TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karl-Erik Karu 164477IABB

ETTEVALMISTUSED LABORI INFOSÜSTEEMI UUENDAMISEKS JA EDASIARENDAMISEKS

Bakalaureusetöö

Juhendaja: Gunnar Piho
PhD

Tallinn 2019

Autorideklaratsioon

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

Autor: Karl-Erik Karu

05.05.2019

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on teha ettevalmistused varasemalt arendatud labori infosüsteemi uuele platvormile üleviimiseks ning edasiarendamiseks.

Labori infosüsteemi (LIMS) uuendamise eesmärk on välja vahetada hetkel kasutuses olev infosüsteemi rakendus uuema versiooni vastu. Selle jaoks tuli kõigepealt tutvuda olemasoleva rakenduse ning ASP.NET Core platvormiga. Uus loodav süsteem peaks järgima SPA põhimõtteid, olema kasutajasõbralik ning turvaline. Süsteemi uuendamise käigus on järgitud Blazor'i ja MVVM arhitektuuri mustrit ning ektreemprogrammeerimise metoodikat.

Lõputöö tulemusena saadakse vajalikud baasteadmised edasiseks arendamiseks ning valmib uuendatud rakenduse prototüüp. Lisaks kujuneb välja rakenduse arenduskava mida magistritöö käigus on plaanis järgida edasiarendamisel kasutades arendajatena ka bakalaureuse üliõpilasi.

Rakendus koosneb andmebaasist, kasutajaliidesest ning põhiprogrammist. Klient peab saama andmebaasi päringu abil luua, lugeda, uuendada ning kustutada talle sobilike andmeid. Suhtlus kliendi ja serveri vahel toimub RESTful veebiteenuse kaudu.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 41 leheküljel, 5 peatükki, 34 joonist.

Abstract

Preparations for Upgrading and Further Development of the Laboratory Information System

The aim of this bachelor’s thesis is to prepare for the transfer to a new platform and further development of the previously developed laboratory information system.

The purpose of updating the laboratory information system (LIMS) is to replace the currently used information system application with a newer version. For this, the first step is to study the existing application and the ASP.NET Core platform. The new system should follow the SPA principles, be user friendly and secure. During the system upgrade, the Blazor, MVVM architectural pattern and the extreme programming methodology have been followed.

As a result of the thesis, necessary knowledge for further development will be obtained and the protoype of the updated application will be completed. In addition, an application development plan will be developed which is planned to be followed during the Master's thesis by using the Bachelor's students as developers.

The application consists of a database, a user interface and a main application. The client must be able to create, read, update and delete the data that is suitable for him by querying the database. Communication between the client and the server takes place via the RESTful web service.

The thesis is in Estonian and contains 41 pages of text, 5 chapters, 34 figures.

Lühendite ja mõistete sõnastik

C#	Objektorienteeritud programmeerimiskeel
LIMS	*Laboratory information management system,* laboriteabe haldamise süsteem
ECMA	*European Computer Manufacturers Association,* kommunikatsioonitehnoloogia ning tarbeelektroonika standardiseerimisega tegelev organisatsioon
ISO	*International Organization for Standardization,* rahvusvaheline standardimisorganisatsioon
API	*Application programming interface,* rakendusliides
ASP.NET	Raamistik dünaamiliste veebirakenduste ja veebiteenuste loomiseks
ASP.NET Core	Veebiarenduse raamistik
WebAssembly	Veebiarenduse raamistiku standart veebilehte kasutavale rakendusele
SPA	*Single-page application,* üheleheline rakendus
Funktsioon	Programmi protseduur, mis tagastab väärtuse
Meetod	Programmi protseduur, mis ei tagasta väärtust
MVVM	*Model-view-viewmodel,* tarkvara arhitektuuriline muster
REST	Tarkvaraarhitektuuri stiil, mida kasutatakse veebiteenuste arendamisel
MVM	*Model-view-controller,* tarkvara arhitektuurimuster
JavaScript	JavaScript on objektorienteeritud programmeerimiskeel, mida kasutatakse peamiselt veebilehtede skriptimiseks
Azure	Microsofti pilvandmetöötlusteenus, mis on loodud rakenduste ja teenuste ehitamiseks, testimiseks, rakendamiseks ja haldamiseks Microsofti andmekeskuste abil

Sisukord

1 Sissejuhatus 10
1.1 Taust ja probleem 10
1.2 Eesmärk 10
1.3 Ülevaade tööst 11
2 Metoodika 12
2.1 Objekt 12
2.1.1 Laboriteabe haldamise süsteem (LIMS) 12
2.2 Tööriistad 13
2.2.1 C# 13
2.2.2 ASP.NET Core 14
2.2.3 Razor, Blazor ja FlexGrid 14
2.2.4 Puhas kood 16
2.2.5 Puhas arhitektuur 17
2.3 Protsess 17
2.3.1 Agiilne tarkvaraarendus 18
2.3.2 Ekstreemprogrammeerimine ehk XP 19
2.3.3 Testimisel põhinev arendus 20
3 Tehtud töö 21
3.1 Arhitektuur 21
3.2 Arhetüübid 24
3.3 Kasutajaliides 32
3.4 Testid 37
4 Analüüs ja arutelu 41
4.1 Millised on peamised konkureerivad süsteemid? 41
4.2 Miks on vaja uut LIMS süsteemi ning mis on selles teisiti? 41
4.3 Milline oli töö autori panus antud süsteemi arendamisse? 42
4.4 Nõuded ja edasise arendamise kava 44
5 Kokkuvõte 45

Kasutatud kirjandus 46
Lisa 1 – Uus CountriesController klass 49
Lisa 2 – Vana CountriesController klass 50
Lisa 3 – Universaalne kontroller andmete lugemise näitel 51
Lisa 4 – Prototüübi „Tabel Measure detailidega” kliendivaade 52
Lisa 5 – Prototüübi „Andmete lisamine” kliendivaade 53
Lisa 6 – Prototüübi „Andmete muutmine” kliendivaade 54

Jooniste loetelu

Joonis 1. Hello World programm c#-is 14
Joonis 2. Blazori arhitektuur [14]. 15
Joonis 3. ASP.NET puhta arhitektuuri näidis [19]. 17
Joonis 4. Agiilse metoodika visuaalne kujutus [25]. 19
Joonis 5. Ekstreemprogrammeerimise visuaalne kujutus [26]. 20
Joonis 6. Projekti üldine struktuur. 21
Joonis 7. Projekti struktuuri täpsem vaade. 22
Joonis 8. Puhta arhitektuuri näidis kihid [29]. 23
Joonis 9. Projekti kihtide omavaheline seos. 24
Joonis 10. Money arhetüübi mustrite paiknemine. 26
Joonis 11. Money arhetüübi Currency teoreetiline muster [30]. 27
Joonis 12. Projektis realiseeritud Currency muster. 27
Joonis 13. Quantity arhetüübi Unit teoreetiline muster [30]. 28
Joonis 14. Projektis realiseeritud Unit muster. 28
Joonis 15. Rule arhetüübi RuleSet teoreetiline muster [30]. 29
Joonis 16. Projektis realiseeritud RuleSet muster. 29
Joonis 17. Party arhetüübi Person teoreetiline muster [30]. 30
Joonis 18. Projektis realiseeritud Person muster. 31
Joonis 19. PartyRelationship arhetüübi teoreetiline muster [30]. 31
Joonis 20. Projektis realiseeritud PartyRelationship muster. 32
Joonis 21. CurrencyView klass. 33
Joonis 22. CurrencyViewFactory klass. 33
Joonis 23. CurrencyController klass. 34
Joonis 24. CurrenciesGrid konfiguratsiooni klass. 35
Joonis 25. Currencies vaate klass. 35
Joonis 26. Traditsioonilise lehekülje eluring ja SPA eluring [32]. 36
Joonis 27. LazyLoad disaini muster [34]. 36
Joonis 28. PriceData klass. 37
Joonis 29. PriceData klassi testid. 38

Joonis 30. Measure klass. 39
Joonis 31. Measure klassi testid. 40
Joonis 32. Projekti koodiridade arv. 43
Joonis 33. Lõik ajakulu graafikust. 43
Joonis 34. Koodi katvus testitega. 44

1 Sissejuhatus

“Tehnoloogia areng põhineb selle sobitamisel nii, et te seda isegi ei märka kuid see on osa teie igapäevaelust” - Bill Gates. Antud tsitaat võtab minu arvates kokku hetkeolukorra meie ühiskonnas. Tavakodanik ei pane tõenäoliselt kõiki muudatusi tähele kuid tehnoloogia arengut on saatnud meeletu kiirus. Alles paarteist aastat tagasi ei eksisteerinud nutitelefonid (tänaste standardite kujul) ning umbes 30 aastat tagasi ei omanud keegi isegi personaalset arvutit [1]. Me oleme jõudnud punkti kus muutuste tempo ja kiirus ületavad meie võimet kohaneda uute tehnoloogiatega [2]. Seoses tehnoloogia kiire edasiarenguga on tähtis, et ka firmad uuendaksid kasutuses olevat tarkvara. Samas leidub ettevõtteid ning riigi asutusi kus pidev tarkvara hooldamine ning edasiarendamine on jäänud tahaplaanile. Tihtipeale langevad paljud neist küberrünnaku ohvriks kuid õnneks leidub ka ettevõtteid, kes on enda või teiste vigadest õppinud ning teadlikud tarkvara hooldamise tähtsusest [3]. Kogu eelnev on põhjenduseks, miks lõputöös kasutatav rakendus vajab uuendamist.

1.1 Taust ja probleem

Erinevalt tarkvara autoritest kipuvad nende süsteemid elama palju kauem kui kunagi oli ette nähtud või arvatud. Seega peavad ettevõtted kaasajastama oma tarkvarasüsteeme nii, et hoida end produktiivsena uue tehnoloogiaga kaasnevas keskkonnas [4].

Antud lõputöös käsitletav tarkvara on juba kasutajavalmis kuid nõuab uuendamist. Hetkel Leeds’i ülikoolis kasutusel olev labori infosüsteem on igati funktsioneeriv kuid arvestades, et antud tarkvara kasutatakse igapäevaselt peab see vastama ka tulevastel nõuetele. Seega on peamiseks probleemiks tarkvara kaasaegsus ehk rakendus vajab uuele platvormile üleviimist, et tagada vajalik funktsionaalsus. Lisaks tuleb testida kirjutatud koodi, et tagada ülikooli ja andmete turvalisus.

1.2 Eesmärk

Antud lõputöö peamiseks eesmärgiks on ette valmistuda labori infosüsteemi viimiseks praeguselt kujult üle teisele platvormile ning edasiarendamiseks ilma, et rakendus kaotaks juba olemasolevat funktsionaalsust. Kui hetkel on tegemist töölauarakendusega, siis eesmärk on muuta see veebirakenduseks, viies see üle ASP.NET Core platvormile. Selle jaoks tuleb selgeks teha antud platvormi ning sellega kaasnevate raamistike tööpõhimõtted, et valida välja sobilikem meetod eesmärgi saavutamiseks. Kohendada arhitektuuri ja katsetada erinevaid SPA raamistike. Viies antud infosüsteem üle veebi, tagame sellega ka kaasaegsuse. Teiseks eesmärgiks on teha süsteem selliseks, mis võimaldaks ka andmete korduvkasutamist. Viimaseks suuremaks eesmärgiks on kogu koodi ühik- ja integratsioontestimine, et tagada maksimaalne turvalisus ning koodi funktsioneeritavus. Kirjutatud kood peab olema puhas, et saavutada minimaalse koodiga maksimaalne funktsionaalsus.

1.3 Ülevaade tööst

Lõputöö esimeses peatükis kirjeldatakse antud töö tausta, tuuakse välja peamised probleemid ning neile vastavalt seatud eesmärgid. Lisaks antakse ka ülevaade tööst. Töö teises peatükis esitatakse metoodika. Kirjeldatakse objekti, kasutatavaid tööriistu ja protsessi. Kolmandas peatükis tuuakse välja tehtud töö ning tulemused. Sinna alla kuuluvad arhitektuur, arhetüübid, kasutajaliides ning testid. Neljandas peatükis esitatakse analüüs ja arutelu ning bakalaureusetöö lõppeb kokkuvõttega.

2 Metoodika

Käesolevas peatükis antakse täpsem ülevaade lõputöö metoodikast ning labori infosüsteemist LIMS. Tuuakse välja peamised tööriistad ja raamistikud mida projekti tegemisel kasutatakse. Esitatakse ka puhta koodi, arhitektuuri ning arendusmeetodi põhimõtteid, mida järgiti rakendust arendades.

2.1 Objekt

Antud lõputöös käsitletav tarkvara on labori infosüsteem ehk tegemist on tarkvarapõhise lahendusega, mis toetab kaasaegse labori tegevusi. Peamised funktsioonid on - kuid mitte ainult - töövoo ja andmete jälgimise tugi, paindlik arhitektuur ja andmevahetuse liidesed, mis "toetavad täielikult selle kasutamist reguleeritud keskkonnas" [5]. Järgnevalt kirjeldan labori infosüsteemi LIMS.

2.1.1 Laboriteabe haldamise süsteem (LIMS)

Laboriteabe haldamise süsteem (LIMS) viitab laboratoorsete operatsioonide tsentraliseeritud haldamisele. Seda tehakse tarkvarapõhiste infotehnoloogiasüsteemide abil, mis aitavad kombineerida laboris erinevaid funktsioone [6]. LIMS võimaldab tõhusalt proove ja nendega seotud andmeid hallata. LIMS-i abil saab labor automatiseerida töövooge, integreerida instrumente ja hallata proove ja nendega seotud teavet [7]. Sellise tsentraliseerimise kombineeritud aspekte nimetatakse LIMS-ks.

Traditsiooniliselt on LIMS mõeldud töötlema ja esitama andmeid, mis on seotud bioloogiliste laboritega, veepuhastusjaamade, ravimikatsete ja muude keeruliste andmepakettidega tegelevate üksuste proovide partiidega. LIMS võib vajada (kuid mitte alati) luba ning nõusolekut erinevatelt tööstusharusid reguleerivate asutuste ja teadustöötajate poolt. Lisaks on LIMS kõige konkurentsivõimelisem rühma-kesksete tingimustega (tegeletakse "partiide" ja "proovidega"), tegelevate teadusuuringutega seotud laborites [5]. Seega sobib LIMS tavaliselt suuremate rajatiste jaoks, mis peavad töötlema suuri andmepartiisid.

2.2 Tööriistad

Arenenud riikides on projektihaldus ettevõtetes peaaegu mõeldamatu ilma pühendatud tarkvara tööriistadeta [8]. Järgnevalt toon välja lõputöö käigus kasutatavad tööriistad.

2.2.1 C#

C# (C Sharp) on üldotstarbeline programmeerimiskeel, mis arendati aastal 2000 Microsofti poolt .NET initsiatiivi raames ning standardiseeritud ECMA ja ISO poolt. C# on üks .NET-raamistiku keeltest. Tema praegust arendusmeeskonda juhib Mads Torgersen kuid algne disainer oli Anders Hejlsberg [9].

ECMA standardi järgi on C# disaini põhimõtted järgmised [9]:

- C# on lihtne, kaasaegne ja üldotstarbeline objektorienteeritud programmeerimiskeel.
- Keel ja selle teostused toetavad tarkvaraarenduse põhimõtteid nagu tugev tüüpimine, massiivi piiride kontrollimine, väärtustamata muutujate kasutamise avastamine ning automaatne mälukoristus. Tähtsad on tarkvara robustsus ja programmeerija tootlikkus.
- Keel on mõeldud tarkvarakomponentide loomiseks hajusates keskkondades.
- Lähtekoodi porditavus on väga oluline, nagu ka äratuntavus programmeerijatele, kes on tuttavad keeltega C ja C++.
- Rahvusvahelikustamise toetus on tähtis.
- C# on sobilik rakenduste kirjutamiseks erinevates süsteemides, alates keerulisi operatsioonisüsteeme kasutavatest süsteemidest lõpetades manussüsteemidega.
- Kuigi C# rakendused peaksid kasutama mälu ja arvutusvõimsust säästlikult, ei võistle keel jõudluses ega mälukasutuses otseselt C või assemblerkeelega.

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        System.Console.WriteLine("Hello, world");
    }
}
```

Joonis 1. Hello World programm c#-is

2.2.2 ASP.NET Core

ASP.NET on populaarne veebiarenduse raamistik veebirakenduste loomiseks .NET-platvormil. ASP.NET Core on ASP.NET'i avatud lähtekoodiga versioon, mis töötab MacOS, Linux ja Windows operatsioonisüsteemidel. See ilmus esmakordselt 2016. aastal ja on ASP.NET'i varasemate Windows-i versioonide ümberkujundamine Microsofti ja tema kogukonna poolt [10].

Mõned ASP.NET Core omadused [11]:

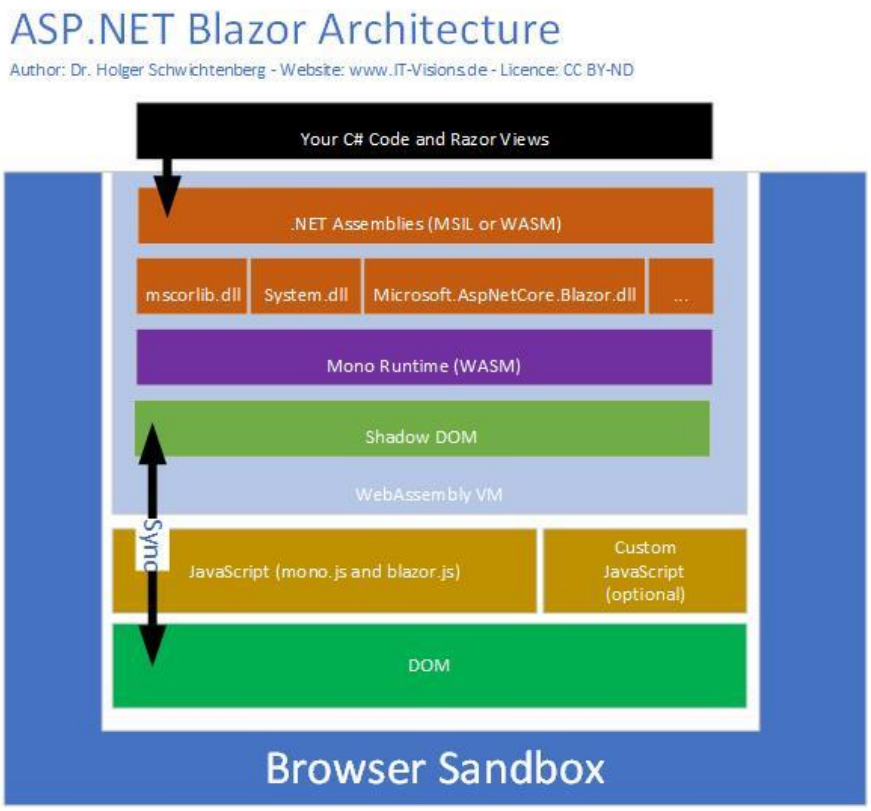
- Avatud lähtekoodiga ja kogukonnapõhine
- Globaliseerumine ja lokaliseerimine
- Ühendatud MVC & Web API raamistikud
- Asünkroonne async/await funktsiooni kaudu
- Üks kiiremaid veebirakenduste raamistikke

2.2.3 Razor, Blazor ja FlexGrid

Razor on ASP.NET programmeerimise süntaks, mida kasutatakse C# või VB.NET programmeerimiskeelega dünaamiliste veebilehtede loomiseks. Razorit hakati arendama aastal 2010 ning esimest korda ilmus see 2011. aasta jaanuaris Microsoft Visual Studio jaoks. Razor'i veebilehti võib kirjeldada kui HTML-lehekülgi, millel on kahte tüüpi sisu: HTML sisu ja Razor'i kood [12].

Blazor on veebi kasutajaliidese raamistik, mis põhineb C #, Razoril ja HTML-il ning see töötab veebibrauserites WebAssembly'i kaudu. Blazori eesmärk on lihtsustada kiiret SPA

loomist. See võimaldab veebiarendajatel kirjutada .NET-põhiseid veebirakendusi, mis töötavad kliendipoolsetes veebibrauserites, kasutades avatud veebistandardeid [13]. Blazoris kasutatakse Razori kliendivaate puhul [14].



Joonis 2. Blazori arhitektuur [14].

FlexGrid on GridView komponent Blazorile, mis võimaldab arendajal andmeid kergelt kuvada, muuta, lisada ja kustutada tabelites. Lisaks pakub FlexGridi võimalust kuvada erinevaid, omavahel seotud andmekogusid läbi Master/Detail vaate [15]. Nii Blazor kui ka GridView on alles arendamisjärgus seega võib neis esineda vigu või puudusi.

Blazori raamistik sai valitud kuna see võimaldab viia kliendiarenduse .NET'i peale ning ära kaotada JavaScripti. FlexGrid'il on küll alternatiive kuid need tulid alles peale projekti alustamist välja. FlexGrid'i kasutamist on mugavamaks teinud ka tema arendaja Jaroslav Surala, kes küsimuste korral alati abi pakub.

2.2.4 Puhas kood

“Puhta koodi kirjutamine on midagi, mida sa pead tegema, selleks et kutsuda ennast professionaaliks.” - Robert C.Martin [16].

Koodi kvaliteet on oluline tarkvarasüsteemide hooldatavuse ja jätkusuutlikkuse seisukohast ning seda käsitletakse juhendite ja vastastik eksperthinnangute järgi. Koodi kvaliteedi jaoks on olemas hästi määratletud meetodid ja standardid - näiteks “Puhta koodi” lähenemine [17]. Puhta koodi kirjutama õppimine on raske töö. See nõuab rohkem kui lihtsalt põhimõtete ja mustrite tundmist - see nõuab distsipliini ja kogemusi.

Mõned puhta koodi omadused ja kirjeldused [16]:

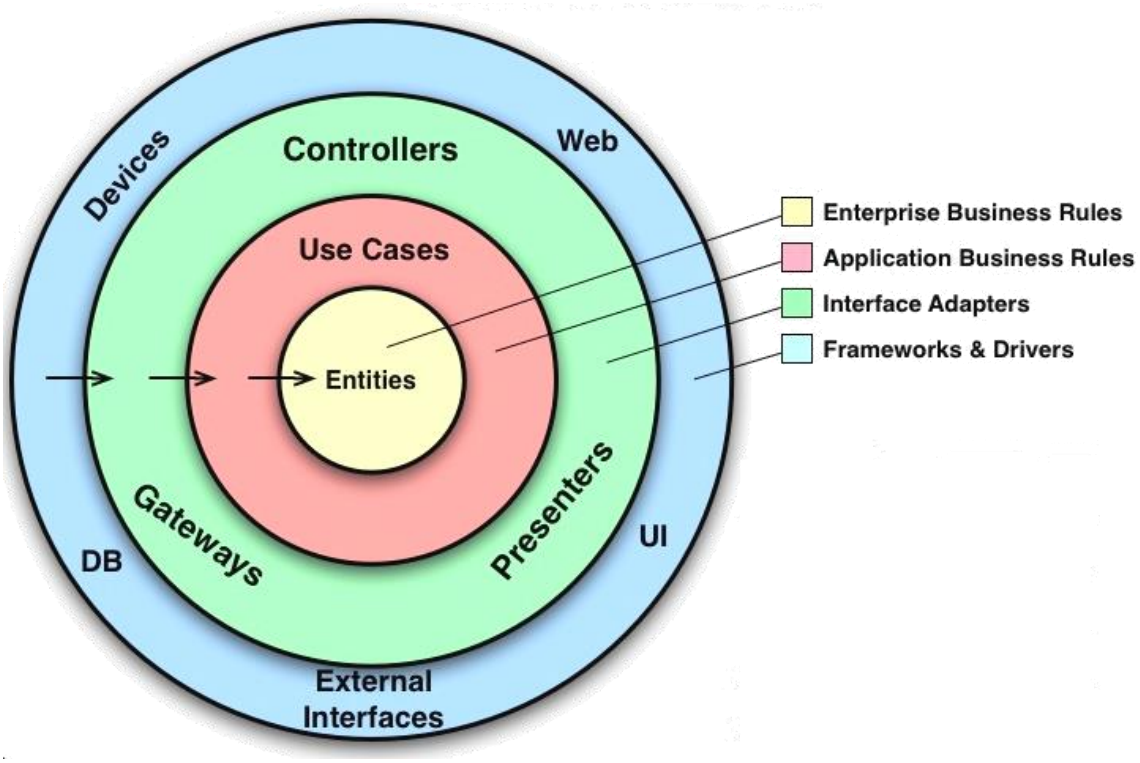
- Puhas kood on fokuseeritud - iga funktsioon, klass ja moodul paljastab üksmeelse suhtumise.
- Puhas kood võiks olla elegantne, et koodi lugemine oleks meeldiv ja arusaadav.
- Kirjutatud koodi eest on hoolt kantud, see on lihtne kuid korrektne
- Määratud nimed võiksid olla tähenduslikud
- Ei sisalda kordusi
- Funktsioonid peaksid täitma ainult ühte eesmärki
- Väldi kommentaare ehk selgita ennast koodis
- Testi koodi
- Jäta kood paremaks kui sa selle leidsid!

Rohkem infot “Puhta Koodi” kohta leiab Uncle Bob’i ehk Robert C. Martin’i kirjutatud raamatust - “Clean Code: A Handbook of Agile Software Craftsmanship”. Robert C. Martin on tarkvara insener ja juhendaja. Teda tuntakse ka kui ühte agiilse manifesti autoritest [18].

Lisas 1 ja 2 on näha sama klassi pärast ja enne universaalset kontrollerit. Universaalne kontroller (lisa 3) võimaldab funktsioonide korduvkasutamist ehk vähendab koodi hulka ning muudab klassid puhtamaks.

2.2.5 Puhas arhitektuur

Puhas arhitektuur on samuti ideoloogia, mis on aastate jooksul tarkvaraarenduse maastikul välja kujunenud. Nii nagu "Puhta koodi" puhul, pole ka siin kindlaid reeglistikke vaid soovitused ja ideed parema tarkvara arhitektuuri saavutamiseks. Peamiseks ideeks on tarkvara kihtide eraldamine üksteisest, et tagada lihtne muudatuste tegemine ning parem ülevaade projektist. Puhtast arhitektuurist räägib nii Martin Fowler oma raamatus "Patterns of Enterprise Application Architecture" kuid ka Robert C. Martin oma raamatus "Clean Architecture: A Craftsman's Guide to Software Structure and Design".



Joonis 3. ASP.NET puhta arhitektuuri näidis [19].

2.3 Protsess

Tarkvaraarenduses on tarkvara arendamise protsess töö jagamine erinevateks etappideks, et parandada disaini, tootehaldust ja projektijuhtimist. Seda tuntakse ka tarkvaraarenduse elutsükli nime all. Enamik kaasaegseid arenguprotsesse võib kirjeldada kui agiilseid. Teiste meetodite hulka kuuluvad ka juga, prototüüpimine, iteratiivne ja järkjärguline

areng, spiraalne areng, kiire rakenduste arendamine ja ekstreemprogrammeerimine [20]. Järgnevalt toon välja töös kasutatud metoodikate kirjeldused.

2.3.1 Agiilne tarkvaraarendus

Agiilsed meetodid on muutnud tarkvara arendamise viisi, rõhutades aktiivset lõppkasutajate kaasatust, tolerantsust muutustele ja toodete evolutsioonilist tarnimist. Algselt oli agiilse tarkvara arendamise meetodid suunatud väikestele arendusmeeskondadele ehk neid kasutati veebisüsteemide ja firmasisemiste IT-süsteemide arendamiseks, kuid nüüd kasutatakse neid mitmetes erinevates valdkonnas, sealhulgas missioonikriitilistes süsteemides [21].

Esimesed kirjeldused kiirest arendusest pärinevad 1980-1990 aastatest ning aastal 2001 kogunes kokku 17 tarkvaraarendajat aruteluks arendusmeetodite üle [22]. Selle tulemusel tekkis tekst nimega "Manifesto for Agile Software Development" ehk agiilse tarkvaraarenduse manifest.

Paremad praktikad tarkvaraarenduses on need, kus hinnatakse:

- **inimesi ja nendevahelist suhtlust** rohkem, kui protsesse ja arendusvahendeid
- **töötavat tarkvara** rohkem, kui kõikehõlmavat dokumentatsiooni
- **koostööd kliendiga** rohkem, kui läbirääkimisi lepingute üle
- **reageerimist muutunud oludele** rohkem, kui algse plaani järgimist

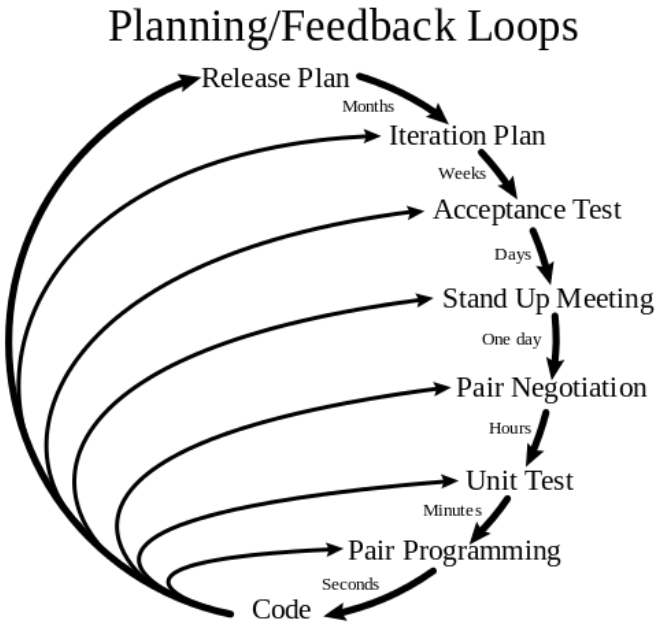
Ka parempoolsetel teguritel on väärtus, kuid vasakpoolseid tegureid hinnatakse kõrgemalt [23].

Enim kasutatud agiilsed metoodikad on Scrum, Lean and Kanban, XP ja Crystal [24].

Joonis 4. Agiilse metoodika visuaalne kujutus [25].

2.3.2 Ekstreemprogrammeerimine ehk XP

Ekstreemprogrammeerimine (kirjeldatud aastal 1999 Kent Beck'i poolt) on tarkvaraarenduse metoodika, mis on mõeldud parandama tarkvara kvaliteeti ja vastavust kliendi muutuvatele vajadustele. Agiilse tarkvaraarenduse alaliigina toetab see sagedasi „avaldamisi" lühikeste arengutsüklite jooksul, parandamaks produktiivsust ja tutvustamaks kontrollpunkte, kus uued klientide nõuded vastu võtta. Elementideks on paarisprogrammeerimine või ulatuslike koodi ülevaatuste tegemine, kogu koodi ühiktestimine, funktsioonide programmeerimise vältimine enne, kui neid on tegelikult vaja, kindel juhtimisstruktuur, koodi lihtsus ja selgus, eeldamine, et kliendi nõuded muutuvad aja möödudes ning probleemi selginedes ja sage suhtlus kliendiga ja programmeerijate vahel [26].



Joonis 5. Ekstreemprogrammeerimise visuaalne kujutus [26].

2.3.3 Testimisel põhinev arendus

Testimisel põhinev arendus (lühendatult TDD) on tarkvaraarenduse meetod, kus testid kirjutatakse enne koodi. Arendusprotsess koosneb lühikestest iteratsioonidest, kus esmalt kirjutatakse ebaõnnestuv test ning seejärel minimaalne kood, et see test läbi läheks. Selline arendus tagab olukorra, kus kogu kood on alati testitud. Sellisel viisil kirjutatud koodi on kergem refaktoreerida ning testid aitavad ka programmi eeldatavat käitumist dokumenteerida. Testimisel põhinevat arendust rakendatakse ka olemasoleva koodi täiustamisel ja silumisel. TTD, mis sai alguse aastal 1999 ning mille loojaks on samuti Kent Beck, on seotud ekstreemprogrammeerimise “testi-esiteks“ kontseptsiooniga [27].

Testimisel põhineva arenduse protsessi tsükkel on [28]:

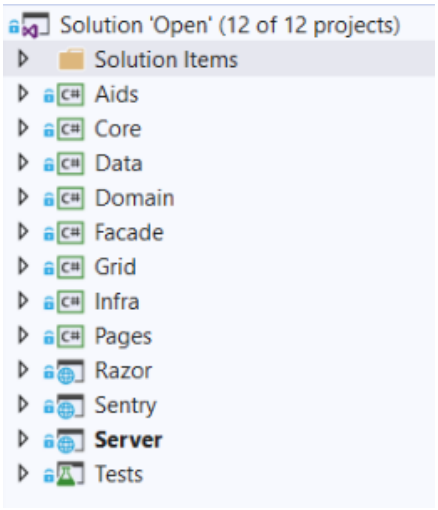
1. Kirjuta test
2. Saa see kompileerima
3. Saa see jooksma
4. Eemalda kordused

3 Tehtud töö

Antud peatükk annab ülevaate rakenduse tehnilisest poolest.

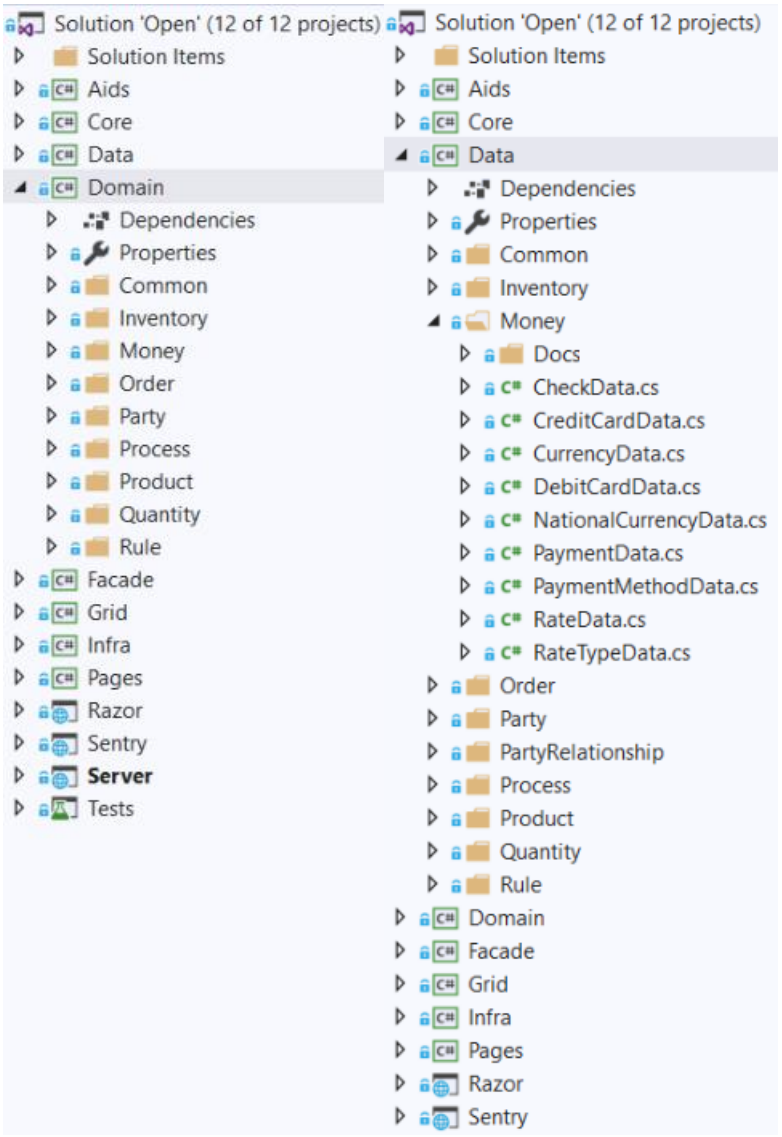
3.1 Arhitektuur

Kogu tarkvara loogikakiht on ära jaotatud puhtal arhitektuuril põhineva ideoloogia järgi.



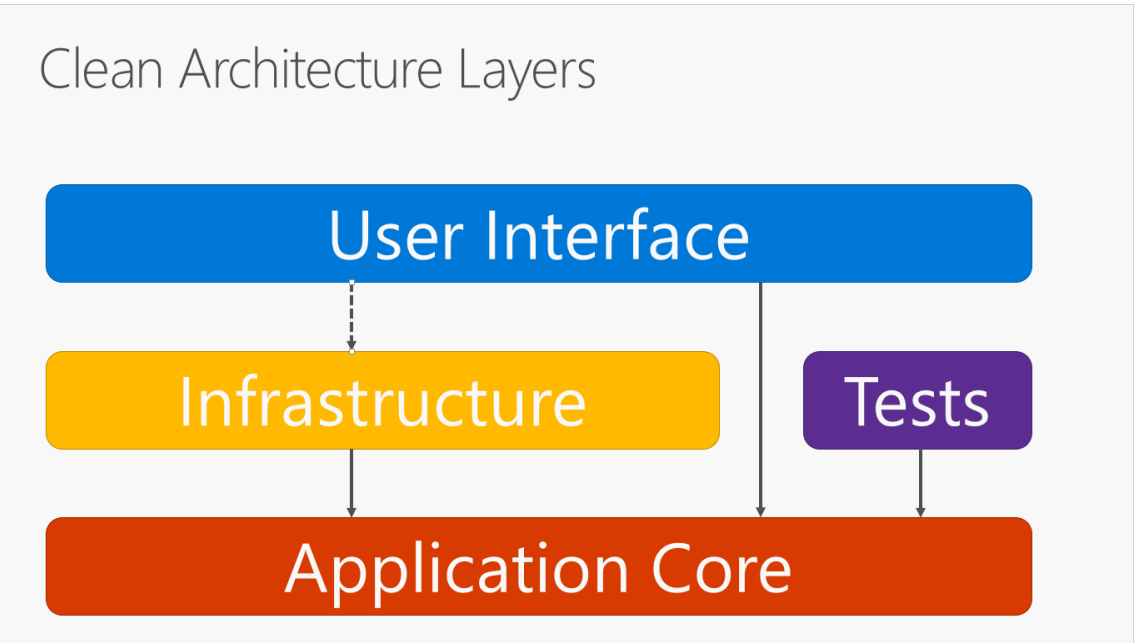
Joonis 6. Projekti üldine struktuur.

Joonisel 6 on kujutatud kogu projekti üldine struktuur. Projekt on jaotatud 12 kihiks ning iga kiht on üksteisest eraldatud. Esiteks tagab antud struktuur parema ülevaate ning arusaadavuse projekti haldajatele. See omakorda lihtsustab ja kiirendab tarkvaraarendaja tööd - näiteks funktsioonide ja andmete korduvkasutamine. Teiseks võimaldab selline jaotus projektis muudatuste tegemise ilma, et miski laguneks. Kõik kihid on omavahel seotud kuid juhul, kui arendamise käigus tekib kuskil viga, siis ülejäänud kihid on endiselt funktsioneerivad. Sama idee kehtib ka uute funktsioonide rakendamisel ja olemasolevate muutmisel.



Joonis 7. Projekti struktuuri täpsem vaade.

Joonisel 7 on näha kuidas erinevad kihid on omakorda struktureeritud. Iga kihi klassid on jaotatud vastavalt oma valdkonnale kausta - põhjus selleks on sama mis eelmises punktis mainitud.



Joonis 8. Puhta arhitektuuri näidis kihid [29].

Antud projekti kihid saab suuremas pildis ära jagada vastavalt ASP.NET Core puhta arhitektuuri näite põhjal nii:

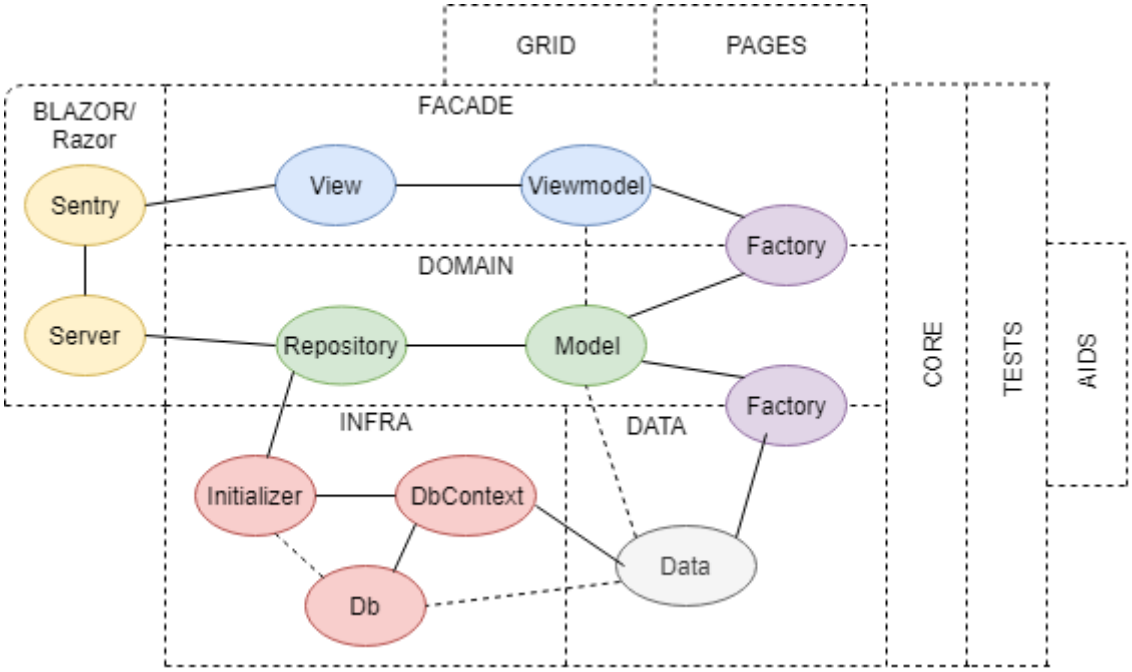
- Kasutajaliidese kiht – Facade
- Infrastruktuuri kiht - Infra
- Rakenduse tuum - Data, Domain, Server, Sentry, Razor, Grid
- Testid - Tests

Facade sisaldab klasse mida kasutatakse kliendivaate genereerimisel (nt. välja nimetus, andmete määramine ja päring Domain'ist). Kui Facade on otseses suhtluses kasutajaliidesega, siis Domain on kiht kasutaja ja rakenduse vahel, mis edastab andmeid ühelt poolt teisele ning vastupidi. Infra kiht paikneb Domain ja Data vahel ning initsialiseerib ja hoiab andmebaasi andmeid, mis liiguvad andmebaasist kliendi poole ja vastupidi. Data sisaldab klasse kus määratakse igale valdkonnale iseloomulikud algomadused. Sentry sisaldab kasutajaliidest (mida toetab Facade) ning Server on n-ö veebirakendus mida kasutab klient. Razor vana kliendivaade, mis ei kasuta Blazori. Lisaks on meil ka "abistav" kiht, kuhu alla kuuluvad - Aids, Core ja Pages.

- Core - Sisaldab kõiki teisi kihte abistavaid klasse
- Tests - Sisaldab kõikide kihtide ühik- ja integratsiooniteste

- Aids - Toetab Tests kihti
- Pages - Abistav kiht kliendivaatega seotud klassidele

Seega on projekti kihid, nii üldvaates kui ka eraldatuna, mingil moel omavahel seotud ning täidavad kindlat ülesannet.



Joonis 9. Projekti kihtide omavaheline seos.

- Db (andmebaas) saab algandmed tänu Initializeri.
- Kasutaja saab andmed läbi Repository, sest see suhtleb läbi DbContexti andmebaasiga. Repository ei suhtle Dataga otse vaid kasutab selleks Modelit.
- DbContext saab Db'st Data abil infot kätte, sest Db on Dataga otseselt seotud.
- Model'ist Data ja vastupidi teeb Factory. Sama kehtib ka Modeli ja Viewmodeli puhul.

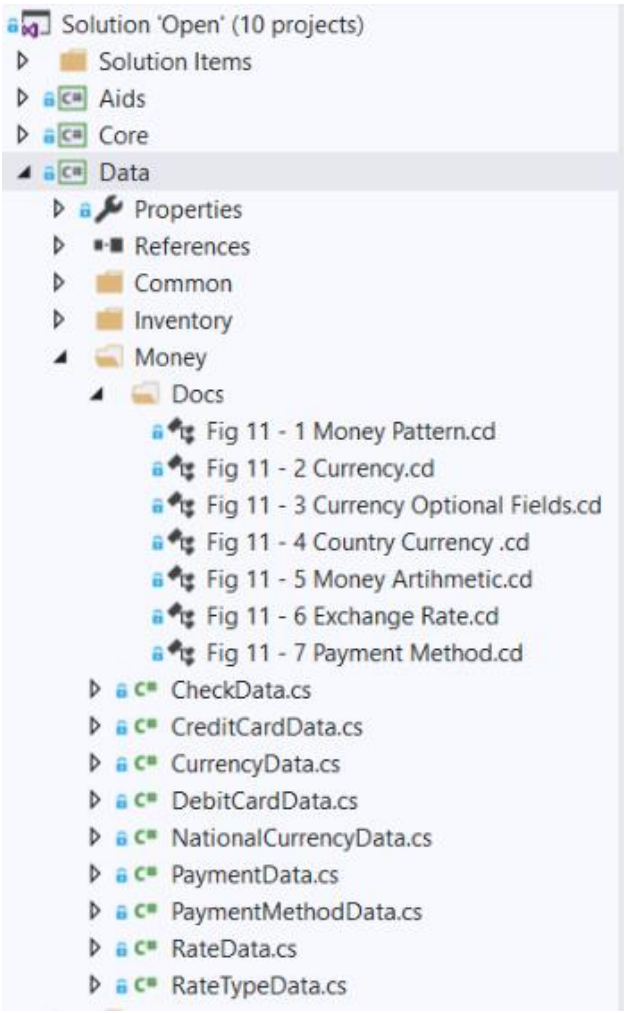
3.2 Arhetüübid

Arhetüüp on millegi algkuju - näiteks ettevõttel on arhetüüp Osapool, mis esindab tuvastatavat ja adresseeritavat üksust, millel võib olla juriidiline staatus. Tavaliselt esindab see isikut või organisatsiooni. Lisaks on olemas ettevõttel ka arhetüübi ja analüüsi muster. Ettevõtte arhetüübi muster on koostöö ärialaste arhetüüpide vahel, mis

toimuvad järjepidevalt ja universaalselt ärikeskkondades ja tarkvarasüsteemides. Ettevõtte analüüsi muster on see-eest kontseptsioonide rühmad, mis kujutavad endast ärimudeli ühist ehitust.

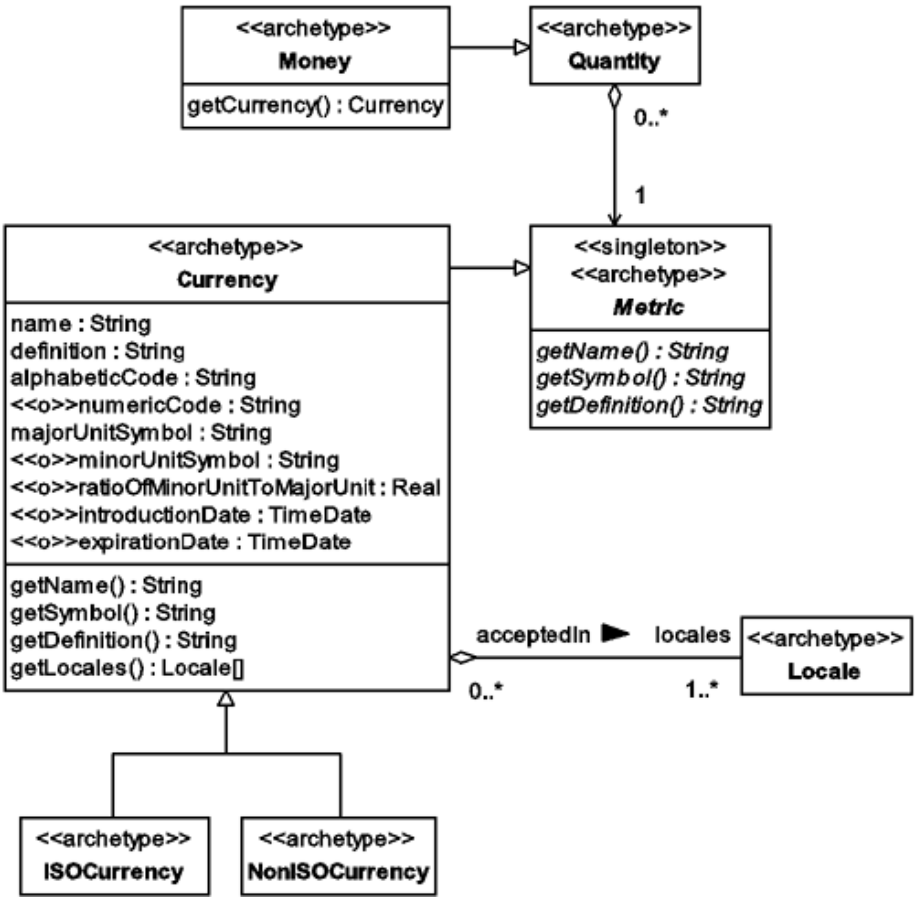
Arhetüübi mustrite kasutamise peamine põhjus on kvaliteedi tõstmine hoides koodi lihtsana. Üks probleem objekt-modelleerimises, eriti analüüsi- ja äritasandil, on see, et üsna raske on öelda, kas tehtud mudelid on õiged. Võttes appi olemasolevad arhetüübi mustrid ning analüüsides tarkvarasüsteemi, et leida kohti kus saaks neid rakendada, on võimalik tagada mudelite õigsus. Arhetüübi mustrite kasutamine on iseloomulik ekstreemprogrammeerimise juures [30].

Antud projektis on samuti järgitud olemasolevaid arhetüüpide mustreid. Nende kasutamine võimaldab meil luua spetsiifilisi komponente, mis on näiteks vajalikud LIMS süsteemis. Näidis mustrid on võetud raamatust “Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML”. Iga realiseeritava arhetüübi mustriga kihi diagrammid asuvad Data kihis vastava teema Docs kaustas.

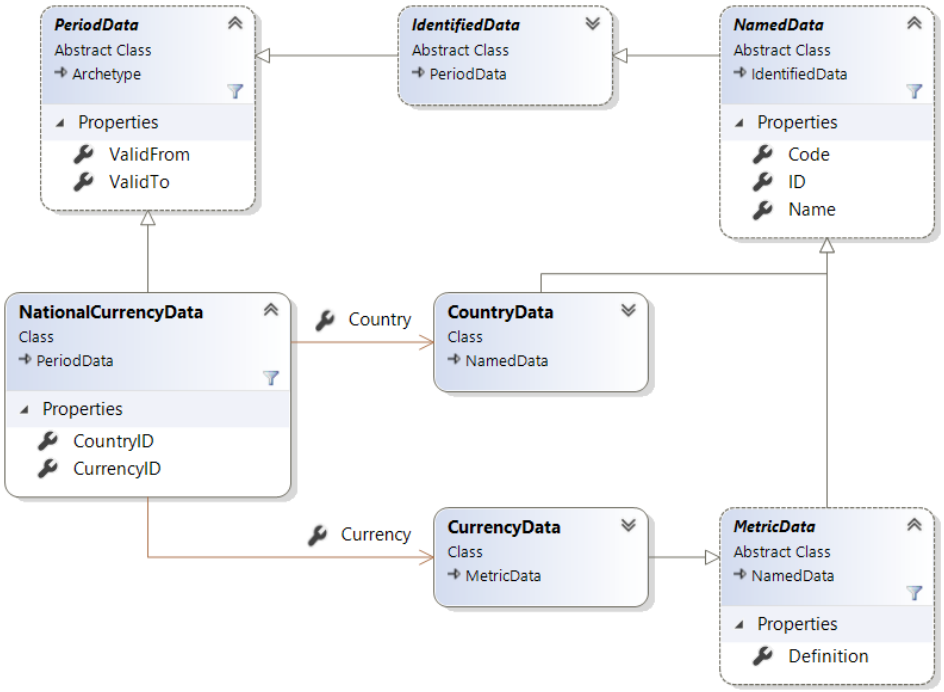


Joonis 10. Money arhetüübi mustrite paiknemine.

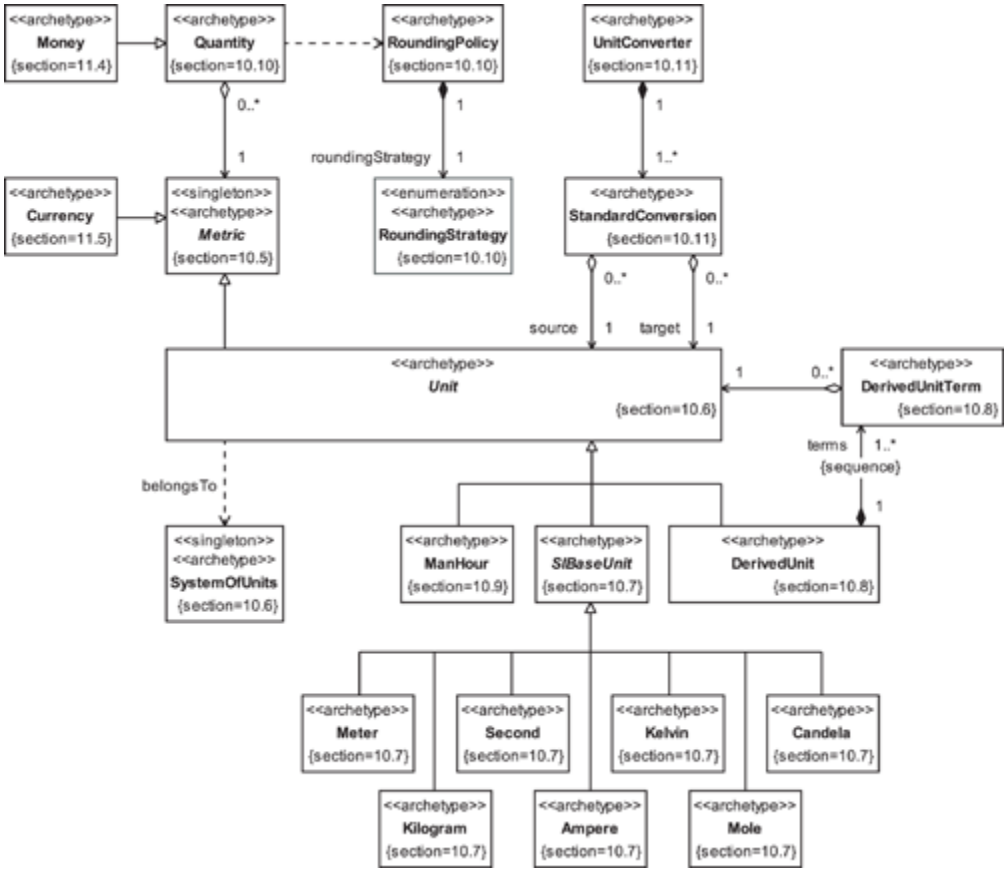
Järgnevalt toon võrdluseks mõned raamatus esitatud mustrid ning meie realiseerimised. Pildid ei ole identsed, kuna lõputöö käigus valmiv rakendus ei nõua samu klasse, kuid üldplaanis sarnased.



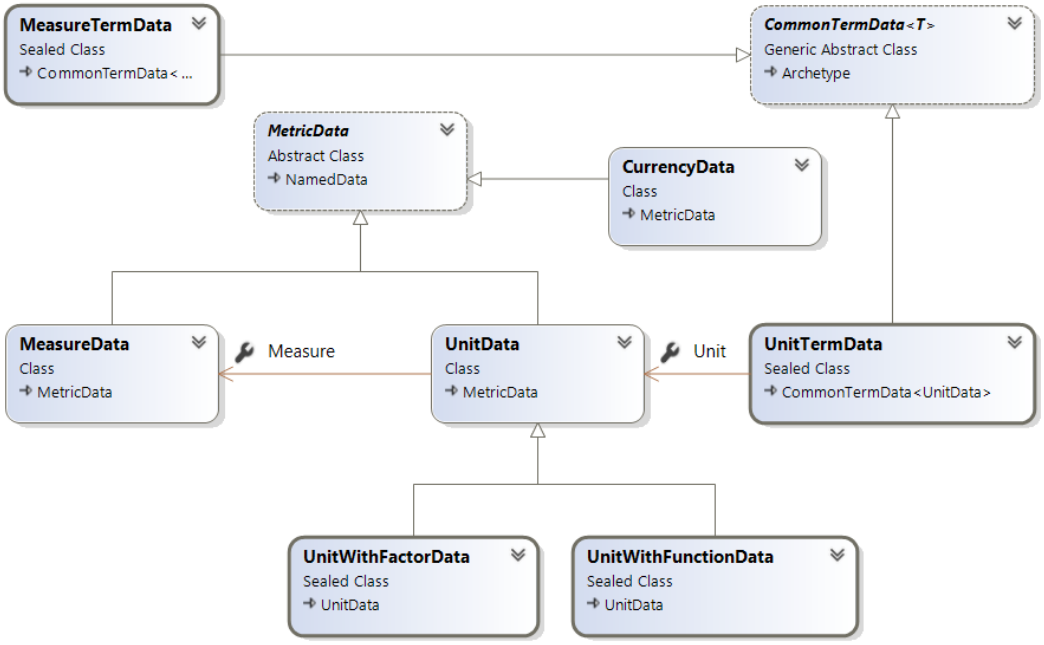
Joonis 11. Money arhetüübi Currency teoreetiline muster [30].



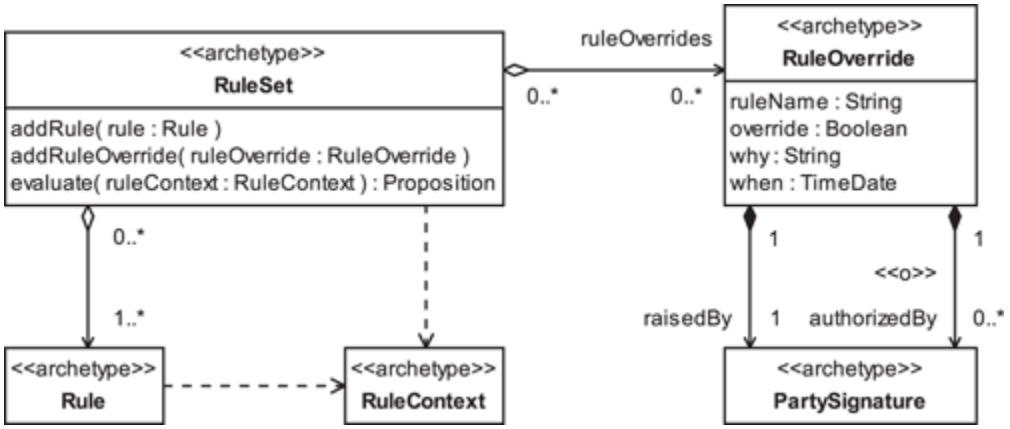
Joonis 12. Projektis realiseeritud Currency muster.



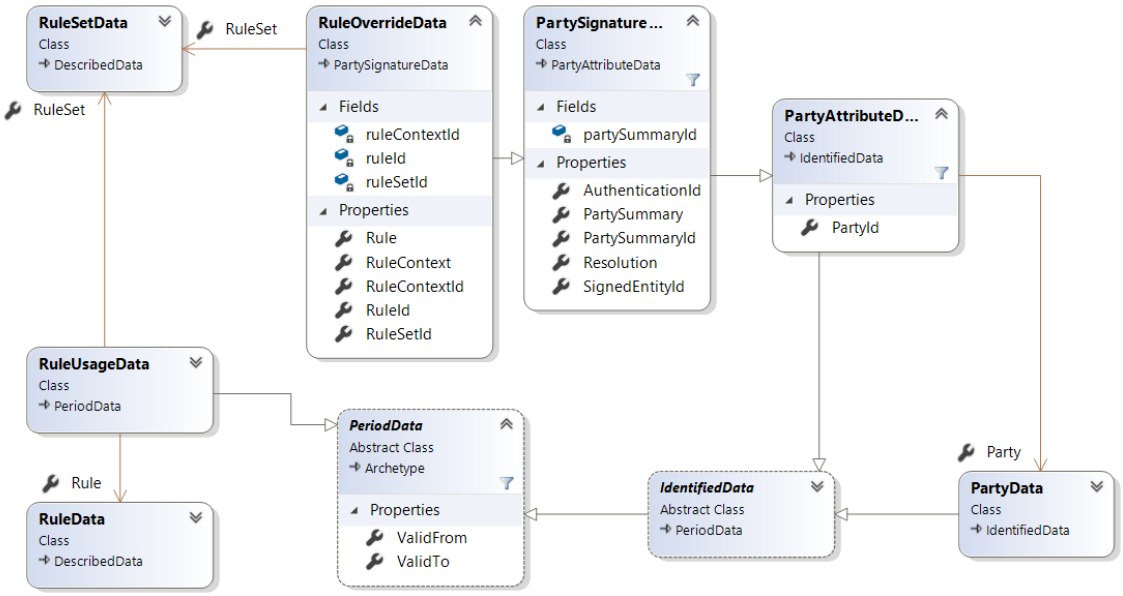
Joonis 13. Quantity arhetüübi Unit teoreetiline muster [30].



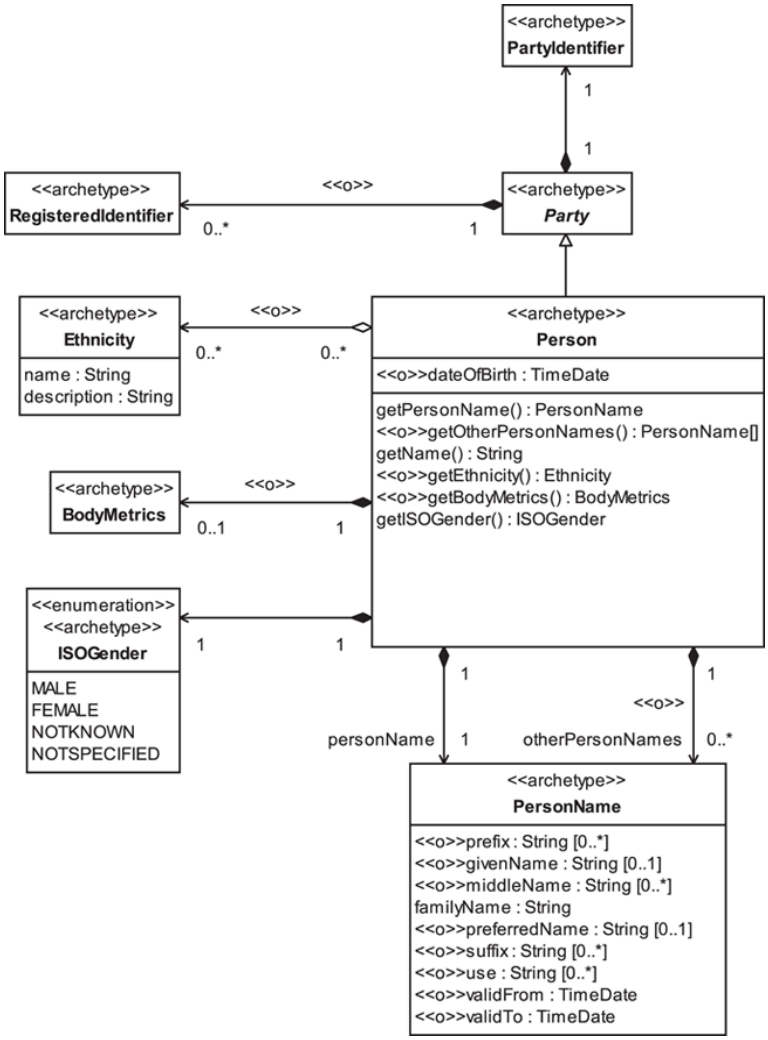
Joonis 14. Projektis realiseeritud Unit muster.



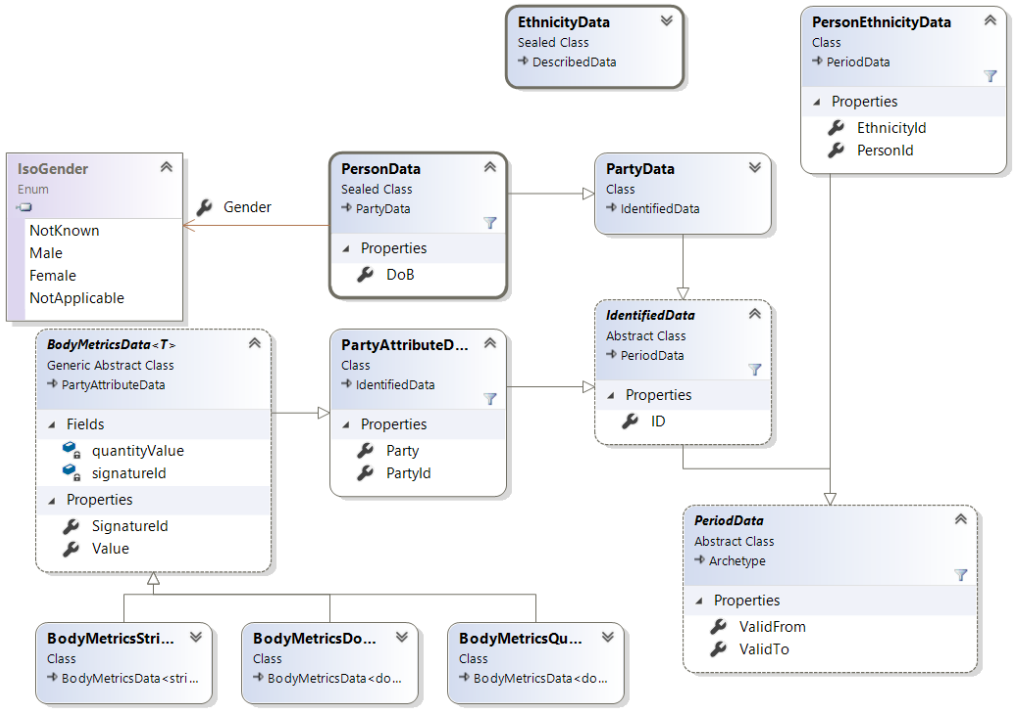
Joonis 15. Rule arhetüübi RuleSet teoreetiline muster [30].



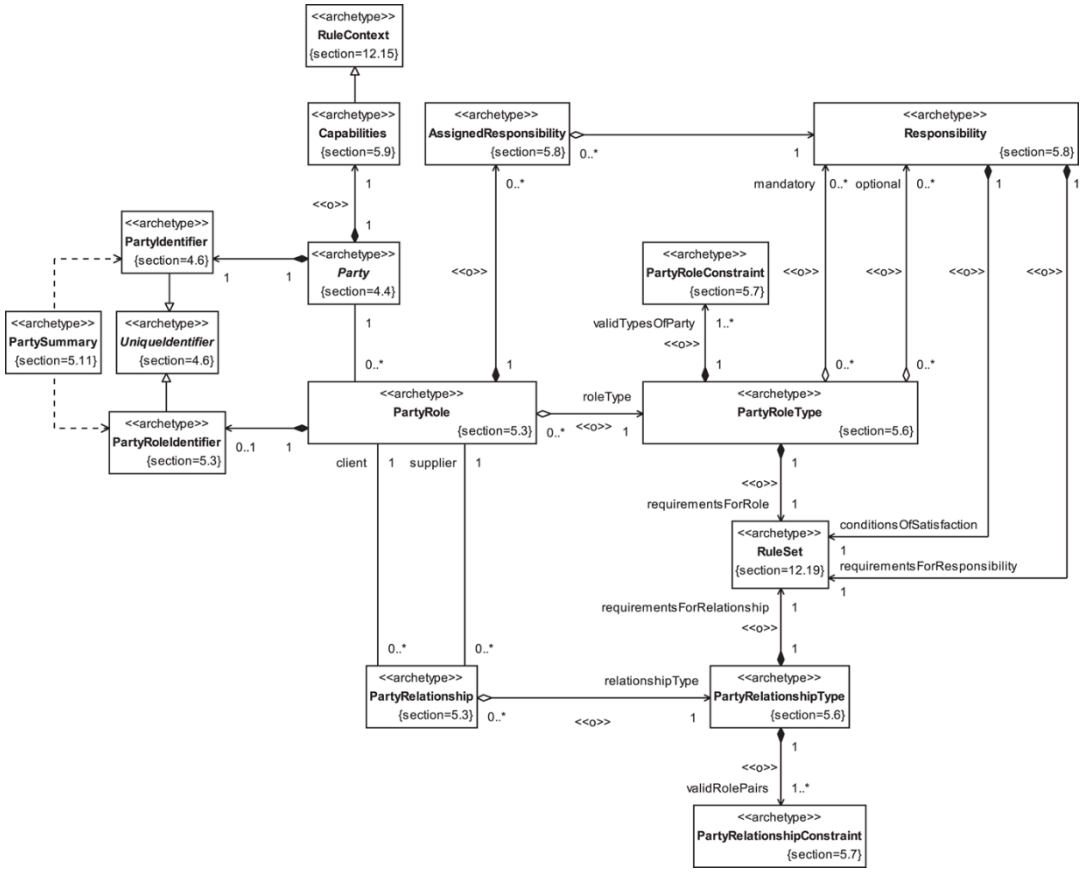
Joonis 16. Projektis realiseeritud RuleSet muster.



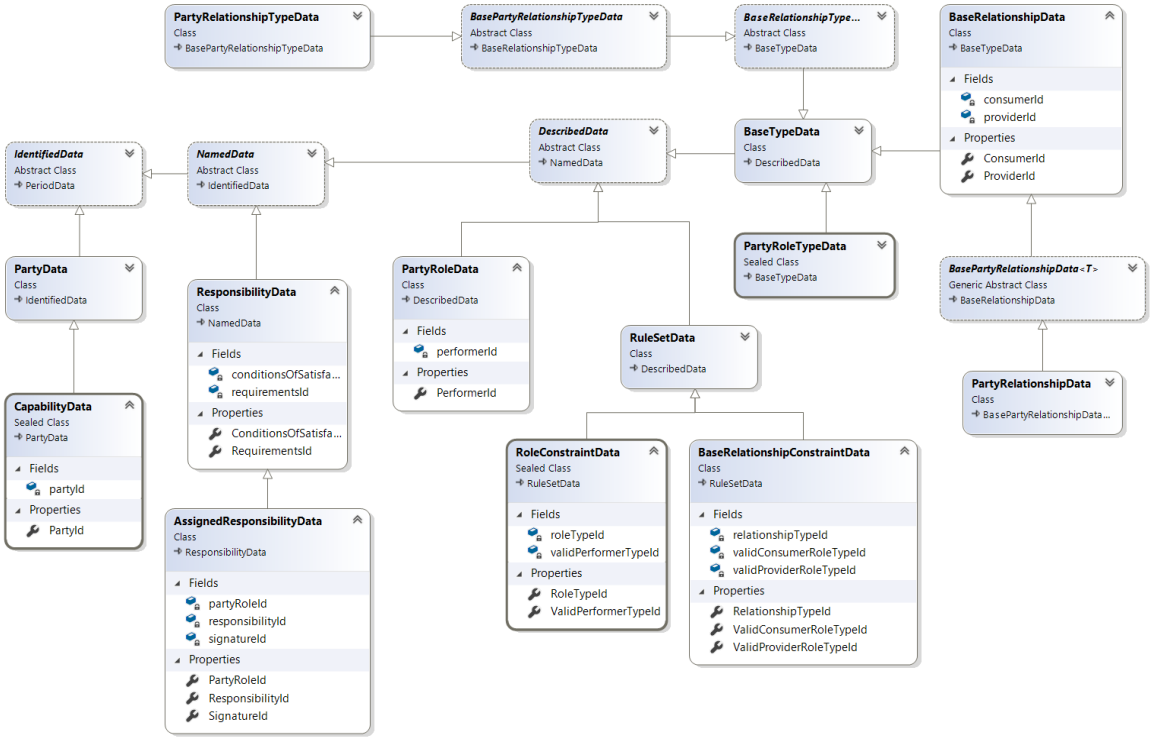
Joonis 17. Party arhetüübi Person teoreetiline muster [30].



Joonis 18. Projektis realiseeritud Person muster.



Joonis 19. PartyRelationship arhetüübi teoreetiline muster [30].



Joonis 20. Projektis realiseeritud PartyRelationship muster.

3.3 Kasutajaliides

Kasutajaliides valmib peamiselt Facade, Sentry, Grid ja Pages kihi klasside abil. Järgnevalt esitan Valuuta vaate, sellega seotud klassid ning kasutaja vaated koos kirjeldusega.

CurrencyView klassis luuakse kaks kohustuslikku omadust - IsoCode ja Symbol. Mõlema väärtused saadakse ja seatakse kas andmebaasist või kasutaja poolt sisestatuna. Lisaks on esitatud nende välja nimed ja tingimused, sest mõlemat välja kuvatakse kliendi vaates. Luuakse ka nimekiri CountryView põhjal, mida kasutatakse samuti kliendi vaates (joonis 21).

```
public class CurrencyView: NamedView
{
    private string isoCode;
    private string symbol;

    [Required]
    [StringLength(3, MinimumLength = 3)]
    [RegularExpression(RegularExpressionFor.EnglishCapitalsOnly)]
    [DisplayName("ISO Currency Code")]
    6 references | 0 exceptions
    public string IsoCode
    {
        get => getString(ref isoCode);
        set => isoCode = value;
    }

    [Required]
    [DisplayName("Currency symbol")]
    5 references | 0 exceptions
    public string Symbol
    {
        get => getString(ref symbol);
        set => symbol = value;
    }

    [DisplayName("Used in countries")]
    4 references | 0 exceptions
    public List<CountryView> UsedInCountries { get; } = new List<CountryView>();
}
```

Joonis 21. CurrencyView klass.

CurrencyViewFactory klassis oleva Create meetodi abil genereeritakse valuutad vaatesse. Lisaks leitakse ning esitatakse ka riigid kus antud valuutad on kasutusel.

```
public static class CurrencyViewFactory {
    5 references | 0 exceptions
    public static CurrencyView Create(Currency o) {
        var v = new CurrencyView {
            Name = o?.Data.Name,
            IsoCode = o?.Data.ID,
            Symbol = o?.Data.Code
        };
        if (o is null) return v;
        v.ValidFrom = o.Data.ValidFrom;
        v.ValidTo = o.Data.ValidTo;
        foreach (var c in o.UsedInCountries) {
            var country = CountryViewFactory.Create(c);
            v.UsedInCountries.Add(country);
        }
        return v;
    }
}
```

Joonis 22. CurrencyViewFactory klass.

CurrencyControlleris kasutatakse eelnevalt mainitud klasse nii HttpGet kui ka HttpPost atribuuti korral. Antud GetList meetodi abil määratakse valuutade põhjal tabeli suurus, lehekülgede arv ja sorteerimine. Lisaks kasutades nii CurrencyView'd ja CurrencyViewFactory't genereeritakse ning kutsutakse välja valuutade tabel.

```
[HttpGet("[action]")]
0 references | 0 requests | 0 exceptions
public async Task<IActionResult> GetList(
    [FromQuery] int pageNumber,
    [FromQuery] int pageSize,
    [FromQuery] SortingParams sortingParams)
{
    currencies.PageIndex = pageNumber + 1;
    currencies.PageSize = pageSize;
    currencies.SortOrder = setSortOrder(sortingParams);
    currencies.SortFunction = setSortFunction();
    var items = await currencies.GetObjectsList();
    var list = new List<CurrencyView>();
    foreach (var i in items)
    {
        list.Add(CurrencyViewFactory.Create(i));
    }
    return Ok(
        new
        {
            Items = list,
            TotalCount = currencies.Count
        });
}
```

Joonis 23. CurrencyController klass.

Joonisel 24 on näha valuuta vaate konfiguratsiooni. Antud meetod järgib Blazor FlexGrid'i GridView komponenti. Valuuta vaate konfiguratsioonis määratakse põhitabel ja temaga seotud detail tabel [15].

```
public class CurrenciesGrid : CustomGrid<CurrencyView>
{
    11 references | karukarlerik, 1 day ago | 2 authors, 3 changes
    public override void Configure(EntityTypeBuilder<CurrencyView> builder)
    {

        masterTable(builder);

        builder.HasDetailRelationship<CountryView>(c => c.IsoCode, o => o.Name)
                .HasLazyLoadingUrl($"api/Currencies/Countries")
                .HasPageSize(DetailTablePageSize)
                .HasCaption(GetMember.DisplayName<CurrencyView>(e => e.UsedInCountries));

        builder.AllowCreateItem(conf =>
             conf.CreateUri = "api/Currencies/Create"
        );
        builder.AllowInlineEdit(conf => {
            conf.AllowDeleting = true;
        });
        addColumn(builder, e => e.IsoCode);
        addColumn(builder, e => e.Symbol);
        addColumn(builder, e => e.Name);
        addColumn(builder, e => e.ValidFrom);
        addColumn(builder, e => e.ValidTo);
    }

}
```

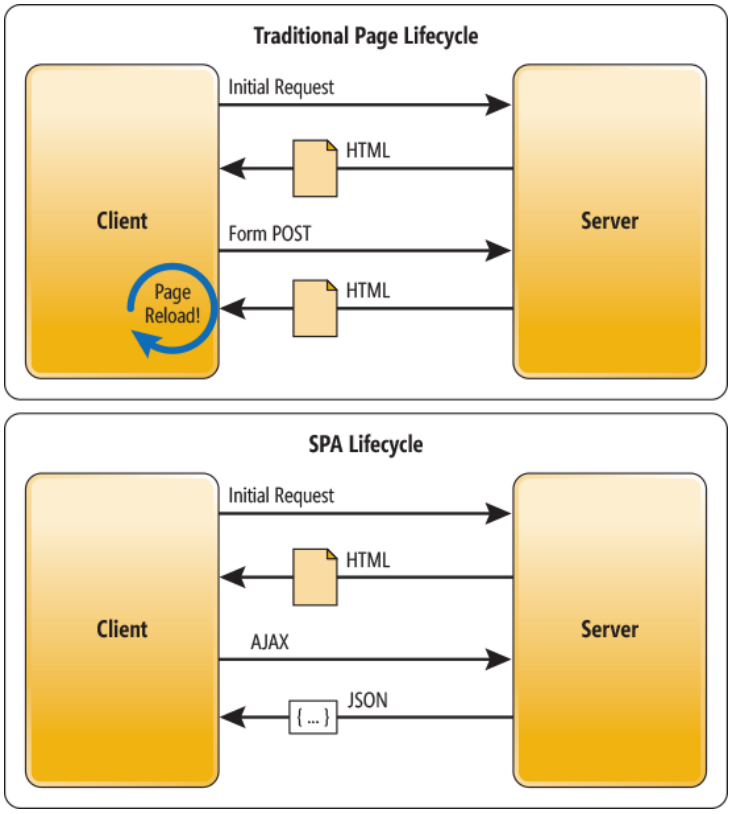
Joonis 24. CurrenciesGrid konfiguratsiooni klass.

Joonisel 25 on näha valuuta Razor vaadet, mis kasutab GridView komponente. Razor vaatele on iseloomulik html koodi kuvamine üleval ning funktsioonid all. Kasutades GridView'd ei pea me tabeleid ise looma kuna GridView kannab selle eest hoolt.

```
@using Open.Core

@inject HttpClient Http
@inject MasterTableDataAdapterBuilder<CurrencyView> MasterAdapterBuilder
@inject LazyLoadedTableDataAdapter<CountryView> countryAdapter
@inject LazyLoadedTableDataAdapter<CurrencyView> currencyAdapter

<h1>@Constants.CurrenciesPageTitle</h1>

<GridView DataAdapter="@masterAdapter"
          LazyLoadingOptions="@(new LazyLoadingOptions() {
                                    DataUri = "api/Currencies/ReadList/",
                                    PutDataUri = "api/Currencies/Update",
                                    DeleteUri = "api/Currencies/Delete/{ID}" })"
          PageSize="12"
          SaveOperationFinished="@ItemSavedOperationFinished"
          DeleteOperationFinished="@ItemDeletedOperationFinished"
          NewItemCreated="@ItemCreated">
</GridView>

@functions{
    MasterTableDataAdapter<CurrencyView> masterAdapter;
    protected override async Task OnInitAsync()
    {
        await Task.CompletedTask;
        masterAdapter = MasterAdapterBuilder
                .MasterTableDataAdapter(currencyAdapter)
                .WithDetailTableDataAdapter(countryAdapter)
                .Build();
    }
    public void ItemSavedOperationFinished(SaveResultArgs saveResultArgs)
    {
        Console.WriteLine($"Item saved result: {saveResultArgs.ItemSucessfullySaved}");
    }
    public void ItemDeletedOperationFinished(DeleteResultArgs deleteResultArgs)
    {
        Console.WriteLine($"Item deleted result: {deleteResultArgs.ItemSuccesfullyDeleted}");
    }
    public void ItemCreated(ItemCreatedArgs itemCreatedArgs)
    {
        Console.WriteLine($"Item created result: {itemCreatedArgs.CreatedItem}");
    }
}
```

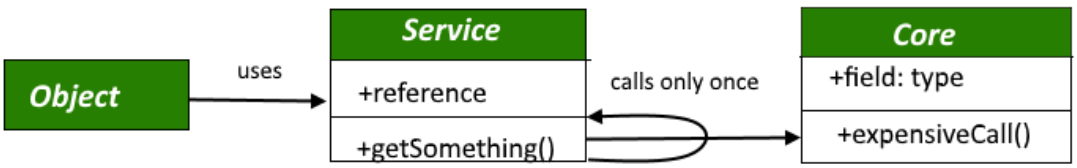
Joonis 25. Currencies vaate klass.

Kasutades Blazor'i raamistiku on meil võimalik luua Single-Page Application ehk SPA. Üheleheküljeline rakendus (SPA) on veebirakendus, mis suhtleb kasutajaga, muutes käesolevat lehekülge dünaamiliselt, mitte laadides kogu lehte serverist. SPA peamiseks eeliseks on kiirus ning reageerivus [31].



Joonis 26. Traditsioonilise lehekülje eluring ja SPA eluring [32].

Kasutades lisaks veel Lazy loading mustrit laaditakse kasutajale ainult vajalik kood. Seda tehes toimub esialgne laadimine kiiremini (kuna laetakse palju vähem koodi) ning kasvab ka üldine kiirus [33].



Joonis 27. LazyLoad disaini muster [34].

Lisades 4,5,6 on vastavalt välja toodud kasutajavaates:

- tabeli kuvamine koos detailidega
- andmete lisamine tabelisse
- andmete muutmine tabelis

3.4 Testid

Selleks, et tagada tarkvara maksimaalne funktsionaalsus on rakendus kaetud ühik- ja integratsiooni testitega. Ühiktestimine on tarkvara testimise etapp, kus testitakse tarkvara üksikuid komponente. Selle eesmärk on kinnitada, et iga üksus toimib nii nagu algselt planeeriti. Üksus on tavaliselt tarkvara väikseim testitav osa ning tavaliselt on tal üks või paar sisendit ja üks väljund [35].

Joonisel 28 ja 29 on kuvatud üks lihtne omadusi loov klass ning tema testid. Sellise klassi testimise esimene samm on luua suvaline PriceData'le iseloomulik objekt, millega on võimalik kontrollida kas antud omadusi on võimalik lugeda ja kirjutada.

```
public class PriceData: RuleSetData
{
    private string productTypeId;

    2 references | 0 exceptions
    public string ProductTypeId
    {
        get => getString(ref productTypeId);
        set => setValue(ref productTypeId, value);
    }
    2 references | 0 exceptions
    public decimal Amount {
        get;
        set;
    }
    2 references | 0 exceptions
    public string CurrencyID {
        get;
        set;
    }

    2 references | 0 exceptions
    public virtual CurrencyData Currency {
        get;
        set;
    }
}
```

Joonis 28. PriceData klass.

```
[TestClass]
0 references
public class PriceDataTests: ObjectTests<PriceData>
{
    99+ references | 0 exceptions
    protected override PriceData getRandomObject()
    {
        return GetRandom.Object<PriceData>();
    }
    [TestMethod]
    0 references | 0 exceptions
    public void ProductTypeIdTest()
    {
        canReadWrite(() => obj.ProductTypeId, x => obj.ProductTypeId = x);
    }
    [TestMethod]
    0 references | 0 exceptions
    public void CurrencyIDTest()
    {
        canReadWrite(() => obj.CurrencyID, x => obj.CurrencyID = x);
    }

    [TestMethod]
    0 references | 0 exceptions
    public void CurrencyTest()
    {
        canReadWrite(() => obj.Currency, x => obj.Currency = x);
    }
    [TestMethod]
    0 references | 0 exceptions
    public void AmountTest()
    {
        canReadWrite(() => obj.Amount, x => obj.Amount = x);
    }
}
```

Joonis 29. PriceData klassi testid.

Joonisel 30 ja 31 on kuvatud keerulisema ehitusega klass ning tema testid. Antud klass sisaldab mitut erinevat meetodit, konstruktorit ning nimekirja loomist. Sellise klassi esimene samm on samuti suvalise objekti loomine. Kasutades antud objekti saame alguses testida kas Termi ja Unit nimekirjad loodi korrektselt. Me vaatame, et loodud objekt ei oleks tühi või null väärtusega, kontrollime ta tüüpi ning võrdleme meie poolt etteantud arvuga. HasTerm ja HasUnit testides genereerime uuesti sobiliku objekti ning kasutame seda talle vastavas meetodis (HasTerm või HasUnit). Sealt saadud objekti loetelu vastus peab olema võrdne meie poolt etteantud numbriga.

```
public sealed class Measure : Metric<MeasureData>
{
    private readonly List<MeasureTerm> terms;
    private readonly List<Unit> units;

    7 references | 0 exceptions
    public Measure() : this(null) { }

    8 references | 0 exceptions
    public Measure(MeasureData r) : base(r ?? new MeasureData())
    {
        terms = new List<MeasureTerm>();
        units = new List<Unit>();
    }
    7 references | 0 exceptions
    public IReadOnlyList<MeasureTerm> Terms => terms.AsReadOnly();
    8 references | 0 exceptions
    public IReadOnlyList<Unit> Units => units.AsReadOnly();

    3 references | 0 exceptions
    public void HasTerm(MeasureTerm measureTerm) {
        if (measureTerm is null) return;
        if (measureTerm.Data.MeasureID != Data.ID) return;
        if (terms.Find(x => x.Data.TermID == measureTerm.Data.TermID) != null)
            return;
        terms.Add(measureTerm);
    }

    3 references | 0 exceptions
    public void HasUnit(Unit unit) {
        if (unit is null) return;
        if (unit.Data.MeasureId != Data.ID) return;
        if (units.Find(x => x.Data.ID == unit.Data.ID) != null)
            return;
        units.Add(unit);
    }
}
```

Joonis 30. Measure klass.

```
[TestClass] public class MeasureTests : ObjectTests<Measure> {
    99+ references | 0 exceptions
    protected override Measure getRandomObject()
    {
        var d = GetRandom.Object<MeasureData>();
        return new Measure(d);
    }
    0 references | 0 exceptions
    [TestMethod] public void TermsTest() {
        Assert.IsNotNull(obj.Terms);
        Assert.IsInstanceOfType(obj.Terms, typeof(IReadOnlyList<MeasureTerm>));
        Assert.AreEqual(0, obj.Terms.Count);
    }
    [TestMethod]
    0 references | 0 exceptions
    public void UnitsTest()
    {
        Assert.IsNotNull(obj.Units);
        Assert.IsInstanceOfType(obj.Units, typeof(IReadOnlyList<Unit>));
        Assert.AreEqual(0, obj.Units.Count);
    }
    [TestMethod]
    0 references | 0 exceptions
    public void HasTermTest() {
        var t = GetRandom.Object<MeasureTermData>();
        obj.HasTerm(new MeasureTerm(t));
        Assert.AreEqual(0, obj.Terms.Count);
        t.MeasureID = obj.Data.ID;
        obj.HasTerm(new MeasureTerm(t));
        Assert.AreEqual(1, obj.Terms.Count);
    }
    [TestMethod]
    0 references | 0 exceptions
    public void HasUnitTest() {
        var u = GetRandom.Object<UnitData>();
        obj.HasUnit(new Unit(u));
        Assert.AreEqual(0, obj.Units.Count);
        u.MeasureId = obj.Data.ID;
        obj.HasUnit(new Unit(u));
        Assert.AreEqual(1, obj.Units.Count);
    }
```

Joonis 31. Measure klassi testid.

Integratsiooni testimine on tarkvara testimise etapp, kus üksikud tarkvara moodulid on ühendatud ning testitud ühtse grupina. Tavaliselt on see järgmine samm peale ühiktestimist. Integratsiooni testimise peamiseks põhjuseks on see, et ühiktestid ei kata kõiki funktsioone ära (andmebaas, kolmanda osapoole paketid, API kontrollerid, jne) [36].

4 Analüüs ja arutelu

Antud peatükis kirjeldan lühidalt konkureerivaid süsteeme, seletan milleks on uut LIMS süsteemi vaja, annan ülevaate enda poolt tehtud tööst ning esitan edaspidise projekti plaani.

4.1 Millised on peamised konkureerivad süsteemid?

LIMS ei ole ainuke labori infosüsteem turul ning juhul kui klient otsustab LIMS´i kasuks siis tuleb tal välja valida kõige sobivam süsteem kuna pakkujaid on palju. Populaarsemad labori infosüsteemid on samas arendatud suurfirmade poolt, kelle pakutavad rakendused on kallid ning kaasnevad muude tingimustega (koolitused, tarkvara haldajad, pilve teenused, jne). Hinnakiri algab tavaliselt üpris kõrgelt ning nö limiit puudub kuna tasutakse aasta ning tarbijate hulga põhiselt [37].

Populaarsemad LIMS'i konkureerivad labori süsteemid on [38]:

- Electronic lab notebook (ELN) ehk Elektrooniline labori märkmik
- Scientific data management system (SDMS) ehk Teadusandmete haldamise süsteem
- Chromatography data system (CDS) ehk Kromatograafiliste andmete süsteem
- Laboratory execution system (LES) ehk Laboratooriumi täitmise süsteem

4.2 Miks on vaja uut LIMS süsteemi ning mis on selles teisiti?

Kuna suurem osa LIMS süsteemi pakkujatest on suurettevõtted, siis on neil võim kontrollida turgu. See annab neile võimaluse lisada süsteemile ebavajalike lisakulutusi ja tingimusi. Teiste ettevõtete poolt loodud süsteemid võivad olla küll odavamad kuid ei pruugi pakkuda sama funktsionaalsust. Sellest tulenevalt on laborite valik ostu puhul piiratud.

Projekti käigus arendatav LIMS süsteemil on kaks põhilist eelist teiste süsteemide ees. Praegused süsteemid toetavad ainult ühte laborit kuid meie süsteem hakkaks toetama paljusi. See annaks laboritele võimaluse kogu süsteemi ise vajadusel muuta ehk igal laboril võib olla oma äriprotsessid ja andmestruktuurid, mis pannakse süsteemi poolt ühtsesse arhetüüpide struktuuri. Tänu sellele vabaneb labor sisseostetud kolmandast osapoolest ja teistest paketiga kaasnevatest kohustustest, mis praeguste süsteemide puhul on tava. Seetõttu oleks meie poolt pakutava labori infosüsteemi maksumus ka madalam. Teiseks eeliseks on andmete korduskasutatavus ehk me võimaldame andmete kättesaadavust ka teistele laboritele ja uurijatele. Lisaks on tegemist kliendipoolse veebirakendusega mis tähendab kiiret ja kasutajasõbralikku süsteemi.

4.3 Milline oli töö autori panus antud süsteemi arendamisse?

Bakalaureusetöö käigus sain ma tegeleda nii teoreetilise kui ka tehnilise poolega. Peamiseks kohustuseks, viimase kahe semestri jooksul, oli tutvuda juba olemasoleva LIMS süsteemiga - uurida koodi ning üritada aru saada selle eesmärgist. Kuna süsteem vajas üleviimist teisele platvormile, siis tuli selgeks teha ka uute raamistike tööpõhimõtted ja rakendusviisid. Kui mingi kindla osa teoreetiline pool sai uuritud, andis juhendaja tavaliselt ülesande, mida tuli ise tarkvara tehnilisel poolel rakendada. Palju tunde kulus tavaliselt meetodite ümberkirjutamisele ning testimisele kuid raskuste esinemise korral sai abi juhendajalt. Hetkel on umbes 42000 koodirida koodi ja 34500 koodirida teste (arvestades ka kommentaare, mis pärinevad eelnevast projektist kuid on sisuliselt dokumenteeritud nõuded ja juhised edasiarendamiseks). Minu osa sellest on hinnanguliselt 1000 rida loogikast, 500 infrast, 7000 Blazori sisseviimisest ja 1500 testimisest ehk umbes 10000 koodirida aasta jooksul. Lisaks on koodi hallatavuse indeks kõrge, mis on iseloomulik puhtale koodile. Märgata on ka koodiridade erinevust vanema Razor (12085 koodirida) süsteemi ja uuema Blazor (6883 koodirida) SPA süsteemi vahel.

Code Metrics Results

Filter: None Min:

Hierarchy	Maintainability In...	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Co...
Aids (Debug)	79	281	1	135	875
Core (Debug)	95	181	2	35	594
Data (Debug)	94	541	11	100	1,952
Domain (Debug)	94	715	7	335	8,922
Facade (Debug)	94	856	11	244	8,312
Grid (Debug)	58	17	2	86	309
Infra (Debug)	80	548	4	686	2,517
Pages (Debug)	76	167	3	110	665
Razor (Debug)	80	1,004	6	692	11,420
Sentry (Debug)	88	379	3	109	5,551
Server (Debug)	79	249	4	240	1,023
Tests (Debug)	90	3,194	10	1,922	34,507

Joonis 32. Projekti koodiridade arv.

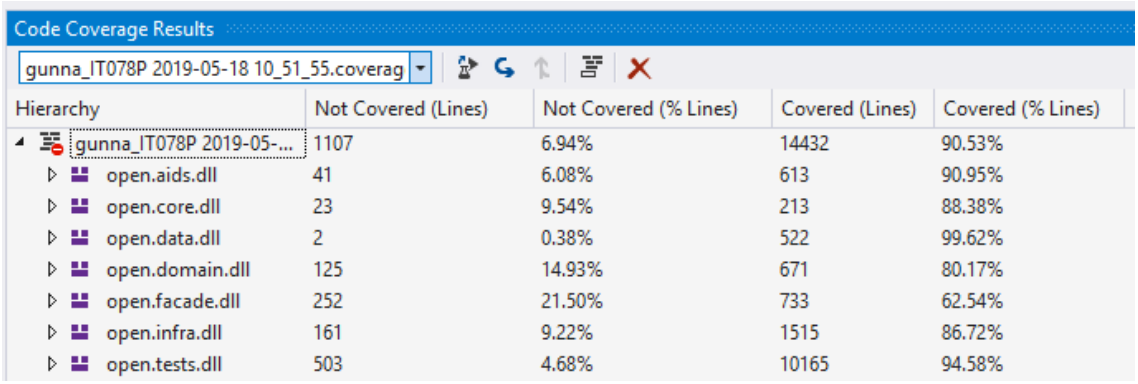
Antud töö on andnud mulle väga palju uusi teadmisi mida ma varem poleks tõenäoliselt ise selgeks õppinud. Artiklite, raamatute ja teiste koodi lugemine ning pidev iseseisev programmeerimine on aidanud mul paremini mõista tarkvara koodi, selle loogikat ning kirjutatud koodi tähtsust. Puhta arhitektuuri ja koodi põhimõtteid kavatsen ma järgida ka edaspidistes arendustes. Palju motivatsiooni sain ma ka teadmisest, et arendatav süsteem on midagi reaalset ja implementeeritavat.

10.apr	11:00	16:30	5:30t	Töö Blazoriga
10.apr	21:00	00:00	3t	Töö Blazoriga
11.apr	11:00	14:00	3t	Töö Blazoriga
11.apr	16:00	18:00	2t	Lõputöö ja koosolek
15.apr	11:00	21:00	10t	Blazor ning GridView
16.apr	13:00	21:00	7t	Blazor ning GridView
17.apr	14:00	20:30	6:30t	Lõputöö
18.apr	11:00	21:00	9t	Lõputöö ja koosolek
21.apr	19:00	00:30	5:30t	Lõputöö
22.apr	14:30	20:00	5:30t	Lõputöö
23.apr	13:00	20:00	7t	Lõputöö ja Measure/Unit vaatele
24.apr	12:00	15:00	3t	Lõputöö ja Measure/Unit vaatele
24.apr	19:30	01:00	5:30t	Lõputöö ja Measure/Unit vaatele
25.apr	18:00	20:00	2t	Koosolek
29.apr	17:00	22:00	5t	Lõputöö
30.apr	11:00	20:00	9t	Lõputöö ja UnitTerms vaatele

Joonis 33. Lõik ajakulu graafikust.

4.4 Nõuded ja edasise arendamise kava

Kuna bakalaureusetöö peamine eesmärk (ettevalmistus uuendamiseks ja edasiarendamiseks) sai täidetud, siis sain ma vajalikud baasteadmised edasiseks arendamiseks ning lisaks valmis lõputöö käigus ka labori infosüsteemi algeline prototüüp. Peamiseks nõudeks edaspidises arenduses on prototüübist toote välja arendamine. Selleks on vaja puudulike kuid vajalike funktsionaalsuste välja töötamine ning nende ning ka veel lisamata jäänud funktsioonide realiseerimine. Sinna alla kuulub näiteks uute arhetüüpi mustrite teostamine ning sisse logimise ja registreerimise süsteem. Probleeme (salvestamine, andmete lisamine, kliendivaate pool) esineb raamistikega, mis on alles arendamisjärgus kuid nendest anname arendajatele ka ise märku, kui ka Azure teenusega, mis ei toeta veel ASP.NET Core 3.0 versiooni. Tuleb jätkata puhta koodi kirjutamist ning ühik- ja integratsiooni testimist. Joonisel 34 on võimalik näha, et suurem osa koodist on testidega kaetud kuid testimist vajavad veel Grid, Pages, Razor, Sentry ja Serveri kiht. Lisaks on vajalik luua võimalikult palju universaalseid klasse, mida saaks ära kasutada sarnaste tööpõhimõttega funktsioonide puhul.



Joonis 34. Koodi katvus testitega.

Antud lõputöö käigus käsitletavat rakendust on plaanis jätkata ka magistriõppe raames. Kui bakalaureuse lõputöö ajal oli üheks peamiseks eesmärgiks nii uue kui ka vana LIMS rakenduse selgeks õppimine, siis magistriõppe raames on plaanis rohkem tehnilise poolega tegeleda, et kasutajavalmis rakendus saaks valmis arendatud. Tõenäoliselt toimub edaspidine arendamine suuremas meeskonnas, juhendades juba bakalaureuse tudengeid.

5 Kokkuvõte

Antud bakalaureusetöö olulisemaks eesmärgiks oli ettevalmistumine labori infosüsteemi uuele platvormile üleviimiseks ning edasiseks arendamiseks. Eesmärgi saavutamiseks tutvuti kõigepealt olemasoleva LIMS rakenduse ja kasutusele võetavate raamistikega. Seejärel viidi vana rakendus üle ASP.NET Core platvormile ning kaeti ühiktestitega.

Vana rakenduse uuendamisel veebirakenduseks pöörati tähelepanu eelkõige funktsionaalsustele mida labori infosüsteemis vaja läheks - peamine neist oli CRUD operatsioon (loo, loe, uuenda ja kustuta). Lisaks tuli koodi kirjutamisel järgida puhta koodi ja arhitektuuri põhimõtteid, et hoida koodihulka minimaalsena ja arusaadavana.

Lõputöö eesmärk sai saavutatud kuna tulemusena saadi vajalikud baasteadmised, mida edasises arenduses magistritöö raames saab kasutada teiste juhendamiseks. Lisaks valmis ka uuel platvormil veebirakenduse algeline prototüüp, mis on kasutajasõbralik üheleheline veebirakendus ning hea jätkupunkt edasiseks arenduseks.

Kasutatud kirjandus

[1] Singularityhub.com, „SingularityHub,“ 15 April 2019. [Võrgumaterjal]. Available: https://singularityhub.com/2016/03/22/technology-feels-like-its-accelerating-because-it-actually-is/.

[2] Noted.co.nz, „Noted,“ 16 April 2019. [Võrgumaterjal]. Available: https://www.noted.co.nz/tech/the-rate-of-technological-change-is-now-exceeding-our-ability-to-adapt/.

[3] PandaSecurtiy, „pandasecurity.com/mediacenter,“ 16 April 2019. [Võrgumaterjal]. Available: https://www.pandasecurity.com/mediacenter/tips/the-importance-of-updating-systems-and-software/.

[4] I. C. Society, „https://www.computer.org,“ 16 April 2019. [Võrgumaterjal]. Available: https://www.computer.org/csdl/magazine/so/2018/06/mso2018060062/17D45Xtvpds.

[5] Wikipedia, „wikipedia.org,“ 17 April 2019. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Laboratory_information_management_system.

[6] SooperArticles, „sooperarticles.com,“ 23 April 2019. [Võrgumaterjal]. Available: https://www.sooperarticles.com/business-articles/management-articles/3-differences-between-lims-lis-1274979.html.

[7] Illumina, „Illumina.com,“ 23 April 2019. [Võrgumaterjal]. Available: https://www.illumina.com/informatics/sample-experiment-management/lims.html.

[8] IEEExplore, „ieeexplore.ieee.org,“ 22 April 2019. [Võrgumaterjal]. Available: https://ieeexplore.ieee.org/document/5546330.

[9] Wikipedia, „wikipedia.org,“ 25 April 2019. [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/C_Sharp.

[10] Microsoft, „dotnet.microsoft.com,“ 1 May 2019. [Võrgumaterjal]. Available: https://dotnet.microsoft.com/learn/web/what-is-aspnet-core.

[11] Stackify, „stackify.com,“ 1 May 2019. [Võrgumaterjal]. Available: https://stackify.com/asp-net-core-features/.

[12] Weblogs, „weblogs.asp.net,“ 5 May 2019. [Võrgumaterjal]. Available: https://weblogs.asp.net/scottgu/introducing-razor.

[13] Wikipedia, „wikipedia.org,“ 6 May 2019. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Blazor.

[14] L. Blazor, „learn-blazor.com,“ 6 May 2019. [Võrgumaterjal]. Available: https://learn-blazor.com/getting-started/what-is-blazor/.

[15] Github, „github.com,“ 8 May 2019. [Võrgumaterjal]. Available: https://github.com/Mewriick/Blazor.FlexGrid.

[16] R. C. Martin, Clean Code A Handbook of Agile Software Craftsmanship, Prentice Hall, 2009.

[17] I. C. Society, „computer.org,“ 28 April 2019. [Võrgumaterjal]. Available: https://www.computer.org/csdl/proceedings-article/2018/aswec/17D45VtKirP/17D45W1Oa5o.

[18] Informit, „informit.com,“ 2 May 2019. [Võrgumaterjal]. Available: http://www.informit.com/articles/article.aspx?p=1711821.

[19] Medium, „engineering.21buttons.com,“ 11 May 2019. [Võrgumaterjal]. Available: https://engineering.21buttons.com/clean-architecture-in-django-d326a4ab86a9.

[20] Wikipedia, „wikipedia.org,“ 5 May 2019. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Software_development_process.

[21] I. C. Society, „computer.org,“ 3 May 2019. [Võrgumaterjal]. Available: https://www.computer.org/csdl/magazine/so/2019/02/08648272/17QjJcTQWBp.

[22] T. Ülikool, „cs.tlu.ee,“ 3 May 2019. [Võrgumaterjal]. Available: http://www.cs.tlu.ee/~inga/TTP/Agiilsed_meetodid_2018.pdf.

[23] Agilemanifesto, „agilemanifesto.org,“ 6 May 2019. [Võrgumaterjal]. Available: http://agilemanifesto.org/iso/et/manifesto.html.

[24] Wikipedia, „wikipedia.org,“ 6 May 2019. [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Agiilsed_metoodikad.

[25] 360logica, „360logica.com,“ 6 May 2019. [Võrgumaterjal]. Available: https://www.360logica.com/blog/the-importance-of-different-agile-methodologies-included-in-agile-manifesto/.

[26] Wikipedia, „wikipedia.org,“ 2 May 2019. [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Ekstreemprogrammeerimine.

[27] Wikipedia, „wikipedia.org,“ 8 May 2019. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Test-driven_development.

[28] K. Beck, Test-Driven Development By Example, Addison-Wesley Professional, 2002.

[29] Microsoft, „docs.microsoft.com,“ 6 May 2019. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/standard/modern-web-apps-azure-architecture/common-web-application-architectures.

[30] I. N. Jim Arlow, Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML, Addison-Wesley Professional, 2004.

[31] Wikipedia, „wikipedia.org,“ 7 May 2019. [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Single-page_application.

[32] ResearchGate, „researchgate.net,“ 6 May 2019. [Võrgumaterjal]. Available: https://www.researchgate.net/figure/Traditional-vs-SPA-lifecycle_fig4_330015992.

[33] Medium, „medium.com,“ 5 May 2019. [Võrgumaterjal]. Available: https://medium.com/wolox-driving-innovation/divide-and-conquer-lazy-loading-for-your-spa-32eb63149e76.

[34] GeeksforGeeks, „geeksforgeeks.org,“ 6 May 2019. [Võrgumaterjal]. Available: https://www.geeksforgeeks.org/lazy-loading-design-pattern/.

[35] S. t. fundamentals, „softwaretestingfundamentals.com,“ 6 May 2019. [Võrgumaterjal]. Available: http://softwaretestingfundamentals.com/unit-testing/.

[36] D. n. curry, „www.dotnetcurry.com,“ 6 May 2019. [Võrgumaterjal]. Available: https://www.dotnetcurry.com/aspnet-core/1420/integration-testing-aspnet-core.

[37] Limswiki, „limswiki.org,“ 6 May 2019. [Võrgumaterjal]. Available: https://www.limswiki.org/index.php/LIMS_vendor.

[38] Limswiki, „limswiki.org,“ 6 May 2019. [Võrgumaterjal]. Available: https://www.limswiki.org/index.php/Laboratory_informatics.

[39] LimsWiki, „limswiki.org,“ 6 May 2019. [Võrgumaterjal]. Available: https://www.limswiki.org/index.php/Laboratory_information_system#Differences_between_a_LIS_and_LIMS.

Lisa 1 – Uus CountriesController klass

```
[Route("api/[controller]")]
1 reference
public class CountriesController : SimpleTableController<CountryView, Country, CountryData>
{
    5 references
    protected internal override Expression<Func<CountryView, object>> getId => v => v.Alpha3Code;

    0 references
    public CountriesController(ICountriesRepository r) : base(r) { }

    5 references
    protected internal override void updateObject(Country o, CountryView v)
    {
        o.Data.Name = v.Name;
        o.Data.Code = v.Alpha2Code;
        o.Data.ValidFrom = v.ValidFrom;
        o.Data.ValidTo = v.ValidTo;
    }
    3 references
    protected internal override string matchViewAndDataNames(string name)
    {
        if (name == GetMember.Name<CountryView>(x => x.Alpha3Code)) return GetMember.Name<CountryData>(x => x.ID);
        if (name == GetMember.Name<CountryView>(x => x.Alpha2Code)) return GetMember.Name<CountryData>(x => x.Code);
        return name;
    }

    5 references
    protected internal override CountryView createView(Country o)
    {
        return CountryViewFactory.Create(o);
    }
    5 references
    protected internal override Country createObject(CountryView v)
    {
        return CountryViewFactory.Create(v);
    }
}
```

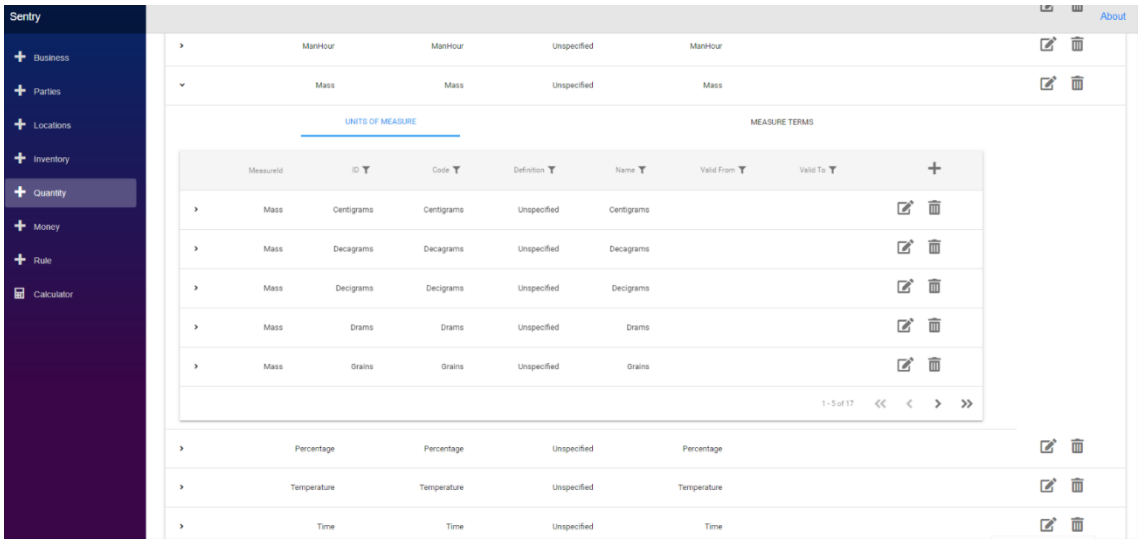
Lisa 2 – Vana CountriesController klass

```
[Route("api/[controller]")]
1 reference
public class CountriesController : ControllerBase
{
    internal protected readonly ICountriesRepository countries;
    0 references
    public CountriesController(ICountriesRepository co) { countries = co; }
    2 references
    internal protected Func<CountryData, object> setSortFunction() { return x => x.Name; }
    [HttpGet("[action]")]
    0 references
    public async Task<IActionResult> GetList(
        [FromQuery] int pageNumber,
        [FromQuery] int pageSize,
        [FromQuery] SortingParams sortingParams) {
        countries.PageIndex = pageNumber + 1;
        countries.PageSize = pageSize;
        countries.SortOrder = setSortOrder(sortingParams);
        countries.SortFunction = setSortFunction();
        var items = await countries.GetObjectsList();
        var list = new List<CountryView>();
        foreach (var i in items) {
            list.Add(CountryViewFactory.Create(i));
        }
        return Ok( new { Items = list, TotalCount = countries.Count });
    }
    [HttpPost("[action]")]
    0 references
    public async Task<IActionResult> GetList(
        [FromQuery] int pageNumber,
        [FromQuery] int pageSize,
        [FromQuery] SortingParams sortingParams,
        [FromBody] IEnumerable<FilterDefinition> filters) {
        countries.PageIndex = pageNumber + 1;
        countries.PageSize = pageSize;
        countries.SortOrder = setSortOrder(sortingParams);
        countries.SortFunction = setSortFunction();
        countries.SearchString = setSearchString(filters);
        var items = await countries.GetObjectsList();
        var list = new List<CountryView>();
        foreach (var i in items) {
            list.Add(CountryViewFactory.Create(i));
        }
        return Ok( new { Items = list, TotalCount = countries.Count });
    }
    1 reference
    private string setSearchString(IEnumerable<FilterDefinition> filters) {
        if (filters is null) return string.Empty;
        foreach (var e in filters)
            return e?.Value?.ToString() ?? string.Empty;
        return string.Empty;
    }
    2 references
    private SortOrder setSortOrder(SortingParams sortingParams) {
        if (sortingParams is null) return SortOrder.Ascending;
        if (sortingParams.SortDescending) return SortOrder.Descending;
        return SortOrder.Ascending;
    }
    [HttpPut("[action]")]
    0 references
    public ActionResult<CountryView> Update([FromBody] CountryView viewData) {
        return viewData;
    }
    [HttpDelete("[action]")]
    0 references
    public async Task<IActionResult> Delete([FromQuery(Name = nameof(
        CountryView.Alpha3Code))] string id) {
        if (id is null) return NoContent();
        var c = await countries.GetObject(id);
        if (c is null) return NotFound();
        await countries.DeleteObject(c);
        return Ok();
    }
}
```

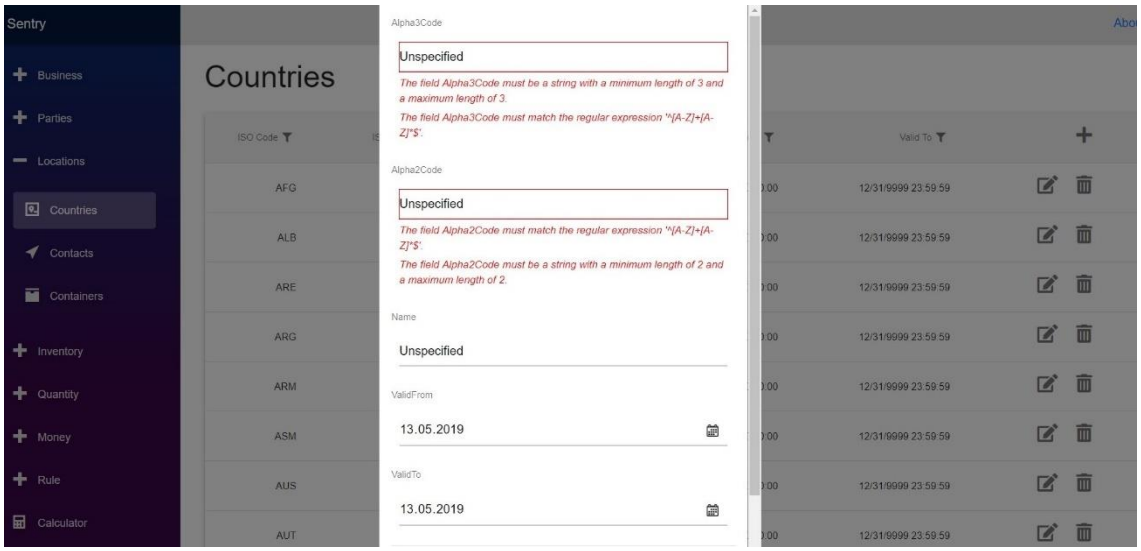
Lisa 3 – Universaalne kontroller andmete lugemise näitel

```
[HttpGet("[action]")]
0 references | karukarlerik, 8 days ago | 2 authors, 2 changes
public async Task<IActionResult> ReadList(
    [FromQuery] int pageNumber,
    [FromQuery] int pageSize,
    [FromQuery] SortingParams sortingParams)
{
    return await readList(pageNumber, pageSize, sortingParams, null);
}

[HttpPost("[action]")]
0 references | karukarlerik, 8 days ago | 2 authors, 3 changes
public async Task<IActionResult> ReadList(
    [FromQuery] int pageNumber,
    [FromQuery] int pageSize,
    [FromQuery] SortingParams sortingParams,
    [FromBody] IEnumerable<FilterDefinition> filters)
{
    return await readList(pageNumber, pageSize, sortingParams, filters);
}

2 references | karukarlerik, 8 days ago | 1 author, 1 change
private async Task<IActionResult> readList(int pageNumber, int pageSize, SortingParams sortingParams, IEnumerable<FilterDefinition> filters)
{
    repository.PageIndex = pageNumber + 1;
    repository.PageSize = pageSize;
    repository.SortOrder = setSortOrder(sortingParams);
    repository.SortFunction = setSortFunction(sortingParams);
    repository.Filter = setFilters(filters);
    var items = await repository.GetObjectsList();
    var list = new List<TView>();
    foreach (var i in items)
    {
        list.Add(createView(i));
    }
    return Ok(new
        {
            Items = list,
            TotalCount = repository.Count
        });
}
```

Lisa 4 – Prototüübi „Tabel Measure detailidega” kliendivaade



Lisa 5 – Prototüübi „Andmete lisamine” kliendivaade



Lisa 6 – Prototüübi „Andmete muutmine” kliendivaade