ИКОНОМИЧЕСКИ УНИВЕРСИТЕТ - ВАРНА

КАТЕДРА „ИНФОРМАТИКА”

докт. Йордан Иванов Йорданов

Реферат на тема:

РАЗРАБОТКА НА ОНЛАЙН МАГАЗИН ЧРЕЗ ASP.NET CORE MVC, БАЗИРАН НА ОБЛАЧНА ПЛАТФОРМА MICROSOFT AZURE

по дисциплина „*Езици за програмиране*“

Варна 2021

Съдържание

Въведение

В последните години облачните технологии се превърнаха във водеща тенденция в софтуерната индустрия. Те предоставят нов начин за изграждане на големи и сложни системи, като по този начин използват пълноценно съвременните практики за разработка на високо-качествен софтуер и налична инфраструктура. Това променя начина на проектиране, интегриране и внедряване на системите. Облачно базираните решения са проектирани да приемат бързо промените, да обслужват голям мащаб от хора и да бъдат устойчиви на всякакъв вид натоварване или хакерски атаки.

Организацията Cloud Native Computing Foundation[1] предлага следното определение: "*Технологиите, базирани на облак, дават възможност на организациите да създават и изпълняват приложения в модерни, динамични среди като публични, частни и хибридни облаци, чрез мрежи от услуги и микроуслуги. Качества на системите са устойчивост, висока наличност и достъпност, мащабируемост и управляемост, които са от критично значение за много от бизнес единиците. Автоматизацията на тези процеси позволява на инженерите да правят промени, с голямо въздействие, но с минимални усилия."*

Приложенията стават все по-сложни, като изискванията, от страна на потребителите, стават все повече и повече, главно насочени към бърза реакция и иновативни функции. Проблеми с производителността или повтарящи се грешки вече не са приемливи.

Предимствата на облачните системи поставят бизнеса една стъпка пред конкурентите. Бизнес системите се развиват от способностите на бизнеса да бъдат инструменти за стратегическа трансформация, която ускорява растежа на компанията. Облачно базираните системи се свързват главно с бързина. Незабавното пускане на иновативните идеи на пазара е важна тема за всички модерни компании, например следните компании са приложили

[1] Cloud Native Computing Foundation. е проект на Linux Foundation, основан през 2015 г., за да подпомогне развитието на контейнерните технологии и да приведе технологичната индустрия в еволюцията си.

успешно тези техники:

- Netflix[2] има над 600 услуги в производствена среда. Стотици пъти на ден се изпълняват нови внедрявания и разгръщания на съществуващи.

- Uber[3] има над 1000 услуги в производствена среда. Обновяват се няколко хиляди пъти всяка седмица.

Както е видно, бизнесът на тези две компании се базира на системи, които се състоят от стотици независими микроуслуги. Този архитектурен стил им позволява бързо да реагират на пазарните условия като постоянно актуализират малки, но важни области. Скоростта и пъргавината на облачния носител се дължат на редица фактори, като на първо място е инфраструктурата на изчислителните ресурси.

По примери и указания на водещи експерти от общността, представяме характеристики и изисквания на функционален облачен продукт, демонстриращ използването на .NET, Docker, Kubernetes в облачната среда на Microsoft Azure за осъществяването на опростен, във функционално отношение, онлайн магазин. Ето някои от основните системни изисквания[4], които магазинът има:

- Колекция от артикули, между които може да се избира определен
- Филтриране на елементите по тип
- Филтриране на артикулите по марка
- Добавяне на артикули в кошницата за пазаруване
- Промяна или премахване на артикули от кошницата
- Разглеждане на детайлите за определен елемент
- Регистриране на акаунт
- Вписване на потребител
- Отписване на потребител
- Преглеждане на текущите поръчки

[2] Как Netflix внедрява код, Infoq, 2013 <https://www.infoq.com/news/2013/06/netflix/>

[3] Микро внедряване на Uber Engineering, Eng.Uber, 2013, <https://eng.uber.com/micro-deploy-code/>

[4] Изброените системни изисквания представляват задачи от високо ниво, за това, какво системата трябва да прави. Обикновено се предоставя от функционален анализатор.

Приложението има и следните нефункционални изисквания[5]:

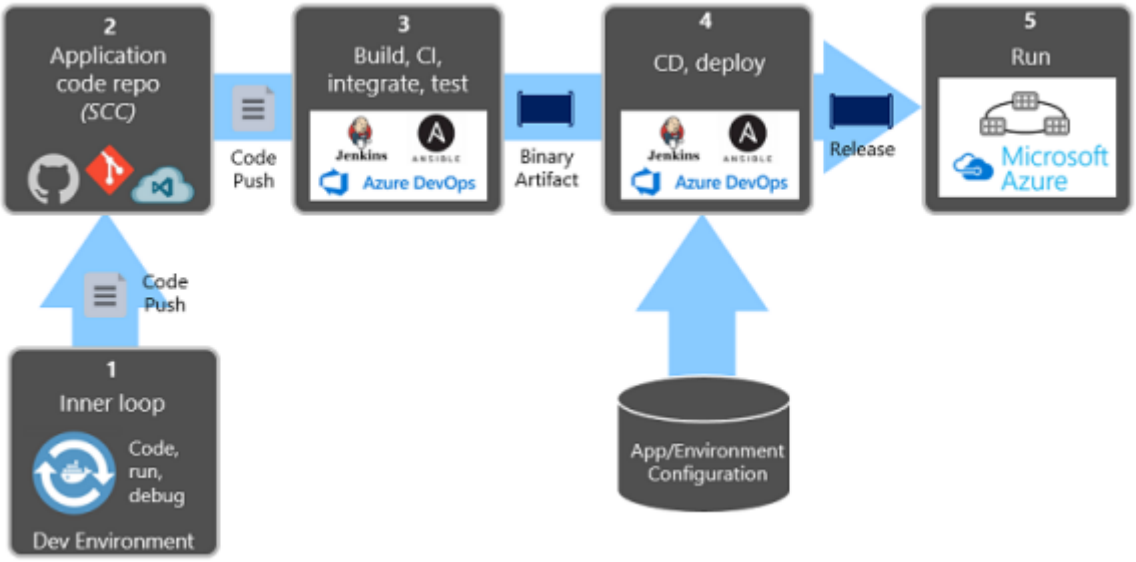
• Трябва да е високо-достъпно и да може автоматично да разширява мащаба, за да отговори на увеличаващия се трафик (също така да намалява мащаба, след като трафикът спадне).

• Трябва да осигурява лесен за използване мониторинг на състоянието на системните единици и диагностични дневници, за да помогне при отстраняване на неизправности или други проблеми, които възникнат по време на работа.

• Трябва да поддържа гъвкав процес на развитие, включително подкрепа за непрекъсната интеграция и внедряване (Continuous integration / deployment).

• Трябва да поддържа уеб интерфейс (традиционно, едностранично и/или мобилно клиентско приложение)

• Трябва да поддържа междуплатформен хостинг и развитие.



Фиг. 1. Представя отделните компоненти и стъпки при процеса на интеграция и внедряване

[5] Под "нефункционални изисквания" имаме предвид дефиниране на техническите атрибути на системата: натоварване, обем на данни, едновременни потребители и др.

1. Архитектурни подходи за дизайн на системата

1.1. Монолитна архитектура на приложение за е-търговия

Повечето традиционни .NET приложения се внедряват като единици, съответстващи на изпълними файлове или казано по друг начин уеб приложения, работещи в рамките на един домейн на IIS сървър. Този подход е най-простият модел за внедряване и обслужва добре много вътрешни и по-малки публични приложения. Това са така наречените монолитни приложения - напълно самостоятелни по отношение на своето поведение. Могат да взаимодействат с други услуги или хранилища на данни в хода на извършване на своите операции, но ядрото на тяхното поведение се изпълнява в рамките на собствен процес и обикновено цялото приложение се разгръща като самостоятелна единица. Ако такова приложение трябва да се мащабира хоризонтално, обикновено то се дублира върху множество сървъри или виртуални машини. Това са приложения от тип „всичко в едно“.

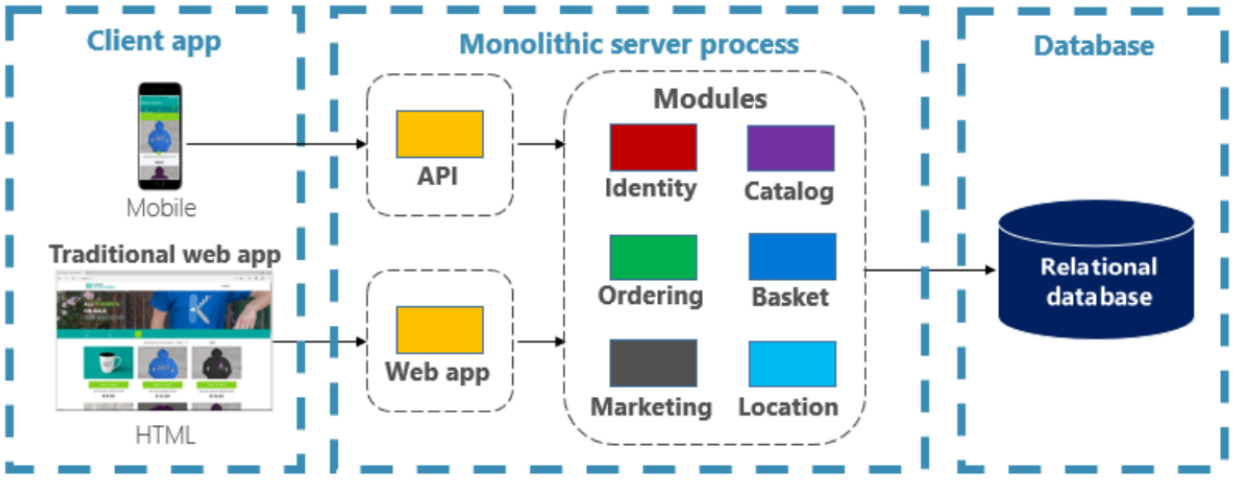
В тази архитектура, цялата логика на приложението се съдържа в един проект, компилиран и внедрен като самостоятелна единица. Шаблонът на нов ASP.NET Core проект, независимо дали е създаден във Visual Studio или от командния ред, започва като прост монолит „всичко в едно“. Той съдържа цялото поведение на приложението, включително логика за визуализация, бизнес и достъп до данни. Разделянето на логиката се постига чрез използването на папки. По подразбиране шаблонът включва отделни папки за отговорности на MVC[6] (модели, изгледи и контролери) както и допълнителни папки за данни и услуги. Макар и просто, монолитното решение за един проект има някои недостатъци: когато размер и сложността на проекта нарастват, броят на файловете и папките също ще продължи да расте.

Бизнес логиката е разпръсната между моделите и класовете на услуги без ясна индикация. Тази липса на организация на ниво проект често води до т.нар.

[6]Модел-Изглед-Контролер (Model-View-Controller или MVC) е архитектурен шаблон за дизайн в програмирането, основан на разделянето на бизнес логиката от графичния интерфейс и данните.

"спагети код"[7]. За да се справят с тези проблеми, приложенията често се развиват в много-проектни решения, където всеки проект отговаря на определен слой на приложението. Чрез организиране на кода в слоеве, общата функционалност на ниско ниво може да бъде преизползвана. Тази повторна употреба е от полза, защото показва, че трябва да се пише по-малко код и стандартизирането на една реализация.

На фиг. 2 е показан примерен дизайн на монолитно приложение за електронна търговия.



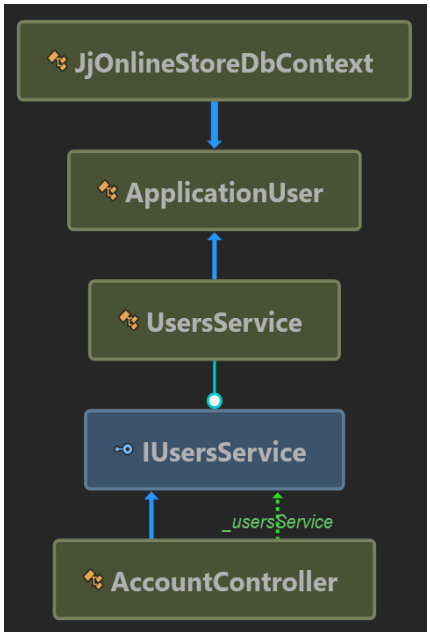
Фиг. 2. Традиционен монолитен дизайн

Модулите, отговарящи за първоначалните изисквания на приложението са два и те включват:

• Удостоверяване - процесът на определяне кой има достъп до системата. В уеб-базирано удостоверяване има няколко действия, които трябва да бъдат извършени: изисква от потребителя информация (потребителско име и парола) за да създадете самоличност, която записва в базата данни, вписва текущия клиент в сървърната сесия, използвайки HTTP бисквитки и отписва, като премахне тази информация. Елементите от приложението и зависимости, които ще обслужват тази част са визуализирани на фиг. 2. **DbContext** и **ApplicationUser** представляват комбинация от класове, които оперират с

[7] Спагети код е пейоративен израз за т.н. изходен код, имащ комплексна и заплетена структурна подредба.

базата от данни. **AccountController** използва тези свойства чрез **UsersService**, който капсулирана логиката, по безопасен за използване начин, и също така отговаря за визуализацията на потребителския интерфейс, чрез генериране на HTML.



Фиг. 3. Структура на класовете, отговарящи за удостоверяване

• Каталог - поддържа обхождане, добавяне, промяна и премахване маркетингови артикули от базата с данни. Подобно на предходния модул, осъществяването на спецификацията се случва чрез ProductsController, ProductsService, и т.н. Целта е всички модули да бъдат структурирани и да изглеждат по сходен начин, който да пази добро ниво на абстракция и капсулация на кода, но в същото време да бъде интуитивен и разбираем.

Структурата на папките на приложениео е добре оформена, по следния функционален, управляван от домейн дизайн:

```
├──eShop
│   ├──src – индикира път към основния код
│   │   ├──eShop.Web - логика за представяне
│   │   ├──eShop.Business – бизнес логиката на приложението
│   │   ├──eShop.Core – експонирани за клиенти модели
│   │   ├──eShop.Domain – вътрешни модели на приложението
│   │   └──eShop.Persistence – логика за достъпа до данните
│   └──tests – индикира път към тестовете на кода
│       └──eShop.Tests – пълен набор от автоматизирани (интеграционни) тестове
```

Много успешни приложения, които съществуват днес, са създадени като монолити. С течение на времето, обаче, се наблюдават някои слаби точки като:

• Новите промени могат да имат нежелани и скъпи странични ефекти.

• Новите функции стават трудни, отнемащи време и скъпи за прилагане.

• Всяка версия изисква пълно обновяване на цялото приложение.

• Един нестабилен компонент може да срине цялата система.

1.2. Микросървисна системна архитектура

За да реши горе описаните, но и много други, проблеми, следва да разгледаме ориентирания към услуги архитектурен стил. Това е подход за изграждане на сървърно приложение като набор от малки, но високо-качествени подуслуги. Съответно, клиентите, на сървърните услуги, могат да бъдат отделни приложения, които да се поддържат и управляват самостоятелно. Всяка услуга работи в собствен процес и комуникира с други процеси, използвайки различен тип и вид протоколи като: HTTP/HTTPS, WebSockets[8], AMQP и мн други. Всеки микросервис притежава специфична бизнес способност, трябва да бъде разработван автономно и да може да се разгръща независимо. Предимства на това архитектурно решение са:

• Всяка микроуслуга може да бъде проектирана, разработена и внедрена независимо една от друга, което осигурява възможно за независима работа по отделни области на приложението.

• Работата може да бъде дистрибутирана между отделни екипи.

• Проблемите са по-изолирани.

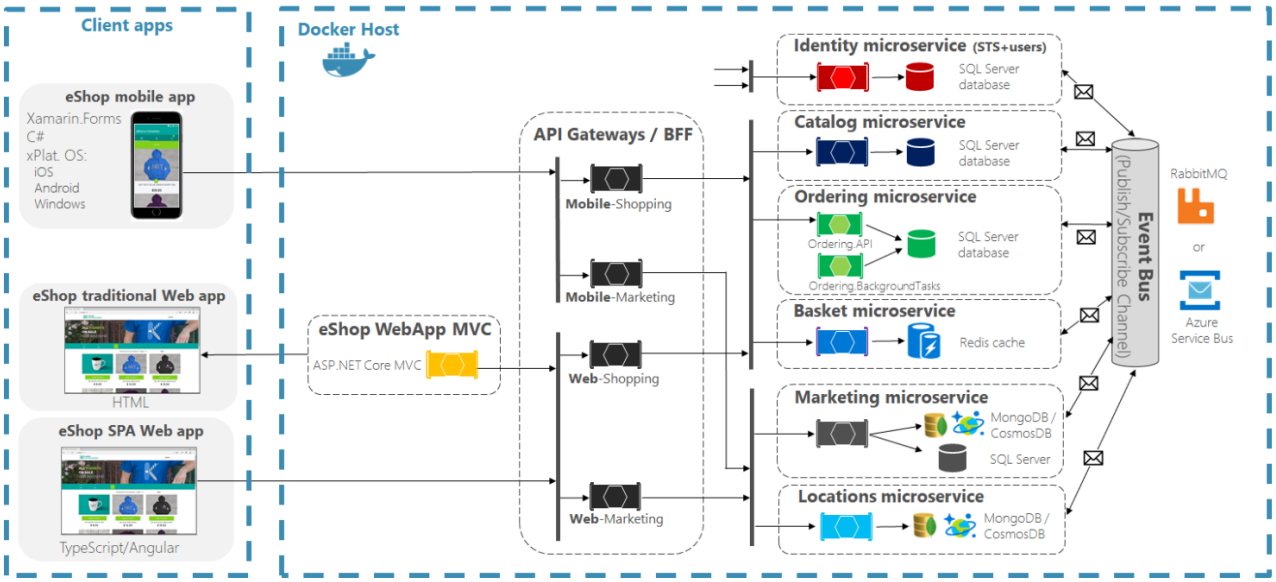
• Позволява използването на най-новите технологии.

Тъй като ориентираната към услуги архитектура носи специфични изисквания и сложност, нека разгледаме интеграцията на онлайн магазинът, изграден от микро-услуги, към облачно базирана среда в следващата глава.

[8]WebSockets - компютърен комуникационен протокол, осигуряващ пълнодуплексни комуникационни канали през една TCP връзка. Проектиран е да работи през портове на HTTP (80 и 443)

2. Модели на подсистемите и хранилищата им за данни

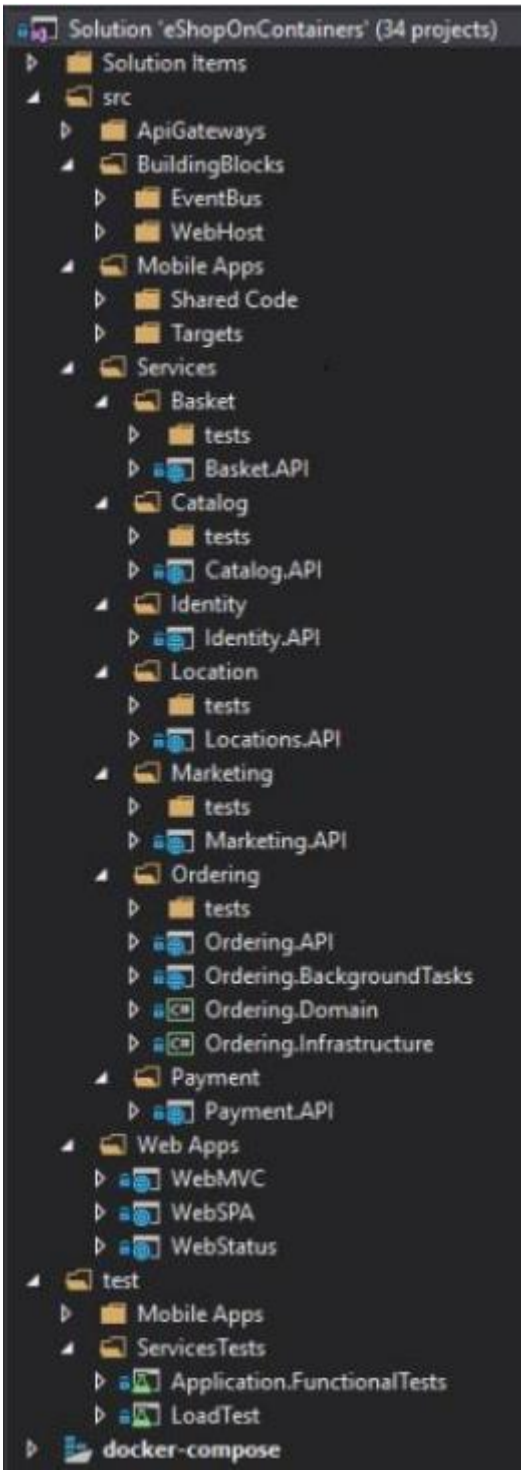
Като начало на тази част, нека разгледаме фиг. 4, която илюстрира как монолитното структурираният модел се превръща в ориентирана към услуги, базирана на облак, система.



Фиг. 4. Микросървисната архитектура за разработка на eShop приложение

Системата е достъпна от уеб или мобилни клиенти, които имат достъп през HTTPS, насочени или към сървърното приложение ASP.NET Core MVC, или към подходящ API шлюз. Функционалността на приложението е разделена на много отделни микроуслуги (надграждащи модулите от монолитния дизайн): удостоверяване и самоличност, управление на потребители, изброяване на артикули от продуктовия каталог, заявяване на поръчки и др. Всяка от тези отделни услуги има свое собствено хранилище за основни данни . Няма единно хранилище за основни данни, с което всички услуги взаимодействат. Всяка от различните микроуслуги е проектирана по различен начин, въз основа на техните индивидуални изисквания.

От гледна точка на изходния код, проектът включва доста отделни решения в Git хранилището си. Фигура 5 показва пълното решение на Visual Studio, в което са организирани подпроектите.



Фиг. 5. Структура на подпроектите във Visual Studio

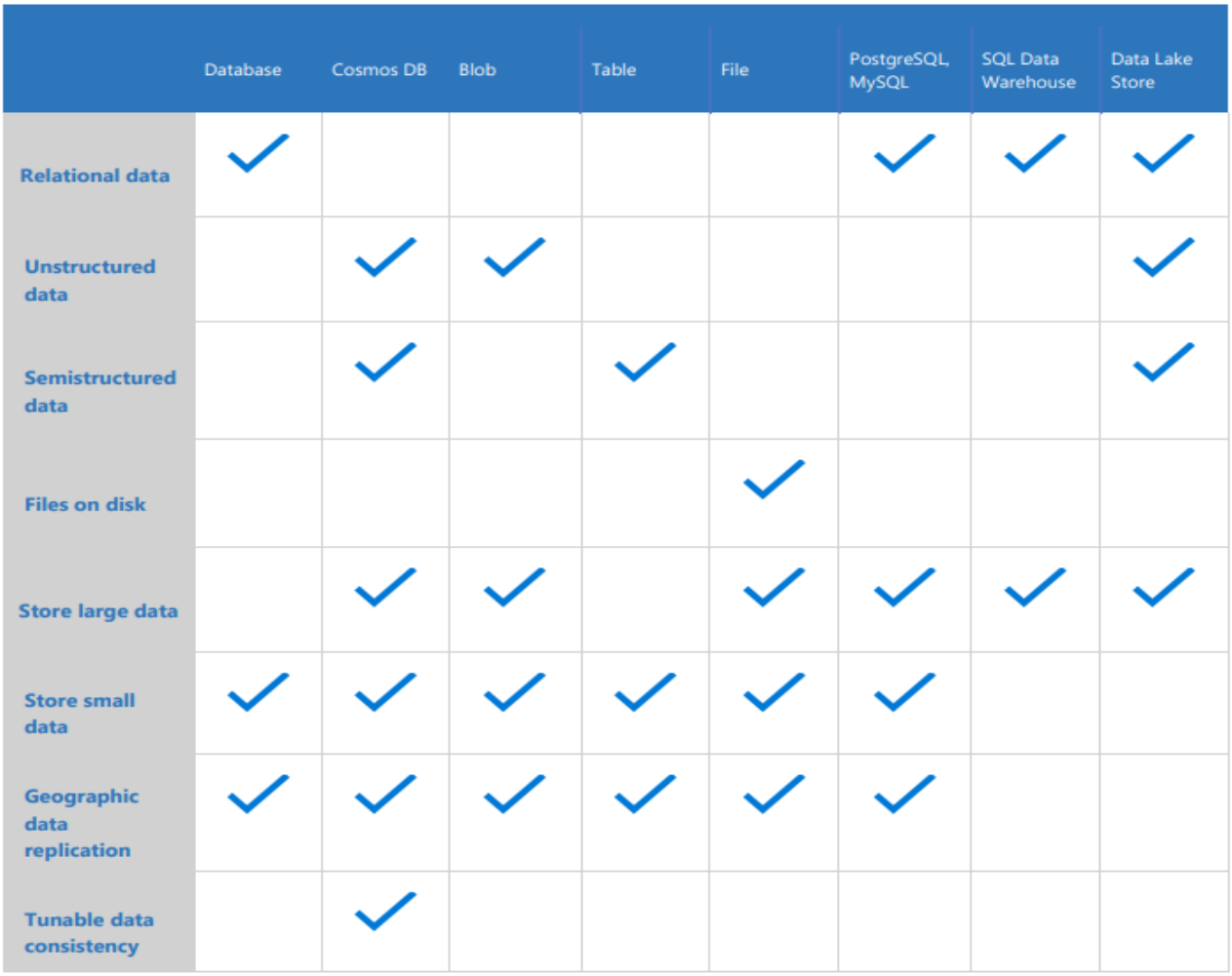
API шлюзовете предлагат няколко предимства, като например разпределяне на заявки между услугите от индивидуални клиенти, с цел осигуряване на по-добра сигурност. В примера, архитектурата демонстрира разделяне на API шлюзовете въз основа на това дали заявката идва от уеб или мобилен клиент. Осъществяването на Azure се нарича API Management (APIM).

То помага на организациите да публикуват програмните интерфейси по последователен и управляем начин.

Различните back-end услуги, използвани от eShop, имат различни изисквания за съхранение на данните. Azure предоставя много видове хранилища за данни, които могат да помогнат за поддръжка и извличане на данни:

- Azure SQL Database - Това е облачно базиран SQL Server. Поведението му е същото като това на основното изпълнение на базата, но предлага и много предимства: репликира в реално време данни в други географски региони, маскира данни за определени потребители, предоставя пълен одит на всички действия, които са се случили върху данните. Услугата е използвана от подсистемите за удостоверяване и каталогът за продуктите.
- Azure Cosmos DB е нов вид нерелационна база данни, която работи с механизъм за съхранение и предоставяне на данни, който използва свободен модел, също така включва ниска латентност, репликация на данни в други географски региони в реално време, управление на трафика, автоматично индексиране на данните. Услугата е използвана от маркетинговата част.
- Azure Blob представлява хранилище за съхраняване на големи неструктурирани данни. Това могат да бъдат фактури, изображения, видео, файлове и други. Услугата е използвана от подсистемата за поръчки.
- допълнение, Azure предоставя услуги за бази данни MySQL, PostgreSQL и MariaDB като универсално достъпни, мащабируеми, силно защитени и напълно управлявани.
- Azure предоставя две хранилища за данни, които са много подходящи за съхранение на големи количества с цел анализ: Data Warehouse & Data Lake.

На фиг. 6 са показани различните услуги според структурата на данните.



Фиг. 6. Показва коя услуга за данни да се използва при определен сценарий

3. Софтуерното внедряване и поддръжка в облачна среда

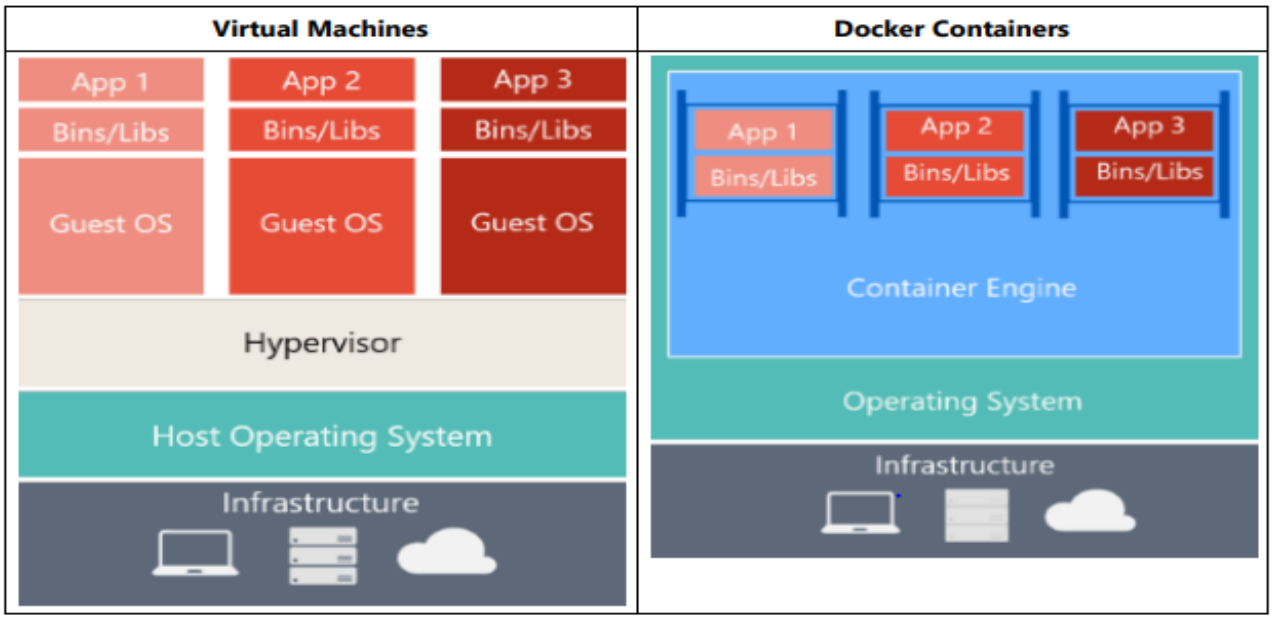
За изграждане, доставка и изпълнение на системи, изградени както като монолитни приложения, така и като ориентирани към услуги, се препоръчва използването на контейнеризирани технологии. **Контейнеризацията** е подход, в сферата на разработката на софтуер, при който кодът на приложение, всички негови зависимости и конфигурации са пакетирани в двоичен файл, наречен **изображение**. Изображенията са „шаблони“ само за четене и се съхраняват в **регистър**, който работи като хранилище или библиотека за изображения. Изображението се трансформира в работещ екземпляр на **контейнер**, който може да се стартира, спира, премества и изтрива. Създават се контейнери за различните части от приложението: уеб услуга, база данни, кеширане и др. Точно както транспортните контейнери позволяват транспортирането на стоки,

независимо от товарите вътре, софтуерните контейнери се възприемат като стандартна единица за внедряване на софтуер, която може да съдържа различен код и зависимости. Контейнеризирането на софтуера дава възможност на разработчиците и ИТ специалистите автоматично да подновяват новите промени в различни среди. Контейнерите също така изолират приложенията едно от друго в споделена операционна система. Приложения се изпълняват върху хостът на контейнерите. От гледна точка на приложението, инстанцирането на изображение означава създаването на контейнер. Друго предимство на контейнеризацията е мащабируемостта. Разширяването става бързо: създават се нови контейнери за краткосрочни задачи. Контейнерите предлагат предимствата на изолация, преносимост, гъвкавост и контрол в целия жизнения цикъл на приложението.

Най-използваната и наложила се като стандарт технология е **Docker**[9]. Docker е проект с отворен код за автоматизиране на внедряването на приложения като преносими, самодостатъчни контейнери, които могат да работят локално или в облака. Docker също е компания, която популяризира и развива тази технология. Docker контейнерите могат да работят върху Linux или Windows. Предимства за разработчиците са: ускорено въвеждане на нови програмисти в проекта, премахнете конфликтите в приложенията, актуализиране и мигриране на софтуера.

На фиг. 7 е представено сравнение между виртуална машина и Docker контейнер.

[9]Осъществяване на идеите на разработчиците с Docker, 2019 <https://www.docker.com/why-docker/>



Фиг. 7. Виртуални машини и Docker контейнерите

Виртуалните машини включват приложението, необходимите библиотеки и пълна операционна система. Изисква пълна виртуализация повече ресурси, повече време за стартиране в сравнение.

Докер контейнерите включват приложението и всички негови зависимости. Те обаче споделят ядрото на ОС с други контейнери, изпълняващи се като изолирани процеси в потребителското пространство на хост операционната система. (с изключение на Hyper-V контейнери, където всеки контейнер работи вътре в специална виртуална машина).

Виртуалните машини имат три основни слоя: инфраструктура, хост, операционна система, Hypervisor и всички необходими библиотеки. Слоевете в Docker са инфраструктурата, ОС и двигател за контейнери, който поддържа изолация, но споделя основните услуги на ОС. Тъй като контейнерите изискват много по-малко ресурси (например не се нуждаят от пълна ОС), те са лесни за изпълнение, внедряване и започват бързо. Основната цел на изображението е да направи зависимостите еднакви в различните среди. Това гарантирана еднакво поведение на всички среди: локална среда, среда за разработка или продуктивна.

Azure предоставя услуги, които могат да помогнат за постигане на много неща, варирайки от обикновени, като създаване на ново приложение с база от

данни – до по-развити като създаване на работни потоци за непрекъсната интеграция (CI) и внедряване (CD). Това са само няколко примера за някои често срещани работни похвати. Много от тях трябва да бъдат създадени индивидуално, но облачната инфраструктура предлага всичко това като услуги. Силата на облака е, че ресурсите са невероятно устойчиви, малко вероятно е аварийно да спрат работа, тъй като центровете за данни са разположени по целия свят, състоящи се от десетки хиляди сървъри. Ако един сървър се повреди, друг поема управлението. Един от най-убедителните аргументи в полза на облака е, че може да разширява мащаба на услуги и ресурси почти безкрайно, в определени моменти, като например "Черен Петък" или голяма маркетингова кампания с промоции и намаления на артикули. Също така, когато натоварването намалее, мащабът може да се намали до обикновените си параметри. Уважавани и опитни облачни доставчици като Microsoft разпознават моделите на използване на нормалните потребители и тези на злонамерените. Инфраструктурата е предпазена от най-често срещаните атаки. Интелигентни инструменти за наблюдение, алгоритми за обучение и изкуственият интелект предоставят възможност да откриват атаки[10] в реално време.

При стартиране на приложения в Azure едно от първите решения, които трябва бъдат вземети, са планираните за използване услуги:

- Azure App Services - един от най-лесните и мощни начини за хостване на приложения. Той е предпочитан при монолитната архитектура. Услугите са достъпни и работят в 99,95% от времето. Споделят мощни функции като автоматично мащабиране, внедряване с нулев застой и лесно удостоверяване, позволяват отстраняването на грешки в приложението докато работи в производствена среда (със Snapshot Debugger). По подразбиране приложението ще бъде достъпно в интернет, без да е необходимо да се настройва име на домейн или да се конфигурира DNS. Работи много добре с контейнери.
- Azure Virtual Machines - позволява преместване на съществуващи

[10]Защо предприятията се доверяват на Azure за своите приложения и данни , 2019 <https://azure.microsoft.com/en-us/blog/why-enterprises-trust-azure-with-their-apps-and-data/ />

приложения от виртуални машини, които вече се изпълняват във център за данни. Има много предварително дефинирани изображения, които могат да бъдат използвани като Windows Server, който работи с IIS и има инсталиран и предварително конфигуриран ASP.NET на него, както и собствени софтуерни лицензи (като за SQL Server). Услугата е подходяща за мигриране на т.нар. „наследена система“, която да бъде използва като подсистема или източник на данни.

- Azure Kubernetes (AKS) - водещ инструмент за управление и мащабиране на контейнери, отговарящ за приложения разпределени между микроусуги. Kubernetes предоставя операции от високо ниво, които да бъдат извършени чрез кода на самите микро-услуги. Работи с инструкции, които са прехвърлени върху облачните машини, така нареченият „клъстар“: набор от виртуални машини на Linux или Windows (наречени възлови точки), върху които се разполагат самите приложения (но не директно). Kubernetes се грижи за маршрутизирането и логистика на микросервизните (най-често използван в тази архитектура).

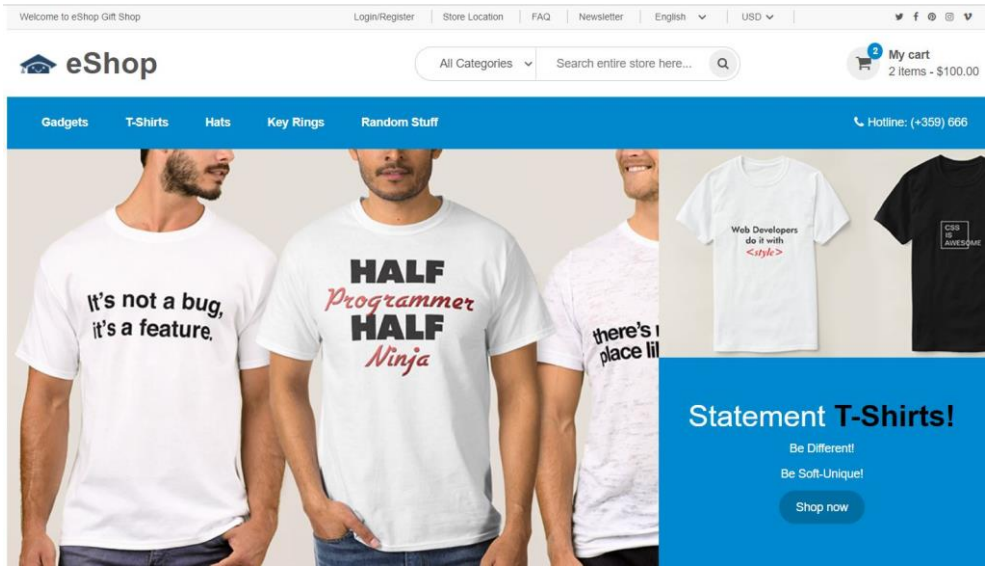
	App Service Web Apps	App Service Mobile Apps	Azure Functions	Logic Apps	Virtual Machines	Azure Kubernetes Service (AKS)	Container Instances
Monolithic and N-Tier applications	✓				✓*		✓
Mobile app back end		✓			✓*		
Microservice architecture-based applications			✓			✓	
Business process orchestrations and workflows			✓	✓			

Фиг. 8. Представя кои услуги на Azure са подходящи за различните типове.

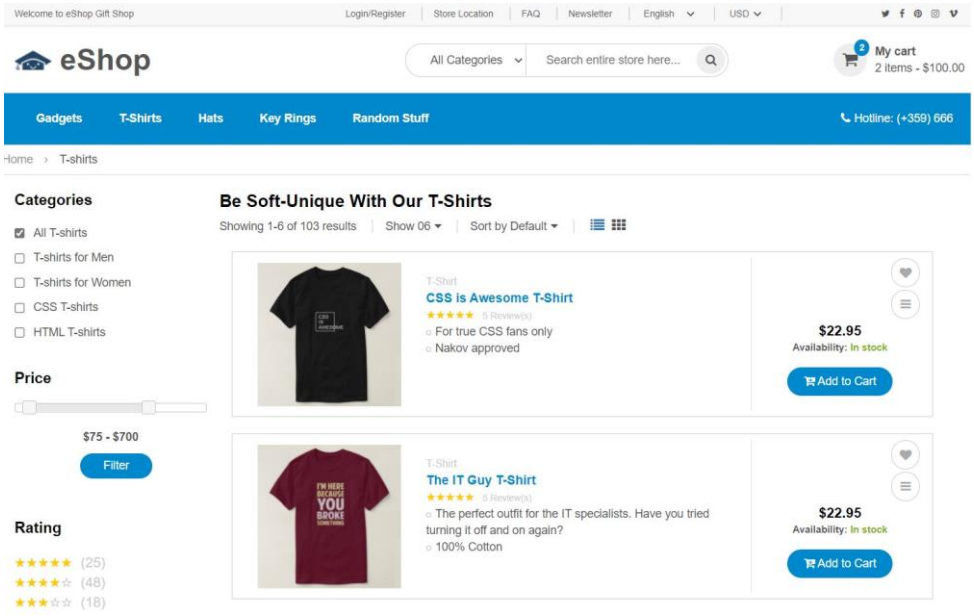
4. Потребителски интерфейс на системата

Потребителският интерфейс на ASP.NET Core уеб приложенията разчитат на клиентските технологии като HTML, CSS и JavaScript, Bootstrap, jQuery. Чрез отделяне на съдържанието на страницата от своето оформление, стил и поведението, сложните уеб приложения следват най-добрите практики за добро форматиране и структура на кода. Това прави бъдещите промени по-лесни за разработка. Важно изискване е да работи правилно в най-новите HTML5 съвместими браузъри: Chrome, Firefox, Edge, Opera, Safari.

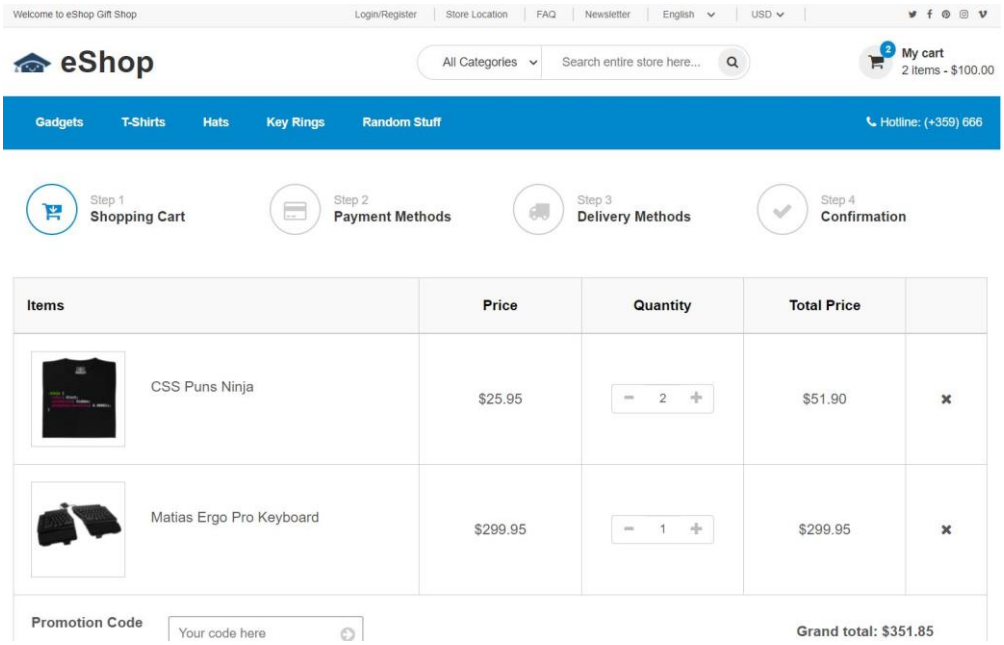
Следните три фигури представят стратегически важни компоненти на системата, от които се осъществява входът в системата.



Фиг. 9. Начален екран на приложението



Фиг. 10. Екран на продуктовият каталог на приложението



Фиг. 11. Екран на продуктовият каталог на приложението

Заключение

В заключение може да се каже, че облачните изчисления са подход за проектиране на модерни приложения, които обхващат бърза промяна, голям мащаб и устойчивост в динамични среди като публични, частни и хибридни облаци. The Cloud Native Computing Foundation, влиятелен консорциум от над 300 големи корпорации, който носи отговорност за приемането на облачни технологии, препоръчва системите да обхващат шест важни стълба:

– Облакът и основният модел на обслужване

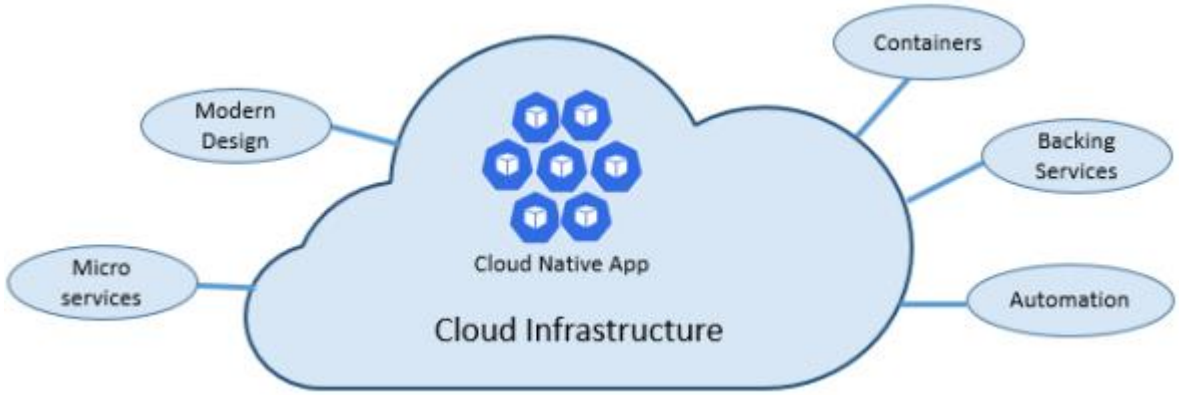
– Модерни принципи на проектиране

– Микроуслуги

– Контейнеризация и оркестрация

– Базирани в облак услуги като бази данни

– Автоматизация



Фиг 12. Основополагащи стълбове на облачните системи

Трябва да се отбележи, че прототипът не е пълно функционален, а демонстрира ключови изпълнения. Целта е да се докаже неговата практическа приложимост и ползваемост.

Използвана литература

1. Robert Vettor, Architecting Cloud-Native .NET Apps for Azure. Microsoft. Redmond, Washington. 2021
2. Steve Smith, Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure. Microsoft. Redmond, Washington. 2021
3. Cesar de la Torre, Containerized Docker Application Lifecycle with Microsoft Platform and Tools. Microsoft. Redmond, Washington. 2021
4. Bill Wagner, .NET Microservices: Architecture for Containerized .NET Applications, Redmond, Washington, 2021
5. Vasilev, J., Sulov, V., Drazhev, S., Nacheva, R. Business Management Systems (MS Dynamics Navision). Developing Web Applications with the .NET Platform. Microsoft Information Systems and Applications Security.. Варна: Наука и икономика, 2015.
6. Сълов, В. Приложение на езиците за програмиране на платформата .NET при разработката на софтуерни приложения. Изв. Сп. Икон. унив. - Варна , 2014, № 1, с. 13 - 22.
7. How Netflix Deploys Code, Infoq, 2013 https://www.infoq.com/news/2013/06/netflix/
8. Uber Engineering’s Micro Deploy: Deploying Daily with Confidence, Eng.Uber, 2013, https://eng.uber.com/micro-deploy-code/
9. Developers bring their ideas to life with Docker, 2019 <https://www.docker.com/why-docker/>
10. Why enterprises trust Azure with their apps and data, 2019 <https://azure.microsoft.com/en-us/blog/why-enterprises-trust-azure-with-their-apps-and-data/ />