 12 tundi

### 4.3.1 Daniel Antonov teostatud tööde logi

Table 1. Daniel Antonov teostatud tööde logi

| **Kirjeldus** | **Kuupäev** | **Kestvus** |
|---|---|---|
| kontor | 21.01.2021 | 04:40:57 |
| frontend data grid | 27.01.2021 | 06:36:51 |
| preview grid | 31.01.2021 | 02:21:00 |
| preview grid | 01.02.2021 | 02:00:00 |
| meet up | 01.02.2021 | 02:00:00 |
| draggable list | 02.02.2021 | 01:00:00 |
| | 03.02.2021 | 01:00:52 |
| validation | 05.02.2021 | 04:15:00 |
| Design | 10.02.2021 | 07:00:00 |
| Meeting | 11.02.2021 | 01:20:00 |
| Üldine ülesannete  planeerimine koos tähtaegadega | 15.02.2021 | 06:01:00 |
| Tried preview | 19.02.2021 | 05:05:00 |
| Exceli preview | 24.02.2021 | 05:58:00 |
| Andmebaas, preview | 25.02.2021 | 02:09:00 |
| Mergeing branches & azure wiki filling | 02.03.2021 | 06:03:58 |
| stand up ja epicute korrigeerimine | 03.03.2021 | 01:23:06 |
| Cashflow ja refactoring branchide Uhendamine | 03.03.2021 | 02:35:00 |
| merging branches | 04.03.2021 | 05:31:55 |

| | | |
|---|---|---|
| merging branches | 07.03.2021 | 00:59:56 |
| Kone Andresega Help, vigakohtade kirja panemine | 08.03.2021 | 00:30:00 |
| changing architecture while merging branches | 09.03.2021 | 02:01:19 |
| stand up | 10.03.2021 | 00:09:00 |
| design improvements | 10.03.2021 | 02:33:10 |
| changing architecture while merging branches | 11.03.2021 | 01:15:51 |
| validation and sql view research | 11.03.2021 | 00:42:30 |
| changing architecture fixing bugs | 13.03.2021 | 04:09:00 |
| | 14.03.2021 | 02:58:07 |
| fixing bugs w/ Andres and making db tables | 15.03.2021 | 02:07:00 |
| endpoints fixing | 16.03.2021 | 02:08:48 |
| endpoints fixing | 16.03.2021 | 02:12:54 |
| stand up | 17.03.2021 | 00:20:00 |
| fix endpoints | 17.03.2021 | 01:40:50 |
| Fix endpoints | 20.03.2021 | 01:41:21 |
| Testing | 23.03.2021 | 04:34:00 |
| Integration tests | 24.03.2021 | 01:19:31 |
| Integration tests | 24.03.2021 | 02:40:48 |
| Integration tests | 24.03.2021 | 00:46:42 |
| | 25.03.2021 | 01:55:00 |
| Testing | 29.03.2021 | 01:12:19 |

| | | |
|---|---|---|
| Testing | 29.03.2021 | 01:08:42 |
| Testing | 29.03.2021 | 02:21:05 |
| Konsultatsioon | 30.03.2021 | 00:37:00 |
| Post integration tests | 30.03.2021 | 02:27:14 |
| Integration tests | 30.03.2021 | 02:31:18 |
| stand up | 31.03.2021 | 00:11:00 |
| mvc fix try | 31.03.2021 | 00:47:18 |
| learning ef core | 31.03.2021 | 00:56:03 |
| Integration testing initialbalance and partner | 01.04.2021 | 04:08:36 |
| all views endpoint | 02.04.2021 | 02:00:00 |
| improving test endapoints | 05.04.2021 | 01:42:36 |
| meeting | 05.04.2021 | 00:40:00 |
| stand up | 07.04.2021 | 00:08:00 |
| initial balance front end | 13.04.2021 | 00:48:19 |
| initial balance front end | 13.04.2021 | 01:55:32 |
| finishing initial balance front end and made overdraft front | 14.04.2021 | 01:33:13 |
| drag and drop front | 15.04.2021 | 01:14:48 |
| drag and drop front and test | 16.04.2021 | 01:26:23 |
| drag and drop | 16.04.2021 | 01:23:59 |
| overview of db and uploader | 17.04.2021 | 03:46:51 |
| overview of db | 18.04.2021 | 00:27:05 |

| | | |
|---|---|---|
| uploader fix + gertha call + import data | 19.04.2021 | 03:10:00 |
| uploader fix + gertha call + import data | 19.04.2021 | 02:10:37 |
| detail view | 20.04.2021 | 04:19:09 |
| detail view | 21.04.2021 | 01:12:04 |
| meeting | 21.04.2021 | 01:19:00 |
| db config | 26.04.2021 | 00:12:35 |
| db config | 26.04.2021 | 01:35:40 |
| put request | 27.04.2021 | 00:35:26 |
| put request | 27.04.2021 | 02:20:00 |
| | 28.04.2021 | 04:08:32 |
| modals and endpoints for dropdowns | 29.04.2021 | 02:53:01 |
| post excel | 29.04.2021 | 01:26:04 |
| help unit test ja uued yl arutamine | 01.05.2021 | 02:09:44 |
| | 01.05.2021 | 03:54:00 |
| uploader ui | 03.05.2021 | 01:01:00 |
| uploader finished, started Income outcome for detail table | 03.05.2021 | 03:19:00 |
| improving cashflow endpoint inflow/outflow, integrating chart,js | 04.05.2021 | 03:46:44 |
| graph | 04.05.2021 | 02:55:00 |
| graph endpoint | 05.05.2021 | 01:24:12 |
| meeting | 05.05.2021 | 00:30:00 |

| | | |
|---|---|---|
| add line to graph, fix from to date, fix modal | 05.05.2021 | 03:07:13 |
| helping Vlad w/ unit test | 06.05.2021 | 01:40:00 |
| dokumentatsiin | 06.05.2021 | 01:25:00 |
| dokumentatsiin | 06.05.2021 | 02:03:07 |
| dokumentatsioon | 07.05.2021 | 01:26:15 |
| dokumentatsioon | 07.05.2021 | 01:10:02 |
| dokumentatsioon | 08.05.2021 | 01:56:00 |
| dokumentatsioon | 08.05.2021 | 01:51:08 |
| dokumentatsioon | 09.05.2021 | 01:48:00 |
| dokumentatsioon | 09.05.2021 | 01:00:00 |
| dokumentatsioon | 10.05.2021 | 01:11:09 |
| kone ylikooli juhendajaga | 10.05.2021 | 00:50:02 |
| dokumentatsioon | 11.05.2021 | 01:33:49 |
| dokumentatsioon | 11.05.2021 | 04:37:51 |
| dokumentatsioon | 12.05.2021 | 01:29:00 |
| dokumentatsioon | 12.05.2021 | 03:38:00 |
| dokumentatsioon | 13.05.2021 | 01:13:51 |
| dokumentatsioon | 13.05.2021 | 02:28:53 |
| dokumentatsioon | 13.05.2021 | 04:35:07 |
| dokumentatsioon | 14.05.2021 | 00:25:02 |
| tests | 15.05.2021 | 01:25:23 |

| | | |
|---|---|---|
| tests | 17.05.2021 | 05:21:54 |
| tests | 17.05.2021 | 02:13:16 |
| Dokumentatsioon | 18.05.2021 | 01:13:04 |
| Integratsiooni testid ja dokumentsioon | 18.05.2021 | 03:45:44 |

### 4.3.2 Ruben Galoyan teostatud tööde logi

Table 2. Ruben Galoyan teostatud tööde logi

| **Kirjeldus** | **Kuupäev** | **Kestvus** |
|---|---|---|
| | 21.01.2021 | 03:36:00 |
| | 22.01.2021 | 04:25:00 |
| | 22.01.2021 | 02:20:00 |
| Main grid, Retrospektiiv, edasine plaan | 29.01.2021 | 05:38:09 |
| Edasise plaani arutelu, exceli valideerimine, jõudluse test | 01.02.2021 | 05:52:53 |
| Trello, andmete valideerimine | 02.02.2021 | 00:39:31 |
| Andmete valideerimine | 03.02.2021 | 00:20:26 |
| Andmete valideerimine | 03.02.2021 | 02:45:30 |
| Validation | 05.02.2021 | 06:09:00 |
| Edasise plaani arutelu, andmete valideerimine | 09.02.2021 | 03:49:25 |
| Trello, üldine planeerimine | 09.02.2021 | 02:36:00 |
| Disaini prototüüp figmas (konsulteerisin Danieli) | 10.02.2021 | 01:53:22 |
| Ülesandepüstitus | 10.02.2021 | 01:36:44 |

| | | |
|---|---|---|
| Edasise plaani arutelu | 11.02.2021 | 02:20:21 |
| Üldine planeerimine | 15.02.2021 | 05:59:44 |
| Andmebaasi struktuuri läbimõtlemine ning muutmine | 17.02.2021 | 03:02:00 |
| Andmebaasi struktuuri läbimõtlemine ning muutmine | 17.02.2021 | 00:50:00 |
| Andmebaasi struktuuri läbimõtlemine ning muutmine, kasutajalood | 18.02.2021 | 01:18:00 |
| Kasutajalood | 18.02.2021 | 01:41:07 |
| Exceli preview | 24.02.2021 | 05:57:41 |
| Andmebaas, preview | 25.02.2021 | 02:09:02 |
| Projektiosade ühendamine | 02.03.2021 | 04:00:00 |
| Azure | 02.03.2021 | 03:45:00 |
| | 03.03.2021 | 00:30:00 |
| | 03.03.2021 | 00:15:00 |
| | 03.03.2021 | 01:02:53 |
| Dokumentatsioon | 05.03.2021 | 00:21:08 |
| Valideerimine | 07.03.2021 | 01:34:00 |
| Konsultatsioon | 08.03.2021 | 00:21:00 |
| Valideerimine | 09.03.2021 | 01:17:26 |
| Valideerimine | 10.03.2021 | 01:00:00 |
| Standup | 10.03.2021 | 00:09:00 |
| Valideerimine | 10.03.2021 | 01:11:00 |

| | | |
|---|---|---|
| Arhitekturi muutus | 11.03.2021 | 00:40:00 |
| Arhitektuuri muutus | 13.03.2021 | 01:25:23 |
| Arhitektuuri muutus | 13.03.2021 | 02:17:00 |
| Exceli valideerimine | 14.03.2021 | 03:02:36 |
| Arhitektuuri muutus | 15.03.2021 | 01:49:47 |
| Andmebaasi muutus, endpointide korrigeerimine, mudelite uuendamine | 16.03.2021 | 01:44:00 |
| Andmebaasi muutus, endpointide korrigeerimine, mudelite uuendamine | 16.03.2021 | 02:11:53 |
| Standup | 17.03.2021 | 00:18:00 |
| | 21.03.2021 | 02:38:57 |
| Exceli valideerimine | 22.03.2021 | 05:16:00 |
| Exceli valideerimine | 23.03.2021 | 04:50:00 |
| meie React rakenduse integreerimine ReportApp-i | 23.03.2021 | 01:10:00 |
| meie React rakenduse integreerimine ReportApp-i | 23.03.2021 | 00:25:00 |
| Exceli valideerimine | 24.03.2021 | 01:30:00 |
| Standup | 24.03.2021 | 00:20:00 |
| Exceli valideerimine | 29.03.2021 | 03:39:58 |
| Meeting | 30.03.2021 | 00:48:00 |
| Meeting | 30.03.2021 | 00:10:00 |
| Testimine, valideerimine, back-endi korrigeerimine | 30.03.2021 | 03:03:03 |

| | | |
|---|---|---|
| Testimine, valideerimine, back-endi korrigeerimine | 30.03.2021 | 00:45:09 |
| Testimine | 30.03.2021 | 00:25:00 |
| Valideerimine | 31.03.2021 | 00:48:43 |
| MVC fix | 31.03.2021 | 00:51:00 |
| Testimine, Back-endi muutmine | 01.04.2021 | 01:35:00 |
| SQL View ja back-endi muutmine | 02.04.2021 | 01:31:00 |
| | 05.04.2021 | 01:55:00 |
| Tablixi integreerimine | 05.04.2021 | 01:12:39 |
| Partner fix, tablixi jaoks API korrigeerimine, SQL View korrigeerimine | 05.04.2021 | 02:30:12 |
| API korrigeerimine, tablix | 06.04.2021 | 01:05:00 |
| API korrigeerimine, tablix | 06.04.2021 | 01:53:59 |
| API korrigeerimine, tablix | 06.04.2021 | 02:15:59 |
| API korrigeerimine, tablix | 06.04.2021 | 01:57:39 |
| API korrigeerimine, tablix | 07.04.2021 | 00:52:28 |
| Valideerimine | 13.04.2021 | 00:28:00 |
| Valideerimine | 13.04.2021 | 01:06:26 |
| Tablixi konfiguratsioon | 15.04.2021 | 00:39:00 |
| excelController fix | 16.04.2021 | 00:50:33 |
| Tablixi konfiguratsioon | 17.04.2021 | 03:39:22 |
| Tablixi konfiguratsioon | 18.04.2021 | 00:24:12 |

| | | |
|---|---|---|
| Tablixi konfiguratsioon, API korrigeerimise plaanid | 19.04.2021 | 02:48:23 |
| | 19.04.2021 | 01:48:40 |
| Azure, plaanid | 20.04.2021 | 00:40:14 |
| | 20.04.2021 | 00:55:39 |
| | 20.04.2021 | 01:29:38 |
| | 21.04.2021 | 01:22:58 |
| | 21.04.2021 | 01:06:50 |
| Meeting | 21.04.2021 | 01:18:00 |
| Nupude konfigureerimine, modaalaknad | 26.04.2021 | 01:11:00 |
| Plaaneerimine, ideed | 27.04.2021 | 00:33:00 |
| API eraldi projektisse | 27.04.2021 | 01:24:10 |
| API eraldi projektisse | 27.04.2021 | 01:10:44 |
| API eraldi projektisse, konfiguratsioon | 28.04.2021 | 05:40:06 |
| Detailvaade | 29.04.2021 | 01:06:20 |
| Detailvaade, modaalid | 29.04.2021 | 02:07:41 |
| | 29.04.2021 | 01:52:23 |
| Testid, plaan, sql view | 01.05.2021 | 02:08:03 |
| SQL view, API | 01.05.2021 | 02:55:11 |
| API, SQL View, meeting | 03.05.2021 | 03:01:56 |
| API | 04.05.2021 | 02:39:53 |
| API | 04.05.2021 | 01:50:29 |

| | | |
|---|---|---|
| DB Configuration, Graph | 05.05.2021 | 01:29:06 |
| | 05.05.2021 | 00:51:02 |
| | 05.05.2021 | 00:27:05 |
| Vigade parandus | 05.05.2021 | 00:28:11 |
| Vigade parandus, testid | 06.05.2021 | 01:30:00 |
| Vigade parandus | 06.05.2021 | 01:48:00 |
| Vahetulemuste valideerimine | 06.05.2021 | 00:32:00 |
| Dokumentatsioon | 06.05.2021 | 01:35:00 |
| Dokumentatsioon | 06.05.2021 | 02:22:36 |
| Dokumentatsioon | 07.05.2021 | 00:49:00 |
| Dokumentatsioon | 07.05.2021 | 01:44:07 |
| Dokumentatsioon | 08.05.2021 | 02:12:00 |
| Dokumentatsioon | 08.05.2021 | 01:03:00 |
| Dokumentatsioon | 08.05.2021 | 00:34:00 |
| Dokumentatsioon | 09.05.2021 | 02:10:00 |
| Dokumentatsioon | 09.05.2021 | 03:50:41 |
| Dokumentatsioon | 10.05.2021 | 01:09:05 |
| Kohtumine juhendajaga | 10.05.2021 | 00:53:00 |
| Dokumentatsioon | 10.05.2021 | 00:21:16 |
| Dokumentatsioon | 11.05.2021 | 02:42:55 |
| Dokumentatsioon | 12.05.2021 | 01:15:00 |

| | | |
|---|---|---|
| Dokumentatsioon | 12.05.2021 | 01:26:51 |
| Dokumentatsioon | 13.05.2021 | 01:20:00 |
| Dokumentatsioon | 13.05.2021 | 05:00:00 |
| Dokumentatsioon | 14.05.2021 | 00:20:00 |
| Dokumentatsioon | 14.05.2021 | 02:36:20 |
| Dokumentatsioon | 15.05.2021 | 01:15:14 |
| Dokumentatsioon | 15.05.2021 | 02:03:06 |
| Dokumentatsioon | 16.05.2021 | 00:39:42 |
| Dokumentatsioon | 17.05.2021 | 00:33:00 |
| Dokumentatsioon | 17.05.2021 | 01:58:00 |
| Dokumentatsioon, testid, refaktoorimine | 17.05.2021 | 04:43:31 |
| Dokumentatsioon, testid | 18.05.2021 | 02:20:43 |

### 4.3.3 Vladislav Jekimtsev teostatud tööde logi

Table 3. Vladislav Jekimtsev teostatud tööde logi

| **Kirjeldus** | **Kuupäev** | **Kestvus** |
|---|---|---|
| Kassavoo edasine plaan | 01.02.2021 | 01:00:00 |
| Andmete valideerimine | 03.02.2021 | 00:30:00 |
| Andmete valideerimine | 05.02.2021 | 03:05:00 |
| Edasise plaani arutelu | 09.02.2021 | 03:50:00 |
| Trello | 09.02.2021 | 02:40:00 |

| | | |
|---|---|---|
| Ülesandepüstitus | 10.02.2021 | 01:40:00 |
| Planeerimine | 11.02.2021 | 02:20:00 |
| Üldine planeerimine | 15.02.2021 | 06:06:00 |
| Preview | 24.02.2021 | 05:57:00 |
| Andmebaasi struktuuri muutmine | 27.02.2021 | 05:41:00 |
| Projekti koogiga ja dokumentatsiooniga tutvustamine | 01.03.2021 | 04:17:00 |
| Projekti koogiga ja dokumentatsiooniga tutvustamine | 01.03.2021 | 03:47:00 |
| Andmebaasi struktuuri muutmine | 01.03.2021 | 02:34:00 |
| Andmebaasi struktuuri muutmine | 02.03.2021 | 07:45:00 |
| Koosolek | 03.03.2021 | 00:15:00 |
| Andmete valideerimine | 07.03.2021 | 04:17:00 |
| Dokumentatsioon | 07.03.2021 | 01:58:00 |
| Standup | 10.03.2021 | 00:10:00 |
| SQL View | 14.03.2021 | 01:49:48 |
| Tabelite lisamine | 15.03.2021 | 04:30:00 |
| Tabelite muutmine | 16.03.2021 | 05:51:00 |
| Anmdete lisamine ja SQL View | 16.03.2021 | 01:45:45 |
| Standup | 17.03.2021 | 00:30:00 |
| SQL View muutmine | 17.03.2021 | 03:02:00 |
| Frontendi lisamine | 21.03.2021 | 01:10:09 |
| Frontendi lisamine | 21.03.2021 | 02:51:00 |

| | | |
|---|---|---|
| Konsultatsioon | 22.03.2021 | 00:15:00 |
| Front | 23.03.2021 | 01:25:00 |
| Stand-up | 30.03.2021 | 00:11:00 |
| Api | 30.03.2021 | 01:00:00 |
| Api | 30.03.2021 | 03:00:00 |
| Api | 30.03.2021 | 01:34:00 |
| View kasutamise võimaluse lisamine | 31.03.2021 | 03:54:00 |
| View kasutamise võimalus | 31.03.2021 | 00:45:00 |
| API muutmine | 01.04.2021 | 02:04:20 |
| API muutmine | 02.04.2021 | 00:39:00 |
| API korrigeerimine ja kontrollimine | 03.04.2021 | 02:00:00 |
| API korrigeerimine ja kontrollimine | 04.04.2021 | 03:00:00 |
| API korrigeerimine | 06.04.2021 | 00:45:00 |
| Unit testide selgeks tegimine | 20.04.2021 | 02:20:00 |
| Koosolek | 21.04.2021 | 01:18:52 |
| Unit testide selgeks tegimine | 21.04.2021 | 00:35:00 |
| Unit testide selgeks tegimine | 21.04.2021 | 01:17:08 |
| Unit testide selgeks tegimine | 21.04.2021 | 00:39:08 |
| Unit testide selgeks tegimine | 22.04.2021 | 00:25:00 |
| Unit testide selgeks tegimine | 25.04.2021 | 00:45:15 |
| Unit testide selgeks tegimine | 25.04.2021 | 00:21:00 |

| | | |
|---|---|---|
| Unit testide selgeks tegimine | 25.04.2021 | 00:08:54 |
| Unit testide selgeks tegimine | 25.04.2021 | 00:31:13 |
| Unit testide selgeks tegimine | 25.04.2021 | 01:28:18 |
| Unit testide selgeks tegimine | 28.04.2021 | 00:40:17 |
| Unit testide selgeks tegimine | 28.04.2021 | 02:05:31 |
| Unit testide selgeks tegimine | 29.04.2021 | 00:11:27 |
| Unit testid ja projektide üleviimine | 29.04.2021 | 02:51:29 |
| Unit testide kirjutamine | 30.04.2021 | 00:57:24 |
| Unit testide kirjutamine | 01.05.2021 | 01:48:51 |
| Unit testide kirjutamine | 01.05.2021 | 01:13:27 |
| Unit testide kirjutamine | 01.05.2021 | 03:21:10 |
| Unit testide kirjutamine | 02.05.2021 | 02:01:00 |
| Unit testide kirjutamine | 06.05.2021 | 01:26:24 |
| Bugide lahendamine | 06.05.2021 | 01:36:00 |
| Koosolek | 06.05.2021 | 00:33:00 |
| Unit testide kirjutamine | 06.05.2021 | 01:46:56 |
| Dokumentatsioon | 06.05.2021 | 02:12:03 |
| Probleemide lahendamine unit testides | 07.05.2021 | 00:38:32 |
| Probleemide lahendamine unit testides | 07.05.2021 | 00:57:00 |
| Probleemide lahendamine unit testides | 07.05.2021 | 00:27:12 |
| Dokumentatsioon | 07.05.2021 | 01:39:45 |

| | | |
|---|---|---|
| Dokumentatsioon | 09.05.2021 | 03:33:58 |
| Dokumentatsioon | 09.05.2021 | 02:45:00 |
| Koosolek juhendajaga | 10.05.2021 | 00:53:00 |
| Dokumentatsioon | 10.05.2021 | 02:34:52 |
| Dokumentatsioon | 11.05.2021 | 00:16:50 |
| Dokumentatsioon | 11.05.2021 | 03:30:27 |
| Dokumentatsioon | 12.05.2021 | 01:31:00 |
| Dokumentatsioon | 12.05.2021 | 01:25:49 |
| Dokumentatsioon | 13.05.2021 | 02:23:19 |
| Dokumentatsioon | 13.05.2021 | 01:56:55 |
| Dokumentatsioon | 13.05.2021 | 01:29:04 |
| Dokumentatsioon | 16.05.2021 | 02:25:29 |
| Dokumentatsioon | 16.05.2021 | 02:47:02 |
| Dokumentatsioon | 17.05.2021 | 02:42:10 |
| Dokumentatsioon | 17.05.2021 | 01:35:02 |
| Dokumentatsioon | 18.05.2021 | 02:49:28 |
| Dokumentatsioon | 18.05.2021 | 01:55:35 |
| Probleemide lahendamine unit testides | 18.05.2021 | 01:09:51 |
| Probleemide lahendamine unit testides | 18.05.2021 | 1:02:09 |

## 4.4 Hinnang projekti teostamise protsessi kohta

Antud peatükis tuleb üldine hinnang projektile, selle juhtimisele ning kitsaskohtadele, meeskonnatööle ning selle teostamise protsessile.

### 4.4.1 Projekti juhtimise ja teostamise protsess

Projekti juhendajaks ülikooli poolt oli Tõnn Talpsepp, kes on tarkvarateaduse instituudi vanemteadur ja ettevõtte poolt oli Kristen Pugi, BI (Business Inteligence) ärisuunajuht.

Columbuse poolt juhtisid meie tegevust BI tiimi liikmed, enamasti Triin Idunurm, kes on konsultant ning Gertha Kilgi ja Andres Sööt, kes on mõlemad arendajad. Iga kolmapäev kell 13:00 toimus meil enamasti läbi Teams kohtumine, kus rääkisime nädalaga tehtud tööst ning esitasime globaalsemaid küsimusi. Väiksemate küsimuste ja probleemide korral kirjutasime otse vajalikule inimesele.

Projekti edasi arendamisega alustasime kohe peale meeskonnaprojekti kaitsmist ehk 21. jaanuar 2021 ja tegeleme sellega endiselt, kuna soovime üleanda Columbusele nõudeid täitnud lahenduse. Esialgu leppisime ettevõttega kokku, kas nad soovivad ning lubavad meil antud projekti arendada edasi lõputööna ning millesed oleksid nende nõuded antud lahendusele.

Leppisime kohe ettevõttega, et nüüd hakkame kohe arendama projekti ReportÄppi sees, paneme kõik tabelid nende andmebaasi ning tabelina kasutame Columbuse arendajate poolt tehtud Tablixi. Kohe peale seda palusime Andresel teha meile ligipääs repositooriumile, panna meie kodused IP aadressid lubatute hulka ning luua eraldi branch, milles me hakkame töötama. Seejärel hakkasime ületõstma back-end'i ning muutma seda vastavaks üldise ReportÄppi arhitektuurile. Järgmise sammuna palusime Gerthal algseadistada andmebaasis meile Tablixi ning öelda, mis kujul peaks talle andmeid saatma. Edaspidi õppisime konfiguratsiooni ise selgeks ning saime kõike vajalike osi ise lisada. Koodi puudutavate küsimuste korral enamasti pöördusime Gertha ja Andrese poole, üldiste küsimustega Triinu pole ning spetsiifiliste küsmistega Kristeni poole. Koostöö sujus väga meeldivalt. Ettevõttepoolsed juhid ning arendajad olid tõesti suureks abiks ning oleme selle eest neile tänulikud.

Oma meeskonnas esialgu olid ülesanded jagatud nelja liikme vahel. Projekti panustamine oli maksimaalne Danieli ning Rubeni poolt, Vladislavi panus oli väiksem ning Pauli panus oli märkamatu, mille tõttu oli tehtud otsus eemaldada teda meeskonnast.

- Daniel pidi enamasti tegelma back-end koodiga ning aitama front-end koodiga. Tegelikult tegeles back-end, front-end koodiga ning integratsiooni testide kirjutamisega.

- Ruben pidi tegelema back-end koodiga ning andmebaasiga, millega tegelikult tegeleski.

- Vladislav pidi tegelema andmebaasiga ja testidega, millega ta enamasti tegeleski.

- Paul pidi tegelema front-endiga ning kuna omab kõige rohkem kogemust aitama üldiselt teisi meeskonnaliikmeid. Esialgu alustas koos Danieliga back-endi integreerimisega, aga lahkus peaaegu kohe ning järgmine kord ilmus kuu aega hiljem pooleldi tehtud valideeerimisaknaga, mida me tegelikult kasutada ei saa, kuna see oli kirjutatud Reactiga.

### 4.4.2 Hinnang projekti protsessile

Projekti teostamise käigus sujus kõik üsna hästi. Kui tekkisid kitsaskohad, saime kiiresti nõu, kas Columbuse arendajatelt või konsultantidelt.

Kõige raskemaks kohaks projektis oli Exceli andmete valideerimine. Oleme päris mitu korda seda algusest ümber kirjutanud, aga lõpuks kõik lahendused on olnud ebaefektiivsed. Kuna sellega oli palju probleeme ning see ei olnud proioriteediks otsustasime teha seda hiljem, kui kõik põhinõuded tehtud saavad. Arvatavasti oleksid kõik nõuded täidetud, kui oleks olnud üks lisa inimene, nagu oli esialgu arvestatud.

Oodatust rohkem aega kindlasti võttis väiksete front-endi elementide lisamine, kuna kellelgi ei ole väga kogemust antud valdkonnas ning ReportAppis kasutatud tehnoloogiatega, nagu jQuery pole varem kokku puutunud. Testide seadistamisega läks kauem, kui arvestasime, kuid nende enda kirjutamisega läks üsna kiirelt.

Üheks takistuseks lahenduse arendamisel oli dokumentatsiooni puudus ning arusaamatud nimed konfiguratsioonis. Selle põhjuseks on see, et ReportÄpp on veel arendamise faasis ning seda pidevalt täiendatakse uue funktsionaalsusega. Meie jaoks tähendas see, et tuli

tihedamini kohtuma ettevõtte arendajatega, kes selgitasid meile arusaamatud kohad lahti. Tablix koosneb mitmest JavaScripti failist, kus on päris suured kooditükid, millest on keeruline aru saada ilma dokumentatsioonita. Vahepeal oleme jõudnud punktidesse, et Tablix või konfiguratsioon ei saa pakkuda meile vajaliku funktsionaalsust või oleme leidnud mingi vea ja meil tuli paluda Columbuse arendajatel viia vastavad muudatused sisse.

Siinkohal peaksime me veel kord tänama ettevõtte meeskonda, kes pole kunagi jätnud meid hätta ning on alati aidanud meil ületada tekkinud takistusi.

Oodatust vähem aega võttis kogu front-end'i pool, kuna arvestasime, et seda kõike hakkame ise kirjutama, aga tegelikult sai läbi andmebaasi konfiguratsiooni kiiresti kõik vajalikud komponendid ülesseatud. Samuti üsna kiiresti saime integreeritud graafiku ning pannud api tagastama selle jaoks õiget väärtused, kuigi oleme sellega ka venitanud, kuni ei võtnud ette ja päevaga valmis saanud.

Meie enda arust läks halvasti ühe põhilise päringu kirjutamisega, mis hetkese väikse andmemahu juures võtab umbes 10 sekundid, et laadida andmeid. Oleme üritanud selle ettevõta ning parandada, aga alati tuli miskit tähtsamat vahele ning üldiselt oli soov saada kõiki nõudeid tehtud. Esimesel võimalusel üritame seda optimiseerida ning vaadata, milles seisnes probleem.

### 4.4.3 Hinnang projekti teostamisele

Üldiselt projekti tegemine sujus hästi, pole venitatud selle alustamisega ning oleme pidevalt olnud suhtluses ettevõttega. Tehniliste probleemide tekkides, saime väga kiiresti abi ettevõtte arendajatelt nii, et otseselt seisma kauaks pole kordagi projektis pidanud.

Esialgu käisime projekti tegemas Columbuse kontoris kohapeal, aga kuna COVID-19 nakatunutemäär hakkas ühel hetkel väga kiiresti tõusma, otsustasime töötada kodunt. See sujus meil enam vähem hästi, ainuke probleem oli, et ei olnud näha, millega konkreetselt keegi tegeleb, aga selle lahendamiseks hakkasime töötama koos läbi Discordi suhtlusplatvormi.

Kõige suuremaks kitsaskohaks projektis oli meeskonnatöö, mis ei sujunud sugugi oodatavalt. Esialgu olid kõik väga motiveeritud, aga see kadus üsna kiiresti ja päris harva oli osadelt meeskonnaliikmetelt üldse näha huvi projekti vastu, mitte räägitud

panustamisest reaalse tööga. Üldiselt terve projekt on tehtud Danieli, Rubeni ja osaliselt Vladislavi poolt. Daniel ja Ruben on olnud pidevas suhtluses nii omavahel kui ka ettevõttega. Vladislavil oli selle võrra raskem, et ta ei olnud meiega meeskonnas algusest peale ning vahepeal nakatus Covid-19'sse ning ei olnud füüsiliselt võimeline kaasa aitama, mille tõttu jäi vahepeal maha ning pärast pidi uuesti süvenema protsessi.

Esialgu kõik kirjutasid omaette koodi, aga saime kiiresti aru, et selline lähenemine ei tööta ning hakkasime kasutama paarisprogrammerimist. Suhtlemiseks kasutasime Discordi, mille kanalites sai rääkida, jagada ekraani ning teha seda mitmekesi. Antud lähenemine väga aitas meid juhul, kui jäime kuskil seisma või kui oli vajadus konsulteerida üks teisega, kas saime püstitatud ülesandest õigesti aru.

## 4.5 Meeskondlik hinnang

Kassavoo lahenduse projekti teostamisel iga liikme panust arvestatakse hindega „-2" kuni „+2", kus „-2" on halvim hinne ehk tähendab, et ei panustanud projekti põhimõttelist üldse. Meeskonnaga oli tehtud otsusel jagada hinded järgmiselt:

- Daniel Antonov: „+1" (panustas projekti maksimaalselt),
- Ruben Galoyan: „+1" (panustas projekti maksimaalselt),
- Vladislav Jekimtsev: „0" (erinevatel põhjustel ei saanud panustada piisavalt),
- Paul Opmann: „-2" (projekti panustamine oli olenematu)

# 5 Kokkuvõte

Projektiks oli ettevõtte rahavoo juhtimise lahenduse loomine Columbus Eesti AS-ile. Põhiprobleemi, mida rakendus aitab lahendada on tagada rahavoogude ehk likviidsuse igapäevast juhtimist. Projekti eesmärgiks oli saavutada kassavoo juhtimise lahendus, kus on infot prognoositavatest sissetulekutest ja väljaminekutest nädala või kuu kaupa ja saada perioodi saldo tasakaalu.

Erinevalt meeskonnaprojektist oleme saanud julgemaks ning esitanud pidevalt küsimusi. Ettevõtte meeskond on olnud suureks toeks kogu projekti raames. Oleme saanud esmase väga hea kogemuse, kuidas töötada eelnevalt arendatud projektiga ehk suure koodibaasiga.

Töö tulemuseks on ReportÄppi lisa, mis aitab kasutajatel tagada parema ülevaade oma rahavoogudest ning hoida kulusid kontrolli all. Lahendusel on veel arenguruumi, osa funktsionaalsusest ei tööta plaanijärgselt ning väike osa nõuetest on veel puudu, kuid rakendus tänasel hetkel on kasutuskõlblik.

Analüüsides meie esialgset lahendust saame aru, et alguses oleme liikunud vales suunas. Üritasime kõike teha nullist ja ei võtnud eelist kasutamaks valmis komponente, mis oleksid kõvasti lihtsustanud meie tööd ja oleks parem Columbusele kasutamiseks, kuna lahendus oleks algusest peale integreeritud õigesse kohta. Jõudsime järeldusele, et kriitiliseks kohaks on meeskond, kus kõik on motiveeritud töötama.

# Kasutatud kirjandus

[1] E. Eessaar, Andmebaaside projektreerimine. Tallinn: TTÜ kirjastus, 2008

[2] J. L. Harrington, Relational Database Design and Implementation, 4th Edition, Burlington: Morgan Kaufmann, 2016. [E-book]. Loetud aadressil: https://learning.oreilly.com/library/view/relational-database-design/9780128499023/ Kasutatud: 12.03.2021

[3] Stephen D. Ritchie, *Pro .NET Best Practices*, New York: Apress, 2011. [E-book]. Loetud aadressil: https://learning.oreilly.com/library/view/pro-net-best/9781430240235/ Kasutatud: 01.03.2021

[4] Rami Vemula, *Real-Time Web Application Development: With ASP.NET Core, SignalR, Docker, and Azure*, New York: Apress, 2017. [E-book]. Loetud aadressil: https://learning.oreilly.com/library/view/real-time-web-application/9781484232705/ Kasutatud: 23.04.2021

[5] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, London: Pearson, 2008. [E-book]. Loetud aadressil: https://learning.oreilly.com/library/view/clean-code-a/9780136083238/ Kasutatud: 10.05.2021

[6] S. Gordoni, "Integration Testing ASP.NET Core Applications: Best Practices", 2020. [Online]. Loetud aadressil: https://app.pluralsight.com/library/courses/integration-testing-asp-dot-net-core-applications-best-practices/table-of-contents Kasutatud: 23.03.2021

[7] J. Roberts, "Mocking with Moq and xUnit", 2020. [Online]. Loetud aadressil: https://app.pluralsight.com/library/courses/mocking-moq-xunit/table-of-contents Kasutatud: 23.04.2021

[8] J. Roberts, “Testing .NET Code with xUnit.net: Getting Started”, 2019. [Online]. Loetud aadressil:https://app.pluralsight.com/library/courses/dotnet-core-testing-code-xunit-dotnet-getting-started/table-of-contents Kasutatud: 26.04.2021

[9] K. Pugi, “Kassavoo juhtimine”, Columbus, Tallinn, Eesti, 2020

[10] Columbus, *Digitaalne muutus, kasv ja tulevikukindlus Sinu ettevõttes*, 2021, [Online]. Loetud aadressil: https://www.columbusglobal.com/et/ettevottest Kasutatud: 12.01.2021

[11] Commercial Capital LLC, *12 Cash Flow Problems and Solutions*, 2014, [Online]. Loetud aadressil: https://www.comcapfactoring.com/about-us/ Kasutatud: 07.05.2021

[12] *unittest.mock — mock object library, version 3.3.*, The Python Standard Library [Online]. Loetud aadressil: https://docs.python.org/3/library/unittest.mock.html Kasutatud: 25.04.2021

[13] M. Spasojevic, “ASP.NET Core Web API – Repository Pattern”, Code Maze [Online]. Loetud aadressil: https://code-maze.com/net-core-web-development-part4/amp/ Kasutatud: 06.05.2021

[14] *Integration tests in ASP.NET Core*, Microsoft ASP.NET Core Documentation [Online]. Loetud aadressil: https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-5.0 Kasutatud: 11.05.2021

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

Meie, Ruben Galoyan, Daniel Antonov ja Vladislav Jekimtsev

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Columbus Eesti AS Kassavoo juhtimise lahenduse loomine“, mille juhendaja on Tõnn Talpsepp
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Oleme teadlikud, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autoritele.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

18.05.2021

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 – Vladislav Jekimtsevi eneseanalüüs

Meeskonna liitumisega esialgselt tutvusin üldise projekti struktuuriga ja kõikide dokumentidega. Uurisin mida teeb konkreetne klass ja selle meetod. See oli minu esimene kogemus, kus oli vaja liituda olemasolevale projektile ja iseseisvalt uurida kõik kättesaadav materjal, mis oli kirjutatud meeskonnaprojekti raames 2020 aasta teise poolel. Minu arvates oli töövoo juurutamine liiga aeglane, kuna puudus korrektne tehniline dokumentatsioon, samuti projekti üldine kirjeldus. Pidin sageli küsima meeskonnaliikmetel, milline failihunnikus kirjeldatud info on praegu asjakohane ning milline mitte.

Täielik süvenemine projekti sisse juhtus alles veebruaris, kuna jaanuari lõpus otsustati, et mina liitun selle projekti juurde ja projekti arendatakse kuni lõputööni nelja liikmelises meeskonnas. Kui alustasin oma tööülesandega ise toimele siis arusaamine koodist tõstis ja edasi oli lihtsam arendada olemasolevad meetodid. Samal ajal sain tuttavaks Columbuse BI osakonna juhendajaga, konsultanditega ning arendajatega. Kellega meil toimusid iganädalased koosolekud, kus saime arutletud jooksvaid asju ja teemasid. Kuna projekti alguses ma olin juba paar kuud töötanud osakoormusega sama ettevõttes teises osakonnas majandustarkvara nooremarendajana rollis, oli füüsiliselt mugav küsida teiste inimeste abi või arvamus.

Veebruaris alustasin andmete valideerimise loogika lisamisega. Selle jaoks esimesena uurisin kas on juba olemas tehtud lahendused MVC arhitektuuri jaoks või mitte. Analüüsides mitu artikli ja näiteprojekti erinevatel foorumitel sain teada, et olemasolevad lahendused ei sobi meie projekti raames ning seda on vaja kirjutada täitsa algusest, otsides eraldi lahenduste näited. Kuna sel hetkel valideerimine ei olnud prioriiteedis, siis alustasime tegeleda olulisemate ülesandetega. Avastasime, et on vaja kogu projekt integreerida ReportÄppi ja selle tõttu andmete valideerimine jäi pooleli ehk tuleviku probleemiks.

Projekti integreerimisel minu esmase ülesandena oli andmebaasi struktuuri ümberkirjutamine vastavalt uuetele tingimustele. Meeskonnaprojekti raames oli loodud lokaalne andmebaas, mis tegelikult vajas ümbermõtlemist ja parenemist. Lisaks see oli

MsSQL andmebaas, mida oli keeruline kasutada ReportÄppis, see võis tuua konfliktid andmete sünkroniseerimisel ja kasutamisel teiste admetedega. Pärast põhjaliku arutlemist BI tiimi arendajaga, oleme valinud seeda teed, et alustame kasutada Azure SQL Database Columbuse poolt tehtud serverid. Esialgu muutsime andmebaasi struktuuri dbdiagram.io mudelis, arutledes koos milline võiks olla parim variant. See oli vajalik samm selleks, et reaalse andmebaasi muutmisel ei tekiks vigu ega konflikti. Andmebaasi osa oli minu vastutusel. See osa oli minu jaoks kõige huvitavam, kuna tabelite ümberkirjutamisel kasutasin saadud teadmised Andmebaasi kursusest ülikoolis, oma teadmised töö juures ning ka lisaks kasutasin Tarkvara teaduse instituudi dotsenti Eerki Eessaar „Andmebaaside projektreerimine“ abimaterjalid. Uute tabelite looomisel otsustasime koos juhendajaga kasutada DAO meetodit, mida olin juba praktiseerinud teiste meeskonnaprojektide raames. Andmbebaasi struktuuri muutmine puudutas kõikide tabelite ja väljade nimetused, adnmetüübide ja tabelisuhete parandamine ning ka lisatud uus tabel. Kuna töökäigus olid ikka lisa täpsustused ja märkused Columbuse poolt, tegin ka väikesed muudatused töö protsessis. Lisaks kogu andmebaasi struktuuri muutmiseks olin teinud eraldi SQL View GridData jaoks. See View oli vajalik olemasoleva SQL Join’i asendamiseks, kuna see Join, mis oli kirjutatud meeskonnaprojekti ajal, nüüd meie projekti jaoks ei sobinud ja sellele pidi leidma alternatiiv, mis vastaks Entity Framework raamistikule. Selle töö osas sain rakendada oma teadmised ehk minu arvates olin teinud hea kvaliteediga struktuuri, mida saab kasutada ning täiendada tulevikus. Pärast andmebaasi struktuuri loomist Azure SQL Database serveris, olin ka selgitanud teiste meeskonnaliikmetele millised muudatused on tehtud. Arutlesime koos, mida on vaja lisada API mudelites ja kontrollites, vastavalt uuele andmebaasi struktuurilie ning kuidas saaks efektiivsemalt kasutada loodud View.

Siis kui andmebaasi osaga oli kõik lõpuni tehtud ja kontrollitud mitu korda, alustasin uurida, kuidas saaks React komponentid meeskonna projektist integreerida ReportÄpp’i sisse. Kuna see teema oli minu jaoks vähem tuntud, kui SQL ja Back-end, seega otsustasin esialgu küsida abi Paul’il, kellel sellega oli rohkem kogemust. Aga tegelikult saime teada, et parem üldse mitte kasutada React meie projekti raames. Järgmisena muutsin API’s olevate kontrollerite atribuutid  ning kontrollisin kõikide meetodite päringud kasutades Postman’i rakenduse. Aprilli alguses kahjuks sain nakkatunud Covid-19’ga ja suurem osa sellest kuust füüsiliselt oli võimatu teha tööd. Siis, kui sain terveks mul läks mõni aeg veel kord projekti süvenemisele, kuna meeskonnaliikmed on sellel ajal üritanud kasutada

eraldi solution'it ning viisid kõik muutused projekti sisse. Lisaks seadistati tablixi ja kirjutati integratsiooni testid.

Pärast detaalse uuringu sellest, mis oli tehtud selle perioodi jooksul, alustasin unit testide kirjutamist. Kuna minu jaoks testimine on päris raske teema, sest polnud piisavalt teadmisi ja oskusi projektide tegelemisel nendega, alustasin selle teema süvenemisega. Vaatasin Pluralsight veebikeskkonnas kursust Mock'ide kohta, sest meeskonnaga otsustasime, et hakkame kirjutama unit testid XUnit'i abiga ning järelikult mock'ide teemat ka pidi selgeks teha. Selle videokursuse abil lõin test projekti, kus sain selgeks algteadmised mock'ide kohta ehk kuidas neid kirjutada, kasutada ja miks just neid. Kuna see kursus oli ainult sissejuhatus unit testide maailma sisse, olin otsustanud läbida sama autori teist kursust, kus oli konkreetsed näited sellest, kuidas saab kirjutada unit testid .NET raamistiku jaoks. Samuti olin lugenud abimaterjalid unit testide teooriaga. Kuna kursused andsid pigem praktilisi oskusi mitte teadmisi sellest, kuidas saaks neid kasutada meie projekti raames. Mina võiks öelda, et parem oli alustada testi kirjutamise õppimisega varem, kuigi see polnud võimalik. Või siis läbida eraldi kursus ülikoolis, sest ilma nendeta ei ole võimalik hakkama saada. Unit testide kirjutamisel esimese väljakutsena oli seadistus ehk teha vajalikute sisendite jaoks seadistatud mock'id. Test meetodite kirjutamine tundus meeldiv ja aruasaadav hetkeni, kui sain silmitsi probleemiga asünhroonsete meetodite testimisega. Pärast esimese kontrolleri eduka testimist leidsin teiste kontrollerite meetodites koha, mille testimisele kuulus päris palju aega. Selle probleemi lahendamiseks küsisin abi nii meeskonnaliikmetel, kui ka Columbuse arendajal, aga kahjuks täielikult see mure ei saanud lahendatud. Pärast osade testide valmis saamisest alustasime koos dokumentatsiooni täiendamisega, kus mina detaalselt kirjeldasin enda töö osad ning teised vajalikud peatükkid.

Antud projekt oli minu jaoks kasulik ning väga silmaringi laiendav. Sain endale suurepärase võimaluse kasutada minu olemasolevad teadmised ning samuti nägin konkreetse suunda, kuhu saan veel edasi minna ja osukused arendada. Mul oli väga meeldiv töötada sõbralikus meeskonnas, kus kõik gruppi liikmed olid abivalmid ja töökad. Eraldi tahan tänada Columbuse töötajaid kiire ning aktiivse koostöö eest.

# Lisa 3 – Daniel Antonovi eneseanalüüs

Kassavoo lahenduse arendamisega alustasin veel septembrikuus meeskonnaprojekti raames. Kogu selle aja vältel, kuni lõputöö kaitsmiseni, olen olnud aktiivne meeskonnaliige ning pidevalt tegelenud ning olnud teadlik, mis seisus projekt on.

Kohe peale meeskonnaprojekti kaitsmist alustasime antud lahenduse edasi arendamisega, aga esialgu suurema, nelja liikmelise meeskonnaga. Esimese nädala jooksul peale kaitsmist üritasime oma lahendust veel loogilise lõpuni arendada, kus üritasime teha oma esialgne tabel paindlikumaks, teha Exceli faili eelvaate ning arutasime, kuidas saaksime luua valideerimise. Kui lõpuks saime aru, et nii ei jõua me kuskile, kohtusime ettevõttepoolsete juhendajatega, et läbi rääkida kõik detailid ning arendada nende jaoks õige ning vajalik lahendus. Selleks, et pakkuda juhendajatele parema ülevaate tulevasest lahendusest, tegin Figmas valmis kassavoo lahenduse interaktiivse disainprototüübi. Kui kõik kohad said üleräägitud ning kokkulepitud alustasime arendamisega. Õnneks kogu ReportÄpp'i lahendus on back-end'is tehtud kasutades C# programmeerimiskeelt ning ASP.NET Core raamistiku, mis tegi kogu töö veidi kergemaks ja arusaadavamaks, kuna ülikooli õpingute ajal oli see meie põhikeeleks.

Kuna esialgne projekt oli tehtud eraldi, aga Columbus soovis saada alamlahenduse kasutuses olevale ReportÄpp'ile ning kasutas teisi tehnoloogiaid, otsustasime sellest loobuda ning kasutada ainult vajalikud koodiosad sealt. Esialgu alustasin koos Pauliga meie projekti integreerimisega ReportÄpp'i, aga lõpuks pidin Rubeni abiga lõpuni tegema, kuna selleks ajaks oli Paul juba kadunud. Kuna ReportÄpp'i API's oli kasutatud repositooriumimuster, millega ise pole eelnevalt kokku puutunud, siis esialgu võttis oma jagu aega sellega tutvumiseks. Teiseks kitsaskohaks minu jaoks oli suur olemasolev koodibaas, kus esialgu oli raske arusaada, mis ja milleks on. Lõpuks mõistsin, et kõik on tegelikult väga loogiliselt ülesehitatud ning projekti edasises käigus oli kõike väga mugav hallata ja seal orienteeruda. Lõpuks, kui sain meie esialgse branch'i integreeritud ReportÄpp'i, seletasin teistele meeskonnaliikmetele üldsõnaliselt uuest arhitektuurist, kus ja milleks on. Integreerimise käigus on lõpuks mind aitanud Ruben bug'ide lahendamisega. Järgmise sammuna koos Rubeniga parandasime API mudelid ning kontrollerid vastavaks uutele andmebaasi tabelitele.

Kuna sain aru, et pole piisavalt hea testide kirjutamises, otsustasin enda ülesandeks võtta testimise. Kuna eelnevalt olen kirjutanud ühikteste ning kasutanud Moq package'i võltsobjektide loomiseks, otsustasin kirjutada esmakordselt integratsiooni teste, mille eesmärgiks on testida mitu erinevat koodikomponenti koos. Õppimiseks nende kohta kasutasin Pluralsight'i kursust „Integration Testing ASP.NET Core Applications: Best Practices" ning edaspidi kirjutamisel kasutasin Microsofti dokumentatsiooni, milles oli kõik päris selgelt kirjutatud. Siin kohal oli otsustatud luua testandmetega andmebaasi, kuhu hakatakse teostama testpäringuid, et tulemus oleks maksimaalselt tõepärane. Esialgu oli kõige raskemaks kohaks testkeskkonna ülesseadistamine, millele mul läks päris mitu täispäeva, aga edasi hakkas mulle väga meeldima testidekirjutamine. Sain väga hästi nendest aru ning viimaste kontrollerite testimisel tuli välja kirjutada teste, mis minu arust väga näiliselt seletavad Clean Code põhimõtteid, millest meile on räägitud ja õpetatud kolme aasta jooksul. Testide kirjutamisel oleme avastanud meie endpoint'ides mitmeid bug'e ja vigu ehk jällegi näitab, kui oluline ning kasulik on kirjutada teste. Vahepeal üritasin ka Vladislavi aidata ühiktestide kirjutamisega, aga enamasti tal olid üsna rasked probleemid, mille lahendamiseks antud hetkel aega ei olnud. Järgmises oma projektis tahan proovida kasutada Test Driven Development meetodikat ehk esialgu kirjutan testi ning seejärel API endpoint'i, kuna alles nüüd sain aru, kui olulised on testid ja kuidas nad kergendavad meie elu.

Selleks hetkeks ei olnud meil ühtegi visuaalset lahendust implementeeritud front-end'is ning kõik oli seotud back-end arenduse ja testidega ning kuna meie juhendajatel hakkas juba tekkima küsimusi, et millal tuleb, hakkasime tegelema front-end'iga. Esialgu tegin modaali ise, mitte läbi konfiguratsiooni nagu lõpuks pidi tegema, et näitada kuidas töötab Exceli üleslaadimine, arvete ja kategooriate lisamine ning kustutamine. Kuna esialgses lahenduses oli kasutatud Reacti component faili üleslaadimiseks, pidin selle algusest looma JavaScrpti ja jQuery abiga, kus viimane valmistas raskusi, kuna pole kunagi kokku puutunud. Seejärel õppisime koos Rubeni ja Gerthaga (Columbuse arendaja) kasutama konfiguratsiooni, mille kaudu sai seadistada front-end'i. Esialgu oli natukene hirmus teha seal muudatusi, kuna kõik tundus raske ja kuna antud konfiguratsioonis olid seadistatud andmed reaalsetele ettevõtetele, kes kasutavad ReportÄppi. Ajaga õppisime kõik meile vajalikud tabelid ära ning julgelt seadistasime enda modaalid ja dimensioonid nendele. Vajadusel lisasime konfiguratsiooni JavaScripti faili kooditükke, mis võimaldasid

seadistada meile vajalikud sisendid või kuvatavad detailid, millega just mina enamasti tegelesingi.

Seejärel hakkasime koos Rubeniga tegelema detailtabeliga, kus esimeseks ülesandeks oli API endpoint'i ümberkirjutamine. Mina esialgu kirjutasin valmis selle mudeli ja päringu, et see töötaks, aga lõpus Ruben parandas natukene selle loogikat. Hiljem koos Rubeniga tegime vastavad seadistused andmebaasi konfiguratsioonis õigete andmete kuvamiseks detailtabelis.

Tähtsaks ülesandeks oli graafiku loomine. Otsustasin kasutada selle loomiseks Chart.js JavaScript'i raamistiku, kus oli kõik vajalik funktsionaalsus. Esialgu aimasin, et see on raske ülesanne ning oleme natukene viivitanud sellega, kuid lõpuks sain selle kiiresti integreerinud ReportÄppi, mille järel ka kirjutanud kontrolleri, mis tagastab konkreetsed väärtused, mida graafik enda joonistamiseks vajab. Seejärel kirjutasin ka antud päringule integratsioone teste.

Vahepeal olen tegelenud ka väiksemate jooksvate ülesannetega nagu teiste meeskonnaliikmete aitamine, konsulteerimine ettevõttepoolsete juhendajatega ja vanemate pisidetailide parandamisega.

Olen ise natukene pettunud meie esialgses meeskonnas ning selle liikmete valikus. Sain projekti jooksul aru, kui tähtis on meeskond, selle ühine motiveeritus ning sarnane visioon. Vahepeal tundsin ükskõiksust projekti vastu osadelt kaaslaselt, mille tulemusel otsustasime ka eemaldada ühe esialgse meeskonnaliikme. Sellelgi on omad plussid, kuna sain ise palju rohkem teada, kui esialgu arvasin, et õpin.

Üleüldse olen väga tänulik Columbus Eesti AS'ile ja kogu BI meeskonnale sellise võimalusele arendada lõputööna midagi tõeliselt vajaliku ning huvitava. Projekti integreerimise ning kirjutamise ajal olen saanud võrratus koguses erinevat kogemust, nii töötamisel reaalse ning suure koodibaasiga, kui ka teiste programmeerimises vajalike teadmisega. Tahaks eraldi tänada BI ärisuunajuhti Kristen Pugi, arendajaid Andres Sööt ja Gertha Kilgi ning konsultanti Triin Idunurme.

# Lisa 4 – Ruben Galoyani eneseanalüüs

Olen arendanud Kassavoo lahendust juba meeskonnaprojekti raames. Peale edukat kaitsmist oleme püüdnud meeskonnaga viia projekti punktini millal saaks öelda, et lahendus on kasutuskõlblik ning peamine osa funktsionaalsusest on valmis saanud. Tegelikuses saime aru, et oleme jõudnud tupikusse.

Üldiselt olen tegelenud back-end koodiga, konfiguratsiooniga ning natuke andmebaasiga.

Oleme kohtunud ettevõttepoolse juhiga ning on alanud ettevalmistusfaas. Arutasime tehtud tööd ning mõtlesime selle peale, millised võiksid olla järgmised sammud selle projekti teostamisel. Selle protsessi olid kaasatud kõik meeskonnaliikmed. Analüüsides meie lahendust saime aru, et meie tabelit pole võimalik seadistada vastavalt nõuetele ning oli arusaadav, et me peaksime loobuma meie tabelist ning võtma kasutusele ettevõtte poolt arendatud tabelit. Oli tehtud otsus, et edasi tööd jätkame kasutades Columbuse andmebaasi ning integreerime meie rakendust ReportÄppi, kus jätkame lahenduse arendamist. Selle etapi jooksul Daniel on teinud disainiprototüübi Figmas, millega olen teda assisteerinud. Meeskonnaprojekti raames oleme pannud kirja kõik ülesanded Trellosse, kuid ettevõtte palus, et me paneks kõike kirja nende Azure'i. Jooksvalt olen tegelenud Excelist saadud andmete valideerimise kirjutamisega.

Peale ettevalmistuse faasi jätkasin täies hoos meie rakenduse arendamisega. Olen lõpuni kirjutanud valideerimise esialgset varianti. Tol hetkel see pole täielikult täitnud püstitatud nõudeid ning seda tuli hiljem täiendada. Meie lahenduse integreerimisel ReportÄppi selgus, et selles on kasutusel teine arhitektuur ning meil tuleb ümber kirjutada mõned kohad API's, sellega tegeles enamasti Daniel, kuid olen aidanud temal seda teha. Meeskonnaprojekti kaitsmisel saime tagasisideks, et meie andmebaas vajab ülevaatust. Olen andmebaasi arhitektuuri läbi vaadanud ettevõtte arendajaga ning korrigeerinud seda diagrammil, et kui see peaks valmis saama vastavad muudatused saaks teha juba andmebaasis. Vladislavil ja Paulil on tekkinud raskusi meie front-endi integreerimisega ReportÄppi, Paul kadus ning mul tuli aidata seda teha. Kahjuks pole see meil õnnestunud ning analüüsides olukorda tulime järeldusele, et keeldume meie loodud front-endist ning võtame täielikult kasutusele ettevõtte poolt pakutavad lahendused (visuaali baasid).

Meie rakenduse integreerimine ReportÄppi võttis oodatust rohkem aega ning alguses oli keeruline suurest olemasolevast koodihulghast aru saada. Hiljem saime sellega hästi tuttavaks ning toimetada sellega oli palju kergem. API integreerimise käigus aitasin Danielil tuvastada tekkinuid vigu ning parandada neid. Umbes selleks ajaks Vladislav jõudis lõpuni muuta andmebaasi struktuuri ning Danieliga koos muutsime mudelid ja kontrollerid sellega vastavusse.

Järgmise sammuna alustasime visuaali ülesehitamisega. Esimeseks sammuks oli peamine tabel, mida aitas seadistada ettevõtte arendaja Gertha. Gertha tutvustas meile konfiguratsiooni ning näitas kuidas teha soovitud muutused. Alguses oli natuke hirmus ning oleme palunud, et Gertha meid kontrolliks. Hiljem saime konfiguratsioonist hästi aru ning julgesime teha soovitud muutusi ise. Vahepeal selgus, et vajalik funktsionaalsus konfiguratsioonis puudub ning oleme palunud seda lisada. Näiteks, ettevõttel puudus võimalus seadistada navigatsioonimenüüs olevad nupud ning oleme palunud ettevõtte arendajatel lisada sellist võimalust. Kui vastavad muudatused said tehtud seadistasin kõike konfiguratsioonis. Üldiselt muutusi konfiguratsioonis oleme teinud enamasti Danieliga.

Üks minu põhiülesannetest oli Exceli valideerimine. Selles osas olen olnud kontaktis ettevõtte arendajaga ning olen katsetanud mitme erineva variandiga. Kahjuks hetkel lahenduses kasutatav variant ei täida kõiki püstitatud nõudeid. Samuti olen suhelnud ettevõtte arendajaga päringust mis tagastab andmed peamise tabeli jaoks. Tema soovitas luua SQL View ning kasutada seda API’s. Olen aidanud Vladislavil aru saada sellest ning aitasin selle kirjutamisel ning kasutusel koodis.

Jooksvalt selgus, et meil tuleb midagi API’s muuta ning tihtipeale olen neid muutusi teinud kas ise või Danieliga koos. Ühel hetkel ettevõtte on otsustanud, et kõik API’d peavad olema eraldi projektides ning olen meie koodi üleviinud eraldi projekti ning muutnud konfiguratsioonis päringuid, et need oleksid vastavuses uue seadistusega. Detailvaate osas olen aidanud Danielil parandada API’s loogikat ning seadistasime temaga koos konfiguratsioonis selle kuvamist.

Teste ma pole ise kirjutanud, kuid olen aidanud nii Vladislavil, kui ka Danielil neid seadistada ja kirjutada. Testide kirjutamisel oleme tuvastanud vigu meie koodis ning olen aidanud neid parandada.

Projekti käigus olen alati aidanud teistel meeskonnaliikmetel ületada tekkivad takistused ning kui me ei saanud ise hakkama siis alati pöördusime ettevõtte kogenud arendajate poole. Ettevõtte meeskonnaga olin alati pidevas kontaktis. Iga nädal kolmapäeviti oleme kohtunud ning arutanud projekti hetkeseisu ning arutanud edasised plaanid. Koodi kirjutamisel olen alati järginud puhta koodi printsiipe ning agiilse arenduse metoodikaid.

Ütleksin, et minule suurimaks takistuseks oli töötamine mitte enda projektiga, vaid juba eelnevalt arendatud kellegi teise poolt. Mul pole kunagi sellist kogemust olnud, eriti konfiguratsioonuga, läbi mille saab seadistada päris paljud asjad. Kitsaks kohaks võiksin nimetada ka majandusmaailma nüanse, millest tuli aru saada selle projekti teostamise käigus. Enamik takistustest sai ületatud ning olen õpinud palju uut.

Olen väga tänulik saadud kogemuse ning meeldiva koostöö eest nii enda meeskonnaliikmetele, kui ka ettevõtte meeskonnale.