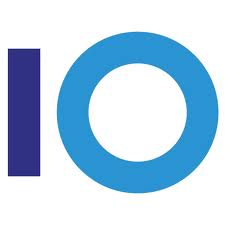
**Универзитет Св. ʻʻКирил и Методијʼʼ – Скопје**

**Факултет за информатички науки и компјутерскo**

**инженерство**

**Дипломска работа**

Анализа на **.NET Core**

Ментор: Кандидат:

Вонр. Проф. Д-р. Иван Чорбев Војдан Гичаровски

Индекс број: 111040

Скопје, 2017

Содржина

[Апстракт 3](#_Toc483586429)

[Клучни зборови 3](#_Toc483586430)

[1. Вовед 4](#_Toc483586431)

[2. Отворен изворен код 5](#_Toc483586432)

[3. Наменета за повеќе оперативни системи и платформи 6](#_Toc483586433)

[3.2. Mono 7](#_Toc483586434)

[3.3. Xamarin 7](#_Toc483586435)

[4. Разлики на .NET Core и .NET Framework повици 9](#_Toc483586436)

[5. Компоненти 9](#_Toc483586437)

[5.1. C# Компајлер – Roslyn 10](#_Toc483586438)

[5.2. Алатка за градење софтвер – MSBuild 12](#_Toc483586439)

[5.3. Алатки за командна линија – dotnet 14](#_Toc483586440)

[5.4. Библиотеки што се користат при извршување на софтвер – CoreCLR 16](#_Toc483586441)

[5.5. Библиотеки – CoreFX 17](#_Toc483586442)

[5.6. .NET Стандард 18](#_Toc483586443)

[5.7. Менаџер на пакети – NuGet 20](#_Toc483586444)

[5.8. Рамка за развивање веб апликации - ASP.NET Core 22](#_Toc483586445)

[5.8.1. OWIN 22](#_Toc483586446)

[5.8.2. Kestrel 24](#_Toc483586447)

[5.8.3. Docker 25](#_Toc483586448)

[6. Пример апликација и разлика при развој со .NET Framework 4.6.2 и .NET Core 1.0 26](#_Toc483586449)

[6.1. NuGet пакети 27](#_Toc483586450)

[6.2. Почеток на апликацијата 30](#_Toc483586451)

[6.3. Инверзија на контрола 31](#_Toc483586452)

[6.4. Начин на хостирање 33](#_Toc483586453)

[7. Заклучок 34](#_Toc483586454)

[8. Референци 34](#_Toc483586455)

# Апстракт

Во оваа дипломска работа е направена анализа на .NET Core рамката изработена од компанијата Microsoft. Се објаснуваат нејзините карактеристики и се разгледуваат нејзините предности во однос на постарата рамка за развој .NET Framework. Се посочуваат придобивките од користење на новата .NET Core рамка за развој на било кој тип на апликација.

Во почетокот на оваа дипломска работа се зборува за историското значење на .NET Core и како тоа влијае на програмерската заедница во целост. Се опишуваат обидите на програмерската заедница да се произведе платформа за развој слична на .NET за употреба на било кој оперативен систем.

Главна цел е да се опфатат новостите и разликите на .NET Core во однос на неговиот претходник .NET Framework, Исто така и се објаснуваат подетално сите негови компоненти и промените кои ги претрпеле со новата верзија на .NET. Се прави анализа на нивната функционалност и промените кои морале да ги претрпат за да се добие рамката .NET Core токму со функционалностите што ги нуди.

Подетално се разгледуваат разликите при развој на веб апликација со ASP.NET Core MVC и ASP.NET MVC 5. Се објаснуваат новите функционалности што ги нуди ASP.NET Core во однос на неговиот претходник и начинот на работа со истите.

## Клучни зборови

.NET Core, .NET Framework, ASP.NET Core, NuGet, Roslyn, CoreCLR, .NET Core CLI, .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. Технологијата ни ги олеснува и забрзува здодевните задачи како плаќање сметки, пазарење, но и ни помага при извршување на работните задачи. Исто така ни овозможуваат да се забавуваме и комуницираме со пријателите, да 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 лампи 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. Поради тоа во денешно време секој може во својот џеб да носи 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. Доколку се направи ретроспектива на оперативните системи може да се забележи дека Windows, Linux и macOS како најприсутни оперативни системи значително напреднале. Windows има посебно значење за .NET рамката затоа што сите претходни верзии можат да се извршуваат само на овој оперативен систем. Од првата верзија на Windows развиена во 1985 до денес може да се забележи 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 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.

.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 различните типови на информации.

# Отворен изворен код

Порано Microsoft била компанија којашто сите свои технологии ги изработувала во тајност, но изворниот код на новата верзија на .NET е објавен заедно со сите негови компоненти, компајлерот и алатките за развој. Со овој чекор Microsoft придонел за развој на технологија којашто е најблизу до потребите на развивачите на софтвер коишто ќе ја употребуваат за развој на најразлични типови на апликации.

Кодот на .NET Core 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 и шаблоните што се употребуваат.

Дел од поставените правила кои што не се од технички аспект се:

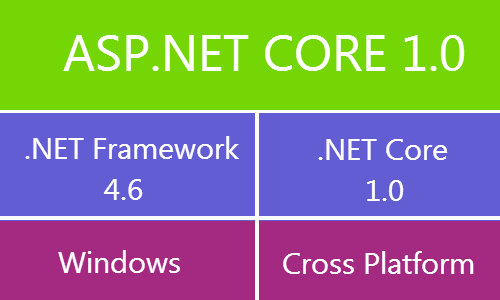
* Стилот на пишување GIT пораки
* Авторски права на код
* Процес за поднесување за барање за додавање или модифицирање на код
* Начин и место за дискусија за проблемите што се треба да се решат
* Стандард на јавни повици и структури којшто библиотеките мора да го запазат

Секоја од компонентите што се развива има и свои специфични правила што мора да се запазат.

Иако кодот е отворен за модификации стандардот за функционалностите што оваа рамка мора да ги нуди е однапред одреден и напишан како засебна платформа .NET Standard. Овој стандард мора да биде интегриран, што во суштина значи дека сите рамки што ја интегрираат оваа базична колекција на библиотеки 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.

# Наменета за повеќе оперативни системи и платформи

Најзначајната промена на .NET рамката со изработката на верзијата .NET Core е што е достапна за развој и користење на повеќе платформи и оперативни системи. Во минатото Microsoft се фокусирале да изградат рамка за развој којашто работи ексклузивно на Windows оперативниот систем. Новата верзија на .NET е способна да се извршува на Windows и на повеќето познати дистрибуции на Linux и macOS.



**Слика бр. 1 Достапни платформи за ASP.NET Core апликација**

Поради тоа што оваа платформа може да се извршува на повеќе оперативни системи одеднаш, развивачите на софтвер не мораат да се потпираат на други алатки за програмирање. Досега развивачите биле принудени да користат алатки како Mono, Cordova или Qt, но сега можат да ја користат рамката .NET Core за истата намена. Исто така оваа промена ги засегнува и клиентите затоа што сега 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.

## Mono

Mono е првиот обид да се произведе платформа со ист јавен стандард како .NET Framework. Целта е оваа рамка да работи на повеќе различни оперативни системи и уреди како: Linux, Solaris, BSD, macOS, Android, iOS, Wii, Xbox 360, PlayStation 3. Овој Mono проект е изработен со цел да може да се извршуваат .NET апликации на Linux оперативниот систем. Исто така се овозможува 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. Со цел да се имплементираат овие шаблони развиени се: C# компајлер за преведување на C# код во среден јазик, CLR (common language runtime) компајлер за статичко, навремено и предвремено компајлирање на среден јазик во машински код за соодветниот процесор и библиотеки според .NET Standard спецификацијата.

## Xamarin

Xamarin е компанија којашто е основана во 2011 од развивачите на проектот Mono, за развој на платформа именувана според компанијата. Целта на платформата Xamarin е да се развие платформа слична на Mono, но за развивање на мобилни апликации. Освен платформата за развој, изработена е интегрирана работна околина за развивање на софтвер за мобилните оперативни системи Android, iOS и Windows Phone. Исто како проектот Mono, јазикот за развој на мобилни апликации со помош на Xamarin е C#.

Развитокот на Xamarin има повеќе големи значења за софтверската индустрија и за развивачите на софтвер за мобилни уреди. Пред да се развие овој проект развивачите на софтвер биле 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 е развиена 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 или не работат како што е предвидено.

Xamarin работи како обвивка на базичните библиотеки што ги нудат алатките за развивање на трите мобилни платформи. Тоа значи дека при компајлирање на напишаниот код во C# тој се преведува во базичен код за соодветната платформа. За користењето на овој софтвер да биде олеснето за развивачите коишто ги користат оперативните системи Linux и macOS, компанијата Xamarin има развиено интегрирана околина Xamarin Studio. Целта на Xamarin Studio e да ја замени околината за развој Visual Studio произведена од Microsoft којашто работи ексклузивно на Windows.

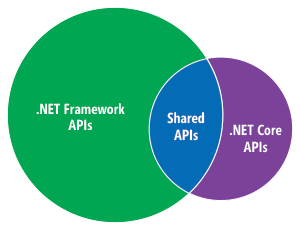


**Слика бр. 2 Xamarin архитектура на мобилни апликации**

Поради зголемениот интерес на заедницата на развивачи на софтвер за Xamarin платформата и студиото, Microsoft ја откупиле оваа компанија во февруари 2016 година. Оваа промена имала големо значење за корисниците затоа што освен развивање апликации со помош на Xamarin платформата, сега може да се развива било каков .NET софтвер во ова студио. Поради тоа што Xamarin студиото, исто како .NET Core платформата може да се користи на Linux и macOS, развивачите имаат можност да изградат .NET Core продукти на овие оперативни системи.

# Разлики на .NET Core и .NET Framework повици

Како што може да се забележи, за разлика од .NET Framework, новата рамка .NET Core може да се извршува на повеќе различни оперативни системи. За тоа да биде возможно потребно е да се исфрлат сите функционалности што се специфични само за оперативниот систем Windows. Најголем дел од исфрлените функционалности се технологиите за изградба на клиентски апликации Windows Forms и WPF. Поради тоа може да се заклучи дека .NET Core претставува подмножество на повици и технологии коишто се способни да се извршуваат на повеќе оперативни системи.



**Слика бр. 3 Разлика помеѓу .NET Core и .NET Framework**

# Компоненти

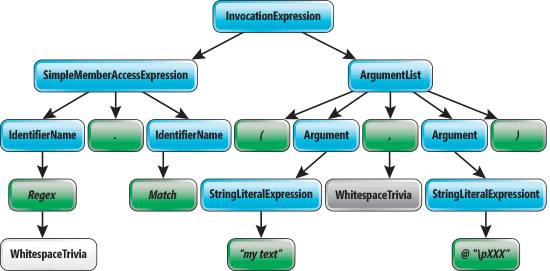
Развивање на софтвер со помош на рамката .NET Core е брзо, едноставно и лесно, но процесот на извршување на напишаниот код не е едноставен. Доколку се навлезе подлабоко во процесите на градење на софтверот како: компајлирање на кодот, динамичко поврзување на библиотеките, интегрирање со глобален репозиториум на библиотеки, извршувањето на напишаниот код и библиотеките, тестирање и пакување со цел оптимално прикачување на апликацијата на соодветната машина, брзо може да се заклучи 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. Начинот на работа на компајлерот Roslyn не е толку едноставен. 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 машинскиот код. Откако ќе се произведе средниот код, тој потоа се извршува со помош на специјална околина за извршување на среден јазик.

.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 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 се дел од .NET Core рамката иако не зависат од нејзините функционалности, а 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. Во .NET Core рамката тие се 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 го поддржува јазикот C# верзија 7.0.



**Слика бр. 4 Пример за визуелизација на синтаксички анализиран код**

При компајлирање на .NET проектите се повикува MSBuild алатката којашто го гради решението според стратегијата најдена во конфигурациска датотека. Во почетокот секоја C# датотека се компајлира посебно со помош на 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 датотеки во реално време (пример razor датотеки), при што тогаш се користи истиот Roslyn компајлер, но се повикува по потреба од страна на самата апликација при нејзино извршување.

## Алатка за градење софтвер – MSBuild

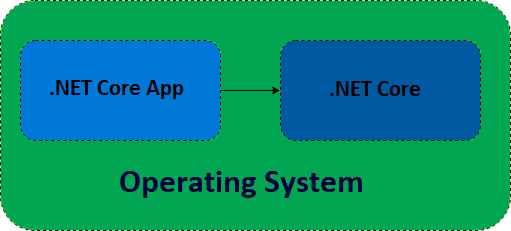
MSBuild (Microsoft Build Tools) е алатка за создавање, компајлирање, тестирање, пакување и прикачување на апликации и генерирање на документација. MSBuild е независен од Visual Studio, но Visual Studio го користи MSBuild во позадина за извршување на своите задачи. Тоа значи дека за да се изгради еден продукт не мора да се користи Visual Studio околината, туку само MSBuild алатката.

Од перспектива на развивачите на софтвер, конфигурирање на MSBuild алатката значи создавање на стратегија за градење на продукт. Попрецизно може да се забележи дека конфигурацијата на продукт е разделена во повеќе задачи коишто може да се однесуваат на било кој циклус од градењето на софтверот (компајлирање, тестирање, итн). Најчесто овие конфигурации се запишувале во соодветна .csproj датотека за секој проект, односно библиотеки и извршни апликации што ги сочинува целиот продукт.

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

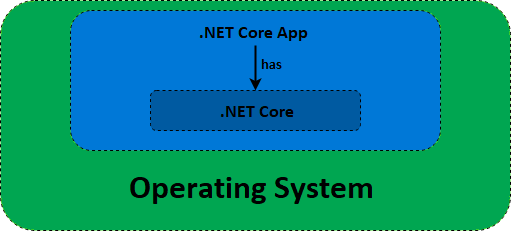
.NET Core апликациите може да се спакуваат и спремат за извршување на два различни начина. Првиот начин е да се искористи .NET рамка инсталирана на оперативниот систем и да се извршува со нејзина помош. Вториот начин е да се вклучи .NET рамката како дел од проектот при што иако корисниот ја нема инсталирано 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). Единствен параметар што мора да се предаде при градење на проектот е името на рамката од којашто зависи проектот, пример aspcore1.1. 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 без повторно да ја градиме апликацијата.



**Слика бр. 5 Апликација зависна од рамка**

Вториот начин се нарекува самостојно градење (анг. Self-contained deployments - SCD). Притоа при градење на продуктот се предава параметар што означува која верзија на извршување на рамката се користи. Доколку е потребно да се изгради апликацијата за Windows 10 x64 мора да се предаде соодветен 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. Поради тоа што целата рамка се наоѓа локално во продуктот, големината на целата апликација и бројот на датотеки е зголемен.



**Слика бр. 6 Апликација со вградена рамка**

## Алатки за командна линија – dotnet

Една од поголемите новости во .NET Core е алатката за командна линија (.NET Core CLI или само dotnet) 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).

Алатката DNX се состои од повеќе компоненти коишто сега со dotnet се заменети или надградени за полесна употреба. Иако примарната употреба на DNX била 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 платформи 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

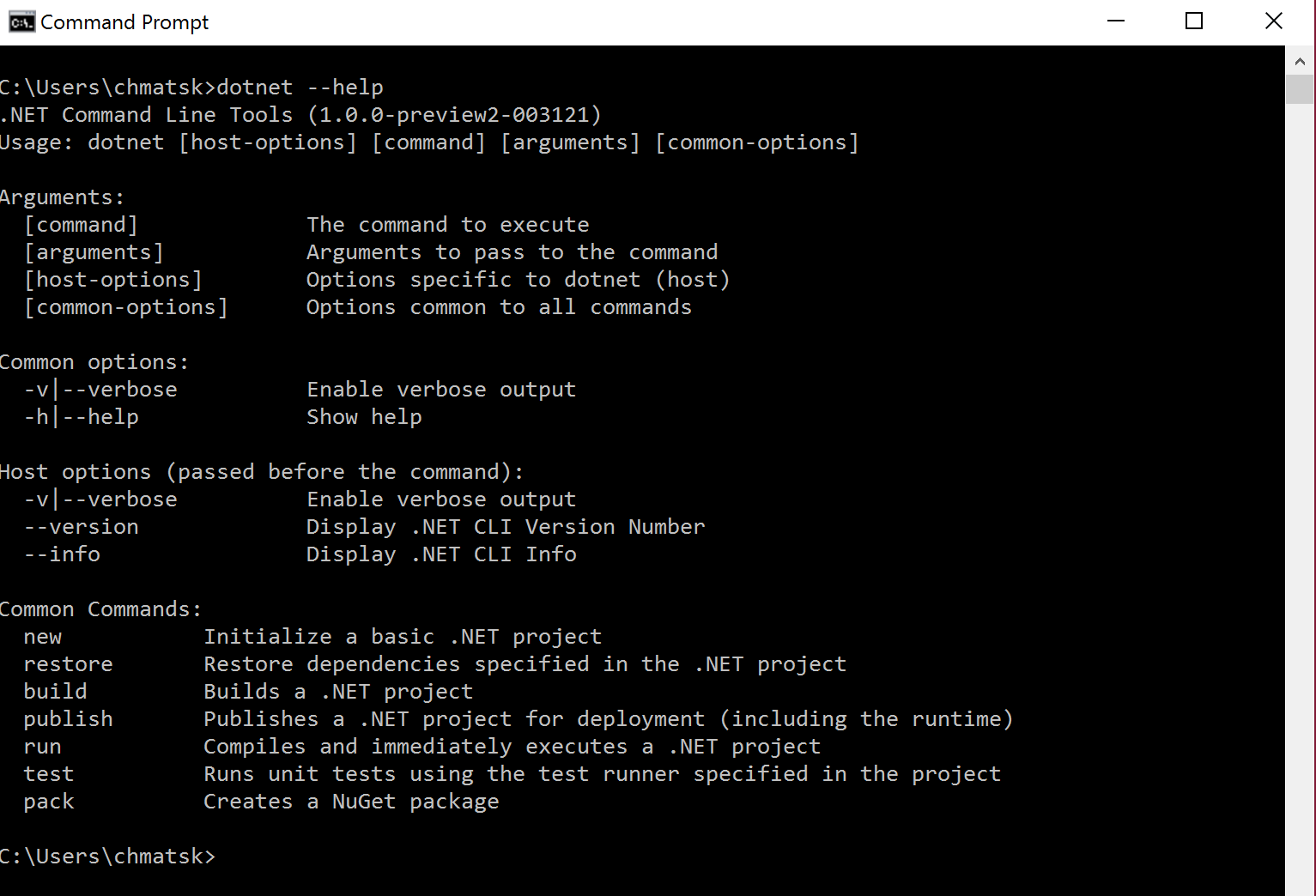
Командата “new” служи за креирање нов проект од типот наведен како следен аргумент. Синтаксата за искористување на оваа команда е dotnet new <Template> каде што <Template> може да се замени со: console, classlib, mstest, xunit, web, mvc, webapi, nugetconfig, webconfig, sln. Оваа команда креира нов проект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

Командата “build” служи за градење и компајлирање на проектот и библиотеките од коишто зависи при што се повикува алатката MSBuild. 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 oд типот апликации зависни од рамката(FDD) преку командна линија. За оваа команда може да се специфицираат верзија на рамка и датотека на конфигурација.

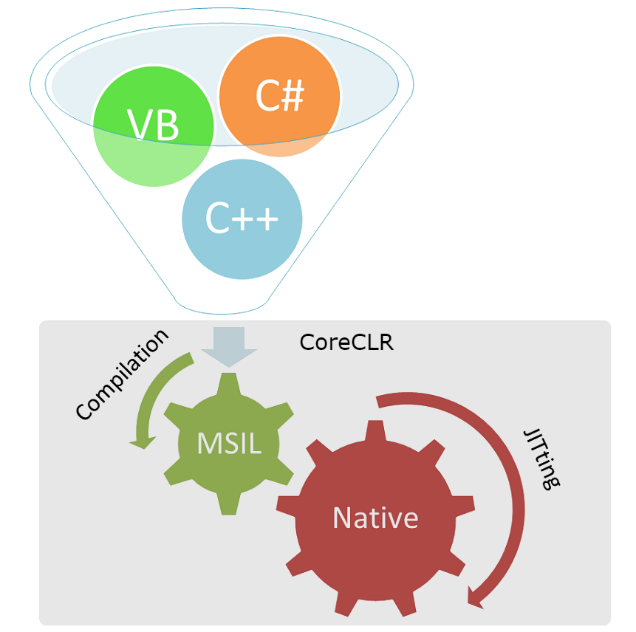


**Слика бр. 7 Пример излез на .NET Core CLI при извршување на командата --help**

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

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

Како еден од најзначајните новости на .NET рамката е CoreCLR. Оваа компонента на .NET Core рамката 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 машински код.



**Слика бр. 8 Пат на преведување и извршување на C#, VB и VS C++ 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. 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:

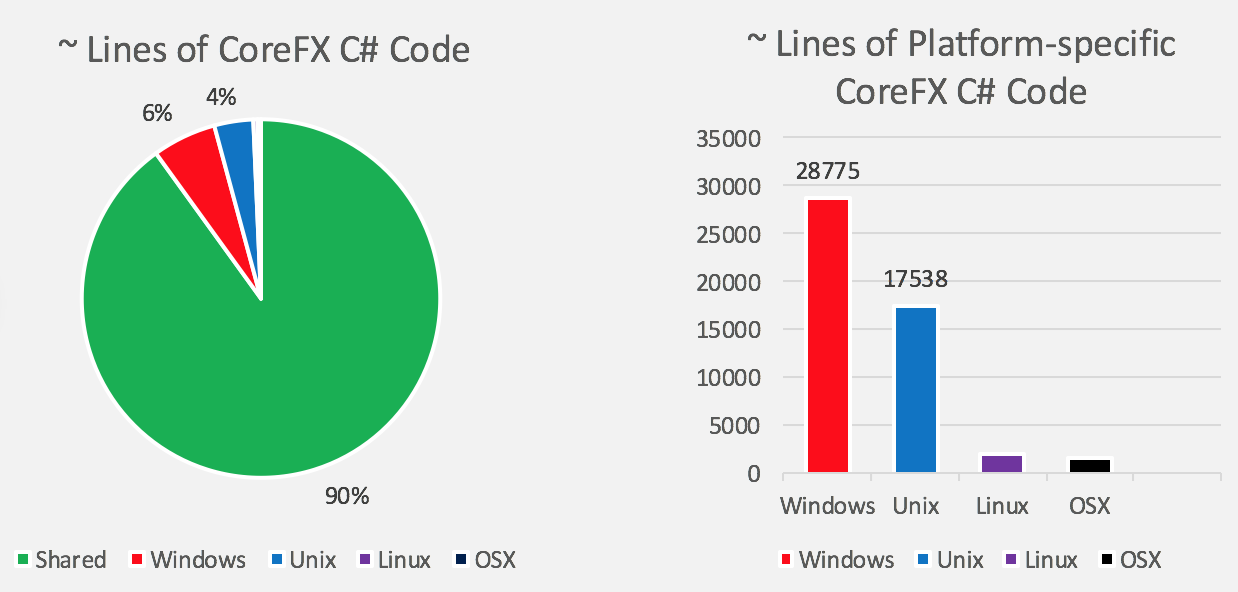
* компајлер во реално време (JIT) за преведување на код во среден јазик во машински код
* асемблер и дисасемблер за средниот јазик
* обвивка за процесирање на динамички поврзани библиотеки
* јавна библиотека за развој врз CoreCLR

## Библиотеки – CoreFX

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

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.

Како дел од CoreFX иако работата на најголемиот број на библиотеки не зависи од оперативниот систем, може да се забележи 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.

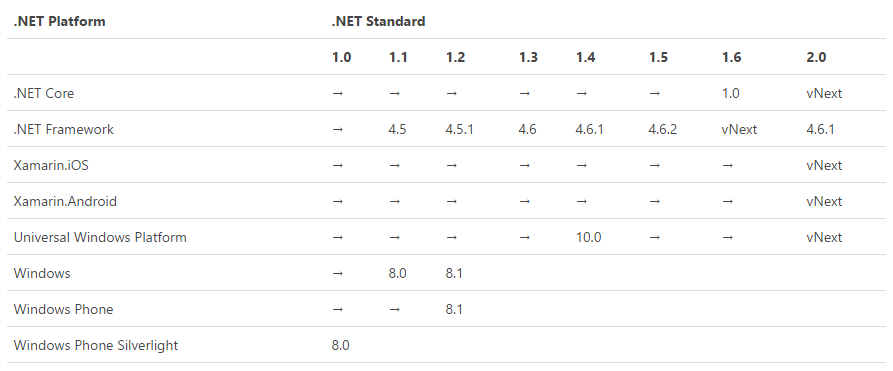


**Слика бр. 9 Број на линии код во CoreFX и нивна распределба по оперативни системи**

## .NET Стандард

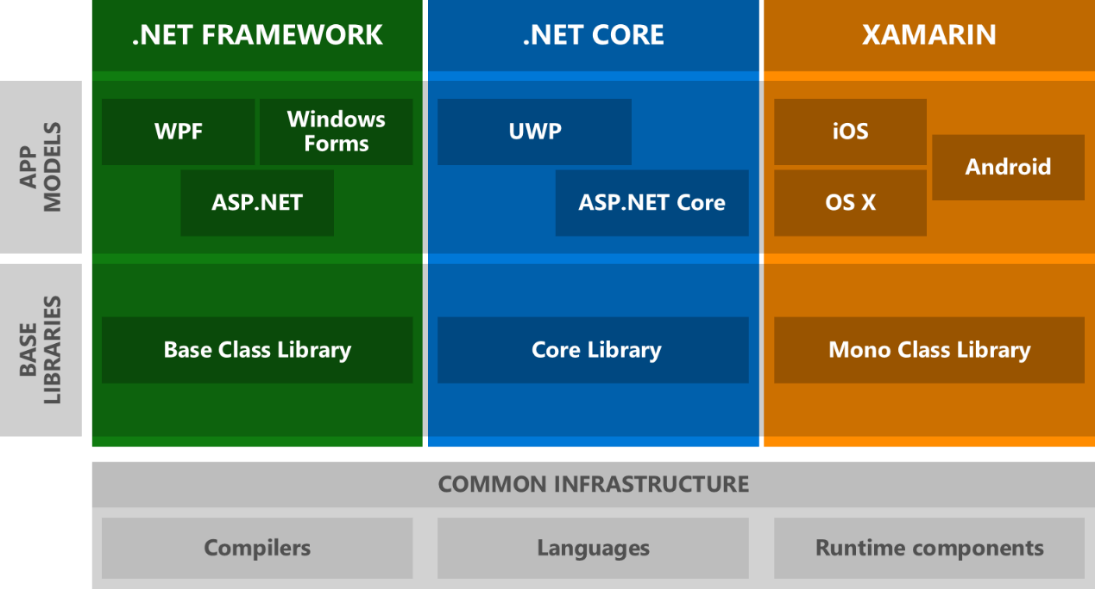
Со цел да може да се извршува .NET Core апликација на било кој оперативен систем, корисникот мора да се осигура дека .NET Core рамката е присутна. Оваа библиотека ги имплементира сите базични функционалности за секој оперативен систем, но постојат и функционалности коишто се специфични за еден оперативниот систем, а не се достапни на некој друг. Поради ова возможно е 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 вовеле 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. Toa значи дека секој оперативен систем којшто ја има инсталирано .NET Core 1.0 рамката ќе може да извршува апликации, односно библиотеки што ги користат апликациите изградени според .NET Standard 1.6.



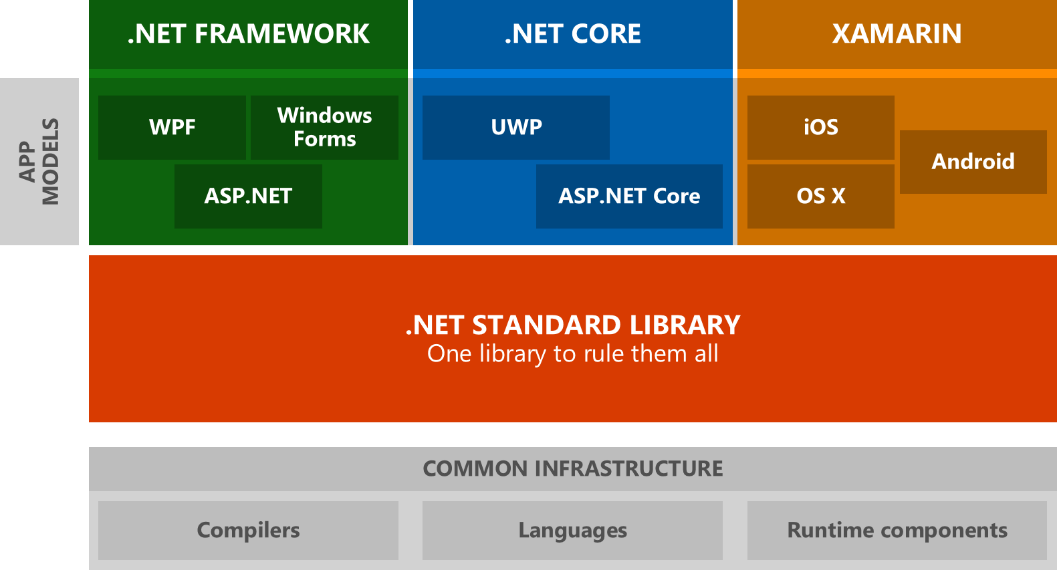
**Слика бр. 10 Поддржани верзии на .NET Standard**

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. Тогаш би се јавил проблем за интеграција на сопствена библиотека во сите три рамки одеднаш. Овој проблем бил 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.



**Слика бр. 11 Проблематичен пристап за работа доколку не постои .NET Standard**

Овој стандард е компатибилен со сите свои постари верзии. Тоа 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.

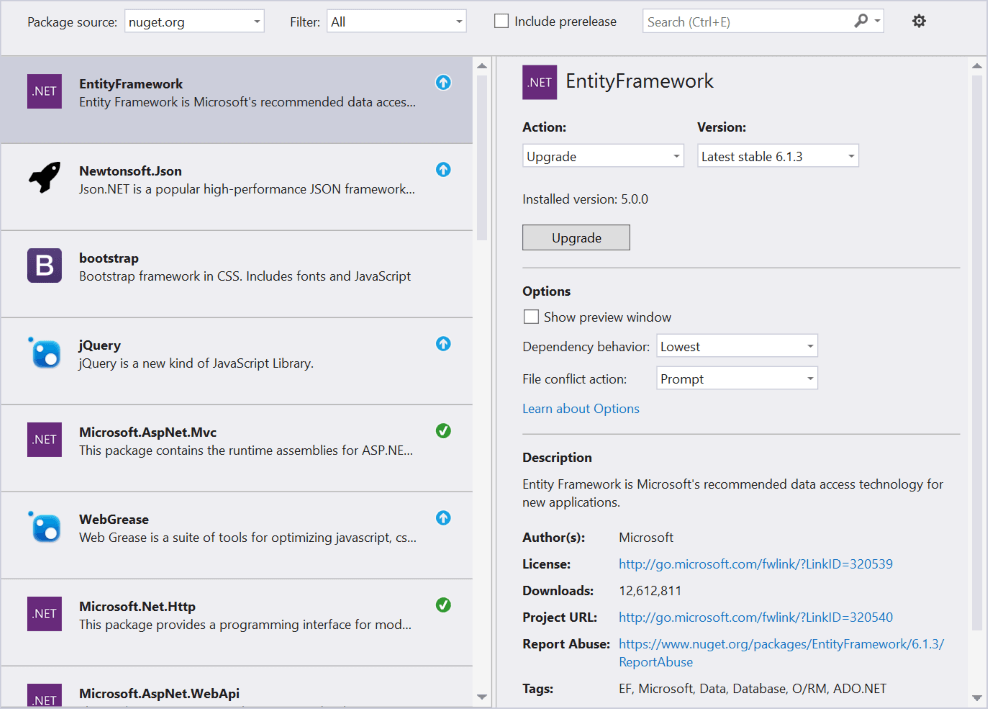


**Слика бр. 12 Архитектура на .NET рамките**

## Менаџер на пакети – NuGet

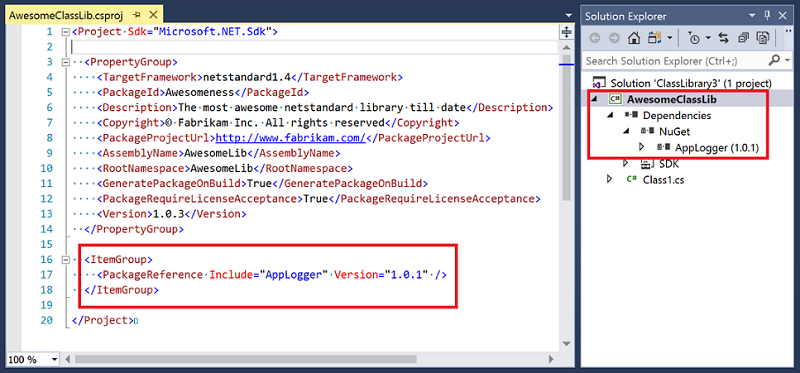
При развој на една .NET апликација потребно е да се користат најразлични библиотеки кои решаваат чести проблеми. За полесен пристап до овие библиотеки развивачите на .NET платформата нудат менаџер за пакети наречен NuGet. Најосновната функција на NuGet е пребарување и консумирање на библиотеки наречени NuGet пакети. Овој менаџер се користи и за проверка и промена на верзиите на веќе спуштените пакети, нивно консолидирање низ повеќе проекти и менаџмент на целната верзија на .NET рамката. Исто така доколку некој сака да изгради наши сопствен пакет за глобална консумација може да го направи со помош на оваа алатка.

Пред официјалното пуштање во употреба на Visual Studio 2017, NuGet менаџерот бил 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 запишувал во packages.config датотека.



**Слика бр. 13 Кориснички интерфејс на NuGet менаџерот**

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. Во новата верзија 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 новата верзија пакетите не се симнуваат повеќе пати при промена на верзијата на пакетот и не се прикажуваат пакетите коишто не се од директна корист за проект што се гради.



**Слика бр. 14 Нов начин на запишување на референци кон пакети**

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

ASP.NET е платформа 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. ASP.NET Core е новата генерација на ASP.NET платформата којашто има многубројни новини во својата структура. За разлика од својот претходник ASP.NET MVC 5 којшто се извршува само со .NET Framework рамката, оваа платформа може да се извршува и со .NET Core рамката. Поради тоа ASP.NET Core претставува прва веб платформа официјално поддржана од Microsoft којашто може да се хостира на Linux и macOS сервер.

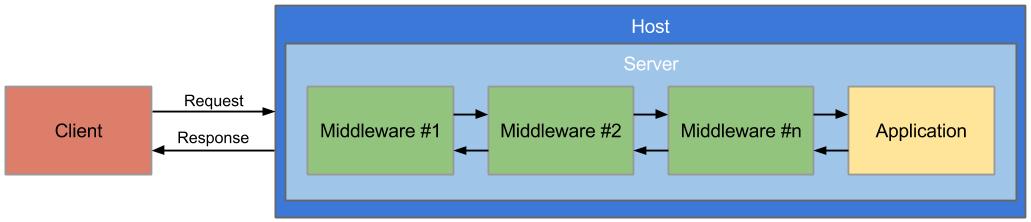
Доколку се направи споредба со ASP.NET MVC 5 платформата може да се забележиме дека новата платформа има вклучен сервер Kestrel во самиот процес на апликацијата. Овој нов сервер лесно може да се интегрира со IIS сервер, додека старата платформа има тесна интеграција со IIS сервер. Ова претставува проблем затоа што со MVC 5 платформата развивачите се принудени да ја хостираат апликацијата на IIS сервер и Windows оперативен систем.

За разлика од ASP.NET MVC 5, ASP.NET Core го имплементира OWIN стандардот во своја имплементација на слоевита архитектура, но нуди и лесна интеграција со базичната OWIN библиотека. Додека за развој на WebApi, со старата верзија на ,NET Framework морало да се изработи нов проект, со ASP.NET Core може да се постигне истиот резултат во истиот проект. ASP.NET Core претставува унификација на WebApi и MVC парадигмата и со едноставна конфигурација може да постојат двете решенија во ист проект. Оваа надградба ја елиминира потребата да се хостираат две различни решенија за извршување на MVC и WebApi решенија.

### OWIN

OWIN преставува отворен веб стандард за .NET. Целта на OWIN е да го раздвои целниот сервер од апликацијата што се извршува на него, со тоа што ќе овозможи комуникација меѓу двете преку стандарден интерфејс. Во пракса за развивачите на софтвер, OWIN игра голема улога при изградба на библиотеки коишто имаат интеракција со корисничките барања до веб апликација.

OWIN претставува платформа за изградба на апликација со слоевита архитектура. Секој слој има контрола на целиот контекст на корисничкото барање и може да одлучи дали има потреба да се повика наредниот слој.



**Слика бр. 15 Слоевита архитектура на апликација со OWIN**

Пример доколку треба да се запишува секое корисничко барање со IP адресата и времето на барање, може да се изгради OWIN слој којшто ќе ја извршува оваа функција. Со ова решение сите кориснички барања ќе го извршуваат новиот слој, освен ако некој претходен слој не го запре понатамошното извршување на некое корисничко барање. Добрата страна на овој пристап е што оваа функционалност може да се искористи на секоја платформа што го имплементира OWIN стандардот, како што се ASP.NET MVC 5, ASP.NET Core или базичната OWIN библиотеката со сопствен веб сервер.

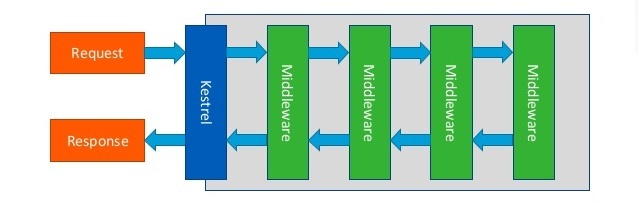
Доколку се направи споредба помеѓу ASP.NET MVC 5, ASP.NET WebApi 2 и ASP.NET Core може да се забележи дека иако сите го имплементираат OWIN стандардот, разликата во неговата интеграција е тоа што:

* ASP.NET MVC 5 има интеграција за прифаќање на кориснички барања со IIS серверот и се пренасочуваат кон MVC и OWIN
* ASP.NET Core го имплементира OWIN стандардот и користи слоевита архитектура, но не ја користи базичната OWIN библиотека туку има можност за адаптирање кон истата
* ASP.NET WebApi2 ја користи базичната OWIN библиотека и самата платформа претставува слој кон таа библиотека со што се нуди можност ASP.NET WebApi 2 да се хостира како конзолна апликација со OWIN веб сервер

Иако имплементацијата на OWIN библиотеката е различна во сите три платформи, користење на сопствена библиотека што претставува OWIN слој е возможна во секоја од овие платформи.

### Kestrel

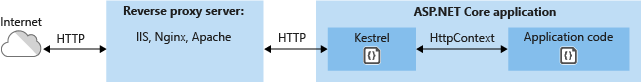
При развој на веб апликација потребен е сервер за хостирање за да се овозможи јавен пристап на веб апликацијата до потенцијалните клиенти. При хостинг на стандардна ASP.NET апликација мора да се користи IIS (анг. Internet Information Services) сервер стриктно на Windows оперативен систем. Овој проблем се јавува затоа што самата ASP.NET платформа не знае како да се справи со било кој формат на клиентско барање туку само со барање опслужено од IIS сервер. За да се реши овој проблем би можела да се адаптира платформата на повеќе веб сервери, но сепак тоа би претставувал макотрпен процес поради големиот број на различни сервери. При хостирање на ASP.NET Core апликација ова ограничување не постои токму заради Kestrel.



**Слика бр. 16 Kestrel и OWIN во ASP.NET Core апликација**

Kestrel претставува веб сервер извршуван во процесот на самата ASP.NET Core веб апликација. Овој сервер е базиран на асинхроната библиотека libuv за процесирање на I/O барања компатибилна со повеќе оперативни системи. Libuv е библиотека развиена од Google за употреба во NodeJS серверот. ASP.NET Core платформата знае како да се справи со клиентски барања опслужени од Kestrel со што овозможува веб апликацијата да се постави на било кој оперативен систем што поддржува .NET Core апликации. Kestrel знае како да се справи со HTTP/S и WebSocket барања и е оптимизиран за работа со Unix сокети и Nginx серверот.

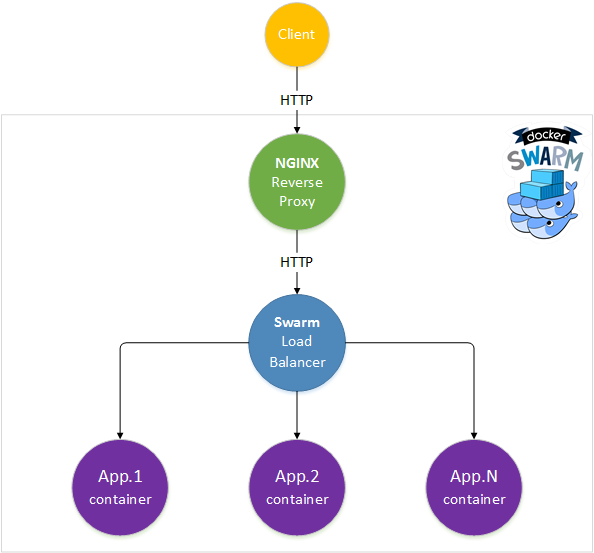
Тимот на Microsoft препорачува да не се користи Kestrel серверот за јавен пристап до веб апликацијата туку да се користи некој друг обратен прокси сервер којшто ќе ги пренасочува сите клиентски HTTP/S барања до Kestrel. За разлика од класичните веб сервери Kestrel не нуди заштита од напади од типот на премногу барања од еден клиент, преголеми барања или преголеми датотеки, менаџмент и лимитирање на достапни процесорски нишки. Токму поради овие проблеми се препорачува некој друг јавен сервер кој може да се справи со овие проблеми и ги пренасочува само валидните барања до Kestrel серверот.



**Слика бр.. 17 Архитектура на хостинг со обратен прокси сервер**

### Docker

Многу често при развој на апликации се случува одредена функционалност којашто зависи од некој сервис да работи на еден компјутер, но да не работи на друг. Овие проблеми најчесто се јавуваат поради различна верзија или конфигурација на сервисот од кој што зависи одредената функционалност. Едно решение за да се елиминира овој проблем е да се доставуваат сервисите и нивната конфигурација заедно со апликацијата. Docker претставува контејнер кој ја овозможува оваа практика за се извршува без помош на виртуелна машина, освен на Windows оперативен систем каде што е потреба Hyper-V виртуелна машина. За разлика од виртуелните машини, контејнерите не доаѓаат заедно со цел оперативен систем, туку само со библиотеки коишто ја изолираат апликацијата од оперативниот систем и ги менаџираат ресурсите потребни за да работи правилно.



**Слика бр. 18 Апликации хостирани во Docker контејнери**

Docker е поддржан на повеќе оперативни системи од кои Windows, повеќето дистрибуции на Linux и macOS се најпопуларни, но исто така имаат посебни интеграции со AWS и Azure сервисите. ASP.NET Core има можност да се извршува преку Docker контејнер со што може да се намалат проблемите при развој на веб апликација на различни оперативни системи. Освен самата апликација може да се дефинираат и база на податоци, сервис за кеширање, обратен прокси сервер и други потребни сервиси за успешна изградба на апликацијата, притоа развивачите не мора да се грижат за оперативниот систем на којшто се извршуваат сите овие сервиси и самата веб апликација.

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

Со помош на .NET рамката можат да се развијат повеќе видови на апликации, но една од најважните платформи е ASP.NET која се користи за развој на веб апликации. Доколку се направи споредба помеѓу развојот на апликации изработени во ASP.NET MVC 5, што претставува последна итерација на ASP.NET платформата што работи со .NET Framework рамката и ASP.NET Core 1.0 што работи со .NET Core 1.0 рамката, може да се забележи дека се направени големи промени, но базичните функционалности се непроменети.

ASP.NET, без разлика на верзијата, претставува платформа за изградба на веб страни со помош на HTML, CSS и JavaScript. Овие компоненти се интерпретираат и сервираат од страна на платформата со MVC (Model – View - Controller) архитектура. Двете платформи ја задржуваат оваа структура и начин на работа со податоците и барањата од клиентите. Покрај овие сличности направени се неколку промени на поновата платформа ASP.NET Core во работата со NuGet пакети, почетокот на апликацијата, како и начинот на којшто се хостира самата апликација.

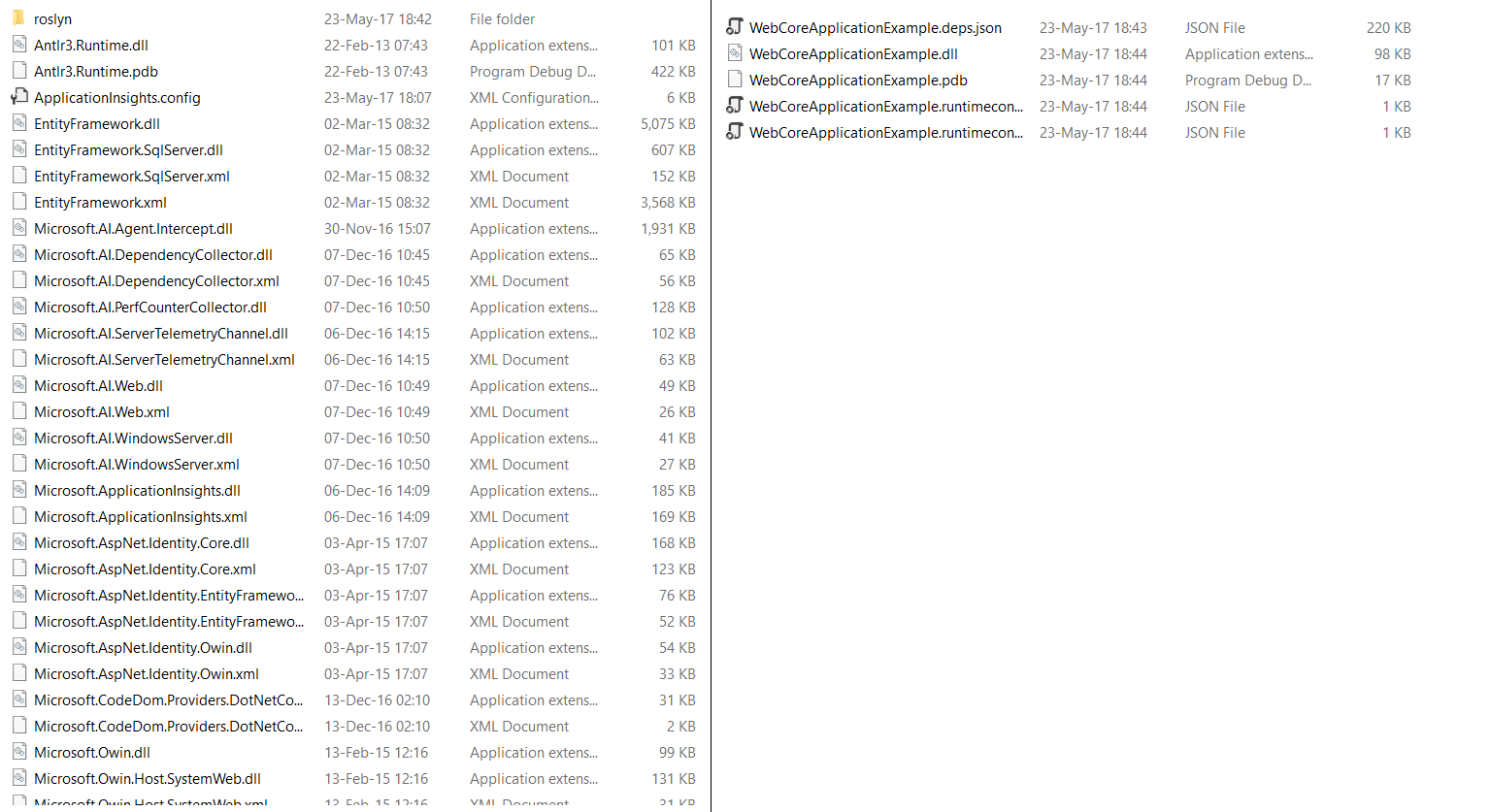
## NuGet пакети

При самата иницијализација на проектите со готовите шаблони за MVC проекти со автентикација и авторизација може да се забележи разлика во NuGet пакетите коишто се користат во двата проекти. Во MVC 5 проектот се вклучени сите пакети од .NET Framework, дополнителни пакети за MVC платформата и библиотеки за работа со база на податоци. За разлика од MVC 5, ASP.NET Core MVC има поделба во типот на потребните пакети. Едниот тип се SDK пакети коишто ја сочинуваат самата .NET Core рамка, а другиот тип се MVC пакетите за ASP.NET Core и пакетите за работа со база на податоци. Оваа поделба е направена затоа што самата .NET Core рамка, којашто припаѓа во SDK пакетите, е присутна на машината на којашто се развива продуктот.

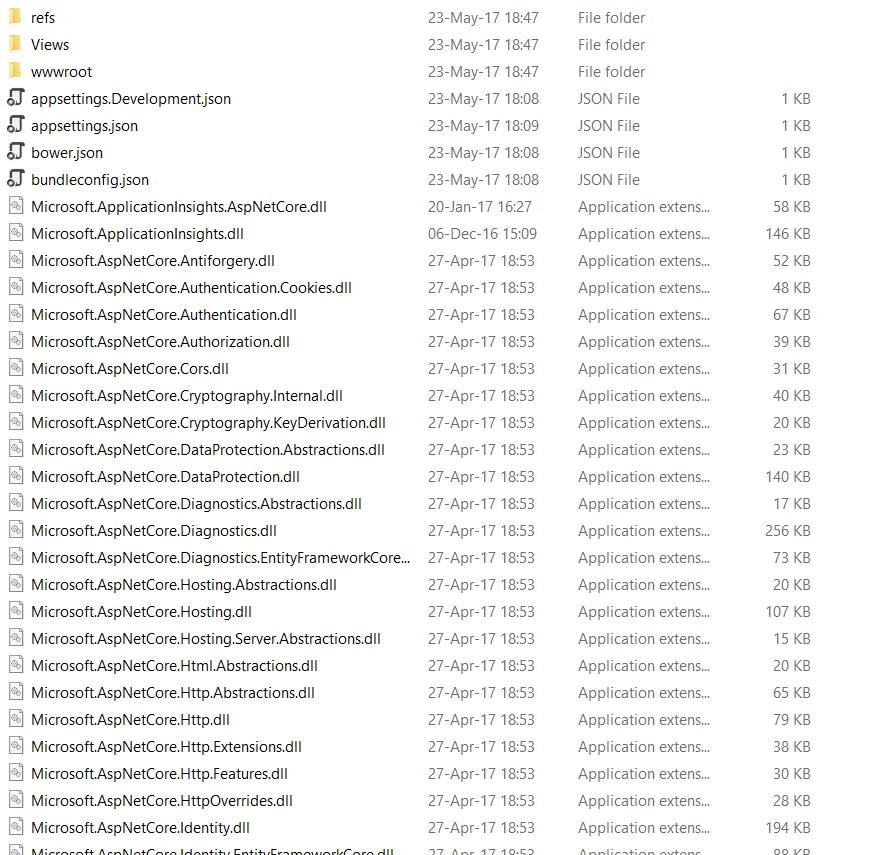
Поради тоа што има два типа на градење на .NET Core продукт, излезот од компајлирањето на кодот е различен за секој тип. Доколку сакаме ASP.NET Core MVC продуктот да ја содржи .NET Core рамката во себе тогаш излезот што го добиваме е сличен со излезот од ASP.NET MVC 5, односно ги сочинува сите .NET Core и NuGet пакети. Доколку сакаме да изградиме продукт зависен од .NET Core рамката што е присутна на машината на која што се извршува тогаш како излез добиваме само една .dll датотека и неколку конфигурациски датотеки за потребните NuGet пакети. При извршување на оваа апликација NuGet пакетите и .NET Core рамката се симнуваат во глобален репозиториум на извршната машина и се користат директно од таа локација.



**Слика бр. 19 NuGet пакети во ASP.NET MVC 5 проект (лево) и NuGet пакети во ASP.NET Core MVC (десно)**



**Слика бр. 20 излез од ASP.NET MVC 5 проект (лево) и излез од FDD ASP.NET Core MVC (десно)**



**Слика бр. 21 излез од SCD ASP.NET Core MVC**

## Почеток на апликацијата

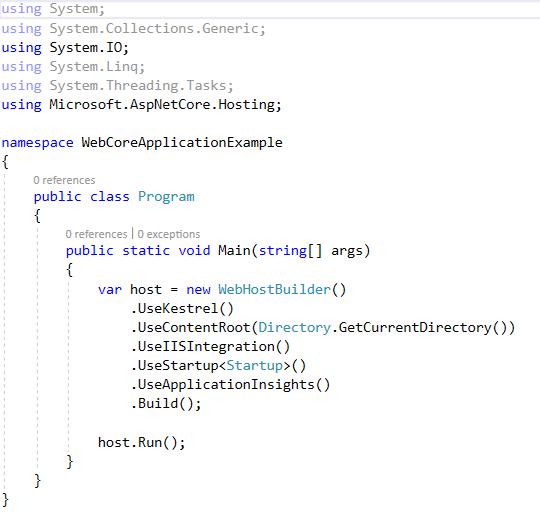
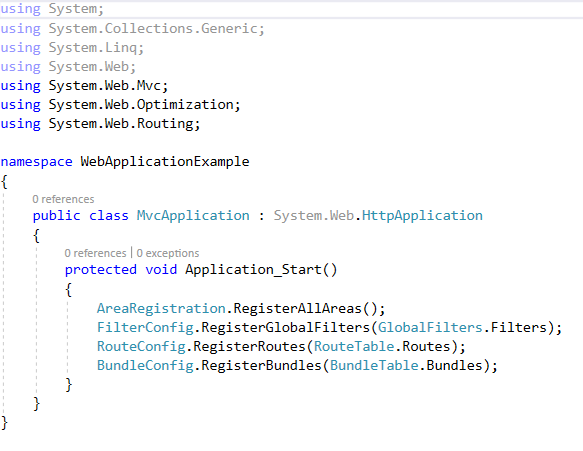
Како и секоја апликација така и веб страните мора да имаат некоја почетна извршна датотека. ASP.NET MVC 5 платформата најпрво започнува со извршување на датотеката Global.asax, додека пак OWIN започнува во Startup.cs одкако ќе се изврши Global.asax. Доколку внимателно се разгледа структурата на Global.asax може да се забележи дека нема стандардна структура, туку има магични функции кои се повикуваат директно од страна на IIS серверот. Токму поради оваа причина ASP.NET MVC апликациите може да се извршуваат само на IIS сервер. Доколку се забележи во стандардниот MVC 5 шаблон, при стартување на апликацијата се конфигурираат рути, филтри и оптимизации на клиентски библиотеки. Постојат повеќе магични методи што може да се имплементираат коишто одговараат на сите животни циклуси на IIS серверот.[[1]](#_Референци)

Новата платформа ASP.NET Core MVC започнува со извршување како и секоја друга апликација, со стартување на функцијата Main. Во оваа функција дефинирани се стартната класа на апликацијата (Startup), серверот што се користи (Kestrel) и останати алатки за мониторирање на перформанси на апликацијата. По конфигурацијата овој сервер се стартува, што го означува почетокот на извршување на Startup класата и прифаќање на кориснички барања. Во Startup класата можат да се забележат конфигурации за целата MVC апликација, односно:

* конфигурирање на профил на извршна околина (Development или Production)
* конфигурација за зависности
* конфигурација на автентикација и авторизација
* конфигурација на MVC контролите и рутите

Сите овие конфигурации се изгенерирани од MVC Core шаблонот, но не мора да се користат на тој начин, туку тие претставуваат најчеста конфигурација на почетен проект.

Поради тоа што во .NET Core MVC се користи серверот Kestrel како приватен сервер, веб апликацијата со .NET Core платформа може да се извршува на било кој јавен сервер, при што тој сервер ќе ги пренасочува валидните кориснички барања до Kestrel. Доколку развивачот сепак сака да ја хостира својата апликација на IIS сервер, развивачите на Microsoft нудат конфигурација при што го адаптираат Kestrel за работа со IIS серверот.



**Слика бр. 22 Стартна датотека: Core MVC лево, MVC 5 десно**

## Инверзија на контрола

Како што може да се забележи во Startup класата на ASP.NET Core MVC апликацијата постои конфигурација на контејнерот за инверзија на контрола. Но уште поголема новост е тоа што може да се употреби инјекција на зависност во самата Startup класа на интеграции што се достапни при самото стартување на апликацијата. Тие класи се најчесто за конфигурација на околината на серверот или околината на целата апликација. По конфигурирање на сопствени интеграции на сервиси, тие може да се пристапат низ целата апликација преку инјектирање.

Поради тоа што може да се инјектираат зависности уште пред да се конфигурира било што во целата апликација, може да се заклучи дека постои некој стандарден контејнер на зависности во ASP.NET Core MVC апликацијата. Имплементација на ваков контејнер не постои во ASP.NET MVC 5 платформата, но постои доставувач на зависности, при што би користел контејнер од некоја библиотека што го имплементира соодветниот интерфејс. Во ASP.NET Core MVC стандардниот контејнер за зависности може да се замени со некоја библиотека и да се користи таа имплементација на контејнер.

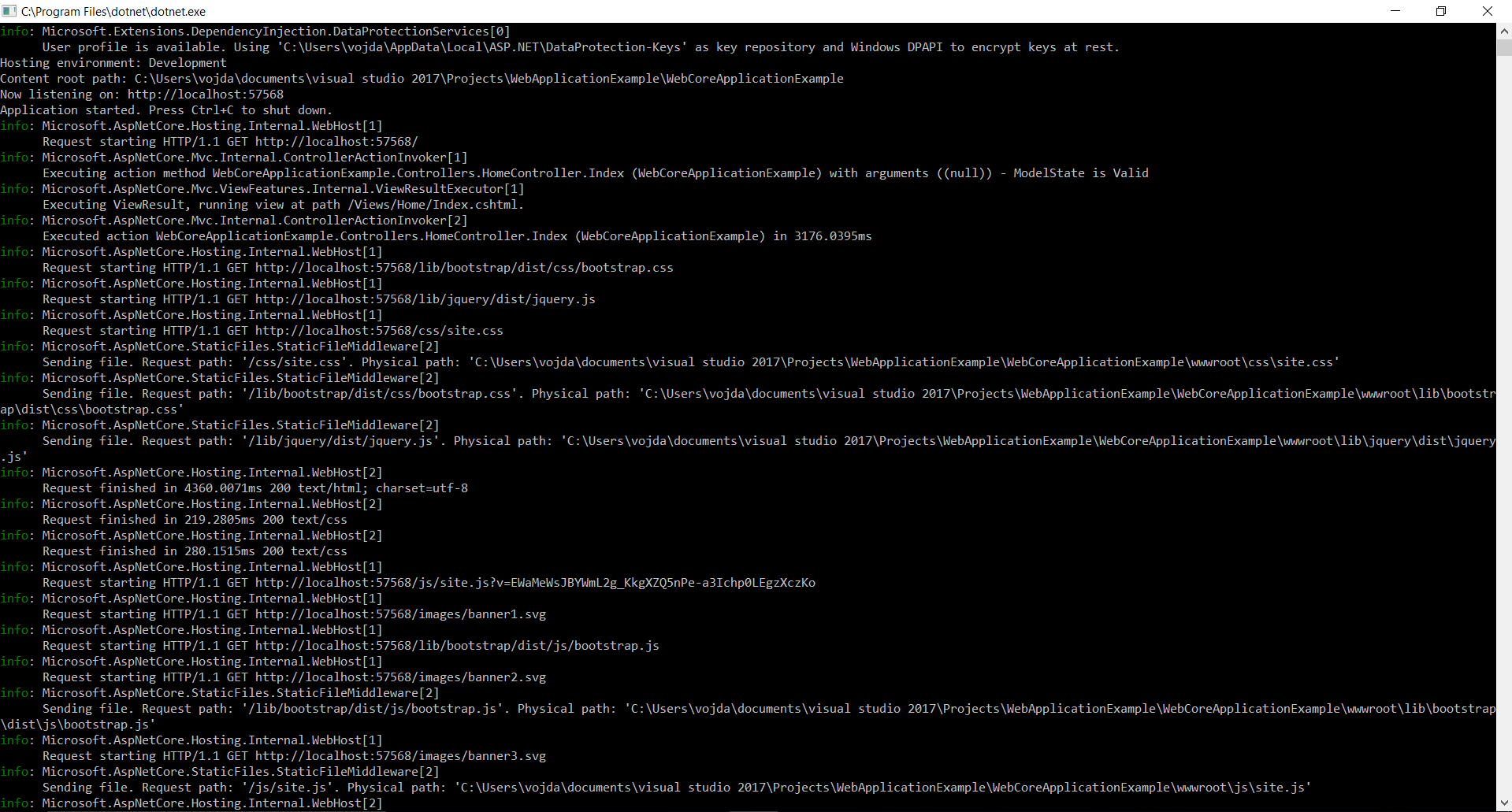


**Слика бр. 23 инјектирање на околина за хостирање и конфигурација на испраќачи на е-пошта и смс**

## Начин на хостирање

Доколку се направи споредба на ASP.NET Core MVC платформата со MVC 5 и WebApi 2 може да се забележат повеќе разлики во нивната функционалност и хостирање. Поради тоа што MVC 5 и WebApi 2 се различни платформи со поделени библиотеки за развој, нивниот начин на хостирање не е меѓусебно зависен, без разлика што се водат според истиот MVC шаблон. WebApi 2 претставува слој во OWIN архитектурата и поради тоа може да се хостира на сите начини на кои што може да се хостира било кој OWIN проект. Најчести начини за хостирање на WebApi 2 проект се: на IIS сервер, како конзолна апликација со свој сервер во меморија или на било кој сервер што го имплементира OWIN стандардот. MVC 5 претставува платформа што е независна од OWIN, но може да ја користи неговата слоевита архитектура. Поради тоа MVC 5 мора да се хостира ексклузивно на IIS сервер и да ги користи магичните методи што ги нуди IIS за пристап до животниот циклус на апликацијата.

ASP.NET Core MVC го имплементира OWIN стандардот, но не претставува слој во базичната OWIN библиотека, туку користи своја имплементација. За разлика од MVC 5 и WebApi 2, ASP.NET Core MVC може да се хостира како засебна конзолна апликација со свој сервер во меморија наречен Kestrel. Оваа рамка нуди можност за интеграција со IIS сервер, но и може да се дозволи јавен пристап до апликацијата преку инверзен прокси сервер.



**Слика бр. 24 Пример на ASP.NET Core MVC апликација хостирана како конзолна апликација**

# Заклучок

Користењето на Microsoft технологиите во софтверскиот свет станува се почесто и поефикасно решение за фирмите коишто се занимаваат со развивање на продукти во сферата на софтверот. Платформите за изработка на софтвер на Microsoft нудат решенија за сите типови на апликации, но со новата .NET рамка овие решенија можат да се искористат на повеќе оперативни системи.

Иако .NET Core претставува нова технологија, нејзината моќност може да се спореди и со најдобрите рамки за развој на софтвер. Продуктите изработени со оваа технологија се со подобри перформанси од продуктите изработени со постарите генерации на .NET рамката. Креирањето на вакви брзи и квалитетни решенија помага на компаниите да се развијат и да понудат поквалитетна услуга на своите клиенти.

Освен што .NET Core претставува оптимизирана верзија на нејзините претходници, оваа рамка има официјална поддршка од Microsoft и од големата заедница на корисници. За развивачите на софтвер оваа рамка нуди брз и стабилен развој на веб и клиентски апликации од било која големина, а особено за поголемите софтверски решенија.

# Референци

1. <http://www.c-sharpcorner.com/blogs/globalasax-events1>