



ИКОНОМИЧЕСКИ УНИВЕРСИТЕТ - ВАРНА
КАТЕДРА „ИНФОРМАТИКА”

х.ас.докт. Йордан Иванов Йорданов

Проект на тема:

**МОДЕЛ НА СЕМПЪЛ ОНЛАЙН МАГАЗИН БАЗИРАН НА ОБЛАЧНАТА
ПЛАТФОРМА MICROSOFT AZURE**

по дисциплина „Езици за програмиране“

Варна 2021

Съдържание

Въведение.....	Error! Bookmark not defined.
1. Подходи за дизайн на системата.....	Error! Bookmark not defined.
2. Модели на подсистемите в онлайн магазина.....	Error! Bookmark not defined.
3. Архитектура на софтуерното внедряване в облачна среда.....	Error! Bookmark not defined.
Заклучение	21
Използвана литература	22

Въведение

За кратко време облачните технологии се превърнаха в водеща тенденция в софтуерната индустрия. Те представят нов начин за изграждане големи, сложни системи. Подход, който се възползва изцяло от съвременните практики за разработка на високо-качествен софтуер и инфраструктура. Предлага промени в начина, по който проектираме, интегрираме и внедряваме 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.

Cloud Native Computing Foundation¹ предоставя официално 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 потребителите са все повече и повече, главно насочени към бърза реакция, иновативни функции и нулев застой. Проблеми с производителността или повтарящи се грешки вече не са приемливи. Тези предимства поставят бизнеса на една стъпка пред конкурентите. Облачно базираните системи се свързват главно с бързина. Бизнес системите се развиват от способностите на бизнеса да бъдат инструменти за стратегическа трансформация, която ускорява растежа на компанията. Незабавното пускане на иновативните идеи на пазара е важна тема за всички модерни компании. Нека разгледаме два технологични компании, които са приложили тези техники:

¹Cloud 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.

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

- Uber има над 1000 услуги в производствена среда. Разгръщат се няколко хиляди пъти всяка седмица.

Както може да видим тези две компании са базирани на системи, които се състоят от стотици независими микроуслуги. Този архитектурен стил им позволява бързо да реагират на пазарните условия като постоянно да актуализират малки, но важни области. Скоростта и пълнотата 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, нека разгледаме характеристики и изисквания на функционален облачен продукт, демонстриращ използването на .NET, Docker, Kubernetes в облачната среда на Microsoft Azure за осъществяването семпъл онлайн магазин. Ето някои от основните системни изисквания², 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 системни изисквания – задачи от високо ниво, за това, какво системата трябва да прави. Обикновено се предоставя от функционален анализатор.

Приложението има и следните нефункционални изисквания³:

- 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.

1. Подходи за дизайн на системата

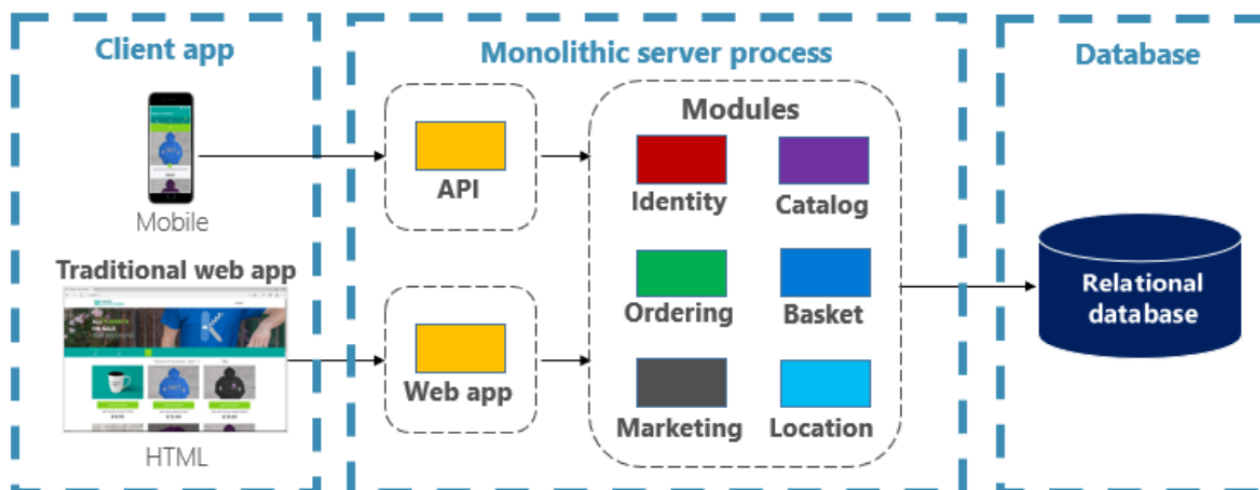
Повечето традиционни .NET приложения се внедряват като единици, съответстващи на изпълними файлове или казано по друг начин уеб приложения, работещи в рамките на един домейн на IIS сървър. Този подход е най-простият модел за внедряване и обслужва добре много вътрешни и по-малки публични приложения. Това 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.

³ нефункционални изисквания – дефиниране на техническите атрибути на системата: натоварване, обем на данни, едновременни потребители и др.

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 MVC⁴ (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.н "спагети код"⁵. 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 пре-използвана. Тази повторна употреба е от полза, защото показва, че трябва да се пише по-малко код и стандартизирането на една реализация.

На фиг. 1 може да видим примерен дизайн на монолитно приложение за електронна търговия.

фиг1. Традиционен монолитен дизайн

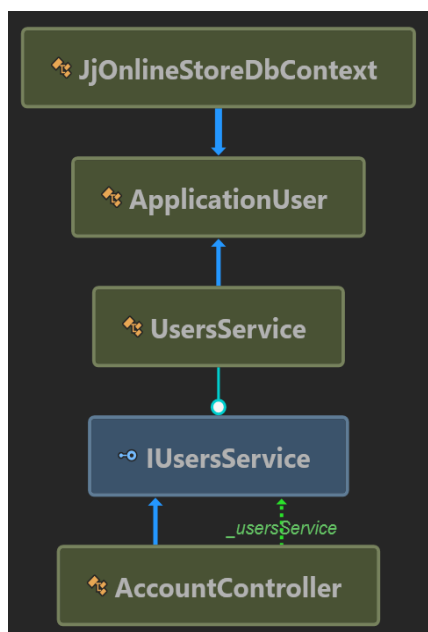


⁴Модел-Изглед-Контролер (Model-View-Controller или MVC) е архитектурен шаблон за дизайн в програмирането, основан на разделянето на бизнес логиката от графичния интерфейс и данните.

⁵Спагети код е пейоративен израз за т.н. изходен код, имащ комплексна и заплетена структурна подредба.

Модулите, които ще отговарят за първоначалните изисквания на приложението са два, които ще 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 премахва тази информация. Елементите от приложението и зависимости, които ще обслужват тази част са визуализирани на



следната фигура 2. **DbContext** и **ApplicationUser** представляват комбинация от класове, които оперират с базата от данни. **AccountController** използва тези свойства чрез **UserService**, който капсулирана логиката по безопасен за използване начин, и също така отговаря за визуализацията на потребителския интерфейс, чрез генериране на HTML .

Фиг2. Структура на класовете, отговарящи за удостоверяване

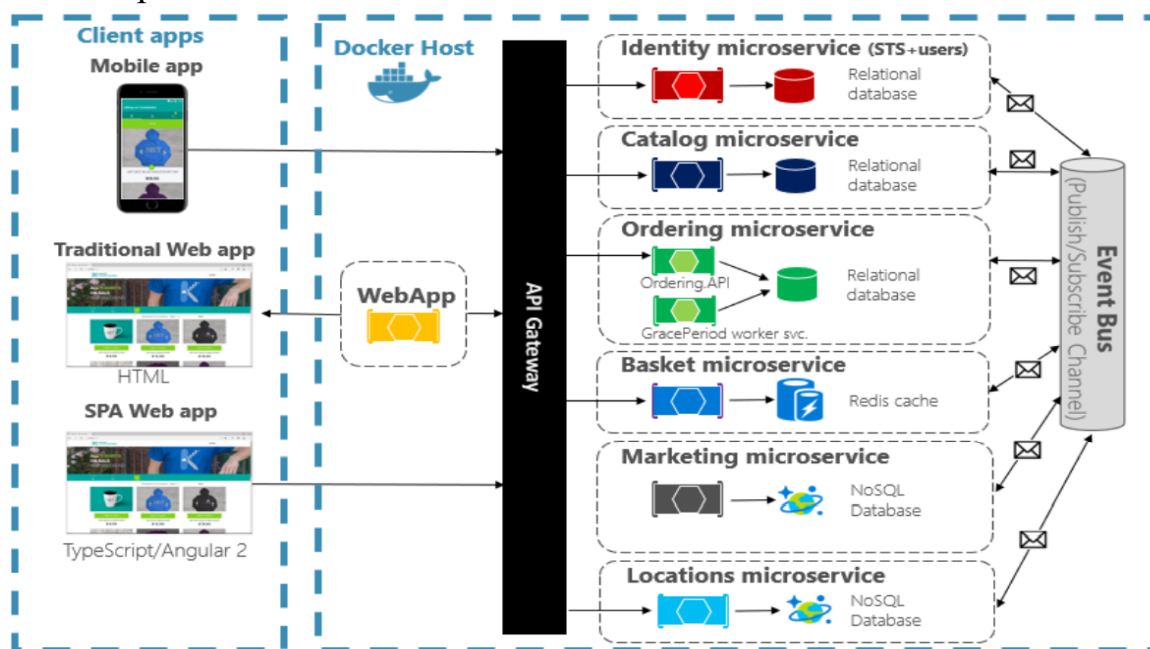
- Каталог - поддържа обхождане, добавяне, промяна и премахване маркетингови артикули от базата с данни. Подобно на Идеята на всички модули, и техните класове, е бъдат структурирани по същия начин, като предходния: **ProductsController**, **OrdersController** **ProductsService**, **OrderService** и т.н.

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

- Приложението може да стане доста сложно, и логиката да бъде трудна за разбиране и поддържане

- Новите промени могат да имат нежелани и скъпи странични ефекти.
- Новите функции стават трудни, отнемащи време и скъпи за прилагане.
- Всяка версия изисква пълно разгръщане на цялото приложение.
- Един нестабилен компонент може да срини цялата система.

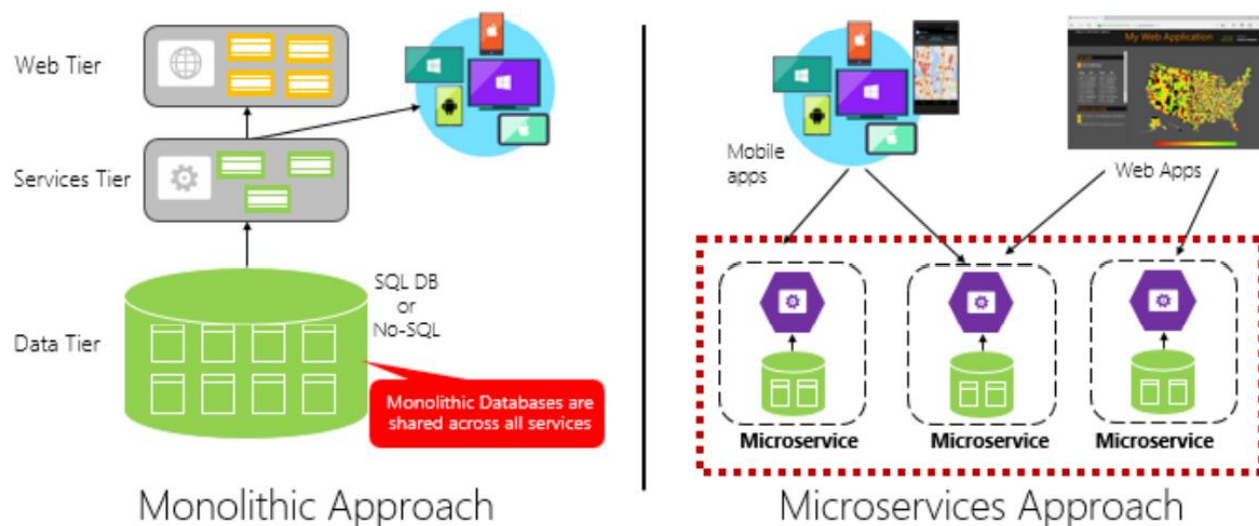
Следната диаграма показва същата система, изградена с прилагане на облачни техники и практики:



Приложението е разделено в набор от малки изолирани микроуслуги - популярен архитектурен стил за изграждане на модерни приложения. Всяка услуга е самостоятелна и капсулира свой собствен код, данни и зависимости. Те споделят следните характеристики:

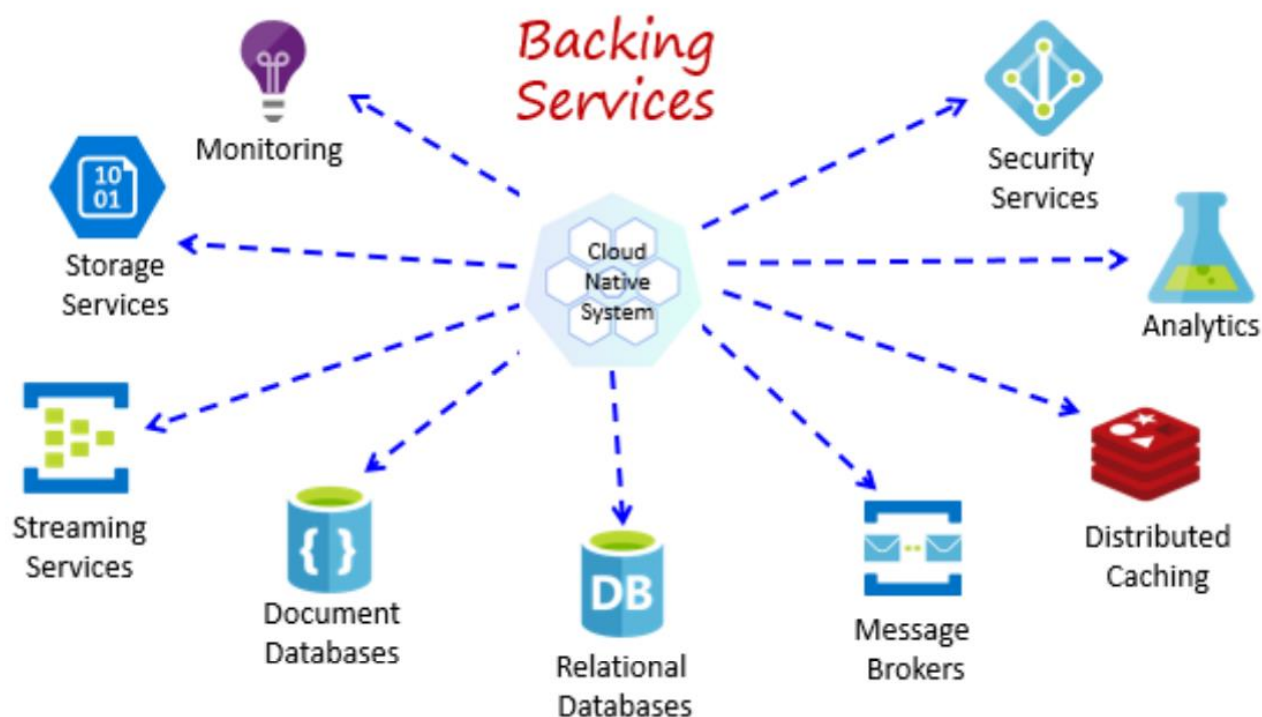
- Всяка реализира специфична бизнес способност в рамките на по-голям контекст
- Всяка от тях се разработва автономно и може да бъде внедрена независимо от другите.
- Всяка е самостоятелна, като капсулира своя собствена технология за съхранение на данни⁶ (SQL, NoSQL) и платформа за програмиране.
- Всяка работи в свой собствен процес и комуникира с другите, използвайки стандартна комуникация протоколи като HTTP/HTTPS, WebSockets или AMQP.

⁶ Бази данни – todo



Микроуслугите осигуряват гъвкавост. Всяка микроуслуга има автономен жизнен цикъл и може да се развива независимо. Може да актуализираме малка област от сложно приложение с по-малък риск от нарушаване на цялата система. Всяка микросервис може да се разширява мащаба си независимо, като фокусът да бъде върху онези услуги, които изискват повече процесорна мощност или друг фактор. Този фин подход към мащабирането осигурява по-голям контрол над системата и помага за намаляване на общите разходи. Микросервизите могат да бъдат създадени с всяка модерна платформа за разработка. Платформата на Microsoft .NET е отличен избор: безплатен и с отворен код, има много вградени функции за опростяване на разработването на микросервизи. .NET е кросплатформен. Приложенията могат да се изграждат и изпълняват на всички операционни системи: Windows, macOS и повечето разновидности на Linux. .NET е с висока производителност и има добри резултати в сравнение с Node.js и други конкурентни платформи.

Облачно базираните услуги зависят от различни спомагателни ресурси, като хранилища на данни, брокери на съобщения, услуги за наблюдение и идентификация. Тези услуги са известни като поддържащи услуги.



Във конкретния пример, функционалността на приложението е разделена на много отделни микрослуги: отговарящи за удостоверяването и самоличността, изброяване на артикули от продуктовия каталог, управление на потребителските пазарски кошници и регистриране на поръчки. Координацията и комуникацията между услугите се извършва при чрез използване на шина за съобщения или друг вид комуникация. Всяка от различните микрослуги е проектирана по различен начин, въз основа на техните индивидуални изисквания. Техният технологичен стек може да се различава. По-простите услуги осигуряват основен достъп от тип "Създаване-Прочетете-Актуализиране-Изтриване" (CRUD) до хранилища на данни, докато по-напредналите услуги използват подходи и модели, управлявани от домейн.

2. Модели на подсистемите в онлайн магазина

Azure е много подходящ за поддръжка на така структуриран онлайн магазин, тъй като проектът е създаден да бъде приложение, базирано на облак. Azure предоставя услуги, които могат да помогнат за постигане на много неща. Те варират от обикновени, като добавяне на ново приложение с база от данни – до към по-екзотични като създаване на работни потоци за непрекъсната интеграция (CI) и непрекъснато внедряване (CD). Това са само няколко примера за някои често срещани проекти, които разработчиците е трябвало многократно да създават индивидуално, но всичко това вече се предлага като услуга, които се използват с много малко усилия. Силата на облака е, че услугите и ресурсите са невероятно устойчиви, малко вероятно е аврийно да спрат работа. Azure има центрове за данни по целия свят, пълни с 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, което е почти невъзможно с локални ресурси. Също толкова, когато вече няма голямо натоварване, лесно може да се намали мащаба. 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 в реално време.

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

- Azure App Services - един от най-лесните и мощни начини за хостване на приложения. 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 и 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. Разработчиците, които трябва да поддържат високо ниво на контрол върху средата може да изпълняват контейнери. Azure предоставя уникални възможности за мобилните приложения: поддържат офлайн синхронизиране, което позволява на приложението да продължи да работи когато няма интернет връзка. А push известията позволяват изпращането на съобщения до мобилните приложения използвайки C# код, независимо от платформата, на която работи (iOS, Android, Windows).

- Azure Functions - малки парчета код, които разработваме, без "притеснение" за основната инфраструктура. Друго наименование на този модел е "Функция като услуга (FaaS)". Целта им е да изпълнят няколко стъпки при вход.

- Azure Virtual Machines - това може да бъде лесен начин да започнете, защото позволява преместване на съществуващи 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).

На следната фиг. са представени кои услуги на Azure са най-подходящи за различните типове приложения:

	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			✓	✓			

Данните са особено важен аспект на всяко модерно .NET приложение. Azure предоставя много видове хранилища за данни, които могат да помогнат за поддръжка и извличане на данни.:

- Azure SQL Database - Това е обачно базиран SQL Server. Поведението му е същото като това на локалното 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

- Azure Cosmos DB е нов вид нерелационната 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 индексирание на данните.;

- Azure Storage е една от основните услуги за съхраняване на данни в различни форми, защото е лесна за използване, бърза и евтина. Също така е и много надежден. По подразбиране данните се репликират три пъти в рамките на център за данни. Azure File Storage, който е един от типовете хранилища, използва протокола SMB, което го прави много подходящ за създаване или преместване на файлове сървър в облака. Azure Blob позволява съхраняване на големи неструктурирани данни –могат да бъдат видеа, изображения, аудио или текстови записи или дори файлове на виртуален твърд диск (VHD) за виртуална машина. Следват се конвенции в кодирането;

- Azure предоставя услуги за бази данни MySQL, PostgreSQL и MariaDB като универсално достъпни, мащабируеми, силно защитени и напълно управлявани.

Следната таблица показва различните услуги според структурата на данните:

	Database	Cosmos DB	Blob	Table	File	PostgreSQL, MySQL	SQL Data Warehouse	Data Lake Store
Relational data	✓					✓	✓	✓
Unstructured data		✓	✓					✓
Semistructured data		✓		✓				✓
Files on disk					✓			
Store large data		✓	✓		✓	✓	✓	✓
Store small data	✓	✓	✓	✓	✓	✓		
Geographic data replication	✓	✓	✓	✓	✓	✓		
Tunable data consistency		✓						

Azure помага за повишаване на производителността на приложенията с услуги като Azure Redis Cache, който осигурява високопроизводителен, базирана на "ключ-стойност" структура за съхранение. Друга подобна услуга е Azure Content Delivery Network, или CDN, която може да репликира статично съдържание, като изображения и видео файлове до точки по целия свят, Azure Traffic Manager може да подобри "отзивчивостта" на приложението, като насочва потребителите към местоположенията с най-ниска латентност.

Освен да стартира приложението и да съхранява данни, трябва да бъде защитено. Azure може да помогне с няколко услуги:

- Azure Active Directory - предоставя удостоверяване и оторизация на потребители (клиенти).
- Azure Key Vault – позволява безопасно използване на тайни като пароли, сървърни потребители и др
- Managed Service Identity - по време на изпълнение на приложението, инжектира идентификационни данни, които след това да бъдат използвани за достъп до други услуги

Azure предоставя водещ в индустрията пакет от услуги, които могат да направят приложенията по-интелигентни. Azure Cognitive Services са набор от програмни интерфейси, които могат да разпознават лица, да конвертират реч в текст и да разбират значението на въведен текст. С помощта на услугата "Разбиране на езика" можете да създавате ботове, които интелигентно да отговаря на заявки в Slack, Microsoft Teams и други форуми. Можете също да създадете свои собствени алгоритми за машинно обучение с Azure Machine Learning.

В Azure има много услуги за извършване на анализ и трансформиране на данни, може да показва информация, използвайки собствен двигател за визуализация.

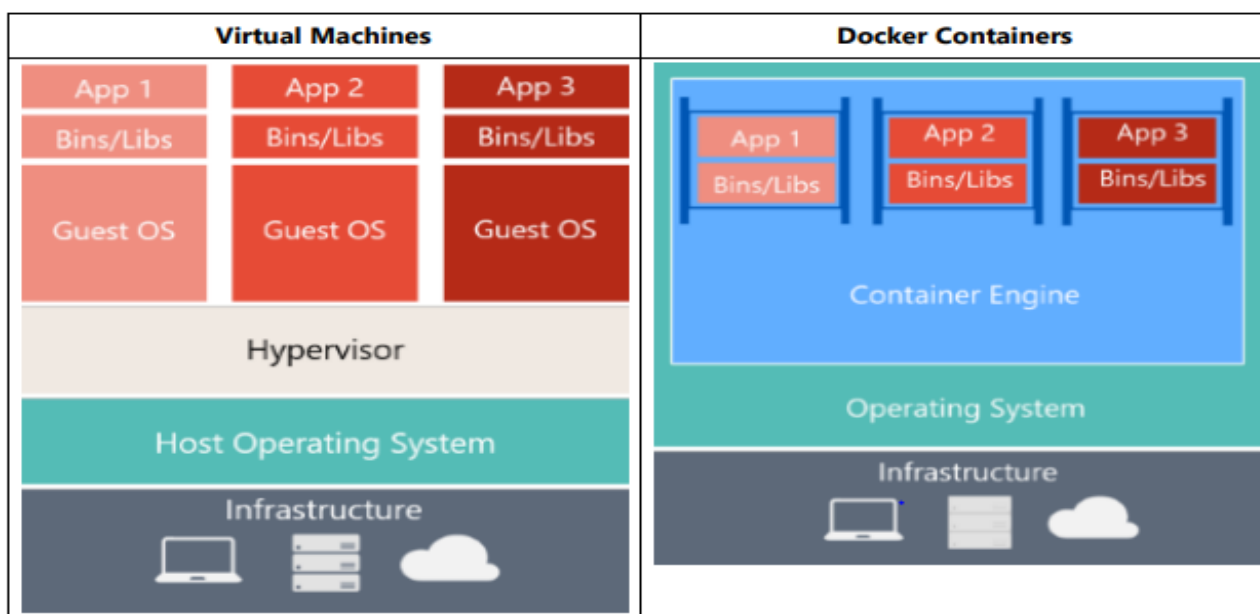
Работата със стрийминг и медийни технологии е предизвикателство за много приложения. Azure Media Services предоставя услуги като кодиране на медийни файлове. Azure Content Protection позволява използването на DRM технологии като PlayReady, за осигоряване на защита на съдържанието (използване само от упълномощени потребители). С помощта на Media Analytics може да се подобри качеството на медията като например да създаде субтитри, базиран на реч от текущ ресурс.

3. Архитектура на софтуерното внедряване и поддръжка в облачна среда

Контейнеризацията е подход, в сферата на разработката на софтуер, при който нодът на приложение, всички негови зависимости и конфигурации са

пакетирани в двоичен файл, наречен изображение на контейнер. Изображенията се съхраняват в регистър на контейнери, който работи като хранилище или библиотека за изображения. Изображението се трансформира в работещ екземпляр на контейнер, който може да се тества като самостоятелна единица. 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 разгръщат новите промени в различни среди. Контейнерите също така изолират приложенията едно от друго в споделена операционна система. Приложения се изпълняват върху хостът на контейнерите. Друго предимство на контейнеризацията е мащабируемостта. Разширяването става бързо: Създават се нови контейнери за краткосрочни задачи. От гледна точка на приложението, инстанцирането на изображение (създаването на контейнер) е подобно на инстанциране на процес като услуга или веб приложение. Контейнерите предлагат предимствата на изолация, преносимост, гъвкавост, мащабируемост и контрол в целия жизнения цикъл на приложението.

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. Фиг. 2 е показва сравнение между VM и Docker контейнерите



Фиг. 2. Виртуални машини и Docker контейнерите

Виртуалните машини включват приложението, необходимите библиотеки или двоични файлове и пълна операционна система за гости. Изисква пълна виртуализация повече ресурси, отколкото контейнеризация

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 на контейнер.).

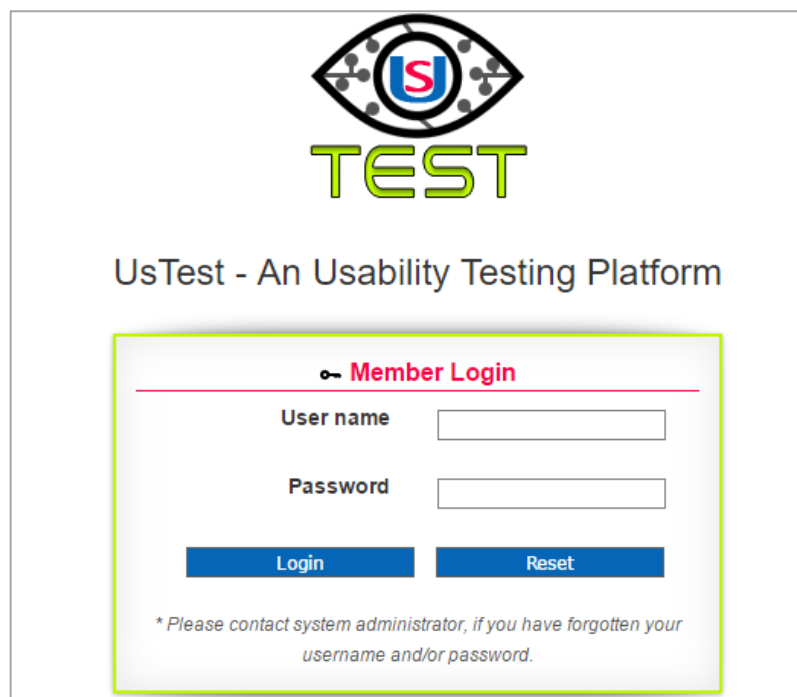
Виртуалните машини имат три основни слоя: инфраструктура, хост, операционна система, хипервизор и всички необходими библиотеки. За Docker слоерете са инфраструктурата, ОС и двигател за контейнери, който поддържа изолация, но споделя основните услуги на ОС. Тъй като контейнерите изискват много по-малко 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.

Информацията за организацията на страниците в модули и достъпът на потребителите до конкретните модули се съхранява в базата данни в специално създадени за целта таблици.

Проектиране на потребителския интерфейс на приложението

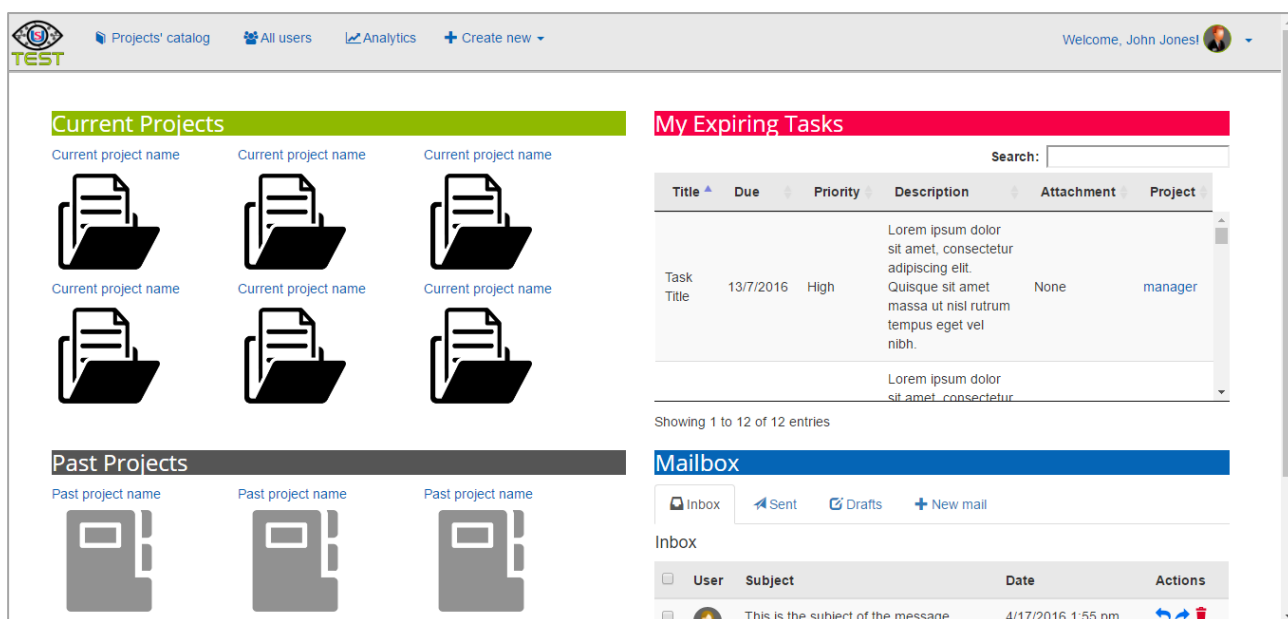
За спестяване на време и усилия при по-нататъшната разработка на прототипа е приложен еволюционния подход на прототипиране, а именно прототипът формира основите на новия продукт. Важно условие е да се използват средства, които ще бъдат прилагани при изграждане на окончателния вид на системата. Използвана е комбинация от средства за разработване на потребителски интерфейси на уеб приложения, а именно HTML, CSS, Bootstrap, jQuery, JavaScript. Прототипът се доближава до крайния вид на системата, като се характеризира с висока степен на детайлност и функционалност. Позволява на потребителите да добият представа за функционирането и визията на крайния продукт. Така се предоставя възможност в бъдеще да се извършат задълбочени тестове за ползваемост, на база на които може да се направи адекватна оценка за системата. Прототипът демонстрира стратегически важни компоненти на системата. Първоначалната реализация е на английски език, като се планира добавяне на другоезични версии на интерфейса, включително и българска.

Поради ограниченията на настоящата форма за представяне на проекта, то не е възможно да бъдат включени всички екрани. Ето защо се спираме само на някои от тях. На фиг. 3 е изобразен екранът, от който се осъществява входът в системата.




Фиг. 3. Вход в уеб базираното приложение за управление на проекти

На Фиг. 4 е показана стартовата страница на потребителите след успешно осъществен вход в системата.



Фиг. 4. Начална страница на потребителите след вход






На Фиг. 5 е изобразена страницата, на която се извежда списък с всички потребители в системата. Предоставя се възможност за филтриране на списъка по роля и ключови думи.


Projects' catalog
All users
Analytics
Create new
Welcome, John Jones!

All Users

Search
Type here...

Filter by user role:
☒ Show all
☐ Manager
☐ Researcher
☐ Tester

Profile picture	Username	User role	Email	Phone number	First Name	Last name	Joined
	manager	Manager	manager@ustest.com	+359 222 333	Manager	Name	2016/02/20
	lenny	Manager	manager1@ustest.com	+359 222 332	Manager	LName	2015/02/15
	danny	Researcher	example@ustest.com	+359 222 444	Researcher	Name	2015/06/12
	jolly	Tester	example@ustest.com	+359 222 555	Tester1	LName	2016/10/21
	taro	Tester	example@ustest.com	+359 222 556	Tester2	LName	2016/10/21

© UsTest - An Usability Testing Platform 2016 | Contacts for more information: r.nacheva[at]ue-varna.bg

Фиг. 5. Списък с всички потребители в системата

Заклучение

В заключение може да се каже, че чрез избрания набор от технологии се цели да се постигне съвместимост между различни платформи и преносимост на кода, като се осигури стандартна работна система, чрез която може да се развие конкретния проект. Усилията са съсредоточени върху разработването на изискванията, а не върху подготовката на инструментариума за разработка на приложението. За добра организация на проекта и осигуряване на лекота при бъдещото му развиване се следват конвенции в кодирането, включително бизнес логиката е разделена от потребителския интерфейс.

Трябва да се отбележи, че прототипът не е пълно функционален, а демонстрира ключови функционалности на уеб базирана система за оценяване на ползваемостта на мобилни приложения и по-конкретно, управлението на проекти. Като следваща логична стъпка от процеса на разработване е проверката на прототипа чрез тестване с потребители. Целта е да се докаже неговата практическа приложимост и ползваемост. В тестовите е подходящо да вземат участие представители на целевата аудитория, а именно дизайнери, разработчици, мениджъри на фирми и най-вече специалисти по ползваемост.

Монолитите приложения са лесни за:

- изграждане
- тестване
- разгръщане
- отстраняване на неизправности
- разширяване на мащаба
- биометрични данни.

Използвана литература

1. Куюмджиев, И., Одит на информационни системи. Дис. Варна. 2011
2. Наков, Св. И др., Въведение в програмирането с Java. Фабер, Велико Търново, 2009 г.
3. Layka, V., Learn Java for Web Development. Apress, 2014
4. Williams, N., Professional: Java® for Web Applications. John Wiley & Sons, Inc., Indianapolis, Indiana, 2014