# Posrednici poruka

**Šubarić, Jan**

SVEUČILIŠTE U RIJECI

**TEHNIČKI FAKULTET**

Preddiplomski sveučilišni studij računarstva

Završni rad

# POSREDNICI PORUKA

Rijeka, rujan 2022.

Jan Šubarić
0069087527

SVEUČILIŠTE U RIJECI

**TEHNIČKI FAKULTET**

Preddiplomski sveučilišni studij računarstva

Završni rad

# POSREDNICI PORUKA

Mentor: Prof. dr. sc. Mladen Tomić

Rijeka, rujan 2022.

Jan Šubarić
0069087527

**SVEUČILIŠTE U RIJECI**
**TEHNIČKI FAKULTET**
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 13. srpnja 2022.

Zavod: **Zavod za računarstvo**
Predmet: **Računalne mreže**
Grana: **2.09.02 informacijski sustavi**

# ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Jan Šubarić (0069087527)**
Studij: Preddiplomski sveučilišni studij računarstva

Zadatak: **Posrednici poruka / Message Brokers**

Opis zadatka:

Proučiti primjene posrednika poruka (message brokers). Opisati motivaciju za uvođenjem posrednika te probleme koji se na taj način rješavaju. Usporediti prednosti i nedostatke implementacije sustava s i bez posrednika. Izložiti nekoliko konkretnih primjera primjene. Napraviti pregled aktualnih posrednika poruka otvorenog koda. Implementirati sustav koji će koristiti odabrani posrednik otvorenog koda za proslijeđivanje poruka.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Jan Šubarić

Zadatak uručen pristupniku: 13. srpnja 2022.

Mentor:

Izv. prof. dr. sc. Mladen Tomić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Kristijan Lenac

# IZJAVA

o samostalnoj izradi **Završnog rada**

Izjavljujem pod punom materijalnom i moralnom odgovornošću da sam ovaj rad izradio samostalno te da u njemu nema kopiranih ili prepisanih dijelova teksta tuđih radova, a da nisu propisano označeni kao citati s navedenim izvorom iz kojeg su preneseni.

U Rijeci, rujan 2022.

Jan Šuharić

(vlastoručni potpis)

# Zahvala

*Zahvaljujem se svom mentoru prof.dr.sc. Mladenu Tomiću na davanju savjeta pri odabiru teme za ovaj završni rad te na iskazanom trudu i strpljenu prilikom pisanja rada.*

*Zahvaljujem se svim profesorima koji su mi u ove tri godine studijskog programa pružili potporu i proširili dotadašnje znanje na području tehničkih znanosti.*

*Posebnu zahvalu iskazujem svojoj obitelji i prijateljima koji su cijelo vrijeme bili uz mene, poticali me i olakšavali ovaj životni put.*

**SADRŽAJ**

# 1. UVOD

## 1.1. Predmet i cilj rada

Ovaj će se rad temeljiti na detaljnom opisu načina primjene i slučaja uporabe posrednika poruka (engl. *Message broker*). Rad će sadržavati opise problema koji se rješavaju uvođenjem posrednika poruka te će se usporediti prednosti i nedostaci implementacije sustava s i bez posrednika. U radu će biti naveden pregled aktualnih posrednika poruka otvorenog koda te njihove glavne karakteristike.

Kao glavni cilj praktičnog dijela rada, biti će prezentiran rad posrednika poruka kroz implementaciju sustava koji koristi odabrani posrednik poruka otvorenog koda. Detaljno će biti opisan proces povezivanja pojedini usluga s posrednikom poruka, njihova međusobna komunikacija unutar sustava i uloga posrednika poruka.

## 1.2. Metode prikupljanja i izvori podataka

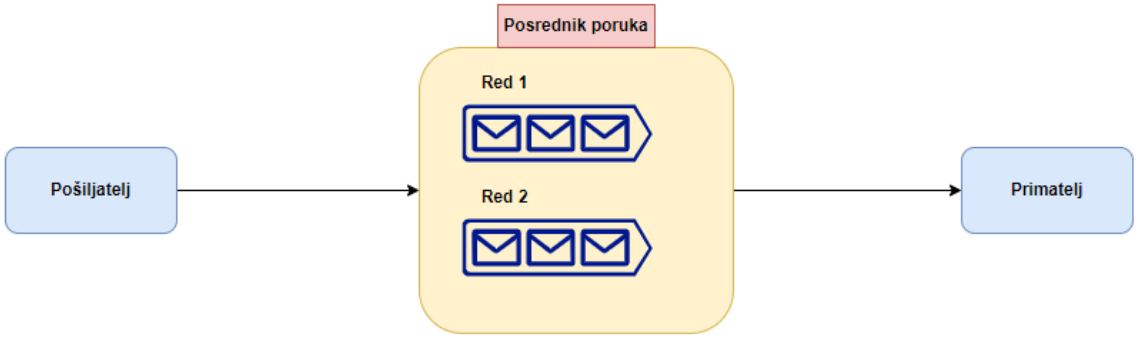
S obzirom da je posrednik poruka novija tehnologija koja se koristi u svijetu računarstva i mnogima predstavlja apstraktan pojam, većina sadržaja koja je korištena za pisanje završnog rada, pronađena je na Internetu, odnosno različitim web stranicama. Zbog manjka literature o navedenoj temi na hrvatskom jeziku, većinom su korišteni izvori na engleskom jeziku. Također, za razumijevanje rada RabbitMQ posrednika poruka koji je korišten u praktičnom dijelu rada, korištena je službena stranica RabbitMQ posrednika poruka koja sadrži detaljnu dokumentaciju za rukovanje navedenim posrednikom poruka. Kao dodatna pomoć prilikom izrade praktičnog dijela rada, korištene su i informacije iz pojedinih video prezentacija.

Nakon što se prikupio i proučio značajan broj kvalitetnih izvora podataka i informacija o zadanoj temi rada, izrada završnog rada je započela s implementacijom zamišljenog sustava koji će koristiti posrednik poruka. Poslije uspješnog implementiranja sustava, započeto je pisanje završnog rada.

# 2. OPĆENITO O POSREDNICIMA PORUKA

## 2.1. Definicija posrednika poruka

Posrednik poruka (engl. *Message broker*) [1] je program koji raznim uslugama, aplikacijama i sustavima omogućuje međusobnu komunikaciju i razmjenu informacija. Posrednik poruka funkcionira na način da prenosi i prevodi poruke između osnovnih komunikacijskih protokola koje koriste različiti sustavi. Takav način rada posrednika omogućuje komunikaciju između neovisnih aplikacija i usluga koje mogu međusobno vršiti komunikaciju bez obzira u kojem su programskom jeziku implementirane i na kojem sučelju se nalaze. Posrednik poruka može se promatrati kao "međuprogram" (engl. *Middleware*), odnosno samostalno komunikacijsko sučelje orijentirano na prijenos poruka između pošiljatelja i primatelja. To je interni medij unutar kojeg se pohranjuju podaci koje šalje pošiljatelj, a podaci ostaju pohranjeni u posredniku sve dok se ne izruče primatelju. Korištenjem posrednika poruka koji će se pobrinuti za prijenos informacija, aplikacija može neometano nastaviti s radom i izvršavati ostale zadatke bez potrebe za čekanjem zahvaljujući asinkronom načinu komunikacije.




*Slika 2.1. Pošiljatelj i primatelj s posrednikom poruka*

## 2.2. Razlozi uvođenja posrednika poruka

### 2.2.1. Standardna uspostava komunikacije

Standardno uspostavljanje komunikacije između dva neovisna sustava uključuje, prije svega, definiranje sučelja preko kojeg će sustavi moći vršiti međusobnu komunikaciju. Prilikom definiranja sučelja preko kojeg će se odvijati komunikacija, potrebno je uspostaviti komunikacijski put i komunikacijski protokol za prijenos podataka kao što je HTTP, FTP ili SMTP te odrediti oblik poruke i vrstu podatka koja će se prenositi. Takav način komunikacije zahtjeva dogovor između dviju strana prilikom implementacije sučelja za prijenos poruka. Sve dok sučelje za prijenos poruka zadržava istu dogovorenu strukturu, sustavi mogu vršiti međusobnu komunikaciju. Iako su sustavi u širem smislu neovisni i dijelovi implementacije individualnog neovisnog sustava se mogu mijenjati, postoji stupanj ovisnosti koji se javlja upravo zbog dogovorene implementacije sučelja za komunikaciju. Ovakav način komunikacije zahtjeva istovremenu dostupnost oba sustava jer koristi sinkroni način komunikacije.

### 2.2.2. Rješavanje problema prilikom komunikacije koristeći posrednik poruka

Prethodno opisani standardni način uspostavljanja komunikacije zahtjeva istovremenu prisutnost oba sustava na mreži. Također, uporaba ovakvog načina komunikacije nije korisna kada:

- Drugi sustav nije pokrenut
- Internetska veza je preopterećena
- Drugi sustav je zauzet/spor
- Sustavi koriste različite komunikacijske protokole
- Pojava greške prilikom komunikacije

Upravo uvođenje posrednika poruka rješava gore navedene probleme. Posrednik poruka je komunikacijsko sučelje koje omogućuje prijenos poruka između dviju ili više zainteresiranih strana. Korištenjem posrednika poruka prilikom usmjeravanja i održavanja komunikacije između aplikacija, sustava ili servisa, postiže se visoki stupanj neovisnosti između dviju ili više strana koje vrše međusobnu komunikaciju.

### 2.3. Glavne značajke korištenja posrednika poruka

Glavni atributi za korištenje posrednika poruka su njegova korisna svojstva i prednosti koje pruža prilikom komunikacije između različitih aplikacija, usluga i sustava. Karakteristike koje omogućuje posrednik poruka unutar komunikacijske arhitekture su [2]:

**Asinkrona komunikacija**

Omogućuje asinkronu komunikaciju između sustava. Sustav na drugoj strani ne treba biti aktivan, već će primiti poruku kasnije kada bude spreman.

**Trajnost podataka**

Posrednik poruka se ponaša kao spremnik. Poruke ostaju pohranjene u posredniku sve dok ih sustav s druge strane ne bude u mogućnosti zaprimiti.

**Stabilnost i pouzdanost isporuke**

Poruke ostaju u posredniku sve dok ih strana kojoj su namijenjene ne preuzme i ne obradi.

**Veća brzina i bolje performanse sustava**

Uporaba posrednika poruka povećava brzinu i performanse sustava jer sustav ne mora čekati na odgovor druge strane i raditi na zahtjevnim procesima, već će poruka biti isporučena kada druga strana bude spremna.

**Redoslijed prilikom isporuke poruka**

Struktura posrednika poruka je red te se poruke šalju FIFO (engl. First in, First out) metodom. Prva poruka koja je poslana u red će biti prva poruka koja će izaći iz reda.

**Omogućuje nezavisnost sustava**

Sustav nema potrebe znati detalje komunikacije, uspostaviti komunikacijski put niti znati točnu lokaciju drugog sustava. Sustavi mogu biti napisani u različitim programskim jezicima, a pri tome vršiti komunikaciju koristeći posrednik poruka.

**Skalabilnost**

Broj redova (engl. *Queue*) unutar kojih se spremaju poruke može rasti i smanjivati se ovisno o potrebi. Omogućeno je slanje iste kopije poruke prema većem broju primatelja.

## 2.4. Modeli prijenosa poruka

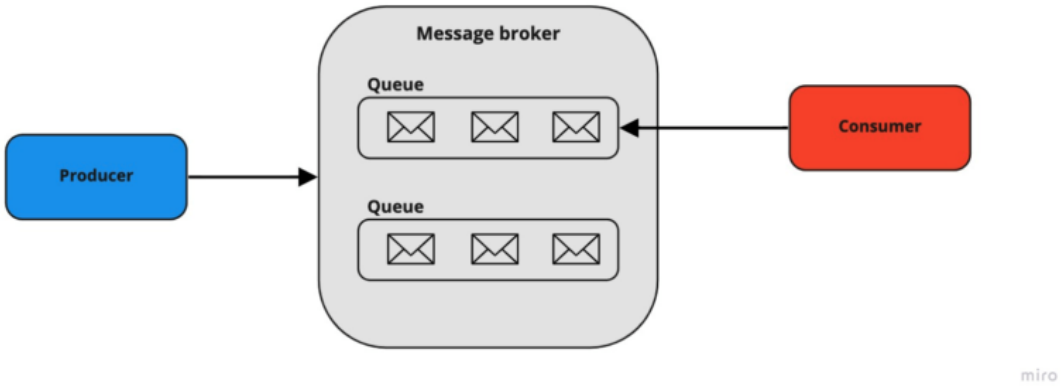
Glavna zadaća posrednika poruka je automatizacija slanja poruka između pošiljatelja i primatelja poruke. Prije samog opisa različitih modela slanja poruka koristeći posrednik poruka, opisat će se glavni dijelovi koji čine strukturu posrednika poruka:

- Pošiljatelj (engl. *Producer*) je aplikacija koja šalje poruku. Povezan je s posrednikom poruka preko veze.
- Primatelj (engl. *Consumer*) je aplikacija koja prima poruku. Također, vezom povezan s posrednikom poruka.
- Poruka (engl. *Message*) je sloj podataka koji šalje pošiljatelj, a prima je primatelj preko posrednika poruka.
- Red (engl. *Queue*) je struktura podataka koju koristi posrednik poruka prilikom komunikacije kako bi pohranio poruke.
- Izmjenjivač (engl. *Exchanger*) je logička struktura koja sadrži pravila za usmjeravanje poruka. Izmjenjivač prima poruke od pošiljatelja i na temelju određenih pravila, prosljeđuje poruku u namijenjeni red. Pri tome, red i aplikacija moraju biti povezani s izmjenjivačem.

### 2.4.1. Point-to-Point model prijenosa poruke

Point-to-Point (P2P) model je model koji služi za prijenos poruka između jednog pošiljatelja i jednog primatelja koji su povezani vezom jedan-na-jedan (engl. *One-to-One*). [4] Svaka poruka koju pošiljatelj šalje je isporučena samo jednom primatelju. Ovaj model prijenosa poruke koristan je kada se poruka mora poslati samo jedanput, bez kopija.

Često se koristi prilikom provođenja plaćanja i financijskih transakcija. Prilikom prijenosa novca, pošiljatelj obavlja slanje novca jedan jedini put, a isto tako primatelj prima samo jednu uplatu. Obje strane tada moraju biti sigurne da se transakcija izvršila samo jedan jedini put i da podaci nisu izgubljeni prilikom transakcije.

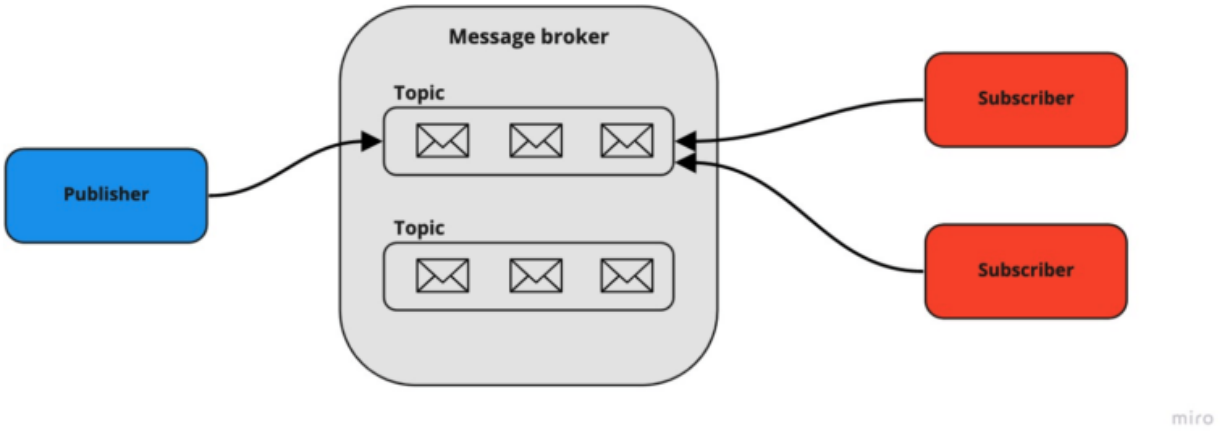



*Slika 2.3. Point-to-Point model prijenosa poruke, s Interneta, https://tsh.io/wp-content/uploads/fly-images/17919/message-broker-example-1_-800x318.png*

### 2.4.2. Publish-Subscribe model prijenosa poruke

Publish-Subscribe model za prijenos poruka, često nazvan i "pub/sub", koristi se prilikom slanja poruke koju će zaprimiti jedan ili više primatelja. [4] U ovom modelu, red ima istu ulogu, ali će se umjesto naziva reda, koristiti naziv tema (engl. Topic). Pošiljatelj objavljuje poruke ovisno o njihovom sadržaju u određeni red za tu temu . Primatelji su pretplaćeni na određene redove s temama iz kojih žele primati poruke. Sve poruke koje primatelj pošalje u red namijenjen za određeni predmet razgovora, isporučit će se primateljima koji očekuju poruke iz tog reda. Za razliku od Point-to-Point modela kod kojeg postoji samo jedan pošiljatelj i jedan primatelj, Publish-Subscribe model može uključivati više primatelja iste poruke.

Publish-Subscribe model se koristi prilikom objavljivanja poruka koje su korisne za više sudionika, a gubitak pojedine poruke nije toliko značajan. Njegova primjena je vidljiva u zračnim lukama prilikom objavljivanja kašnjenja ili slijetanja zrakoplova. Informacije će tada biti korisne mnogim djelatnicima zračne luke. Također, koristi se i prilikom čitanja podataka s raznih senzora. Primjerice, spremljene vrijednosti temperaturnih očitanja u određenom redu će moći pročitati sustavi koji su pretplaćeni na red s navedenom temom. Sustavu će najbitnija biti trenutna temperatura, a u slučaju da izgubi koje očitanje, sustav neće biti ugrožen jer će ubrzo stići novo očitanje.




Slika 2.4. Poublish-Subscribe model prijenosa poruke, s Interneta, https://tsh.io/wp-content/uploads/fly-images/17918/message-broker-example-2_-800x332.png

### 2.4.3. Hibridni model prijenosa poruke

Hibridni model se koristi rjeđe nego ostala dva modela za prijenos poruka kod posrednika poruka. Najčešće se koristi kada je posrednik poruka implementiran na poslužitelju. Hibridni model uključuje istovremenu uporabu Point-to-Point modela i Publishe/Subscribe modela.

U slučaju da više usluga treba kopiju iste poruke uz isključenu mogućnost gubitka poruke, korisno je koristiti hibridni model prijenosa. Kao i kod Publish/Subscribe modela, poruke se prosljeđuju u redove određenih tema. Svaka zainteresirana usluga za primanje određenih poruka će biti pretplaćena na određeni red. Jedino će zainteresirane strane moći preuzimati poruku i odlučivati o njenom daljnjem prijenosu na ostale zainteresirane strane.

## 2.5. Usporedba sustava s i bez posrednika poruka

### 2.5.1. Prednosti i nedostaci sustava bez posrednika poruka

Sustave bez posrednika poruka najčešće karakteriziraju sinkroni način komunikacije, brže slanje i primanje poruka te lakša konfiguracija sustava.

Prednosti korištenja sustava bez posrednika poruka su[5]:

- Korisno za prijenos manje količine podataka
- Smanjeno kašnjenje prilikom direktnog slanja poruka
- Isključuje mogućnost pogreške na posredniku poruka
- Manja složenost sustava prilikom konfiguracije
- Lakše pronalaženje i otklanjanje pogreške prilikom komunikacije

Upravo je takav oblik komunikacijske arhitekture pogodan kada klijent želi što prije doći do zatraženih informacija. Tada je, prilikom međusobne komunikacije, potrebna istovremena dostupnost s obje strane jer se prijenos dešava u realnom vremenu. Zbog toga se s vremenom i javila potreba za uvođenjem posrednika poruka koji će biti korisni za asinkroni način komunikacije među sustavima.

Nedostaci komunikacijske arhitekture bez posrednika poruka [5]:

- Obje strane u komunikaciji moraju stvoriti komunikacijski put i znati točnu lokaciju drugog sustava s kojim komuniciraju
- Mogućnost gubitka podataka u slučaju da se desi greška prilikom međusobne komunikacije
- Teža implementacija sustava za sigurnu isporuku poruke
- Zahtjeva istovremenu prisutnost oba sustava prilikom razmjene poruka

### 2.5.2. Prednosti i nedostaci sustava s posrednikom poruka

Posrednici poruka su važan dio mnogih današnjih komunikacijskih arhitektura među sustavima, uslugama i aplikacijama. Posrednici poruka omogućuju asinkronu komunikaciju između različitih sustava koji sudjeluju u komunikaciji bez potrebe za čekanjem na odgovor drugog sustava. Takav način komunikacije znatno poboljšava performanse samih sustava i povećava pouzdanost prilikom isporuke poruke.

Prednosti komunikacijske arhitekture s posrednikom poruka [6]:

- Osigurava asinkroni način komunikacije između sustava koji nisu istovremeno aktivni. Asinkrona komunikacija je korisna prilikom komunikacije u kojoj sudjeluje veliki broj sustava koji nisu uvijek dostupni prilikom prijenosa poruke.
- Poboljšava performanse sustava tako što omogućuje asinkronu komunikaciju. Sustavi mogu rješavati ostale zadatke čekajući na odgovor drugog sustava. Komunikacija je brža, sigurnija, izbjegavaju se pogreške i bolje se koriste resursi sustava.
- Povećava pouzdanost za isporuku poruke. Korisno je za sustave kojima je potrebna sigurna i pouzdana isporuka poruke kako je predviđeno dogovorom. Poruke ostaju spremljene u posredniku poruka koji se ponaša kao spremnik te ostaju u posredniku sve dok ih druga strana ne obradi.
- Puno fleksibilniji način komunikacije. Sustavi ne moraju biti napisani u istim programskim jezicima niti komunicirati istim protokolom. Nije potrebno uspostaviti put i znati lokaciju drugog sustava s kojim se vrši komunikacija.
- Omogućuje nezavisnost sustava koji međusobno komuniciraju.
- Skalabilnost posrednika poruka omogućava pametnije iskorištenje memorije i ostalih resursa. Broj redova unutar posrednika poruka se povećava i smanjuje prema potrebi i količini poruka.
- Posrednici poruka nude i mehanizam ponovnog slanja poruka. U slučaju pogreške prilikom isporuke, posrednik poruka će ponoviti slanje odmah ili u dogovoreno vrijeme. Posrednici poruka podržavaju još i mehanizam usmjeravanja nedostavljenih poruka, a taj mehanizam je još poznat pod nazivom "mehanizam mrtvih pisama".

Iako postoji veliki broj prednosti korištenja komunikacijske arhitekture s posrednikom poruka, s druge strane pojavljuje se i nekoliko nedostataka prilikom uporabe posrednika poruka. Unatoč njihovoj prisutnosti u raznim sustavima koji zahtijevaju asinkronu i pouzdanu komunikaciju, postoje određeni sustavi i slučajevi kada korištenje posrednika poruka i nije baš najbolja opcija. Većina takvih sustava zahtjeva sinkronu komunikaciju te posrednici poruka tada nisu pogodni za uporabu. Prije uporabe posrednika poruka, treba sagledati nedostatke njegove uporabe.

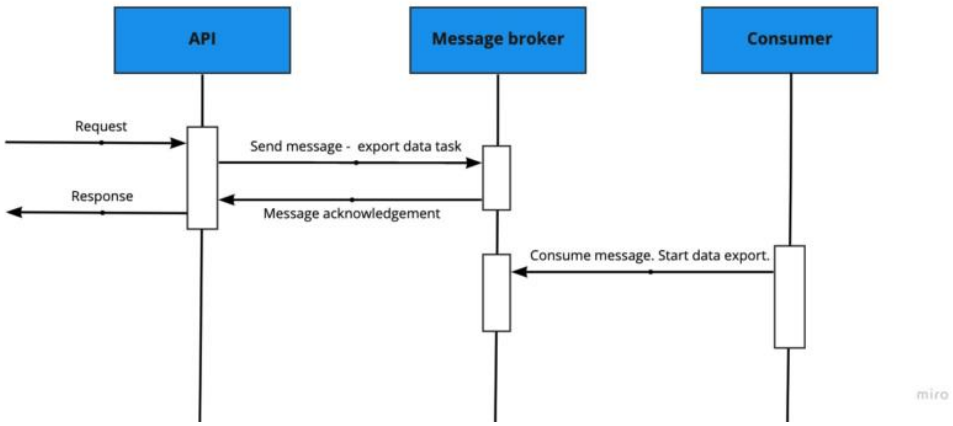
Nedostaci korištenja posrednika poruka u komunikaciji [6]:

- Povećava kompleksnost cjelokupnog sustava. Proces uvođenja posrednika poruka kao novog elementa u postojeći sustav je kompleksan i zahtjevan. Teže je održavanje i praćenje sustava s posrednikom poruka. Zbog toga je potrebno obratiti pažnju na održavanje veza između komponenti, ali i na sigurnosne probleme.
- Smanjuje dosljednost prilikom slanja poruka. Pojedini sustavi mogu kasniti prilikom ažuriranja svojih podataka jer ih posrednik poruka još nije obradio i proslijedio. Također, navedeni problem dosljednosti smanjuje performanse ugroženog sustava koji čeka na isporuku.
- Teže je otklanjanje pogreški prilikom korištenja posrednika poruka. Ako postoji problem s posrednikom poruka, teško je otkriti izvor samog problema. Dodatno, teško je otkloniti pogreške u porukama koje su usmjerene kroz posrednik poruka.
- Posrednici poruka mogu uzrokovati sigurnosne probleme. Kako bi se izbjegao problem sigurne isporuke, posrednici poruka se moraju zaštiti korisnim mehanizmima.
- Uzrokuje povećanje troškova prilikom korištenja posrednika poruka u sustavu. Kupnja i sama konfiguracija posrednika poruka stvara dodatne troškove. Zahtjeva specijalizirane stručnjake za upravljanje, konfiguraciju i održavanje posrednika poruka. Potrebno je odrediti brojne varijable prilikom konfiguracije kao što su veličina reda i oblik poruka, model komunikacijske arhitekture, parametre isporuke te vrijeme življenja poruke TTL (engl. *Time-to-live*).
- Smanjuje performanse sustava ako su posrednici poruka konfigurirani na pogrešan način. Loše odražavani posrednici poruka mogu stvarati dodatna kašnjenja u komunikaciji te smanjiti propusnost prijenosa poruka.

## 2.6. Posrednici poruka i ostale komunikacijske tehnologije

### Posrednik poruka i aplikacijsko programsko sučelje

Reprezentacijski prijenos stanja (engl. *Representational State Transfer)*, REST, je softverska arhitektura koja predstavlja jedinstveno sučelje između fizički odvojenih komponenti. Često se koristi pomoću uporabe interneta u arhitekturi poznatoj kao klijent-poslužitelj (engl. *Client-Server*). Većina mikrousluga koristi navedenu arhitekturu za međusobnu komunikaciju te očekuju sinkrone odazive. Reprezentacijski prijenos stanja koristi HTTP (engl. *Hypertext Transfer Protocol*) protokol prilikom komunikacije. HTTP protokol je protokol koji koristi mehanizam zahtjev/odgovor (engl. *request/response)* između klijenta i poslužitelja. Prilikom slanja zahtjeva poslužitelju od strane klijenta, taj isti klijent zahtjeva što brži odgovor na njegov zahtjev. Ovakav način komunikacije koja uključuje čekanje na odgovor s druge strane, može uzrokovati zastoj i neisporuku poruke. U slučaju da poslužitelj nastoji poslati odgovor, ali klijent koji je poslao zahtjev nije dostupan u tom trenutku, poslužitelj će biti blokiran i čekati će na odgovor klijenta. Upravo se zbog toga moraju implementirati mehanizmi za otklanjanje pogrešaka i izbjegavanje beskonačnog čekanja s obje strane prilikom komunikacije.




*Slika 2.5. Prikaz komunikacije između posrednika poruka i aplikacijskog programskog sučelja, s Interneta, https://tsh.io/wp-content/uploads/fly-images/17917/message-broker-example-3_-800x366.png*

S druge strane, korištenje posrednika poruka omogućuje asinkronu komunikaciju te se izbjegava čekanje druge strane na obradu poruke. Takav način osigurava fleksibilnost te povećava izbjegavanje pogrešaka prilikom komunikacije.

Dodatno, posrednik poruka s modelom Publish-Subscribe podržava skalabilnost prilikom mijenjanja broja pretplaćenih strana unutar komunikacije. Također, posrednik poruka bilježi stanja sudionika komunikacije te će poruka biti isporučena nakon što odredišni sustav bude dostupan. Kako bi se iskoristile prednosti obje navedene komunikacijske arhitekture, moguće je istovremeno korištenje REST-a i posrednika poruka prilikom implementacije.

**Posrednik poruka i sučelje za prijenos događaja**

Sučelje za prijenos događaja (engl. *Event Streaming Platform*) je komunkacijska arhitektura koja podržava isključivo Publish-Subscribe model prijenosa poruka. [3] Sučelja za prijenos događaja su izrazito skalabilna jer rade na prijenosu velike količine poruka te mogu spremiti poruke u teme (engl. Topic) na određeni vremenski period. Teme su korisne jer podržavaju istovremeno slanje poruka od strane više pošiljatelja i primanje poruka od strane više primatelja.

Za razliku od posrednika poruka, sučelja za prijenos događaja ne mogu osigurati pouzdanu isporuku poruke niti pratiti koje su pretplaćene strane zaprimile poruku. Sučelja za prijenos događaja će pohranjivati sve poruke koje se šalju tijekom prijenosa, dok će posrednik poruka odmah nakon primitka poruke od strane primatelja, tu istu poruku obrisati. Iako su sučelja za prijenos događaja značajno skalabilna i korisna za veći broj sudionika, često nemaju dovoljno razvijene mehanizme za otklanjanje pogrešaka, slabije usmjeravaju poruke i nije prisutna fleksibilnost kao što je to slučaj kod posrednika poruka.


*Slika 2.6. Prikaz komunikacijske arhitekture sučelja za prijenos događaja, s Interneta,* https://files.speakerdeck.com/presentations/a929c8ef60a64e7b929d7cc4883b0ef9/slide_10.jpg

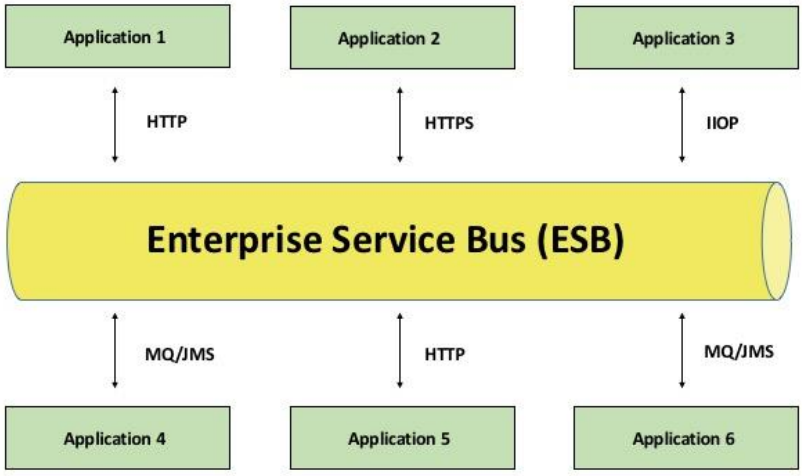
**Posrednik poruka i Enterprise Service Bus (ESB)**

S obzirom da još uvijek ne postoji službeni prijevod naziva "Enterprise Service Bus" na hrvatskom jeziku, u daljnjem tekstu će se koristiti skraćenica ESB.

ESB je komunikacijska arhitektura koja uključuje različite servise i aplikacije u poslužiteljsko-orijentiranoj arhitekturi. Kao i posrednik poruka, ESB omogućuje razmjenu poruka između sustava bez ovisnosti o dostupnosti drugog sustava i čekanja. ESB kombinira razmjenu poruka, web usluga, transformaciju podataka i podržava inteligentno usmjeravanje. Inteligentno usmjeravanje je skup pravila i načela za integriranje brojnih aplikacija zajedno preko bus infrastrukture.

Poput posrednika poruka, ESB omogućuje asinkronu razmjenu poruka te posjeduje mogućnost prevođenja raznih formata poruka. ESB je idealan za implementaciju zahtjeva koji služe kao sigurnosna provjera, kao što su autentifikacije korisnika. Često se koristi kao sloj između "frontend sloja" i "backend sloja" aplikacija.

ESB je složena struktura koja zahtjeva specijalizirane stručnjake za konfiguraciju i daljnji proces održavanja. Otklanjanje pogrešaka je otežano i može imati značajne negativne posljedice na sve povezane sustave. Iako je ESB omogućio odmak od integracije „točka-do-točka“, pojavom novih izazova na tehnološkoj sceni ESB je djelomično izgubio svoju svrhu. Upravo je korištenje posrednika poruka nadoknadilo funkcije koje je pružao ESB uz puno veću fleksibilnost, manje pogrešaka i jeftinije troškove održavanja.




*Slika 2.7. Shematski prikaz komunikacije unutar Enterprise Service Bus-a, s Interneta,* [https://it.ucla.edu/sites/default/files/media/images/esb_diagrams_1b.jpg](https://it.ucla.edu/sites/default/files/media/images/esb_diagrams_1b.jpg)

## 2.7. Primjeri konkretne uporabe posrednika poruka

U današnje vrijeme, potrebe tržišta i klijenata su svakim danom sve veće te je potrebno uslugu pružiti u što boljim uvjetima. Kako se s tehnološkim razvojem povećavao broj različitih usluga, bilo je potrebno osmisliti sučelje koje će omogućiti komunikaciju između različitih sustava napisanih različitim programskim jezicima. Upravo je posrednik poruka komunikacijska arhitektura koja je omogućila olakšano povezivanje više različitih sustava istovremeno. Zahvaljujući posredniku poruka, omogućena je asinkrona komunikacija između sustava, otklanjanje pogrešaka je olakšano te je osigurana sigurna i pouzdana isporuka poruka.

U nastavku poglavlja će biti opisane najčešće uporabe posrednika poruka pri rješavanju konkretnih problema komunikacije među sustavima [8]:

- Financijske transakcije i internetsko plaćanje
  Sve veći broj korisnika danas koristi internetsko bankarstvo za obavljanje različitih financijskih transakcija. Prilikom plaćanja proizvoda ili usluge na internetu, poželjno je da se transakcija plaćanja izvrši na siguran način i samo jedan jedini put. Upravo ovakav način komunikacije između klijenta i poslužitelja prilikom obavljanja platne transakcije omogućuje korištenje posrednika poruka kao sučelja za komunikaciju. Prilikom provođenja platnih transakcija, posrednik poruka omogućuje prijenos financijskih informacija bez mogućnosti gubitka informacija i osigurava provođenje transakcije bez obzira na dostupnost druge strane koja sudjeluje u komunikaciji. Poželjno je koristiti model Point-to-Point prijenosa poruka zbog sigurnosti i pouzdane isporuke.

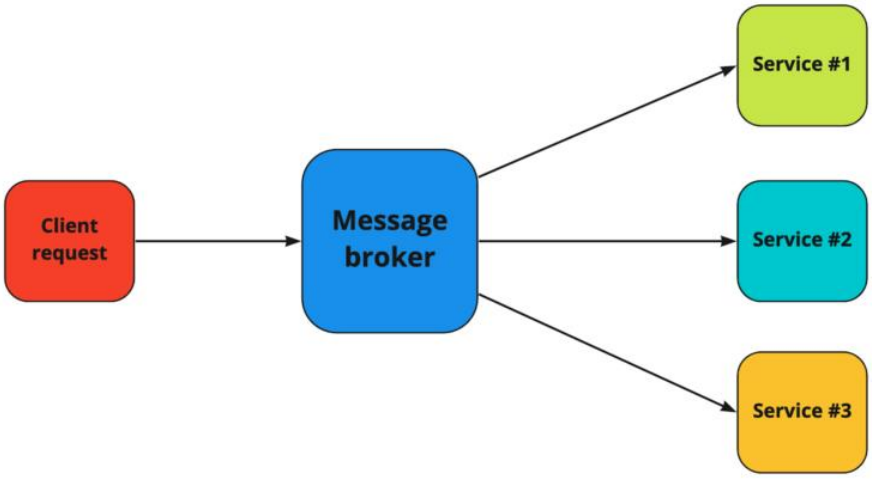
- Internetska trgovina i naručivanje
  Korištenje posrednika poruka prilikom procesa naručivanja robe ili usluga na internet trgovini, omogućuje sigurno i nesmetano obavljanje kupovine bez mogućnosti gubitka podataka ili narudžbe.

- Dugotrajni i zahtjevni zadaci
  U slučaju da korisnik koristi aplikaciju koja mora izvršiti složen i dugotrajan zadatak, bez posrednika poruka, korisnik ne bi mogao koristiti aplikaciju sve dok zadatak ne bude izvršen.

Uvođenjem posrednika poruka prilikom obavljanja složenih zadataka, omogućeno je neometano korištenje aplikacije dok se pokrenuti zadatak ili zahtjev obavlja u pozadini uz pomoć posrednika poruka.

- Komunikacijska arhitektura mikrousluga

  Danas je sve veći broj velikih sustava koji su funkcionalna cjelina manjih mikrousluga. Kako bi se omogućila neometana komunikacija između većeg broja takvih mikrousluga, potrebno je uspostaviti komunikacijsku arhitekturu. Iako je komunikacija izvediva pomoću aplikacijskog programskog sučelja, problemi se javljaju kasnije povećanjem ili smanjenjem broja mikrousluga koje sudjeluju u komunikaciji. Također, u slučaju kvara jedne ili više mikrousluga dolazi do pojave složenih grešaka na aplikacijskom programskom sučelju. Kako bi se otklonili prethodno nabrojani nedostaci s aplikacijskim programskim sučeljem, odabran je posrednik poruka kao odgovarajuća komunikacijska arhitektura. Posrednik poruka unutar sustava ima funkciju središnjeg usmjerivača koristeći Publish-Subscribe model prijenosa poruka. Tada se svaka usluga može pretplatiti na odgovarajući red (engl. *Queue*) iz kojeg želi primati poruke. Korištenje posrednika poruka olakšava dodavanje nove usluge bez potrebe za ponovnom konfiguracijom komunikacijske arhitekture. U slučaju da usluga u određenom trenutku nije dostupna, posrednik poruka će isporučiti poruku kada usluga bude ponovno dostupna isključujući mogućnost gubitka poruke ili pojave greške.



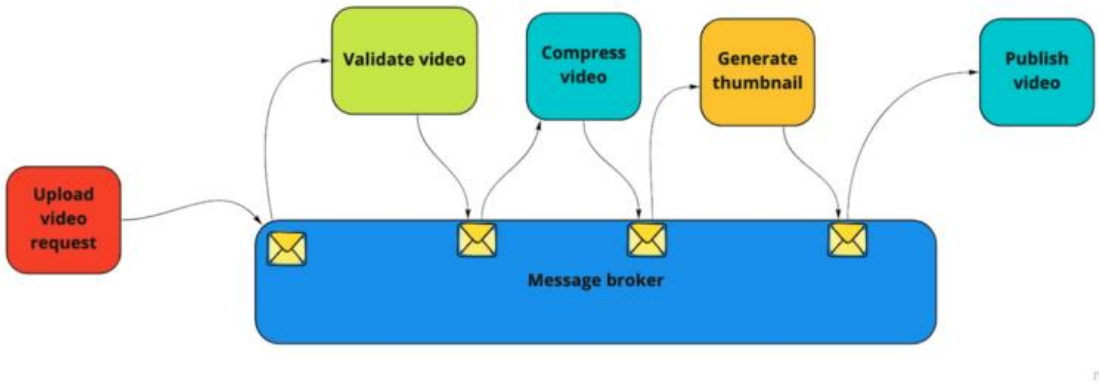

*Slika 2.8. Prikaz komunikacije unutar arhitekture mikrousluga koristeći posrednik poruka, s Interneta, https://tsh.io/wp-content/uploads/fly-images/17916/message-broker-example-4_-800x511.png*

- Mobilne aplikacije

  Većina modernih aplikacija kao što su Facebook, Twitter, Instagram i druge, koriste push obavijest. Kako bi se zaobišlo, često zahtjevno, implementiranje aplikacijskih programskih sučelja, poželjno je koristiti posrednik poruka. Koristeći posrednik poruka, svaki klijent koji je dostupan na mreži i pretplaćen na određeni red u posredniku poruka s Publish-Subscribe modelom prijenosa, moći će primiti push obavijest kada ju aplikacija objavi.

- Mrežne usluge za prijenos videozapisa

  Brojne web aplikacije za prijenos videozapisa koriste različite procese prilikom objavljivanja videozapisa. Prilikom prijenosa videozapisa na mrežnu uslugu za dijeljenje videozapisa kao što je današnji popularni YouTube, videozapis mora proći kroz nekoliko odvojenih procesa. Nakon prvotnog učitavanja (engl. *uploading*) videozapisa, videozapis mora proći proces validacije, kompresije te se mora generirati "sličica" (engl. *thumbnail*) videozapisa. Po završetku prethodno obavljenih procesa, videozapis može biti objavljen na mrežnoj usluzi. Svaki zadatak mora biti započet nakon što je prethodni završen. Upravo ovakav prijenos videozapisa omogućuje komunikacijska arhitektura posrednika poruka. Kako bi se izbjegao problem sinkrone komunikacije i nepotrebnog čekanja, pojedine procese obrade videozapisa možemo gledati kao neovisne usluge koje se istovremeno ponašaju i kao klijenti i kao poslužitelji. Nakon što jedna od usluga za obradu videozapisa završi sa svojim zadatkom, šalje se potvrda o završetku posla posredniku poruka. Posrednik poruka tada pokreće sljedeću uslugu koja obrađuje video.




*Slika 2.9. Proces učitavanja videozapisa na mrežnu uslugu koristeći posrednik poruka, s Interneta, https://tsh.io/wp-content/uploads/fly-images/17923/message-broker-example-5_-800x307.png*

## 2.8. Pregled aktualnih posrednika poruka otvorenog koda

Danas postoji nekoliko poznatih posrednika poruka otvorenog koda. To su korisni i napredni alati za komunikaciju koji se u svojoj osnovi međusobno ne razlikuju. U slučaju obavljanja složenijih procesa i slanja većeg obujma podataka, pojavljuju se razlike koje čine određeni posrednik poruka idealnim alatom za određene slučajeve. U nastavku teksta će biti nabrojani poznati posrednici poruka, opisane njihove karakteristike i osnovni principi rada.

### RabbitMQ

RabbitMQ je jedan od najčešće korištenih i vrlo popularnih posrednika poruka otvorenog koda. [9] Napisan je u Erlang programskom jeziku i održavan od strane „Pivotal Software Foundation“. RabbitMQ osigurava sigurnu lokaciju za slanje i primanje poruka te podržava veći broj različitih komunikacijskih protokola. Korisničke knjižice za upravljanje posrednikom poruka su dostupne za veliki broj poznatih programskih jezika. RabbitMQ podržava četiri vrste izmjenjivača i rješava složene probleme usmjeravanja. Izmjenjivač je dio posrednika poruka koji usmjerava poruke u odgovarajući red. Zbog toga u RabbitMQ posredniku poruka, poruke prvo odlaze u izmjenjivač, a tek onda budu proslijeđene u odgovarajuće redove. RabbitMQ koristi AMQP protokol. Dostupan je na brojnim operacijskim sustavima i kompatibilan s popularnim programskim jezicima kao što su Java, .NET, Ruby, Python, PHP, JavaScript, Rust i drugi [14].

### Apache Kafka

Apache Kafka je vrhunski posrednik poruka otvorenog koda koji služi kao platforma za prijenos veće količine podataka. [10] Napisan je u Java i Scala programskom jeziku te je održavan od strane „Apache Software Foundation“. Izvorno je razvijen od strane „LinkedIn-a“ u siječnju 2011. godine te je dobio naziv po poznatom češkom književniku Franzu Kafki. Ponaša se kao sučelje za prijenos događaja i može podržati veliku količinu podataka. Poruke koje prenosi Apache Kafka, pohranjuju se na disk i mogu se nesmetano prenositi iz jedne u drugu točku.

Redovi u Apache Kafka posredniku poruka su replicirani unutar cijelog Kafka sustava kako bi se izbjegao nepotreban gubitak podataka. Apache Kafka je dizajniran za prijenos velike količine podataka tijekom dužeg perioda ili za prijenos podataka prilikom brzih i skalabilnih procesa.

**Amazon SNS**

Amazon Simple Notification Service se može promatrati kao posrednik poruka koji služi kao usluga za slanje obavijesti. [11] Razvijen je od strane „Amazon Web Services“ 2010. godine. Omogućuje prijenos velike količine poruka s niskom cijenom infrastrukture. Najčešće se koristi kod mobilnih uređaja za slanje obavijesti uz Publish-Subscribe model prijenosa poruka. Može prenijeti poruke u više od dvjesto država svijeta. Za razliku od ostalih nabrojenih usluga, Amazon SNS nije otvorenog koda.

**Amazon SQS**

Amazon Simple Queue Service je razvijena usluga koja pruža korištenje redova za prijenos poruka. [12] Razvijena je 2004. godine od strane „Amazon.com“ kao beta verzija, a dostupnom je postala 2006. godine. Kao i Amazon SNS, Amazon SQS je dio „Amazon Web Service-a“. Amazon SQS je prilagodljive veličine ovisno o količini podataka koja se prenosi. Usluga pruža određeni broj redova za prijenos poruka besplatno, dok je za proširenje potrebno platiti manji novčani iznos.

Za razliku od Amazon SNS usluge, Amazon SQS obavlja prijenos poruka koristeći push metodu direktnog slanja poruke bez potrebe za dodatnim pohranjivanjem iste poruke. Također, klijent koji prima poruku koristi pull metodu prilikom preuzimanja poruke. Nakon što je poruka unutar reda, klijent, samostalno, metodom pull preuzima poruku iz Amazon SQS-a. Amazon SQS garantira isporuku poruke jedan jedini put te podržava veliki broj programskih jezika kao što su Java, Ruby, Python, .NET, PHP i JavaScript.

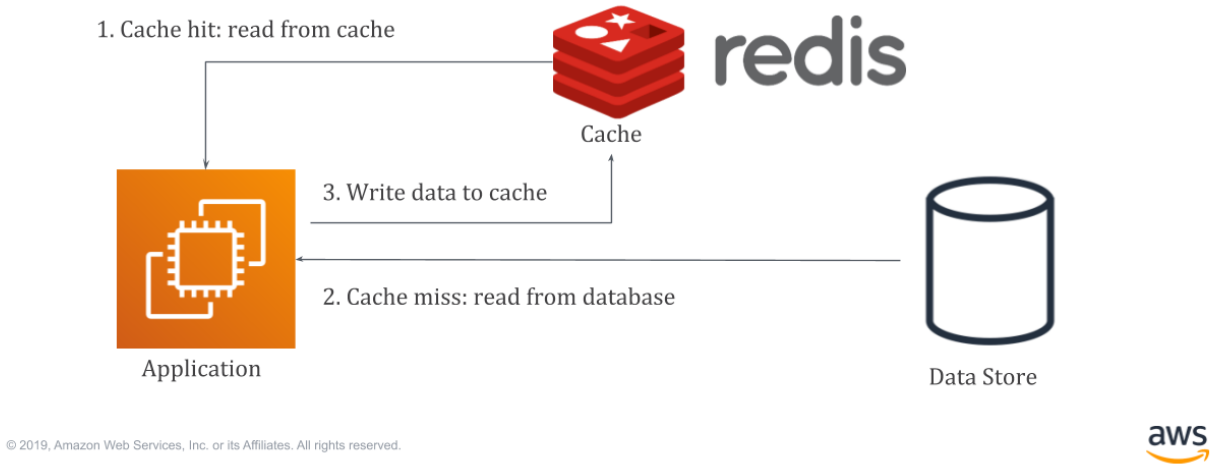
Za razliku od posrednika poruka otvorenog koda kod kojih korisnici sami vrše konfiguraciju servera, Amazon odrađuje posao implementacije posrednika poruka i održavanja servera što se naplaćuje ovisno o uporabi. Mnoge popularne mrežne usluge kao što su Netflix, Dropbox i Nextdoor koriste uslugu Amazon SQS-a prilikom pružanja svojih usluga.

**Redis**

Redis je usluga otvorenog koda koja pruža pohranu različitih struktura podataka unutar memorije. [13] Redis je korišten kao ključ-vrijednost baza podataka, priručna memorija (engl. *Cache)* i posrednik poruka.

Podržava različite strukture podataka kao što su znakovni nizovi, liste, mape i setovi. Razvijen je od strane Salvatorea Sanfillippoa  u 2009. godini i napisan je u C programskom jeziku.

Razlikuje se od standardnih baza podataka jer istovremeno može služiti i kao spremnik i kao priručna memorija. Dizajniran je na način da se podaci čitaju i uređuju u radnoj memoriji, ali se dodatno pohranjuju na sekundarnu memoriju kako bi se u slučaju ponovnog pokretanja servera vratili u radnu memoriju. Kompatibilan je s brojnim programskim jezicima na klijentskoj strani kao što su C, C++, C#, Java, JavaScript, Rust, PHP i brojni drugi.




*Slika 2.10. Prikaz komunikacije koristeći Redis posrednik poruka, s Interneta,* https://www.knot35.com/images/2020/12/Redis-Cache-1.png

# 3. OPIS PRAKTIČNOG DIJELA ZAVRŠNOG RADA

## 3.1. Priprema radnog okruženja

Početak izrade praktičnog dijela završnog rada je započeo s osmišljavanjem i okvirnim dizajniranjem što bi sustav, koji će koristiti posrednik poruka, trebao sadržavati. Kao radno okruženje prilikom pisanja koda odabran je Visual Studio Code zbog jednostavnosti korištenja, preglednosti koda i prilagođenosti prilikom dizajniranja sustava. Kao sustav koji će koristiti posrednik poruka, implementiran je sustav koji će korisniku pružati uslugu aktivacije mobilne tarife. Sustav je implementiran na Windows 11 operativnom sustavu.

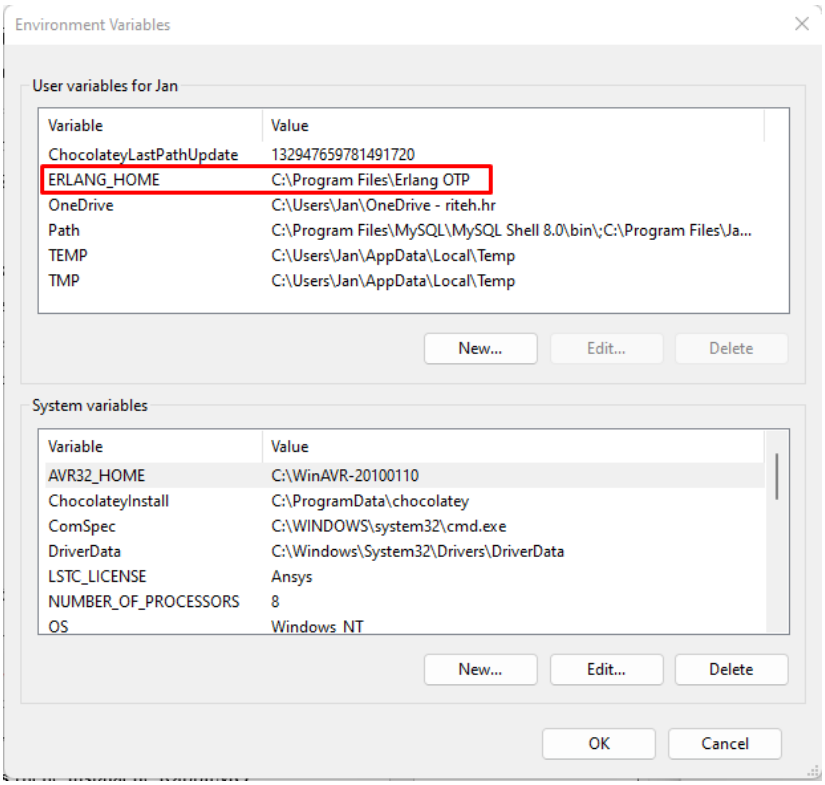
Kao glavni cilj, uspostavljena je veza između pojedinih usluga od kojih se sustav sastoji koristeći posrednik poruka. Posrednik poruka koji je odabran prilikom implementacije sustava je RabbitMQ. Glavni razlozi za odabir posrednika poruka kao što je RabbitMQ je njegova jednostavna implementacija, otvorenost koda, dostupna dokumentacija i kompatibilnost s raznim programskim jezicima. Kao jezik u kojem će se implementirati usluge, odnosno cijeli sustav, odabran je JavaScript u okruženju Node.js.

## 3.2. Instalacija potrebnih alata

Prije same implementacije sustava, bilo je potrebno instalirati određene programe i alate za uspostavljanje funkcionalnosti zamišljenih usluga. Kao program za pružanje usluge baze podataka, izabran je MongoDB baza podataka s razvojnim alatom MongoDB Compass za upravljanje i praćenje baze podataka. Express.js je odabran kao okvirni poslužitelj za pokretanje Node.js aplikacije. RabbitMQ je odabran kao posrednik poruka te je njegova instalacija izvršena ručno na Windows 11 operativnom sustavu. U nastavku će biti opisan proces ručne instalacije RabbitMQ posrednika poruka. Kao dodatni program, za provjeru funkcionalnosti i testiranje pojedinih usluga, korišteno je Postman sučelje za komunikaciju s web aplikacijama. Također, kao knjižica za prikazivanje pogrešaka prilikom pokretanja programa i lakše pronalaženje pogrešaka, korištena je knjižica Loggin. Za slanje SMS obavijesti je korištena Twillio usluga. Potrebno je izraditi korisnički račun na službenoj Twillio web stranici koja će nakon registracije korisniku isporučiti privremeni telefonski broj za korištenje usluge.

### 3.3. Ručna instalacija RabbitMQ posrednika poruka

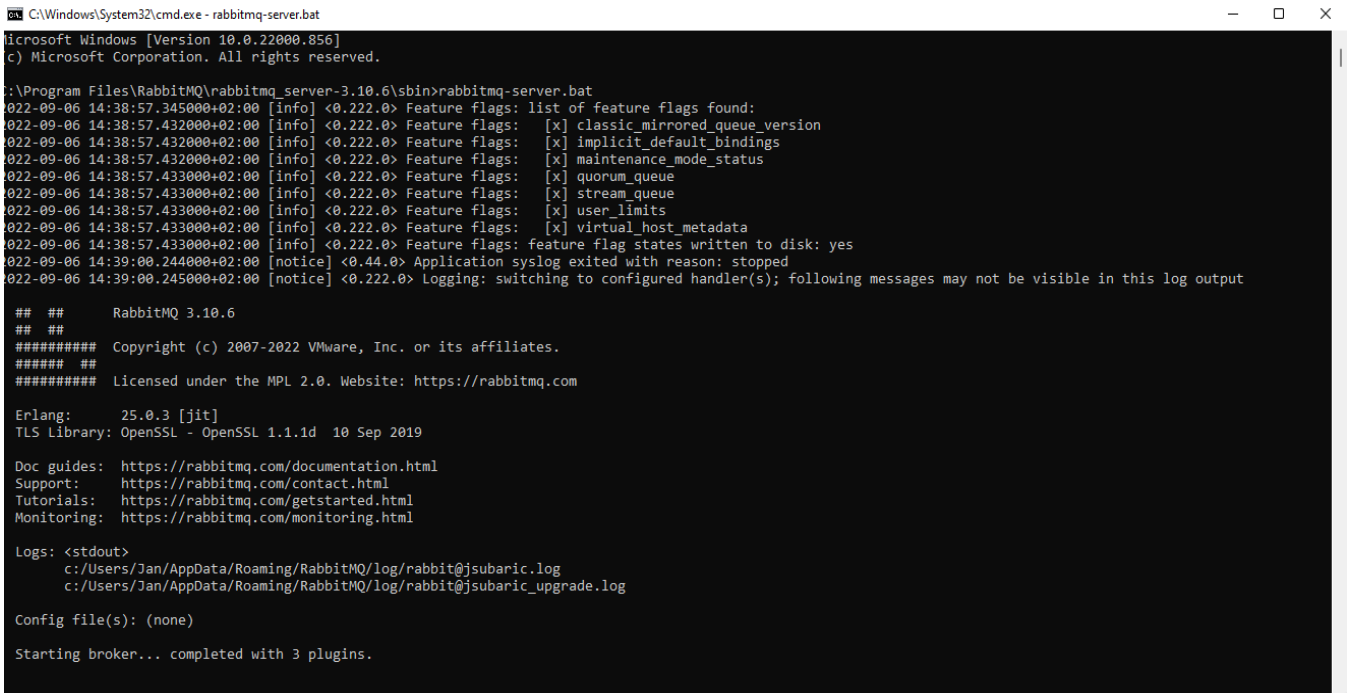
Prije same instalacije RabbitMQ programa za posredovanje porukama, bilo je potrebno preuzeti Erlang. [14] Erlang je funkcionalni programski jezik korišten za implementaciju velikih i promjenjivih sustava, a RabbitMQ je napisan u Erlang programskom jeziku. Odabrana je najnovija verzija Erlang-a i pokrenuta je programska instalacija na operativnom sustavu. Kako bi se omogućilo korištenje Erlang-a iz naredbenog retka, bilo je potrebno dodati lokaciju Erlang programa u skup varijabli okruženja (engl. *Environment Variables*) pod nazivom ERLANG_HOME. Uspješnost konfiguracije se može provjeriti naredbom *echo %ERLANG_HOME%* koja će u slučaju ispravne instalacije vratiti ispravnu lokaciju Erlang-a.




*Slika 3.1. Postavljanje lokacije programskog jezika Erlang unutar varijabli okruženja*

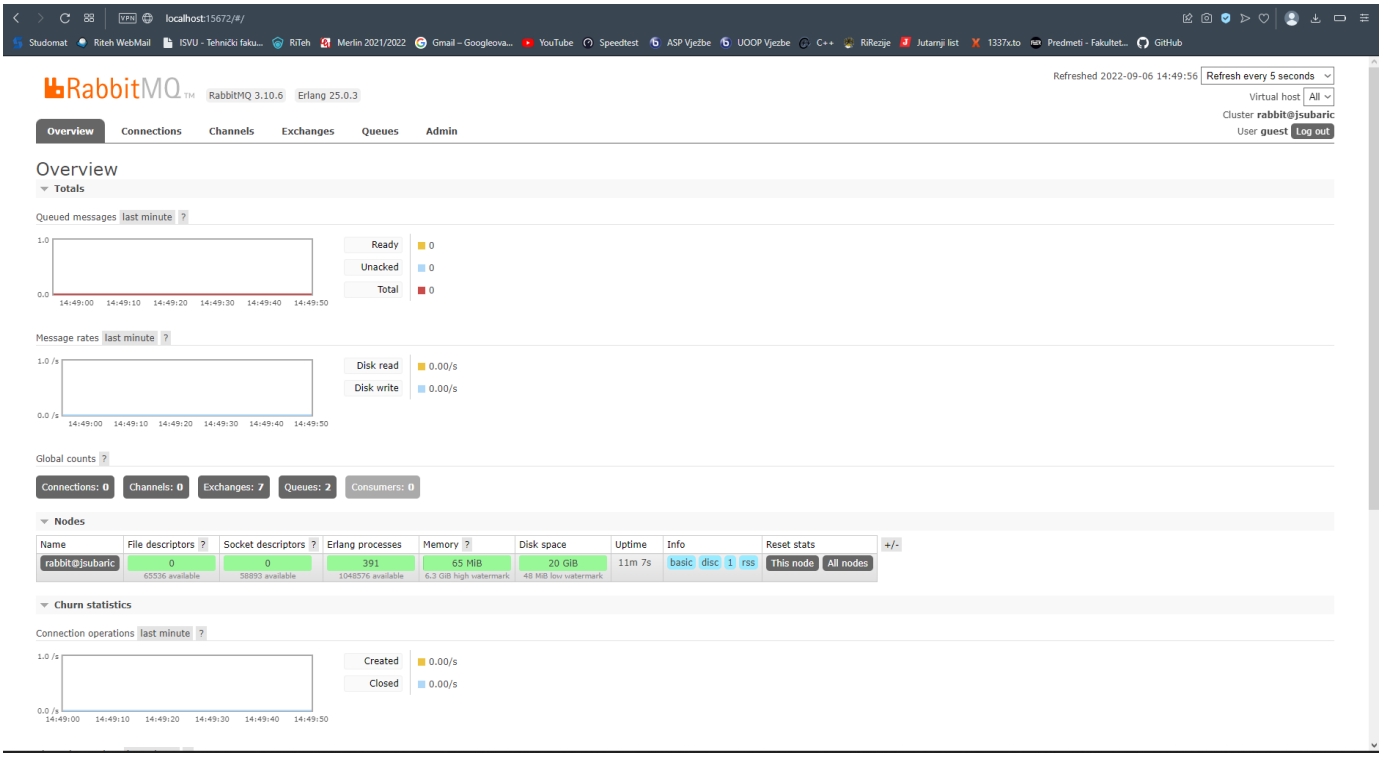
Nakon uspješne instalacije i postavljanja Erlang programskog jezika, može se preći na samu instalaciju RabbitMQ programa. Preko GitHub platforme, preuzet je službeni paket za instalaciju RabbitMQ usluge za Windows operativni sustav. Nakon preuzimanja paketa, paket je raspakiran u C:\Program Files\RabbitMQ i pokrenut je naredbeni redak unutar *sbin* mape.

S naredbom *rabbitmq-plugins.bat enable rabbitmq_management* su instalirani i omogućeni potrebni dodaci za normalno funkcioniranje RabbitMQ servera. Pokretanje RabbitMQ servera je izvršeno s naredbom *rabbitmq-server.bat*. [15]




*Slika 3.2. Pokretanje RabbitMQ servera kroz naredbeni redak*

Nakon uspješnog pokretanja RabbitMQ servera kroz naredbeni redak, omogućen je pristup korisničkom upravljačkom sučelju za upravljanje posrednikom poruka na web pregledniku na adresi *http://rabbitmq:15672/*. Prilikom pristupa aplikaciji za upravljanje posrednikom poruka, potrebno je unijeti vrijednost *guest* pod username te vrijednost *guest* pod password.



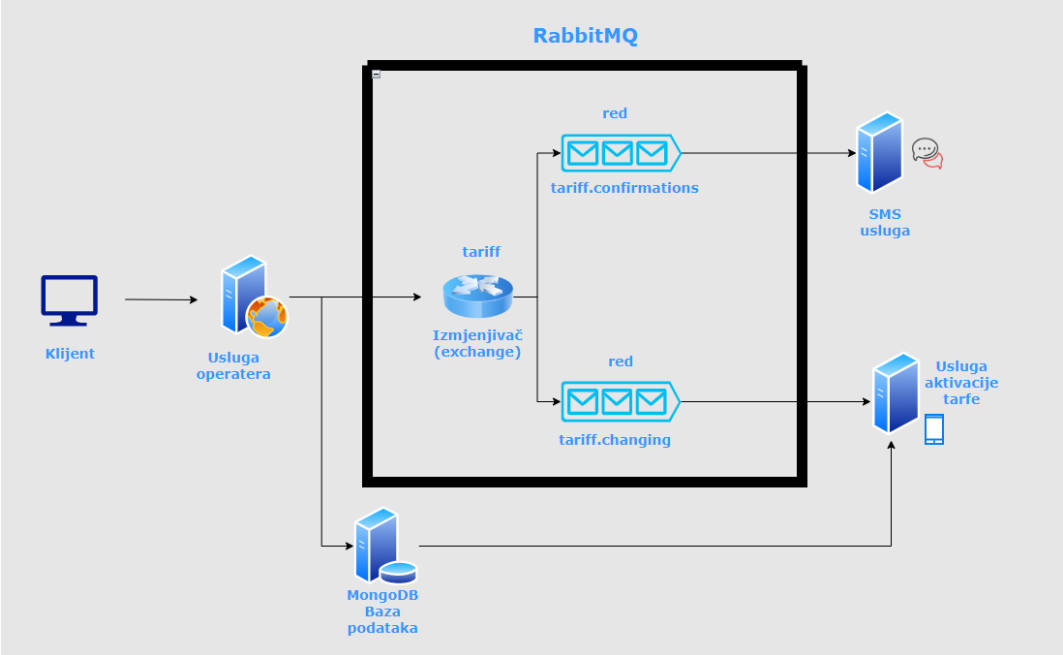

Slika 3.3. Prikaz RabbitMQ Managment usluge na adresi http://rabbitmq:15672/

### 3.4. Sustav i njegove komponente

Prilikom dizajniranja sustava, odlučeno je da će se sustav sastojati od tri zasebne usluge. Sve tri usluge postoje samostalno te koriste posrednik poruka kao komunikacijsko sredstvo. Sustav za aktivaciju mobilne tarife je zamišljen kao sustav preko kojeg korisnici mogu odabrati tarifu i tražiti njenu aktivaciju, a pri tome dobiti SMS potvrdu o potvrdi njihovog zahtjeva. Korisnik će moći provjeriti stanje svojeg zahtjeva koristeći identifikacijski broj zahtjeva. Kako je sustav za promjenu tarife dizajniran kao sustav koji je integriran od tri različite usluge, potrebno je implementirati:

- Uslugu za slanje SMS obavijesti
- Uslugu operatera
- Uslugu aktivacije tarife

U sljedećim poglavljima će biti opisane osnovne funkcije implementiranog sustava i detaljno objašnjen način povezivanja s RabbitMQ posrednikom poruka za prijenos poruka. Naglasak će biti na opisu povezivanja pojedine usluge s posrednikom poruka, dok će razvoj pojedinih pozadinskih aplikacija biti ukratko objašnjen.



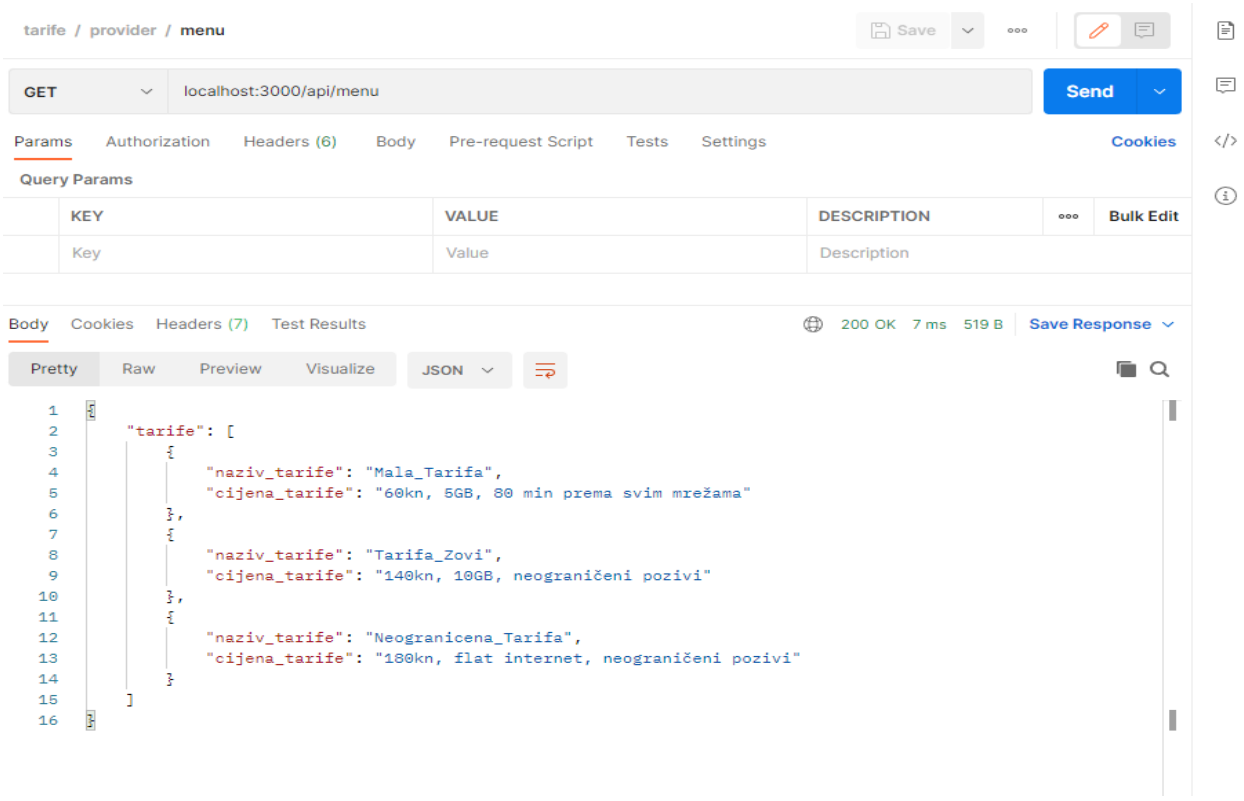

*Slika 3.4. Prikaz sheme sustava za aktivaciju mobilne tarife*

### 3.5 Opis funkcija sustava

Usluga operatera je najsloženija usluga unutar sustava koja omogućava korisniku da provjeri koje su tarife dostupne, pokrene aktivaciju tarife i prati stanje aktivacije u stvarnom vremenu. Usluga operatera zahtijeva povezanost s RabbitMQ poslužiteljem i MongoDB bazom podataka. Prije pokretanja usluga, potrebno je instalirati potrebne pakete naredbom *npm install* unutar konzole te nakon toga pokrenuti uslugu naredbom *npm start*. Nakon što se usluga pokrene i ispiše se obavijest o uspješno uspostavljenoj vezi s RabbitMQ posrednikom poruka i MongoDB bazom podataka, korisnik može koristiti njene funkcionalnosti.

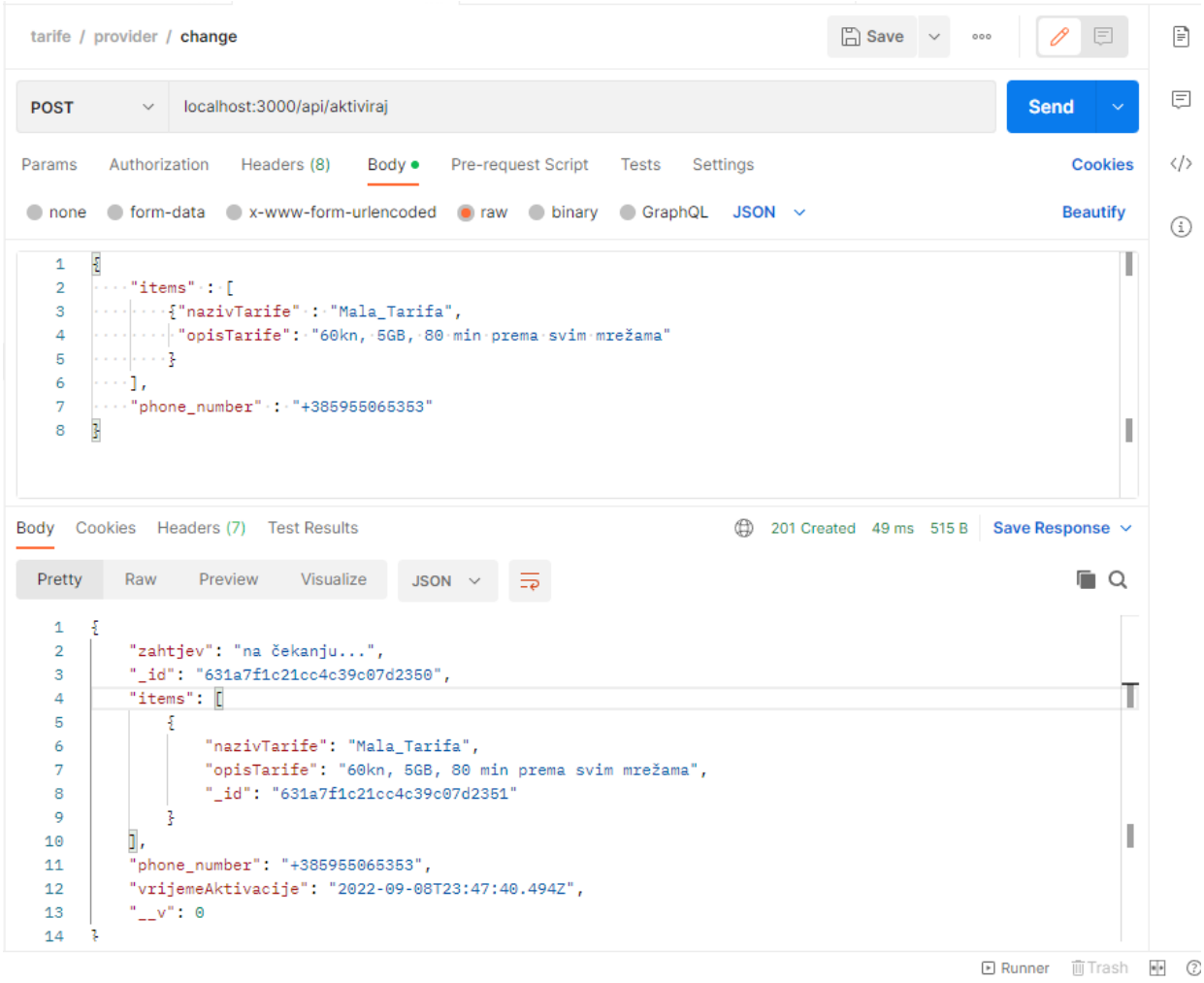
Prilikom testiranja usluga je korišten Postman alat jer sustav nema dizajnirano grafičko korisničko sučelje, već je implementiran samo pozadinski dio kako bi se objasnio rad s posrednikom poruka.

Za dohvaćanje dodatnih informacija o dostupnim tarifama koristi se GET metoda te se pristupa URL lokaciji *localhost:3000/api/menu* kroz sučelje alata Postman. Ako je dohvaćanje podataka uspješno, ispisuju se tarife i njihovi opisi. Uspješno dohvaćanje podataka je vidljivo na slici 3.5. ispod.




*Slika 3.5. Dohvaćanje informacija o dostupnim tarifama unutar Postman alata*

Nakon što je korisnik dohvatio informacije o dostupnim tarifama i njihovim pogodnostima, može odabrati tarifu koja mu najviše odgovara te ju aktivirati kroz sustav. Aktivacija željene tarife uključuje POST metodu na URL lokaciji *localhost:3000/api/aktiviraj*. Unutar tijela metode, potrebno je unijeti naziv tarife i opis željene tarife te se unosi telefonski broj na kojem se tarifa želi aktivirati. Ako je zahtjev ispravno unesen, na konzoli se ispisuju detalji o aktivaciji tarife.

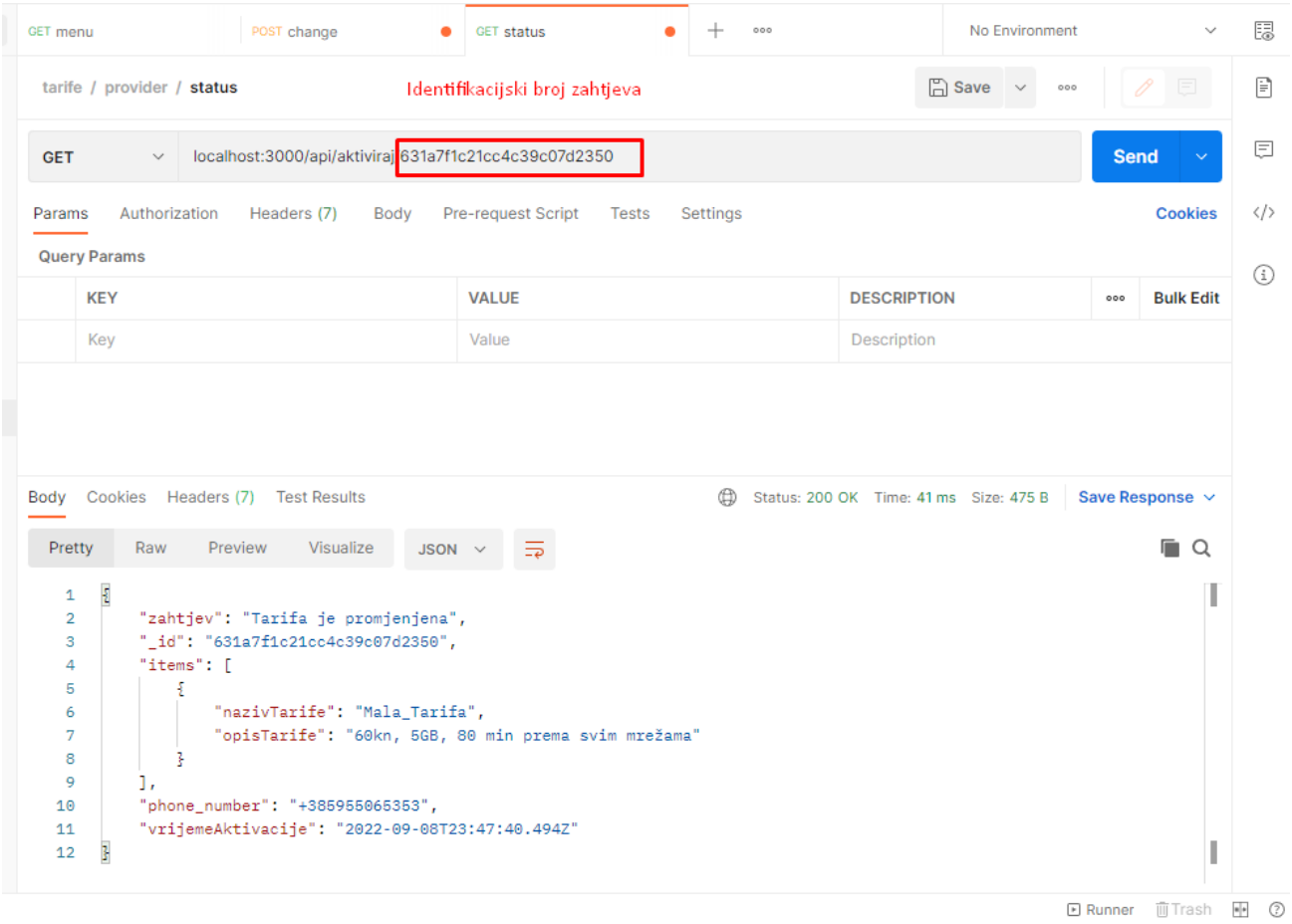



*Slika 3.6. Prikaz slanja zahtjeva za aktivaciju tarife koristeći POST metodu unutar alata Postman*

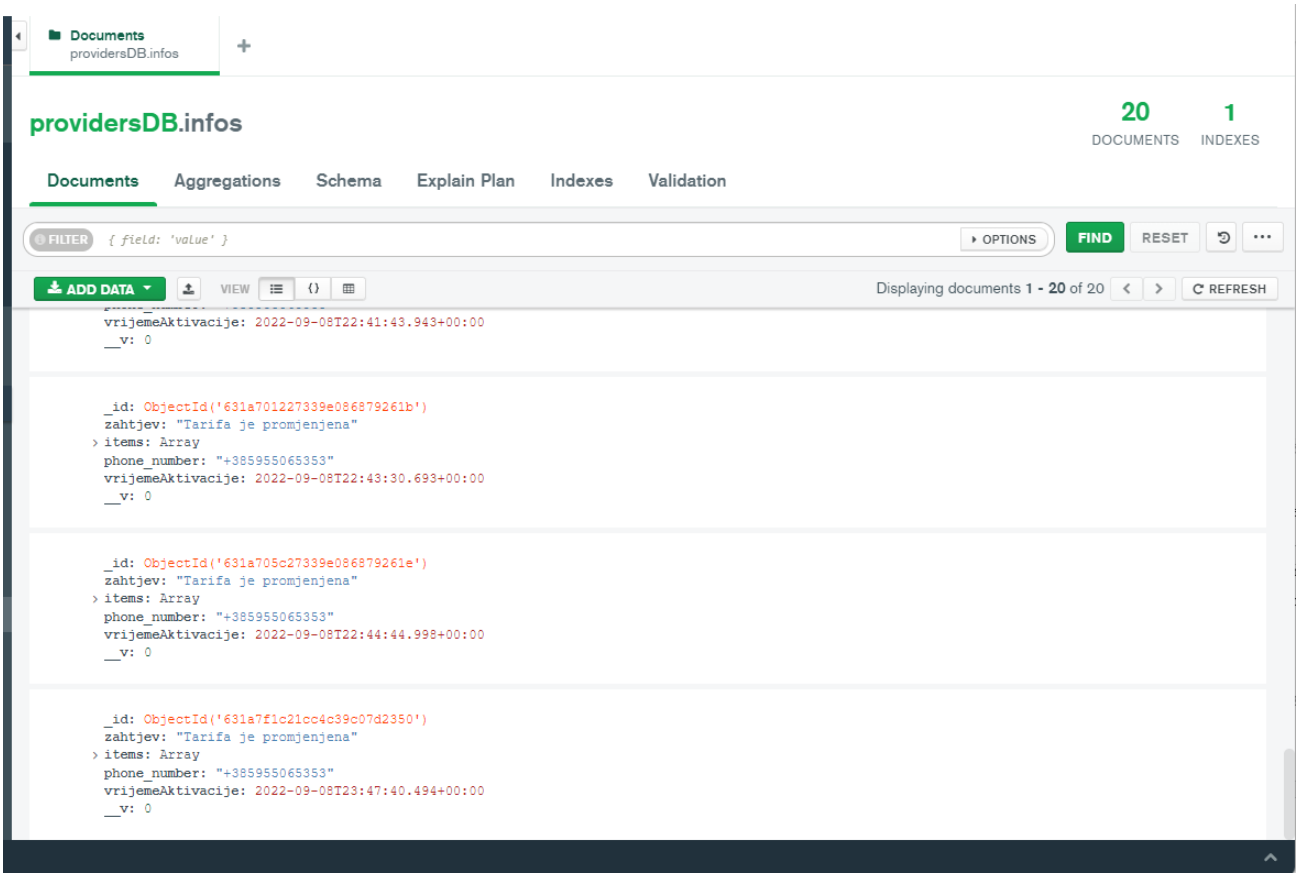
Nakon što je zahtjev za aktivacijom tarife uspješan, korisnik dobiva SMS poruku o statusu aktivacije te može koristiti identifikacijski broj, dobiven unutar SMS poruke, kako bi pratio realizaciju zahtjeva. Kako bi gore navedene funkcije bile moguće, usluga operatera mora poslati poruku koja sadrži podatke o željenoj tarifi prema ostalim dvjema uslugama. Prijenos poruke se ostvaruje preko RabbitMQ posrednika poruka koristeći fanout metodu usmjeravanja poruka.

Zahvaljujući posredniku poruka, usluge mogu neometano obavljati svoje zadatke bez potrebe za čekanjem. Povezivanje pojedinih usluga s RabbitMQ poslužiteljem je implementirano unutar datoteke *rabbitMQ.js* te će biti detaljnije objašnjeno.

Korisnik nakon dobivene SMS poruke može provjeriti status zahtjeva za aktivaciju tarife pomoću identifikacijskog broja zahtjeva koji je automatski kreiran. Također, kroz sučelje Postman alata, unosi se identifikacijski broj zahtjeva te se ispisuju informacije o njegovoj realizaciji. Podaci o zahtjevu se pohranjuju unutar MongoDB baze podataka. Korisnik treba koristiti GET metodu te pristupiti URL lokaciji *localhost:3000/api/aktiviraj/(ID zahtjeva)*. Nakon što korisnik pošalje upit o statusu zahtjeva, na konzoli se ispisuju informacije o aktivaciji. Način dohvaćanja podataka je vidljiv na slici 3.7.




*Slika 3.7. Dohvaćanje podataka o realizaciji zahtjeva za aktivaciju tarife*




*Slika 3.8. Zahtjevi za aktivacijom tarife pohranjeni unutar MongoDB baze podataka*

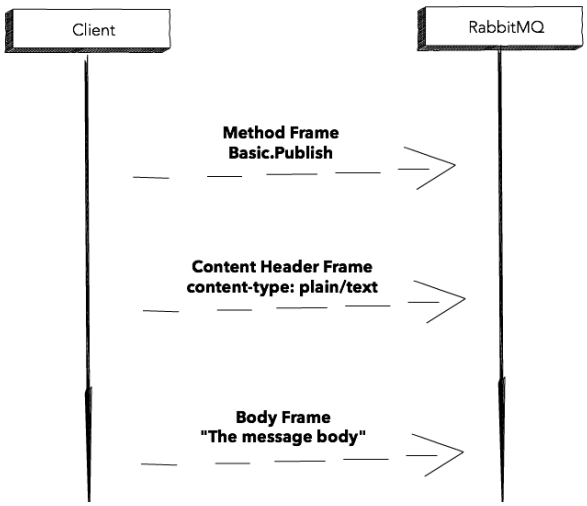

*Slika 3.9. Primjer aktivacijske SMS poruke koja se šalje korisniku*

### 3.6 AMQP protokol

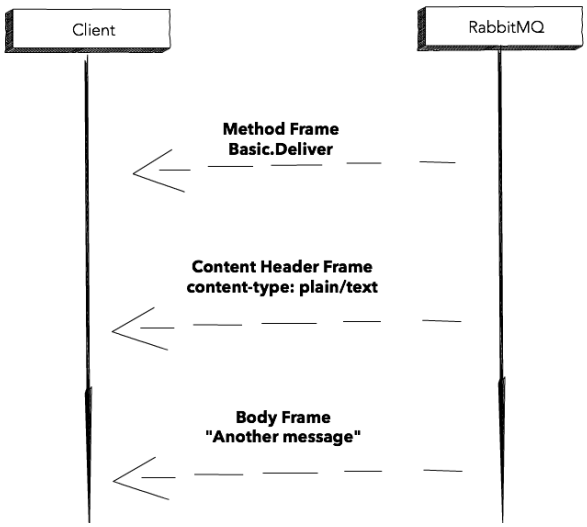
Prije samog povezivanja usluga s RabbitMQ serverom potrebno je ukratko objasniti najčešće korišteni protokol kojeg koristi RabbitMQ. Učitavanje programske knjižice *amqplib* omogućuje korištenje AMQP protokola.

Advanced Message Queuing Protocol (AMQP) je komunikacijski protokol kojeg koristi klijentska aplikacija prilikom povezivanja s RabbitMQ posrednikom poruka. RabbitMQ koristi AMQP protokol kao osnovni komunikacijski protokol prilikom usmjeravanja poruka, upravljanja redovima čekanja te prilikom osiguranja pouzdanosti i sigurnosti prilikom slanja poruka. AMQP je binarni protokol za razliku od ostalih protokola koji su napisani u čitljivom tekstualnom formatu kao što je HTTP protokol. Poruke se prenose u binarnom formatu u obliku okvira (engl. *Frames).* Okvir se sastoji od pet dijelova. Prva tri bajta odnose se na zaglavlje okvira (engl. *Frame header)* te određuju vrstu okvira, kanal kojem pripada okvir te njegovu veličinu u bajtovima. Uloga pojedinih okvira varira tako da će svaki od pet postojećih okvira imati različiti format uloge. AMQP protokol prilikom komunikacije koristi pet različitih okvira. Korišteni okviri su zaglavlje protokola, metoda okvira, zaglavlje sadržaja, tijelo i okvir "otkucaja srca" (engl. *heartbeat).* Treba naglasiti da je AMQP protokol "dvosmjerni" protokol te omogućuje istovremeno slanje poziva od strane aplikacije i od strane RabbitMQ kroz jednu TCP vezu.

Na slici 3.10. i 3.11. se vidi uspostava komunikacije i slanje poruka između klijenta i RabbitMQ-a.




*Slika 3.10. Klijent uspostavlja vezu s RabbitMQ. Šalju se tri okvira i određuje metoda usmjeravanja publish, s Interneta, https://www.brianstorti.com/assets/images/sequence.png*




*Slika 3.11. RabbitMQ uspostavlja vezu s klijentom i šalje tri potrebna okvira, s Interneta, https://www.brianstorti.com/assets/images/sequence-deliver.png*

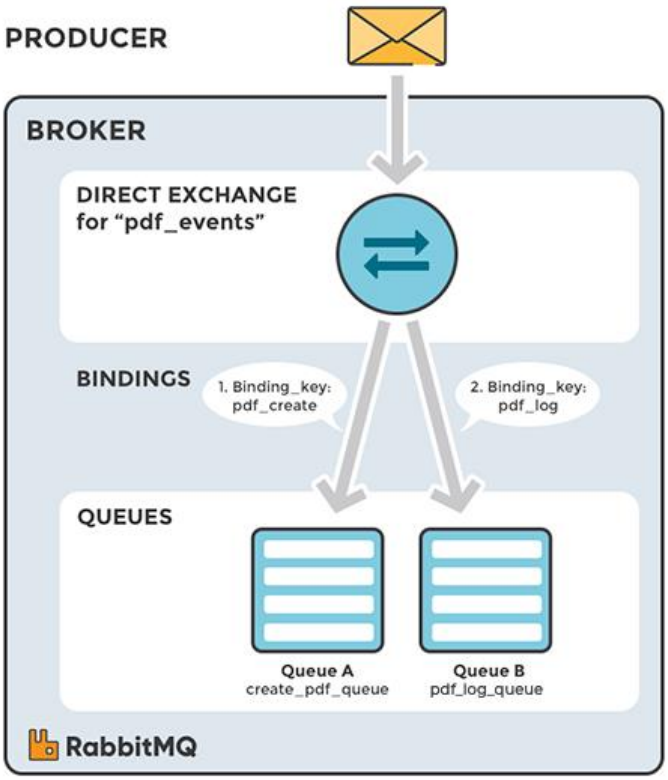
AMQP protokol koristi pet različitih načina za razmjenu poruka, a za potrebe razumijevanja rada biti će objašnjena osnovna načela svake razmjene[17]:

1. Zadano usmjeravanje

   Zadano usmjeravanje je unaprijed određeno direktno usmjeravanje od strane posrednika poruka. Jednostavno je za implementaciju jer se redovi automatski kreiraju s unaprijed zadanim ključem za usmjeravanje (engl. *routing key).* Vrijednost ključa za usmjeravanje je jednaka nazivu reda unutar posrednika poruka.

2. Direktno usmjeravanje

   Direktno usmjeravanje prosljeđuje poruke na temelju vrijednosti ključa za usmjeravanje. Redovi su povezani s izmjenjivačem na temelju ključa za usmjeravanje. Kada nova poruka pristigne u izmjenjivač, izmjenjivač će je proslijediti na red ukoliko ključ za usmjeravanje kojeg posjeduje red odgovara ključu za usmjeravanje kojeg posjeduje poruka.

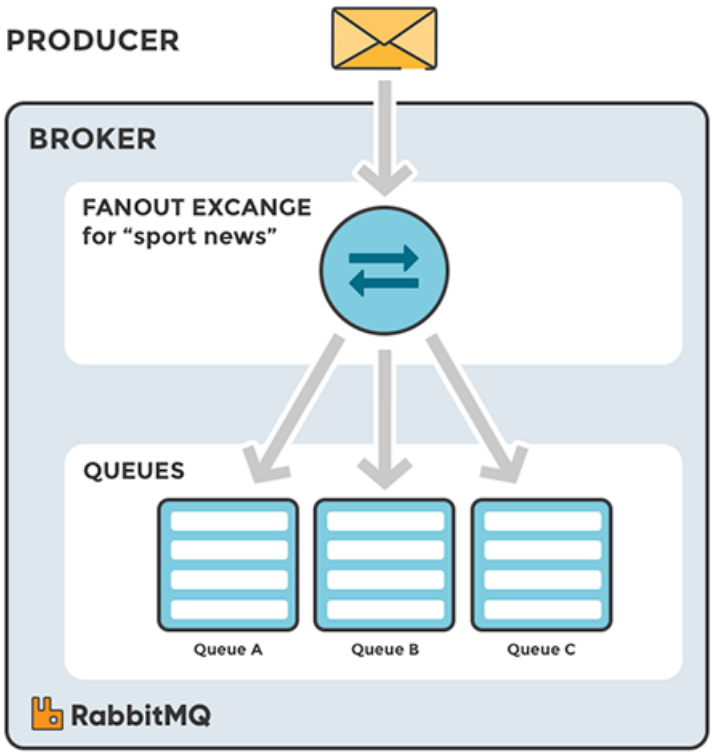



*Slika 3.12. Prikaz direktnog načina usmjeravanja poruka koristeći RabbitMQ, s Interneta, https://www.cloudamqp.com/img/blog/direct-exchange.png*

3. Fanout usmjeravanje

Fanout usmjeravanje omogućuje prosljeđivanje poruka u sve redove koji su povezani s fanout izmjenjivačem. Ovaj način usmjeravanja ne koristi ključ za usmjeravanje prilikom prosljeđivanja poruka u redove, već stvara kopije poruke i prosljeđuje je na sve povezane redove. Ako je izmjenjivač povezan s N brojem različitih redova, poruka koja će stići na izmjenjivač će biti poslana u svih N redova.

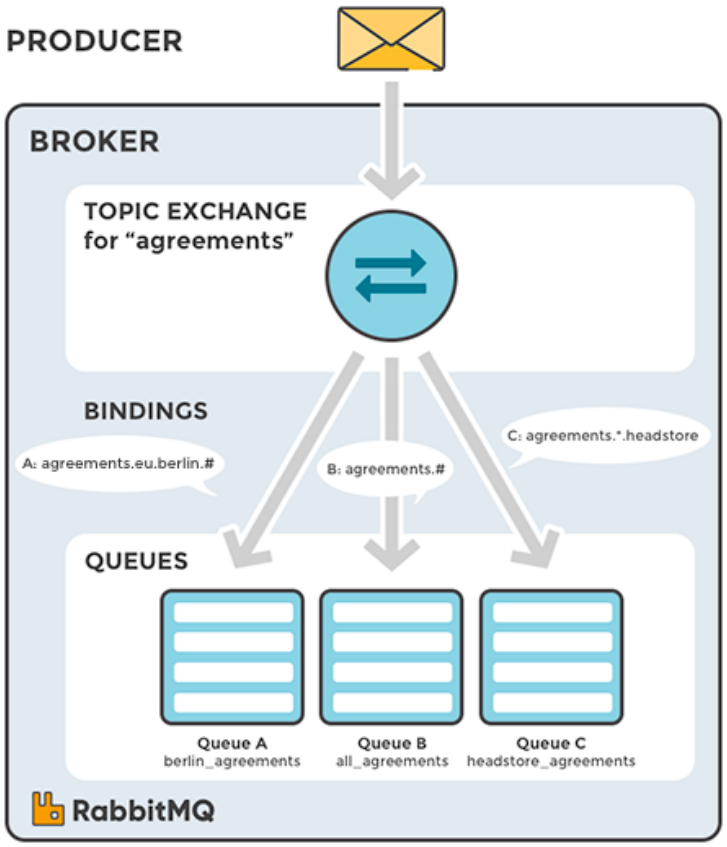
Fanout usmjeravanje je često korišteno prilikom emitiranja (engl. *broadcasting)* podataka svim zainteresiranim stranama. Primjerice, korisno je prilikom slanja podataka koji su korisni većem broju korisnika kao što su vremenske nepogode ili rezultati sportskih utakmica. Sustav za aktivaciju željene mobilne tarife koristi ovaj način usmjeravanja poruka kroz izmjenjivač.




*Slika 3.13. Prikaz fanout načina usmjeravanja poruka koristeći RabbitMQ , s Interneta* [https://www.cloudamqp.com/img/blog/fanout-exchange.png](https://www.cloudamqp.com/img/blog/fanout-exchange.png)

4. Usmjeravanje na temelju teme

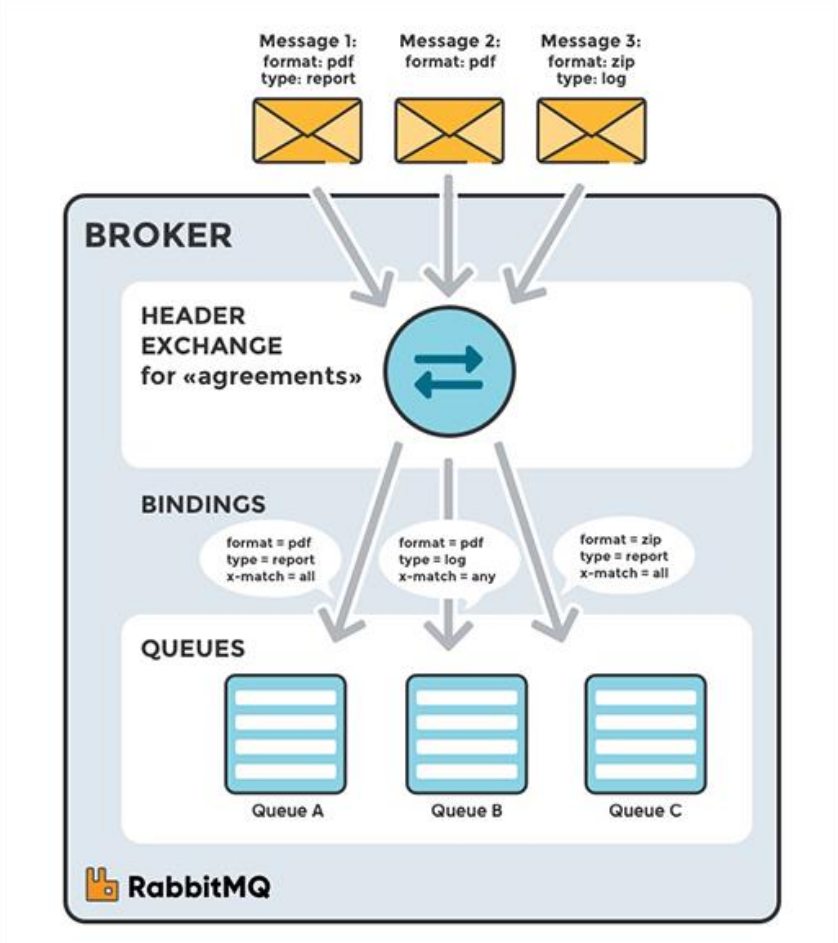
   Usmjeravanje na temelju teme (engl. *Topic exchange)* je način usmjeravanja poruke na jedan ili više redova na temelju podudaranja ključa za usmjeravanje kojeg posjeduje poslana poruka i uzorka s kojim su redovi povezivani s izmjenjivačem. Ključ za usmjeravanje kojeg posjeduje poruka je najčešće sastavljen od liste riječi odvojenih točkom. Često su korišteni uzorci ljestve # (zamjenjuje nula ili više riječi) i zvjezdica * (zamjenjuje samo jednu riječ) prilikom definiranja uzorka veze redova s izmjenjivačem. Usmjeravanje na temelju teme je često korišteno s različitim varijacijama Publish/Subscribe načina usmjeravanja poruka.




*Slika 3.14. Prikaz usmjeravanja poruka na temelju teme koristeći RabbitMQ, s Interneta, [https://www.cloudamqp.com/img/blog/topic-exchange.png](https://www.cloudamqp.com/img/blog/topic-exchange.png)*

5. Usmjeravanje na temelju zaglavlja

   Usmjeravanje na temelju zaglavlja koristi više atributa prilikom usmjeravanja poruka. Ovaj način usmjeravanja ne koristi ključ za usmjeravanje, već koristi atribute navedene u zaglavlju poruke. Poruka se prosljeđuje na red ako vrijednosti zaglavlja odgovaraju vrijednostima na kojima je uspostavljena veza s redom. Unutar veze između reda i izmjenjivača koristi se vrijednost x-match koja određuje koje vrijednosti se moraju podudarati kako bi poruka bila isporučena u određeni red. X-match vrijednost može poprimiti dvije vrijednosti all  ili any. U slučaju da se koristi zadana vrijednost all, sve vrijednosti zaglavlja poruke se moraju podudarati s uzorkom veze. S druge strane, ako se koristi vrijednost any, barem jedna ili više vrijednost unutar zaglavlja mora biti jednaka vrijednostima na uzorku veze.




*Slika 3.15. Prikaz usmjeravanja poruka na temelju zaglavlja poruke koristeći RabbitMQ, s Interneta, https://www.cloudamqp.com/img/blog/headers-exchange.png*

### 3.7. Implementacija posrednika poruka unutar sustava

Sve tri usluge koje su integrirane unutar sustava koriste RabbitMQ posrednik poruka. RabbitMQ omogućuje asinkronu komunikaciju između pojedinih usluga te isključuje potrebu čekanja na odgovor druge strane i smanjuje mogućnost pogreške. U slučaju da jedna od usluga sustava nije dostupna u određenom trenutku, RabbitMQ će ponoviti slanje poruke kada usluga bude dostupna te će se tako izbjeći mogućnost gubitka poruke, ali i nepotrebno čekanje na reakciju nedostupne usluge.

Kao što je prethodno opisano, usluga operatera služi kao klijentska aplikacija preko koje korisnik može provjeriti koje su tarife dostupne, aktivirati određenu tarifu te pratiti stanje njene aktivacije. Kako bi usluga operatera bila dostupna korisniku prilikom obavljanja pozadinskih zadataka, usluga mora poslati poruke usluzi za slanje SMS obavijesti i usluzi za promjenu tarife koristeći posrednik poruka. Na ovaj način, usluga poslužitelja ostaje dostupna i nema potrebe da korisnik čeka na obradu pojedinih zadataka, već se sve odvija u pozadini zahvaljujući posredniku poruka.

Nakon što korisnik odabere tarifu i pokrene zahtjev za njenom aktivacijom, usluga operatera pohranjuje zahtjev unutar MongoDB baze podataka te šalje zahtjev prema izmjenjivaču poruka kojeg koristi RabbitMQ prilikom usmjeravanja poruka. Izmjenjivač unutar posrednika poruka koristi fanout način usmjeravanja poruka te šalje poruke prema svim redovima s kojima je povezan. U konkretnom slučaju, izmjenjivač posrednika poruka šalje poruke prema redovima koji su spojeni s uslugom za SMS obavijesti i s uslugom za aktivaciju tarife. Nakon što usluga operatera pošalje poruku prema izmjenjivaču, ostale dvije usluge mogu preuzeti poruke sa svojih redova. Svaka usluga tada dobiva potrebne informacije za obavljanje svojih zadataka. Usluga za slanje SMS obavijesti preuzima poruku s reda *tariff.confirmations* te obavlja funkciju slanja SMS obavijesti na korisnikov broj. S druge strane, usluga za promjenu tarife preuzima istu (kopiju) poruke s reda *tariff.changing* te odrađuje svoj zadatak. Usluga za promjenu tarife nakon preuzimanja poruke mijenja stanje zahtjeva aktivacije tarife nakon određenog vremena te navedenu promjenu pohranjuje unutar baze podataka.

Koristeći RabbitMQ, sustavu je osigurana asinkrona komunikacija, sigurna isporuka poruka, usluge su nezavisnije te se izbjegava nepotrebno čekanje. Također, sustav može povećavati ili smanjiti broj usluga na jednostavniji način koristeći prednosti posrednika poruka. U slučaju prestanka rada neke od usluga sustava, sustav se neće narušiti, već će posrednik poruka pričekati da određena usluga bude ponovno dostupna i tada poslati pripremljenu poruku.

RabbitMQ posrednik poruka koristi algoritam kružnog dodjeljivanja (engl. *Round Robin)* poruka prilikom isporuke poruka prema uslugama. Sve tri usluge koriste metodu slanja ACK poruke (engl. *acknowledge)* prema RabbitMQ posredniku poruka. ACK poruka služi kako bi se izbjeglo ponovno slanje istog zahtjeva u red i kako bi RabbitMQ znao da je usluga obavila svoj zadatak. Također, postavljeno je ograničenje prilikom preuzimanja poruka s reda za uslugu promjene tarife. Kako usluga za promjenu tarife ne bi bila preopterećena brojnim zahtjevima korisnika, usluga preuzima samo dvije poruke s reda, dok ostale poruke čekaju na svoje preuzimanje.

### 3.7.1 Razvoj koda za uspostavljanje veze s RabbitMQ poslužiteljem

Sve tri usluge sadrže datoteku *rabbitMQ.js* unutar koje je implementiran kod za povezivanje usluge s RabbitMQ poslužiteljem. Pristup datotekama je moguć preko poveznice za Github platformu gdje je javno objavljen repozitorij sustava za aktivaciju tarife. Github poveznica je navedena u popisu literature. U nastavku će biti opisana *rabbitMQ.js* datoteka za svaku pojedinu uslugu.

providerService > src > JS rabbitMQ.js > ...

```
1   const amqp = require("amqplib");
2
3   const { logger } = require('./logging')
4
5   const HOST = process.env.HOST || 'localhost';
6   const URL = `amqp://${HOST}:5672`;
7   const EXCHANGE = "tariff";
8   let channel = null;
9
10  const rabbitConnection = async () => {
11      try {
12          const connection = await amqp.connect(URL);
13          channel = await connection.createChannel();
14
15          await channel.assertExchange(EXCHANGE, 'fanout', {
16              durable: false
17          });
18
19          logger.info(`AMQP - uspostavljena veza s ${URL}`)
20
21      }
22      catch (ex) {
23          logger.log('fatal',`greška - ${ex}`);
24          process.exit();
25      }
26  }
```

*Slika 3.16. Kod unutar datoteke rabbitMQ.js za povezivanje usluge operatera s posrednikom poruka*

Usluga operatera ima funkciju pošiljatelja poruka (engl. *Publisher)* prema ostalim uslugama. Ona ne sadrži red jer ne prima poruke, već se samo uspostavlja veza s RabbitMQ poslužiteljem i njegovim izmjenjivačem kako bi se omogućilo slanje poruka.

Kako bi se omogućilo povezivanje s RabbitMQ poslužiteljem, potrebno je učitati knjižicu *amqplib* koja sadrži unaprijed definirane funkcije za uspostavljanje veze. Također, određene su konstante koje će se koristiti kao parametri određenih funkcija prilikom povezivanja usluge s RabbitMQ-om. Za korištenje RabbitMQ poslužitelja, koristit će se lokalni poslužitelj na portu 5627. Izmjenjivač je nazvan *tariff.*

Kod za uspostavljanje veze s RabbitMQ poslužiteljem je napisan unutar try…catch bloka. U slučaju pogreške prilikom izvođenja koda unutar try bloka, ispisat će se pogreška. Kreiran je objekt *connection* te je pozvana funkcija *amqp.connect()* koja omogućuje stvaranje virtualne TCP veze između usluge i izmjenjivača posrednika poruka. Pomoću naredbe *createChannel()* stvoren je komunikacijski kanal unutar TCP komunikacijskog protokola. Stvaranjem kanala omogućen je prijenos poruka između RabbitMQ poslužitelja i usluge.

Nakon uspostavljanja veze s RabbitMQ poslužiteljem, potrebno je definirati izmjenjivač koji će se koristiti prilikom usmjeravanja poruka. Svaki izmjenjivač mora biti deklariran prije svoje uporabe. Izmjenjivač će preuzimati poruke od pošiljatelja te će ih usmjeravati prema svim redovima s kojima je povezan. Izmjenjivač je kreiran pomoću funkcije *assertExchange()* koja kao parametre prima naziv izmjenjivača te željeni način usmjeravanja poruke. Također, može se upravljati i ostalim postavkama izmjenjivača kao što je u ovom slučaju trajnost (engl. *durability)* koja je postavljena na vrijednost *false* . To znači da će se izmjenjivač prilikom ponovnog podizanja posrednika poruka ponovno kreirati.

Nakon što je uspješno uspostavljena veza s RabbitMQ poslužiteljem, usluga može slati poruke prema izmjenjivaču. Slanje zahtjeva za aktivaciju tarife se obavlja pomoću deklarirane funkcije *publishTariff*. Funkcija *publish()* koja se poziva nad objektom *channel* kao argumente prima naziv izmjenjivača kojem se šalje poruka, ključ za usmjeravanje te poruku koja se šalje. U ovom slučaju nije naveden ključ za usmjeravanje jer se koristi fanout način usmjeravanja poruka pa je puštena prazna vrijednost. Nakon što su informacije o aktivaciji tarife uspješno poslane prema izmjenjivaču, ispisuje se poruka s identifikacijskim brojem tarife koja je u obradi. Također, deklarirana je funkcija *expressService* koja služi kao posrednik za dohvaćanje unesenih objekata unutar zahtjeva, točnije informacija o aktivaciji tarife koje korisnik unosi preko Postman alata.

```
27
28  const publishTariff = (tariff) => {
29      channel.publish(EXCHANGE,'', Buffer.from(JSON.stringify(tariff)));
30      logger.info(`AMQP - zahtjev ${tariff._id} je na čekanju`);
31  }
32
33  const expressService = (req, res, next) => {
34      const exchangeServices = {
35          publishTariff: publishTariff
36      }
37      req.exchangeServices = exchangeServices;
38      next();
39  }
40
41  module.exports = {
42      expressService: expressService,
43      rabbitConnection: rabbitConnection
44  }
45
```

*Slika 3.17. Nastavak koda unutar datoteke rabbitMQ.js za povezivanje operatera s posrednikom poruka*

Za razliku od usluge operatera koja ima ulogu pošiljatelja poruka (engl. *Publisher)* prema RabbitMQ serveru, ostale dvije usluge se ponašaju kao primatelji poruka (engl. *Consumer)*. S obzirom na sličnost implementacije uspostavljanja veze s RabbitMQ poslužiteljem, objašnjenje će se temeljiti na jednoj usluzi, dok će razlike biti posebno navedene. Povezivanje s RabbitMQ poslužiteljem će biti objašnjeno na temelju usluge za slanje SMS obavijesti.

```
SMSservice > src > JS rabbitMQ.js > [@] rabbitConnection
1   const amqp = require("amqplib");
2
3   const { sendConfirmation } = require('./smsOptions')
4   const { logger } = require('./logging')
5   const {EXCHANGE, QUEUE} = require('./data.js');
6
7   const MESSAGE_COUNT = parseInt(process.env.MESSAGE_COUNT) || 2;
8   const HOST = process.env.HOST || 'localhost';
9   const URL = `amqp://${HOST}:5672`;
10  let channel = null;
11
12  const rabbitConnection = async () => {
13      try {
14
15          // uspostavljanje veze s RabbitMQ serverom
16
17          const connection = await amqp.connect(URL);
18          logger.info(`Uspješno uspostavljena AMQP veza s lokalnim RabbitMQ serverom ${URL}`)
19
20          // otvaranje kanala za komunikaciju s RabbitMQ serverom
21
22          channel = await connection.createChannel();
23
24          //spajanje s izmjenjivačem, odabir modela usmjeravanja, isključena opcija trajnosti izmjenjivača
25
26          await channel.assertExchange(EXCHANGE, 'fanout', {
27              durable: false
28          });
```

*3.18. Prikaz dijela koda unutar datoteke rabbitMQ.js za povezivanje SMS usluge s posrednikom poruka*

Kao i unutar datoteke operatera, u zaglavlju se učitava potrebna knjižica *amqplib* za uspostavljanje veze s RabbitMQ poslužiteljem te se definiraju konstante koje će se koristiti kao parametri prilikom stvaranja veze. Jedina razlika u zaglavlju je konstanta koja postavlja ograničenje na broj poruka koje će SMS usluga preuzeti s reda.

Unutar funkcije *rabbitConnection* implementirano je povezivanje s RabbitMQ poslužiteljem. Funkcije za uspostavljanje veze, kreiranje komunikacijskog kanala i povezivanje s izmjenjivačem poruka su identične kao i kod usluge operatera. Konstanta EXCHANGE za naziv izmjenjivača i konstanta QUEUE za naziv reda, definirane su unutar datoteke *data.js.*

Razlika koja se javlja unutar koda primatelja poruka prilikom povezivanja s RabbitMQ poslužiteljem je kreiranje reda i povezivanje s redom u koji se šalju poruke. Na slici 3.19. je vidljiv dio koda gdje je implementiran red za primanje isporučenih poruka iz izmjenjivača.

```
28          });
29
30
31          // povezivanje s redom, ako red ne postoji, kreiraj novi
32
33          await channel.assertQueue(QUEUE);
34          await channel.bindQueue(QUEUE, EXCHANGE, '');
35
36          //postavljanje ograničenja preuzimanje poruka s reda
37
38          channel.prefetch(MESSAGE_COUNT);
39
40          //obradivanje podataka s reda
41
42          channel.consume(QUEUE, order => {
43              sendConfirmation(order, channel);
44          });
45
46      }
47      catch (ex) {
48          logger.log('fatal',`AMQP - ${ex}`);
49          process.exit();
50      }
51  }
52
53  module.exports = {
54      rabbitConnection: rabbitConnection
55  }
56
```

*Slika 3.19. Nastavak koda unutar datoteke rabbitMQ.js za povezivanje SMS usluge s posrednikom poruka*

Funkcija *channel.assertQueue()* prima naziv reda kao argument te se koristi prilikom kreiranja reda za primanje poruka. Nakon stvaranja reda, poziva se funkcija *channel.bindQueue()* koja uspostavlja vezu sa stvorenim redom naziva QUEUE. QUEUE je konstanta koja je definirana unutar datoteke *data.js.*

SMS usluga je povezana s redom naziva *tariff.confirmations*, dok usluga za aktivaciju tarife koristi red naziva *tarrif.changing*. Dodatno, postavljeno je ograničenje na broj poruka koje usluge mogu primiti i obraditi. Ograničenje je postavljeno pomoću funkcije *channel.prefetch()* koja kao argument prima konstantu MESSAGE_COUNT kojom se određuje broj poruka koje će usluga primiti s reda.

Nakon što je uspješno uspostavljena veza s RabbitMQ poslužiteljem i kreiran je red za primanje poruka, usluga može preuzimati poruke s reda koje je poslao primatelj. Koristeći funkciju *channel.consume()* usluga čeka na preuzimanje poruke. Kada poruka pristigne u red, usluga preuzima poruku i poziva se definirana funkcija *sendConfirmation()* za slanje SMS obavijesti. Prethodno navedena funkcija je definirana unutar datoteke *SMSoptions.js*.

Za razliku od SMS usluge, usluga za promjenu tarife preuzima poruku s reda *tariff.changing* te nakon određenog vremena mijenja stanje zahtjeva aktivacije iz vrijednosti ACCEPTED u vrijednost CHANGED i sprema novu vrijednost unutar baze podataka. Vrijednosti su unaprijed deklarirane unutar datoteke *data.js* te se odnose na stanje zahtjeva aktivacije.

```
9
10  const changeTariff = (info, channel) => {
11      infoData = JSON.parse(info.content.toString());
12      changeStatus(infoModel, infoData._id, ACCEPTED);
13      setTimeout(() => {
14          changeStatus(infoModel, infoData._id, CHANGED);
15          channel.ack(info);   Slanje ACK potvrde
16      }, ORDER_DELIVERY_TIME);
17  }
18
```

*Slika 3.20. Prikaz funkcije changeTariff unutar koje se poziva funkcija za slanje ACK potvrde*

Nakon što su obje usluge obavile svoj dio zadatka, šalje se ACK poruka posredniku poruka kao potvrda o uspješnoj obradi poruke. Slanje ACK poruke se vrši koristeći funkciju *ack()* koja se poziva nad objektom komunikacijskog kanala i kao argument prima poruku koja je obrađena. Ovakav način potvrde sprječava ponovno slanje iste obrađene poruke unutar reda.

```
13  const sendConfirmation = (tariff, Channel) => {
14      tariffDetailes = JSON.parse(tariff.content.toString());
15      smsOptions.body += `Zahtjev za aktivacijom tarife je prihvaćen. Odabrana tarifa će biti aktivirana. \n
16      Broj zahtjeva: ${tariffDetailes._id}`
17      smsOptions.to = tariffDetailes.phone_number;
18      sendSMS(smsOptions, (error, info) => {
19          if (error) {
20              logger.log('crit',`sms - neuspješno slanje potvrde na telefonski broj ${tariffDetailes.phone_number} za aktivaciju tarife. i
21          } else {
22              logger.info(`sms - uspješno slanje potvrde na telefonski broj ${tariffDetailes.phone_number} za aktivaciju tarife. id_zahtje
23              Channel.ack(tariff);
24          }      slanje ACK potvrde
25      })
26
27  }
```

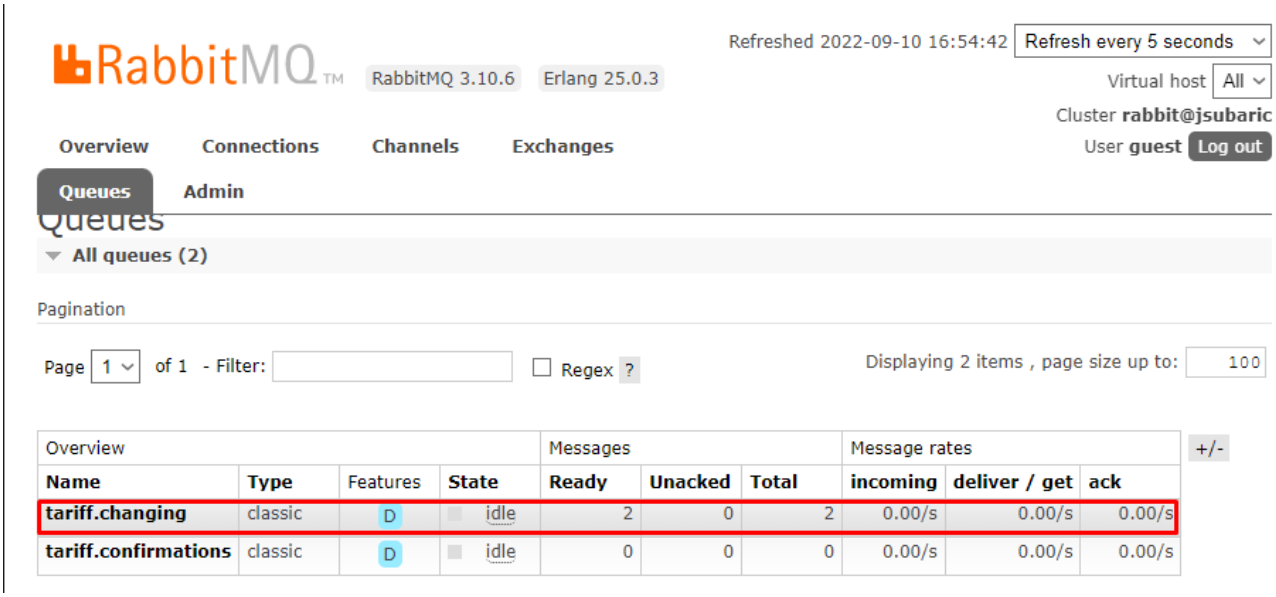
*Slika 3.21. Prikaz funkcije sendSMS unutar koje se poziva funkcija za slanje ACK potvrde*

### 3.8. Testiranje sustava

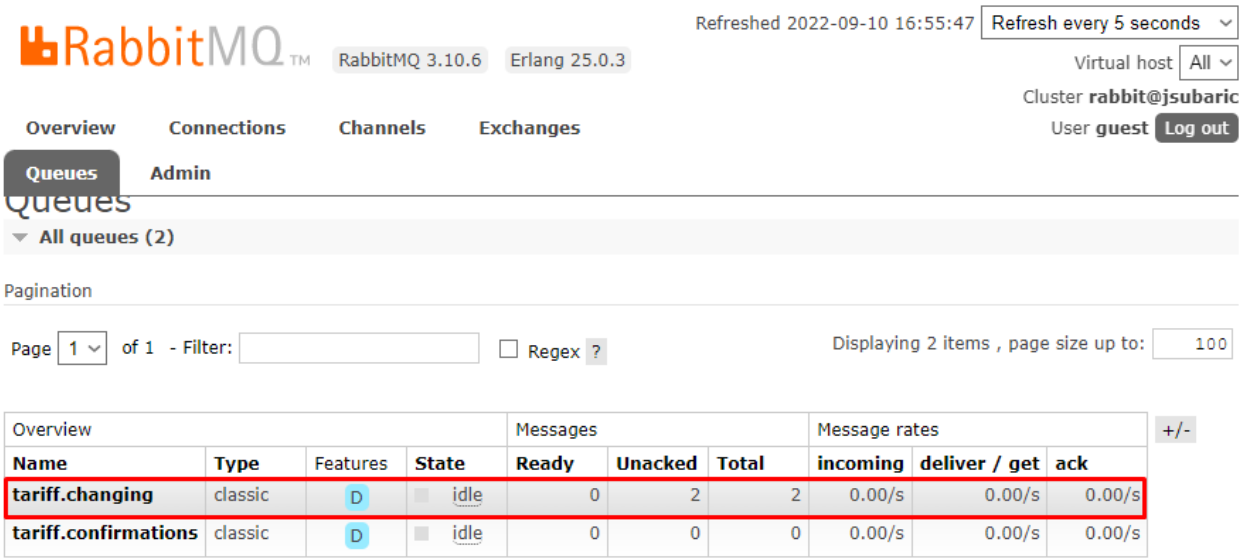
Kako bi se prikazala konkretna uloga posrednika poruka unutar sustava, namjerno će biti izazvana greška unutar sustava gašenjem jedne od usluga. Prilikom testiranja rada posrednika poruka biti će ugašena usluga za aktivaciju tarife te će bilježiti ponašanje sustava u tom slučaju.

Korisnik će provjeriti dostupnost tarifa, odabrati tarifu i pokrenuti njenu aktivaciju koristeći Postman alat. Nakon što je korisnik pokrenuo aktivaciju tarife, a usluga za aktivaciju tarife nije dostupna, status zahtjeva aktivacije će biti "na čekanju". Zahvaljujući posredniku poruka koji će preuzeti poruku i poslati je na red za aktivaciju tarife, promjena će biti izvršena kada usluga aktivacije tarife bude ponovno dostupna. Sustav neće zahtijevati čekanje da usluga postane dostupna, već će obrađivati i buduće zahtjeve koji će se slati u posrednik poruka. Na ovaj način omogućena je asinkrona komunikacija i izbjegavanje čekanja zbog nedostupnosti sustava koji treba obraditi poruku. U slučaju da SMS usluga nije dostupna, opisan proces vrijedit će i za nju te će SMS poruka zasigurno biti isporučena.

U konkretnom primjeru će se isključiti usluga za aktivaciju tarife, a istovremeno će biti poslana dva zahtjeva za aktivaciju tarife. Zahtjevi će čekati u redu dok usluga ponovno ne postane dostupna. Nakon određenog vremena, pokrenut će se usluga za aktivaciju tarife te će oba zahtjeva biti preuzeta s reda i obrađena.




*Slika 3.22. Prikaz reda za aktivaciju tarife koji čeka na dostupnost usluge kako bi isporučio poruke*




*Slika 3.23. Prikaz reda za aktivaciju tarife nakon uključenja usluge i preuzimanja poruka s reda*

```
PM - info: Stanje zahtjeva 631ca80b9a73cb1fb081c5d2: Aktivacija tarife prihvaćena.
PM - info: Stanje zahtjeva 631ca80c9a73cb1fb081c5d5: Aktivacija tarife prihvaćena.
PM - info: Stanje zahtjeva 631ca80c9a73cb1fb081c5d5: Tarifa je aktivirana
PM - info: Stanje zahtjeva 631ca80c9a73cb1fb081c5d5: Tarifa je aktivirana
```

*Slika 3.24. Ispis na konzoli nakon što je usluga za aktivaciju tarife uspješno obradila oba zahtjeva*

# 4. ZAKLJUČAK

Tema ovog završnog rada je bila opis posrednika poruka kao komunikacijskog medija koji se danas koristi među različitim sustavima, aplikacijama i uslugama. Opisane su brojne koristi koje posrednik poruka omogućuje imajući ulogu komunikacijskog sučelja unutar sustava te su navedeni konkretni primjeri uporabe. Kako bi se uvidjela razlika, uspoređena je implementacija sustava bez posrednika poruka i implementacija sustava s posrednikom poruka. Može se uočiti da je u nekim primjerima, posebice prilikom uspostave asinkrone komunikacije, korisnije koristiti posrednik poruka nego ostale standardne načine uspostave komunikacije.

Naglasak je bio na implementaciji sustava za aktivaciju mobilne tarife kako bi se opisala konkretna uloga i shvatio način rada posrednika poruka unutar sustava koji integrira tri različite usluge. Rad na implementaciji sustava s posrednikom poruka je omogućio šire shvaćanje funkcioniranja samog posrednika poruka unutar sustava. Korištenje RabbitMQ posrednika poruka otvorenog koda s velikim brojem korisne korisničke dokumentacije i s vlastitim upravljačkim sučeljem, omogućilo je lakše praćenje isporuke poruka i razumijevanje samog rada navedenog posrednika poruka. Implementacija posrednika poruka unutar sustava je značajno unaprijedila značajke sustava i omogućila njegovo neometano funkcioniranje. Zahvaljujući posredniku poruka, korisnici mogu koristiti sustav bez obzira na dostupnost pojedinih usluga pri tome znajući da će se njihov zahtjev sa sigurnošću obraditi. Također, asinkrona komunikacija između pojedinih usluga sustava isključuje potrebu nepotrebnog čekanja na obradu zahtjeva ili primanja poruka od druge strane.

Naposlijetku, može se zaključiti da je uvođenje posrednika poruka kao komunikacijskog sučelja u svijetu računarstva uveliko olakšalo uspostavljanje komunikacijske arhitekture unutar različitih sustava. S obzirom da su današnje aplikacije i sustavi sve složeniji i naglasak je na poboljšanju kvalitete korisničkog iskustva, bez posrednika poruka sve to ne bi bilo moguće. Također, radom na implementaciji sustava s posrednikom poruka, prošireno je moje dosadašnje znanje na području računalnih mreža.

# LITERATURA


[1] IBM Cloud Education: “What are Message Brokers?“, s Interneta, https://www.ibm.com/cloud/learn/message-brokers, 25. srpnja 2022.

[2] Amazon Web Services: “Benefits of Message Queues“, s interneta, https://aws.amazon.com/message-queue/benefits, 25. srpnja 2022.

[3] Rangan, K.: “What Is a Message Broker? How It Helps You Scale for Success“, s Interneta, https://www.g2.com/articles/message-broker, 3. kolovoza 2022.

[4] Jena, M.: “ Message Brokers: Key Models, Use Cases & Tools Simplified 101“, s Interneta, https://hevodata.com/learn/message-brokers/, 5. kolovoza 2022.

[5] Muench, J.: “Why Do We Need Message Broker?“, s Interneta, https://betterprogramming.pub/why-do-we-need-message-broker-7382ce0e46c6, 5. kolovoza 2022.

[6] Ali, J.: “ What Is A Message Broker: Advantages and Disadvantages“, s Interneta, https://networkustad.com/2022/06/25/what-is-a-message-broker-advantages-and-disadvantages/, 10. kolovoza 2022.

[7] Armand, S.: “ What are some use cases for message queues in real life, and what are some architectures that achieve the same goal without using queues explicitly?“, s Interneta, https://www.quora.com/What-are-some-use-cases-for-message-queues-in-real-life-and-what-are-some-architectures-that-achieve-the-same-goal-without-using-queues-explicitly, 10. kolovoza 2022.

[8] Łuczak, B.: “ 5 use cases of message brokers. When you should consider adopting a message broker in your system?“, s Interneta, https://tsh.io/blog/message-broker/, 10. kolovoza 2022.

[9] Johansson, L.: “ RabbitMQ for beginners - What is RabbitMQ?, s Interneta, https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html, 17. kolovoza 2022.

[10] Wikipedia: “Apache Kafka“, s Interneta, https://en.wikipedia.org/wiki/Apache_Kafka, 17. kolovoza 2022.

[11] Wikipedia: “ Amazon Simple Notification Service“, s Interneta, https://en.wikipedia.org/wiki/Amazon_Simple_Notification_Service, 17. kolovoza 2022.

[12] Wikipedia: “Amazon Simple Queue Service“, s Interneta, https://en.wikipedia.org/wiki/Amazon_Simple_Queue_Service, 17. kolovoza 2022.

[13] Wikipedia: “Redis“, s Interneta, https://en.wikipedia.org/wiki/Redis, 17. kolovoza 2022.

[14] Erlang: “OTP Versions Tree“, s Interneta, http://erlang.org/download/otp_versions_tree.html, 24. kolovoza 2022.

[15] RabbitMQ: “Installing on Windows Manually“, s Interneta, https://www.rabbitmq.com/install-windows-manual.html, 24. kolovoza 2022.

[16] video sadržaj, s Interneta, https://www.youtube.com/watch?v=wS1ZAzYqChE, 24. kolovoza 2022.

[17] RabbitMQ: “ AMQP 0-9-1 Model Explained“, s Interneta, https://www.rabbitmq.com/tutorials/amqp-concepts.html, 24. kolovoza 2022.

[18] RabbitMQ : “Tutorial for Javascript“, s Interneta, https://www.rabbitmq.com/tutorials/tutorial-four-javascript.html, 24. kolovoza 2022.

[19] Storti, B.: “ Speaking Rabbit: A look into AMQP's frame structure“, s Interneta, https://www.brianstorti.com/speaking-rabbit-amqps-frame-structure/#:~:text=A%20frame%20is%20AMQP%20's,your%20application%20and%20vice-versa, 27. kolovoza 2022.

[20] Github repozitorij s projektnim datotekama, s Interneta, https://github.com/jansubaric/MessageBroker

## SAŽETAK

U ovom završnom radu nastoji se opisati i prezentirati rad posrednika poruka. Navedena je definicija posrednika poruka, razlog njihovog uvođenja i objašnjene su glavne karakteristike posrednika. Također, navedene su prednosti i nedostaci sustava s posrednikom poruka i sustava bez posrednika poruka. U radu se opisuju konkretni primjeri uporabe posrednika poruka te se navodi popis aktualnih posrednika poruka. Uz napisanu dokumentaciju, u sklopu završnog rada implementiran je sustav s RabbitMQ posrednikom poruka otvorenog koda kako bi se opisala konkretna uloga posrednika poruka. Izrada i instalacija sustava je detaljno objašnjena unutar dokumentacije te su navedene osnovne funkcije sustava i provedeno je testiranje. Implementirani sustav je napravljen za svrhe učenja i namijenjen je svima.



Ključne riječi: Posrednik poruka, sustav s posrednikom poruka, RabbitMQ



## ABSTRACT

The purpose of this paperwork is to describe and present the work of message brokers. The definition of message brokers, the reason for their introduction, and the main characteristics of brokers are explained. Also, the advantages and disadvantages of systems with a message broker and systems without a message broker are listed. The paper describes concrete examples of the use of message brokers and provides a list of current message brokers. In addition to the written documentation, as part of the final work, a system with RabbitMQ, an open source message broker, was implemented to describe the specific role of the message broker. The creation and installation of the system is explained in detail in the documentation, and the basic functions of the system are listed and testing is carried out. The implemented system is made for learning purposes and is intended for everyone.



Key words: Message broker, system with a message broker, RabbitMQ