Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 4.9
DOI: https://doi.org/10.24867/30BE09Krsmanovic

RAZVOJ APLIKACIJE KORIŠĆENJEM MODULAR MONOLIT ARHITEKTURE I EKSTRAKCIJA MIKROSERVISA IZ MODULA

DEVELOPMENT OF APPLICATION USING MODULAR MONOLITH ARCHITECTURE AND EXTRACTION OF MICROSERVICE FROM MODULE

Đorđe Krsmanović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIČKO I RAČUNARSKO INŽENJERSTVO

Kratak sadržaj – *U okviru rada kreirana je specifikacija imaginarnog softverskog sistema, koji je razvijen upotrebom modular monolit arhitekture. Sistem je dovoljno kompleksan, kako bi se simulirao rad na realnom projektu. Zatim je izvršena ekstrakcija mikroservisa iz modula, čime se prelazi na mikroservisnu arhitekturu. Cilj rada je istražiti koliko je izazovno izvršiti tranziciju sa modular monolitne na mikorservisnu arhitekturu. U radu su predstavljene obje arhitekture sa akcentom na načine komunikacije i rad sa podacima. Opisan je proces rada i koraci koje je potrebno uraditi kako bi se softverski sistem implementirao. Zatim su putem C4 dijagrama predstavljeni rezultati rada na aplikaciji i opisane izmjene do kojih dolazi prilikom tranzicije sa modularne na mikroservisnu arhitekturu. Na samom kraju date su prednosti i mane obje arhitekture, kao i smjernice za poboljšanje rada i dalje istraživanje.*

Ključne reči: *arhitektura softvera, modularni monolit, mikroservisna arhitektura,*

Abstract – *As part of this paper, a specification for an imaginary software system is created, which was developed using a modular monolith architecture. The system is sufficiently complex to simulate work on a real-world project. Subsequently, microservice was extracted from the module, transitioning the system to a microservice architecture. The goal of the thesis is to explore the challenges involved in transitioning from a modular monolith to a microservice architecture. The paper presents both architectures, focusing on communication methods and data handling. The development process is described, outlining the necessary steps for implementing the software system. The results of the work are illustrated using C4 diagrams, showcasing the changes that occur during the transition from a modular to a microservice architecture. Finally, the advantages and disadvantages of both architectures are discussed, along with recommendations for improving the process and suggestions for further research.*

Keywords: *software architecture, modular monolith, microservices architecture*

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Nikola Luburić, docent.

1. UVOD

Arhitektura softvera je od suštinske važnosti u razvoju softvera, jer određuje osnovu za strukturu, funkcionalnost, skalabilnost i održivost sistema. Veliki broj inženjera se odlučuje za upotrebu mikroservisne arhitekture, jer određene komponente sistema koje koristi veći broj korisnika lako možemo skalirati i postići dobre performanse sistema. Međutim, mikroservisna arhitektura je sama po sebi kompleksna i troši više resursa na razvoj i testiranje softvera. Najbolji pristup bio bi kreirati softver na takav način da, kada uvidimo potrebu za skaliranjem određenog dijela aplikacije, imamo mogućnost da od njega relativno lako napravimo mikroservis. Modular monolit arhitektura upravo to omogućava.

Ovaj rad bavi se razvojem softvera i izborom pravilne arhitekture. Na početku je kreirana specifikacija jednog softverskog sistema. Zatim je taj softverski sistem implementiran upotrebom modular monolit arhitekture. Cilj rada je da se utvrdi koliko je izazovno od jednog modula aplikacije napraviti mikroservis.

U drugom poglavlju biće opisane teorijske osnove modular monolit i mikroservisne arhitekture. Naredno poglavlje baviće se opisom procesa rada na softverskom sistemu, gdje će biti navedene sprovedene aktivnosti tokom razvoja softvera. U četvrtom poglavlju biće opisani rezultati rada, uz dijagrame aplikacija, te će biti utvrđene razlike u arhitekturi sistema. Naredno poglavlje opisuje iskustvo razvoja softvera s obje arhitekture, uključujući njihove prednosti i mane. Poslednje poglavlje donosi zaključak sa sažetkom istraživanja i prijedlozima za buduća unapređenja.

2. TEORIJSKE OSNOVE

U ovom poglavlju biće razmatrane teorijske osnove modularne monolitne i mikroservisne arhitekture, sa posebnim akcentom na aspekte komunikacije i izolacije podataka.

2.1. Arhitektura modularnog monolita

Modular monolit arhitektura kombinuje paradigmu modularnog razvoja softvera i monolitne arhitekture. Modularni monolit je arhitektonski obrazac koji strukturira aplikaciju u nezavisne module ili komponente sa dobro definisanim granicama. Moduli su podijeljeni na osnovu logičkih granica, grupišući zajedno povezane funkcionalnosti, te svaki ima dobro definisan interfejs za komunikaciju. Iako moduli mogu biti organizovani u odvojene jedinice, oni su čvrsto integrisani i ne mogu

potpuno funkcionisati bez međusobne saradnje. Ova arhitektura koristi zajedničku bazu koda i skladište podataka, što pojednostavljuje razvoj i isporuku, ali istovremeno omogućava dobru organizaciju koda. Fleksibilnost je još jedna prednost, jer omogućava dodavanje i brisanje modula bez velikog uticaja na cijeli sistem, a preopterećeni moduli mogu se izdvojiti u mikroservise i horizontalno skalirati. Isporuka softvera je jednostavnija zbog postojanja samo jednog artefakta.

2.1.1. Komunikacija između modula

Jedno od osnovih pravila modularne arhitekture jeste da moduli ne mogu da referenciraju tuđe funkcionalnosti direktno, već samo mogu da im pristupe putem javnog API drugog modula.

Najjednostavniji način komunikacije je sinhrona komunikacija, sa direktnim pozivima metoda između modula. Mana ovog pristupa je čvrsta spregnutost između modula, ukoliko jedan modul nije dostupan to će uticati i na dostupnost drugog modula.

Sljedeći način komunikacije je asinhrona komunikacija putem poruka. Asinhrona komunikacija smanjuje spregnutost između modula, ali se povećava kompleksnost, jer se uvodi sistem za razmjenu poruka.

2.1.2. Izolacija podataka

Izolacija podataka između modula je ključna za održavanje njihove nezavisnosti i slabe povezanosti, pri čemu svaki modul može pristupiti samo svojim tabelama u bazi podataka i nije dozvoljeno da vrši upite nad podacima drugih modula. Postoji nekoliko tehnika za izolaciju podataka:

- odvojene tabele – sve tabele se nalaze u istoj bazi podataka, što može biti jednostavno u ranim fazama razvoja
- odvojene šeme – tabele se grupišu i logički izoluju pomoću šema u bazi, gdje svaki modul ima sopstvenu šemu
- odvojene baze – svaki modul ima svoju bazu podataka, što omogućava veću izolaciju ali dodaje kompleksnost u održavanju infrastrukture
- drugačija vrsta skladišta - koriste se različiti tipovi skladišta, poput graf baza ili drugih nerelacionih baza, za specifične poslovne potrebe modula.

2.2. Mikroservisna arhitektura

Mikroservisna arhitektura je pristup razvoju jedne aplikacije kao skupa malih usluga, gdje se svaka pokreće u svom procesu i komunicira jednostavnim mehanizmima, često pomoću HTTP-a [1]. Svaki servis implementira zaseban poslovni zahtjev i može se nezavisno pustiti u rad potpuno automatizovano. Postoji minimum centralizovanog upravljanja ovim uslugama, koje mogu biti napisane na različitim programskim jezicima i koristiti različite tehnologije za skladištenje podataka [2].

Može se reći da u mikroservisnoj arhitekturi aplikacija je sačinjena od velikog broja slabo spregnutih i nezavisno isporučenih komponenti. Mikroservisi se organizuju prema biznis logici koju trebaju da rješavaju. Pojedinačni servisi obavljaju jedan zadatak koji predstavlja jednu poslovnu funkciju cjelokupnog sistema. Prilikom razvoja mikroservisa svaki mikroservis može da ima različite tehnologije uključujući izbor baze i programskog jezika.

2.2.1. Komunikacija između mikroservisa

Servisi moraju sarađivati da bi ispunili korisničke zahtjeve. Postoje dva osnovna tipa komunikacije:

- Sinhrona komunikacija – klijent šalje zahtjev mikroservisu i čeka odgovor. Najčešće se koristi REST [3] ili gRPC [4] protokol. REST koristi HTTP glagole i URL-ove za manipulaciju resursima, dok gRPC omogućava pozivanje metoda na udaljenim serverima koristeći binarne protokole, čineći ga efikasnijim, ali zahtijeva HTTP/2.
- Asinhrona komunikacija – servisi komuniciraju slanjem poruka putem posrednika (message broker), npr. RabbitMQ [5], ActiveMQ [6], Apache Kafka [7]. Posrednik baferuje poruke i šalje ih kada primalac bude spreman. Izbor posrednika zavisi od zahtjeva aplikacije, pri čemu se balansira između latencije i pouzdanosti.

2.2.2. Rad sa podacima u mikroservisima

Većina servisa ima potrebu da sačuva svoje podatke u nekoj vrsti baze podataka. Postoje dva različita pristupa:

- Dijeljena baza podataka – koristi se jedinstvena baza podataka za više servisa. Svaki servis slobodno pristupa svojim podacima i podacima drugih servisa koristeći lokalne ACID transakcije.
- Posebna baza za svaki servis – svaki servis ima svoju bazu podataka. Podaci su privatni za servis. Transakcije koje radi servis odnose se samo na njegovu sopstvenu bazu. Podaci koji njemu ne pripadaju, dostupni su isključivo putem API-ja drugih servisa

Pristup posebna baza po servisu se najčešće koristi prilikom razvoja aplikacija korišćenjem mikroservisne arhitekture. On omogućava da servisi budu labavo spregnuti, te promjena u bazi podataka jednog servisa ne utiče na druge servise. Svaki servis može da koristi tip baze podataka koji najviše odgovara njegovim potrebama.

3. METODOLOGIJA

Metodologija rada uključuje razvoj imaginarnog sistema za zakazivanje smještaja kako bi se simulirao rad na realnom sistemu i isprobali različiti arhitekturalni pristupi. Projekat je obuhvatio specifikaciju dizajna, implementaciju modularnog monolita i ekstrakciju mikroservisa.

Specifikacija dizajna uključuje klasne i sekvencijalne dijagrame koji prikazuju statičku i dinamičku strukturu sistema. Prilikom izrade klasnog dijagrama izvršena je podijela aplikacije na tri modula:

- Accommodation
- Commerce
- User Access.

Implementacija modularnog monolita je rađena u .NET [8] i Angular [9] radnim okvirima, uz korišćenje Domain Driven Design-a [10] i čiste arhitekture [11].

Ekstrakcija mikroservisa je počela sa Accommodation modulom, jer se očekivalo da će korisnici najviše koristiti funkcionalnosti ovog module, te bi ga trebalo skalirati. Prilikom ekstrakcije, modularni monolit je podijeljen tako da je svaki mikroservis funkcionisao kao zasebna jedinica, sa sopstvenim repozitorijumom koda. Organizacija koda u više repozitorijuma je bila ključna jer je omogućila nezavisan razvoj mikroservisa bez uticaja na druge dijelove aplikacije. Zajednički dijelovi koda, takođe su ekstrahovani u zasebne repozitorijume kako bi mogli biti referencirani od više mikroservisa. Dodat je API Gateway za rutiranje, autentifikaciju i autorizaciju i predstavlja ulaznu tačku u sistem.

4. REZULTATI RADA

U ovom poglavlju biće predstavljeni rezultati rada na projektima. Fokus će biti na C4 [12] modelu dijagrama koji su nacrtani kako bi se što bolje opisala arhitektura aplikacija. C4 dijagrami su metoda za modelovanje softverske arhitekture koja koristi hijerarhijski pristup kako bi se prikazali različiti nivoi detalja sistema.

U potpoglavlju 4.1 biće prikazan i opisan dijagram komponenti koji prikazuje razvoj aplikacije koristeći modularni monolit. Naredno potpoglavlje će prikazati samo promjene do kojih dođe kada se izvrši estrakcija mikroservisa iz modula.

4.1. Modular monolit arhitektura

Dijagram komponenti na slici 4.1 pruža uvid u unutrašnju strukturu sistema. Glavna komponente sistema su klijentska aplikacija, serverska aplikacija, baza podataka i komponenta za razmjenu poruka.

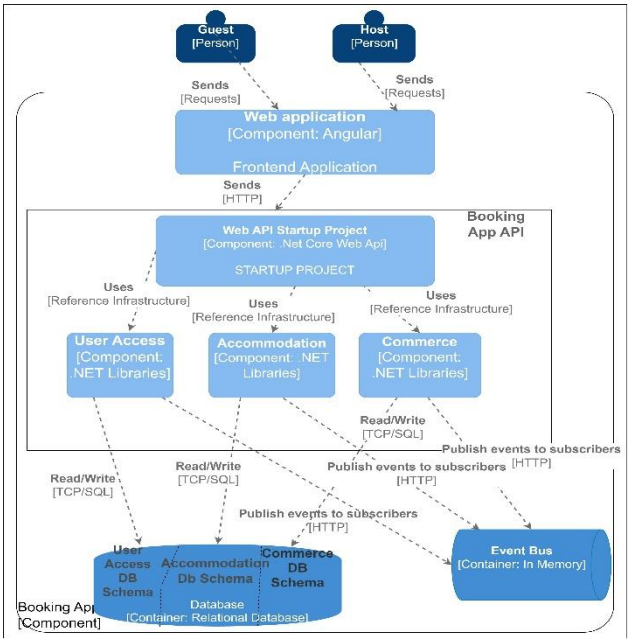
Svaki modul sastoji se od pet .NET klasnih biblioteka, koje predstavljaju prezentacioni, aplikativni, domenski i infrastrukturni sloj, dok peta sadrži tipove poruka koji se razmjenjuju između modula. Web API projekat je centralni dio koji pokreće aplikaciju i konfiguriše module.

Modularna aplikacija koristi jednu relacionu bazu podataka, gdje svaki modul ima sopstvenu šemu baze podataka. Moduli komuniciraju razmjenom poruka, što smanjuje spregnutost i olakšava prelazak na mikroservisnu arhitekturu. U modular monolit arhitekturi se koristi in-memory transport za brzu razmjenu poruka unutar jednog procesa, poboljšavajući performanse bez potrebe za dodatnom infrastrukturom u vidu korišćenja posrednika poruka.

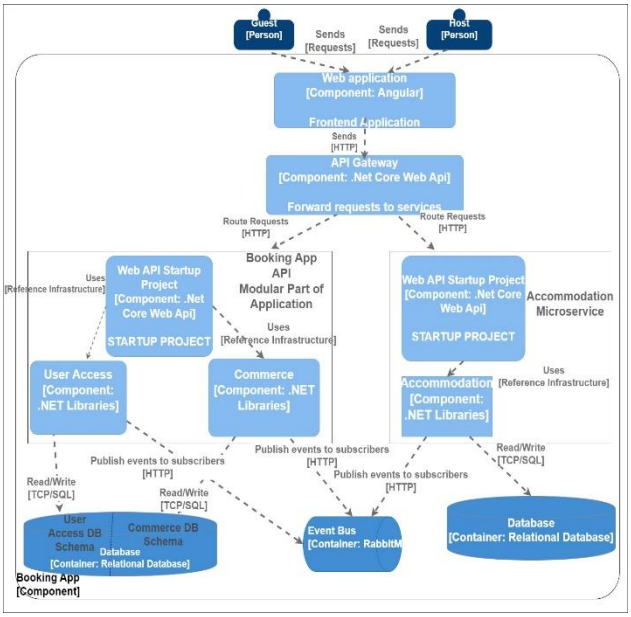
4.2. Mikroservisna arhitektura

Pogledom na dijagram komponenti na slici 4.2 mogu se vidjeti prve promjene koje su posljedica izmjene arhitekture. Samim činom ekstrakcije mikroservisa iz modula, prelazi se na mikroservisnu arhitekturu, koja zahtijeva da servis ima sopstveno skladište. Kreirana je nova baza podataka kojoj će pristup imati ekstrahovani mikroservis. Ostatak modularnog dijela aplikacije u ovom slučaju čini drugi mikroservis. Ovaj mikroservis koristi bazu podataka koja sadrži dvije šeme, gdje svakoj šemi pristupa određeni modul. Takođe dodat je novi Web Api projekat koji ima ulogu API Gateway-a.

Sa prelaskom na mikroservisnu arhitekturu, potrebno je zamijeniti in-memory mehanizam za razmjenu poruka distribuiranim rješenjem koje podržava mrežnu komunikaciju, jer mikroservisi mogu biti na različitim lokacijama. Iz tog razloga, izabran je RabbitMQ kao novi posrednik poruka.



Slika 4.1. *Dijagram komponenti sistema, modular monolit arhitektura*



Slika 4.2. *Dijagram komponenti sistema, mikroservisna arhitektura*

5. DISKUSIJA

U ovom poglavlju biće predstvljene prednosti i mane implementiranih arhitektura. Modularni monolitni pristup nije pogodan za jednostavne aplikacije bez dovoljno biznis logike, ali je odličan za složene aplikacije, jer omogućava dobru organizaciju koda i lako upravljanje složenošću. Moduli su grupisani prema skupu zajedničkih funkcionalnosti, što olakšava migraciju na mikroservise ako su ispraćene dobre prakse razvoja aplikacije. Nedostaci ovog pristupa su nemogućnost Join operacija

između podataka koji pripadaju različitim modulima. Zbog toga većina logike za agregaciju podataka treba da bude na klijentskoj strani.

Mikroservisna arhitektura je idealna za aplikacije koje zahtijevaju skalabilnost i timsku podjelu rada, uz fleksibilnost u izboru tehnologija. Mane uključuju složenost infrastrukture, testiranja, verzionisanja i upravljanja zajedničkim kodom, kao i izazove sa održavanjem konzistentnosti sistema.

6. ZAKLJUČAK

U radu su predstavljeni modularni monolit i mikroservisna arhitektura, pri čemu su razmatrane prednosti i mane obje arhitekture. Implementiran je projekat koristeći modularni monolit kao arhitektonski obrazac, nakon čega je izvršena ekstrakcija mikroservisa iz modula. Motivacija za pisanje rada bila je sprovođenje eksperimenta radi provjere koliko je izazovno preći sa monolitne arhitekture na mikroservisnu.

U današnje vrijeme često imamo monolitne aplikacije, koje je potrebno prebaciti u mikroservise radi što boljeg skaliranja i lakšeg održavanja. Na osnovu rezultata, ukoliko imamo monolitnu aplikaciju, najbolje bi bilo prvo preći na modularni monolit sa jasno definisanim modulima. Ovo bi predstavljalo jednu vrstu međukoraka, jer se suštinski aplikacija sa stanovišta arhitekture ne mijenja značajno. Potrebno je izvršiti refaktorisanje koda i formiranje modula. Nakon toga, kao što je prikazano u radu, tranzicija na mikroservise može se relativno lako izvesti.

Kao unapređenje ovog rada, predlože se analiza oba sistema sa DevOps strane. Dobro bi bilo automatizovati isporuku i testiranje modularne aplikacije i vidjeti do kojih sve promjena dolazi kada se izvrši tranzicija na mikroservisnu arhitekturu. Analizirajući ovaj aspekt razvoja i održavanja sistema, dobila bi se cjelokupnu slika koliki je zaista napor potrebno uložiti da se izmijeni arhitektura aplikacije.

7. LITERATURA

[1] Zvanična dokumentacija HTTP protokola, https://httpwg.org/specs/ [datum pristupa jul 2024]

[2] Microservices article, https://martinfowler.com/articles/microservices.html [datum pristupa jul 2024]

[3] Richardson, Leonard; Ruby, Sam (2007). RESTful Web Services. Sebastopol, California: O'Reilly Media

[4] Zvanična dokumentacija gRPC protokola, https://grpc.io/docs/ [datum pristupa jul 2024]

[5] Zvanična dokumentacija RabbitMQ, https://www.rabbitmq.com/docs [datum pristupa jul 2024]

[6] Zvanična dokumentacija ActiveMQ, https://activemq.apache.org/ [datum pristupa jul 2024]

[7] Zvanična dokumentacija Apache Kafka, https://kafka.apache.org/documentation/, [datum pristupa jul 2024]

[8] Zvanična dokumentacija .NET radnog okvira, https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet [datum pristupa jul 2024]

[9] Zvanična dokumentacija Angular radnog okvira, https://v17.angular.io/docs [datum pristupa jul 2024]

[10] Eric Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software

[11] Robert C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design

[12] Zvanična dokumentacija C4 modela dijagrama, https://c4model.com/ [datum pristupa avgust 2024]

Kratka biografija:

Đorđe Krsmanović rođen je 14. januara 1999. godine u Foči. Osnovno obrazovanje je stekao u osnovnoj školi „Sveti Sava“ u Foči. Nakon toga upisuje gimnaziju – prirodno matematički smjer u Srednjoškolskom centru u Foči. Školske 2018/2019 godine upisuje Fakultet tehničkih nauka u Novom Sadu, odsjek Elektrotehnika i računarstvo, smjer Računarstvo i automatika. Po završetku studija u roku, upisuje master akademske studije koje završava 2024. godine.