УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Елена Попадић ПР91/2018

Миграција монолитне веб апликације за доставу хране на микросервисну

ПРОЈЕКАТ

- Примењено софтверско инжењерство (ОАС) -

Нови Сад, 04.08.2022.

Садржај

ОПИС РЕШАВАНОГ ПРОБЛЕМА

Потребно је мигрирати монолитну веб апликацију за доставу хране под називом *Delivery App* 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 молонитне апликације се налази у оквиру једног пројекта и обично постоји само једна база података. Предност оваквих 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 монолинтним 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 молонитном апликацијом коју је потребно целу ископирати како би се умножио број инстанци. [1]

ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА И АЛАТА

При изради овог решења коришћени су следећи алати и технологије:

- **Angular 13.3.3** – 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. [2]
- **Visual Studio Code** – Мајкрософтово радно окружење коришћено за развој предње стране апликације, укључује *IntelliSense* [3] подршку за лакше писање кода апликације.
- **Visual Studio 2019** – Мајкрософтово радно окружење коришћено за развој задње стране апликације, такође укључује *IntelliSense*.
- **ASP .NET Core 5.0** – Радни оквир отвореног кода за израду веб апликација са подршком за више платформи. Коришћена верзија 5.0 има дуготрајну подршку компаније Мајкрософт (енг. Microsoft). [4] Овај радни оквир је коришћен за израду задње стране веб апликације.
- **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. [5]
- **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. [6]
- **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. [7]
- **Ocelot API Gateway** – Главна функционалност *Ocelot API Gateway-a* је да преузме долазне ХТТП захтеве и проследи их на низводну услугу, тренутно као још један ХТТП захтев. Оцелот описује усмеравање једног захтева ка другом као “*ReRoute*”. [8]
- **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]

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 или се пријавити. Након пријаве, корисницима је све до одјаве са сајта на свим рутама доступан хоризонтални 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 је урађена као *REST API* сервис посредством *.NET Core 5.0* радног оквира. Архитектура задње стране је урађена у 3 слоја: слој контролера, слој сервиса и слој података.

Слој контролера је задужен за пријем и обраду *HTTP* захтева и враћање адекватних статус 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 Гугла (енг. *Google*). Битно је напоменути да апликација верује искључиво токенима издатим од стране њене задње стране, тако да се 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 2 поддомена који приближно задовољавају ове критеријуме (слика 1.).

слика 1. - Поддомени монолитног доменског модела података

Људски ресурси чине поддомен везан за људе који поручују храну, достављају храну и управљају апликацијом за доставу. Задаци микросервиса представљеног овим моделом података су:

- Управљање корисничким налозима
- Аутентификација и ауторизација
- Креирање и управљање јелима
- Додавање, управљање и верификација достављача
- Слање мејла као потврде о успешној верификацији

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.

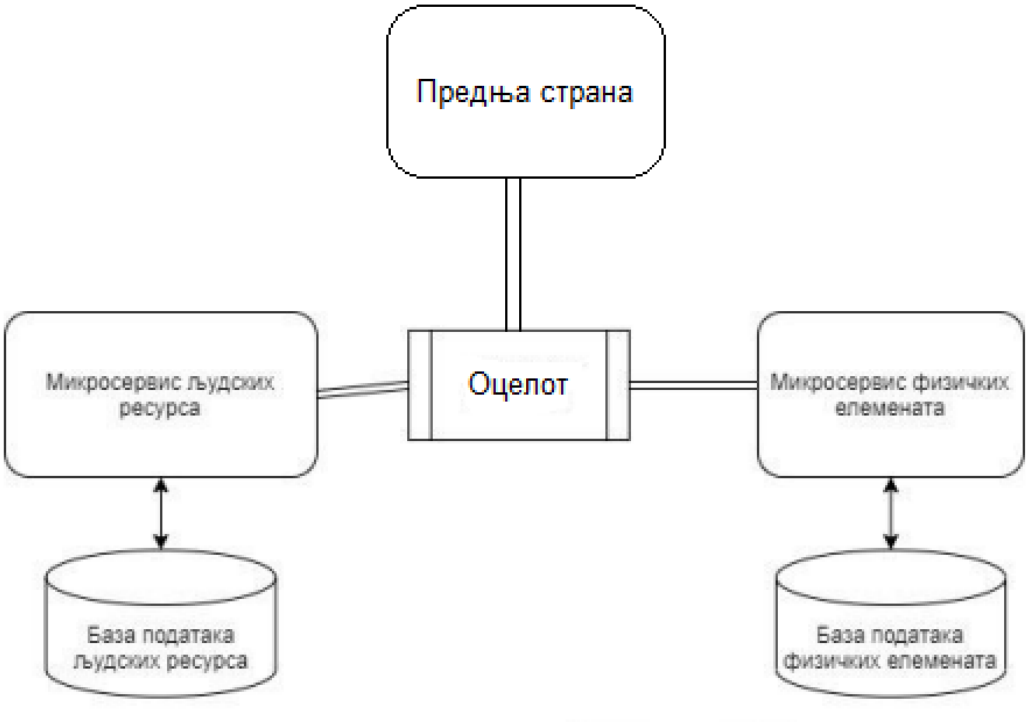
Физички елементи су поддомен који репрезентује производе и поруџбине које чине систем за доставу хране. Задаци микросервиса представљеног овим моделом података су:

- Преглед свих поруџбина у систему (прошлих, тренутних и непреузетих)
- Преузимање поруџбине
- Додавање, измена и брисање производа односно јела
- Добављање свих прозвода у систему
- Праћење историје поруџбина

Архитектура микросервисне апликације

На основу претходне поделе домена података на подскупове креиране су *REST API* веб апликације које представљају сваки од микросервиса (слика 2). 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. [10] Поред тога, предња страна не сме директно да комуницира са микросервисима, већ то ради преко *Ocelot API Gateway* који рерутира захтеве ка индивидуалним микросервисима.



слика 2. Архитектура микросервисне 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.

Комуникација између предње стране и микросервиса

За комуникацију између предње стране и микросервиса користи се *Ocelot API Gateway*. Приликом иницирања комуникације предња страна се обраћа *Ocelot API Gateway-y*, који затим прослеђује захтеве ка индивидуалним микросервисима, у зависности с којим типом података се ради. 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.

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.

ПРЕДЛОЗИ ЗА ДАЉА УСАВРШАВАЊА

Представљено решење креирања апликације са микросервисном архитектуром и њена 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.

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. Kubernetes је валидна замена за тренутни оркестратор и пружа могућност креирања више реплика микросервиса, као и балансирање оптерећења између њих.

ЛИТЕРАТУРА

[1]https://microservices.io/ , Архитектуре апликација

[2]https://angular.io/guide/what-is-angular , Технологија предње стране апликације

[3]https://code.visualstudio.com/docs/editor/intellisense , IntelliSense

[4]https://docs.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-5.0?view=aspnetcore-6.0 , Технологија задње стране апликације

[5]https://www.ibm.com/cloud/learn/rest-apis , REST API

[6]https://www.ibm.com/docs/es/app-connect/11.0.0?topic=apis-swagger , Swagger

[7]https://docs.microsoft.com/en-us/ef/core/ , Entity Framework Core

[8]https://docs.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/implement-api-gateways-with-ocelot , Ocelot API Gateway

[9]https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15 , SQL Server Management Studio

[10]https://microservices.io/patterns/data/database-per-service.html , База података по сервису