TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Marko Bode 185638IADB

.NET Upgrade Assistant tööriista parendamine

Bakalaureusetöö

Juhendaja: Liisa Jõgiste
MSc

Tallinn 2022

Autorideklaratsioon

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

Autor: Marko Bode

16.05.2022

Annotatsioon

Lõputöö eesmärgiks on parendada Microsofti poolt välja töötatud .NET *Upgrade Assistant* tööriista muutes seda paindlikumaks andes arendajatele rohkem funktsionaalsust tööriista kasutamisel.

Lõputöös antakse ülevaade .NET ökosüsteemist, muutustest .NET raamistike vahel ja analüüsitakse uuendamist abistavaid tööriistu.

Tulemusena valmisid 2 laiendust kasutades .NET *Upgrade Assistanti* laienduste süsteemi. Laiendused publitseeriti NuGet paketina kõigile tasuta kasutamiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 33 leheküljel, 6 peatükki, 13 joonist, 1 tabelit.

Abstract

Enhancement of the .NET Upgrade Assistant Tool

The objective of current thesis is to improve .NET Upgrade Assistant tool developed by Microsoft. The tool is enhanced by adding functionality and as a result making it more flexible.

Part of the thesis gives an overview of .NET ecosystem and the changes between .NET frameworks and from there on an analysis of tools to assist upgrading the framework is provided.

As a result of the thesis 2 extensions are created by using the .NET Upgrade Assistant`s extension system. The extensions are published as NuGet packages for public use.

The thesis is in Estonian and contains 33 pages of text, 6 chapters, 13 figures, 1 table.

Lühendite ja mõistete sõnastik

.NET	Arendusplatvorm rakenduste ja teenuste loomiseks, seahulgas näiteks veebirakendused, mobiilrakendused, pilveteenused ja mikroteenused
.NET Core	Platvormidevaheline ja vabatarkvaraline raamistik modernseks rakenduste arenduseks
.NET Framework	Raamistik rakenduste arenduseks Windowsile, .NET Core eelkäija
.NET Standard	Ametlik spetsifikatsioon, mis määrab API saadavuse .NET erinevatel versioonidel
API	*Application Programming Interface*, rakendusliides
ASP.NET	Raamistik veebirakenduste ehitamiseks .NET platvormil
ASP.NET Core	Modernsem versioon ASP.NET raamistikust
Class Library	Projektitüüp rakenduse koodi eraldamiseks, klasside teek
C#	Programmeerimiskeel
Excel	Tabelarvutussüsteem
F#	Programmeerimiskeel
Git	Versioonihaldustarkvara
IDE	*Integrated Development Environment*, integreeritud programmeerimiskeskkond
JSON	*JavaScript Object Notation*, andmevahetusvorming
LTS	*Long-term support*, viide elutsüklile
MVC	*Model-View-Controller*, arhitektuurimuster
NuGet	.NET platvormi paketihaldur
Razor	ASP.NET raamistiku programmeerimissüntaks
SARIF	*Static Analysis Results Interchange Format*, staatilise analüüsi väljundi standardvorming
Separation of Concerns	disainipõhimõte
TFM	*Target Framework*, viide rakenduses sihitud raamistikule
XML	*Extensible Markup Language*, märgistuskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesolevas lõputöös analüüsitakse .NET raamistiku uuendamist ja uuendamist abistavaid tööriistu keskendudes ASP.NET ja ASP.NET Core projektidele. Tööriistad, mida uuritakse on .NET *Portability Analyzer, Try-convert* tööriist ja .NET *Upgrade Assistant* (edaspidi *Upgrade Assistant*).

Praktilises osas luuakse *Upgrade Assistanti* puudusest tulenevalt laiendused, mis muudavad tööriista paindlikumaks ja läbinähtavamaks, vähendades seejuures käsitööd rakenduse uuendamist läbiviivale arendajale. Laienduse loomiseks kasutatakse *Upgrade Assistanti* laienduste süsteemi. Laiendused publitseeritakse NuGet paketina kõigile tasuta kasutamiseks.

Laienduste töötamise valideerimiseks kasutatakse loodud laiendusi koos *Upgrade Assistantiga*, et uuendada üks ASP.NET ja üks ASP.NET Core projekt värskemaile .NET LTS versioonile, milleks lõputöö koostamise hetkel on NET 6. Projektides sihitavad .NET raamistiku versioonid enne uuendamist on .NET Framework 4.8 ja .NET 5.

Lõputöö koosneb viiest peatükist – probleemi kirjeldus, taust, olemasoleva tööriistade analüüs, laienduste realiseerimine koos kasutamisega ja tulemuse kirjeldus.

1.1 Probleemi kirjeldus ja eesmärk

.NET raamistikul anti välja uus LTS versioon .NET 6. See tähendab, et hiljemalt 3. detsembriks 2022, millal lõppeb eelmise LTS versiooni toetamise aeg, on uute funktsioonide, turvaparanduste ja jõudlust parandavate uuenduste saamiseks vaja viia .NET raamistikul põhinevad projektid .NET 6 peale [1].

Uuenduse läbiviimiseks on Microsoft töötanud välja oma avatud lähtekoodiga tööriista .NET *Upgrade Assistant*. Tööriist aitab .NET raamistikul põhinevaid projekte järk-järgult üle viia modernsemale raamistiku versioonile, märkimisväärselt kiirendades uuendamisprotsessi [2].

Tööriist on mugav, kuid tulemus ei ole alati selline nagu esialgu eeldada võiks- kogu projekti siiski ei pruugita uuendada arendaja poolt valitud raamistiku versioonile.

Antud lõputöö eesmärk on uurida .NET raamistiku uuendamist ja pakkuda arendajale rohkem paindlikust ja läbinähtavust kasutades *Upgrade Assistant* tööriista.

1.2 Metoodika

Käesolev lõputöö on arendusuuring, mille eesmärgiks on leida olemasolevast tööriistast puudused ja pakkuda neile lahendus. Arendusuuringu protsessid jaotuvad etappidesse: eeluuring, realiseerimine, realiseeritu toimimise tõestamine ja kokkuvõte.

Analüütilises osas teostatakse eeluuring .NETi ökosüsteemist ja olulisematest muudatustest versioonide vahel. Seejärel uuritakse *Upgrade Assistant* tööriista võimekust ja puuduseid ning antakse sisend praktilisele osale.

Praktilises osas teostatakse laienduste realiseerimine ja valideeritakse toimimist kasutades loodud laiendust.

Kokkuvõtvas osas tuuakse selgelt välja lõputöö tulemused ja tõstetakse esile võimalikud edasiarendused.

1.3 Skoop

Käesoleva lõputöö skoobis on .NET raamistiku uuendamine värskeimale .NET LTS versioonile, mis lõputöö kirjutamise hetkel on .NET 6. Lõputöö skoobist jääb välja rakenduse viimine muudele .NET versioonidele.

Lõputöös keskendutakse ASP.NET rakendustel, sest ASP.NET on populaarseim .NET raamistikul põhinev tehnoloogia [3]. Samuti on skoobis *Class Libraryd*, sest väga tihti on ühes ASP.NET rakenduses kasutusel *Class Libraryd* tulenevalt murede eraldatavuse arenduspritsiibist, tuntud ka kui *Separation of Concerns*.

2 .NET raamistiku uuendamine

Uuendamise eripära .NET rakenduste puhul sõltub suuresti, milliselt versioonilt millisele minnakse. Antud lõputöös on päevakorras rakenduse uuendamine värskeimale .NET LTS versioonile. Vaadeldakse ka võimalust rakendus .NET Standardile viia. Järgnevas osas antakse täpsem ülevaade .NET ökosüsteemist, mis aitab paremini mõista eelnevalt mainitut. Lisaks tuuakse välja suuremad muutused .NET raamistike ja versioonide vahel ja kaalutakse vajadust projekti uuele LTS versioonile uuendada.

2.1 .NET ökosüsteem

.NET ökosüsteemis on tänapäeval 3 peamist komponenti, millest on tarvis aru saada, et projekti edukalt uuendada: .NET Framework, .NET Core ja .NET Standard. Lõputöö skoobist lähtuvalt tutvustatakse siin alapeatükis ka ASP.NETi.

Raamistikud on funktsioonide kogumid, mis aitavad rakendada rakendustele ühiseid baasloogikaid nagu näiteks rakenduse käivitamine, andmebaasiga ühildumine ja turvalisuse tagamine kasutajatele [4]. Seeläbi muudavad raamistikud koodi erinevatel rakendustel sarnasemaks ja arendajatel on tänu ühtsusele võimalik efektiivsemalt tundmatud koodi lugeda.

.NET Framework on eelnevalt mainitud raamistikest varaseim ja kategoriseerib tänapäeval pärandkoodi alla. .NET Framework sisaldab endas .NETi käitusplatvormi, mis omakorda on osa ühisest infrastruktuurist, mis on võimeline jooksutama koodi, sisaldab endas Just-in-Time kompilaatorit ja korraldab prügi koristust. .NET Frameworkil on põhiversioonid 1-4, millel võib omakorda olla alamversioone [5].

2016. aastal tuli välja .NET Core, mis on eraldiseisev raamistik ning .NET Frameworki edasiarendus, kuid pole .NET Frameworki asendus. .NET Core on kirjutatud nullist, selleks et raamistik oleks kiirem, kergem ja ühilduks samal ajal mitme erineva operatsioonisüsteemiga- see on ka nende kahe raamistiku peamine erinevus. .NET

Frameworki suurim nõrkus on rakenduse ehitamine ainult Windows operatsioonisüsteemile. [5] [6]

.NET Core'il on põhiversioonid 1-3. Pärast .NET Core 3.1 otsustati .NET Core 4 vahele jätta, sest .NET Frameworki viimane põhiversioon oli 4 ja see vähendaks segadust versioonide vahel. Samuti otsustati eesliitest eemaldada „Core", et rõhutada asjaolu, et edaspidi on tegu .NETi peamise teostusega. Alates siit nimetatakse versioone eesliitega „.NET", kuid .NET 5 ja 6 puhul on siiski tegu .NET Core raamistiku versioonidega. [7]

Kuna .NET Framework ja .NET Core on erinevad raamistikud, siis saab järeldada, et liikudes .NET raamistike versioonide vahel, näiteks eelmisest LTS versioonist (.NET Core 3.1) uude LTS versiooni (.NET 6), on kasutaja koodis vaja teha vähem muudatusi kui liikudes mõnelt .NET Frameworki versioonilt mõne .NET Core versiooni peale.

.NET Framework ja .NET Core sisaldavad endas erinevaid rakendusliideseid. See teeb koodi taaskasutamise raskeks, sest ühe käituskeskkonna rakendusliiges ei kompileeru teises keskkonnas. Selle mure lahendamiseks lõi Microsoft .NET Standard teegi [5].

.NET Standardit implementeerivad nii .NET Frameworki versioonid alates 4.5-st kui ka kõik .NET Core versioonid [5]. See tähendab, et projekt, mis sihib .NET Standardi mõnda versiooni, on kasutatav erinevate raamistike ja versioonide peal nii kaua kui vastav raamistiku versioon rakendab vastavat .NET Standardi versiooni.

Samuti on .NET Standard tagasiühilduv, mis tähendab, et .NET Standard 2.0 implementeeriv teek kompileerub nii .NET Framework 4.8 kui ka .NET 6 projektis kuigi .NET 6 implementeerib .NET Standard 2.1 [8].

.NET Standardi versiooninumbri kasvades toetatud rakendusliideste arv kasvab, kuid toetatud .NET Frameworki ja .NET Core versioonide arv väheneb. Microsoft soovitab sihtida .NET Standard 2.0, et saada võimalikult lai haare mõlemalt poolt. [9] [8]

Rakenduse uuendamisel .NET 6-le saab arendaja valida, kas rakenduse *Class Libraryd* peaks sihtima .NET Standardit või .NET 6. .NET Standardit sihtides saab teeke kasutada ka muudes .NET lahendustes, kus raamistiku versioon on teine, kuid sihtides .NET 6 on võimalus kasutada uusimaid raamistiku võimalusi kui ka uusimaid C# versioone [10].

2.1.1 ASP.NET

ASP.NET laiendab .NET platvormi erinevate tööriistade ja teekidega, mis aitavad ehitada veebirakendusi. Näiteks sisaldab ASP.NET endas võimekust töödelda veebipäringuid, autentimissüsteemi, mis toetab kahefaktorilist autentimist ja autentimist läbi väliste teenuste nagu näiteks Google. [11]

Sarnaselt .NET Frameworki ja .NET Core omavahelistele seostele on seotud ASP.NET ja ASP.NET Core. ASP.NET Core on ASP.NETi ümberkirjutus, mille eeliseks on parem testitavus, võime ehitada nii API-liidest kui ka kasutajaliidest, mis kasutab Razor lehtesid. Samuti on ASP.NET Core'i on sisse ehitatud sõltuvussüstimine ja rakendust on võimalik hostida IISil, Kestrelil, Apachel ja Dockeril. [12]

Üldiselt kui rakendus kasutab .NET Frameworki, siis peaks rakendus kasutama ASP.NETi, kuid varasemaid ASP.NET Core'i versioone on võimalik kasutada .NET Frameworkil. Kui rakendus on .NET Core peal, siis ainuke valik on kasutada ASP.NET Core'i. [13]

2.1.2 NuGet paketid

NuGet on Microsofti poolt toetatud haldur, läbi mille saavad arendajad jagada ja kasutada endale vajalikku koodi, mida kutsutakse pakettideks [14]. NuGet pakett on ZIP fail, milles on kompileeritud kood ning failid seotud selle koodiga [14]. NuGet paketi faililaiend on *.nupkg* ja paketi konfiguratsioon asub failis *.nuspec*.

NuGet pakette saab teha igaüks ning jagada neid vastavalt oma soovile- kõigile, oma organisatsioonile või kasutada ainult iseenda projektides [14].

Projektis kasutusel olevate pakettide referentse ja versioone talletakse kahel erineval viisil. .NET Frameworkil hoitakse referentse vaikimisi *packages.config* failis (vt Joonis 1) ja .NET Core'il hoitakse neid projekti failides (edaspidi ka kui *.csproj* fail) kasutades *PackageReference* sõlme (vt Joonis 2) [15].

```xml
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Castle.Core" version="3.3.3" targetFramework="net48" />
  <package id="Castle.Windsor" version="3.4.0" targetFramework="net48" />
  <package id="EntityFramework" version="6.1.3" targetFramework="net48" />
  <package id="EntityFramework.Extended" version="6.1.0.168" targetFramework="net48" />
```

Joonis 1. Osa packages.config failist.

```xml
<ItemGroup>
    <PackageReference Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="8.1.1" />
    <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="5.0.5" />
    <PackageReference Include="Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore" Version="5.0.5" />
    <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="5.0.5" />
```

Joonis 2. Osa .csproj failis olevatest PackageReference sõlmedest.

.NET Framework toetab ka *PackageReference'i* ning kasutades Visual Studio 2017 IDE-t või hilisemat versiooni sellest saab konverteerida ühest teise. [12]

Samas ei toetata hetkel konverteerimist ASP.NET rakendustel. Lisaks pole kõigil NuGet pakettidel toetatud *PackageReference*. Kui konverteerimise käigus peaks taoline mittetoetamine ilmsiks tulema, siis on võimalik kogu konverteerimine tagasi pöörata. [16]

2.2 Muutused raamistike ja versioonide vahel

Kuna .NET Core on .NET Frameworki ümberkirjutus, siis on raamistike vahel palju projekti lõhkuvaid muudatusi [17]. Sama raamistiku eri versioonide vahel on samuti lõhkuvaid muudatusi, mis nõuvad mõnel määral koodi ümberkirjutamist, kuid neid on vähem ja lõhutud koodi on kergem parandada.

Kõiki muudatusi kirjeldada on väga ajakulukas ning pole antud lõputöös peamine eesmärk. Järgnevaga tutvustakse mõningaid suuremaid ja tähtsamaid muudatusi koodis ja projekti struktuuris, mis on lõputöö raames olulisemad. Samuti on suur osa muutuseid väga olukorrast sõltuvad ja rakendusespetsiifilised, mistõttu on kõige mõistlikum arendajal tutvuda ametliku dokumentatsiooniga.

Suurteks valukohtadeks raamistiku uuendamisel on muutused rakenduse konfigureerimisel, käivitumisel, sõltuvussüstimisel, marsruutimisel ja hostimisel. Lisaks

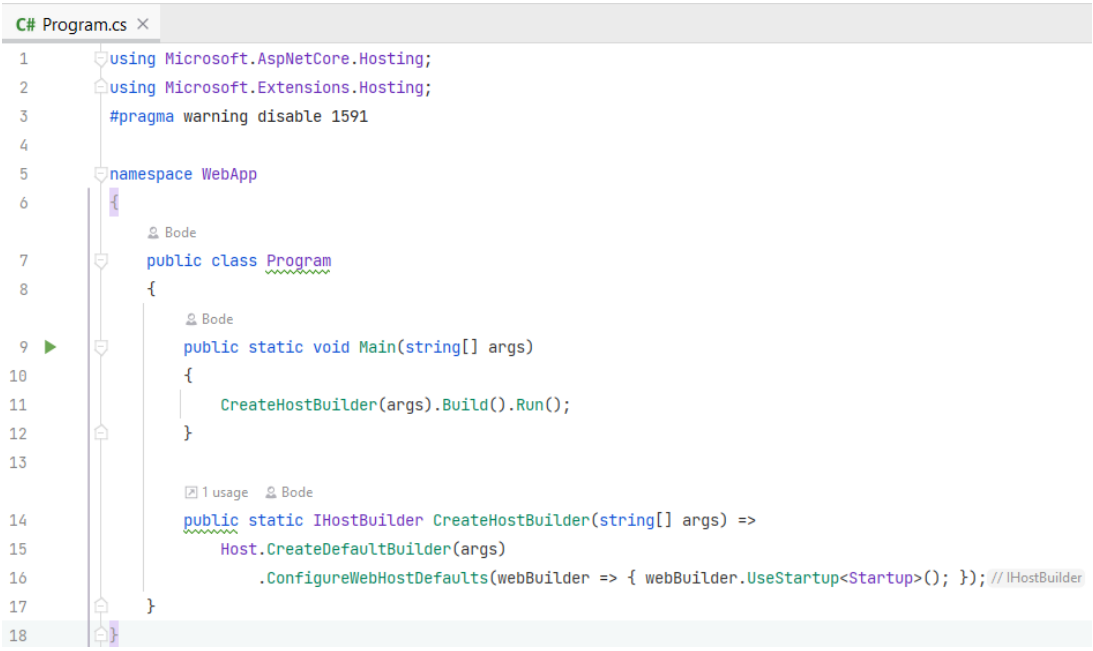
võivad suurt tähelepanu nõuda NuGet pakettide muutused- osad paketid, mis on .NET Framework projektis kasutusel ei pruugi olla saadaval .NET Core peal või kui NuGet pakett mõnel põhjusel sihib midagi muud peale .NET Standardi, siis võib olla oht, et pakett ei tööta pärast uuendamist, isegi kui uuendatakse .NET Core versioonilt .NET Core versioonile.

Rakenduse uuendamisel on tähtis uurida NuGet pakettidest sõltuvusi. Kuigi on palju selliseid pakette, millel on olemas .NET 6-ga ühilduv versioon, siis leidub palju selliseid pakette, mida .NET 6 peal kasutada ei saa. Reeglina, mida suurem on uuendamisel tehtav hüpe, seda rohkem protsentuaalselt selliseid pakette on.

Selliste pakettide puhul tuleb üldjuhul leida asendus [37], mis võib tähendada suurt koodi ümberkirjutamist. Kui pakett on vabatarkvaraline, siis teine valik on pakett ise uuendada .NET Standardile või .NET 6-le [37] ja seejärel sama paketti edasi kasutada.

ASP.NET rakendusel paikneb konfigureerimine *Global.asax* failis, kus konfigureeritakse nii marsruutimist kui ka registreeritakse rakendusse sõltuvussüstimine. ASP.NET Core'i rakendusele uuendamisel tuleb kogu konfiguratsioon liigutada *Startup.cs* faili, kusjuures tuleb vastavalt API-de kasutusele osa konfigureerimist nullist kirjutada.

ASP.NET Core'i peale uuendades tuleb arvestada, et sellised rakendused sisaldavad *Program.cs* faili (vt Joonis 3), mis käitub sisenemispunktina ja kutsub välja *Startup.cs* faili sisu. Uuendades mõnelt varasemalt .NET Core versioonilt .NET 6 peale, on samuti hea teada, et enam pole eraldiseisva *Startup.cs* faili loomine vaikimisi lähenemine, kuid siiski toetatud [17].



Joonis 3. Lihtne näide .NET 5 Program.cs failist.

Uueks vaikimisi lähenemiseks on kogu kood kirjutada *Program.cs* faili (vt Joonis 4). Samuti enam ei pea looma *Program* klassi, vaid rakendus leiab käivituskoodi faili nime järgi.

```
C# Program.cs
1   var builder = WebApplication.CreateBuilder(args);
2
3   // Add services to the container.
4   builder.Services.AddControllersWithViews();
5
6   var app:WebApplication = builder.Build();
7
8   // Configure the HTTP request pipeline.
9   if (!app.Environment.IsDevelopment())
10  {
11      app.UseExceptionHandler("/Home/Error");
12      // The default HSTS value is 30 days.
13      // You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
14      app.UseHsts();
15  }
16
17  app.UseHttpsRedirection();
18  app.UseStaticFiles();
19
20  app.UseRouting();
21
22  app.UseAuthorization();
23
24  app.MapControllerRoute(
25      name: "default",
26      pattern: "{controller=Home}/{action=Index}/{id?}");
27
28  app.Run();
```

Joonis 4. Lihtne näide .NET 6 Program.cs failist.

ASP.NET-ile pole sisse ehitatud sõltuvussüstimist. Seetõttu on paljud rakendused kasutusele võtnud NuGet paketid, mis annavad rakendusele *Inversion of Conrol* konteineri. Populaarsemateks pakettideks on näiteks Autofac, Unity ja Castle Windsor [17].

Kui aplikatsioon kasutab mõnda sellist paketti, siis tuleb kontrollida, kas paketil on olemas .NET Core versioon- olemasolemise korral on hea võimalus, et sõltuvussüstimist pole vaja nullist konfigureerida. Kui uue versiooniga paketti pole, siis on mõistlik proovida ASP.NET Core'i sisse ehitatud konteinerit. [17]

2.3 Uuendamisel sihitava versiooni valik

Enne raamistiku uuendamist on vaja kindlaks teha, kas peaks üldse uuendama. ASP.NET Core'ile tasuks uuendada, sest tegu on ühe kiireima veebiraamistikuga, mis siiani loodud on [18]. Samuti tasuks uuendada kui on soov saada ühilduvust muude platvormidega kui Windows. Lisaks on .NET Core järjest edasi arendatav raamistik ning toetab uusi C# versioone- viimane .NET Frameworkis kasutatav C# versioon on C# 8. [16, 18]

Uuendada pole tarvis kui .NET Frameworkil olev projekt pole aktiivses arendusfaasis ja ei saaks suurt kasu eelnevalt väljatoodud plussidest [17]. Samuti ei toeta .NET Core osasid tehnoloogiaid, mida toetab .NET Framework. Sellised tehnoloogiad on näiteks rakenduse domeen, rakenduse domeenide vaheline suhtlemine ja *System.EnterpriseServices* [19].

Kasutades mõnda mainitud tehnoloogiatest võib olla mõistlikum jääda .NET Frameworkile.

Kaaludes migreerimist varasemalt .NET Core versioonilt .NET 6-le on hea teada, et 3. detsembril 2022 lõppeb eelmise LTS versiooni toetamise aeg ja uute funktsioonide, turvaparanduste ja jõudlust parandavate uuenduste saamiseks on vaja viia .NET raamistikul põhinevad projekt .NET 6 peale [1].

Lisaks on .NET 6 kiireim täispinu raamistik ja koostöös Visual Studio 2022-ga pakutakse kuuma uuesti laadimist ja uusi tööriistu testimiseks ja diagnostikaks [20].

Võimalus on projekti uuendamisel sihtida .NET Standardit. Sedasi oleks näiteks üks *Class Library* kasutatav erinevates rakendustes kui selleks peaks olema vajadus. See-eest sedasi ei saa *Class Library* kasutada .NET 6 eeliseid. Lahenduseks saab ühes projektis sihtida mitut raamistikku, näiteks nii .NET Standard 2.0 kui ka .NET 6 korraga.

Kokkuvõttes tuleks projekti uuendada kui aktiivsel arendamisel oleks uutest võimalustest kasu. Uuendades tuleks sihtida .NET 6, sest tegu on kiireima raamistikuga, sellel on palju uut funktsionaalsust ja olles LTS versioon, tehakse .NET 6-le lisaarendusi ja turvaparandusi veel ligi 3 aastat.

2.4 Uuendamise strateegiad

Uuendades .NET raamistikku saab eristada mitut strateegiat.

Esimene neist on uuendada projekt otse .NET 6 peale. Sedasi on uuendamisprotsess ühekordne, kuid lõhkuvate muudatuste arv võib olla suur ja uuendamine võib olla väga mahukas [21]. Olles juba .NET Core peal, on antud strateegia mõistlikuim, sest lõhkuvaid muudatusi on mõnevõrra vähem ja samm haaval uuendamine ainult pikendaks uuendamisprotsessi.

Teine valik on uuendus läbi viia samm haaval ehk uuendada järk-järgult uuemale versioonile. Sedasi on lõhkuvate muudatuste kogus sammudes väiksem ja võib osutuda arendajale mõistlikumaks strateegiaks [21]. Selline strateegia võib olla sobilikum .NET Frameworkilt .NET 6-le uuendamisel.

Kolmas strateegia on uuendada .NET Standard 2.0 peale ja sealt edasi saab vajadusel uuendada .NET 6 peale.

Kui rakendusele on kirjutatud testid, siis tuleks uuendamist alustada testide projektist. See võimaldab teise ja kolmanda strateegia puhul rakendust testida vahesammudel. Esimese strateegia puhul pole testide projektis alustamine nii oluline ja testide projekti võib uuendada viimasena.

3 Uuendamist abistavate tööriistade kasutamine ja analüüs

Raamistiku uuendamine uuele versioonile nõuab planeerimist ja palju korduvtegevust. Uuendades .NET Frameworkilt .NET Core'ile on projektifaile ühtemoodi vaja viia uuele struktuurile. Alati on uuendamisel vaja ära vahetada projektifailides sihitavad raamistiku ja NuGet pakettide versioonid.

Planeerimise ja korduvmuudatuste tegemiseks, mis on paljudel projektidel ühtemoodi, on loodud mitmeid abistavaid tööriistu. Tuntuimad tööriistad on .NET *Portability Analyzer, Try-convert* töörist ja .NET *Upgrade Assistant*.

Järgnevaga analüüsitakse tööriistade võimekust ja puudujääke ning antakse sisend lõputöö praktilisele osale.

3.1 .NET *Portability Analyzer*

.NET *Portability Analyzer* on tööriist, mis annab arendajale ülevaate raamistiku uuendamise keerukusest. Tööriist toob välja rakenduse sõltuvused, mida on vaja muuta, et uuendada sihitud raamistiku versioonile [22].

Tööriist on laiendusena saadaval ainult Visual Studio 2017 ja 2019 IDEdes, kuid on võimalik kasutada ka *ApiPort* konsooliaplikatsiooni. *ApiPort* võimaldab teha kõike, mida suudab tööriist, kuid lisaks on võimalik tekitada pilt rakenduse sõltuvustest. [23]

Laiendus lubab analüüsida igat rakenduses olevat projekti ja klassiteeki eraldi. Kasutaja saab valida ühe või mitu raamistiku versiooni, mida sihtida ning tulemuses antakse ülevaade iga versiooni kohta.

Tulemuses, mida saab näiteks genereerida Excel failina, on toodud kokkuvõte, detailid pakettide toetatavuse kohta ja paketid, mida tööriist ei suutnud töödelda.

Kokkuvõttest (vt Joonis 5) on näha kõik projektid ja millisel versioonil nad tööriista käitamise hetkel on. Seadistusse märkis autor, et soovib analüüsi .NET Core 3.1, .NET 5 ja .NET Standard 2.0 jaoks. Hetkel valikut .NET 6 jaoks pole, kuid vaadates minimaalset

erinevust .NET Core 3.1 ja .NET 5 vahel, siis saab järeldada, et ka .NET 6-le migreerimise protsendid on väga sarnased.

Assembly Name	Target Framework	.NET Core,Version=v3.1	.NET Standard,Version=v2.0	.NET,Version=v5.0
Common, Version=1.0.0.0,	.NETFramework,Version=v4.8	92,26	89,75	92,26
DAL, Version=1.0.0.0, Cultu	.NETFramework,Version=v4.8	99,27	98,72	99,27
Service, Version=1.0.0.0, Cu	.NETFramework,Version=v4.8	98,91	98,26	98,57
WebApp, Version=1.0.0.0, (	.NETFramework,Version=v4.8	93,78	92,84	93,37

Joonis 5. .NET Portability Analyzeri kokkuvõte.

Tabelis olevad protsendid näitavad kui paljud projektis kasutusel olevad API-d on saadaval sihitud raamistiku versioonil [23]. Samas tuleb arvestada, et kuigi ühilduvusprotsendid on suured, siis on suur ka projektis olevate API-de arv. See tähendab, et ka suurte ühilduvus protsentide korral võib mitteühilduvaid APIsid olla arvuliselt palju.

Detailide lahtris (vt Joonis 6) on toodud kõik APId, mida pole sihitud raamistikes saadaval. Kuna .NET 5 on viimane versioon enne .NET 6, siis jooksutas autor tööriista uuesti valides seadistustest sihitud raamistikuks ainult .NET 5.

Target type	Target member	Assembly Name	.NET,Version=v5.0
T:Newtonsoft.Json.JsonSerializerSettings	T:Newtonsoft.Json.JsonSerializerSettings	Common	Not supported
T:Newtonsoft.Json.JsonSerializerSettings	T:Newtonsoft.Json.JsonSerializerSettings	Service	Not supported
T:Newtonsoft.Json.JsonSerializerSettings	T:Newtonsoft.Json.JsonSerializerSettings	WebApp	Not supported
T:Newtonsoft.Json.JsonSerializerSettings	M:Newtonsoft.Json.JsonSerializerSettings.#ctor	Service	Not supported

Joonis 6. Osa detailide lahtrist.

On näha, et tööriist annab hea ülevaate, milliste APIde välja vahetamisele tuleks mõelda. Miinuseks saab tuua koondnimekirja puudumise. Hetkel kuvatakse iga API kasutus igas projektis eraldi real ja sedasi võib detailide lahter olla mitusada rida suur. Arendajale oleks abiks selline koondnimekiri pakettidest, kus pole dupliaktsiooni, sest nii ei pea tervet faili läbi vaatama, vaid saab ülevaate palju vähema ridade arvuga.

3.2 .NET *Upgrade Assistant*

Upgrade Assistant on edasiarendus Try-convert tööriistast [24]. See tähendab, et Try-convert tööriista eraldi antud lõputöös ei käsitleta. Samuti on hea ära märkida, et tööriist töötab ainult Windows operatsioonisüsteemiga arvutitel.

Rakendusetüübid, mida *Upgrade Assistant* toetab, on ASP.NET, *Windows Forms*, *Windows Presentation Foundation*, konsoolirakendus ja *Class Library* [25].

Kui .NET *Portability Analyzer* ei muutunud projekti failides ühtegi rida koodi, siis .NET *Upgrade Assistant* teeb seda väga suures koguses. *Upgrade Assistant* on Microsofti poolt loodud käsurea tööriist, mis vähendab oluliselt arendaja tööd raamistiku uuendamisel. Käsureale hakkavad pärast käivitamist jooksma informatsioon tööriista tegevuste kohta ja stopp-start süsteemi sammud (vt Joonis 7), läbi mille saab arendaja juhtida kogu uuendamisprotsessi.

```
Choose a command:
   1. Apply next step (Select an entrypoint)
   2. Skip next step (Select an entrypoint)
   3. See more step details
   4. Configure logging
   5. Exit
>
```

Joonis 7. Stopp-start süsteemi samm.

Pärast igat stopp-start süsteemi sammu on arendajal võimalik uuendus nii-öelda pausile panna ja vaadata, mida tööriist projektis muutnud on. Enamus muudatustest logitakse konsooli.

Tööriistal on 2 põhifunktsionaalsust: teha analüüs projektis olevate sõltuvuste kohta ja viia läbi koodimuudatustega uuendus sihitud raamistikule. Mõlemal funktsionaalsusel on käivitamiseks oma käsk: *upgrade assistant analyze* ja *upgrade assistant upgrade*.

3.2.1 Analüüsi võimekus

Upgrade Assistanti käsuga *analyze* koostatakse fail SARIF formaadis, mis põhineb JSON-il. Seda saab vaadata kõigi tekstiredaktoritega, kuid soovitatakse kasutada laiendust *Visual Studios* (vt Joonis 8). [26]

	Code	Description	Project	File
⚠	UA103	Reference to System.Web.WebPages.Deployment needs to be deleted.		WebApp.csproj
⚠	UA103	Reference to System.Web.Extensions needs to be deleted.		WebApp.csproj
⚠	UA106	Package Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers, Version=0.3.310801 needs to be added.		WebApp.csproj
⚠	UA103	Reference to System.Web.Routing needs to be deleted.		WebApp.csproj
⚠	UA103	Reference to System.Web.Mvc needs to be deleted.		WebApp.csproj

Joonis 8. Osa SARIFi vaatamise laienduse väljakuvast.

Tulemuses on välja toodud muudatused, mida on vaja uuendamise käigus teha. „*Code*“ tulbas on link, mis peaks viima muudatust selgitavale lehele, kuid hetkel enamik linkidest viib hoopis *Upgrade Assistanti* lehele ja sellest suurt abi pole.

Upgrade Assistanti analüüsi tulemus on teisttüüpi kui .NET *Portability Analyzer* tulemus. Kuvatakse reaalsed muudatused, mida on vaja läbi viia, kuid ei kuvata, millised paketid pole sihitud versioonis toetatud.

Samuti ei anna *Upgrade Assistanti* analüüsi tulemus nõu uuendamise teostatavuse osas ja eeldatakse, et arendaja on selle eelnevalt endale selgeks teinud, näiteks kasutades .NET *Portability Analyzer* tööriista [26].

3.2.2 Uuendamise võimekus

Upgrade Assistanti käsuga *upgrade* tehakse järgmised toimingud:

1. Määratakse parim projektide uuendamise järjekord [26]. Tööriist määrab järjekorra sõltuvuste arvu järgi projektide vahel, kuid lõpp kokkuvõttes valib uuendamise järjekorra tööriista kasutaja.

2. Uuendades .NET Frameworkilt .NET Core’ile, muudab tööriist projektifailid uuele kujule [26]. Samuti kaotatakse *packages.config* fail ja lisatakse kõik NuGet sõltuvused projektifailidesse kasutades eelmises peatükis kirjeldatud *PackageReference* sõlme.

3. Projektifailides muudetakse ära sihitud raamistiku versioon vastavalt tööriista käivitamisel kaasa antud parameetrile. Muudetakse ära NuGet pakettide versioonid, millele leitakse uue raamistiku versiooniga ühilduv versioon [26].

4. Muudetakse mõnes kohas C# lähtekoodi, kus on selge, milliseks see kood muuta tuleb [26]. Muid keeli peale C# (näiteks F#) hetkel ei muudeta.

5. .NET Frameworkilt .NET Core’ile uuendades lisatakse projekti failid, mida uuel versioonil vaja on. Sellised failid näiteks *Progam.cs*, *Startup.cs* ja *appSettings.json* [26]. Lisatud failid täidetakse lihtsate vaikeväärtustega ja kogu varasem konfiguratsioon, mis asus *Global.asax* failis tuleb arendajal endal ümber kirjutada.

Paremaks visuaalseks ülevaateks tööriista võimekusest koostab autor tabeli (vt Tabel 1).

Tabel 1. .NET *Upgrade Assistanti* võimekus

Tegevus	**Tööriista võimekus**	**Arendaja käsitöö**
Projektifaili kuju muutmine	Suudab täielikult konverteerida .NET Framework projektifaili uuele .NET Core kujule	Puudub
NuGet pakettidega seotud tegevused	Vajadusel ja võimalusel uuendab, lisab ja eemaldab NuGet pakette, *packages.config* faili sisu teisaldatakse projektifaili	Osa NuGet pakette tuleb uuendada või välja vahetada iseseisvalt sõltuvalt rakenduse eripäradest
Vajalike failide lisamine, rakenduse konfiguratsioon	Lisab rakendusse nii *Program.cs*, *Startup.cs* kui ka *appSettings.json* faili vaikimisi väärtustega kui neid varem polnud	Nimetatud failid tuleb ise täita korrektse rakenduse konfiguratsiooniga, muuhulgas näiteks marsruutimise ja sõltuvussüstimisega
Uuendamine erinevatele .NET versioonidele	Hetkel on võimalus uuendada ainult .NET 6-le, kusjuures võib kohata ootamatut käitumist *Class Libraryte* uuendamisel	Muudele versioonidele uuendada soovides on tööriist kasutamatu. *Class Libraryd* peab mõnel juhul käsitsi uuendama.
Uuendusjärgne koristus	Puudub	Ebavajalike NuGet pakettide eemaldamine, ebavajalike täpsustite eemaldamine

Pärast tööriista uuendamisprotsessi läbisaamist on väga väike võimalus, et projektikood kompileerub. Tihti on vaja teha veel käsitööd osades kohtades koodi parandamiseks ning uute NuGet pakettide kasutusele võtmiseks.

3.2.3 Parameetrid

Nii *upgrade* kui ka *analyze* käsuga saab kaasa anda parameetreid, mis ühel või teisel viisil muudavad tööriista käitumist. Esimene nendest on sihitava raamistiku parameeter. Parameetri saab väärtustada kolme väärtusega: LTS, *Current* ja *Preview*. 03.04.2022 seisuga viitavad kõik parameetri väärtused .NET 6-le. See tähendab, et rakendust saab *Upgrade Assistanti* kasutades uuendada ainult .NET 6-le. Lähitulevikus peaks hakkama *Preview* viitama .NET 7 *Preview* versioonile.

Lisaks saab konsoolirakendust käitades kaasa anda parameetri sisenemispunkti täpsustamiseks, projekti varukoopia mitte tegemiseks ja stopp-start süsteemi maha keeramiseks [26].

Nii *upgrade* kui ka *analyze* käsule saab kaasa anda --*extension* parameetri, läbi mille saab arendaja käivitada enda või teiste loodud laiendusi. Laiendusetele saab omakorda anda kaasa parameetreid kasutades --*options* parameetrit.

3.2.4 Laiendatavus

Upgrade Assistantil on sisse ehitatud laienduste süsteem. Laiendustega on võimalik kohandada tööriista käitumist või lisada uusi samme ilma, et peaks tööriista ümber ehitama [27].

Laiendus peab sisaldama *ExtensionManifest.json* faili ja võib olla saadaval mitmes formaadis: ainult *ExtensionManifest.json* fail, kataloog või zip fail *ExtensionManifest.json* failiga ja NuGet paketina [27].

3.2.5 Puudused

Suureks nõrkuseks *Upgrade Assistanti* juures on *Class Libraryte* sunnitud uuendamine .NET Standard 2.0 peale, mitte sellele versioonile, mis rakenduse käivitamisel valiti.

Upgrade Assistanti arendustiim põhjendab sellist käitumist asjaoluga, et tihti kasutatakse ühte *Class Libraryt* mitmes projektis korraga [28]. Sihtides .NET Standardit oleks sama *Class Library* kasutatav nii .NET Framework 4.8 peal kui ka näiteks .NET 6 peal.

See aga ei vasta tõele näiteks selliste projektide puhul, kus tehakse klientidele kohandatud lahendusi suures spektris ja *Class Library* korduvkasutus on pigem harv nähtus. Samuti sai peatükis 1 välja toodud, et .NET 6 sihtides saab *Class Librarytes* kasutada kõiki uusi funktsionaalsuseid ning uusi C# versioone, kuid .NET Standardit sihtides seda teha ei saa.

Sellest tulenevalt võib arendustiimil olla soov uuendada ka *Class Libraryd* .NET 6-le. Lisaks oleks sedasi rakendus ühtsem- kõikides projektides kasutatakse samu C# versioone ja ei esine olukorda, kus mõnda süntaksi ootamatult ei eksisteeri.

Lisaks saab puuduseks tuua hetkese seisu sihitavate versioonidega. Kuna kõik 3 sihitava raamistiku parameetri väärtust viitavad .NET 6-le, siis ei saa näiteks arendaja uuendada

kasutades samm haavalist raamistiku uuendamise strateegiat, mis on Microsofti poolt üks väljatoodud võimalustest ja millest oli juttu peatükis 2.4.

Mõnelt varasemalt .NET Core versioonilt liikudes .NET 6 peale võib kohata ootamatut käitumist, kus uuendatakse käivitatav projekt .NET 6-le, kuid *Class Libraryd* jäävad esialgsele versioonile. *Upgrade Assistant* logib olukorra seletuseks, et tulenevalt mõnedest NuGet paketi sõltuvustest otsustati osasid projekte mitte uuendada. Tegemist on väga ettenägematu käitumisega, eriti juhul kui arendaja on varasemalt analüüsides veendunud, et rakendust on teoreetiliselt võimalik uuendada uuele versioonilt või leidnud endale asenduseks uued NuGet paketid. Sellist tööriista käitumist pole võimalik vältida ja teiste projektide uuendus tuleb läbi viia käsitsi.

Upgrade Assistant lisab oma töö käigus igale rakenduses olevale projektile analüüsimisega seotud NuGet paketi, mida pole võimalik tööriista kasutades eemaldada. See omakorda tähendab, et arendaja peab ise tööriista järgi koristama hakkama, suurendades käsitsi tehtavat tööd.

Samuti on olulised puudused *Upgrade Assistanti* toetatavustes. Tööriista saab ainult kasutada Windows operatsioonisüsteemil ja tööriista ei saa kasutada .NET projektide uuendamiseks, kus on kasutusel F#.

4 Realisatsioon

Järgnevaga kirjeldatakse loodavad laiendused, kirjeldatakse sügavamalt *Upgrade Assistanti* laienduste süsteemi ja programmeeritakse need laiendused. Seejärel publitseeritakse loodud laiendused NuGet paketina kõigile kasutamiseks. Viimasena kasutatakse loodud laiendusi kahe projekti uuendamise läbiviimiseks.

Arendatakse .NET *Upgrade Assistant* tööriistale 2 laiendust. Esimene laiendus *UpgradeAssistant.Extension.ForceTFM* sunnib *Upgrade Assistant* tööriista uuendama kõik rakenduses olevad projektid alati valitud versioonile. Teine laiendus *UpgradeAssistant.Extension.CleanUp* koristab tööriista poolt lisatud ebavajalikud NuGet paketid tervest rakendusest vähendades seeläbi käsitsi tehtavat tööd.

4.1 Ühele versioonile sundimise laiendus

Eelnevalt välja toodud *Upgrade Assistanti* nõrkustest peab autor suurimateks *Class Libraryte* sunnitud uuendamise .NET Standard 2.0 peale ja .NET Core versioonide vahel liikudes *Class Libraryte* üldse mitte uuendamine olukordades, kus tööriist ei tuvasta mõnda NuGet paketi uut versiooni. Seda sellepärast, et uuendades .NET 6 peale on tihti eesmärk saada uusi funktsionaalsuseid kogu rakenduses mitte ainult käivitatavas projektis.

Esimese laienduse eesmärk saab olema tööriista käitumise muutmine vähem jäigaks, andes arendajale valiku sundida terve rakendus samale .NET versioonile ilma, et tööriist omavoliliselt muudaks arendaja poolt valitud sihitud raamistikku. Autoril endal on vajadus sellise funktsionaalsuse üle ja vajadus on ka mitmel teisel arendajal, kes on oma muret kurtnud *Upgrade Assistanti* GitHubi repositooriumi foorumites.

4.1.1 Analüüs

Laienduse programmeerimine hakkab laienduste dokumentatsiooni põhjalikust läbi lugemisest. Selgeks saab kohe, et rakendusetüüp laiendusel peaks olema *Class* Library, sest rakendus pole käivitatav. Sihitavaks raamistikuks loodavas laienduses saab olema .NET 6, sest *Upgrade Assistant* ise on ehitatud .NET 5 ja .NET 6 peale ning kuna tegu

pole paketiga, mis on nii-öelda iseseisev, siis ei tule laienduse kättesaadavuse ulatuses suurenemist kui sihtida .NET Standardit.

Ühes laienduses olema *ExtensionManifest.json* fail, kus saab määrata laienduse nime ja laiendusest tulenevalt muud olulist. *ExtensionManifest.json* võib jääda täiesti tühjaks, kuid ta kindlasti peab olema olemas [27].

ExtensionManifest.json faili tuleb käesoleva laienduse puhul lisada *„ExtensionServiceProviders“* väli, kus täpsustatakse kompileeritud koodi asukoht, kust *Upgrade Assistant* peaks otsima teostusi *IExtensionServiceProvider* liidesest. *IExtensionServiceProviderit* teostavates klassides lisatakse sõltuvussüstimise konteinerisse vajalikud teenused ja kasutajavalikud. Selleks, et *Upgrade Assistanti* liideseid kasutada saaks tuleb laiendusele lisada *Microsoft.DotNet.UpgradeAssistant.Abstractions* NuGet pakett [27].

Upgrade Assistanti lähtekoodist on näha, et on olemas liides *ITargetFrameworkSelector*, kus on defineeritud üks meetod *SelectTargetFrameworkAsync*. Just seda liidest implementeerides on võimalik kujundada tööriista raamistiku sihtimise käitumist.

Lähtekoodis olev teostus toimib sedasi, et leitakse käivitatava projekti, näiteks ASP.NET veebirakenduse, hetkel kasutusel olev versioon ja vastavalt hetkel uuendamisel oleva projekti tüübile valitakse sihitav raamistik. Valikut juhitakse läbi selektorite, kust on loogika, millisel juhul peaks sihtima arendaja poolt valitud raamistikku ja millisel juhul peaks sihtima .NET Standard 2.0 [29].

Laienduse põhieesmärgiks saab olema uue *ITargetFrameworkSelector* teostuse loomine, mis asendab lähtekoodis oleva teostuse. *ITargetFrameworkSelector* on üks sellistest liidestest, millele saab laienduse käigus uut teostust registreerida läbi sõltuvussüstimise.

Lahenduses tuleb eemaldada selektorid, mille eesmärk on sihtimine suunata mõnele muule raamistikule, ja sedasi hakkab tööriist alati sihtima valitud raamistikku.

Laienduses sunnitud raamistiku versiooni valik peaks käituma sarnaselt *Upgrade Assistanti* raamistiku versiooni valikule. Valikus on *Current*, LTS ja *Preview* ning kõik nad viitavad hetkel .NET 6-le. Lisades laiendus *Upgrade Assistantile* külge, muutub *Upgrade Assistanti* enda sihitava raamistiku valik kasutuks, sest laienduse valik on see,

mida arvestatakse. Põhjus, miks ei saa kasutada *Upgrade Assistanti* enda sihitava raamistiku valikut, on selles, et sellele valikule ei saa läbi laienduse ligi.

Lisaks võiks olla arendajatel võimalus muuta *Current*, LTS ja *Preview* väärtust nii nagu arendaja enda soov on. Sedasi saab arendaja valida ka muid versioone kui .NET 6 ning kui autor ei jõua tulevikus õigel ajal *Current*, LTS ja *Preview* väärtust muuta, siis arendajad saavad seda ajutiselt ise teha.

4.1.2 Laienduse programmeerimine

Autor alustab Giti koodihoidla loomisest (vt Lisa 2), kuhu ta lisab esialgse *README.md* faili, *.gitignore* faili ja MIT litsentsi.

Upgrade Assistantil ja Microsofti poolt loodud laiendustel on litsentsiks määratud MIT litsents. Autor soovib hoida ühtlast joont olemasolevate laiendustega ja kuna MIT litsentsi sisu on autorile sobiv, siis loodav laiendus saab omale MIT litsentsi. Litsentsis on muuhulgas märgitud, et autor lubab laiendust tasuta kasutada, muuta ja jagada.

Seejärel loob autor *Class Library*, mis on sihib .NET 6. Loodud *Class Library* on laienduse põhiosa, kus eksisteerib kogu kood ja millest pärast laienduse valmimist tehakse NuGet pakett. Laiendus saab nimeks *UpgradeAssistant.Extension.ForceTFM*, kus nime esimene ja teine osa annavad teada, et tegu on *Upgrade Assistanti* laiendusega ja kolmas osa viitab laienduse tegevusele- sunnitakse kogule rakendusele sama sihitud raamistik.

Järgmiseks luuakse *ExtensionManifest.json* fail, kuhu lisatakse laienduse nimi, milleks saab olema *ForceTFM* ehk *Force Target Framework*, ja viide *Class Library* kompileeritud koodile, kus hakkab olema *IExtensionServiceProvider* teostus (vt Joonis 9).

```
{
  "ExtensionName": "ForceTFM",

  "ExtensionServiceProviders": [
    "UpgradeAssistant.Extension.ForceTFM.dll"
  ],

  "ForceTFM": {
    "TFM": "LTS"
  },

  "TFMOptionMap": {
    "Current": ".net6.0",
    "Preview": ".net6.0",
    "LTS": ".net6.0"
  }
}
```

Joonis 9. *ForceTFM* laienduse ExtensionManifest.json fail.

Järgmiseks on vaja luua *ITargetFrameworkSelectori* teostus ja selleks on tarvis luua võimalus kasutajal valida, millist raamistiku versiooni sihtida. Autor loob selleks kaks abistavat klassi, kus ühte talletatakse informatsioon kasutaja valiku kohta ja teise kaardistatakse *Current*, LTS ja *Preview* väärtused.

Mõlemad klassid saavad oma klassimuutujatele väärtused *ExtensionManifest.json* failist (vt Joonis 9), *ForceTFM* - kasutaja valik ja *TFMOptionMap*- sihitavad versioonid. Vaikimisi on sihitava raamistiku väärtus LTS.

Selleks, et *ExtensionManifest.json* failist väärtused jõuaksid klassidesse, tuleb klassid lisada sõltuvussüstimise konteinerisse (vt Joonis 10). Tuleb jälgida, et *ExtensionManifest.json* failis parameetrite nimed kattuksid klassimuutujate nimedega.

```
services.AddExtensionOption<TfmOption>( sectionName: "ForceTFM");
services.AddExtensionOption<TfmOptionMap>( sectionName: "TFMOptionMap");
```

Joonis 10. Klasside sõltuvussüstimise konteinerisse lisamine.

Seejärel on võimalik väärtustatud klassid süstida *ITargetFrameworkSelectori* teostusse. Teostuses valideeritakse, et sihitud raamistiku väärtus oleks korrektne ja parsitakse sõne *TargetFrameworkMoniker* tüüpi objektiks, millest saab meetodi tagastusväärtus.

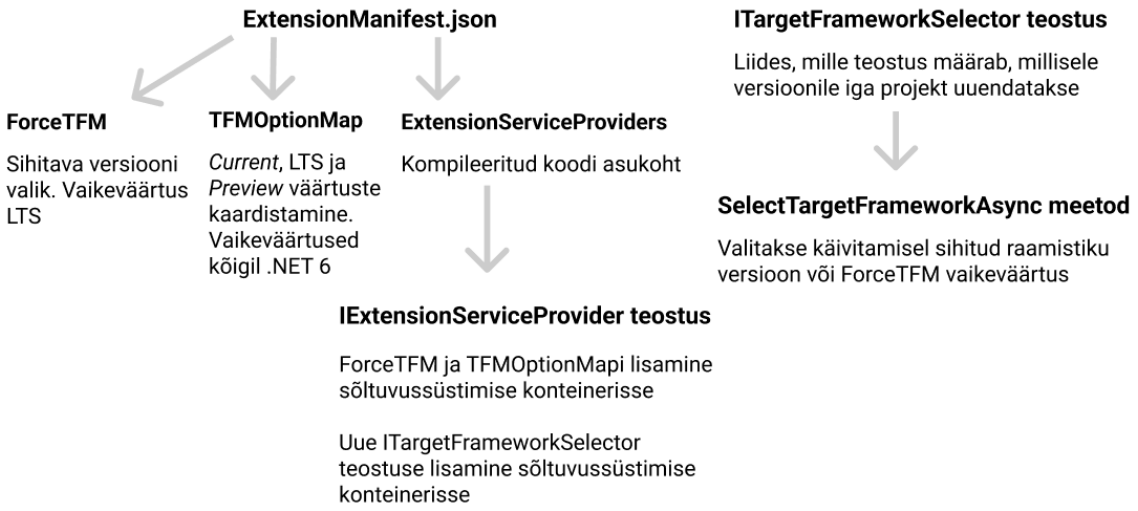
Tulenevalt *Upgrade Assistanti* laienduste süsteemist, on kõiki *ExtensionManifest.json* failis olevaid väärtuseid muuta kui need lisatakse sõltuvussüstimise konteinerisse (vt Joonis 10). See tähendab, et kasutajale on eelnevaga loodud võimalus muuta nii *ForceTFM* väärtust kui ka *TFMOptionMapis* olevaid väärtuseid.

Nüüd on võimalik *Upgrade Assistanti* kasutada koos laiendusega kasutades konsoolis järgnevat käsku: *upgrade assistant upgrade* [tee rakenduse .sln failile] *--extension* [tee laienduse kaustani, mis sisaldab *ExtensionManifest.json* faili ja kompileeritud koodi]. Kasutaja saab *ForceTFM* ja *TFMOptionsMapi* muuta lisades käsule*:* *--option „ForceTFM:TFM*=[väärtus]“ või *--option „TFMOptionsMap:*[pointer]=[väärtus]“.

Näiteks:

1. --*option „ForceTFM:TFM*=Current“
2. --*option „TFMOptionsMap:*Current=.net5.0“.

Laienduse arhitektuuri selgemaks mõistmiseks koostas autor joonise (vt Joonis 11).



Joonis 11. ForceTFM laienduse arhitektuur.

4.1.3 Laienduse publitseerimine

Kuigi laiendust saab edukalt kasutada kloonides Gitist kood oma arvutisse, siis mugavamaks kasutamiseks ja paremaks leitavuseks saab laienduse avaldada NuGet paketina.

NuGet paketi loomiseks tuleb täiendada projektifaili erinevate sõlmedega. Sõlmedesse saab talletada loodava paketi pealkirja, lühikirjelduse, Giti koodihoidla ja litsentsile viitamise lingid ning NuGeti veebilehel paketi otsingu hõlbustamiseks saab määrata sildid. Lisaks toimub läbi *Version* sõlme paketi versioneerimine. Vaikimisi on paketi versioon 1.0.0, kuid koodis muudatusi tehes peab määrama paketile uue versiooni, sest kahte sama versiooniga paketti publitseerida ei saa.

NuGet paketi nimeks saab sama, mis on loodud laienduse *Class Library* nimi ja litsents jääb samuti MIT litsentsiks. Sildiks määrab autor .NET *Upgrade Assistant* ning NuGet lõhub määratud sildi tühikute juurest kolmeks. Sellise sildiga kuvatakse *ForceTFM* laiendust otsingutulemustes kui otsitakse siltide järgi *Upgrade Assistant* tööriista ennast.

Laiendust saab nüüd pakkida .nupkg failiks kasutades *dotnet pack* käsku käsureal ning seejärel saab .nupkg faili publitseerida *dotnet nuget push* käsuga.

Esimesel pakkimisel tuleb ilmsiks, et JSON faile niisama .nupkg faili ei pakita. *ExtensionManifest.json* on fail, mis peab ühes *Upgrade Assistanti* laiendavas NuGet paketis olema ja et seda kaasa pakitaks, tuleb projektifailis seda määrata kasutades *Content* sõlme.

Paketi avaldamisel NuGet keskkonda ja seejärel seda kasutada üritades selgub, et *ExtensionManifest.json* faili ei leita, kuigi kontrollides .nupkg faili sisu, on ta *content* kaustas olemas. Olukorrast tulenevalt võrdles autor Microsofti poolt loodud NuGet pakettide failistruktuuri *ForceTFM* paketi failistruktuuriga ja avastas, et *Upgrade Assistanti* laiendustel eeldatakse, et nii *ExtensionManifest.json* kui ka kompileeritud kood asuvad .nupkg juurkaustas.

Vaikimisi pannakse kompileeritud kood .nupkg faili *lib* kausta ja *Content* sõlmedega kaasatud failid *content* kausta ning kuna *Upgrade Assistanti* laienduste dokumentatsioon ei anna ülevaadet, milline on oodatud laienduse .nupkg faili struktuur, siis autor väidab,

et antud dokumentatsioon on puudulik. Lisaks pole dokumentatsioonis kirjeldatud, kuidas failistruktuur õigele kujule viia.

Failistruktuuri õigele kujule saamiseks leidis autor ainukeseks lahenduseks kohandatud .nuspec faili loomise. Tegu on konfiguratsioonifailiga, mis on XML fail ja asub .nupkg *zip* faili sees. Luues .nuspec fail käsitsi tuleb sinna ise lisada sama informatsioon, mis sai lisatud projektifaili ja täiendada sisu *files* sõlmega, kus saab konfigureerida paketi failistruktuuri. Seadistades failide asukohti võtab autor eeskuju Microsofti laienduste struktuurist ja suunab neli faili paketi juurkausta (vt Joonis 11).

```
<files>
    <file src="UpgradeAssistant.Extension.ForceTFM\bin\Debug\.net6.0\*.dll" target="" />
    <file src="UpgradeAssistant.Extension.ForceTFM\bin\Debug\.net6.0\*.pdb" target="" />
    <file src="UpgradeAssistant.Extension.ForceTFM\bin\Debug\.net6.0\*.deps.json" target="" />
    <file src="UpgradeAssistant.Extension.ForceTFM\bin\Debug\.net6.0\ExtensionManifest.json" target="" />
</files>
```

Joonis 12. Failistruktuuri konfiguratsioon.

Tulemuseks on NuGet pakett, mille struktuur ja kasutus on Microsofti enda laiendustega samasugune [30].

4.2 Uuendusjärgse koristuse laiendus

Teise laienduse eesmärk saab olema uuendamisjärgse koristuse läbi viimine. Peamiseks ülesandeks on *Upgrade Assistanti* poolt lisatud NuGet pakettide eemaldamine rakendusest, mida muidu peaks tegema käsitsi. Selline laiendus on hea, sest sedasi on võimalik, et .NET Core versioonide vahel liikudes pole arendajal vaja ühtegi faili ise käsitsi muuta- kogu uuendamine ja koristamine toimub automatiseeritult.

4.2.1 Analüüs

Koristamise teostuseks tuleb *Upgrade Assistanti* uuendussammude hulka lisada uus samm. Uut sammu saab lisada teostades *UpgradeStep* liidest ja seejärel lisades teostus sõltuvussüstimise konteinerisse.

Käesolev laiendus hakkab toimima sedasi nagu *ForceTFM* laiendus, kus tööriista käitumine muutub kohe kui laiendus tööriista külge lisada. Laiendus on mõeldud

kasutamiseks pärast seda kui uuendusprotsess on läbi viidud ja seda tuuakse välja ka laienduse dokumentatsioonis.

Kui uuendamisprotsess on läbi viidud, siis tööriista uuel käivitamisel sisse ehitatud uuendussamme enam ei läbita ja käesoleva laienduse uuendusamm saab olema ainuke samm mida läbitakse.

UpgradeStep liidese teostuses on tarvis teostada 3 meetodit. Ühes meetodis otsustatakse, kas sammu peaks käivitama, teises initsialiseeritakse uuendussamm ja kolmandas meetodis samm teostatakse.

4.2.2 Laienduse programmeerimine

Esimesed sammud programmeerimisel on sarnased ForceTFM laiendusega. Autor loob Giti koodihoidla (vt Lisa 3), *README.md* ja *.gitignore* failid ja MIT litsentsi, kus on määratud, et laiendust võib tasuta muuta, kasutada ja jagada. Seejärel luuakse .NET 6 *Class Library* ja laiendusele antakse nimi *UpgradeAssistant.Extension.CleanUp*, kus nime esimene ja teine osa annavad teada, et tegu on *Upgrade Assistanti* laiendusega ja kolmas osa viitab laienduse tegevusele- uuendusjärgne koristustöö.

Järgmiseks luuakse *ExtensionManifest.json* fail, kuhu lisatakse laienduse nimi, milleks saab olema *CleanUp* ja viide *Class Library* kompileeritud koodile, kus hakkab olema *IExtensionServiceProvider* liidese teostus.

Seejärel luuakse *IExtensionServiceProvider* ja *UpgradeStep* liideste teostused ja uus uuendussamm lisatakse sõltuvussüstimise konteinerisse.

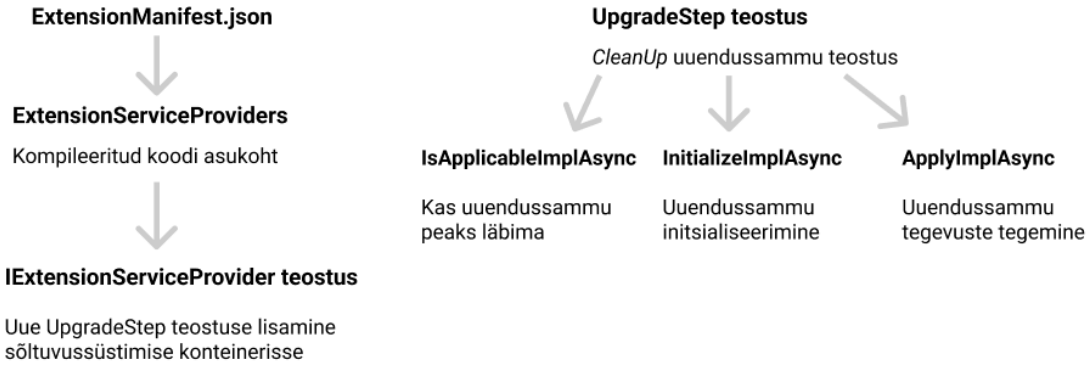
Kui tavaliselt peab igat uuendamissammu läbima iga rakenduses olevas projektis eraldi, siis antud olukorras pole see mõistlik, sest laiendus koodi ei lõhu ja laiendus saab edukalt oma töö ära teha kõikide projektide peal ilma, et peaks stopp-start süsteemis igat projekti eraldi ette võtma.

Uue uuendussammu teostuse põhiosa on *ApplyImplAsync* meetodis, kus itereeritakse üle kõikide projektide. Kuigi NuGet pakette saab eemaldada kasutades *IProjectFile* liidese meetodit *RemovePackages*, siis see pole hetkel parim lahendus, sest eemaldataval paketil tuleb määrata kindel versioon ja sedasi uue paketi versiooni välja andmise puhul seda

paketti enam ei eemaldata. Paremaks lahenduseks on projektifailist kindlad paketid välja otsida paketi nime järgi ja seejärel need eemaldada.

Selleks on tarvis *Microsoft.Build* NuGet paketti, läbi mille saab projektifailis olevatele XML sõlmedele ligi. Seejärel saab leida kõik *PackageReference* sõlmed ja nende hulgast välja filtreerida välja need, mida on tarvis eemaldada.

Laienduse arhitektuuri selgemaks mõistmiseks koostas autor joonise (vt Joonis 13).



Joonis 13. CleanUp laienduse arhitektuur.

4.2.3 Laienduse publitseerimine

Erisusi *ForceTFM* laienduse publitseerimisega pole. .nuspec failis määratakse paketi nimi, versioon, litsents, Giti koodihoidla ja sildid ning koostatakse vajalik failistruktuur. Tulemuseks on NuGet pakett, mille struktuur ja kasutus on Microsofti enda laiendustega samasugune [31].

4.3 Laienduste kasutamine

Järgnevaga valideeritakse laienduste toimimist. Laiendusi kasutades uuendatakse kaks projekti .NET 6-le: üks projekt on enne uuendamist .NET 5 peal ja teine projekt .NET Framework 4.8 peal. Sedasi saab valideeritud nii .NET Core'ilt uuendamine kui ka .NET Frameworkilt uuendamine. Muudelt versioonidelt uuendamist ei demonstreerita, sest tööriist ja laiendus toimivad samamoodi sõltumata versioonist.

Mõlema uuendamise puhul tuleb lisada *ForceTFM* laiendus *Upgrade Assistanti* külge ja seejärel saab tööriista käivitada andes kaasa parameetreid tööriista enda jaoks kui ka

laienduse jaoks. Kuna *ForceTFM* laiendusele vaikimisi valib sihitud raamistikuks LTS, mis hetkel viitab .NET 6-le, siis tööriista käivitamisel pole laiendusega seotud parameetreid vaja kaasa anda.

4.3.1 .NET Framework 4.8 rakendus

Laienduste valideerimise eesmärgil genereerib autor ühe lihtsa .NET Framework 4.8 peal oleva ASP.NET MVC rakenduse kasutades JetBrains Rider IDE-t ja uuendab selle .NET 6 peale. Rakenduses on olemas kontroller, 3 Razoris kirjutatud lehte ja algeline konfiguratsioon, sealhulgas marsruutimine.

Rakendusse lisatakse 1 *Class Library* eesmärgiga kontrollida laienduse toimimist nii ASP.NET MVC projektis kui ka .NET Framework *Class Library* puhul.

.NET *Portability Analyzer* tööriist annab teada, et rakenduses olevatest API-de kasutustest on 73.7% üle viidavad .NET 5 peale. Detailide lahtrist on näha, kuidas kõik problemaatilised API-de kasutused on seotud *System.Web* API-ga, mis tuleb asendada käsitsi tänapäevasemate API-de kasutusega.

Autor käivitas *Upgrade Assistanti* ilma *ForceTFM* laienduseta ja tulemusena sihiti ASP.NET MVC projekt .NET 6 peale, kuid *Class Library* .NET Standard 2.0 peale. *Upgrade Assistanti* käivitades koos laiendusega sihitakse mõlemas projektis .NET 6.

Pärast tööriista uuendustsükli läbimist on tulemuseks katkine rakendus .NET 6 peal. *Upgrade Assistant* suutis ära teha suure osa tööd, kuid mõned probleemid tuleb lahendada käsitsi. Kõik probleemid on seotud varasemalt mainitud *System.Web* API-ga ja Razor lehtedes skriptide ja stiililehtede laadimisega.

Käsitsi tuleb ASP.NET Core projektist eemaldada *Global.asax* fail ja varasem rakenduse käivitamisega seotud kood, sest ASP.NET Core rakendustel on rakenduse konfiguratsioon *Startup.cs* failis. *Upgrade Assistant* on iseseisvalt lisanud *Startup.cs* ja *appSettings.json* failid ja konfigureerinud need vaikimisi väärtustega, millest praeguse lihtsa rakendusega piisab. Suurema ja keerulisema rakenduse puhul tuleb *Startup.cs* faili ümber kirjutada varasem rakenduse konfiguratsioon, mille hulka kuuluvad näiteks sõltuvussüstimine, marsruutimine ja autoriseerimine.

Skripte ja stiilifaile laetakse ASP.NET rakenduse Razor lehtedes kasutades *Styles.Render* ja *Scripts.Render* meetodeid. ASP.NET Core Razor lehtedes selliseid meetodeid pole ja uueks viisiks stiilifaile ja skripte laadida on kasutades *link* ja *script* sõlmesid.

Parandades koodivead on rakendus käivitatav ja kogu esialgne funktsionaalsus toimib. .NET Framework 4.8 rakendus on edukalt uuendatud .NET 6 peale (vt Lisa 5). Silmas peab pidama, et tegu oli laienduse valideerimisega ning suurema rakenduse uuendamine võib osutada väga mahukaks ja ajakulukaks sõltuvalt rakenduse eripäradest.

Viimasena lisatakse *Upgrade Assistanti* külge *CleanUp* laiendus ja jooksutatakse tööriista 1 kord veel. Pärast stopp-start süsteemi sammude läbimist on rakendusest eemaldatud kõik paketid, mis olid vajalikud *Upgrade Assistanti* tööks, kuid ebavajalikud rakenduse toimimiseks.

4.3.2 .NET 5 rakendus

Tegemist on rakendusega, kus on nii ASP.NET Core MVC kui ka ASP.NET Core API koondatud ühe projekti alla. Lisaks on rakenduses 16 *Class Libraryt* ja projekt testide jaoks.

.NET *Portability Analyzer* tööriista praegu kasutada ei saa, sest varasemalt selgitati, kuidas tööriist hetkel ei toeta .NET 6-le üleviimise analüüsimist ning asendusena sai analüüsida .NET 5-le üleviimist, sest .NET 5-lt .NET 6-le uuendades on lõhkuvate muudatuste arv väike. Sellest lähtudes eeldab autor rakenduse head ühilduvust .NET 6-ga.

Järgneb tüüpiline Upgrade Assistanti uuendustsükkel, kus võetakse iga projekt arendaja poolt valitud järjekorras ette ja uuendatakse sihitud raamistikule. Laiendusest tulenevalt sunnitakse igale projektile peale .NET 6. Kuna rakenduses on testide projekt, siis tuleb tööriista käivitada 2 korda ja teine kord valida uuendamiseks testide projekt.

Valideerimaks, et just laiendus sunnib sihitud raamistiku, käivitas autor uuendamata rakenduse peal *Upgrade Assistanti* ilma laienduseta ja tööriist oli võimeline 18-st projektist ainult ühe uuendama .NET 6 peale. Teiste projektide puhul arvas tööriist, et sõltuvused NuGet pakettidest ei luba uuendada .NET 6 peale.

Pärast tööriista töö lõppemist on kõik rakenduses olevad projektid .NET 6 peal. Rakenduse kood kompileerus ning rakenduse testid läbisid ilma käsitsi koodi muutmata. ASP.NET Core MVC käivitamisel hakkas rakendus tööle nagu varasemalt ning ASP.NET Core API-liides vastas päringutele korrektselt. Tulemuseks on töötav rakendus .NET 6 peal (vt Lisa 4).

Lõpetuseks lisatakse *Upgrade Assistanti* külge *CleanUp* laiendus ja jooksutatakse tööriista 1 kord veel. Pärast stopp-start süsteemi sammude läbimist on rakendusest eemaldatud kõik paketid, mis olid vajalikud *Upgrade Assistanti* tööks, kuid ebavajalikud rakenduse toimimiseks. Kogu uuendus on läbi viidud käsitsi koodi muutmata.

5 Tulemused

Lõputöö käigus arendati .NET *Upgrade Assistant* tööriistale 2 laiendust. Esimene laiendus *UpgradeAssistant.Extension.ForceTFM* sunnib *Upgrade Assistant* tööriista uuendama kõik rakenduses olevad projektid alati valitud raamistiku versioonile. Laiendus muudab tööriista käitumise läbinähtavamaks ja paindlikumaks- arendaja saab olla kindel, et tööriist sihib alati valitud .NET versiooni ja ei teki ootamatusi, kus mõni projekt on uuendatud .NET Standardile või pole üldse uuendatud. Teine laiendus *UpgradeAssistant.Extension.CleanUp* koristab tööriista poolt lisatud ebavajalikud NuGet paketid tervest rakendusest vähendades seeläbi käsitsi tehtavat tööd. Eelnevalt mainitust tulenevalt said lõputöö eesmärgid täidetud.

Arendatud laienduse toimimist valideerides sai uuendatud .NET 6 peale ASP.NET Core rakendus, kus on nii ASP.NET Core MVC kui ka ASP.NET Core API-liides koos 16 *Class Library* ja 1 testide projektiga. Uuendamist läbi viies polnud rakenduses tänu laiendustele vaja käsitsi ühtegi rida koodi muuta.

Arendatud laiendused publitseeriti NuGet paketina kõigile tasuta kasutamiseks. 15.05.2022 seisuga on *ForceTFM* laiendusel 693 allalaadimist, keskmine päevane allalaadimiste arv 19 [30]. 15.05.2022 seisuga on *CleanUp* laiendusel 35 allalaadimist, keskmine päevane allalaadimiste arv 5 [31].

Suurimaks keerukuseks laienduste arendamisel oli *Upgrade Assistanti* lähtekoodi lugemine ja mõistmine, milline sealne arhitektuur on. Arhitektuuri mõistma õppimine muutis laienduste programmeerimise lihtsamaks.

Lõputöö koostamise tulemusena puutus autor kokku mitmete tema jaoks uute tehnoloogiate, arendustegevuste ja probleemidega. Peamisteks keerukusteks arendustegevusel olid *Upgrade Assistanti* lähtekoodi uurimine leidmaks võimalust muuta tööriista käitumist ja NuGet paketi publitseerimine, kus tuli leida lahendus failistruktuuri ebakorrektsele kujule. Nii vanemate kui ka uuemate .NET raamistike tundmaõppimine on autorit tugevasti arendanud ja annab seejuures enesekindlust professionaalse tarkvaraarendajana töötamisel.

5.1 Võimalikud edasiarendused

UpgradeAssistant.Extension.ForceTFM laiendusele saaks juurde arendada võimaluse kasutajal valida iga rakenduses oleva projekti puhul sihitav raamistik. Sedasi saaks näiteks uuendada vajadusel osad *Class Libraryd* .NET Standardile ja ülejäänud .NET 6-le.

UpgradeAssistant.Extension.CleanUp laiendusele saaks juurde arendada rohkem koodi koristamist. Näiteks saaks arendada ebavajalike kvalifikaatorite eemaldamise selleks, et lõpptulemus oleks puhtam.

Edasiarendusena saab tuua ka uute laienduste loomise. *Upgrade Assistanti* analüüsis väljatoodud puudustest lähtudes oleks vajalik laiendus võimalus uuendada rakendus .NET Core 3.1-le vahesammuna.

6 Kokkuvõte

Käesoleva lõputöö on arendusuuring, mille eesmärgiks oli .NET *Upgrade Assistanti* töö parendamine muutes tööriista läbinähtavamaks ja paindlikumaks. Eesmärgi täitmiseks koostati ülevaade probleemist ja pakuti lahendus valitud kitsaskohtadele. Mittevalitud probleeme ei lahendatud ajaraamidest tulenevalt ja toodi välja võimalikud lisa- ja edasiarendused.

Lõputöö alguses anti ülevaade .NET ökosüsteemist, muutustest raamistike ja versioonide vahel ja uuendamise läbiviimise eripäradest. Toodi välja mõttekohad, millal uuendada ja millal mitte ning erinevad uuendamisestrateegiad.

Järgnevalt tutvustati ja analüüsiti uuendamist abistavaid tööriistu: .NET *Portability Analyzer*, *Try-convert* tööriist ja .NET *Upgrade Assistant*. Toodi välja võimekused, puudused ja laiendavatus ning anti sisend lõputöö praktilisele osale- kahe laienduse loomine. Esimene laiendus sunnib kõik rakenduses olevad projektid valitud raamistikule ja teine laiendus korraldab uuendamisprotsessi järgset koristustööd.

Praktilises osas kirjeldati loodavaid laiendusi ja uuriti, kuidas laiendusi programmeerida. Seejärel programmeeriti ja publitseeriti laiendused, kusjuures selgitati ette tulnud probleeme NuGet paketi loomisel.

Loodud NuGet paketi toimimist valideeriti uuendades 2 projekti .NET 6-le kasutades .NET *Upgrade Assistanti* koos laiendusega. Üks projekt oli enne uuendamist .NET 5 peal ja teine .NET Framework 4.8 peal.

Viimasena kirjeldati lõputöö edasiarendused ja tulemused, kus muuhulgas jõuti järeldusele, et lõputöö eesmärk on täidetud ja .NET *Upgrade Assistanti* käitumine on tänu loodud laiendustele läbinähtavam ja paindlikum.

Kasutatud kirjandus

[1] Microsoft, „Releases and support for .NET,“ Microsoft, 22 03 2022. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/dotnet/core/releases-and-support. [Kasutatud 07 04 2022].

[2] Microsoft, „.NET and .NET Core Support Policy,“ Microsoft, 2022. [Võrgumaterjal]. Loetud aadressil: https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core. [Kasutatud 07 04 2022].

[3] Stack Overflow, „2021 Developer Survey,“ Stack Overflow, 05 2021. [Võrgumaterjal]. Loetud aadressil: https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologie. [Kasutatud 15 04 2022].

[4] M. Megyesi, „Why Frameworks?,“ 8th Light, 12 09 2012. [Võrgumaterjal]. Loetud aadressil: https://8thlight.com/blog/myles-megyesi/2012/09/12/why-frameworks.html. [Kasutatud 20 04 2022].

[5] M. Milanovic, „A Brief Walk-Through of the .NET Ecosystem,“ DZone, 21 01 2021. [Võrgumaterjal]. Loetud aadressil: https://dzone.com/articles/a-brief-walk-through-net-ecosystem. [Kasutatud 08 04 2022].

[6] Interview Bit, „.NET Core vs .NET Framework,“ Interview Bit, 09 11 2021. [Võrgumaterjal]. Loetud aadressil: https://www.interviewbit.com/blog/net-core-vs-net-framework/. [Kasutatud 10 04 2022].

[7] Microsoft, „What's new in .NET 5,“ Microsoft, 19 02 2022. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-5. [Kasutatud 07 04 2022].

[8] Microsoft, „.NET Standard,“ Microsoft, 26 01 2022. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/dotnet/standard/net-standard. [Kasutatud 07 04 2022].

[9] Microsoft, „.NET Standard versions,“ Microsoft, 2022. [Võrgumaterjal]. Loetud aadressil: https://dotnet.microsoft.com/en-us/platform/dotnet-standard#versions. [Kasutatud 09 04 2022].

[10] Microsoft, „.NET 5+ and .NET Standard,“ Microsoft, 26 01 2022. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/dotnet/standard/net-standard?#net-5-and-net-standard. [Kasutatud 11 04 2022].

[11] Microsoft, „What is ASP.NET?,“ Microsoft, 2022. [Võrgumaterjal]. Loetud aadressil: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet. [Kasutatud 13 04 2022].

[12] Microsoft, „Overview to ASP.NET Core,“ Microsoft, 26 03 2022. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0. [Kasutatud 14 04 2022].

[13] Microsoft, „Choose between ASP.NET 4.x and ASP.NET Core,“ Microsoft, 26 03 2022. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/choose-aspnet-framework?view=aspnetcore-6.0. [Kasutatud 14 04 2022].

[14] Microsoft, „An introduction to NuGet,“ Microsoft, 02 03 2022. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/nuget/what-is-nuget. [Kasutatud 12 04 2022].

[15] Microsoft, „PackageReference in project files,“ Microsoft, 13 04 2022. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/nuget/consume-packages/package-references-in-project-files. [Kasutatud 12 04 2022].

[16] Microsoft, „Migrate from packages.config to PackageReference,“ Microsoft, 02 03 2022. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/nuget/consume-packages/migrate-packages-config-to-package-reference. [Kasutatud 12 04 2022].

[17] S. Smith, "Porting existing ASP.NET apps to .NET 6", Washington: Microsoft Developer Division, 2022, pp. 2-16, 39.

[18] TechEmpower, „Web Framework Benchmarks,“ TechEmpower, 08 02 2021. [Võrgumaterjal]. Loetud aadressil: https://www.techempower.com/benchmarks/#hw=ph&test=plaintext. [Kasutatud 11 04 2022].

[19] Microsoft, „Unavailable technologies,“ Microsoft, 27 01 2022. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/dotnet/core/porting/#unavailable-technologies. [Kasutatud 12 04 2022].

[20] Microsoft, „What's new in .NET 6,“ Microsoft, 27 01 2022. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-6. [Kasutatud 08 04 2022].

[21] Microsoft, „Strategies for migrating incrementally,“ Microsoft, 15 03 2022. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/dotnet/architecture/porting-existing-aspnet-apps/incremental-migration-strategies. [Kasutatud 16 04 2022].

[22] Microsoft, „.NET Portability Analyzer,“ Microsoft, 01 06 2018. [Võrgumaterjal]. Loetud aadressil: https://github.com/microsoft/dotnet-apiport/tree/dev/docs/VSExtension. [Kasutatud 15 04 2022].

[23] Microsoft, „The .NET Portability Analyzer,“ Microsoft, 23 03 2022. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/dotnet/standard/analyzers/portability-analyzer. [Kasutatud 15 04 2022].

[24] R. Reedy, „Jump-starting the Migration to .NET Core with Upgrade Assistant,“ Progress Telerik, 29 03 2021. [Võrgumaterjal]. Loetud aadressil: https://www.telerik.com/blogs/jump-starting-migration-dotnet-core-with-upgrade-assistant. [Kasutatud 16 04 2022].

[25] Microsoft, „.NET Upgrade Assistant,“ Microsoft, 2022. [Võrgumaterjal]. Loetud aadressil: https://dotnet.microsoft.com/en-us/platform/upgrade-assistant. [Kasutatud 08 04 2022].

[26] Microsoft, „.NET Upgrade Assistant Github,“ Microsoft, [Võrgumaterjal]. Loetud aadressil: https://github.com/dotnet/upgrade-assistant. [Kasutatud 15 04 2022].

[27] Microsoft, „.NET Upgrade Assistant Extensibility,“ Microsoft, [Võrgumaterjal]. Loetud aadressil: https://github.com/dotnet/upgrade-assistant/blob/main/docs/extensibility.md. [Kasutatud 15 04 2022].

[28] Microsoft, „.NET Upgrade Assistant Discussion,“ Microsoft, 09 07 2021. [Võrgumaterjal]. Loetud aadressil: https://github.com/dotnet/upgrade-assistant/discussions/692. [Kasutatud 16 04 2022].

[29] Microsoft, „TargetFrameworkSelector.cs,“ Microsoft, [Võrgumaterjal]. Loetud aadressil: https://github.com/dotnet/upgrade-assistant/blob/main /src/components/Microsoft.DotNet.UpgradeAssistant/TargetFramework /TargetFrameworkSelector.cs. [Kasutatud 17 04 2022].

[30] M. Bode, „UpgradeAssistant.Extension.ForceTFM NuGet pakett,“ NuGet, [Võrgumaterjal]. Loetud aadressil: https://www.nuget.org/packages/UpgradeAssistant.Extension.ForceTFM. [Kasutatud 17 04 2022].

[31] M. Bode, „UpgradeAssistant.Extension.CleanUp NuGet pakett,“ NuGet, [Võrgumaterjal]. Loetud aadressil: https://github.com/markobode11/extension-cleanup. [Kasutatud 11 05 2022].

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

Mina, Marko Bode

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „.NET Upgrade Assistant tööriista parendamine“, mille juhendaja on Liisa Jõgiste.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

16.05.2022

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

Lisa 2 – *UpgradeAssistant.Extension.ForceTFM* koodihoidla

https://github.com/markobode11/extension-forceTFM

Lisa 3 – *UpgradeAssistant.Extension.CleanUp* koodihoidla

https://github.com/markobode11/extension-cleanup

Lisa 4 – .NET 5-lt .NET 6-le uuendatud rakenduse koodihoidla

https://github.com/markobode11/Gym-Buddy

Lisa 5 – .NET Framework 4.8-lt .NET 6-le uuendatud rakenduse koodihoidla

https://github.com/markobode11/SimpleWebApp