УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Предраг Главаш ПР44/2017

Миграција монолитне веб апликације за надгледање радова над дистрибутивном електроенергетском мрежом на микросервисну

ПРОЈЕКАТ

- Примењено софтверско инжењерство (ОАС) -

Садржај

ОПИС РЕШАВАНОГ ПРОБЛЕМА

Потребно је мигрирати монолитну веб апликацију за надгледање радова над дистрибутивном електроенергетском мрежом под називом *Smart Energy* на микросервисно оријентисану.

Монолитне веб апликације су оне код којих се срж апликације налази унутар једног пројекта. Сама апликација може имати више слојева који енкапсулирају различите, логички одвојене скупове функционалности и тада се слојеви одвајају у посебне фасцикле. Цео доменски модел података монолитне апликације се налази у оквиру једног пројекта и обично постоји само једна база података. Монолитне апликације типично скалирају хоризонтално, што значи да се дуплицира цела апликација на посебне виртуелене машине или сервере. Предност монолитних апликација је у томе што је комуникација између слојева и компоненти апликације лакша као и постизање конзистентности података јер се сви подаци налазе на једном месту.

Код микросервисних апликација насупрот монолитним делови функционалности су одвојени у посебне сервисе који су потпуно одвојени једни од других а комуницирају међусобно путем *Http/Https* протокола или неког реда са порукама за асинхрону комуникацију. Сваки микросервис има свој посебни доменски модел података и по правилу мора бити довољно аутономан да може обавити одређени задатак уз што мање комуникације са осталим сервисима. Свака додатна зависност једног микросервиса од неког другог смањује његову стабилност и агилност јер падом сервиса од ког зависи постаје неупотребљив. Предност микросервиса је и у томе што је могуће скалирати сваки појединачни микросервис по потреби, што није случај са монолитном апликацијом коју је потребно целу ископирати како би се умножио број инстанци. [1]

Поред саме миграције на микросервисну архитектуру потребно је и контејнеризовати све микросервисе као и њихове базе података и оркестрирати их уз помоћ неког од контејнерских оркестратора. За комуникацију између микросервиса потребно је користити *DAPR .NET* библиотеку.

Контејнеризација ће омогућити независан развој и постављање нових верзија микросервиса и њихову високу портабилност између различитих рачунарских окружења тиме што ће све зависности и подешавања, заједно са извршним датотекама бити у оквиру слике контејнера. [2]

Оркестрација контејнера и дизајн апликације морају бити такви да омогуће несметано функционисање микросервиса у случају отказа неког другог микросервиса, као и несметано поновно подизање микросервиса након отклањања проблема или имплементације надоградње. Потпуна независност доменских модела података микросервиса у овом случају неће бити могућа због природе домена апликације где је све уско везано за инциденте у електроенергетској мрежи и радове на отклањању истих. Сходно томе потребно је прво пажљиво поделити доменски модел података како се као резултат миграције не би добио дистрибуирани монолит уместо микросервисне апликације.[3]

ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА И АЛАТА

При изради овог решења коришћени су следећи алати и технологије:

- ***ASP .NET Core 3.1*** – Радни оквир отвореног кода за израду веб апликација са подршком за више платформи. Коришћена верзија 3.1 има дуготрајну подршку компаније Мајкрософт (енг. *Microsoft*). [4] Овај радни оквир је коришћен за израду задње стране веб апликације.

- ***Angular 11*** **–** Радни оквир базиран на Јаваскрипту који омогућава израду предњих страна веб апликација које се састоје из више компоненти. Свака компонента у себи енкапсулира свој дизајн, изглед и логику и самим тиме ју је могуће користити више пута на различитим местима.[5]

- ***DAPR .NET*** **–** Пројекат отвореног кода, спонзорисан од стране Мајкрософта који омогућава асинхрону комуникацију између сервиса посредством *HTTP* протокола. Овај пројекат имплементира шаблон „приколице“ где је инстанца *DAPR* сервиса у посебном меморијском простору у односу на главни сервиса, а комуницира са главним сервисом преко портова 80 или 443. Микросервиси како би комуницирали међусобно обраћају се својим *DAPR* сервисима који ту комуникацију обављају за њих. [6]

- ***Docker desktop*** **–** Платформа за развој и контејнеризацију апликација где се све зависности и извршни код једне апликације пакују у контејнер који се касније може покретати на различитим машинама. Предност у односу на виртуелне машине је што не мора имати свој оперативни систем већ користи оперативни систем рачунара „домаћина“ на ком је покренут.[7]

- ***Visual Studio Code*** **–** Мајкрософтово радно окружење коришћено за развој предње стране апликације, укључује *IntelliSense* [8] подршку за лакше писање кода апликације.

- ***Visual Studio 2019*** **–** Мајкрософтово радно окружење коришћено за развој задње стране апликације, такође укључује *IntelliSense.*

- ***SQL Server Management Studio 2018*** **–** Мајкрософтов алат за управљање релационим базама података, превасходно *SQL* сервером. Омогућава повезивање са постојећим инстанцама сервера базе података и измену табела и података у табелама.[9]

- ***REST API*** **–** (енг. *Representational State Transfer Application Programming Interface*) Концепт за моделовање и приступ подацима апликације као објектима и колекцијама објеката. Подацима се приступа преко јасно дефинисаних и стандардизованих веза а подаци се шаљу и примају у текстуалном облику што омогућава слабу повезаност између клијентских и серверских апликација и високу флексибилност. [10]

- ***Swagger/Open API*** **–** Технолошки агностична спецификација *REST API* тачака приступа која се користи за њихово документовање, тестирање и касније лакше конзумирање. За сваку тачку приступа дефинише везу за њено позивање, глагол који употребљава, улазне и излазне податке као и могуће повратне кодове.[11]

- ***Angular material UI*** **–** Библиотека са готовим графичким компонентама које се могу користити у развоју предњих страна апликација уз *Angular* радни оквир пружа широк избор конфигурабилних компоненти које се често користе у веб апликацијама , а које би инжењери иначе морали сами да развијају.[12]

- ***Entity Framework Core*** **–** Технологија за приступ бази података и мапирање торки у базама података на објекте доменских модела података апликације, пружа ниво апстракције путем ког се врше упити ка бази података. [13]

ОПИС РЕШЕЊА ПРОБЛЕМА

Преглед постојећег монолитног решења

Пре описа процеса миграције монолитне апликације на микросервисну урадиће се преглед постојећег монолитног решења. Преглед неће обухватати све детаље њеног функционисања јер служи само за боље размевање базе кода са које је вршена миграција.

Предња страна монолитног решења је урађена у *Angular* радном оквиру као једнострана веб апликација (енг. *Single Page Application*). Дизајн предње стране је урађен претежно користећи *Angular Material UI* компоненте. Нерегистрованим корисницима се приказује страница са које је могуће регистровати се, пријавити се или пријавити квар. Након пријаве , корисницима је све до одјаве са сајта на свим рутама доступан хоризонтални и вертикални навигациони мени. Све форме за унос података су урађене по реактивној филозофији а у специфичним случајевима су написани и посебни валидатори података. Компоненте предње стране су логички груписане у модуле, а по потреби модули користе и компоненте других модула. Делови кода за комуникацију са задњом страном су енкапсулирани у сервисе који се инјектују кроз конструкторе компоненти. Код приказа већег броја података имплементирано је парцијално довлачење података (по странама) како учитавање приказа не би временски превише трајало.

Задња страна се урађена као *REST API* сервис посредством .NET Core 3.1 радног оквира. Архитектура задње стране је урађена у 3 слоја:

- Слој контролера
- Слој сервиса
- Слој података

Слој контролера је задужен за пријем и обраду *HTTP* захтева и враћање адекватних стаус кодова у зависности од успешности извршења захтева. Слој контролера не врши валидације података поред стандардног бацања изузетка у случају немогућности десеријализације података од стране радног оквира. Операције над подацима или њихово прибављање, слој контролера делегира сервисном слоју.

Слој сервиса енкапсулира бизнис логику апликације и све потребне валидације, податке прибавља позивајући контекст базе података који је уједно и слој података у овом случају. У случају да је валидација података неуспешна баца се изузетак, који се хвата у слоју контролера и на основу њега враћа адекватан одговор. Сервиси по потреби сарађују како би се избегло дуплирање кода.

Безбедност предње стране апликације почива на заштитама путања и валидацији токена добијених од задње стране. Корисницима је допуштено да се региструју на систем креирајући нови налог или повезујући постојећи налог са друштвене мреже Фејсбук (енг. *Facebook*) или Гугла (енг. *Google*). Битно је напоменути да апликација верује искључиво токенима издатим од стране њене задње стране, тако да се Фејсбукови и Гуглови токени користе само као доказ да корисник има налог на тим веб сајтовима. Такви токени се прво валидирају позивајући релевантне спољне сервисе па уколико су валидни кориснику се издаје нови токен са потписом задње стране апликације са којим даље може вршити захтеве. Токенима се при валидацији и на задњој и на предњој страни проверава датум истека који је по правилу 20 минута након издавања. Поред провере датума истека важења токена, на задњој страни се проверава потпис издаваоца као и публика којој је намењен како би се избегло подметање лажних токена или крађа од стране неауторизованих домена.

Подела модела података

Први и најважнији корак при миграцији монолитне апликације јесте адекватна подела доменског модела података тако да је сваки добијени поддомен довољно независан од остатка података да се помоћу њега може репрезентовати скуп функционалности које се могу извршавати и у случају недоступности других података. У случају ове апликације то није сасвим могуће као што је и раније поменуто, али пажљивим разматрањем се могу издвојити 3 поддомена који приближно задовољавају ове критеријуме (слика 1.).

слика 1. – Идентификовани поддомени монолитног доменског модела података

Људски ресурси чине поддомен везан за људе који користе, одржавају и управљају електроенергетским системом кроз апликацију. Задаци микросервиса представљеног овим моделом података су :

- Управљање корисничким налозима
- Аутентификација и ауторизација
- Креирање и управљање екипама
- Додавање и управљање потрошачима електричне енергије

Из друге тачке задатака овог микросервиса може се закључити да је од његовог функционисања зависан читав систем, јер без адекватне аутентификације и ауторизације цела апликација не може функционисати. Ово значи да би његовим испадом практично већина апликације била неупотребљива, међутим овај задатак се мора доделити неком од микросервиса како би се избегла комплексност дистрибуирања оваквих операција. Разлог за доделу тако важног задатка управо овом микросервису јесте потреба за приступом корисничким подацима сваки пут при аутентификацији и ауторизацији.

Физички елементи су поддомен који репрезентује уређаје електроенергетске мреже и локације које чине електроенергетски систем. Локације су везане и за потрошаче те се намеће питање због чега подаци о локацијама нису смештени заједно са подацима о потрошачима. Одговор лежи у већој потреби да се у неком моменту зна на којој тачно локацији и географским координатама се налази покварени уређај, него да се зна адреса потрошача. Једноставније речено, ако би били познати покварени уређаји, али не и њихове тачне локације (из разлога испада микросервиса са локацијама) квар никад не би могао бити отклоњен.

Задаци микросервиса представљеног овим моделом података су

- Управљање свим локацијама у систему
- Додавање, измена и брисање уређаја електроенергетске мреже

Документи су највећи поддомен монолитног модела података који представља скуп докумената и прилога потребних да се неометано одвијају радови над електроенергетском мрежом. Видно је већи од остала два скупа те је логично запитати се зашто није подељен на више подскупова. Даља подела овог

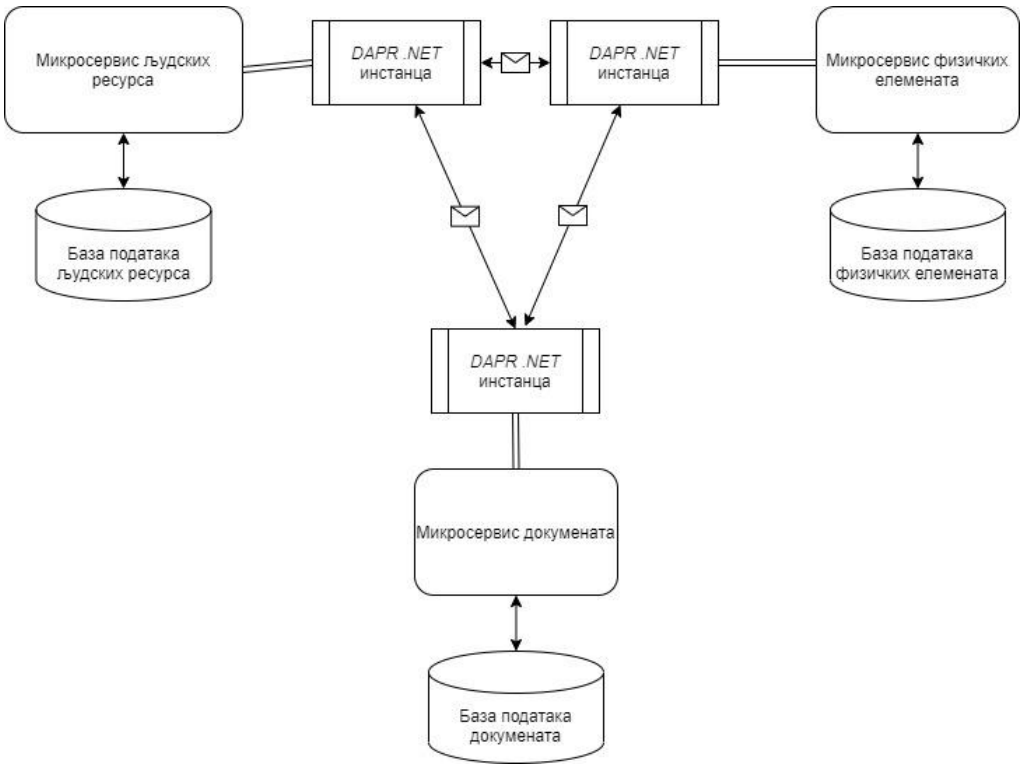
поддомена не би била адекватна из разлога што постоје везе у редоследу настајања докумената, инцидент претходи формирању налога за рад, налог за рад је основа за креирање плана рада итд... Поред тога, не постоји потреба за скалирањем апликације вертикално у правцу појединачних сервиса за сваки од типова докумената.

Задаци микросервиса докумената су:

- Управљање позивима/пријавама квара
- Креирање, измена и брисање инцидената на основу пријава квара
- Креирање, измена и брисање налога за рад на основу инцидената
- Креирање, измена и брисање планова рада на основу налога за рад
- Креирање, измена и брисање безбедносних докумената на основу планова рада
- Управљање мултимедијалним прилозима
- Праћење историје промена докумената

Архитектура микросервисне апликације

На основу претходне поделе домена података на подскупове креиране су *REST API* веб апликације које представљају сваки од микросервиса. Што се тиче базе података, коришћен је „база података по сервису“ (енг. *Database per service*) приступ, где је за сваки микросервис направљена посебна база података. Овакав приступ са једне стране ствара слабу везу између микросервиса, али са друге стране уноси комплексност дистрибуираних трансакција и отежава одржавање конзистентности података у базама. [14] Поред тога, сваки микросервис има инстанцу *DAPR .NET* сервиса који му служи за комуникацију са другим микросервисима. Скица архитектуре се може видети на слици 2.



слика 2. – Архитектура микроссервисне апликације

Креирање база података микросервиса

Приликом креирања база података микросервиса уклоњена су сва ограничења референцијалних интегритета која су у монолитном решењу била између табела које су сада у различитим микросервисима. Страни кључеви из којих произилазе ова ограничења су сада сведени на обична обележја тако да практично на нивоу базе података сада не постоји никаква валидација за њих, валидација се преместила на апликативни ниво.

Овакво растављање база података је додатно оптеретило апликативни ниво јер је у неким случајевима потребно урадити спајање табела које се налазе у различитим микросервисима. Уколико један од микросервиса није у функцији није могућа адекватна агрегација података, те је како би се колико-толико очувала функционалност апликације потребно уметнути неке податке уместо недостајућих који ће јасно означавати да праве податке није било могуће добавити. Пример овакве безбедне агрегације може се видети на слици 3.

```
foreach (UserDto us in returnValue.Users)
{
	try
	{
		us.Location = await _daprClient.InvokeMethodAsync<LocationDto>(HttpMethod.Get,
		"smartenergyphysical", $"/api/locations/{us.LocationID}");

	}catch
	{
		us.Location = new LocationDto();
	}
}
```

слика 3. – Пример безбедне агрегације података, добављање локација корисника

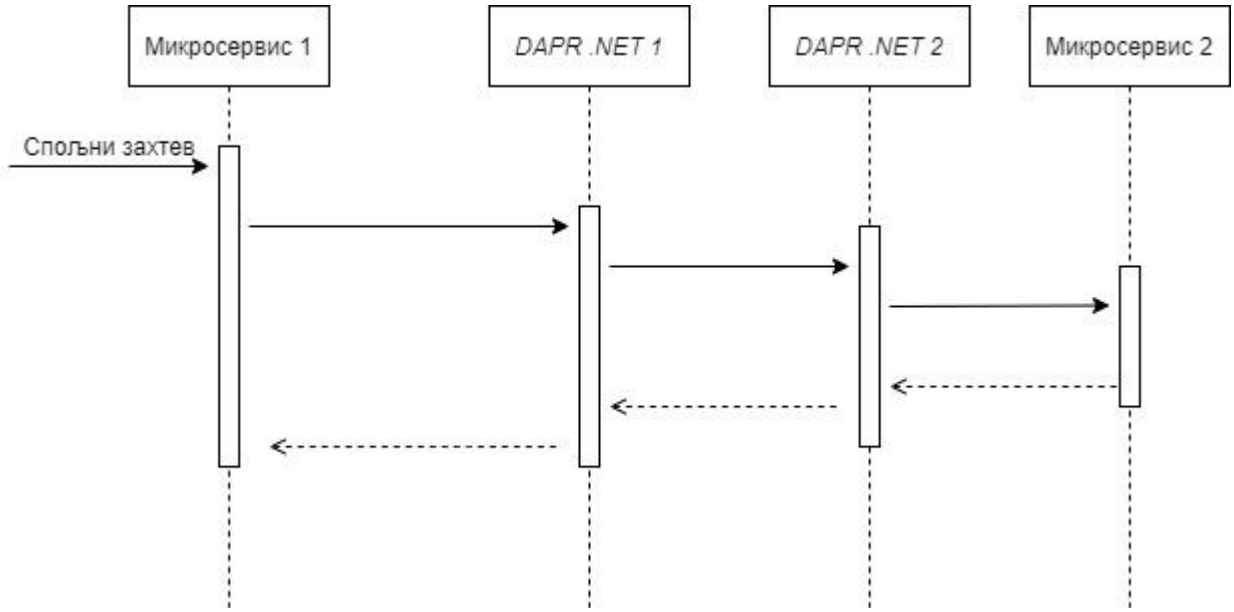
Уклањање ограничења референцијалних интегритета намеће додавање нових валидација у микросервисе како би се осигурала конзистентност података. Потребно је сваки пут проверити да ли вредност обележја страног кључа који се уписује у неку табелу, а који показује на табелу у другом микросервису заиста и постоји као примарни кључ табеле у другом микросервису. Овако нешто наравно није могуће уколико други микросервис од ког се зависи није доступан, што значи да се у том случају морају одбацити чак и потенцијално валидне операције над базом података и превентивно забранити на апликативном нивоу. На слици 4 се може видети пример валидација референцијалног интегритета локације корисника и одбацивање трансакције у случају невалидности.

```
public async Task<UserDto> InsertAsync(UserDto entity)
{
	try
	{

		LocationDto userLocation = await
		_daprClient.InvokeMethodAsync<LocationDto>(HttpMethod.Get,
		"smartenergyphysical", $"/api/locations/{entity.Location.ID}");

	}
	catch
	{
		throw new Exception("User does not have a valid location");
	}

	User user = _mapper.Map<User>(entity);
...
```

слика 4. – Пример валидације референцијалних интегритета на апликативном нивоу

Комуникација између микросервиса

За комуникацију између микросервиса се користи *DAPR .NET* модул са подршком асихнроне комуникације. Подршка асинхроне комуникације је јако битна за микросервисне апликације јер смањује њихову међусобну зависност и повезаност.[15] Приликом иницирања комуникације микросервис се обраћа својој *DAPR .NET* инстанци (видети слику 2), која затим позива *DAPR* инстанцу другог микросервиса како би од ње добавила податке. Други микросервис предаје податке својој *DAPR* инстанци и ланац уназад се наставља до позиваоца. Битно је напоменути да микросервиси могу потраживати податке искључиво преко *REST API* тачака приступа других микросервиса, навођењем одговарајућих путања при позивању. У случају непостојања траженог ресурса, отказа позваног микросервиса или сличног проблема, бациће се изузетак. Скица тока комуникације налази се на слици 5.



слика 5. – Скица тока комуникаиције између миркосервиса

DAPR инстанце се на сервисну апликацију повезују преко портова 80 или 443, први за *HTTP* а други за *HTTPS.* У случају да микросервисна апликација није доступна у моменту покушаја повезивања, *DAPR* ће остати да је сачека у блокирајућем режиму. Портови преко којих *DAPR* сервиси међусобно комуницирају су подразумевано 50001 , али их је могуће променити при покретању како не би дошло до преклапања са другим сервисима или апликацијама.

Модел података који се размењује између микросервиса није доменски, већ транспортни (енг. *Data Transfer Object, DTO*) што значи су везе између сервиса додатно ослабљене, а могућност надоградње индивидуалних сервиса побољшана. Наравно, овим се наметнула потреба да се увек при комуникацији мапира доменски модел података сваког микросервиса на транспортни који морају познавати сви микросервиси како би уопште могли комуницирати. На први поглед додавање још једног модела података о ком је потребно бринути непотребно усложњава софтвер, међутим дугорочно се постиже флексибилност у развоју индивидуалних микросервиса. Неки микросервис може, примера ради, почети да добавља податке са неке спољне тачке, а затим их комбинује са онима у својој бази података и тако шаље као одговор, ово наравно не би било могуће када би се свуда размењивали доменски модели података. Неки микросервис такође може слати само део података сваке класе, на пример за приказ корисника система није потребно (а ни безбедно) добављати осетљиве податке као што су лозинке, па је управо због оваквих случајева уведен транспортни модел.

Контејнеризација решења

Финални корак при миграцији апликације на микросервисе јесте контејнеризација микросервиса и њихова оркестрација помоћу *Docker-compose* оркестратора. Овај оркестратор је одабран због мањих хардверских захтева и веће доступности адекватне документације за његову употребу у комбинацији са већ коришћеним технологијама за израду апликације. Према архитектури решења, јавља се потреба за следећим контејнерима:

- Контејнер са сликом микросервиса људских ресурса
- Контејнер са сликом микросервиса физичких елемената
- Контејнер са сликом микросервиса докумената
- Три контејнера са сликом *SQL* сервера базе података за сваки микросервис
- Три контејнера са сликом *DAPR .NET* сервиса

```
smartenergyusers:
    image: ${DOCKER_REGISTRY-}smartenergyusers
    build:
      context: .
      dockerfile: SmartEnergyUsers/Dockerfile
    ports:
      - "51000:50001"
      - "44372:80"
    depends_on:
      - users-db

  users-db:
    image: mcr.microsoft.com/mssql/server
    ports:
      - "1401:1433"
    environment:
      ACCEPT_EULA: "Y"
      SA_PASSWORD: "Your+password123"

  smartenergyusers-dapr:
    image: "daprio/daprd:latest"
    command: [ "./daprd", "-app-id", "smartenergyusers", "-app-port", "80" ]
    depends_on:
      - smartenergyusers
    network_mode: "service:smartenergyusers"
```

слика 6. – Исечак конфигурације за оркестрацију микросервиса људских ресурса

На слици 6 се види исечак конфигурације за оркестрирање контејнера. Исечак приказује конфигурацију за оркестрирање контејнера са сликом микросервиса људских ресурса, контејнера са његовом базом података и контејнером *DAPR* сервиса. Код оркестрације сваког микросервиса потребно је подесити мапирање релевантних портова унутар контејнера на оне видљиве споља. У овом случају то су портови 50001 и 80, чији је значај детаљно објашњен у поглављу о комуникацији између микросервиса. Контејнер микросервиса зависи од контејнера базе података у коме се покреће слика *MSSQL* сервера, што значи да се чека на покретање контејнера са базом података како би се започело покретање контејнера микросервиса.

Серверу базе података се може приступити на порту 1433, али је неопходно мапирање на неки од портова споља како би се могла остварити комуникација са њим, у овом случају порт 1401. За сваки сервер базе података порт 1433 се мора мапирати на различит спољни порт како не би дошло до преклапања. Поред подешавања порта, за сервер базе података неопходно је дефинисати променљиве окружења преко који се прослеђује лозинка администратора базе података.

Инстанца *DAPR* сервиса у свом контејнеру се покреће након микросервиса како би могла да се повеже са њим преко порта дефинисаног након „*-app-port* “ команде у конфигурацији. Још једно битно подешавање је назив инстанце, који се дефинише након „*-app-id*“ команде. Касније позивање овог микросервиса од стране неког другог ће бити могуће само ако се познаје назив инстанце.

При оркестрацији овако конфигурисаних контејнера може се јавити проблем повезивања микросервиса са базом података при покретању на слабијем хардверу. Сам контејнер микросервиса ће се покренути након покретања контејнера базе података, али то не значи да ће у том моменту и сам сервер базе података бити потпуно иницијализован и спреман за употребу. Сходно овоме, при раном покушају повезивања на базу података због миграције модела јавиће се изузетак и оркестрација се неће адекватно извршити.

Овај проблем је решен на апликативном нивоу додавањем аутоматских узастопних покушаја извршавања миграције. Број покушаја је постављен на три, а паузе између покушаја су 60 секунди. Број покушаја и дужина паузе су експериментално одређени.

ПРЕДЛОЗИ ЗА ДАЉА УСАВРШАВАЊА

Представљено решење креирања апликације са микросервисном архитектуром и њена контејнеризација а затим и оркестрација је урађена са превасходно акцентом на адекватну поделу доменског модела података. Велика пажња је посвећена истраживањима у правцу начина адекватне поделе монолитних решења на микросервисна. У овом процесу су, због тога, изостављени битни елементи безбедности апликација и у том правцу постоји простор за унапређења.

Због недостатка одговарајућег хардвера откривени су проблеми у оркестрацији сервера базе података и микросервиса, који иначе не би били детектовани. Због робусности решења и стабилности на различитом хардверу одлучено је да се ови проблеми ипак реше на апликативном нивоу.

Аутентификација и ауторизација у монолитној апликацији је токен базирана, где су се при том у сваки токен уметали подаци о валидним конзументима токена. У микросервисном решењу, тако нешто није урађено и не постоји никаква валидација у том смислу. Било која трећа страна која дође у посед овако издатог токена, може га потенцијално злоупотребити да се представи као валидан конзумент. Поред тога, нигде се не врши инвалидација токена након одјављивања корисника из апликације па су могуће злоупотребе и у том погледу.

Приликом контејнеризације микросервиса са документима није било могуће успешно подесити постојећи сервис за скенирање датотека на вирусе, па у односу на монолитно решење постоји озбиљан безбедносни ризик доспевања нежељеног софтвера или рачунарског вируса на сервер. Потребно је у наредним итерацијама побољшавања софтвера дати приоритет проналажењу адекватног, лако конфигурабилног антивирусног решења које може функционисати у контејнерском окружењу.

Приликом регистрације нових корисника, нуди се избор свих постојећих локација у система на основу ког корисник бира своју адресу. Такође, при регистрацији нових чланова екипа нуди им се избор једне од постојећих екипа у систему. Овакав приступ открива податке о унутрашњости система неауторизованим корисницима и може довести до злоупотреба у физичком домену. Уместо избора локације приликом регистрације, кориснику се може дозволити да упише своју адресу ручно. Наравно, валидација корисничких адреса мора бити адекватна и стриктна на апликативном нивоу, али се такође мора захтевати и потврда од стране администратора система као додатна сигурност. Избор екипа при регистрацији је потребно пребацити на време након потврде корисникове регистрације од стране администратора система, тиме би се спречило да непознати корисници стекну увид у екипе у систему.

Додатни безбедносни пропуст представља недостатак чувања записа о акцијама које су корисници покушали у систему или догађајима који су се десили. Записивање акција би, поред информација о потенцијалним неовлашћеим покушајима приступа систему допринело и лакшем поправљању грешака у софтверу и побољшању корисничког искуства.

Микросервисно решење за разлику од монолитног мора на апликативном нивоу вршити агрегацију података. Такав приступ је у општем случају спорији него спајање табела у бази података те су перформансе оваквог решења слабије. У будућности је потребно истражити могућности побољшања перформанси дистрибуираног спајања табела кроз ефикасне алгоритме.

Тренутно решење не подразумева репликацију микросервиса, те у случају отказа једног од њих не постоји резерва која га може заменити. Потребно је са растом апликације размислити о преласку на другачију технологију оркестрације контејнера. *Kubernetes* је валидна замена за тренутни оркестратор и пружа могућност креирања више реплика микросервиса, као и балансирање оптерећења између њих.

Литература

[1] ***https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures***, ***Архитектуре апликација***
[2] ***https://www.docker.com/resources/what-container***, ***Контејнеризација***
[3] ***https://codeopinion.com/rest-apis-for-microservices-beware/***, ***Дистрибуирани монолит***
[4] ***https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1*** , ***Технологија задње стране апликације***
[5] ***https://angular.io/guide/what-is-angular*** , ***Технологија предње стране апликације***
[6] ***https://docs.microsoft.com/en-us/dotnet/architecture/dapr-for-net-developers/***, ***DAPR пројекат***
[7] ***https://docs.docker.com/get-started/overview/*** , ***Docker desktop***
[8] ***https://code.visualstudio.com/docs/editor/intellisense*** , ***IntelliSense***
[9] ***https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15*** , ***SQL Server Management Studio***
[10] ***https://www.ibm.com/cloud/learn/rest-apis*** , ***REST API***
[11] ***https://www.ibm.com/docs/es/app-connect/11.0.0?topic=apis-swagger*** , ***Swagger***
[12] ***https://material.angular.io/*** , ***Angular Material UI***
[13] ***https://docs.microsoft.com/en-us/ef/core/*** , ***Entity Framework Core***
[14] ***https://microservices.io/patterns/data/database-per-service.html*** , ***База података по сервису***
[15] ***https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture*** , ***Асихнирона комуникација између микросервиса***