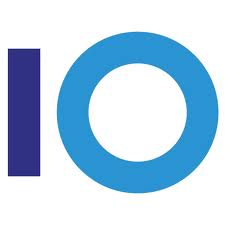
**Универзитет Св. ʻʻКирил и Методијʼʼ – Скопје**

**Факултет за информатички науки и компјутерскo**

**инженерство**

**Дипломска работа**

Анализа на **.NET Core**

Ментор: Кандидат:

Проф. Д-р. Иван Чорбев Војдан Гичаровски

Индекс број: 111040

Скопје, 2017

Содржина

[Апстракт 3](#_Toc478651775)

[Клучни зборови 3](#_Toc478651776)

[Вовед 4](#_Toc478651777)

[Отворен изворен код 5](#_Toc478651778)

[Наменета за повеќе оперативни системи и платформи 6](#_Toc478651779)

[Mono 6](#_Toc478651780)

[Xamarin 7](#_Toc478651781)

[Компоненти 8](#_Toc478651782)

[C# Компајлер – Roslyn 8](#_Toc478651783)

[Алатка за градење софтвер – MSBuild 9](#_Toc478651784)

[Алатки за командна линија – dotnet 10](#_Toc478651785)

[Библиотеки што се користат при извршување на софтвер – CoreCLR 11](#_Toc478651786)

[Библиотеки – CoreFX 11](#_Toc478651787)

[.NET Стандард 12](#_Toc478651788)

[Менаџер на пакети – NuGet 13](#_Toc478651789)

[Виртуелна машина – Docker 13](#_Toc478651790)

[Рамка за развивање веб апликации - ASP.NET MVC 13](#_Toc478651791)

[Kestrel 13](#_Toc478651792)

[Owin 13](#_Toc478651793)

[Интегрирана околина за развој - Visual Studio 2017 13](#_Toc478651794)

[Пример апликации и нивна разлика при развој со .NET Framework 4.6.2 и .NET Core 1.0 13](#_Toc478651795)

# Апстракт

Во оваа дипломска работа се прави анализа на .NET Core рамката изработена од компанијата Microsoft. .NET Core е следната верзија на рамката .NET Framework којашто се користи при изработка на сите типови на апликации, како веб, мобилни и клиентски апликации, со помош на јазикот C#.

Во почетокот на оваа дипломска работа ќе се зборува за историското значење на .NET Core и како тоа влијае на програмерската заедница во целост, но и како влијае конкретно на компанијата Microsoft.

Главна цел е да се опфатат новостите и разликите на .NET Core во однос на неговиот претходник .NET Framework, но исто така и да се објаснат подетално сите негови компоненти и промените кои ги претрпеле со новата верзија на .NET.

## Клучни зборови

.NET Core, .NET Framework, ASP.NET Core, NuGet, Roslyn, .NET Standard, Open source, Cross platform, Docker, OWIN, Kestrel

# Вовед

Денес живееме во дигитално доба каде што компјутерите и мобилните уреди ни помагаат при извршување на секојдневните обврски, ни ги олеснуваат и забрзуваат здодевните задачи како плаќање сметки, пазарење, пишување дипломски задачи, но и ни овозможуваат да се забавуваме и комуницираме со пријателите, можеме да играме игри, да слушаме музика или да гледаме некој интересен филм.

Развитокот на компјутерите не е краток и едноставен, од првиот електричен компјутер со сијалици оспособен за програмирање, до денешните компјутери со микро-транзистори и по неколку гигабајти меморија и огромна процесорска моќ поминати се повеќе од 70 години. Развојот на хардверот и материјалите што се користат за негова изработка, но и самиот процес на изработка напреднале значително за во денешно време да можеме во нашиот џеб да носиме компјутер со поголема процесорска моќ од компјутерот што се користел за лансирање и водење на леталото Аполо 11.

Паралелно како што се развивал хардверот со него се развивал и софтверот којшто го користи. Од првата верзија на Windows развиена во 1985 до денес можеме да забележиме огромен напредок во функционалностите што ги нуди, интеракцијата и начинот на изработка на апликации.

За да може да се изработи една апликација потребни се компајлер за соодветниот јазик кој што се користи за програмирање, кој во тоа време бил јазикот Ц, и библиотеки за интеракција со периферните уреди. Во денешно време постојат голем број на јазици за програмирање меѓу кои и јазикот C# којшто е осмислен, изработен и одржуван од страна компанијата Microsoft. Овој јазик сам по себе не може да се користи за да се направи продуктивна апликација, туку мора да се употребува во комбинација со рамката .NET за изработка на апликација способна за извршување и комуникација со периферните уреди.

.NET претставува колекција на библиотеки за менаџмент на меморија, редослед на извршување и преведување на програмскиот код во инструкции разбирливи за компјутерот. Во сржта .NET содржи библиотека наречена CLR (common language runtime) што всушност го преведува напишаниот код во соодветни машински инструкции спремни за извршување. Останатите библиотеки помагаат за интеракција со периферните уреди и ги олеснуваат операциите со различните типови на информации (броеви, букви, реченици, низи).

# Отворен изворен код

Во изминатите години Microsoft беше компанија којашто своите технологии ги изработуваше во тајност, но со новата верзија на .NET и воедно со промената на главниот извршен директор Сатија Надела (анг. Satya Nadella) се реши да го објави изворниот код на технологијата .NET Core, компајлерот и компонентите. Со овој чекор Microsoft придонесе за развој на технологија којашто е најблизу до потребите на развивачите на софтвер коишто ќе ја употребуваат за развој на најразлични апликации.

Кодот е отворен за разгледување и надополнување за сите заинтересирани развивачи, но тие мораат да ги почитуваат правилата поставени од страна тимот на Microsoft што е одговорен за соодветната технологија. Овие правила обично се однесуваат на стилот на пишување код, шаблоните што се употребуваат, стилот на пишување GIT пораки, авторски права на код, процес за поднесување за барање за додавање или модифицирање на код, начин и место за дискусија за проблемите што се треба да се решат, стандард на јавни повици и структури којшто библиотеките мора да го запазат и други правила кои што се специфични за секоја компонента поодделно.

Иако кодот е отворен за модификации стандардот за функционалностите што оваа рамка мора да ги нуди е однапред одреден и напишан како засебна библиотека .NET Standard којашто мора да биде интегрирана, што во суштина значи дека сите рамки што ја интегрираат оваа базична библиотека ќе го имаат истиот јавен интерфејс за своите базични функционалности. Сите останати дополнителни функционалности развиени во рамките на .NET Core ќе бидат специфични само за оваа рамка.

Овој стандард го решава проблемот кој се јавува при споделување на библиотеки во различни платформи и рамки. На пример ако еден развивач на софтвер напише библиотека за манипулација на дати и временски зони и таа библиотека е зависна од .NET Standard 1.6 тогаш таа библиотека ќе може да биде искористена во било која апликација што користи рамка (.NET Framework vNext или .NET Core 1.0) што го интегрира соодветниот стандард.

# Наменета за повеќе оперативни системи и платформи

Најзначајната промена на .NET рамката со изработката на верзијата .NET Core е што ќе биде достапна за развој и користење на повеќе платформи и оперативни системи. Во минатото Microsoft се фокусираа да изградат рамка за развој којашто работи ексклузивно за Windows, но сега тие изработуваат рамка којашто ќе може да се извршува на повеќето познати дистрибуции на Linux и на macOS.

Оваа промена има посебно значење за развивачите на софтвер затоа што сега нема да мора да се потпираат на други алатки за програмирање за повеќе платформи одеднаш како Mono, Cordova или Qt, туку ќе можат да ја користат рамката .NET Core.

Исто така оваа промена ги засегнува и клиентите затоа што со оваа промена веб апликациите не мора да бидат хостирани ексклузивно на Windows сервер, туку може и на сервер со Linux оперативен систем доколку има поддршка за .NET Core рамката. Со оглед на тоа што Linux серверите се поевтини, цената на хостирање на ASP .NET апликации би паднала значително на светско ниво.

## Mono

Mono е првиот обид да се произведе платформа со ист јавен стандард како .NET Framework којашто би работела на повеќе различни оперативни системи и уреди како Linux, Solaris, BSD macOS, Android, iOS, Wii, Xbox 360, PlayStation 3. Овој проект е изработен со цел да се олесни развивањето на .NET апликации на Linux оперативниот систем, но и да се даде можност да се развиваат квалитетни апликации со користење на C# јазикот за повеќе платформи одеднаш.

Структурата на Mono платформата е направена според стандардите и шаблонот на .NET Framework коишто ги има објавено Microsoft под стандардите ECMA-334 и ECMA-335, односно развиени се C# компајлер за преведување на C# код во бајт код, CLR (common language runtime) компајлер за статичко, навремено и предвремено компајлирање на бајт код во машиснки код за соодветниот процесор, библиотеки според .NET Standard спецификацијата.

## Xamarin

Xamarin е компанија којашто е основана во 2011 од развивачите на проектот Mono со главна цел да развие платформа и интегрирана работна околина за развивање на софтвер за мобилните оперативни системи Android, iOS и Windows Phone во јазикот C# на било кој оперативен систем.

Овој потфат има повеќе големи значења за софтверската индустрија и за развивачите на софтвер за мобилни уреди. Пред да се развие овој проект развивачите на софтвер беа принудени да развиваат апликации поодделно за секој мобилен оперативен уред што најчесто значи развивање на истата апликација три пати во различни технологии и јазици соодветно за Android, iOS и Windows Phone. Потоа се разви платформата Cordova којшто во суштина апликациите ги прикажува како веб страни, иако овој начин на развивање апликации за повеќе оперативни системи одеднаш заштедува време, најчесто не се практикува затоа што апликациите развиени на овој начин се многу бавни и нивните анимации не се прикажуваат како што треба.

Xamarin работи како обвивка на нативните библиотеки што ги нудат алатките за развивање на трите мобилни платформи што значи дека при компајлирање на напишаниот код во C# тој се преведува во нативен код за соодветната платформа. Со цел за користењето на овој софтвер да биде лесно и за развивачите коишто ги користат оперативните системи Linux и macOS, компанијата Xamarin има развиено интегрирана околина Xamarin Studio за работа и за овие оперативни системи со цел да ја замени околината за развој Visual Studio произведена од Microsoft којашто работи ексклузивно на Windows.

Во февруари 2016 година Microsoft и Xamarin потпишаа договор за откуп со што сега оваа компанија припаѓа на Microsoft. Ова има големо значење за развивачите на софтвер што користат Linux и macOS оперативни системи затоа што Microsoft започна со официјална поддршка на интегрираната околина Xamarin Studio. Со развитокот на .NET Core и неговата одлика дека може да се извршува на повеќе оперативни системи, развивачите на софтвер за прв пат добиваат шанса да можат да развијат веб, клиентски и мобилен софтвер во Xamarin Studio, што наликува со своите функционалности на Visual Studio (но ги нема сите), во јазикот C# на било кој оперативен систем што е поддржан од Xamarin Studio и .NET Core.

# Компоненти

Развивањето на софтвер со помош на рамката .NET Core можеби изгледа како едноставна и лесна работа, но доколку навлеземе подлабоко во компајлирањето и извршувањето на напишаниот код, тестирање и пакување со цел оптимално прикачување на апликацијата на соодветната машина што ќе ја извршува брзо ќе сфатиме дека целата рамка е многу сложена и со цел да се олесни нејзиното одржување и ажурирање таа е поделена во повеќе компоненти.

## C# Компајлер – Roslyn

Како што работата на развивач на софтвер е да ја преведе бизнис логиката побарана од клиентот во соодветен код, така работата на компајлерот е да го преведе тој код во соодветни машински инструкции што може да ги разбере компјутерот, но во овој случај не се случува тоа. Поради тоа што првично програмерите имаат избор да развиваат софтвер со јазиците C# и Visual Basic, решението на Microsoft е овие јазици да се преведат во некој среден јазик (CIL – Common intermediate language) на пониско ниво близу машински код а тој потоа да се изврши со помош на CLR.

.NET Platform Compiler или попознат како Roslyn компајлер е алатката којашто го преведува C# кодот во соодветен CIL код. Со новата верзија на .NET Core доаѓаат и нови функционалности на компајлерот во форма на анализатори на синтакса, семантички анализатори и декомпајлери. Овие новости не се зависни од .NET Core рамката туку од верзијата и синтаксата којашто се поддржува на C# јазикот, но тоа не значи дека веќе постоечките функционалности не може да се подобруваат и да се поправаат доколку има некоја грешка или нелогичност при нивно извршување.

Библиотеките коишто го сочинуваат компајлерот сепак се дел од .NET Core рамката иако не зависат од нејзините функционалности, со тоа се и дел од проектот што го градиме. Во претходните .NET Framework верзии овие библиотеки не беа дел од рамката, туку се додаваа во проектот како NuGet пакети со што се олеснува нивното ажурирање, но во .NET Core рамката тие се дел составен дел и секогаш се користат истите верзии од овие библиотеки во секој проект што ја користи оваа рамка, па со тоа се одржува конзистентност на нивното верзионирање. Најновата верзија на Roslyn компајлерот ја поддржува јазикот C# верзија 7.0.

При компајлирање на .NET проектите се повикува MSBuild алтката којашто го гради решението според стратегијата најдена во конфигурациска датотека, но секоја C# датотека се компајлира посебно со помош на csc.exe како повикувачки команден интерфејс, односно C# компајлерот Roslyn кој доаѓа заедно со .NET рамката со соодветна верзија вклучена во оперативниот систем. При извршување на апликацијата понекогаш се јавува потреба да се интерпретираат дадотеки во реално време (пример razor датотеки), при што тогаш се користи истиот Roslyn компајлер, но се повикува по потреба од страна на самата апликација.

## Алатка за градење софтвер – MSBuild

MSBuild (Microsoft Build Tools) е алатка за создавање, компајлирање, тестирање, пакување и прикачување на апликации и генерирање на документација. MSBuild е независен од Visual Studio, но Visual Studio го користи MSBuild во позадина за извршување на своите задачи. Тоа значи дека доколку сакаме да изградиме некој продукт можеме да го направиме тоа без воопшто да користиме Visual Studio, туку само алатката MSBuild.

Во првичната алфа верзија на рамката .NET Core се отфрли користењето на конфигурацискаta датотека .csproj со .xproj и project.json, со што се раздели стратегијата за компајлирање на проектот во датотеката .xproj и листата на NuGet пакетите и верзиите на .NET рамката којашто се користи во датотеката project.json. Оваа промена не траеше долго затоа што тимот од Microsoft што ја изработува рамката .NET Core одлучи во верзијата 1.1.1 да се врати на старата структура на конфигурирање со .csproj датотека.

.NET Core апликациите може да се спакуваат на два различни начина, едниот начин е да се искористи инсталираната .NET рамка и да се извршува со нејзина помош, а вториот начин е да се вклучи .NET рамката како дел од проектот при што иако корисниот ја нема инсталирано на својот компјутер сепак ќе може да ја изврши апликацијата без никаков проблем.

Првиот начин на користење на .NET Core се нарекува изградба зависна од рамка (анг. Framework-dependent deployments - FDD) при што единствен параметар е името на рамката од којашто зависи проектот. Овој начин на градење на апликацијата има предност затоа што апликацијата може да се користи на било кој оперативен систем којшто ја има инсталирано .NET Core рамката.

Вториот начин се нарекува самостојно градење (анг. Self-contained deployments - SCD) и притоа се предава параметар што означува која верзија на извршување на рамката се користи, пример доколку сакаме да ја изградиме апликацијата за Windows 10 x64 мораме да предадеме параметар win10-64. Овој начин нуди можност да може да се извршува оваа апликација на било кој компјутер со соодветниот оперативен систем без разлика дали ја има инсталирано .NET Core рамката затоа што целата библиотека е вградена во самата апликација.

## Алатки за командна линија – dotnet

Една од поголемите новости во .NET Core е алатката за командна линија (.NET Core CLI) којашто служи за управување со животниот циклус на една апликација во развој. Во претходните верзии на .NET рамката оваа задача ја имаше околината DNX (Dot Net eXecution).

DNX се состои од повеќе компоненти коишто сега се заменети или надградени за полесна употреба. Иако примарната употреба на DNX беше за изградба на ASP.NET MVC 5 апликации, послужи како база за изградба на .NET Core CLI. Во стржта DNX ги содржи целата .NET Framework рамка, команден интерфејс за интеракција со DNX и апликацијата што се гради со негова помош, менаџмент на NuGet библиотеки и околина за извршување.

Со .NET Core CLI веќе нема потреба да се користи .NET Core библиотеката локално затоа што таа сега е NuGet библиотека којашто се симнува по потреба. Околината за извршување на .NET Core апликации самостојно е способна да се користи на повеќе платформии и е дел од .NET Core рамката, па поради тоа .NET Core CLI нема потреба да ја содржи оваа околина.

Базичните команди коишто можат да се искористат се:

* new
* restore
* build
* publish
* run

Командата “new” служи за креирање нов проект од типот наведен како следен аргумент. Синтаксата за искористување на оваа команда е dotnet new <Template> каде што <Template> може да се замени со console, classlib, mstest, xunit, web, mvc, webapi, nugetconfig, webconfig, sln. Оваа команда креира нов проект од содветиот тип и ги иницијализира соодветните NuGet библиотеки.

Командата “restore” ги симнува од NuGet репозиториумот соодветните NuGet библиотеки наведени како зависности во проектот. Оваа команда мора да се изврши во главниот директориум на проектот за којшто сакаме да ги симнеме соодветните библиотеки.

Командата “build” служи за градење и компајлирање на проектот и библиотеките од коишто зависи. Излезот од оваа команда содржи .dll датотеки за самата апликација и библиотеките и .pdb датотеки за дебагирање на апликацијата. Оваа команда има додатни параметри за специфицирање на профилот за градење и целната платформа.

Командата “publish” е слична со командата “build” со таа разлика што резултатот не содржи датотеки за дебагирање.

Командата “run” служи за извршување на апликациите преку командна линија. За оваа команда може да се специфицираат верзија на рамка и датотека на конфигурација.

Во скоп на оваа алатка може да се користат и команди за менаџирање на миграции на бази на податоци, извршување на тестови и менаџирање не NuGet пакети.

## Библиотеки што се користат при извршување на софтвер – CoreCLR

Како еден од најзначајните новости на .NET рамката е CoreCLR којашто претставува околина за извршување на средниот јазик CIL. Откако компајлерот Roslyn ќе го искомпајлира C# кодот во CIL и ќе создаде .exe и .dll датотеки, при извршување на апликацијата задачата на CoreCLR е да го преведе овој CIL код во нативен машински код.

Во претходните верзии на .NET оваа околина за извршување апликации беше достапна само на Windows оперативниот систем или како дел од проектот Mono или DNX за други оперативни системи, но сега со .NET Core CLR може да се извршуваат апликации на Windows, Linux, OSx и FreeBSD оперативните системи и на x86, x64 и ARM архитектури на процесори.

CoreCLR во себе содржи компоненти за алокација на меморија, вчитување на класи, систем на типови, систем за менаџмент на процеси, имплементација на примитивни типови како броеви, букви и реченици. Освен овие базични компоненти, репозиториумот на CoreCLR содржи и компоненти коишто се во блиско сродство, како:

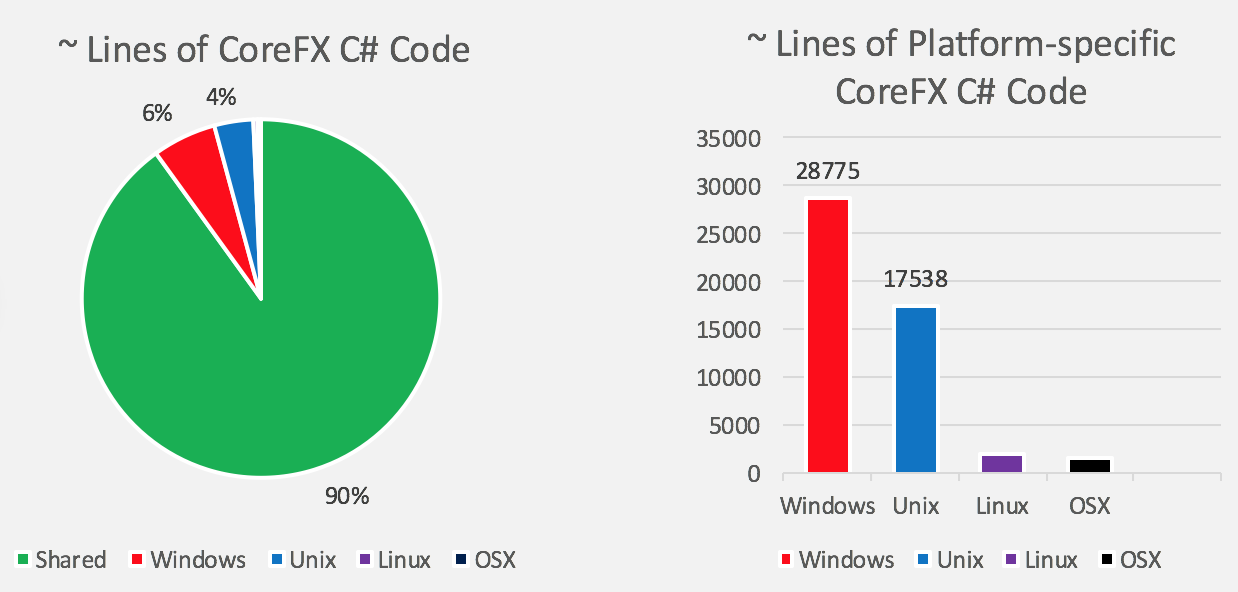
* компајлер во реално време (JIT) за преведување на код во среден јазик во машински код
* асемблер и дисасемблер за средниот јазик
* обвивка за процесирање на динамички поврзани библиотеки
* јавна библиотека за развој врз CoreCLR

## Библиотеки – CoreFX

CoreFX претставува колекција на библиотеки коишто припаѓаат на .NET Core рамката. Секој директориум којшто почнува со Microsoft.\* или System.\* во “src” директориумот претставува имплементација на библиотека. Секоја библиотека има “src” директориум каде што се наоѓа имплементацијата на соодветната библиотека и “ref” директориум каде што се наоѓа дефиницијата за таа библиотека.

Некои од библиотеките имплементираат базични функционалности па поради тоа нивниот изворен код е дел од CoreCLR репозиториумот, а во CoreFX се наоѓа нивната дефиниција. При пристап на овие базични библиотеки се повикува имплементацијата дефинирана во System.Private.Corelib асемблито во CoreCLR библиотеката.

Како дел од CoreFX иако работата на најголемиот број на библиотеки не зависи од оперативниот систем, може да забележиме дека има библиотеки коишто мора да зависат од оперативниот систем на којшто би се извршувале. Ваков пример се библиотеките за криптографија и библиотеките за работа со периферни уреди, па поради оваа причина тие се имплементирани засебно за секој оперативен систем. Исклучок на ова правило е библиотеката за работа со регистарот на податоци којшто постои исклучиво само на Windows оперативниот систем, па поради тоа постои имплементација само за овој оперативен систем.



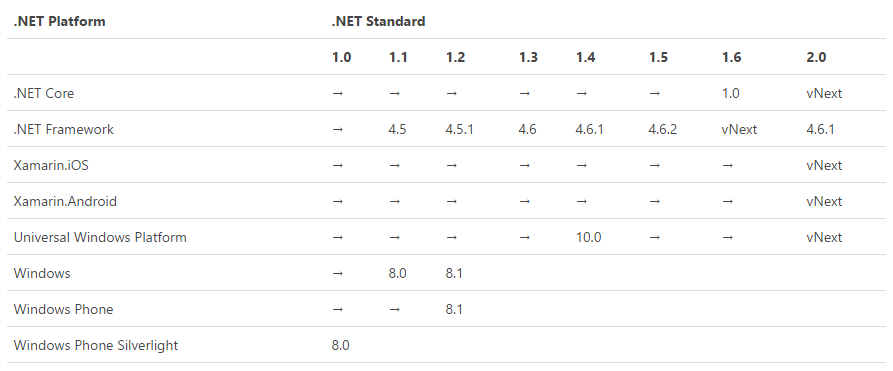
## .NET Стандард

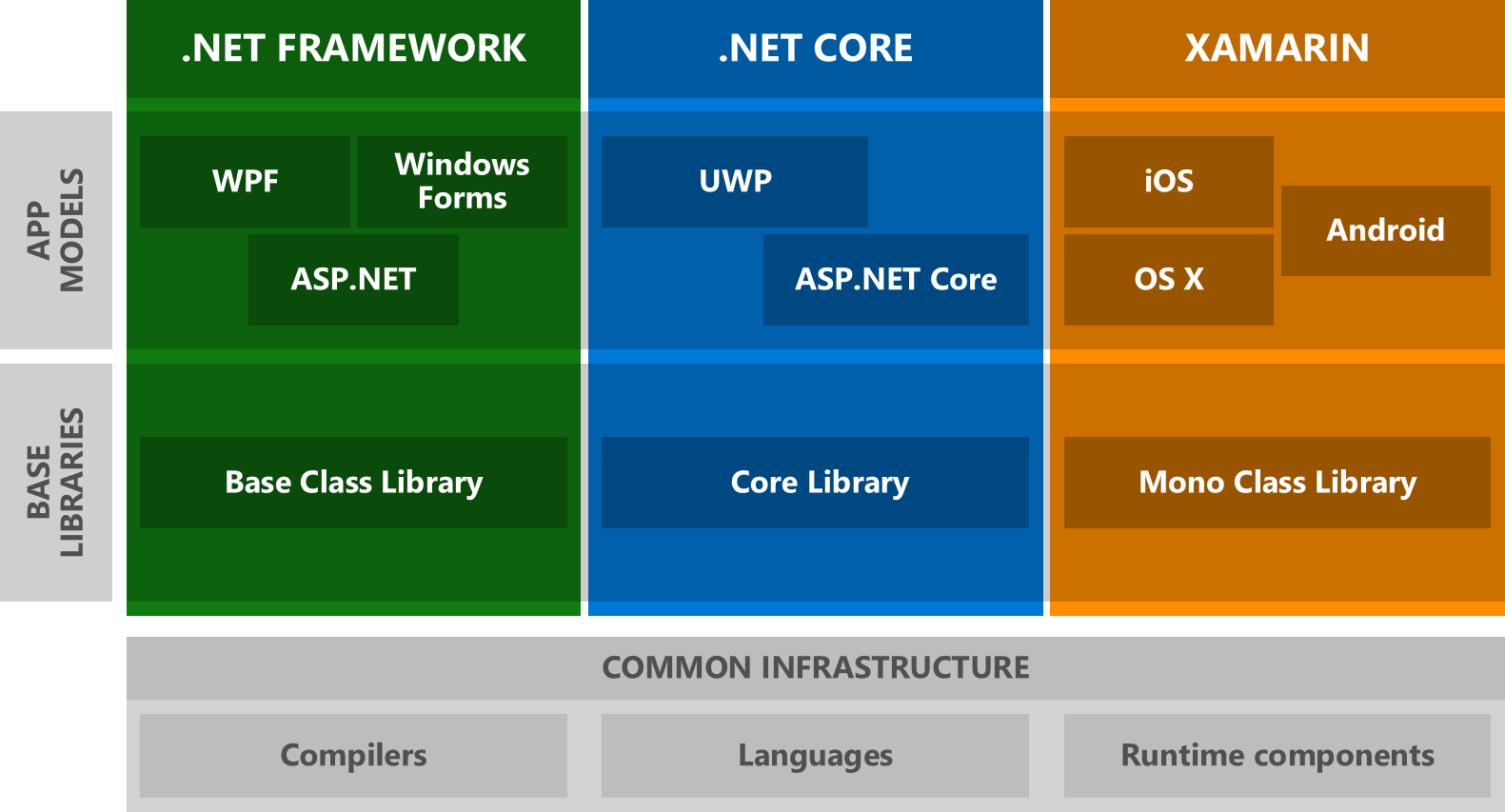
Со цел да може да се извршува .NET Core апликација на било кој оперативен систем, корисникот мора да се осигура дека .NET Core Runtime библиотеката е присутна на оперативниот систем. Оваа библиотека ги имплементира сите базични функционалности за секој оперативен систем, но постојат и функционалности коишто се специфични за оперативниот систем на којшто се извршува, што значи дека е возможно корисникот да пристапува функционалност достапна само на Windows или на Linux. За да се минимизираат проблемите коишто може да се јават од оваа природа, Microsoft воведоа стандард наречен .NET Standard којшто всушност претставува колекција на библиотеки коишто гарантирано се достапни на секој оперативен систем што има инсталирано соодветна верзија на .NET рамката.

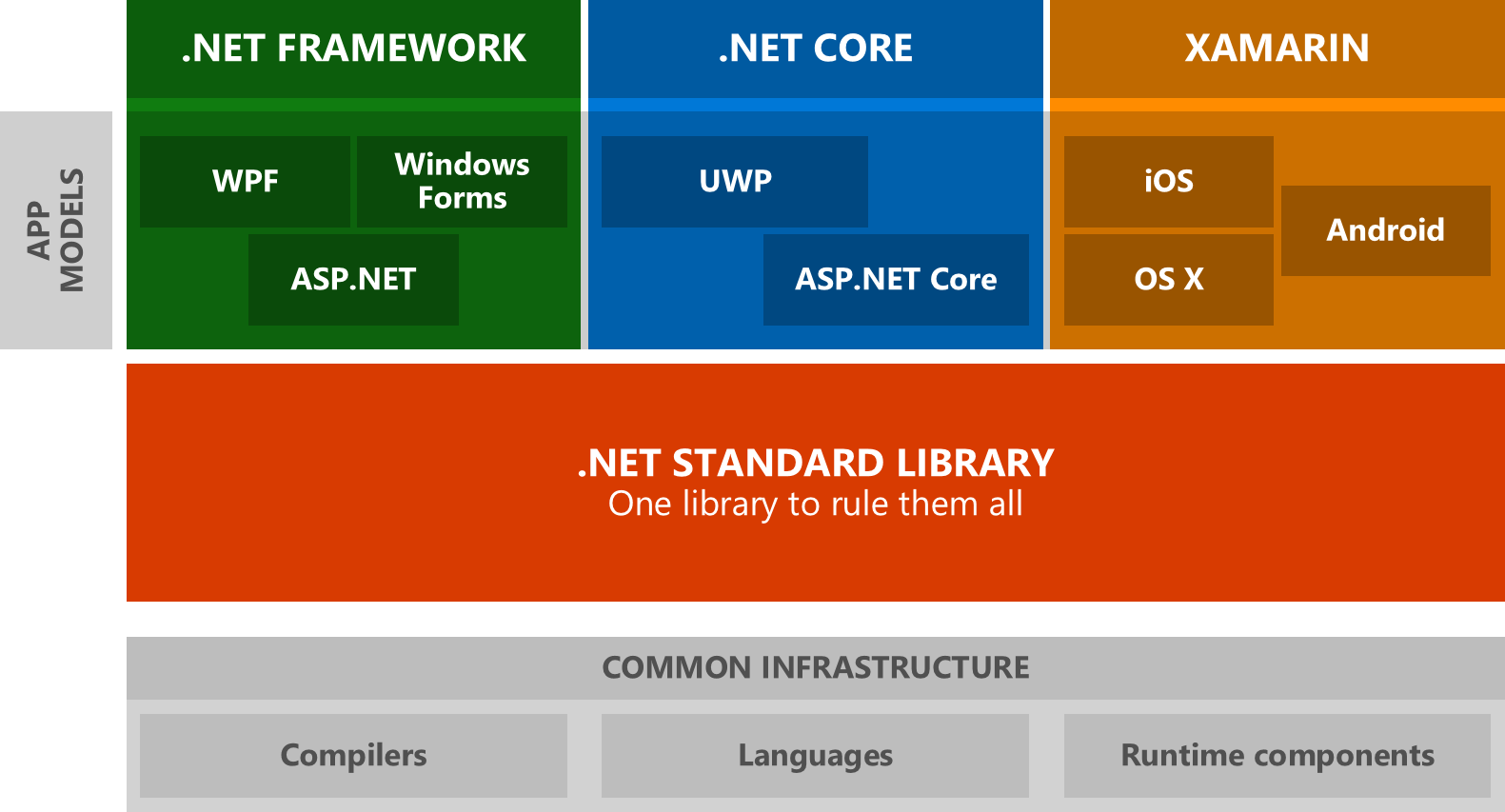
.NET Core 1.0 верзијата се води според .NET Standard 1.6, што значи дека секој оперативен систем којшто ја има инсталирано .NET Core 1.0 Runtime библиотеката ќе може да извршува апликации, односно библиотеки што ги користат апликациите изградени според .NET Standard 1.6.

Постојат три официјално поддржани рамки за развој на апликации од различна природа: .NET Framework, .NET Core и Xamarin. Доколку .NET Standard не постои тогаш секоја од овие рамки би имала различна колекција на базични библиотеки при што би се јавил проблем за интеграција на сопствена библиотека во сите три рамки одеднаш. Овој проблем беше решен со изградба на портабилни класни библиотеки коишто можат да се извршуваат на сите рамки без проблеми, но тогаш само пресекот на функционалностите што ги нудат рамките би биле достапни за развивачите на сопствената библиотека. За да се унифицираат функционалностите што ги нудат сите рамки, тие мора да се водат според стандард на заеднички јавни повици .NET Standard.

Овој стандард е компатибилен со своите постари верзии, што значи дека доколку корисникот има инсталирано .NET Core 1.0 верзија на својата машина, тој може да интегрира библиотеки изградени со било која верзија на .NET Standard помала или еднаква на верзијата 1.6. Овој начин на имплементација на оваа колекција на библиотеки го елиминира проблемот кој се јавува при надградба на верзијата на .NET рамката на соодветната машина.







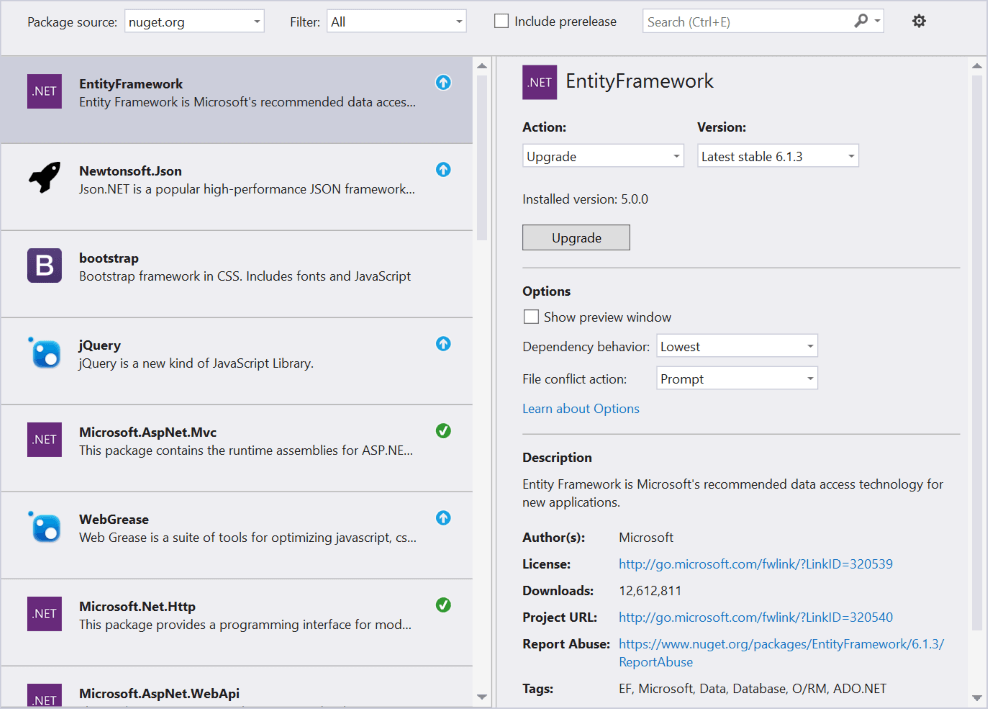
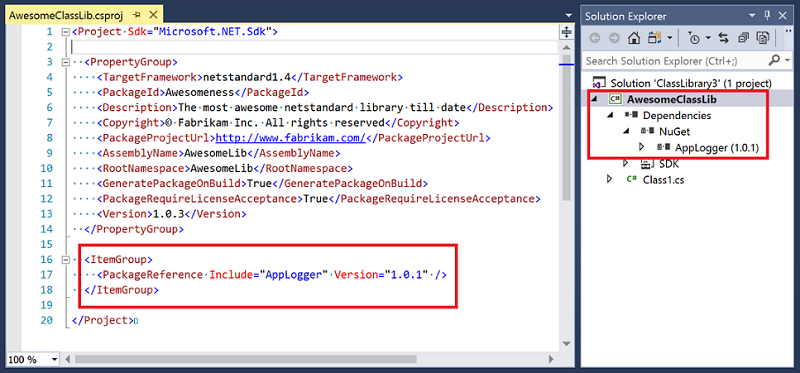
## Менаџер на пакети – NuGet

При развој на една .NET апликација потребно е да се користат најразлични библиотеки кои решаваат чести проблеми, за полесен пристап до овие библиотеки развивачите на .NET платформата нудат менаџер за пакети наречен NuGet.

Најосновната функција на NuGet е пребарување и консумирање на библиотеки наречени NuGet пакети, но исто така и менаџмент на верзиите на веќе спуштените пакети, менаџмент на целната верзија на .NET рамката, нивно консолидирање низ повеќе проекти и градење на наши сопствени пакети за глобална консумација.

Пред официјалното пуштање во употреба на Visual Studio 2017, NuGet менаџерот беше посебен продукт интегриран во околината за развој. При спуштање на NuGet пакети, информациите за верзијата на целната .NET рамка, името и верзијата на NuGet пакетот се запишуваа во packages.config датотека.

Новата верзија на NuGet 4.0 менаџерот веќе не е посебен продукт, туку е интегриран како дел на MSBuild со што ја снемува потребата за packages.config датотеката, затоа што сега пакетите се дефинираат во .csproj датотеката како PackageReference запис. Покрај оваа промена подобрени се времето на симнување на пакети и приказот на пакетите од кои зависи нашиот проект со тоа што не се симнуваат повеќе пати при промена на верзијата на пакетот и не се прикажуваат пакетите коишто не се од директна корист за нашиот проект.

## Рамка за развивање веб апликации - ASP.NET Core

ASP.NET е рамка со отворен изворен код наменет за изработка на модерни веб апликации и сервиси со помош на C#, HTML5, CSS и JavaScript коишто можат да опслужат милиони корисници.

### Kestrel

### Owin

## Виртуелна машина – Docker

# Интегрирана околина за развој - Visual Studio 2017

# Пример апликација и разлика при развој со .NET Framework 4.6.2 и .NET Core 1.0