Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/28BE10Stosic

MIKROSERVISNA ARHITEKTURA NA PRIMERU WEB APLIKACIJE ZA PERSONALNE TRENERE

MICROSERVICE ARCHITECTURE APPLIED TO THE PERSONAL TRAINER WEB APPLICATION

Petar Stošić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – RAČUNARSTVO I AUTOMATIKA

Kratak sadržaj – *U ovom radu izneta je teorijska osnova o različitim tipovima softverske arhitekture, iznete su ključne karakteristike kao i prednosti i mane svakog od njih. Fokus je bio na mikroservisnoj arhitekturi sistema koja predstavlja najpopularniju, ali i najčešće pogrešno shvaćenu i pogrešno korišćenu arhitekturu.*

Ključne reči: *Mikroservisna arhitektura, monolitna arhitektura, saga patern, tipovi interprocesne komunikacije.*

Abstract – *The paper describes microservice architecture, other types of software system architecture, their benefits but also it shows their problems as well. It explains what decisions developers need to make on the way when choosing the right architectural approach of their project. Microservice architecture is most popular nowadays, but it really has some high financial and organizational impacts, which needs to be foreseen.*

Keywords: *Microservice architecture, monolite architecture, saga pattern, interprocess communication patterns.*

1. UVOD

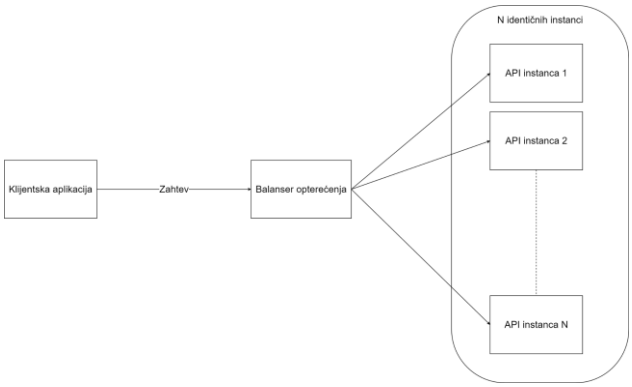
U današnjem dinamičnom svetu informacionih tehnologija, sposobnost brze i efikasne dostave softverskih rešenja postaje sve važnija. U potrazi za boljim performansama, skalabilnošću i olakšanim održavanjem, razvijaju se različite arhitekture sistema [1]. Jedna od njih koja se sve više ističe je mikroservisna arhitektura.
Ova inovativna arhitektura omogućava razdvajanje velikih aplikacija na manje, samostalne i međusobno povezane delove, poznate kao mikroservisi. U poređenju sa tradicionalnom monolitnom arhitekturom, mikroservisna arhitektura donosi brojne prednosti, ali takođe nosi i određene izazove i mane.

2. MONOLITNA ARHITEKTURA

Jedna od bitnijih karakteristika softvera baziranog na monolitnoj arhitekturi jeste ta da je njegova čitava funkcionalnost obuhvaćena (enkapsulirana) jednom aplikacijom, pa tako njeni moduli ne mogu biti izvršavani odvojeno.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Aleksandar Kupusinac, red. prof.

Ovaj tip arhitekture softvera smatra se usko povezanom (eng. tightly-coupled), tako da se čitava logika oko obrade zahteva izvršava u okviru jednog procesa. Korišćenjem osnovnih jezičkih struktura i osobina, moguće je ovakve aplikacije unaprediti u smislu strukture, zavisnosti i performansi.
Ovakve sisteme uz određena ograničenja moguće je horizontalno skalirati. Horizontalno skaliranje ovakvih aplikacija najčešće se implementira korišćenjem servisa za balansiranje sistema (eng. Load balancer-a) kao što prikazuje Slika 1. Ovakav tip aplikacije može poslužiti za kreiranje inicijalne verzije i upoznavanjem kompleksnosti sistema kao i granica između različitih komponenata. Međutim, kada projakat i broj ljudi u timu poraste, ovakav tip arhitekture aplikacije počinje da pokazuje svoje nedostatke.
Slika 1. prikazuje način potencijalnog horizontalnog skaliranja aplikacije bazirene na monolitnoj arhitekturi.



Slika 1. *Primer skaliranja monolitne aplikacije*

2.1. Prednosti monolitne arhitekture

Monolitna ahitektura ima određene prednosti. Naime, ovakav pristup ogranizacije softverskog sistema pogodniji je za brzi razvoj aplikacija čiji je glavni cilj brzi izlazak na tržište i na kojima radi manji tim ljudi. Veličina tima je jako bitna jer se čitav kod kod monolitnih aplikacija nalazi na jednom repozitorijumu, i čitava aplikcaija izvršava u okviru istog procesa. Da ovakav pristup ne bi krenuo u pravcu toga da se izgubi ikakva arhitektura, jako je bitno usredsrediti se na to da se izoluju komponente sistema. Ukoliko se komponente jasno izdvoje i definišu u okviru monolitne arhitekture, kasnija evolucija ka mikroservisnoj arhitekturi će biti dosta

olakšana. Još jedna bitna prednost ovog načina organizacije jeste dosta lakši proces (eng. build-ovanja) i (eng. release-ovanja) aplikacije. Naime potrebno je kreirati samo jedan paket i odraditi (eng. deploy).

2.2 Problemi monolitne arhitekture

Osim prednosti, ovaj tip arhitekture sistema nosi sa sobom i neke probleme.

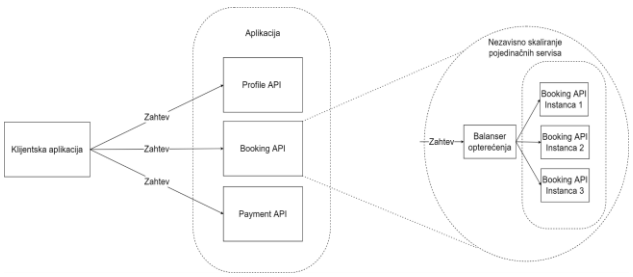
- Poteškoće sa skaliranjem
- Sporiji razvoj i isporuka kada su u pitanju veliki razvojni timovi
- Tehnička raznolikost je veoma ograničena
- Problemi održavanje i kompleksnost projekta vremenom postaju sve veći.
- Tehnički dug (eng. Tech debt) vremenom postaje sve veći.

3. MIKROSERVISNA ARHITEKTURA

Mikroservisna arhitektura predstavlja pristup kreiranja softverskog rešenja koje se sastoji iz više malih servisa koji svojim odgovornostima upotpunjavaju ukupnu funkcionalnu celinu aplikacije. Svaki od tih servisa, trebao bi biti nezavistan i izolovan proces, koji komunikaciju sa ostalim servisima ostvaruje nekim od mehanizama za među procesnu komunikaciju. Jedan od najčešće korišćenih mehanizama je HTTP protokol. Mikroservisi se kreiraju tako da predstavljaju neku od odvojenih celina koje grade određeni poslovni sistem. Idealno je da budu što je moguće više samostalni i da imaju što manje zavisnosti ka ostalim servisima. Najčešće se za ovakve sisteme konfigurišu automatizovani procesi za (eng. build) i (eng. release) fazu. Na taj način omogućava se pouzdan sistem razvoja, i sprečava se uticaj ljudskog faktora, gde bi zbog većeg broja servisa bilo gotovo nemoguće posao odraditi bez greške.

Dekompozicijom monolita na mikro servise, postiže se i veća otpornost sistema na greške. Ukoliko jedan od servisa ima problem u određenom trenutku, ostatak sistema (koji ne zavisi od problematičnog dela) može neometano da nastavi sa radom. U monolitu, ukoliko bi postojao sličan problem, čitav sistem bi bio ugrožen.

Slika 2. prikazuje primer sistema zasnovanog na mikroservisnoj arhitekturi, kao i mogućnosti nezavisnog skaliranja servisa koji ima veće opterećenje od ostalih.



Slika 2. *Sistem zasnovan na mikroservisnoj arhitekturi*

3.1 Prednosti mikroservisne arhitekture

Mogu se izdvojiti nekoliko ključnih prednosti koje karakterišu mikroservisnu arhitekturu sistema:

- Omogućava implementiranje (eng. CI/CD)-a velikih i kompleksnih aplikacija
- Servisi su najčešće mali i lako održivi
- Servisi se mogu razvijati nezavisno jedan od drugog
- Servisi se mogu skalirati nezavisno
- Omogućava autonomiju timova
- Omogućava eksperimentisanje i prihvatanje novih tehnoligija
- Karekteriše je bolja tolerancija grešaka

3.2 Problemi mikroservisne arhitekture

I ova arhitektura poseduje više ozbiljnih problema koje treba pravilno uočiti, identifikovati i rešiti. Veći deo tih problema je takav da je moguće rešiti ga, ali generalno prilikom projektovanja ovakvih sistema treba biti veoma oprezan i upoznat sa domenom problema koji se rešava. Neki od ključnih problema ove arhitekture su:

- Pronalaženje pravog skupa servisa
- Distribuirani sistemi su dosta zahtevni i kompleksni i čine sve aspekte razvoja softvera dosta težim
- (eng. Deploy-ovanje) novih stvari koje uključuju više od jednog servisa je dosta zahtevno i traži dobru koordinaciju i planiranje
- Donošenje odluke kada preći na mikroservisnu arhitekturu.

4. PROJEKTOVANJE SISTEMA METODOM DEKOMPOZICIJE

S obzirom da aplikacije postoje kako bi izvršavale zahteve, prvi korak bi upravo trebao biti identifikacija zahteva na apstraktnom nivou, kao i njihova formalizacija u okviru (eng. User story-a). Ove zahteve možemo definisati u vidu sistemskih operacija. Sistemska operacija je apstrakcija konretnog zahteva i kao takva omogućava nam da na ovom nivou nemamo previše nepotrebnih detalja koji će doći kasnije u procesu konkretizacije. Sistemska operacija može se podeliti na dva tipa:

- prvi tip su operacije koje menjaju stanje sistema, nazivaju se komande (eng. Commands)
- drugi tip operacije onaj koji ne menja stanje sistema već potražuje stanje sistema, nazivaju se upiti (eng. Queries).

Ponašanje svake komande je definisano na nivou abstraktnih domanskih modela, koji su takođe preuzeti iz zahteva. Sistemske operacije su na taj način svojevrsni arhitektonski scenariji, koji ilustruju kako servisi međusobno komuniciraju. Nakon identifikacije zahteva koje sistem mora da podrži, sledeći korak je identi- fikacija servisa. U ovom trenutku postoji više načina na koji se tom problemu može pristupiti a dva najčešća su:

- Na osnovu domenskog modela (eng. Domain Driven) na manje pod domene
- Na osnovu stvari koje čine biznis i koje on može da podrži (eng. Business capabilities)

Suština oba pristupa je da se na kraju servisi organizuju na osnovu koncepata koji postoje u samom biznis domenu, a ne na osnovu tehničkih koncepata. Treći korak u definisanju arhitekture aplikacije je određivanje i definisanje API -a svakog od identifikovanih servisa. To se postiže tako što se prethodno identifikovane sistemske operacije u prvom koraku dodeljuju identifikovanim servisima u okviru drugog koraka. Neke od operacija servisi mogu da izvrše samostalno, međutim neke

kompleksnije operacije zahtevaju koordinaciju nekada i više od dva servisa.

5. INTERPROCESNA KOMUNIKACIJA

Zbog dekomponovanja monolitnog sistema, u fizički odvojene servise, komunikacija na nivou jezičkih konstrukcija nije više dostupna. Često servisi nisu ni napisani korišćenjem istih programskih jezika niti softverskih paradigmi tako da je za komunikaciju među ovim servisima potreban drugi mehanizam. Ovakvi mahanizmi se nazivaju Interprocesna komunikacija (eng. Interprocess com- munication) IPC . Postoji više tipova IPC-a, a glavna podela se može izvršiti na osnovu prirode same komunikacije [2] na:

- Sinhronu (komunikaciju zasnovanu na zahtevima i odgovorima)
- Asinhronu (komunikaciju zasnovanu na porukama)

IPC je, kao i druge odluke vezane za arhitekturu sistema, veoma kompleksna i zahtevaju dobru analizu konkretnog problema. Izbor načina komunikacije odrediće ponašanje sistema, njegovu dostupnost kao per- formanse sistema. Postoje različita tehnološka rešenja i konkretizacije za sinhroni i asinhroni tip komunikacije, koja se najčešće razlikuju u protokolima i formatima poruka koje se razmenjuju. Postoje protokoli koji su ljudski čitljivi, a postoje i oni kod kojih se podaci prenose u binarnom formatu. Koji god od mehanizama da se odabere, jako je bitno prvo definisati API servisa koristeci jezike za definisanje interfejsa nezavisno od konkretnih implementacija (eng. interface definition language) IDL. Način na koji se API specificira, zavisi od toga koji IPC mehanizam odaberemo. Ukoliko je u pitanju komunikacija zasnovana na porukama, API definicija se sastoji od specifikacije kanala, tipa poruke, i formata poruke. Ukoliko se koristi HTTP protokol, API definicija se sastoji od specifikacije URL-a, naziva HTTP metode, formata zahteva i formata odgovora.

6. TRANSAKCIONA LOGIKA

Transakcije su esencijalni činilac svake poslovne aplikacije. Bez transakcija ne bi bilo moguće garan- tovati konzistentnost podataka. Transakcija predstavlja nedeljivi skup operacija koje je na nivou softverskog sistema potrebno izvršiti kako bi se jedan korisnički zahtev uspešno obradio. Kod sistema zasnovanih na monolitnoj arhitekturi koji koriste jednu instancu baze podataka za rad sa transakcijama koriste se postojeći mehanizmi koje najčešće poseduje sama baza podataka.

Problem nastaje kada je za obradu zahteva potrebno orkestrirano ponašanje više od jednog servisa u mikroservisnoj arhitekturi.

Može se desiti da jedan servis sačuva podatke u svoju bazu, dok sledeći servis iz nekog razloga nije u mogućnosti da uspešno sačuva podatke. Na taj način konzistencija podataka sistema je narušena jer imamo parcijalno sačuvano stanje. Postoje dva načina da se problem transakcija u mikroservisnim sistemima reše:

- Distribuirane transakcije
- Saga šablon (eng. Saga pattern)

Distribuirane transakcije se najčešće implementiraju korišćenjem standarda (eng. X/Open Distributed Transaction Processing (DTP) Model) (X/Open XA). X/Open XA koristi algoritam dvostruke potvrde (eng. two-phase commit), da bi osigurao da su svi učesnici u transakciji potvrdili ili poništili izmene. Da bi se koristio mehanizam distribuiranih transak- cija u sistemu, sve komponente koje se koriste moraju biti implementirane na način da podržavaju X/Open XA standard.

Saga pattern [3] se koristi kako bi se održala konzistentnost podataka u mikroservisnoj arhitekturi, bez ko- rišćenja prethodno pomenutih distibuiranih transakcija. Prilikom korišćenja ovog paterna, bitno je da se definise nova saga za svaku komandu u sistemu koja mora da izmeni podatke u više od jednog servisa. Saga predstavlja niz lokalnih transakcija. Svaka od lokalnih transakcija na nivou jednog servisa garantuje atomičnost izvršavanja ACID, odnosno nedeljivost operacije. Svaki korak u okviru izvršavanja jedne sage, nakon završetka izvršavanja poziva prosleđivanjem odgovarajuće asinhrone poruke sledeći korak. Za razmenu poruka koriste se asinhroni mehanizmi, pa je tako mala verovatnoća da će neka poruka biti izgubljena i neobrađena. Međutim kod saga paterna postoji problem vraćanja na prethodno stanje ukoliko jedan od koraka sage ne bude validan i ne može da se izvrši sa pozitivnim ishodom. Dok je kod sistema koji imaju ACID transakcije vraćanje na prethodno stanje (eng. roll-back) dostupan na nivou samog (eng. framework)-a, kod sage se za to mora kreirati kompenzujuća operacija. Dva osnovna tipa saga sablona su distribuirana koreografija [4] i orkestrator [5].

7. APLIKACIJA ZA PERSONALNE TRENERE

Cilj ovog rada je razvoj rešenja koje će biti korišćeno od strane registrovanih klijenata (ličnih trenera) kao i od strane korisnika usluga konkretnih trenera. Prilikom izrade ovog aplikativnog rešenja, u obzir su uzete sve prethodno navedene metodologije modelovanja softverske arhitekture. Korišćena arhitekture serverskog dela sistema je mikroservisna anrhitektura.

7.1. Korišćene tehnologije

Aplikativno rešenje sačinjeno je od infrastrukturnog, servisnog i dela za interakciju sa korisnicima. Fokus ovog rada je na infrastrukturnom i servisnom delu. Za opipsivanje i definisanje infrastrukture korišćen je (eng. Terraform) programski alat. Za razvoj samog aplikativnog rešenja korišćen je programski jezik C# i .NET7 pragramska platforma. Za čuvanje podataka korišćena su dva tipa prostora za skladištenje, zavisno od potrebe. Za čuvanje struktuiranih podataka, korišćena je SQL baza, dok je za čuvanje fajlova, i drugih nestruktuiranih informacija korišćen (eng. Azure Blob Storage). Za razmenu poruka u komunikaciji servisa korišćen je (eng. Azure Service Bus) koji omogućava veoma pouzdan način razmene poruka, i podržava oba modela razmene poruka, (eng. one to one) i (eng. one to many). Za skrivanje arhitekture sistema od klijentskih aplikacija korišćen je (eng. API management) [6]. Razvojna okruženja kokja su korišćena su:

- Visual Studio Professional 2022
- Visual Studio Code
- SQL Management Studio
- Azure Data Explorer

7.2. Opis rešenja problema

Polazna tačka prilikom rešavanja ovog problema bila je modelovanje zahteva odnosno njihovo formalno prikupljanje. Ovom prilikom identifikovane su dve osnovne, kao i jedna administrativna ulogs korisnika u sistemu. Za svaku od navedenih uloga, identifikovan je konačan skup operacija koje mogu izvoditi u sistemu.
Identifikovane uloge u sistemu:

- Administrator
- Klijent
- Trener

Za svaku od navedenih uloga identifikovane su operacija u sistemu koje konkretna uloga može da izvrši. Nakon što smo pažljivom analizom došli do skupa operacija, obavljeno je grupisanje operacija na osnovu njihove koherentnosti u domenskom smislu. Identifikovani su sledeći servisi:

- (eng. User Management Service)
- (eng. Membership Service)
- (eng. Payment Service)
- (eng. Feed Service)
- (eng. Training Data Service)
- (eng. Training Material Service)
- (eng. Slot Service)
- (eng. Booking Service)
- (eng. Messages Service)
- (eng. Notification Service)

Zavisno od problema koji svaki od ovih servisa rešava definisane su strukture, kao i način čuvanja podataka. Neki od servisa podatke čuvaju u Azure blob storage-u, u formatu manje striktnosti. Dok neki od servisa koriste SQL baze podataka.

8. ZAKLJUČAK

Odabirom mikroservisne arhitekture, veoma je povećana kompleksnost sistema. Sam razvoj je bio u velikoj meri otežan, samom količinom aplikacija, njihovom konfiguracijom, procesom deploy-ovanja kao i povezivanjem komunikacije među servisima. Za veličinu tima koja je mala, definitivno bi trebalo dobro razmisliti da li je inicijalno mikroservisna arhitektura prava opcija. Vreme razvoja bi bilo značajno kraće kada bi ova aplikacija bila razvijena u monolitnoj arhitekturi i kada bi se koristila centralizovana baza podataka. Takođe treba napomenuti da je cena izvršavanja ovakvog sistema na Cloud-u značajno veća od monolitnog sistema sa istim skupom funkcionalnosti.

Količina App Service-a, SQL baza, Blob Storage-a u značajnoj meri doprinose tome da cena ovakvog sistema bude visoka. Ukoliko bi tim bio veći, to bi dodatno povećalo potrebu za postojanjem procesa razvoja, a samim tim i troškova kompanije koja bi takav sistem razvijala.
Zaključeno je da pored toga što mikroservisna arhitektura ima ogroman skup prednosti koje su nabrojane u prethodnom delu rada, poseduje i određene probleme koji nisu toliko tehničke prirode, već se tiču finansijskih mogućnosti naručioca sistema, veličine tima, ali i stepena zrelosti proizvoda.

9. LITERATURA

[1] https://www.synopsys.com/glossary/ what-is-software-architecture.html (pristupljeno u decembru 2023.)
[2] https://learn.microsoft.com/en- us/azure/ architecture/guide/technology-choices/messaging (pristupljeno u decembru 2023.)
[3] https://learn.microsoft.com/en-us/azure/ architecture/reference-architectures/saga/saga (pristupljeno u decembru 2023.)
[4] https://learn.microsoft.com/en-us/azure/architecture/ patterns/choreography (pristupljeno u decembru 2023.)
[5] https://learn.microsoft.com/en- us/azure/architecture/ reference-architectures/saga/saga#orchestration (pristupljeno u decembru 2023.)
[6] https://learn.microsoft.com/en-us/azure/api-management/api-management-key-concepts (pristupljeno u decembru 2023.)

Kratka biografija:

Petar Stošić rođen je u Nišu 1993. godine. Osnovne akademske studije na Fakultetu tehničkih nauka Univerziteta u Novom Sadu upisao je 2012. godine. Diplomirao je 2016. godine i iste godine upisuje master akademske studije.

Kontakt: petar.stosic993@gmail.com