TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Tarkvarateaduse instituut

Tauri Türkson 143014IABB

RAHA ARHETÜÜPMUSTRI KUI VALDKONNAMUDELI REALISEERIMISE NÄIDE „INFOSÜSTEEMIDE ARENDAMINE II“ ÕPETAMISEL

bakalaureusetöö

Juhendaja: Gunnar Piho

dotsent

Tallinn 2018

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: Tauri Türkson

20.05.2018

Annotatsioon

Käesoleva töö „Raha arhetüüpmustri kui valdkonnamudeli realiseerimise näide "Infosüsteemide arendamine II" õpetamisel“ peamiseks eesmärgiks on võtta Jim Arlow ja Ila Neustadt poolt kirja pandud raha valdkonna mudel ja realiseerida see MVC rakendusena.

Raha valdkonnamudeli realiseerimisel järgitakse puhta arhitektuuri ja puhta koodi põhimõtteid.

Töö tulemusena on loodud terviklik MVC rakenduse prototüüp, kus raha valdkonnamudeli näitel on realiseeritud andmete juurdepääsukiht, valdkonnamudeli kiht ja rakenduse esitluskiht.

Võrdlen oma töös ka teisi publitseeritud raha mudeleid ning toon välja erinevused, võttes aluseks Jim Arlow ja Ila Neustadt’i poolt kirja pandud mudeli.

Antud tööd saavad näidisprojektina kasutada üliõpilased oma individuaalsete projektide koostamisel aine "Infosüsteemide arendamine II" raames.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 32 leheküljel, 3 peatükki, 32 joonist.

Abstract

The main aim of work, „The money archetype pattern as a domain model example in the "Information Systems Development II" course“ is to take money domain model, described by Jim Arlow and Ila Neustadt and to realize it as a Model-View-Controller application.

When implementing money domain model, clean architecture and clean code principles are followed.

As a result of the work, complete MVC application prototype is created, where by example of money domain model data access, domain logic and presentation layers are realized.

In the work, I will also compare other published money models and bring out the differences, setting Jim Arlow and Ila Neustadt model as a base model.

Finished work is used as sample project for students creating their own individual project in the "Information Systems Development II" course.

The thesis is written in Estonian and contains 32 pages of text, 3 chapters, 32 figures.

Sisukord

Jooniste loetelu

Sissejuhatus

Valdkonnamudel (*domain model*) kirjeldab tegevusvaldkonda iseloomustavate olemite vahelisi seoseid [1], luues selleks omavahel seotud objektide kogumi. Objektiks võib olla nii suurettevõte kui ka tellimuse rida. Iga objekt omab teatud määral andmeid, mis üldjuhul esitatakse atribuutidena [2].

Töös realiseeritakse raha arhetüüpmustri valdkonnamudel, mida kasutatakse Tallinna Tehnikaülikoolis õppeaine „Infosüsteemida arendamine II“ [3] õpetamisel. Aine kuulub äriinfotehnoloogia 2017 aasta õppekavasse [4].

Mudeli realiseerimisel kasutatakse MVC (*model-view-controller*) arhitektuurimustrit, mis on jagatud vastutusega süsteem, kus üks osa rakendusest tegeleb päringute vastuvõtmisega, interpreteerimisega ja rakenduse töö juhtimisega (*controller*), teine osa tegeleb päringute töötlemisega ja töötlemise tulemusel tekitatud andmestruktuuride hoidmisega (*model*) ja kolmas osa (*view*) tegeleb päringute töötlemise tulemusel toodetud andmete vormindamise ja näitamisega (kliendile saatmisega) [5].

Probleem: Üliõpilastele demonstreerimiseks on vajadus luua näiteprojekt, mille abil saab õpetada valdkonnamudelite alusel infosüsteemide realiseerimist. Näiteprojekti põhjal saavad tudengid luua enda iseseisvaid projekte infosüsteemide arendamise aine raames.

Eesmärk: Võtta Jim Arlow ja Ila Neustadt poolt kirja pandud raha valdkonna mudel [6] ja see realiseerida MVC rakendusena, tuginedes testidel tuginevale arendusele ja puhtale koodile [7].

Töö struktuur: Esimeses peatükis kirjeldatakse töö teoreetilisi ja metoodilisi aluseid ning tuuakse erinevaid näiteid nende paremaks mõistmiseks. Teises peatükis jõutakse diplomitöö põhiosani, kus antakse ülevaade tehtud tööst. Kolmas peatükk keskendub peaasjalikult analüüsile ja järeldustele, kus võetakse kokku tehtud töö sisu. Tuuakse välja töö põhitulemused ning vastatakse küsimusele, kas postitatud eesmärk sai täidetud.

1 Teoreetilised ja metoodilised alused

1.1 ASP.NET Core

Valitud mudel realiseeritakse Microsoft Visual Studio Enterprise 2017 tarkvara ja ASP.NET Core 2.0 raamistikku [8] kasutades. Need vahendid on kasutusel ka aine „Infosüsteemide arendamine II“ õpetamisel.

Ettevõtte infosüsteemide arendamine ASP.NET Core vahendeid kasutades on vaatluse all, sest ASP.NET Core toetab:

- Erinevaid platvorme;
- Mikroteenuseid;
- Konteinertehnoloogiat (Docker);
- Skaleeritavust ja jõudlust; ning
- Samas serveris erinevate .NET versioonide kasutamist [9].

Kõik see oli võimalik ka traditsiooniliste .NET rakenduste korral, kuid ASP.NET Core on eelmainitutele optimeeritud.

1.2 Puhas kood

Puhas kood on:

- **Loetav** - Puhta koodi eesmärk on suurendada koodi loetavust - koodile peale vaadates on koheselt arusaadav, mida kood teeb. Oluline on valida õiged nimetused ja vältida selgitavaid kommentaare (kood peaks olema loetav ilma kommentaarideta). Väljakommenteeritud kood ei ole puhas.
- **Muudetav** - Mingi muutuja väärtuse muutmiseks peab seda vajadusel saama muuta võimalikult vähestes kohtades, moonutamata olemasolevat funktsionaalsust. Kui süsteem on ehitatud viisil, et seda on raske muuta, võib rakenduse mahu kasvades vea hilisem parandamine osutuda üsna kulukaks.

- **Struktuurne** - Erinevatel elementidel on teatud süsteemne järjestus.
- **Taaskasutatav** - Mingit osa koodist peab olema võimalik kasutada nii mitmeski teises kohas (universaalsus). Koodi ei tohi dubleerida.
- **Testitav** - Kood peab olema täielikult testitav. Testid peavad olema lihtsad ja mitte ajamahukad.
- **Fokusseeritud** - Iga funktsioon, meetod, klass ja muu moodul peab täitma vaid üht ülesannet ja ei midagi rohkemat. Koodiblokid peavad olema üksteisest eraldatavad ja toimima eraldiseisvate üksustena. Koodi hulk peab olema võimalikult minimaalne.
- **Korratav** - Kood peab igal käivitumisel andma sama tulemuse – tuleb vältida raskesti korratavaid olekuid [10].

1.3 Puhas arhitektuur

Lähtuvalt vastutusaladele (*responsibility of concerns)* jaotatakse rakendused kihtideks. Üldiselt jagatakse kihid kolme kategooriasse. Seda illustreerib Joonis 1 [11].

Joonis 1 Rakenduse arhitektuur

Kasutades seda arhitektuuri, teevad kasutajad päringu kasutajaliidese (*User Interface*) kihi kaudu, mis suhtleb ainult loogikakihiga (*Business Logic).* Loogikakiht saab aga omakorda pöörduda vaid andmekihi (*Data Access*) poole, pärides sealt andmeid. Kasutajaliidese kiht ei tohi suhelda otse andmekihiga. Samal viisil ei tohi ka andmekiht

suhelda otse kasutajaliidesega, vaid kasutama selleks loogikakihti. Ülejärgmiste kihtide olemasolu peab olema teadmata. Seega on igal kihil enda kindel vastutusala ja kihtidel omavahel kindel hierarhia.

Kihilise arhitektuuri eelised:

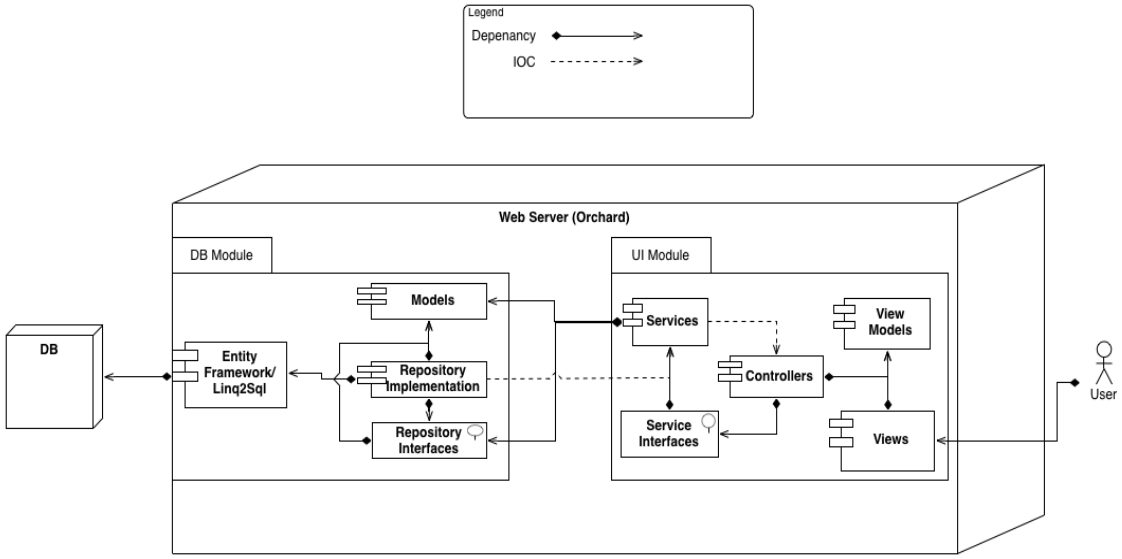
- Võimaldab eraldada üldkasutatavat funktsionaalsust
- Saab määrata piiranguid, milliseid kihte milline kiht kasutada saab
- Funktsionaalsuse asendamine on lihtne (näiteks erinevad andmebaasid)
- Testimise ajaks saab terveid kihte asendada võltsitud (*mocking*) kihtidega [11]

1.4 Model View Controller

Annan lühidalt ülevaate MVC komponentidest.

- **Model** - Rakenduse toimeloogika ehk äriloogika (*business logic*) ja andmeobjektid. Selles kihis on rakenduse komponendid, mis realiseerivad rakenduse funktsioone ehk niinimetatult „tegeliku töö tegijaid“.
- **View** - Rakenduse esitlusloogika, mis saab rakenduse seisundi (*current state*) ehk andmed äriloogika kihi komponentidelt (enamasti kontrolleri vahendusel) ja formeerib kasutajaliidese vastavalt kasutatavale suhtlusprotokollile ja kliendirakendusele (veebirakenduse puhul HTTP/HTML). Esitlusloogika komponendid kutsutakse kontrolleri poolt välja pärast seda, kui äriloogika kihi töö on tehtud (seega töötulemused on teada ja koos nendega ka võimalus otsustada, millist vaadet kasutajale näidata).
- **Controller** – Kontroller seob MVC rakenduses äriloogika ja esitluskihi. Kontroller vastutab pöördumiste kättesaamise eest, tema vastutada on pöördumiste „tõlkimine“ rakenduse sündmusteks, sündmustele vastava töötleja (*event handler*) leidmine ja selle töölerakendamine (äriloogika kihist) ning kasutajale vastuse saatmise organiseerimine vastavate esitluskihi komponentide väljakutsumise teel. [5]

Töös kasutatud Model-View-Controller arhitektuuri selgitab Joonis 2 [12].



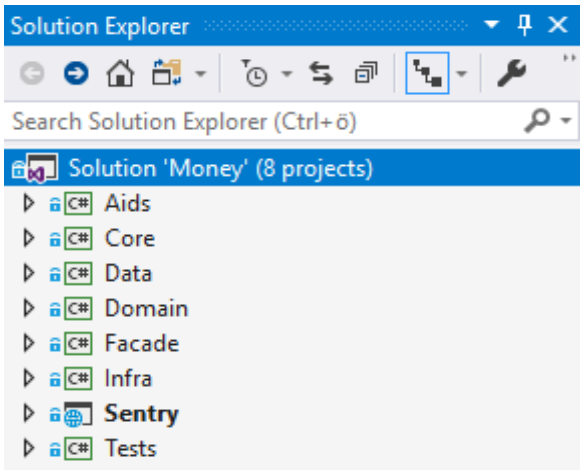
Joonis 2 MVC arhitektuur

.NET Entity Framework [13] raamistikku kasutatakse andmebaasiga suhtlemiseks. Entity Framework on oma olemuselt O/RM (Object-Relational Mapper) [14].
ORM all mõistetakse domeeniobjektide sünkroniseerimist relatsioonilise andmebaasiga, kus relatsiooniline mudel teisendatakse objektorienteeritud mudeliks ja vastupidi. [5]
Selleks, et teha enda valdkonnamudel sõltumatuks Entity Framework raamistikust, on antud projektis realiseeritud *repository* mustri [2] alusel andmehoidla (*repository*).

1.5 Sentry raamistik

Sentry (Sample Entry[1]) on baasklasside ja üldiste funktsioonide raamistik, mis on kasutuses aines „Infosüsteemide Arendamine II“. See raamistik ei ole antud töö autori tehtud, kuid kuna seda raamistikku kasutatakse nii loengutes, kui ka harjutustundides ja kuna seda raamistikku saavad kasutada ka üliõpilased oma isiklikes projektides, siis on ka antud raamistik kasutusel antud lõpuprojektis. Sentry raamistiku struktuur on illustreeritud järgneval joonisel (Joonis 3).

[1] Raamistik, mille töö juhendaja on kirjutanud Leedsi Ülikoolis kasutatava proteoomiksite labori tarkvara uue versiooni jaoks ja mida kasutatakse ka aines „Infosüsteemide arendamine II“



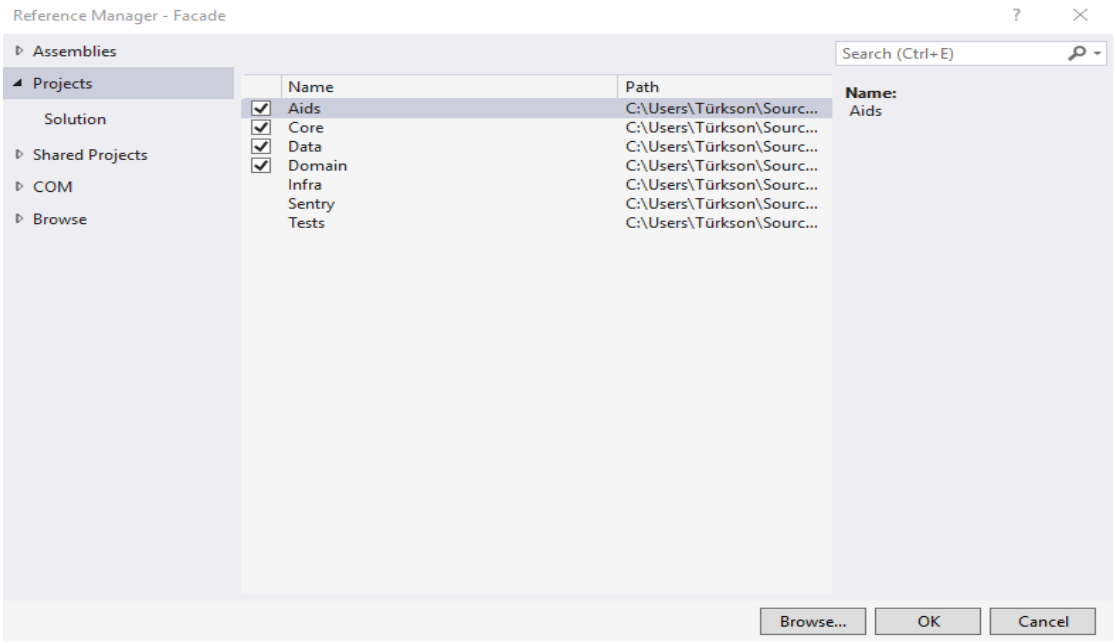
Joonis 3 Mudeli realisatsiooni projektid

Kokku on raamistikus kaheksa projekti. Selgitan lühidalt, mis on mingi projekti eesmärk:

- Aids - Projekt, mis sisaldab kõikvõimalikke abiklasse ja abimeetodeid. Peamiselt on tegu *facade* klassidega selleks, et vabaneda sõltuvusest kolmandate osapoolte koodist. See tähendab seda, et kolmandate osapoolte koodi kasutatakse ainult läbi Aids abiklasside ja abimeetodite.
- Core - Projekt, mis sisaldab abiklasse, mis ei ole kolmanda osapoole poolt kirjutatud, ent on rakenduse toimimiseks vajalikud ja mis on kasutusel suuremas osas rakenduse projektides.
- Data - Projekt, mis tegeleb rakenduses käideldavate andmetega, luues nii objekte kui ka andemebaasikirjeid. Siin asuvad andmekihi klassid, mida kasutatakse andmebaasiga suhtlemisel (O/RM).
- Domain - Projekt, milles sisalduvad klassid tegelevad rakenduse loogikaga. Siin on realiseeritud äriloogika klassid.
- Facade - Projekt, milles sisalduvad esitlusloogika klassid.
- Infra - Projekt, mis ainsana tegeleb otseselt konkreetse andmebaasiga. See on ainuke koht projektis, mis konkreetsest andmebaasist sõltub. Kui tahetakse andmebaasi muuta, tuleb ainult see projekt ümber kirjutada. See projekt varustab projekti Domain andmetega andmebaasist ja teeb kõik vajalikud andmebaasiga seotud CRUD (Create, Read, Update ja Delete) operatsioonid.
- Sentry - *ASP.NET Core 2.0 Web Application* tüüpi projekt, kus on realiseeritud MVC kontrollerid (kasutab projekti Infra) ja vaated (kasutab projekti Facade).

- Tests - *Unit Test Project* kus asuvad nii ühiktestid, integratsioonitestid (projekti Infra testid), kui ka vastuvõtutestid (projekti Sentry testid).

Selline rakenduse (*solution*) jagamine erinevateks projektideks (*project*) vastavalt vastutusaladele (*separation of concerns*) [7] suurendab rakenduse hallatavust (*maintainability*). Juurdepääsud on organiseeritud läbi viidete (*reference*), kus iga projekt saab ligipääsu ainult temast „ülalpool“ olevatele projektidele. Seda illustreerib Joonis 4.

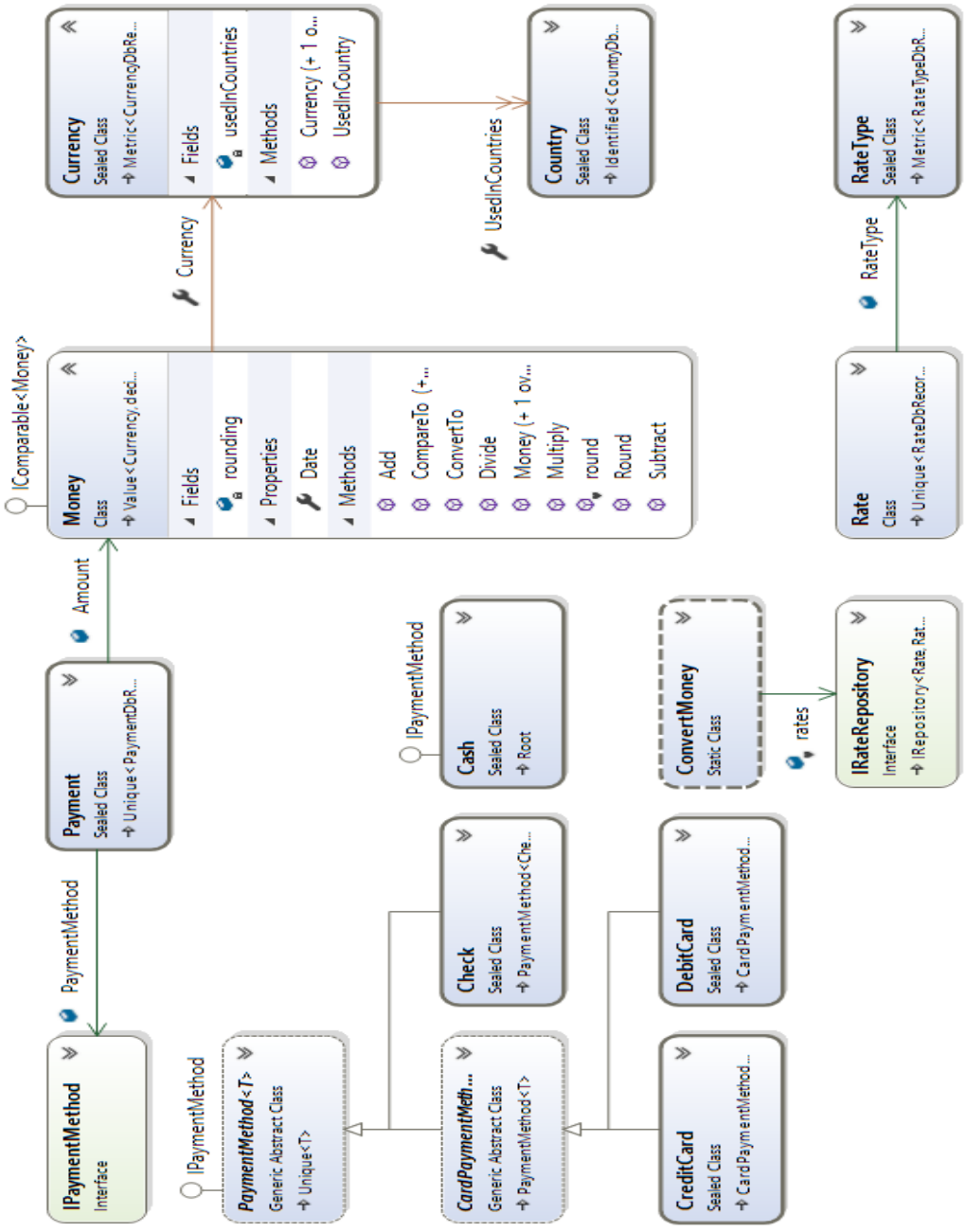


Joonis 4 Vastutusalad

Seejuures ei pääse näiteks Aids projektis sisalduv kood ligi teistele projektidele. Infra projektis olev kood see-eest pääseb ligi kõigile temast ülevalpool asuvatele projektidele. Sentry projekt pääseb ligi ainult Facade ja Infra projektidele, sest teiste projektidega otsest suhtlust tarvis ei ole. Tests klass pääseb ligi kõikidele rakenduses olevatele projektidele, kuna ilma ligipääsuta on koodi toimimist võimatu testida.

2 Tehtud töö

Antud töö raames tegeletakse raha arhetüübi kui valdkonnamudeli realiseerimisega. Realiseeritud valdkonnamudel on nähtav Joonisel 5



Joonis 5 Money üldine mudel

Peamised klassid, mis kuuluvad Money mudelisse on Money ja Currency. Lisaks veel maksete sooritamiseks vajalikud klassid Payment ja PaymentMethod (ja temast tuletatud klassid Cash, Check, CreditCard, DebitCard) ning ka valuutade teisendamiseks vajalikud klassid Rate, RateType ja ConvertMoney. Seejuures Currency ja Country ei ole autori tehtud. Kõik teised raha valdkonnamudeli klassid (Money, Payment, PaymentMethod, Cash, Check, CreditCard, DebitCard, Rate, RateType, ConvertMoney ja Calculator) aga on autori loodud ja testitud. Lisaks on autori loodud ja testitud raha valdkonnamudelile vajalikud ja vastavad andmekihi (*Data Layer* [15]) klassid (PaymentDbRecord, PaymentMethodDbRecord, CheckDbRecord, CreditCardDbRecord, DebitCardDbRecord, RateDbRecord, RateTypeDbRecord) ning esitluskihi klassid (MoneyView, PaymentView, PaymentMethodView, CashView, CheckView, CreditCardView, DebitCardView, RateView ja RateTypeView). Samuti on autori loodud projektis Infra asuvad vastavad andmehoidla klassid (PaymentsRepository, PaymentMethodsRepository, RateRepository ja RateTypeRepository) ning projektis Sentry asuvad vastavad kontrollerid (PaymentsController, PaymentMethodsController, RateController ja RateTypeController) ning nende vaated. Lisaks on loodud ka kontroller CalculatorController ja vastavad vaated selleks, et manuaalselt testida rahaga tehtavaid operatsioone. Autori panuse võtab kokku Tabel 1.

Tabel 1 Projektis loodud klassid

Klass	Projekt	Milleks vaja	Koodi-ridade arv klassis	Koodi-ridade arv test-klassis
Money	Domain	Defineerib raha olemi	73	189
Payment	Domain	Defineerib makse olemi	15	34
PaymentMethod	Domain	Defineerib maksemeetodi olemi	9	24
Cash	Domain	Defineerib sularaha olemi	7	14
Check	Domain	Defineerib tšeki olemi	7	16
CreditCard	Domain	Defineerib krediitkaardi olemi	12	19
DebitCard	Domain	Defineerib deebetkaardi olemi	7	14
Rate	Domain	Defineerib kursi olemi	15	22
RateType	Domain	Defineerib kursi tüübi olemi	9	14

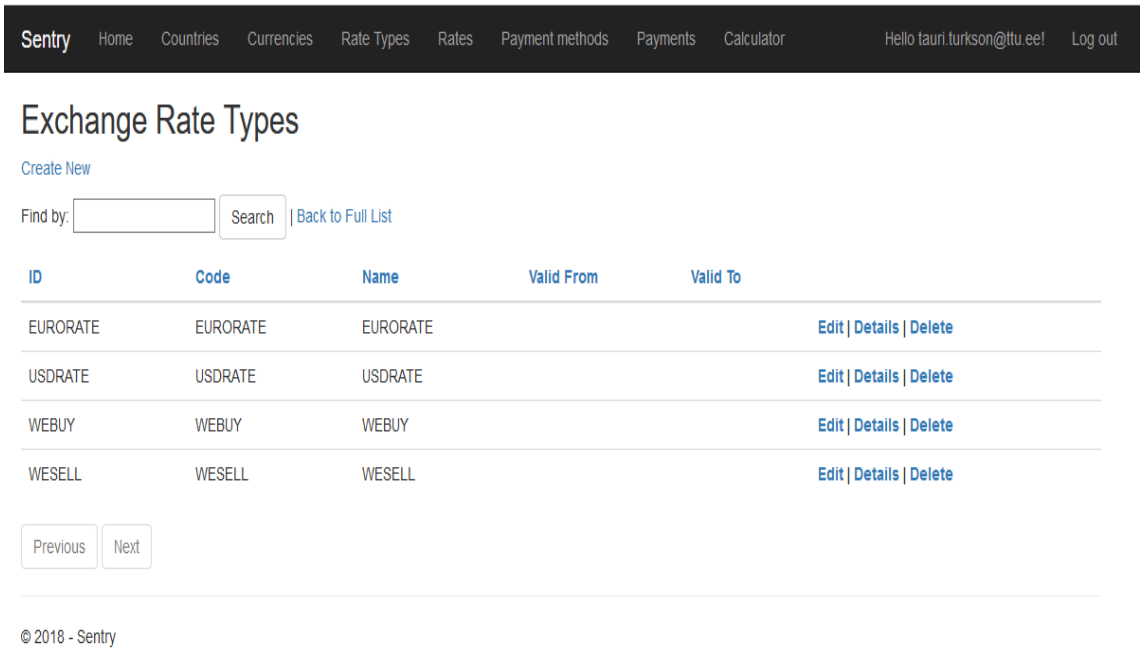
ConvertMoney	Domain	Loob etteantud (raha, valuuta) väärtuste alusel uue raha objekti, millel väärtusteks valuuta, kogus ja kuupäev	44	24
PaymentDbRecord	Data	Defineerib makse andmebaasikirje olemi	14	55
PaymentMethodDbRecord	Data	Defineerib arve tüübi andmebaasikirje olemi	15	53
CheckDbRecord	Data	Defineerib tšeki andmebaasikirje olemi	11	19
CreditCardDbRecord	Data	Defineerib krediitkaardi andmebaasikirje olemi	5	19
DebitCardDbRecord	Data	Defineerib deebetkaardi andmebaasikirje olemi	3	14
RateDbRecord	Data	Defineerib kursi andmebaasikirje olemi	14	36
RateTypeDbRecord	Data	Defineerib kursi tüübi andmebaasikirje olemi	5	16
MoneyView	Facade	Defineerib raha kasutajaliidese vaate	9	19
PaymentView	Facade	Defineerib makse kasutajaliidese vaate	18	26
PaymentMethodView	Facade	Defineerib makse meetodi kasutajaliidese vaate	13	19
CashView	Facade	Defineerib sularaha kasutajaliidese vaate	7	11
CheckView	Facade	Defineerib tšeki kasutajaliidese vaate	17	39
CreditCardView	Facade	Defineerib krediitkaardi kasutajaliidese vaate	10	15
DebitCardView	Facade	Defineerib deeebetkaardi kasutajaliidese vaate	9	12

RateView	Facade	Defineerib kursi kasutajaliidese vaate	39	33
RateTypeView	Facade	Defineerib kursi tüübi kasutajaliidese vaate	29	21
CalculatorView	Facade	Defineerib kalkulaatori kasutajaliidese vaate	46	-
PaymentsRepository	Infra	Defineerib makse andmehoidla olemi	35	-
PaymentMethodsRepository	Infra	Defineerib makse meetodi andmehoidla olemi	72	-
RateRepository	Infra	Defineerib kursi andmehoidla olemi	41	-
RateTypeRepository	Infra	Defineerib kursi tüübi andmehoidla olemi	19	-
PaymentsController	Sentry	Defineerib maksete kontrolleri	99	-
PaymentMethodsController	Sentry	Defineerib maksemeetodite kontrolleri	224	-
RateController	Sentry	Defineerib kursside kontrolleri	116	-
RateTypeController	Sentry	Defineerib kursi tüüpide kontrolleri	114	-
CalculatorController	Sentry	Defineerib kalkulaatori kontrolleri	111	-

2.1 Kasutajaliides

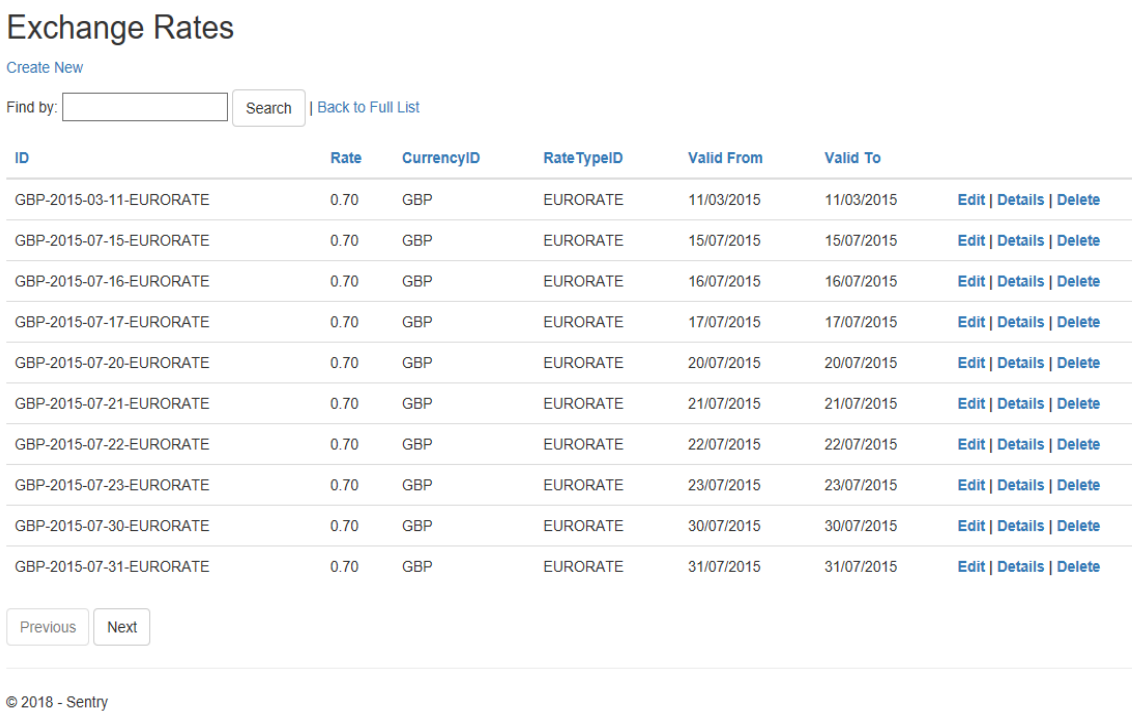
Töös on kasutatud ühtset Sentry raamistiku kasutajaliidese disaini. Järgnevalt ainult need vaated, mis on arendatud töö autori poolt.

Joonis 6 illustreerib Rate Types menüüst avanevat kursi tüüpide loetelu.



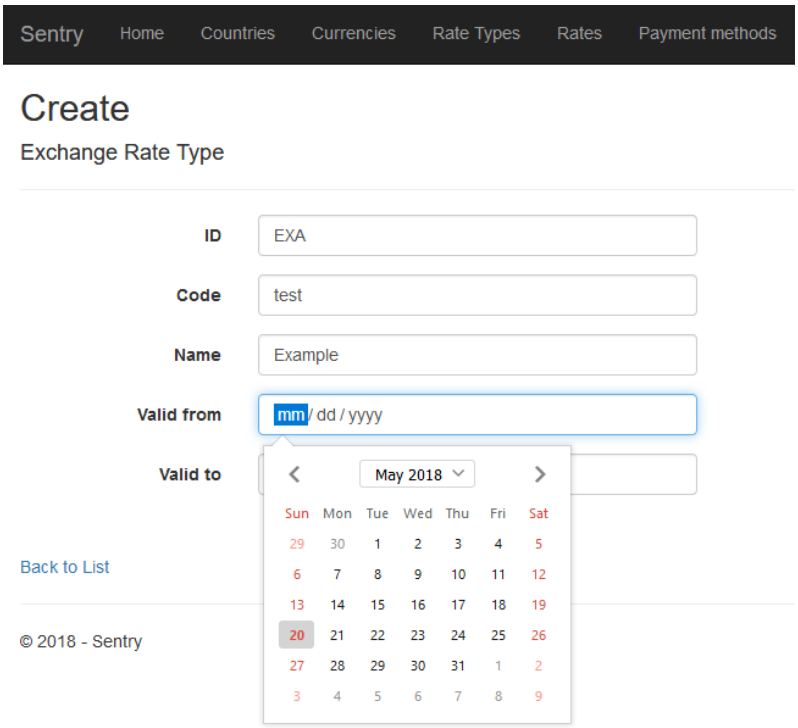
Joonis 6 Kursi tüübid

Joonis 7 illustreerib Rates menüüst avanevat kursside loetelu.



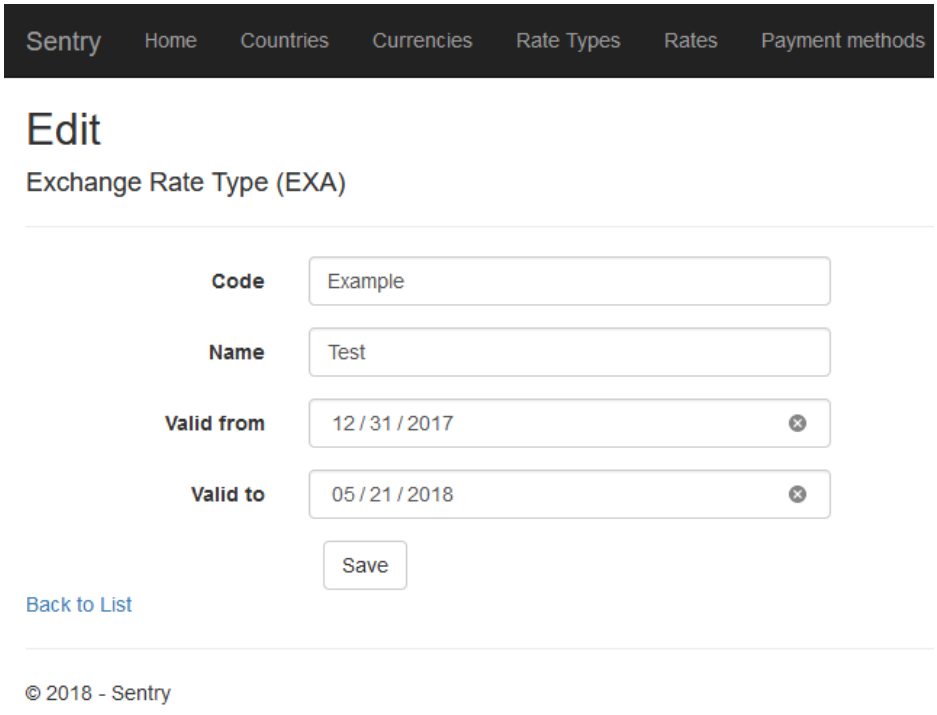
Joonis 7 Kursid

Joonis 8 illustreerib andmete sisestamist.



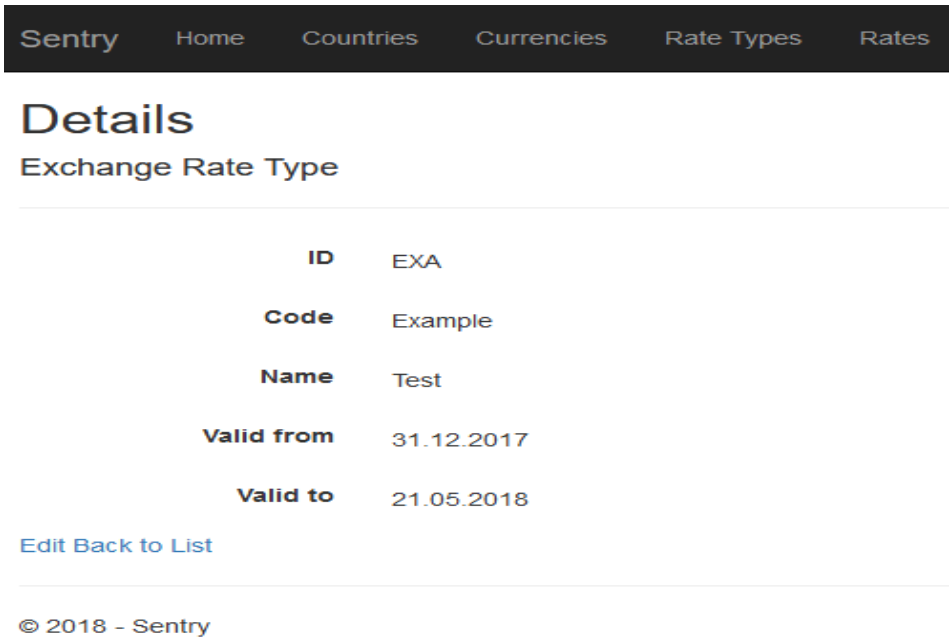
Joonis 8 Andmete sisestamine

Joonis 9 illustreerib andmete muutmist.



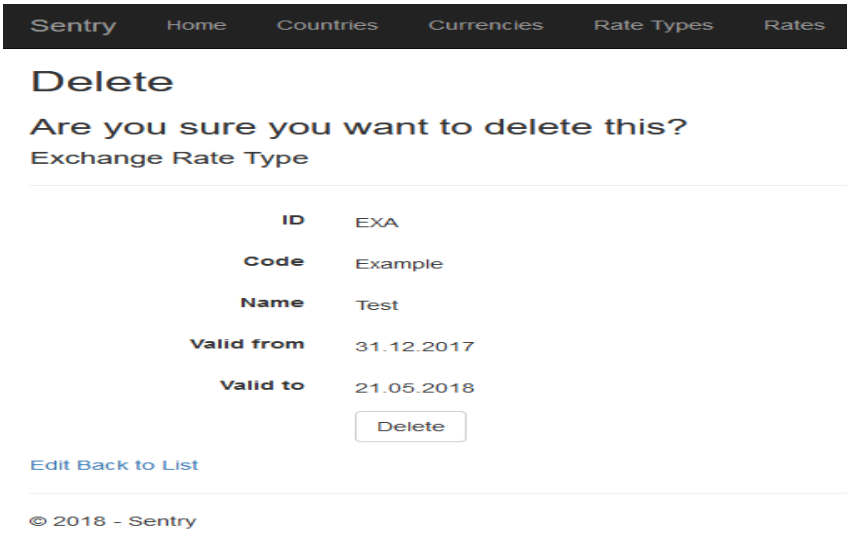
Joonis 9 Andmete muutmine

Joonis 10 illustreerib detailandmete vaatamist.



Joonis 10 Andmete vaatamine

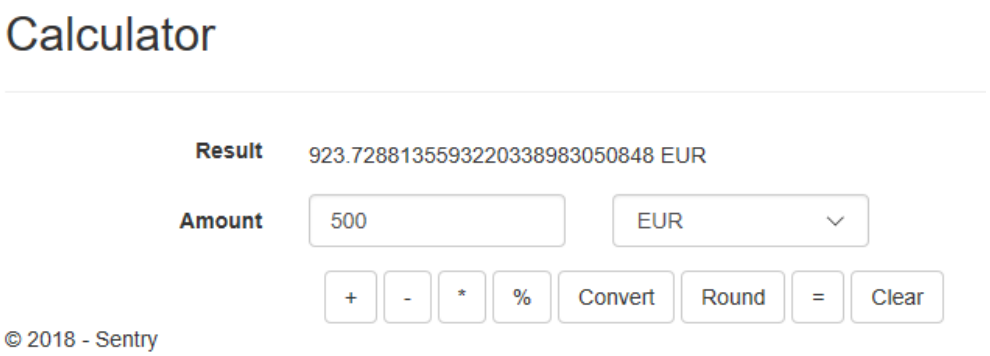
Joonis 11 illustreerib andmete kustutamist.



Joonis 11 Andmete kustutamine

Analoogiliselt lihtsad on ka kõikide teiste andmete sisestamiseks, muutmiseks, vaatamiseks ja kustutamiseks mõeldud vaated.

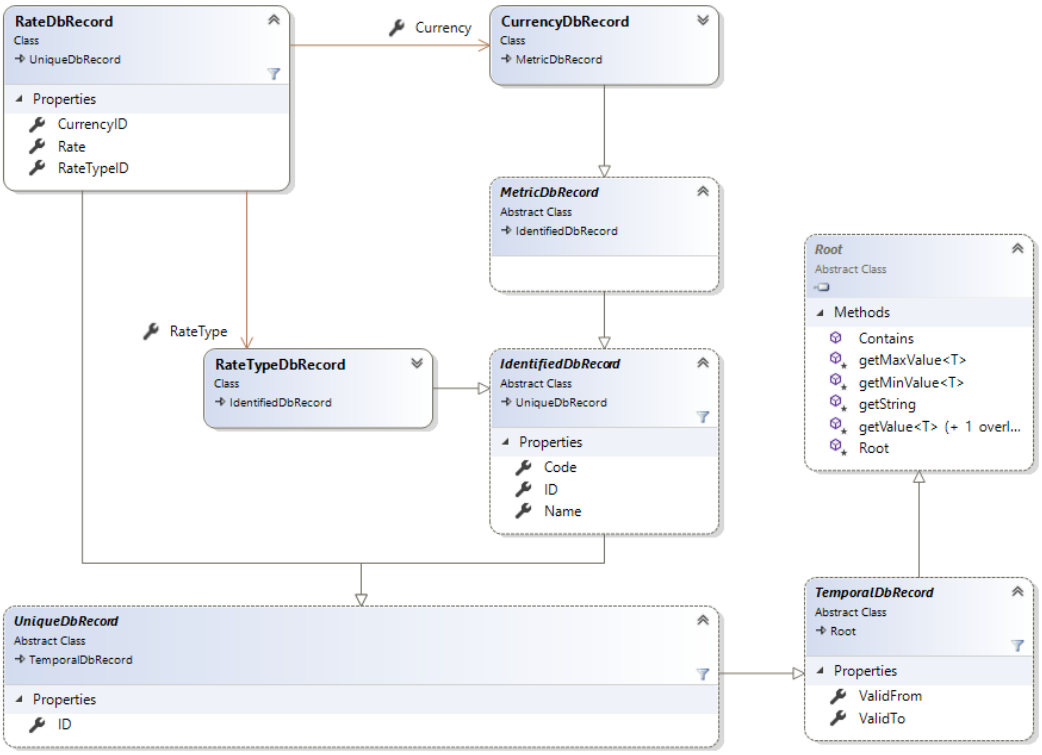
Rahaga teostatavate tehete illustreerimiseks ja manuaalseks testimiseks loodud vaadet illustreerib Joonis 12. Antud juhul on lisatud väärtusele 500USD väärtus 500EUR.



Joonis 12 Kalkulaator

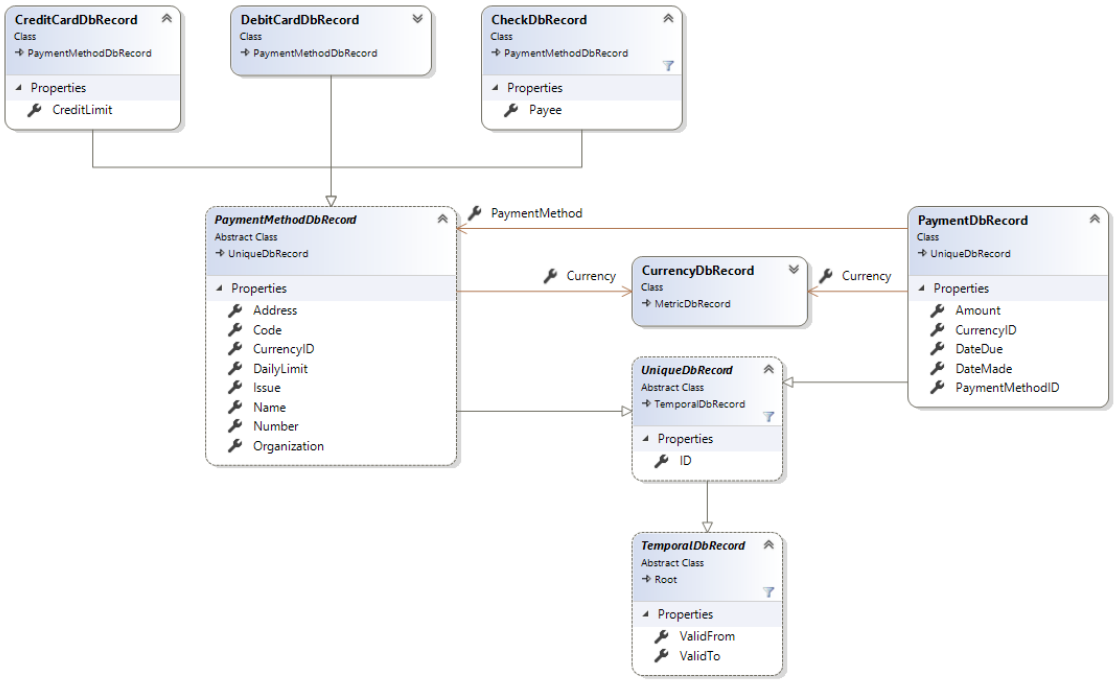
2.2 Andmetele juurdepääsu kiht

Projektis Data on realiseeritud andmetele juurdepääsu kiht (Data Access Layer). Need on klassid, mis esindavad konkreetse andmebaasi tabeli ühte konkreetset rida. Autoril tuli siin realiseerida ja testida Rate ja RateType klassid (Joonis 13) ning Payment ja PaymentMethod klassid (Joonis 14). Baasklassid nagu MetricDbRecord ja kõik MetricDbRecord baasklassid kuuluvad Sentry raamistiku hulka ja ei ole autori tehtud. Ka ei ole autori tehtud CurrencyDbRecord ning ka CountryDbRecord.



Joonis 13 Kurss – andmetele juurdepääsu kiht

Valuutakursi mudeli korral on tähtsaimaks objektiks RateDbRecord. See on seotud RateTypeDbRecord ja CurrencyDbRecord klassidega, ilma milleta ei oleks kursi „kättesaamine“ võimalik.



Joonis 14 Maksemeetod – andmetele juurdepääsu kiht

Antud realisatsioonimudelil on põhiobjektiks PaymentMethodDbRecord. Sellest on päritud kolm klassi – CreditCardDbRecord, DebitCardDbRecord ja CheckDbRecord.

Kui võrrelda Arlow ja Neustdt [6] raha mudeliga, siis on andmetele juurdepääsu kihi mudel sellest oluliselt erinev. Erinevuse peamiseks põhjuseks on muuta andmetabelite hulk võimalikult väikseks ja ka andmetabelites olevate veergude hulk võimalikult väikseks.

Tüüpiliselt on selles projektis olevad klassid väga lihtsad ja sisaldavad ainult andmeid nagu on Joonis 15 näidatud.

```
using Open.Data.Common;
using System;

namespace Open.Data.Money {
    22 references | Gunnar Piho, 2 days ago | 2 authors, 7 changes
    public class PaymentDbRecord : UniqueDbRecord {
        9 references | Gunnar Piho, 32 days ago | 1 author, 4 changes | 0 exceptions
        public decimal Amount { get; set; }

        7 references | Gunnar Piho, 39 days ago | 1 author, 1 change | 0 exceptions
        public DateTime DateDue { get; set; }

        10 references | Gunnar Piho, 39 days ago | 1 author, 1 change | 0 exceptions
        public DateTime DateMade { get; set; }

        8 references | Gunnar Piho, 2 days ago | 1 author, 4 changes | 0 exceptions
        public string CurrencyID { get; set; }

        5 references | Gunnar Piho, 32 days ago | 1 author, 3 changes | 0 exceptions
        public virtual CurrencyDbRecord Currency { get; set; }

        8 references | Gunnar Piho, 2 days ago | 1 author, 2 changes | 0 exceptions
        public string PaymentMethodID { get; set; }

        10 references | Gunnar Piho, 39 days ago | 1 author, 1 change | 0 exceptions
        public virtual PaymentMethodDbRecord PaymentMethod { get; set; }
    }
}
```

Joonis 15 Koodinäide PaymentDbRecord

Sellised ainult andmete hoidmiseks mõeldud klassid vastavad puhta koodi andmestruktuuride eraldamise ja ühese vastutuse (*Single Responsibility Only)* nõuetele [7]. Ka on selline lähenemine hea andmetabelite genereerimiseks koodi põhjal, mida kogu .NET raamistik ja eriti Entity Framework toetab. Konkreetselt selle klassi ja vastava andmetabeli seos realiseeritakse ASP.NET Core 2.0 puhul DbContext klassis (antud rakenduse projekti Infra klassis SentryDbContext) ja on illustreeritud Joonis 16-l.

```
private static void createPaymentTable(ModelBuilder b) {
    const string table = "Payment";
    createForeignKey<PaymentDbRecord, CurrencyDbRecord>(b, table, x => x.CurrencyID, x => x.Currency);
    createForeignKey<PaymentDbRecord, PaymentMethodDbRecord>(b, table, x => x.PaymentMethodID, x => x.PaymentMethod);
}
```

Joonis 16 Koodinäide SentryDbContext

Eelneva koodi põhjal genereeritakse andmebaasi andmetabel „Payment“, mille struktuur on illustreeritud Joonis 17-ga.

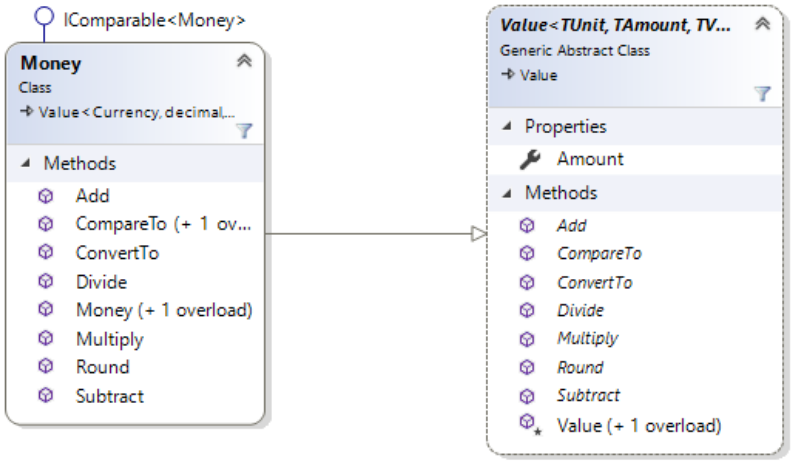
```
CREATE TABLE [dbo].[Payment] (
    [ID]              NVARCHAR (450)  NOT NULL,
    [Amount]          DECIMAL (18, 2) NOT NULL,
    [CurrencyID]      NVARCHAR (450)  NULL,
    [DateDue]         DATETIME2 (7)   NOT NULL,
    [DateMade]        DATETIME2 (7)   NOT NULL,
    [PaymentMethodID] NVARCHAR (450)  NULL,
    [ValidFrom]       DATETIME2 (7)   NOT NULL,
    [ValidTo]         DATETIME2 (7)   NOT NULL,
    CONSTRAINT [PK_Payment] PRIMARY KEY CLUSTERED ([ID] ASC),
    CONSTRAINT [FK_Payment_Currency_CurrencyID] FOREIGN KEY ([CurrencyID]) REFERENCES [dbo].[Currency] ([ID]),
    CONSTRAINT [FK_Payment_PaymentMethod_PaymentMethodID] FOREIGN KEY ([PaymentMethodID]) REFERENCES [dbo].[PaymentMethod] ([ID])
);


GO
CREATE NONCLUSTERED INDEX [IX_Payment_CurrencyID]
    ON [dbo].[Payment]([CurrencyID] ASC);


GO
CREATE NONCLUSTERED INDEX [IX_Payment_PaymentMethodID]
    ON [dbo].[Payment]([PaymentMethodID] ASC);
```

Joonis 17 Koodinäide Payment andmebaasitabel

Seda andmebaasi tabelit saab muuta samuti koodis. Näiteks saab vähendada unikaalse identifikaatori pikkust. Selline tabelite peenhäälestus jäi aga praegu töö skoobist välja.

2.3 Valdkonnaloogika kiht

Projektis Domain on realiseeritud valdkonnaloogika ehk äriloogika kiht (Domain Logic Layer). Loogikakihi klassidest on autori tehtud ja testitud sellised klassid nagu Money (Joonis 18), valuutateisenduseks vajalikud klassid (Joonis 19) ning maksete haldamiseks vajalikud klassid (Joonis 20).



Joonis 18 Money operatsioonid – valdkonnaloogika kiht

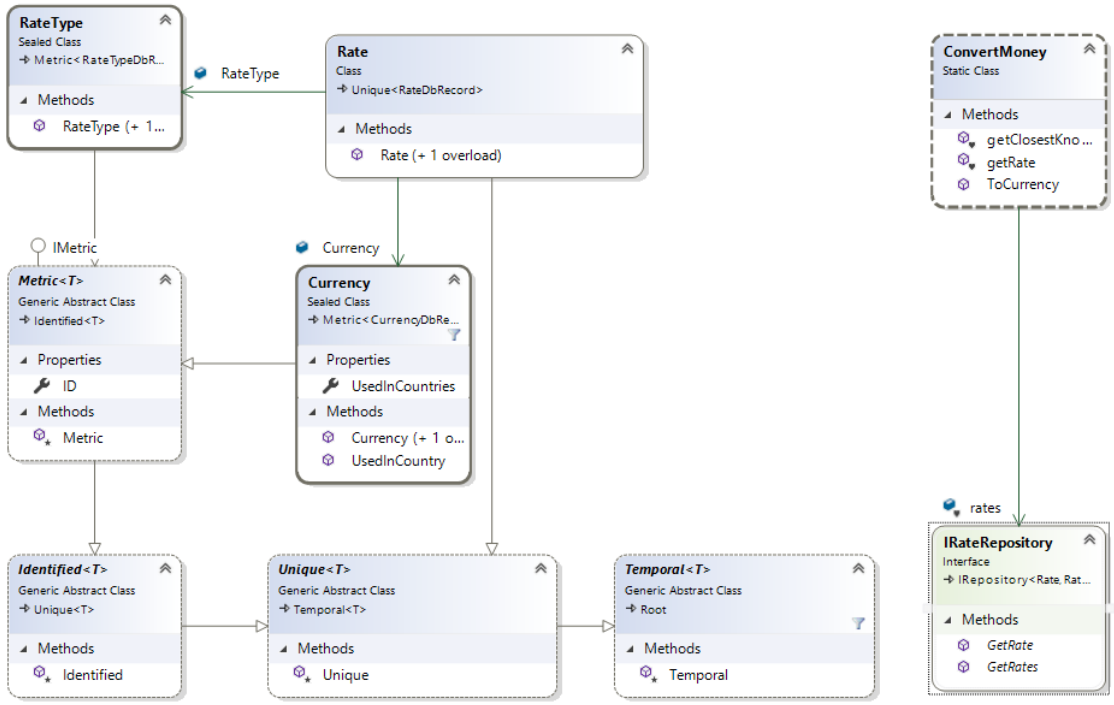
Realiseeritud mudel kirjeldab infosüsteemis võimalikke matemaatilisi operatsioone, mida on võimalik rahaga teostada.

Järgnevalt selgitan iga matemaatilise operatsiooni kui parameetri eesmärki süsteemis:

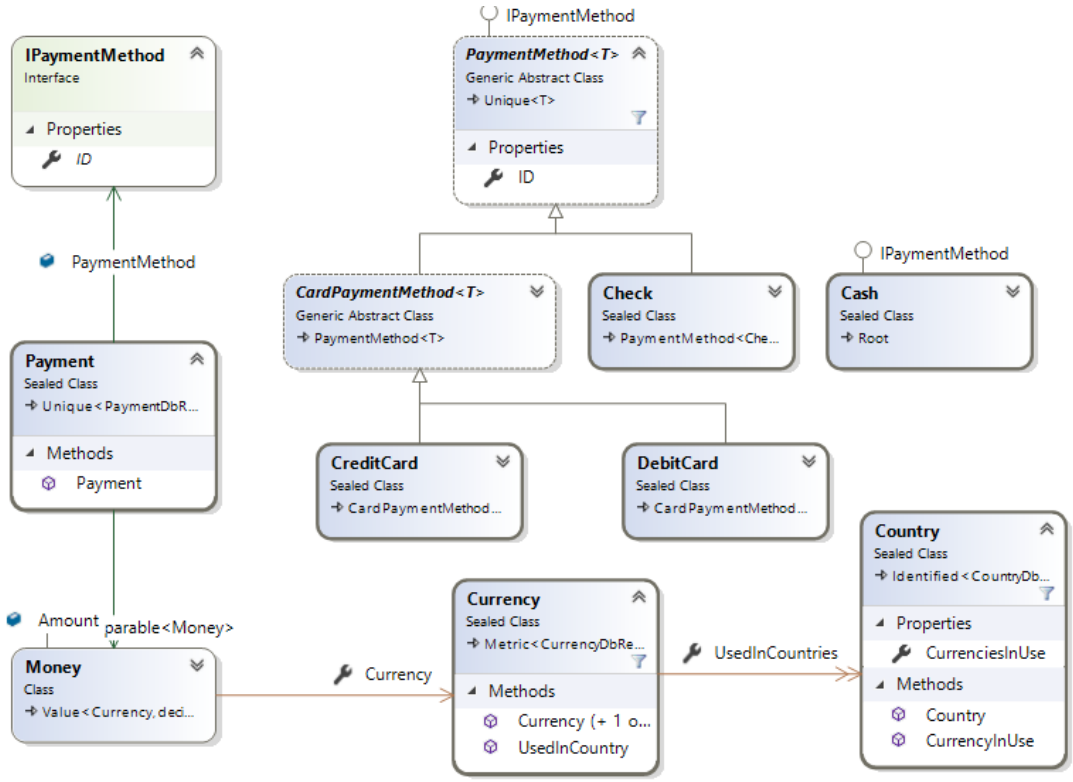
- Add – liitmisoperaator, liidab olemasolevale väärtusele juurde etteantud väärtuse. Liita saab kahte erineva valuutaga raha. Valuuta teisendu tehakse automaatselt. Resultaat valuutaks valitakse selle raha valuuta, mis antakse argumendina.
- CompareTo – võrdlusoperaator, tagastab kas etteantud väärtus on meetodi algväärtusest väiksem, võrdne või suurem.
- CovertTo – teisendusoperaator, tagastab uue Money objekti, mis on teisendatud teise valuutasse vastavalt valuutakursile.
- Divide – jagamisoperaator, jagab olemasoleva väärtuse etteantud reaalarvulise väärtusega. Nulliga jagamise korral tagastatakse decimal.MaxValue.
- Multiply – korrutamisoperaator, korrutab olemasoleva väärtuse etteantud reaalarvuga.
- Round – ümardamisoperaator. Kokku on realiseeritud üheksa erinevat ümardamise reeglit.
- Subtract – lahutamisoperaator, lahutab olemasolevalt väärtuselt etteantud väärtuse. Lahutada saab kahte erineva valuutaga raha. Valuuta teisendu tehakse automaatselt. Resultaat valuutaks valitakse selle raha valuuta, mis antakse argumendina.

Realiseeritud raha klassi kood on näidatud lisas Lisa 1 – Money.cs kood.
Valuutateisendi puhul on põhiobjektideks Currency ja Rate – nende abil teisendatakse klassis ConvertMoney konkreetne rahaline väärtus ühest valuutast teise. Seejuures käib teisendus RateType põhjal. Praegu on rakenduses arvestatud nelja erineva kursitüübiga: EURORATE (EUR kurss on 1 ja teiste valuutade kursid on antud koefitsiendiga euro suhtes); USDRATE (USD kurss on 1 ja teiste valuutade kursid on antud koefitsiendiga USD suhtes) ja kaks lokaalset kursside tüüpi (WESELL ja WEBUY). Viimastes on siis kohaliku kursi koefitsient 1 ning teiste valuutade kursid siis vastavalt suhtega kohalikku valuutasse.

Praegu on realiseeritud ainult EURORATE kursside kasutamine. Valuutakursside allalaadimise kood on näidatud lisas Lisa 2 – EuroRatesDbTableInitializer.cs ja klassi ConvertMoney kood on näidatud lisas Lisa 3 – ConvertMoney.cs



Joonis 19 Valuutateisendi – valdkonnaloogika kiht

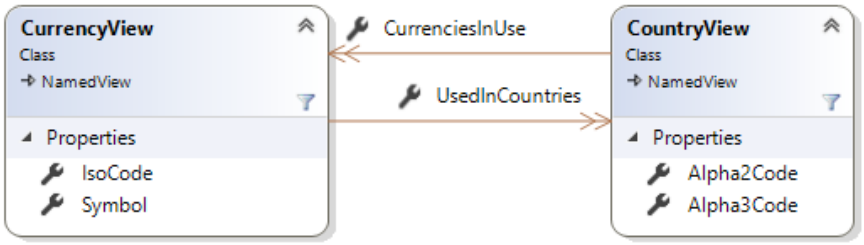


Joonis 20 Makse – valdkonnaloogika kiht

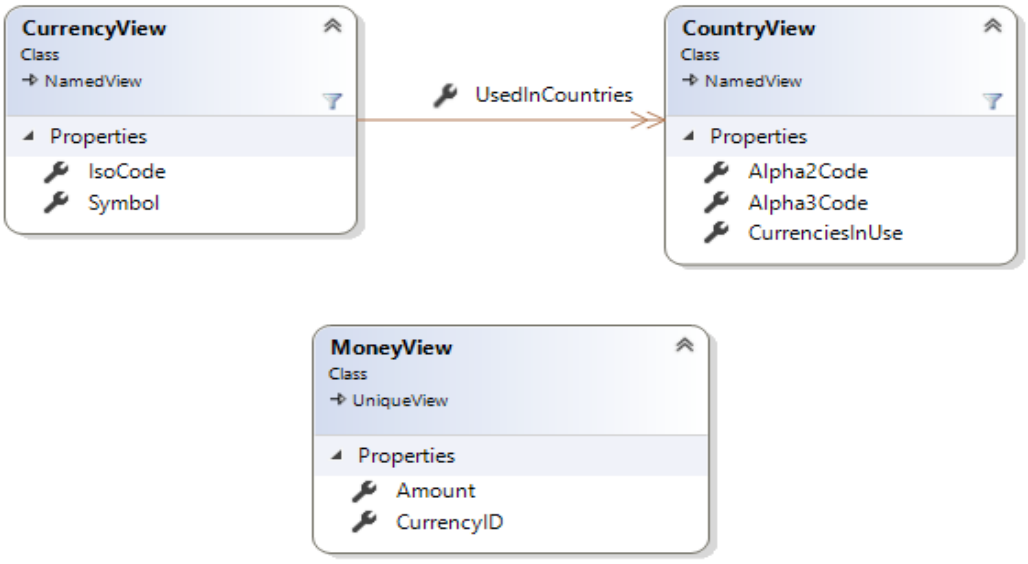
Realiseeritud makse mudeli puhul on põhiobjektiks Payment, mis sisaldab konkreetset rahalist väärtust (Money klass), mis on sooritatud mingi konkreetset maksemeetodit (PaymentMethod) kasutades. Maksemeetodiks on kas Cash, Check, CreditCard või DebitCard.

2.4 Esitluskiht

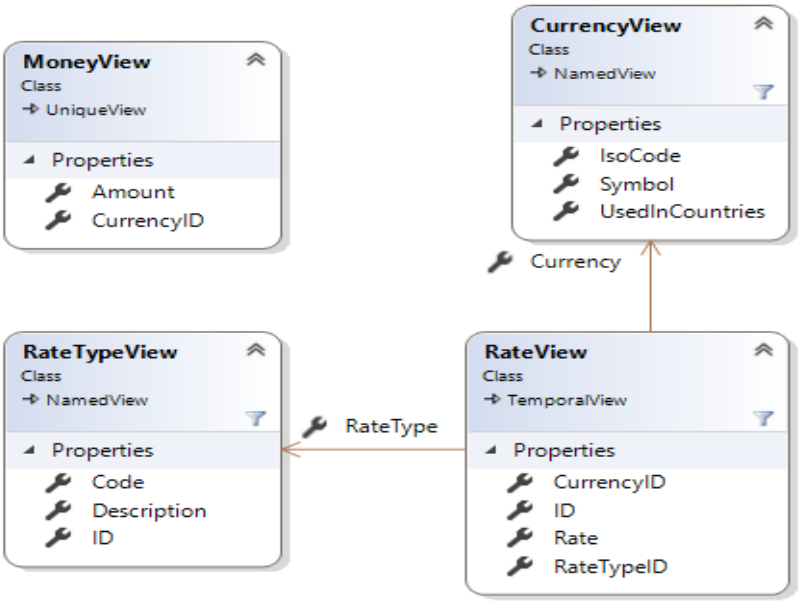
Projektis Facade on realiseeritud esitluskiht (Presentation Layer). Esitluskihis realiseeritud koodi saab kokku võtta mudelitega CurrencyView (Joonis 21), MoneyView (Joonis 22), RateView (Joonis 23) ja PaymentMethodView (Joonis 24).



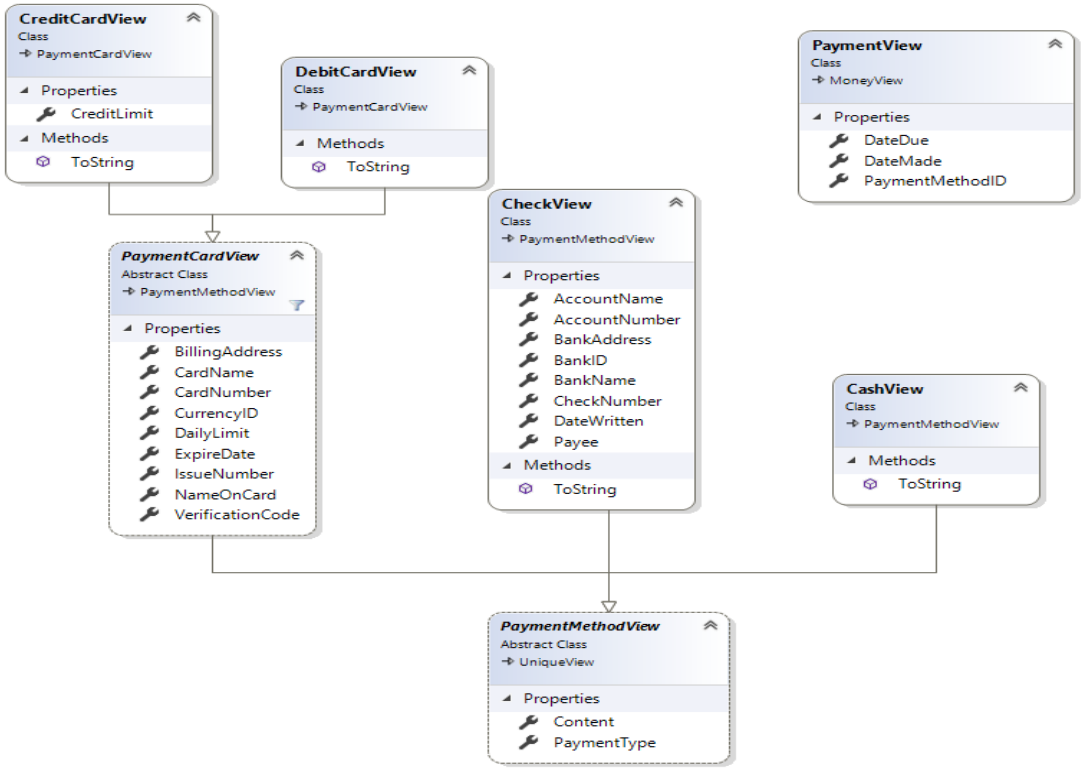
Joonis 21 Valuuta - esitluskiht



Joonis 22 Raha – esitluskiht



Joonis 23 Kurss - esitluskiht



Joonis 24 Maksemeetod - esitluskiht

Kui andmetele juurdepääsukihis on näiteks maksemeetodid disainitud selliselt, et andmeid saaks võimalikult minimaalselt andmebaasi tabelitesse salvestada (maksete mudel Joonis 14 kus peamiselt kogu informatsioon on klassis PaymentMethodDbRecord), siis esitluskihis on klassid disainitud vastavalt sellele, mis

on konkreetsete andmebaasis hoitavate väljade tähendus kasutajale. Seda illustreerib Joonis 25.

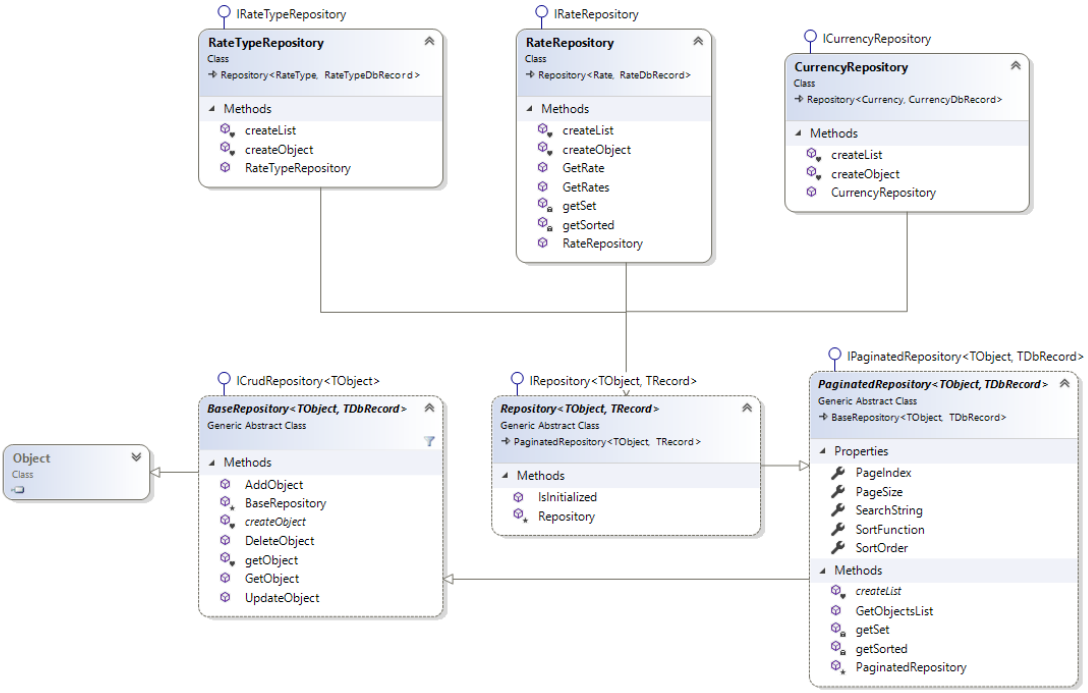
```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace Open.Facade.Money {
    17 references | TauriTurkson, 2 days ago | 2 authors, 5 changes
    public class CheckView : PaymentMethodView {

        5 references | Gunnar Piho, 2 days ago | 1 author, 3 changes | 0 exceptions
        public string Payee { get; set; }

        5 references | Gunnar Piho, 2 days ago | 1 author, 3 changes | 0 exceptions
        [DisplayName("Account Name")] public string AccountName { get; set; }
        5 references | TauriTurkson, 2 days ago | 2 authors, 4 changes | 0 exceptions
        [DisplayName("Account Number")] public string AccountNumber { get; set; }
        5 references | TauriTurkson, 2 days ago | 2 authors, 4 changes | 0 exceptions
        [DisplayName("Bank Address")] public string BankAddress { get; set; }
        5 references | TauriTurkson, 2 days ago | 2 authors, 4 changes | 0 exceptions
        [DisplayName("Bank Identification Number")] public string BankID { get; set; }
        6 references | TauriTurkson, 2 days ago | 2 authors, 4 changes | 0 exceptions
        [DisplayName("Bank Name")] public string BankName { get; set; }
        6 references | TauriTurkson, 2 days ago | 2 authors, 4 changes | 0 exceptions
        [DisplayName("Check Number")] public string CheckNumber { get; set; }

        [DataType(DataType.Date)] [DisplayName("Date Written")]
        3 references | TauriTurkson, 2 days ago | 2 authors, 4 changes | 0 exceptions
        public DateTime? DateWritten { get => ValidFrom; set => ValidFrom = value; }

        3 references | Gunnar Piho, 2 days ago | 1 author, 2 changes | 0 exceptions
        public override string ToString() { return $"Check ({CheckNumber}, {BankName})"; }
    }
}
```

Joonis 25 Koodinäide CheckView

Nii näiteks läheb andmebaasi tabelist olev väli Aadress klassis CheckView korral välja BankAddress, kuid klasside CreditCardView ja DebitCardView korral välja BillingAddress.

2.5 Infrastruktuuri kiht

Infrastruktuuri kiht (Infrastructure Layer) on selleks, et valdkonnamudel, mis on disainitud infrastruktuurist sõltumatult [16], saaks konkreetse juurdepääsu konkreetselt kasutatavale andmebaasile. Kui kogu valdkonnamudel opereerib ainult andmehoidla liidestega, siis siin realiseeritakse need konkreetsed andmehoidlad. Autori poolt on realiseeritud ja testitud kursi andmehoidlad RateTypeRepository ja RateRepository (Joonis 17) ning analoogiliselt ka Payment ja PaymentMethod andmehoidlad. Andmehoidlate baasklassid nagu Repository, PaginatedRepository ja BaseRepository ei ole autori tehtud.

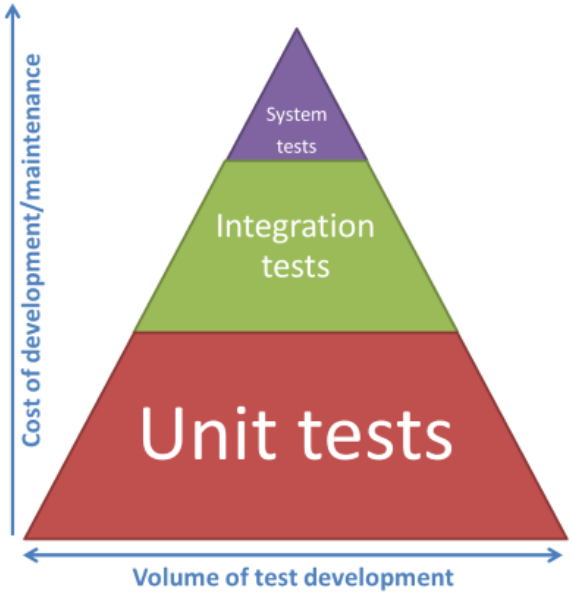


Joonis 26 Kurss – infrastruktuuri kiht

2.6 Testid

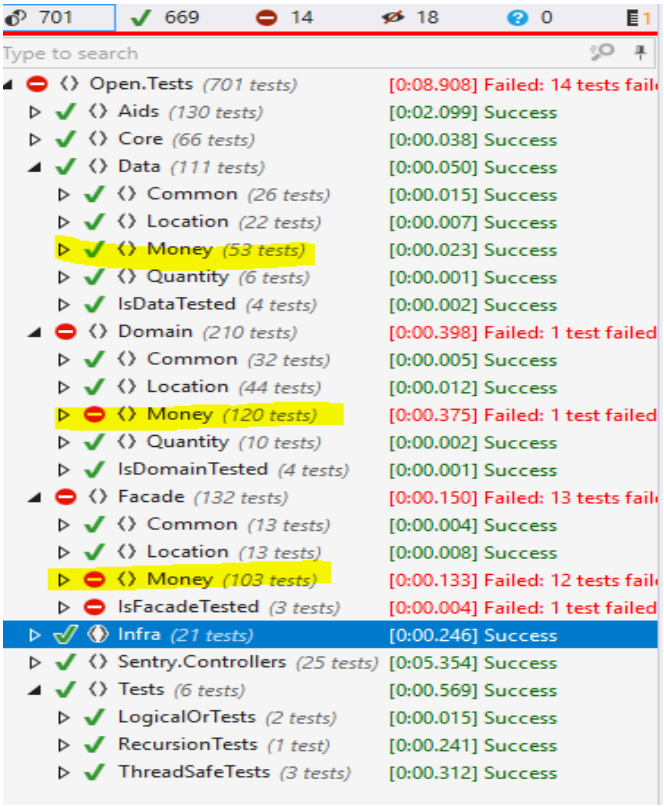
Ühik(elementaar)testid (*unit tests*) moodustavad ja peavad moodustama suurema osa (umbes 70%) kõikidest testidest, seda illustreerib ka Joonis 27 [15]. Teste kirjutades peab jälgima, et need alluks F.I.R.S.T (*fast, isolated, repeatable, self-validating, timely)* tingimusele [16]. Ühiktestid ei tegele sõltuvuste ja infrastruktuuriga, ei tohi olla seotud failidega, võrguga ega andmebaasidega.

Integratsioonitestid testivad koodi erinevate osade koostoimet, moodustades kõikidest testidest umbes 20%. Selle alla kuuluvad ainult sellised testid, mida ei saa ühiktestidega katta. Seljuhul testitakse tavaliselt koodi koos testandmebaasiga või koos testserveritega. Kui integratsiooni- ja ühiktestid on testid arendaja (kas tehakse õigesti) vaates, siis funktsionaalsed testid (vahel ka vastuvõtutestid või süsteemitestid) on kasutaja vaates (kas tehakse õiget asja) kirjutatud testid. Funktsionaaltestides on vahel vaja ka kasutajaliidese kaasamist testimisse. [19]



Joonis 27 Testid

Antud projekt on kaetud kokku 701 automaatse ühik(elementaar) testiga (*unit testing*), millest läbi kukub veel 14 (Joonis 28). Autor on teinud testid, mis seotud Money mustriga – kokku 276 testi,, millest töö kirjutamise ajal 13 pole veel jõutud realiseerida. Kui testid puuduksid, ei oleks kood kontrollitav ja seeläbi ka selle töökindlus usaldatav.



Joonis 28 Ühiktestid

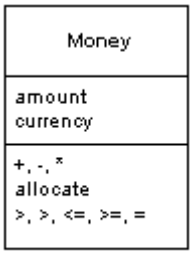
Antud projekti raames on põhitähelepanu olnud ühiktestidel. Integratsiooni ja vastuvõtutestideni veel aja ja ka oskuste puudumise tõttu ei jõutud. Samas omab ASP.NET Core väga võimsad vahendeid nii integratsioonitestide kui ka vastuvõtutestide tegemiseks. Mõningad katsetused on tehtud, kuid täiemahuliste testideni veel ei ole jõutud.

3 Analüüs ja järeldused

Järgnevalt vaatlen ja analüüsin lähemalt eri raha mudeleid. Välja toodud mudelitel on funktsionaalsus vaid osaliselt kujutatud. Realiseeritud Arlow ja Neustadt-i mudelis [6] on aga allolevates mudelites kirjeldatud puudused kõrvaldatud.

3.1 Money – Martin Fowler

Muster esitab rahalist väärtust.



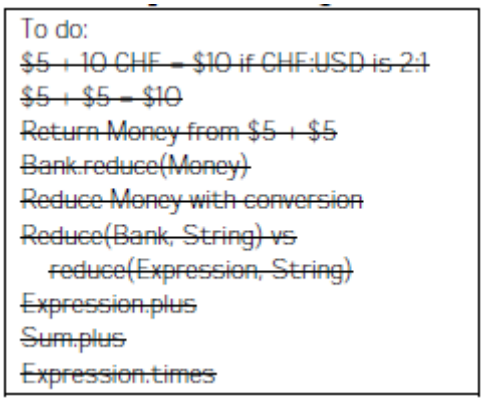
Joonis 29 Raha mudeli muster Martin Fowleri järgi

Mustri ehk klassi eesmärk on siduda omavahel kogus (*amount*) ja valuuta (*currency*). Kogus võib olla kas täis- (*integral)* või kümnendarvu (*decimal*) tüüpi. Kindlasti tuleks vältida ujukoma (*float*) tüüpi – see tekitab ümardamisprobleeme, mida antud muster just vältida püüab. Aritmeetilistest operatsioonidest on mudelil realiseeritud liitmine, lahutamine, korrutamine, jagamine ja võrdlemine [2].

Raha mudel Martin Fowleri eeskujul on projektis täielikult realiseeritud. Erinevalt realiseeritud mudelist, on Martin Fowleri mudelis piirdutud vaid ühe olemiga.

3.2 Money Example – Kent Beck

Kent Beck kasutab oma raha mudeli realiseerimiseks testipõhist metoodikat (Test Driven Development). Kuna kogu arendamine on testipõhine, siis ei looda esmase tegevusena ühtegi objekti – alustatakse testide kirjutamisest [17]. Järjepanu koodile läbi testide funktsionaalsust lisades (Joonis 30) loob ta mudeli, mis on võimeline valuutasid omavahel liitma, lahutama, korrutama ja neid omavahel ümardama.



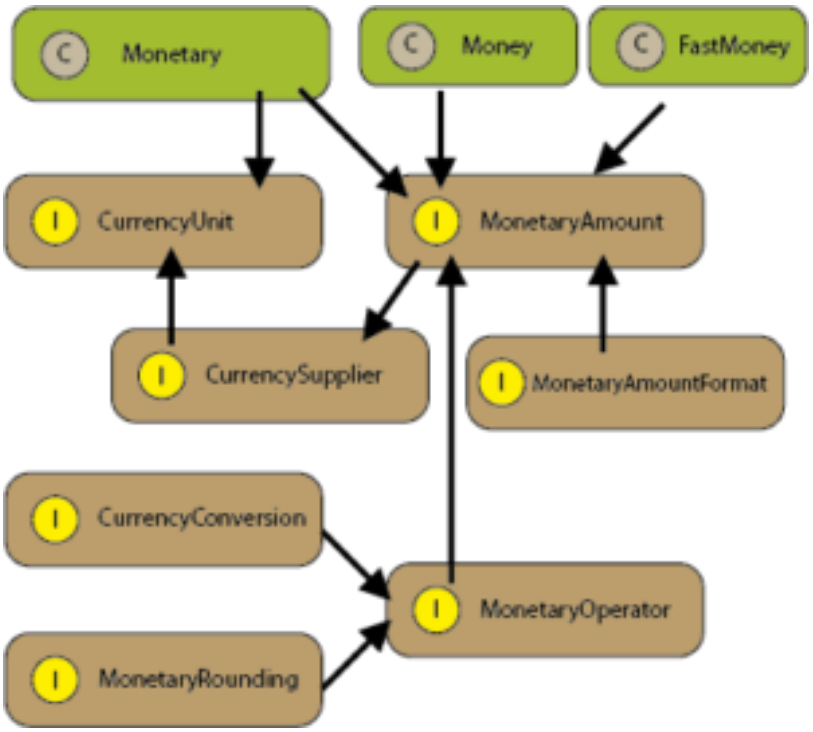
Joonis 30 TDD Money mudel

Erinevalt antud töö raames realiseeritud mudelist puuduvad Test Driven Development kaasabil loodud mudelil paljud klassid (näiteks Country selgitamaks, mis riigis mis valuutad käibel on ja Payment kui rahaga tehtava makse info).

3.3 Java Money API

Java platvormile on samuti arendatud API, mis esitab, edastab ja teostab raha ja valuutaga keerukaid kalkulatsioone. Nimeks on sel [21], mis on muutunud valuuta ja muude rahaliste suuruste osas standardiks. Mudel suudab läbi viia arvutusi, defineerida klassid, mis esitavad valuutasid ja rahalisi väärtusi, samuti ka rahalist ümardamist. API tuleb toime ka valuutavahetuskurssidega ning valuutade ja rahaliste väärtuste vormindamise ja *parse*'misega.

JSR-354 spetsifikatsioonis kaetud põhilisi klasse illustreerib Joonis 31.



Joonis 31 JSR 354: Money and Currency API

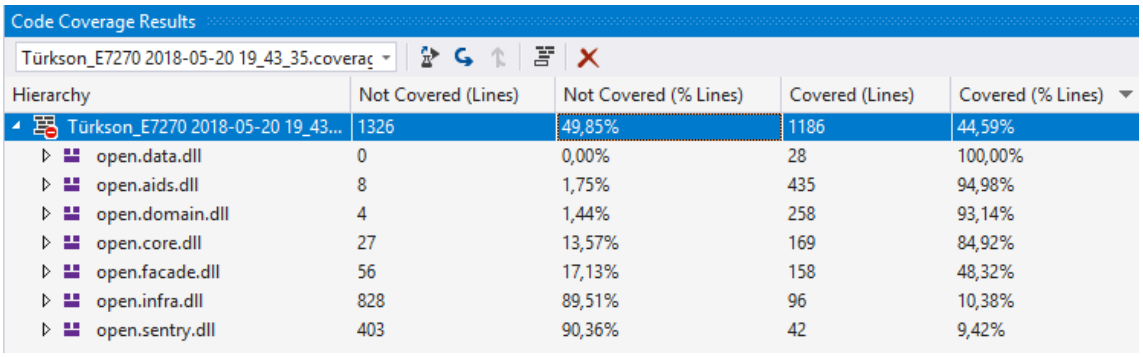
Antud mudelis on kaheks põhiliideseks *CurrencyUnit* ja *MonetaryAmount. CurrencyUnit* modelleerib valuutale omased atribuudid. Selle instantsid on kättesaadavad kasutades Monetary.getCurrency meetodit. Realiseeritud mudelis on selle vasteks klass *Currency.*

MonetaryAmount on aga rahalise väärtuse numbriline esitus ning seda seostatakse alati *CurrencyUnit väärtusega.* [19] Realiseeritud mudeli puhul oleks vasteks klass Currency. Siiski ei ole aga Java mudelis defineeritud näiteks maksete ja maksemeetodite klasse.

3.4 Koodi katvus

Koodi katvuse (code coverage [23]) all mõistetakse arvulist suurust, mis näitab kui suur osa lähtekoodist on testidega kaetud. Mida suurem on koodi katvus, seda paremat tööd on programmeerija teinud.

Code coverage töövahend ASP.NET Core rakenduses on leitav peale testide käivitamist menüüst Test -> Analyze Code Coverage -> All Tests. Realiseeritud projekti puhul on koodi katvus näha Joonis 32.



Joonis 32 Koodi katvus

3.5 Mis jäi tegemata?

- Nii mõnedki testid on veel poolikud ja seetõttu kukuvad läbi või neid ignoreeritakse
- Rakendus ei ole väga turvaline, neid atribuute ei ole veel ei lisatud ega ka testitud. Need on esialgu meelega välja jäetud ja tulevad kindlasti sisse vastuvõtutestide juures.
- Kasutajaliideses ei ole kõike kuvatud nii nagu tahaks. Valid from ja Valid to kuvamine võiks olla natuke teine, lehekülgede vahetuse disain ning ka vajalike valikute (Drop-down-list) kuvamine on asjad, millega tuleb veel tegeleda
- Pole proovitud rakenduse jõudlust (kui suurele koormusele vastu peab)
- Andmete allatõmbamine Euroopa Keskpangast on ajamahukas
- Valuutakursid saadakse praegu Euroopa Keskpangast, samasuguse andmete pärimise võiks teha USA Föderaalreservist

3.6 Peamised õppetunnid

- Programmeerimisele tuleb palju aega panustada, eriti kui teha asja hästi (lähtudes muuhulgas ka puhta koodi praktikast)
- Kõike, mida soovin saavutada ei oska ja/või ei jõua

- Suured ülesanded tuleb jagada väikesteks osadeks
- Ei tohi karta abi paluda
- Enda vastu tuleb aus olla, kui midagi on juba planeeritud, ei ole mõistlik selle tegemist edasi lükata

3.7 Edasised plaanid

Eesmärk on hetkel määramata ajal lõpetada tegemata asjad, millele sai viidatud peatükis 3.5. Kindlasti võtab see aega ja võib olla sobivaks lõputööteemaks nii magistriõppekaval kui ka mõne teise tudengi bakalaureuseõppekaval (olemasoleva projekti edasiarendus ja täiendus).

Loodud rakenduse alusel tuleks luua ka tehniline dokument kui juhend tudengitele aines „Infosüsteemide arendamine II“ samalaadse projekti koostamiseks.

Võimalus on veel integreerida loodud projekt Leedsi Ülikooli proteoomiksi uurimislabori töötajate poolt kasutatava Sentry rakenduse uude versiooni, kuna seal on raha mudel veel realiseerimata, kuid siiski vajalik selleks, et antud toodet oleks võimalik muuta ka teistele laboratooriumitele kasutatavaks.

Kokkuvõte

Üliõpilastele demonstreerimiseks on vajadus luua näiteprojekt, mille abil saab õpetada valdkonnamudelite alusel infosüsteemide realiseerimist. Seejuures võetakse Jim Arlow ja Ila Neustadt poolt kirja pandud raha valdkonna mudel [6] ja realiseeritakse see MVC rakendusena, tuginedes testidel tuginevale arendusele ja puhtale koodile [7].

Valitud mudel realiseeritakse Microsoft Visual Studio Enterprise 2017 tarkvara ja ASP.NET Core 2.0 raamistikku [8] kasutades. Need vahendid on kasutusel ka aine „Infosüsteemide arendamine II“ õpetamisel, millele on näiteprojekt orienteeritud.

Andmebaasiga suhtlemiseks kasutatakse raamistikku NET Entity Framework [13]. Entity Framework on oma olemuselt O/RM (Object-Relational Mapper) [14]. Selleks, et teha enda valdkonnamudel sõltumatuks Entity Framework raamistikust, on antud projektis realiseeritud ka veel andmehoidla vastavalt *repository* mustri [2] järgi.

Kuna nii mõnigi osa koodist jäi realiseerimata ja nii mõnedki testid on veel lõpuni tegema, siis on võimalik seda sama projekti ka veel tulevikuski täiendada. Projekti edasiarenduse korral on loodud kood plaanis lisada ka Sentry uude versiooni.

Kasutatud kirjandus

[1] A. Järviste, „Tarkvara arendusprotsess (Software Engineering Process),“ [Võrgumaterjal]. Available: kodu.ut.ee/~kiho/TVTkonspekt/sla2_4.ppt. [Kasutatud 19 05 2018].

[2] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2003, pp. 488-495.

[3] „IDK1604 - Infosüsteemide Arendamine II,“ [Võrgumaterjal]. Available: https://ained.ttu.ee/course/view.php?id=39.

[4] „IABB17/17,“ [Võrgumaterjal]. Available: https://ois.ttu.ee/portal/page?_pageid=37,674560&_dad=portal&_schema=PORTAL&p_action=view&p_fk_str_yksus_id=50001&p_kava_versioon_id=50402&p_net=internet&p_lang=ET&p_rezhiim=0&p_mode=1&p_from=.

[5] „Model-View-Controller arhitektuur,“ 08 04 2013. [Võrgumaterjal]. Available: http://maurus.ttu.ee/ained/IDU0200_2014/doc/82/MVC.pdf.

[6] J. Arlow ja I. Neustadt, Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML, Addison-Wesley Professional, 2004, p. Chapter 11. Money archetype pattern.

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

[8] „Introduction to ASP.NET Core,“ 28 02 2018. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/index?view=aspnetcore-2.0.

[9] „Architect Modern Web Applications with ASP.NET Core and Azure,“ Microsoft, [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/standard/modern-web-apps-azure-architecture/. [Kasutatud 19 05 2018].

[10] S. Gupta, „How to write clean code? Lessons learnt from “The Clean Code”— Robert C. Martin,“ Mindorks, 18 02 2018. [Võrgumaterjal]. Available: https://medium.com/mindorks/how-to-write-clean-code-lessons-learnt-from-the-clean-code-robert-c-martin-9ffc7aef870c. [Kasutatud 19 05 2018].

[11] „Common Web Application Architectures,“ 06 10 2017. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/standard/modern-web-apps-azure-architecture/common-web-application-architectures.

[12] „Modular enterprise architecture using MVC and Orchard CMS,“ 17 10 2012. [Võrgumaterjal]. Available: https://softwareengineering.stackexchange.com/questions/170417/modular-enterprise-architecture-using-mvc-and-orchard-cms.

[13] „Entity Framework Core Quick Overview,“ Microsoft, [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/ef/core/. [Kasutatud 19 05 2018].

[14] „Object-relational mapping,“ Wikipedia, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Object-relational_mapping. [Kasutatud 19 05 2018].

[15] „Three-Layered Services Application,“ Microsoft Developer Network, [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/ff648105.aspx. [Kasutatud 20 05 2018].

[16] O. Eini, „Infrastructure Ignorance,“ Ayende Rahien, 12 2 2008. [Võrgumaterjal]. Available: https://ayende.com/blog/3137/infrastructure-ignorance. [Kasutatud 20 05 2018].

[17] K. Ammor, „Structurer sa démarche de test,“ [Võrgumaterjal]. Available: https://afsy.fr/avent/2017/18-structurer-sa-demarche-de-test.

[18] T. Ottinger ja J. Langr, „Agile in a Flash,“ [Võrgumaterjal]. Available: http://agileinaflash.blogspot.com.ee/2009/02/first.html.

[19] „Testing Overview,“ Microsodt Developer Network, [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/ff647247.aspx. [Kasutatud 20 05 2018].

[20] K. Beck, Test Driven Development: By Example, Addison-Wesley Professional, 2002, p. Section I: Money Example.

[21] „JSR 354: Money and Currency API,“ Java Community Process, [Võrgumaterjal]. Available: https://jcp.org/en/jsr/detail?id=354. [Kasutatud 19 05 2018].

[22] „Java Money and the Currency API,“ 13 01 2018. [Võrgumaterjal]. Available: http://www.baeldung.com/java-money-and-currency.

[23] „Using Code Coverage to Determine How Much Code is being Tested,“ Microsoft Developer Network, [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/dd537628.aspx. [Kasutatud 19 05 2018].

Lisa 1 – Money.cs kood

```
using Open.Core;
using Open.Domain.Common;
using System;

namespace Open.Domain.Money {
   public class Money : Value<Currency, decimal, Money>, IComparable<Money> {
      private RoundingPolicy rounding;

      public Money() : this(null) { }

      public Money(Currency c, decimal amount = 0, DateTime? dt = null) : base(
         c ?? new Currency(), amount) {
         Date = dt ?? DateTime.Now;
      }

      public override Money Add(Money m) {
         var money = ConvertMoney.ToCurrency(this, m.Currency);
         return new Money(m.Currency, m.Amount + money.Amount, m.Date);
      }

      public override int CompareTo(Money m) {
         var money = ConvertMoney.ToCurrency(m, Currency);
         return Amount.CompareTo(money.Amount);
      }

      public override Money ConvertTo(Currency c) { return ConvertMoney.ToCurrency(this, c); }

      public DateTime Date { get; }

      public override Money Divide(decimal d) {
         d = (d == decimal.Zero)? decimal.MaxValue : Amount / d;
         var m = new Money(Currency, d, Date);
         return m;
      }

      public Currency Currency => unit;

      public override Money Multiply(decimal d) {
         d = Amount * d;
         var m = new Money(Currency, d, Date);
         return m;
      }


      public override Money Round(RoundingPolicy policy) {
         rounding = policy;
         var d = round();
         return new Money(Currency, d);
      }

      public override Money Subtract(Money m) {
         var money = ConvertMoney.ToCurrency(this, m.Currency);
         return new Money(m.Currency, money.Amount - m.Amount, m.Date);
      }
```

```
        internal decimal round() {
            var p = rounding;
            var d = Amount;
            if (p.Strategy == RoundingStrategy.RoundUp) return RoundOff.Up(d, p.Decimals);
            if (p.Strategy == RoundingStrategy.RoundDown) return RoundOff.Down(d, p.Decimals);
            if (p.Strategy == RoundingStrategy.RoundUpByStep) return RoundOff.UpByStep(d, p.Step);
            if (p.Strategy == RoundingStrategy.RoundDownByStep)
                return RoundOff.DownByStep(d, p.Step);
            if (p.Strategy == RoundingStrategy.RoundTowardsPositive)
                return RoundOff.TowardsPositive(d, p.Decimals);
            if (p.Strategy == RoundingStrategy.RoundTowardsNegative)
                return RoundOff.TowardsNegative(d, p.Decimals);
            return RoundOff.Off(d, p.Decimals, p.Digit);
        }

        public override int CompareTo(object obj) { return CompareTo(obj as Money); }
    }
}
```

Lisa 2 – EuroRatesDbTableInitializer.cs

```
using Open.Aids;
using Open.Domain.Money;
using System;
using System.IO;
using System.Linq;
using System.Xml;

namespace Open.Infra.Money {
   public static class EuroRatesDbTableInitializer {
      internal const string example = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
                              "<gesmes:Envelope xmlns=\"http://www.ecb.int/vocabulary/2002-08-01/eurofxref\" "
+
                              "xmlns:gesmes=\"http://www.gesmes.org/xml/2002-08-01\">" +
                              "<gesmes:subject>Reference rates</gesmes:subject><gesmes:Sender>" +
                              "<gesmes:name>European Central Bank</gesmes:name></gesmes:Sender>" +
                              "<Cube><Cube time=\"2015-03-05\"><Cube rate=\"1.1069\" currency=\"USD\"/>" +
                              "<Cube rate=\"133.10\" currency=\"JPY\"/><Cube rate=\"1.9558\"
currency=\"BGN\"/>" +
                              "<Cube rate=\"27.422\" currency=\"CZK\"/><Cube rate=\"7.4542\"
currency=\"DKK\"/>" +
                              "<Cube rate=\"0.72510\" currency=\"GBP\"/><Cube rate=\"305.36\"
currency=\"HUF\"/>" +
                              "<Cube rate=\"4.1397\" currency=\"PLN\"/><Cube rate=\"4.4453\"
currency=\"RON\"/>" +
                              "<Cube rate=\"9.2140\" currency=\"SEK\"/><Cube rate=\"1.0697\"
currency=\"CHF\"/>" +
                              "<Cube rate=\"8.5460\" currency=\"NOK\"/><Cube rate=\"7.6585\"
currency=\"HRK\"/>" +
                              "<Cube rate=\"67.6095\" currency=\"RUB\"/><Cube rate=\"2.8663\"
currency=\"TRY\"/>" +
                              "<Cube rate=\"1.4205\" currency=\"AUD\"/><Cube rate=\"3.3009\"
currency=\"BRL\"/>" +
                              "<Cube rate=\"1.3770\" currency=\"CAD\"/><Cube rate=\"6.9382\"
currency=\"CNY\"/>" +
                              "<Cube rate=\"8.5847\" currency=\"HKD\"/><Cube rate=\"14363.07\"
currency=\"IDR\"/>" +
                              "<Cube rate=\"4.4270\" currency=\"ILS\"/><Cube rate=\"68.9098\"
currency=\"INR\"/>" +
                              "<Cube rate=\"1218.84\" currency=\"KRW\"/><Cube rate=\"16.6566\"
currency=\"MXN\"/>" +
                              "<Cube rate=\"4.0416\" currency=\"MYR\"/><Cube rate=\"1.4777\"
currency=\"NZD\"/>" +
                              "<Cube rate=\"48.848\" currency=\"PHP\"/><Cube rate=\"1.5156\"
currency=\"SGD\"/>" +
                              "<Cube rate=\"35.885\" currency=\"THB\"/><Cube rate=\"13.0163\"
currency=\"ZAR\"/>" +
                              "</Cube></Cube></gesmes:Envelope>";
      internal const string historyUrl = "http://www.ecb.europa.eu/stats/eurofxref/eurofxref-hist.xml";
      internal const string dailyUrl = "http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml";
      private const string cubeString = "Cube";
      private const string timeString = "time";
      private const string rateString = "rate";
      private const string currencyString = "currency";
      public static void LoadRatesHistory(Action<string, decimal, DateTime> addRate,
```

```
        bool readFromEcb = false, DateTime? toDate = null) {
        var date = toDate ?? DateTime.MinValue;
        var s = Utils.IsTesting(readFromEcb) ? example : WebService.Load(historyUrl);
        ReadXml(s, date, addRate);
    }
    public static void LoadDailyRates(Action<string, decimal, DateTime> addRate, bool readFromEcb =
false) {
        var s = Utils.IsTesting(readFromEcb) ? example : WebService.Load(dailyUrl);
        ReadXml(s, DateTime.MinValue, addRate);
    }
    public static void ReadXml(string s, DateTime toDate, Action<string, decimal, DateTime> addRate) {
        var doc = ToXmlDocument(s);
        foreach (XmlNode node in doc.ChildNodes) {
            foreach (XmlNode cube in node.ChildNodes) {
                if (cube.Name != cubeString) continue;
                var i = 0;
                foreach (XmlNode time in cube.ChildNodes) {
                    i++;
                    if (i > 10000) return;
                    var date = GetDate(time);
                    if (date < toDate) break;
                    AddRates(time, date, addRate);
                }
            }
        }
    }
    public static void AddRates(XmlNode time, DateTime date,
        Action<string, decimal, DateTime> addRate) {
        foreach (XmlNode rate in time.ChildNodes) { AddRate(date, rate, addRate); }
    }
    public static void AddRate(DateTime date, XmlNode rate,
        Action<string, decimal, DateTime> addRate) {
        if (!GetRate(rate, out decimal value)) return;
        var currency = GetCurrency(rate);
        addRate(currency, value, date);
    }

    public static string GetCurrency(XmlNode node) { return GetValue(node, currencyString); }

    public static DateTime GetDate(XmlNode node) {
        var date = GetValue(node, timeString);
        return DateTime.ParseExact(date, "yyyy-MM-dd", UseCulture.Invariant);
    }
    public static bool GetRate(XmlNode node, out decimal rate) {
        var v = GetValue(node, rateString);
        return decimal.TryParse(v, out rate);
    }
    public static XmlDocument ToXmlDocument(string s) {
        var stream = new StringReader(s);
        var r = XmlReader.Create(stream);
        var d = new XmlDocument();
        d.Load(r);
        return d;
    }
    public static string GetValue(XmlNode n, string name) {
        return Safe.Run(() => {
            var a = n.Attributes[name];
            return a.Value;
        }, string.Empty);
    }

    public static void Initialize(SentryDbContext c) {
        var i = 0;
        void addRate(string currencyId, decimal rate, DateTime date) {
```

```
            var r = RateFactory.Create(currencyId, rate, date, RateFactory.EuroRate);
            c.Rates.Add(r.DbRecord);
            if (i++ < 1000) return;
            c.SaveChanges();
            i = 0;
        }

        c.Database.EnsureCreated();
        if (c.Rates.Any()) return;
        LoadRatesHistory(addRate);
        c.SaveChanges();
    }
  }
}
```

Lisa 3 – ConvertMoney.cs

```
using System;

namespace Open.Domain.Money {
  public static class ConvertMoney {
    internal static IRateRepository rates;

    public static Money ToCurrency(Money m, Currency c) {
      var dt = m?.Date ?? DateTime.Now;
      var amount = m?.Amount ?? 0M;
      var rateFrom = getRate(m?.Currency, dt);
      var rateTo = getRate(c, dt);
      try {
        amount = amount / rateFrom;
        amount = amount * rateTo;
      } catch { amount = decimal.MaxValue; }
      return new Money(c, amount, dt);
    }

    internal static decimal getRate(Currency c, DateTime dt) {
      if (c is null) return decimal.Zero;
      var rateID = RateFactory.CalculateID(c.ID, dt, RateFactory.EuroRate);
      var rate = rates?.GetRate(rateID);
      var r = rate?.DbRecord?.Rate;
      if (r is null || r == 0) r = getClosestKnownRate(c, dt);
      return (decimal) r;
    }

    internal static decimal getClosestKnownRate(Currency c, DateTime dt) {
      var r = decimal.One;
      var d = DateTime.MaxValue.Ticks;
      if (c is null) return decimal.Zero;
      var l = rates?.GetRates(c);
      if (l == null) return r;
      foreach (var e in l) {
        var d1 = Math.Abs(e.DbRecord.ValidFrom.Date.Ticks - dt.Date.Ticks);
        if (d1 > d) continue;
        d = d1;
        r = e.DbRecord.Rate;
      }

      return r;
    }
  }
```