|  |  |  |
| --- | --- | --- |
|  | **UNIVERZITET U NOVOM SADU**  **FAKULTET TEHNIČKIH NAUKA U NOVOM SADU** |  |

Kristijan Salaji

**IMPLEMENTACIJA MEHANIZMA ZA RAZMENU PODATAKA PO UGLEDU NA *APACHE* *KAFKA* ARHITEKTURU**

MASTER RAD

- Master akademske studije-

Novi Sad, 2019

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ ⚫ **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | | | |
| **КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА** | | | |
| Редни број, **РБР**: | | |  | |
| Идентификациони број, **ИБР**: | | |  | |
| Тип документације, **ТД**: | | | Монографска публикација | |
| Тип записа, **ТЗ**: | | | Текстуални штампани документ/ ЦД | |
| Врста рада, **ВР**: | | | Мастер рад | |
| Аутор, **АУ**: | | | Кристијан Салаји | |
| Ментор, **МН**: | | | Проф. Др Бранислав Атлагић, Доцент | |
| Наслов рада, **НР**: | | | Имплементација механизма за размену података на основу *Kafka* решења. | |
| Језик публикације, **ЈП**: | | | Српски (латиница) | |
| Језик извода, **ЈИ**: | | | Српски/енглески | |
| Земља публиковања, **ЗП**: | | | Србија | |
| Уже географско подручје, **УГП**: | | | Војводина | |
| Година, **ГО**: | | | 2019 | |
| Издавач, **ИЗ**: | | | Ауторски репринт | |
| Место и адреса, **МА**: | | | Факултет техничких наука (ФТН), Д. Обрадовића 6, 21000 Нови Сад | |
| Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога) | | | 9/18/0/0/23/0/0 | |
| Научна област, **НО**: | | | Електротехничко и рачунарско инжењерство | |
| Научна дисциплина, **НД**: | | | Примењене рачунарске науке и информатика | |
| Предметна одредница/Кључне речи, **ПО**: | | | *Producer, Consumer, Broker* | |
| **УДК** | | |  | |
| Чува се, **ЧУ**: | | | Библиотека ФТН, Д. Обрадовића 6, 21000 Нови Сад | |
| Важна напомена, **ВН**: | | |  | |
| Извод, **ИЗ**: | | | Са појавом све већих система у којима се размењују велике количине информација, потребно је те информације на ефикасан начин дистрибуирати кроз систем. *Publish/Subscribe* механизам омогућава дистрибуцију информација до ентитета који су за те информације заинтересовани. Не врши се дистрибуција информација до свих ентитета у систему и тиме се смањује оптерећење на систем и повећавају се перформансе. У овом раду ће бити имплементиран *Publish/Subscribe* механизам који врши филтрирање порука на основу теме. | |
| Датум прихватања теме, **ДП**: | | |  | |
| Датум одбране, **ДО**: | | |  | |
| Чланови комисије, **КО**: | | Председник: | Проф. Др Владимир Павлица, Доцент |
|  | | Члан: | Проф. Др Дарко Чапко, Ванредни Професор | Потпис ментора |
|  | | Члан, ментор: | Проф. Др Бранислав Атлагић, Доцент |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | UNIVERSITY OF NOVI SAD ⚫ **FACULTY OF TECHNICAL SCIENCES**  21000 NOVI SAD, Trg Dositeja Obradovića 6 | | | |
| **KEY WORDS DOCUMENTATION** | | | |
| Accession number, **ANO**: | | |  | |
| Identification number, **INO**: | | |  | |
| Document type, **DT**: | | | Monographic publication | |
| Type of record, **TR**: | | | Textual material, printed/CD | |
| Contents code, **CC**: | | | Master thesis | |
| Author, **AU**: | | | Kristijan Salaji | |
| Mentor, **MN**: | | | Prof. Dr Branislav Atlagić, Assistant Professor | |
| Title, **TI**: | | | Implementation of data sharing mechanism based on Kafka solution. | |
| Language of text, **LT**: | | | Serbian | |
| Language of abstract, **LA**: | | | Serbian/English | |
| Country of publication, **CP**: | | | Serbia | |
| Locality of publication, **LP**: | | | Vojvodina | |
| Publication year, **PY**: | | | 2019 | |
| Publisher, **PB**: | | | Author’s reprint | |
| Publication place, **PP**: | | | Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | | 9/18/0/0/23/0/0 | |
| Scientific field, **SF**: | | | Electrical and computer engineering | |
| Scientific discipline, **SD**: | | | Applied computer science and informatics | |
| Subject/Key words, **S**/**KW**: | | | Producer, Consumer, Broker | |
| **UC** | | |  | |
| Holding data, **HD**: | | | Library of Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad | |
| Note, **N**: | | |  | |
| Abstract, **AB**: | | | With the emergence of increasingly large systems in which large amounts of information are exchanged, it is necessary to distribute this information efficiently through the system. The Publish/Subscribe mechanism allows distribution of information to entities that are interested in this information. No information is distributed to all entities in the system, thereby reducing the burden on the system and increasing performance. In this paper, the Publish/Subscribe mechanism will be implemented to filter messages based on the topic. | |
| Accepted by the Scientific Board on, **ASB**: | | |  | |
| Defended on, **DE**: | | |  | |
| Defended Board, **DB**: | | President: | Prof. Dr Vladimir Pavlica, Assistant Professor |
|  | | Member: | Prof. Dr Darko Čapko, Assistant Professor | Menthor's sign |
|  | | Member, Mentor: | Prof. Dr Branislav Atlagić, Assistant Professor |  |

|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ ⚫ **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Датум: |
|  |
| **ЗАДАТАК ЗА ИЗРАДУ МАСТЕР РАДА** | Лист/Листова: |
| 1/1 |

*(Податке уноси предметни наставник - ментор)*

| Студијски програм: | Примењено софтверско инжењерство |
| --- | --- |
| Руководилац студијског програма: | Др Драган Поповић |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | Кристијан Салаји | Број индекса: | E5 3 2018 |
| Област: | Индустријски комуникациони протоколи у електроенергетским системима | | |
| Ментор: | Др Бранислав Атлагић | | |
| НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА  ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:   * проблем – тема рада; * начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; * литература | | | |

**НАСЛОВ ДИПЛОМСКОГ-МАСТЕР РАДА:**

|  |
| --- |
| Имплементација механизма за размену података на основу *Kafka* решења |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| Потребно је развити *Publish/Subscribe* компоненту за потребе репликационе инфаструктуре критичних система. Пре имплементације потребно је написати тестове. Компоненту је потребно развити у С# програмском језику и потврдити тестовима исправност и пропусност. |

|  |  |
| --- | --- |
| Руководилац студијског програма: | Ментор рада: |
|  |  |

|  |
| --- |
| Примерак за:  - Студента;  - Ментора |

**SPISAK KORIŠĆENIH SKRAĆENICA**

|  |  |
| --- | --- |
| **Skraćenica** | **Pun naziv** |
| MOM | Message Oriented Middleware |
| AMQP | Advanced Message Queuing Protocol |
| CLR | Common Language Runtime |
| API | Aplication Programming Interface |
| CIL | Common Itermediate Language |
| JIT | Just In-Time Compiler |
| IDE | Integrated Development Environment |
| GUI | Graphical User Interface |
| TDD | Test-Driven Development |
| WCF | Windows Communication Foundation |

Sadržaj

[1. Uvod 1](#_Toc21605370)

[1.1. Distribuirani sistemi 1](#_Toc21605371)

[*1.2.* *Apache Kafka* 3](#_Toc21605372)

[1.2.1. Kafka teme i particije 3](#_Toc21605373)

[*1.2.2.* *Kafka* proizvođač 4](#_Toc21605374)

[*1.2.3.* *Kafka* potrošač 4](#_Toc21605375)

[1.2.4. *Kafka* grupe potrošača 4](#_Toc21605376)

[*1.2.5.* *Kafka* posrednik 5](#_Toc21605377)

[*1.2.6.* *Kafka* klaster 5](#_Toc21605378)

[2. Teorijske osnove 6](#_Toc21605379)

[2.1. Replikacioni servis 6](#_Toc21605380)

[*2.2.* *Failover* 6](#_Toc21605381)

[3. Opis korišćenih tehnologija i alata 7](#_Toc21605382)

[*3.1.* *.NET framework* 7](#_Toc21605383)

[3.2. C# programski jezik 7](#_Toc21605384)

[*3.3.* *Microsoft Visual Studio* 7](#_Toc21605385)

[*3.4.* *NUnit* 8](#_Toc21605386)

[*3.5.* *Open Cover* 8](#_Toc21605387)

[3.6. *Windows Communication Foundation* (WCF) 8](#_Toc21605388)

[3.7. Generički tipovi podataka 9](#_Toc21605389)

[*3.8.* *Callback Handler* 10](#_Toc21605390)

[4. Implementirano programsko rešenje 11](#_Toc21605391)

[4.1. Arhitektura realizovanog rešenja 11](#_Toc21605392)

[4.2. Implementacija realizovanog rešenja 12](#_Toc21605393)

[*4.2.1.* Proizvođač (*Producer*) 12](#_Toc21605394)

[*4.2.2.* Potrošač (*Consumer*) 13](#_Toc21605395)

[*4.2.3.* Menadžer (*Manager*) 14](#_Toc21605396)

[*4.2.4.* Posrednik (*Broker*) 16](#_Toc21605397)

[4.2.5. Replikacioni servise (*Replication Service*) 20](#_Toc21605398)

[4.3. Struktura podataka 22](#_Toc21605399)

[5. Dijagram sekvence 23](#_Toc21605400)

[5.1. Sinhrono slanje podataka 23](#_Toc21605401)

[5.2. Asinhrono slanje podataka 23](#_Toc21605402)

[5.3. Replikacija podataka 24](#_Toc21605403)

[5.4. Poravnanje podataka (*Integrity Update*) 25](#_Toc21605404)

[6. Performanse 26](#_Toc21605405)

[6.1. Performanse kod sinhronog slanja podataka 26](#_Toc21605406)

[6.2. Performanse kod asinhronog slanja podataka 27](#_Toc21605407)

[7. Zaključak 28](#_Toc21605408)

[8. Literatura 29](#_Toc21605409)

[9. Podaci o kandidatu 30](#_Toc21605410)

# Uvod

U ovom poglavlju će biti predstavljeni distribuirani sistemi i *Apache Kafka* arhitektura. Poglavlje 2 sadrži teorijske osnove kako bi bolje razumeli čitav zadatak. Tehnologije i alati korišćeni prilikom izrade zadatka su navedeni i opisani u poglavlju 3. Arhitektura i implementacija rešenja je detaljno opisana u poglavlju 4. U poglavlju 5 je prikazan dijagram sekvenci izvršavanja. Performanse rešenja su prikazane u poglavlju 6. Zaključak kao i dalji razvoj rešenja opisan je u poglavlju 7. Literatura koja je upotrebljena prilikom pisanja rada navedena je u poglavlju 8. Poglavlje 9 sadrži podatke o kandidatu.

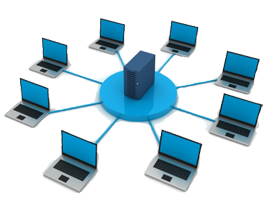
## Distribuirani sistemi

Distribuirani sistem jeste skup nezavisnih računara koje korisnici vide kao jedan koherentan sistem [1]. Svi ti računari, povezani računarskom mrežom, rade zajedno u cilju izvršavanja zajedničkog zadatka. Računari mogu biti geografski dislocirani i to je jedna od stvari koja utiče na kompleksnost distribuiranih sistema.

Razlozi za razvoj distribuiranih sistema su sledeći [1]:

* Jednostavnije pristupanje resursima – jedan od glavnih ciljeva distribuiranih sistema jeste da obezbedi korisnicima lak pristup udaljenim resursima,
* Transparentnost – važno je sakriti činjenicu da se procesi i resursi nalaze na više distribuiranih mašina,
* Otvorenost – distribuirani sistem izlaže svoje servise po unapred definisanim standardima i pravilima
* Skalabilnost – dodavanje novih korisnika i resursa u sistem.

Distribuirani sistemi se po arhitekturi dele na centralizovane, decentralizovane i hibridne [1]. Centralizovana arhitektura implementira većinu softverskih komponenti i funkcionalnosti, dok klijenti mogu pristupati serveru pomoću komunikacione mreže. Kod decentralizovane arhitekture, čvorovi imaju podjednaku ulogu u sistemu. Hibridna arhitektura je spoj centralizovane i decentralizovane arhitekture.



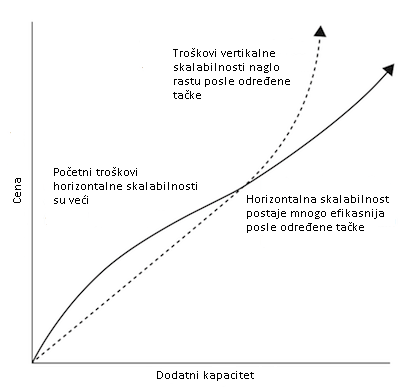
Slika 1.1 – Centralizovan distribuirani sistem [17]

Slika 1.1 prikazuje centralizovani distribuirani sistem. Većina današnjih distribuiranih sistema je centralizovana [3]. Decentralizovani sistem ne može biti posedovan od strane jedne kompanije, inače viče ne bi bio decentralizovan [3]. Kada uzmemo tu činjenicu u obzir, dolazimo do zaključka da je decentralizovane sisteme mnogo teže kreirati i održavati.

Prednost distribuiranih sistema u odnosu na sisteme sa jednim serverom su sledeće [3]:

* Jedini način da se kod sistema sa jednim serverom poveća produktivnost je da se unapređuje računar na kojoj se aplikacija izvršava dodavanjem RAM, procesora, itd., što se nazivavertikalno skaliranje.
* Kod distribuiranih sistema povećanje produktivnosti se može postići tako što se u mrežu postojećih računara dodaju novi računari (horizontalno skaliranje)

Smanjenje performansi [3] kod horizontalnog skaliranja se rešava dodavanjem nove mašine. Nove mašine se mogu dodavati do određenog limita u zavisnosti od implementacije sistema. Loše skaliranje sistema dovodi do smanjenih mogućnosti proširenja kapaciteta sistema. Jednostavno skaliranje nije jedina prednost koju poseduju distribuirani sistemi.



Slika 1.2 - Razlika troškova vertikalne i horizontalne skalabilnosti [3]

U početnoj fazi, kada potrebe za kapacitetom nisu velike, isplativija je vertikalna skalabilnost (Slika 1.2). Potrebe za dodatnim kapacitetom vremenom postaju veće te u jednom trenutku vertikalna skalabilnost ne može to da isprati. Horizontalna skalabilnost postaje isplativija kako potrebe za kapacitetom postaju sve veće [3].

Jedan od izazova prilikom razvoja distribuiranih sistema jeste replikacija podataka (2.1) i konzistentnost istih. Neki od glavnih razloga za replikaciju podataka jesu pouzdanost i unapređenje performansi [1]. Podaci se repliciraju na više mesta kako bi se povećala sigurnost sistema. Ukoliko dođe do problema sa jednom replikom, sistem nastavlja sa radom tako što se prebaci na drugu repliku. Performanse sistema se mogu povećavati replikacijom, na primer, tako što se mogu čitati sa više mesta istovremeno.

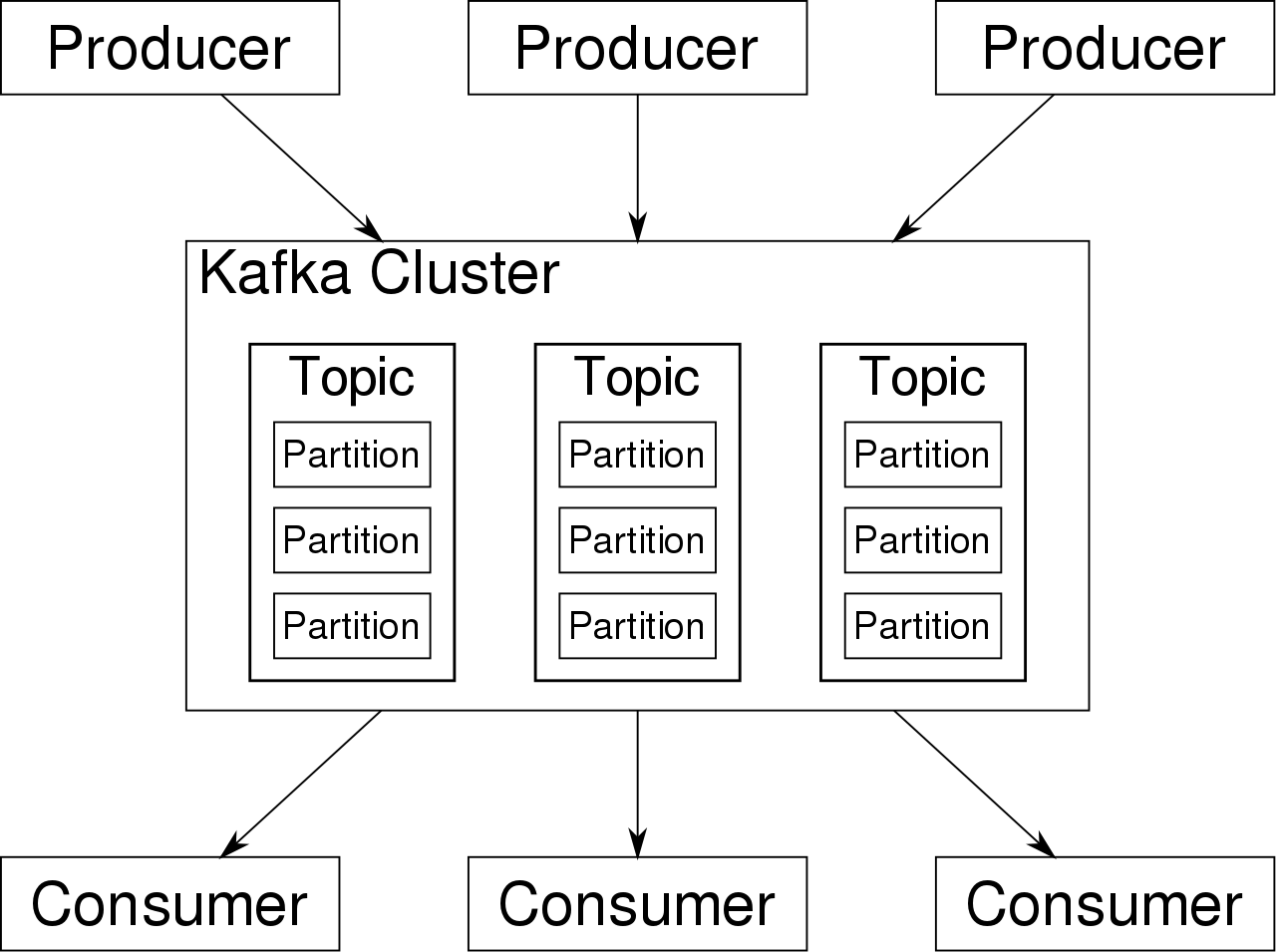
Kada kažemo da su podaci konzistentni smatramo da se u bazu podataka upisuju samo validni podaci [9]. Tada kažemo da su podaci poravnati. Ako u bazu pokušamo da upišemo nevalidne podatke, cela transakcija će se odbaciti i baza podataka će se vratiti u prethodno konzistentno stanje [9].

## *Apache Kafka*

U ovom poglavlju ćemo detaljnije opisati *Apache* *Kafka* arhitekturu [5]. *Apache Kafka* je softverska platforma za obradu podataka koja ima za cilj da obezbedi prenos podataka u realnom vremenu, sa visokom tačnošću i minimalnim kašnjenjem [2]. *Apache Kafka* je zasnovana na dnevniku objava (*Commit Log*), koji omogućava korisnicima da se pretplate na neku od objava, ili da sami objave podatke dostupne bilo kom od računara u Kafka sistemu i aplikacijama koje rade u realnom vremenu [2].

Softversko rešenje izloženo u radu je bazirano na *Apache Kafka* arhitekturi (Slika 1.2). Razvijeni programski sistem za distribuciju poruka u celosti je saglasan *Apache* *Kafka* modelu.

Kao što je već pomenuto u prethodnom poglavlju, količina podataka koja se razmenjuje između uređaja raste [10]. Jedan od mehanizama za razmenu poruka između čvorova distribuiranog sistema predstavljaju sistemi za razmenu poruka – *Message Orientated Middleware* (MOM). Postoje različite implementacije MOM sistema. Njihova uloga je da obezbede razmenu podataka između različitih komponenti u sistemu. Mnogi *Message Oriented Middleware-i* implementiraju klasičan *Publish/Subscribe* šablon i nisu specijalno dizajnirani za razmenu ogromne količine *Stream* podataka [4]. Većina pomenutih *Middleware-a* sadrži posrednika koji ima izložen *Advanced Message Queuing Protocol* (AMQP) za asinhronu komunikaciju [4].



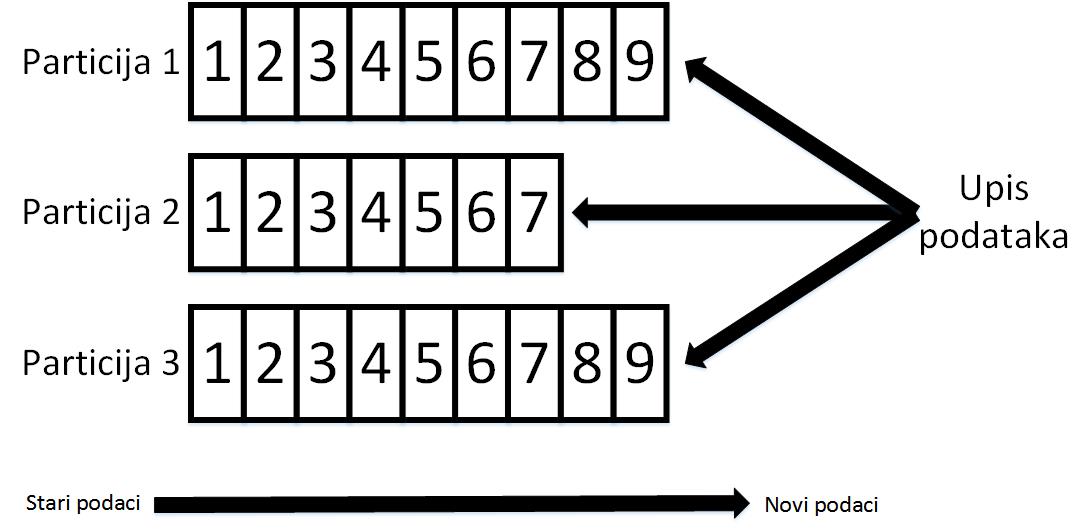
Slika 1.3 ' Kafka arhitektura [2]

Pojednostavljena *Kafka* arhitektura se sastoji od klastera (*Cluster*), proizvođača (*Producer*) ipotrošača (*Consumer*).. *Kafka* klaster se sastoji od više posrednika (*Broker*). Proizvođač šalje podatke posredniku koji iste skladišti u bazu podataka. Potrošač šalje zahteve posredniku kada je spreman da prihvati i obradi potrebne podatke.

### Kafka teme i particije

U okviru *Kafka* arhitekture, svaki podatak objavljen od strane bilo kog proizvođača vezan je za određenu temu (*Topic*) [5]. Podaci vezani za jednu temu se čuvaju na više particija koje mogu da se nalaze kod različitih posrednika [5]. Potrošači mogu slati zahteve kako bi dobili podatke vezane za jednu ili više tema. Proizvođači takođe mogu da objavljuju podatke na jednu ili više tema. Svi ti podaci se čuvaju u particijama koja se nalazi kod posrednika. Podaci vezani za jednu temu se čuvaju na više particija koje mogu da se nalaze kod različitih posrednika[5].

Jedna od glavnih uloga particije jeste da se ubrza rukovanje podacima, uvođenjem paralelizma u radu. Podaci se istovremeno upisuju na više particija, kao što se mogu i čitati [4].



Slika 1.4 - Kafka particije

Slika 1.4 prikazuje kako se podaci upisuju na particije. Može da postoji veći broj particija, u zavisnosti od kapaciteta kojim raspolaže server. Podaci se istovremeno upisuju u sve particije, što u mnogome ubrzava čitav proces rukovanja podacima [5]. Upisivanje u particije se vrši sekvencijalno. To znači da se podaci upisuju redom kojim pristižu. Podaci upisani u particije se nazivaju zapisi (*Records*). Svakom zapisu se dodeljuje pozicija (*Offset*)koji jedinstveno identifikuje zapis u određenoj particiji [5].

### *Kafka* proizvođač

U *Kafka* arhitekturi proizvođač se poklapa sa izdavačem (*Publisher*) u *Publish/Subscribe* arhitekturi. Proizvođač predstavlja izvor podataka koji se upotrebljavaju u celokupnom *Kafka* sistemu. Podaci se objavljuju na postojeće teme i kasnije čuvaju u particijama u sklopu posrednika.

### *Kafka* potrošač

Potrošači čitaju podatke sa željenih tema koji se nalaze u sklopu posrednika. Glavna razlika koja se uvodi u *Kafka* arhitekturi jeste ta da potrošači neće automatski dobijati podatke sa tema koji su im od interesa. Kada su spremni da obrade podatke, potrošači će sami poslati zahtev za podacima [4]. Takođe potrošač mora da vodi računa o pozicija kursora koji govori do koje pozicije u particije je stigao sa čitanjem podataka. Pozicija kursora se naziva *Offset*. Potrošač ima mogućnost da čita podatke od trenutka kada je poslao zahtev. Takođe ima mogućnost da čita podatke od početka particije ili da pošalje zahtev za podacima koje nije uspešno obradio. Svaki potrošač pripada najmanje jednoj grupi potrošača (1.2.4).

### *Kafka* grupe potrošača

Grupa potrošača (*Consumer Groups*) se sastoji od jednog ili više potrošača. U većini slučajeva je jedna grupa potrošača vezana za jednu temu. Svakom potrošaču u grupi dodeljuje se jedna ili više particija određene teme [4]. Ova podela potrošača u grupe omogućava paralelizam u radu i ubrzava proces dobavljanja i obrade potrebnih podataka.



Slika 1.5 ' Kafka consumer grupe [5]

Slika 1.5 prikazuje kako se dobavljanje podataka paralelizuje preko rada u grupama. P0, P1, P2 i P3 predstavljaju particije koje pripadaju temama. Grupa A ima dva potrošača a grupa B četiri. Potrošačima iz grupe A je dodeljena po jedna particija iz svake teme što znači da svaki potrošač čita podatke iz dve particije. Pošto grupa B ima četiri potrošača, svakom potrošaču je dodeljena tačna jedna particija. Tako će grupa B duplo brže pribavljati podatke od grupe A.

### *Kafka* posrednik

Posrednik (*Broker*) se nalazi u okviru *Kafka* klastera. Posrednik može da vodi računa o više particija [4]. Primarni zadatak posrednika jeste da prima podatke, pretvara ih u zapise kojima dodeljuje pozicijui nakon toga zapise smešta u particije. Particije vezane za jednu temu mogu biti raspoređene na nekoliko posrednika [4].

Posrednici mogu da imaju primarnu i sekundarnu ulogu. Posrednik koji ima primarnu ulogu prima pristigle podatke i obrađuje iste. Nakon procesa obrade i smeštanja zapisa u particije, primarni posrednik šalje posredniku koji imaju sekundarnu ulogu pristigle podatke. Posrednici sa sekundarnom ulogom takođe čuvaju podatke u sopstvene particije. Razmena podataka između posrednika se ostvaruje uz pomoć replikacionog sistema (2.1). Posrednici sa sekundarnom ulogom su u svakom trenutku spremni da preuzmu primarnu ulogu jer su podaci sačuvani u njihovim particijama isti kao podaci koji su sačuvani u particijama primarnog posrednika.

### *Kafka* klaster

*Kafka* klaster (*Cluster*)se sastoji od više posrednika. U okviru svakog klasterajedan posrednik se proglašava za upravljača (*Controller*). Njegova uloga jeste da dodeljuje particije ostalim posrednicima, obavlja administrativne operacije i vodi računa o otkazima preostalih posrednika [4].

Ukoliko dođe do otkaza posrednika koji ima ulogu kontrolera, drugi posrednik preuzima njegovu ulogu. Ukoliko otkaže posrednik koji nije kontroler, particije koje su njemu bile dodeljene se dodeljuju drugim posrednicima tako da opterećenje na posrednike bude podjednako.

# Teorijske osnove

U ovom poglavlju ćemo detaljno prikazati teorijske osnove neophodne za razumevanje implementiranog programskog rešenja.

## Replikacioni servis

Poravnavanje podataka na više instanci istog servisa se naziva replikacija podataka. Replikacijom podataka obezbeđuje se konzistentnost, povećava nivo sigurnosti sistema i smanjuje verovatnoća gubitka podataka usled pada sistema.

Replikacioni servis omogućava jednostavno kopiranje podataka na jednu ili više lokacija. Za replikacioni servis su vezani partnerski replikacioni servisi i komponenta koja replicira ili dobija replikacione podatke.

Komponenta koja replicira podatke prosleđuje podatke replikacionom servisu. Replikacioni servis te podatke dalje šalje partnerskim replikacionim servisima bilo da ih je više ili samo jedan. Partnerski replikacioni servis takođe poseduje komponentu koja je za njega vezana i koja očekuje da primi replikacione podatke. Partner te podatke prosleđuje komponenti koja ih čuva u odgovarajućoj bazi podataka.

Ukoliko dođe do pada sistema, komponenta koja je do tog trenutka primala replikacione podatke može nastavi sa radom nesmetano jer poseduje sve potrebne podatke. Time se izbegava vreme otkaza sistema i povećava se sigurnost sistema.

## *Failover*

*Failover* predstavlja operativni režim rada gde sekundarna komponenta preuzima funkcije sistema kada primarna komponenta postane nedostupna zbog kvara ili unapred planiranih prekida rada [11].

Stanja u kojima se nalaze komponente sistema koji podržava *Failover*:

* *Hot* – komponenta koja se nalazi u *Hot* stanju jeste primarna komponenta i ona izvršava funkcije sistema u datom trenutku
* *StandBy* – sekundarna komponenta jeste komponenta koja se nalazi u *StandBy* stanju. Ova komponenta je u svakom trenutku u pripravnosti da preuzme funkcije sistema ukoliko primarna komponenta nije u mogućnosti da izvršava iste

Sistem koji podržava *Failover* jeste sistem čija je otpornost na greške na visokom nivou. *Failover* jeste sastavni deo sistema sa kritičnom misijom koji moraju biti dostupni u svakom trenutku [11]. Postupak prebacivanja funkcionalnosti na sekundarnu komponentu treba da bude što je više moguće neprimetan za krajnjeg korisnika [11].

Tehničko objašnjenje *Failover* režima [12]: između dve komponente, primarne (*Hot*) i sekundarne (*StandBy*), se kreira komunikacioni kanal. U većini slučajeva sekundarna komponenta proverava da li je primarna komponenta “živa“ njenim prozivanjem. Sve dok se prozivanje uspešno izvršava sekundarna komponenta neće preduzimati nikakve akcije. Kada primarna komponente ne odgovori na *ping* u predefinisanom vremenu, sekundarna komponenta će preuzeti funkcionalnosti sistema i obavestiće centar odgovoran za funkcionalnost koji će ponovo podići primarnu komponentu [12].

Kako se radi o distribuiranom sistemu, sekundarni čvor nikada ne može biti potpuno siguran da li je primarni čvor ugašen ili je jednostavno prekinuta veza između njih. Rešavanja ovog problema je van rešavanja ovog rada.

# Opis korišćenih tehnologija i alata

U ovom poglavlju će biti navedene tehnologije i alati koji su bili upotrebljeni prilikom razvoja rešenja, njihove karakteristike kao i prednosti.

## *.NET framework*

*.NET framework* služi za razvoj i izvršavanje aplikacija na *Windows* operativnim sistemima. *.NET framework* je deo *.NET* platforme, koja predstavlja kolekciju tehnologija za razvoj aplikacija na različitim operativnim sistemima. Komponente koje čine jezgro *.NET framewrok-a su* [6].

Komponente koje čine jezgro *.NET framewrok-a su* [6]:

* *Common Language Runtime* (CLR)
* *.NET Framewrok Class Library* (FCL)

*Common Language Runtime* omogućava izvršavanje aplikacija. Takođe pruža servise za upravljanje nitima, rukovanje izuzecima, uklanjanje nepotrebnih objekata iz memorije. *.NET* FCL predstavlja skup biblioteka koje olakšavaju korisnicima razvoj aplikacija. U pomenutoj biblioteci se nalaze predefinisani tipovi podataka, API (*Application Programming Interface*) za čitanje i pisanje u datoteke, povezivanje sa bazom podataka, crtanje, itd. [6].

*.NET* aplikacije koja se razvija u C# programskom jeziku (3.2) se prvo prevodi u CIL (*Commom Intermediate Language*). Tako preveden kod se čuva u datotekama sa nastavkom .dll ili .exe. Kada se aplikacija pokrene, CLR čita datoteku sa gore navedenim nastavkom i prevodi je u mašinski kod koji se može izvršavati na računaru. Ovaj proces se naziva neposredno prevođenje (*Just In-Time Compiling, JIT*) [6].

## C# programski jezik

C# (C *Sharp*) je objektno orjentisan programski jezik i čini sastavni deo *.NET framework-a*. Jedan je od mlađih programskih jezika i veoma je sličan Java programskom jeziku. C# kombinuje računarsku moć C++ sa lakoćom programiranja koju poseduje *Visual Basic* [7].

Koristeći C# prilikom razvoja programeri mogu da nadograde postojeći kod umesto da ga više puta dupliraju [7]. C# time ubrzava razvoj novih proizvoda i usluga .Jezik je opšte primene i poseduje visok nivo apstrakcije.

## *Microsoft Visual Studio*

Za razvoj projekta opisanog u ovom dokumentu korišćen je *Microsoft Visual Studio*, integrisano razvojno okruženje (IDE). *Visual Studio* ima široku upotrebu prilikom razvoja *web* aplikacija i *web* servisa, *web* sajtova, i drugih računarskih programa .

Karakteristike:

* Editor koda - mogućnost refaktorisanja koda
* *Debugger* – integrisan u okviru *Microsoft Visual Studio-a*. Alat za statičku analizu koda
* *Designer* – formi za kreiranje GUI (*Graphical User Interface*) aplikacija, šema baza podataka

U okviru *Visual Studio-a* se nalazi alat *Performance Profiler*. Pomoću njega se mogu meriti performanse sistema, zauzeće memorije, broj poziva funkcija, vreme provedeno u funkcijama i ostale korisne stvari koje mogu pomoći prilikom unapređenja performansi razvijene aplikacije.

## *NUnit*

Testiranje ispravnosti napisanog koda vrši se pomoću automatskih testova od kojih neki testiraju pojedinačne klase (*Unit* testovi) a drugi čitave komponente (*Component* testovi). *Unit* testovi predstavljaju manje celine koda napisane od strane testera ili programera. Pomoću ovih testova proveravaju se delovi koda koji imaju jedinstvenu funkcionalnost, kao što su jedna metoda ili funkcija

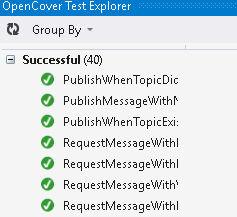
Razvoj softvera je tekao po TDD (*Test-Driven Development*) principu. Prvo su pisani testovi koji imaju određene zahteve. Nakon toga se piše programska logika sve dok svi zahtevi testova ne budu ispunjeni. Novi testovi mogu da se dodaju tek nakon što svu ispunjeni zahtevi prethodno napisanih testova.

Neke od prednosti automatskih *Unit i* komponentnih testova [7]:

* pokretanje testova može biti automatizovano i da se ponavlja nakon određenih izmena
* testovi su mnogo precizniji nego testiranje preko GUI

## *Open Cover*

Za merenje procenta pokrivenosti koda testovima koristi se alat pod nazivom *Open Cover*. *Open Cover* je besplatan alat koji može da se primenjuje na *.NET* 2.0 i novijim verzijama. Pomoću ovog alata možemo da pratimo pokrivenost grana kao i pokrivenost određenih sekvenci koda.



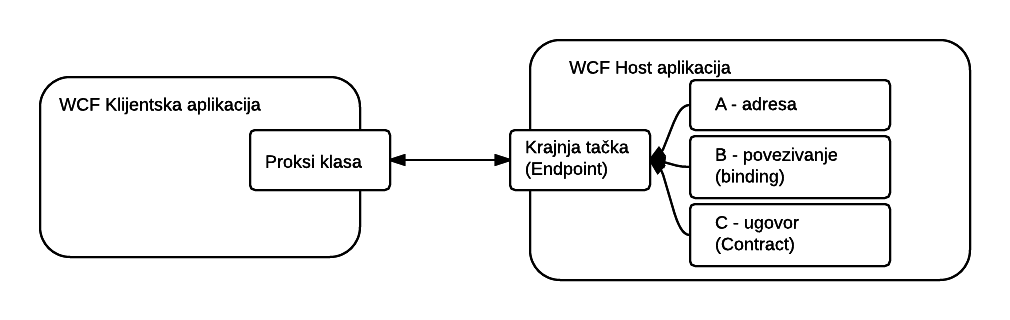
Slika 3.1 - OpenCover Test Explorer

*Open Cover* poseduje vizuelni prikaz (Slika 3.1) svih testova. U posebnim odeljcima se prikazuju testovi koji su uspešno izvršeni, oni koji nisu uspešno izvršeni i testovi koji nisu pokrenuti.

## *Windows Communication Foundation* (WCF)

*Windows Communication Foundation* (WCF) jeste model za razmenu podataka koji omogućava komunikaciju preko mreže ili lokalnu komunikaciju. WCF uključuje set biblioteka koje su razvijene za distribuirano programiranje [13].

WCF se upotrebljava prilikom razvoja servisno orjentisanih aplikacija. Sam dizajn podržava razvoj distribuiranih sistema, to jest da obezbedi komunikaciju između tačaka koje su geografski dislocirane. Karakteristika WCF-a jeste da su komponente labavo povezane [14].



Slika 3.2 - WCF dijagram [13]

Slika 3.2 prikazuje na koji način se ostvaruje komunikacija između klijenta i servera. Klijent kreira kanal prema serveru na osnovu adrese, povezivanja i ugovora. Adresa jeste URL pomoću kojeg se pronalazi krajnja tačka. Povezivanje određuje protokol koji se koristi. Ugovor ili interfejs sadrži informacije o funkcijama koje su klijentima na raspolaganju. Da bi klijent bio u mogućnosti da poziva funkcije sa servisa, potrebno je da bude povezan sa servisom preko protokola koji je određen krajnjom tačkom [13].

WCF omogućava nekoliko šablona za razmenu podataka [14]:

* *Request-Response* šablon– šablon koji se najviše koristi. Jedna krajnja tačka zahteva podatke od druge krajnje tačke. Druga krajnja tačka odgovara.
* *OneWay* šablon – jedna krajnja tačka šalje podatke i ne očekuje nikakav odgovor. Ovaj šablon se najčešće upotrebljava kod asinhrone komunikacije
* *Duplex Exchange* šablon – nešto složeniji šablon. Dve krajnje tačke uspostavljaju vezu i mogu da šalju podatke jedna drugoj.

## Generički tipovi podataka

Prilikom razvoja programskog rešenja u velikoj meri su se upotrebljavali generički tipovi podataka. U ovom delu ćemo detaljno objasniti šta su to generički tipovi podataka i koje su njihove prednosti.

Generički tipovi podataka su uvedeni u C# verziji 2.0. Oni omogućavaju da se definiše klasa sa rezervisanim mestom za tip polja, tip povratne vrednosti funkcije ili tip parametra u samoj funkciji [15]. Generički tipovi podataka se najviše upotrebljavaju kod metoda koje rukuju sa kolekcijama [16].

Generički tipovi podataka mogu da se upotrebljavaju kod: interfejsa (ugovora), apstraktnih klasa, klasa, metoda, statičkih metoda, polja, događaja, delegata, operatora [15].

Prednosti generičkih tipova podataka su sledeće [15]:

* Povećava ponovnu upotrebu koda
* Generički tipovi podataka su sigurni za tip (*Type Safe*). Ukoliko se pokuša korišćenje neodgovarajućeg tipa prijaviće se greška prilikom prevođenja.
* Stiče se prednost u performansama jer se uklanja potreba za pakovanjem (*Boxing*) i raspakivanjem (*Unboxing*) podataka

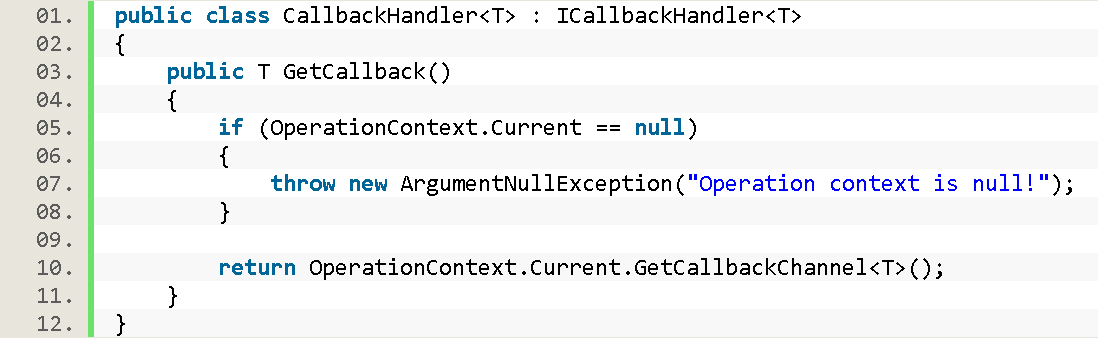
Upotreba generičkih tipova podataka može se videti u sledećem primeru. Listing 4:18 - *IReplicationClient* interfejs prikazuje jedan interfejs gde generički tip (u ovom slučaju R) predstavlja tip koji će se koristiti kao tip podataka koji će se replicirati.

## *Callback Handler*

U poglavlju 3.6 smo govorili o WCF-u i o načinu razmene podataka u okviru WCF-a. U ovom poglavlju ćemo detaljnije opisati dvostruku komunikaciju kao i šta je to *Callback* i kako se upotrebljava.

WCF prilikom kreiranja dvostrukog kanala ne omogućuje punu dvostruku komunikaciju. Kada se podaci šalju u jednom smeru nije moguće u istom trenutku slati podatke u suprotnom smeru. To je jedna od mana dvostrukog kanala kod WCF-a.

*Callback* je interfejs definisan kao atribut drugog interfejsa koji izlaže servis. *Callback* interfejs treba da implementira klijent koji ostvaruje vezu sa serverom. *Callback* se koristi kada server treba da obavesti klijenta da se desio neki događaj ili da isporuči klijentu neke podatke bez potrebe da klijent kreira *Host.*

**

Listing 3:1 - CallbackHandler klasa

Uz pomoć *Callback Handler-a* (Listing 3:1) izvlačimo klijenta koji je pozvao određenu metoda i na taj način možemo da ga obavestimo ili mu pošaljemo podatke. Konkretno, *Callback* sistem omogućuje da server poziva udaljene funkcije na klijentu.

Listing 4:6 - Implementacija *PublishSync* metodeprikazuje upotrebu *Callback Handler-a*. Izvlačimo *Callback* klijenta koji je pozvao *PublishAsync* metodu i obaveštavamo ga da li su podaci uspešno poslati.

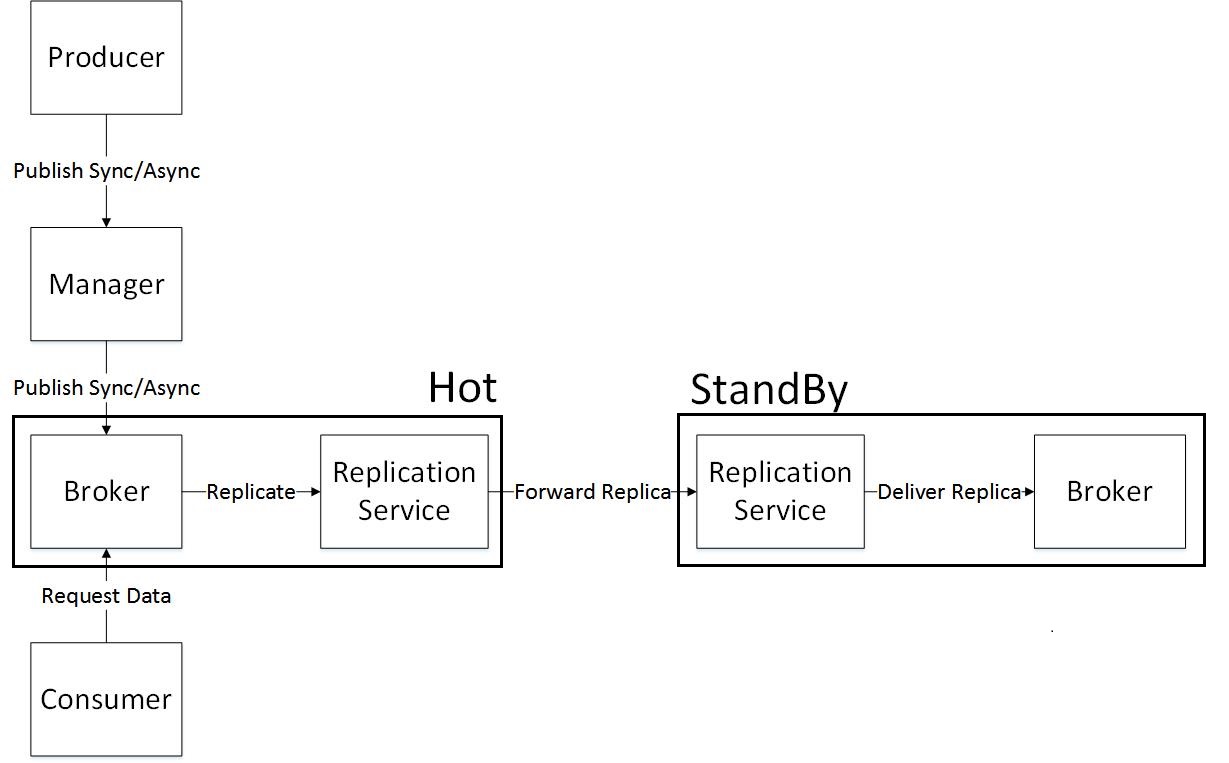
# Implementirano programsko rešenje

U ovom poglavlju prikazano je rešenje za razmenu velike količine podataka u distribuiranim sistemima kao i detaljan osvrt na implementirane funkcionalnosti programskog rešenja. Postoji mogućnost sinhronog i asinhronog slanja podataka od strane komponenti koje generišu podatke koji se kasnije koriste u sistemu. Tako generisani podaci se šalju ka komponenti koja ih skladišti u odgovarajućoj bazi podataka. Komponente kojima su ti podaci potrebni šalju zahteve za podacima komponentama koje su prethodno te podatke skladištile.

## Arhitektura realizovanog rešenja

Komponente koje čine implementirano rešenje (Slika 4.1):

* Proizvođač (*Producer*)(4.2.1)
* Menadžer (*Manager*)(4.2.3)
* Posrednik (*Broker*)(4.2.4)
* Potrošač (*Consumer*)(4.2.2)
* Replikacioni servis (*Replication Service*)(4.2.5)



Slika 4.1 - Arhitektura implementiranog rešenja

Slika 4.1 prikazuje arhitekturu implementiranog rešenja. Prethodno navedene komponente čine mehanizam za razmenu podataka po ugledu na *Kafka* rešenje. Na slici je prikazana *Hot* i *StandBy* strana sistema. *Hot* strana je aktivna strana i komponente koja se tu nalaza aktivno učestvuju u razmeni podataka. *StandBy* strana je strana koja prima replikacione podatke i uvek je u stanju pripravnosti da nastavi sa radom ukoliko dođe do problema na *Hot* strani.

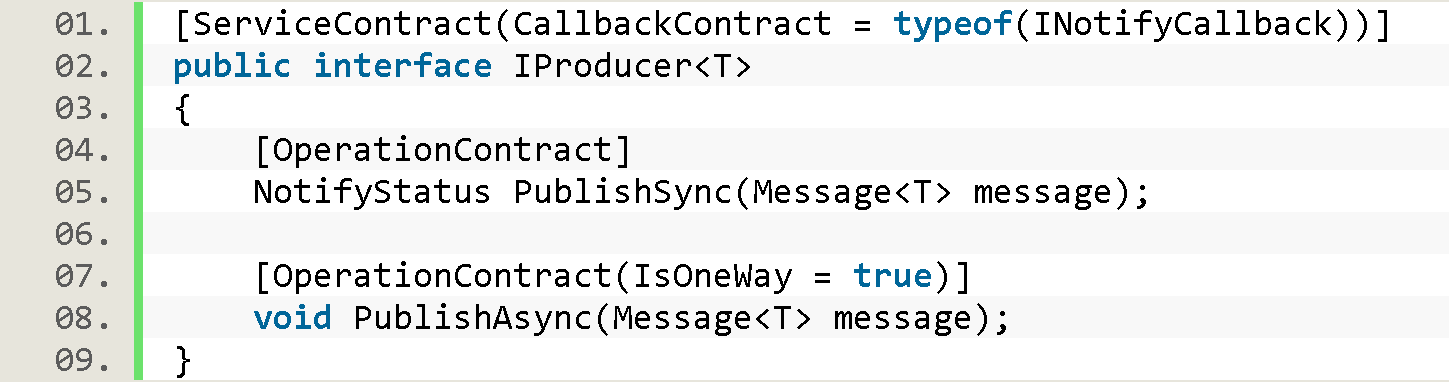
## Implementacija realizovanog rešenja

Detaljan opis implementacije rešenja biće predstavljen u ovom poglavlju. Svaka komponenta pomenuta u prethodnom poglavlju (4.1) će biti predstavljena pomoću izabranih delova implementacione logike.

### Proizvođač (*Producer*)

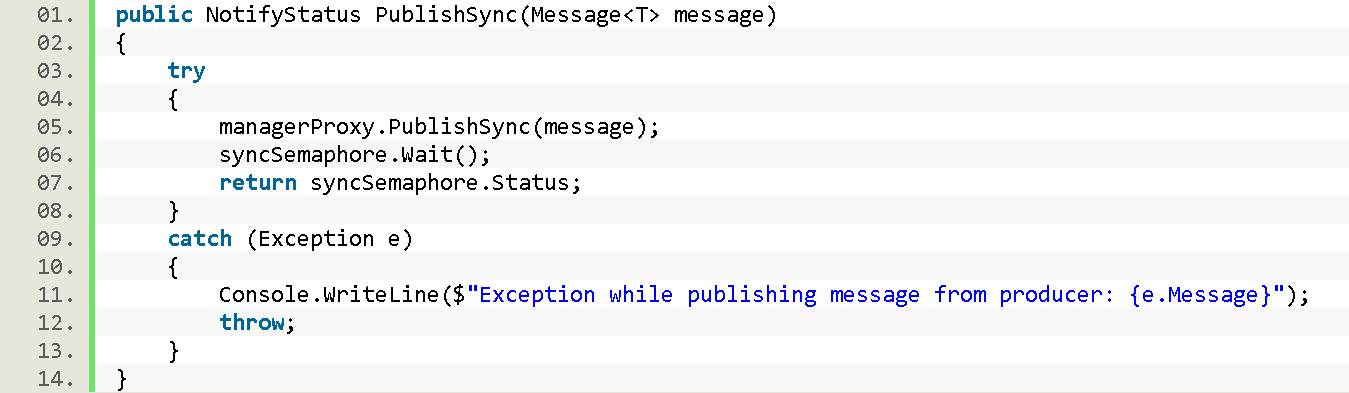
U okviru rešenja predstavljenog u ovom poglavlju proizvođač predstavlja glavni izvor podataka. Sama komponenta implementira interfejs *IProducer* (Listing 4.1). Pomenuti interfejs sadrži dve metode

* *PublishSync* – Pomoću ove metode se vrši sinhrona objava podataka i dobija se odgovor u obliku *NotifyStatus-a* koji nam govori da li su podaci uspešno isporučeni ili ne.
* *PublishAsync* – Asinhrona objava podataka. Ne čeka se odgovor i nakon poziva metode se nastavlja sa obavljanjem narednih funkcionalnosti



Listing 4.1 - IProducer interfejs

Proizvođač komponenta komunicira samo sa menadžer komponentom (4.2.3). U pozadini se otvara WCF (*Windows Communication Foundation*) kanal ka menadžer komponenti. Prilikom slanja podataka sinhronim putem (pozivom *PublishSync* metode) (Listing 4.2) pomoću prethodni kreiranog kanal se takođe poziva metoda *PublishSync* čija se implementacija nalazi na menadžer komponenti.



Listing 4.2 - PublishSync metoda

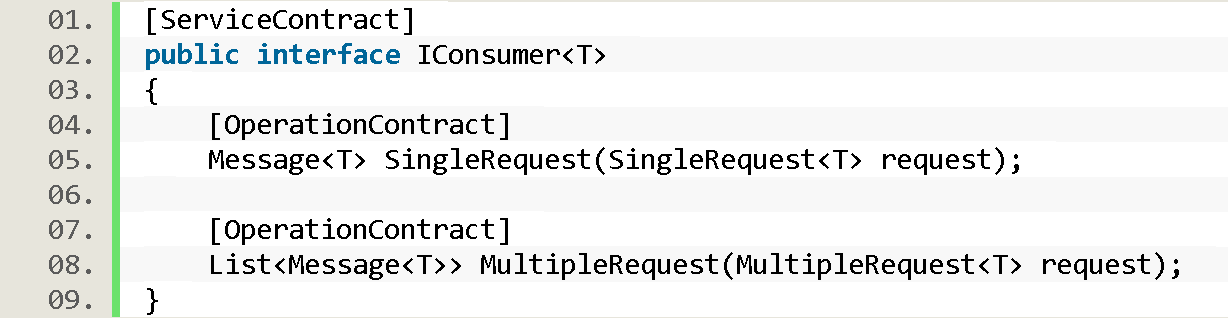
Nakon poziva metode *PublishSync* čeka se povratna informacija koja se dobija od menadžera koja nam govori da li je poruka uspešno poslata. Menadžer obaveštava proizvođača o statusu poruke pomoću *Callback* metode *Notify* koja otpušta semafor. Ta informacija se vraća kao povratna vrednost pozvane metode u vidu *NotifyStatus-a*. Na osnovu povratne vrednosti proizvođač odlučuje o daljim koracima izvršavanja u zavisnosti od njegove implementacije.

Što se tiče asinhronog slanja podataka, poziva se metoda *PublishAsync*. Pozivom ove metode podaci se prosleđuju menadžer komponenti i nastavlja se sa radom bez čekanja ikakvog odgovora ili potvrde od menadžera.

### Potrošač (*Consumer*)

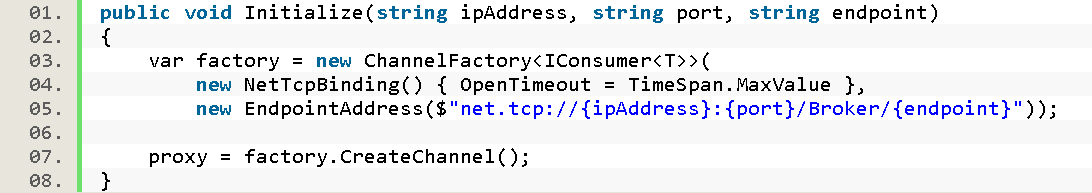
Funkcionalnost ove komponente jeste da šalje zahteve posredniku kada je spremna da obrađuje iste. Potrošač ima zadatak da vodi računa o poziciji, to jest da zna do koje pozicije u particiji je stigao sa zahtevima. Potrošač ima mogućnost da uputi zahtev za podacima na dva načina (Listing 4.3):

* *SingleRequset* – ovaj zahtev se šalje kada je potrebno dobaviti samo jedan podatak
* *MultipleRequest* – ovaj zahtev se šalje kada je potrebno dobaviti više podataka od jednom



Listing 4.3 - IConsumer interfejs

Potrošač komunicira samo sa posrednikom putem WCF-a (Listing 4.4). U pozadini se kreira kanal pomoću kojeg potrošač šalje zahteve za potrebnim podacima. Komunikacija između potrošača i menadžera je sinhrona jer je potrošaču neophodan odgovor u vidu zahtevanih podataka da bi mogao da nastavi sa izvršavanjem logike.



Listing 4.4 - Kreiranje kanala prema Broker servisu

Listing 4.4 opisuje kreiranje kanala prema servisu podignutom od strane menadžera. *IpAddress*, *Port* i *Endpoint* se čitaju iz *AppConfig-a* i prosleđuju se kao parametri funkcije *Initialize*. U funkciji *Initialize* se kreira kanal preko kog mogu da se pozivaju metode iz *IConsumer* interfejsa. Ove metode su implemetirane u okviru posrednika.

Ukoliko se kao povratna vrednost metoda *SingleRequest* i *MultipleRequest* dobije vrednost *null*, to je rezultat jednog od sledećih slučajeva:

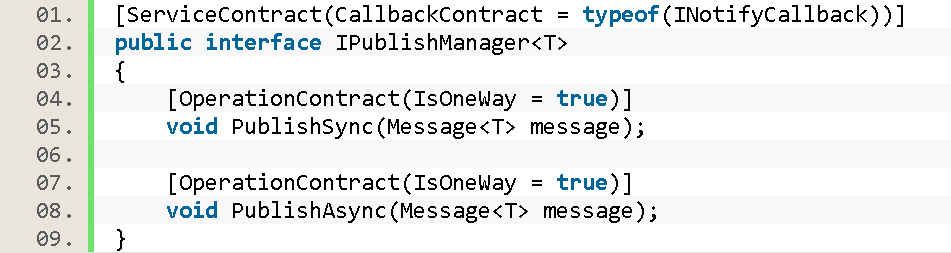
* Ne postoje podaci objavljeni na temu na koju je poslat zahtev
* Ne postoji tema na koju je poslat zahtev
* Zahtev nije dobro popunjen i nije prošao proveru
* Komunikacioni kanal nije ispravno kreiran

### Menadžer (*Manager*)

Menadžer komponenta predstavlja jednu vrstu mehanizma za isporuku podataka. Podaci primljeni od proizvođača se u zavisnosti od vrste slanja (sinhrono ili asinhrono) skladište u red za asinhrono slanje ili se direktno prosleđuju posredniku. Sa jedne strane menadžer ima podignut servis, prema kome proizvođač kreira komunikacioni kanal. Sa druge strane menadžer kreira komunikacioni kanal prema servisu koji je podignut u sklopu posrednika. Pomoću tog kanala menadžer isporučuje prethodno dobijene podatke.

Menadžer implementira metode interfejsa *IPublishManager*. U okviru ovog interfejsa se nalaze dve metode (Listing 4.5):

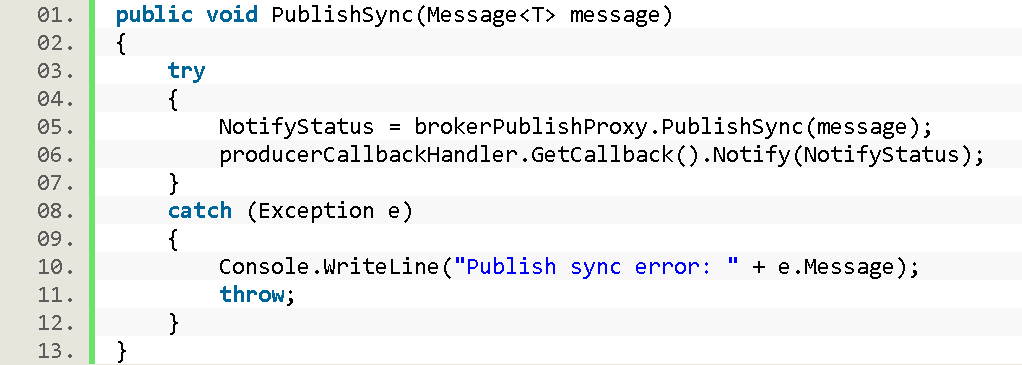
* *PublishSync* – sinhrono slanje podataka
* *PublishAsync* – asinhrono slanje podataka



Listing 4.5 - IPublishManager interfejs

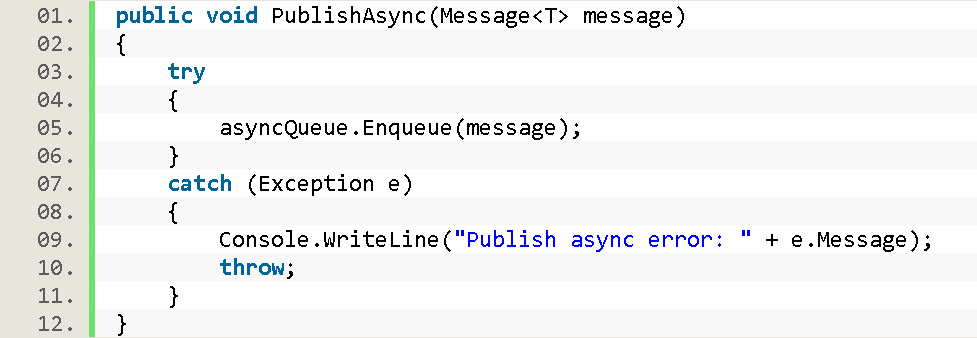
Obe metode koje pripadaju interfejsu *IPublishManager* imaju postavljenu vrednost atributa *IsOneWay* na *true*. To znači da se ne očekuje nikakva povratna vrednost i takve metode mogu biti samo tipa *void*.

Primarni zadatak menadžer komponente jeste da isporuči sve podatke koje dobije od proizvođača. Kada je u pitanju sinhrona komunikacija, pozivom metode *PublishSync* (Listing 4.6) dobijeni podaci se prosleđuju ka posredniku. Posrednik kao povratnu vrednost pozvane metode vraća status da li je operacija uspešno izvršena ili ne. Status se preko *Callback-a* prosleđuje proizvođaču koji na osnovu sopstvene implementacije preduzima odgovarajuće aktivnosti.



Listing 4.6 - Implementacija PublishAsync metode

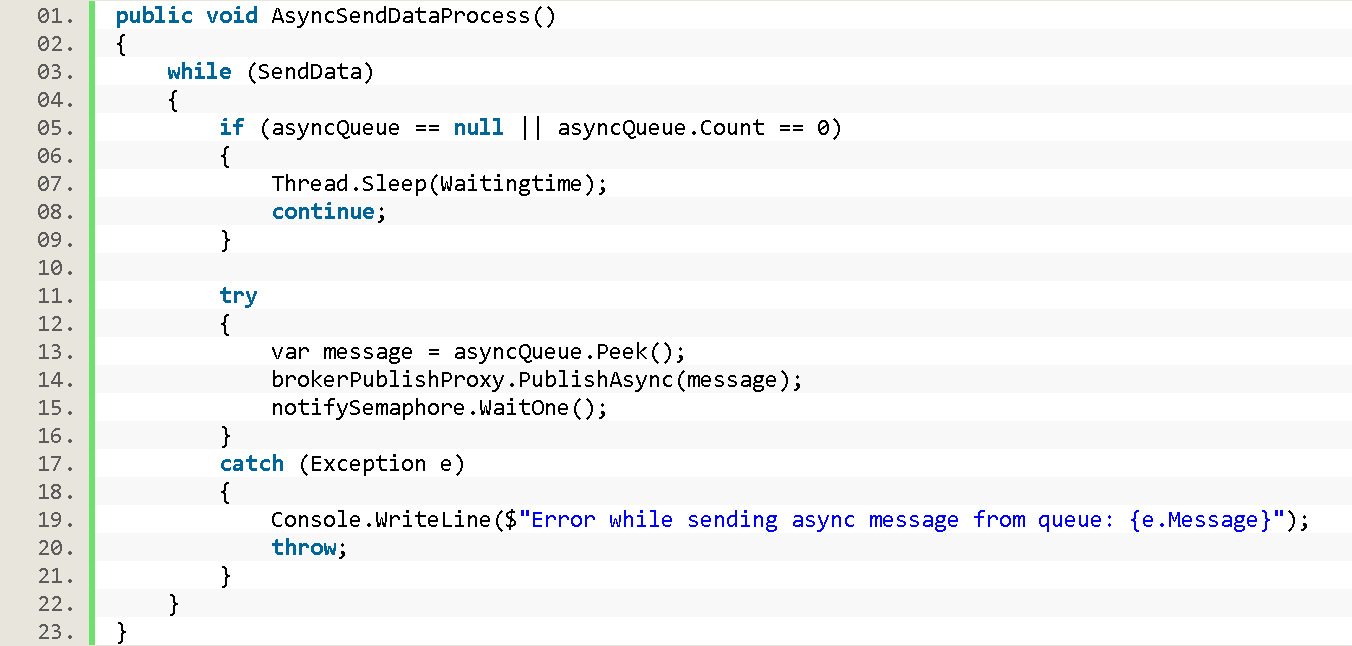
Kada je u pitanju asinhrona komunikacija, menadžer poseduje red u koji se skladište svi podaci primljeni od proizvođača kada proizvođač pošalje podatke putem *PublishAsync* metode.



Listing 4.7 - Implementacija metode PublishAsync

Listing 4.7 prikazuje implementaciju metode *PublishAsync*. Implementacija je veoma jednostavna jer se u okviru pomenute metode dobijeni podaci smeštaju u red za asinhronu komunikaciju (*asyncQueue*).

Prilikom pokretanja menadžera kreira se nova nit u okviru koje se izvršava metoda *AsyncSendDataProces* (Listing 4.8). U pozadini se proverava da li u redu postoje pristigli podaci. Ukoliko je red prazan, čeka se određeni period vremena i ponovo se vrši ista provera.

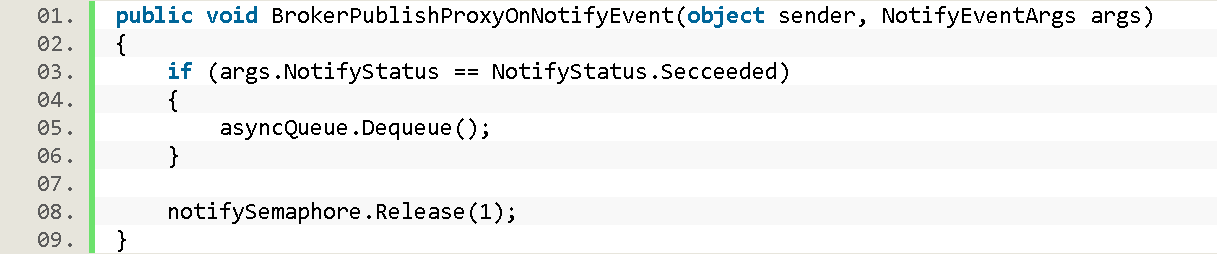


Listing 4.8 - Implementacija metode AsyncSenddataProcess

Ukoliko red za asinhronu komunikaciju nije prazan i postoje podaci, uzima se prvi podataka ali se ne uklanja iz reda. Taj podatak se šalje posredniku preko prethodno kreiranog komunikacionog kanala. Zatim se čeka na odgovor pomoću semafora *notifySemaphore*. Čekanje je neophodno jer menadžer mora da obavi svoju primarnu funkcionalnost a to je da isporuči svaku poruku koju je od proizvođača dobio.

Menadžer dobija informaciju da li je poslati podataka uspešno obrađen od posrednika tako što se pretplati na *NotifyEvent* događaj. Pomenuti događaj se nalazi u okviru prethodno kreiranog komunikacionog kanalaprema posredniku.

Listing 4.9 prikazuje implementaciju metode *BrokerPublishProxyOnNotifyEvent.* Ona se izvršava kada se okine *NotifyEvent* događaj.



Listing 4.9 - Metoda BrokerPublishProxyOnNotifyEvent

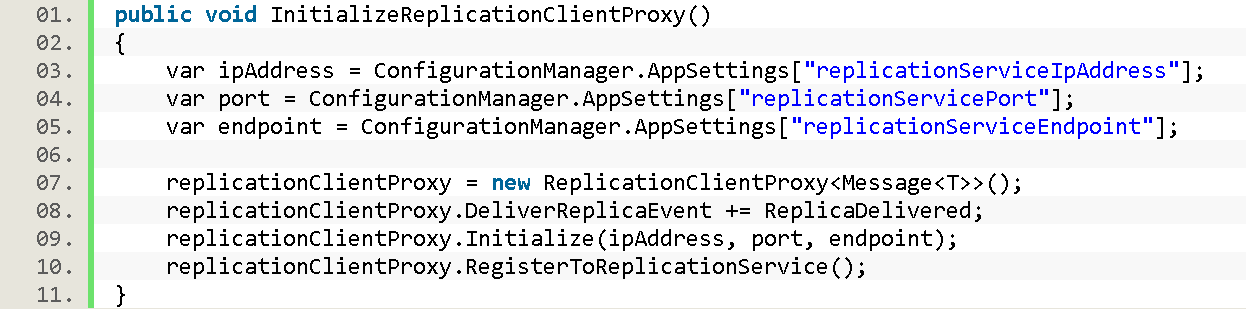
Preko parametra *NotifyEventArgs* se dobija informacija da li je prethodna poruka uspešno obrađena i sačuvana. Ta informacija se dobija u obliku *NotifyStatus-a* (*Succeeded* ili *Failed*). Ukoliko je poruka uspešno obrađena, ona se uklanja iz reda i *notifySemaphore* se otpušta kako bi se nastavilo sa asinhronim slanjem podataka. Ukoliko poslata poruka nije uspešno obrađena, povratna vrednost je *Failed,* poruka se neće ukloniti iz reda i semafor će se otpustiti. Neuspešno obrađena poruka od strane posrednika nije uklonjena iz reda jer mora ponovo da se pošalje. Taj proces se ponavlja sve dok se poruka ne isporuči uspešno.

### Posrednik (*Broker*)

Posrednik predstavlja centralnu komponentu implementiranog rešenja. Sa jedne strane prima podatke od menadžera i skladišti iste dok sa druge strane obrađuju zahteve od potrošača i prosleđuje im tražene podatke. Ima mogućnost rada sa više menadžera i potrošača istovremeno.

U okviru posrednika se nalazi rečnik pod nazivom *streamData* čiji je ključ tema na koju se podaci objavljuju a vrednost rečnika jeste lista zapisa. Čuva se *Callback* klijenta. Pomoću tog *Callback-a* posrednik obaveštava klijenta, u slučaju ove implementacije menadžera, da li je poruka uspešno obrađena.

Jedna od glavni karakteristika posrednika jeste mogućnost replikacije podataka. Preduslov za replikaciju podataka jeste da se kreira komunikacioni kanal (Listing 4.10) prema replikacionom servisu (4.2.5).



Listing 4.10 - Kreiranje proxy-a prema replikacionom servisu

Pozivom metode *RegisterToReplicationService* vrši se čuvanje *Callback-a* posrednika na replikacionom servisu. Ovaj korak je neophodan da bi se omogućilo slanje replikacionih podataka sa replikacionog servisa ka posredniku.

Posrednik implementira interfejs *IBroke*. *IBroker* nasleđuje sledeće interfejse:

* *IProducer* (Listing 4.1)
* *IConsumer* (Listing 4.3)

Implementacija prethodno navedenih interfejsa je neophodna jer u suprotnom ne bi mogle da se izvršavaju sve funkcionalnosti prema stranama koje šalju i zahtevaju podatke.

#### Prijem podataka

Kada je u pitanju prijem podataka, posrednik ima mogućnost da prima podatke sinhrono i asinhrono. U oba slučaja se poziva metoda *WriteRecord* (Listing 4.11)*.*



Listing 4.11 - Implementacija WriteRecord metode

U okviru *WriteRecord* metode se prvo proverava da li je pristigla poruka validna pozivom metode *CheckMessage*. Ukoliko poruka nije validna, baciće se izuzetak sa opisom greške. Nakon uspešne provere validnosti poruke kreira se zapis koji se kasnije čuva u rečniku. Zatim se proverava da li u rečniku postoji tema na koju je objavljuje poruka. Ukoliko tema ne postoji u rečniku povratna vrednost metode će biti *Failed*. Ako tema ipak postoji, zapis će se dodati u listu zapisa na pomenutu temu.

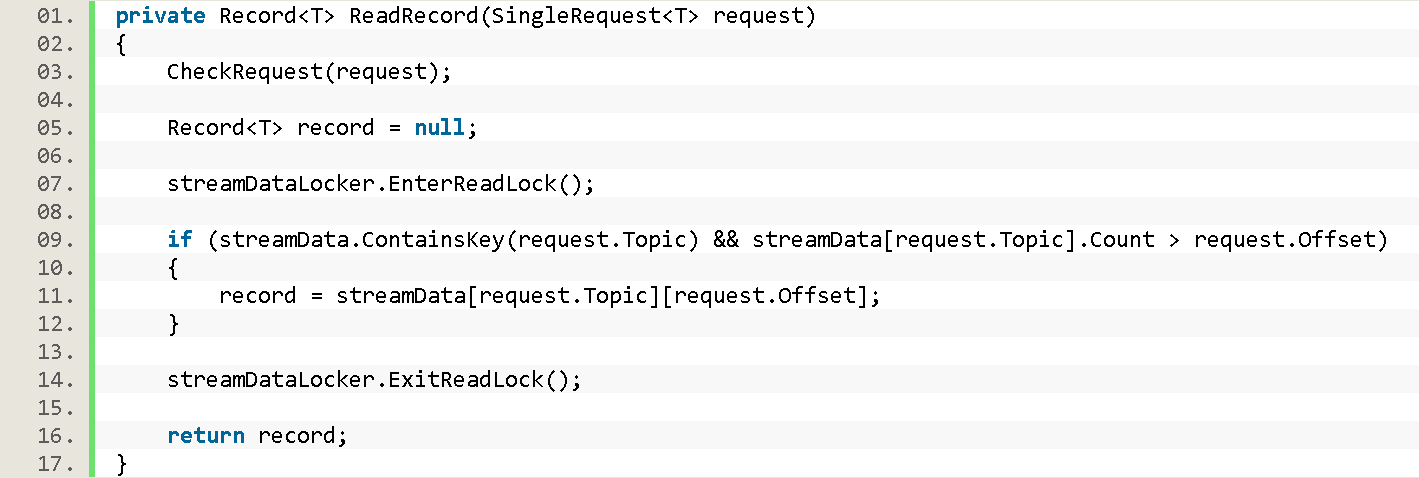
Ukoliko je stanje posrednika postavljeno na *Hot* i posrednik ima uspešno kreiran komunikacioni kanal prema replikacionom servisu, poruka koja je pristigla će se replicirati dalje do posrednikakoji je u *StandBy* stanju ukoliko on postoji.

Kod sinhronog prijema podataka, klijentu se vraća povratna vrednost metode *WriteRecod*. Kada je reč o asinhronom prijemu podataka, klijent koji je pozvao metodu za asinhrono slanje može da nastavi dalje sa izvršavanjem. On će potvrdu o statusu obrade poruke dobiti preko *Callback-a* i na taj način će odlučiti o daljim koracima u zavisnosti od sopstvene implementacije.

#### Obrada zahteva za podacima

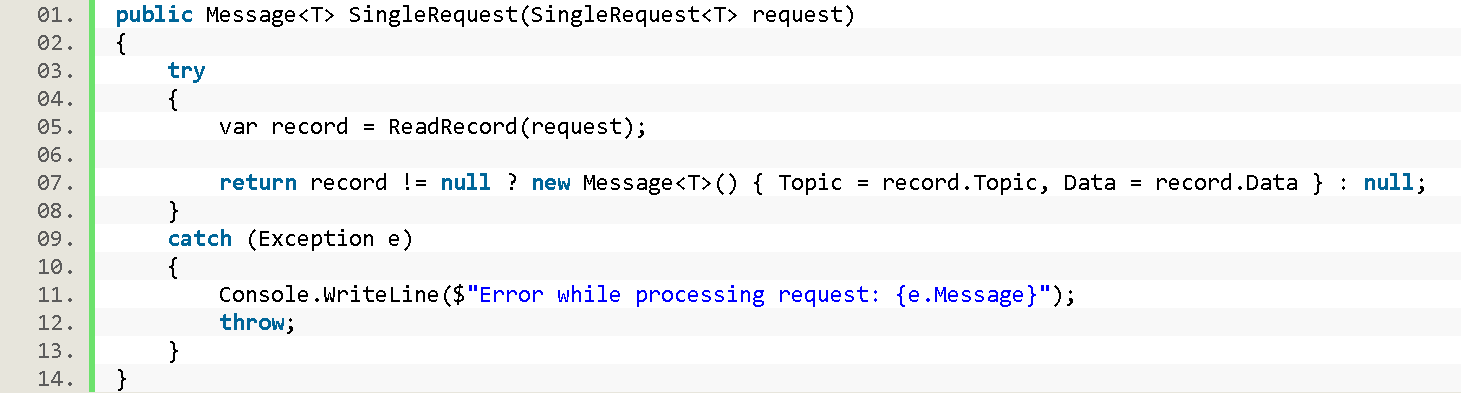
Klijenti šalju zahteve za podacima koje posrednik obrađuje i kao povratne vrednosti vraća tražene podatke. Postoji mogućnosti zahteva za većim brojem poruka od jednom ili da se zahteva jedna poruka.

Prilikom obrade zahteva u oba slučaja poziva se metoda *ReadRecord* (Listing 4.12). Prvo se proverava validnost pristiglog zahteva. Ukoliko zahtev nije validan baca se izuzetak



Listing 4.12 - Implementacija metode ReadRecord

U okviru zahteva se nalazi tema sa koje se čitaju podaci i pozicija (*Offset*) sa koje se iz liste čita podatak. Ukoliko postoji tema iz zahteva u rečniku i ukoliko postoji podataka na traženoj poziciji, taj podatak se čita iz liste i vraća se kao povratna vrednost metode. U suprotnom se vraća vrednost *null.* To nam govori da nismo dobro popunili zahtev.

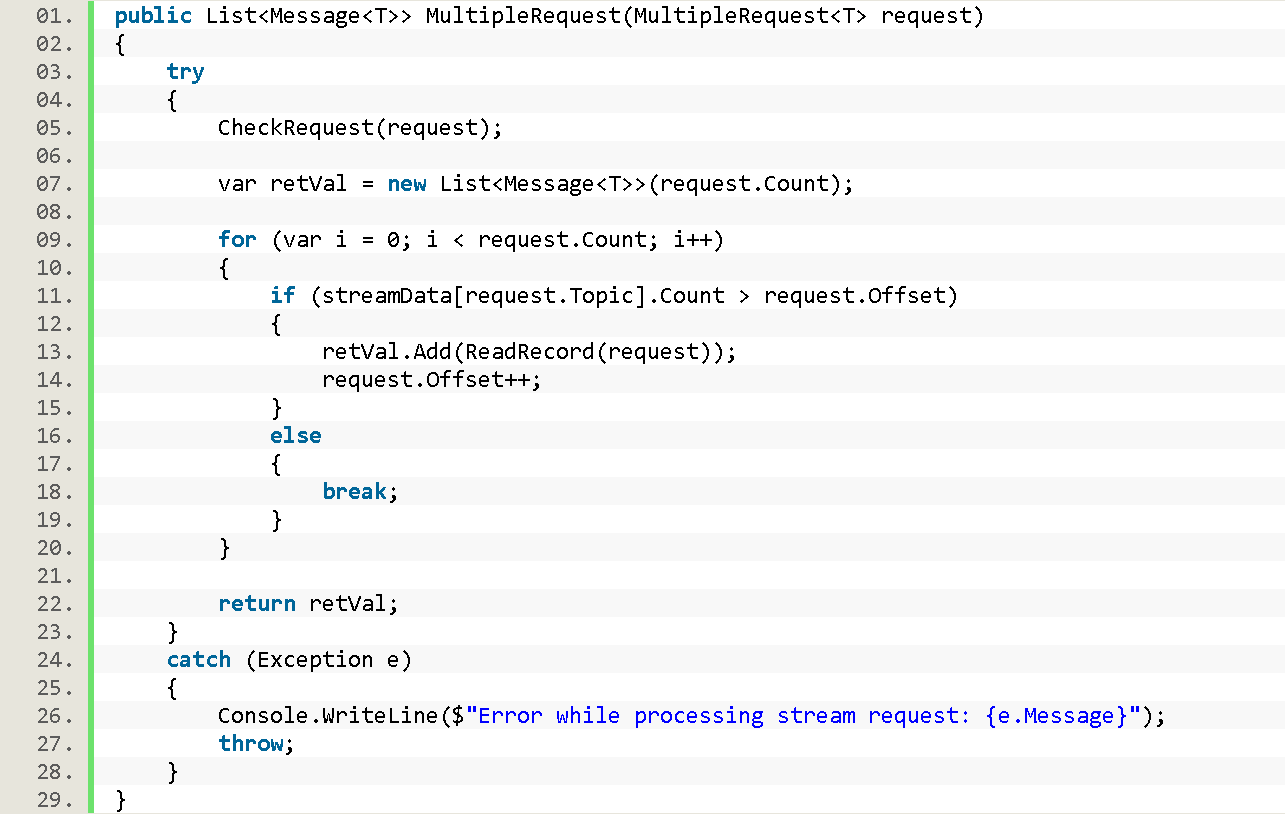


Listing 4.13 - Implementacija SingleRequest metode

Listing 4.13 prikazuje implementaciju metode *SingleRequest*. Pomoću ove metode klijent šalje zahtev za jednim podatkom. U okviru ove metode se poziva metoda *ReadRecord* (Listing 4.12). Ako je povratna vrednost metode *ReadRecord* *null*, ta vrednost se vraća kao povratna vrednost metode *SingleRequest*. Vrednost *null* znači da ne postoji podataka objavljen na tu temu sa traženom pozicijom. Kada se pronađe željeni zapis, on se prepakuje u poruku koja se vraća klijentu kao povratna vrednost metode *SingleRequest*.

Zahtev za više podatak od jednom (metoda *MultipleRequest*) (Listing 4.14) se realizuje tako što se kroz *MultipleRequest* prosledi tema, pozicija (*Offset*) i broj narednih podatak koje je potrebno dobaviti. Na početku metode se proverava validnost zahteva kao i kod metode *SingleRequest.* Nakon toga se *for* petljom čita svaki zapis i dodaje se u listu zapisa koja je kasnije vraća kao povratna vrednost metode. Ukoliko je broj zahteva veći od broja zapisa u listi, vraća se onoliko zapisa koliko ih je preostalo do kraja liste. *Null* vrednost se vraća kada ne postoji tema ili je pozicija prosleđena kroz zahtev pogrešna.

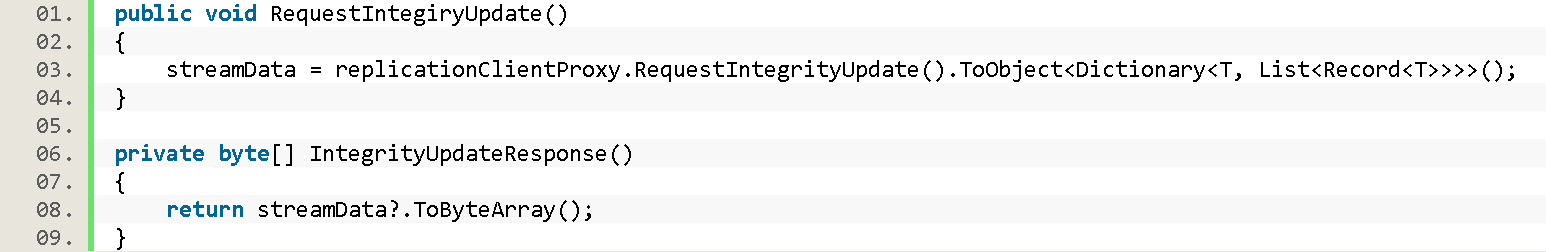
Oba tipa zahteva za podacima, *SingleRequset* (Listing 4.13) i *MultipleRequest* (Listing 4.14) se izvršavaju sinhrono. Sinhrono izvršavanje je neophodno jer klijenti ne mogu da nastave sa radom dok ne dobiju zahtevane podatke. Postoje implementacije klijenata koje mogu da izvršavaju dalje aktivnosti bez traženih podataka i njim bi bilo optimalnije da koriste asinhrone zahteve ali taj slučaj nije pokriven prilikom implementacije ovog rešenja.



Listing 4.14 - Implementacija MultipleRequest metode

#### Poravnanje podataka (*Integrity Update*)

Kada se posrednik sa stanjem *StandBy* pokrene, prva akcija jeste zahtev za poravnanjem podataka (*Integrity Update*). Posrednik šalje zahtev replikacionom servisu, koji prosleđuje zahtev partnerskom replikacionom servisu koji dalje taj zahtev prosleđuje posredniku koji je u *Hot* stanju. Komunikacija između replikacionih servisa će biti detaljno opisana u narednom poglavlju (4.2.5).



Listing 4.15 ' Metode za zahtev i odogov na Integrity Update

Listing 4.15 prikazuje dve metode:

* *RequestIntegrityUpdate* –Posrednik koji je u *StandBy* stanju poziva ovu metodu kako bi poravnao podatke sa posrednikom koji se nalazi u *Hot* stanju.
* *IntegrityUpdateResponse* – Povratna vrednost ove metode jesu svi zapisi koji se trenutno nalaze u rečniku posrednika pretvoreni u niz bajta. Ukoliko je rečnik prazan povratna vrednost će biti *null.*

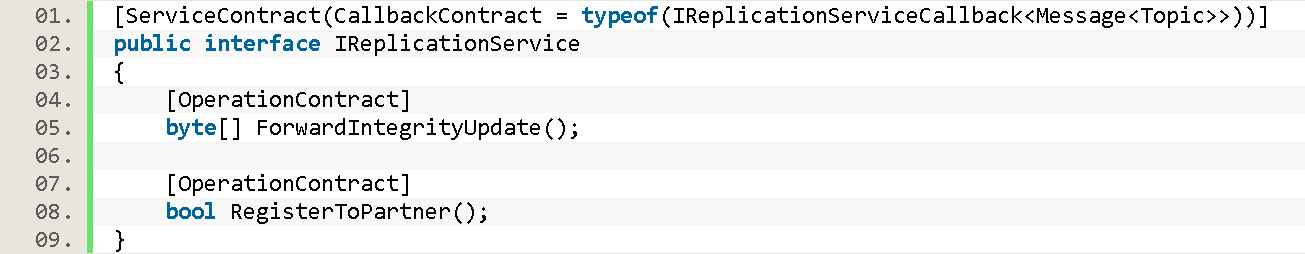
### Replikacioni servise (*Replication Service*)

Za potrebe implementacije replikacionog servisa (1.3) u ovom projektu postoje četiri interfejsa:

* *IReplicationService*
* *IReplicationServiceCallback*
* *IReplicationClient*
* *IReplicationClientCallbacak*

Za komunikaciju između replikacionog servisa i partnerskog replikacionog servisa koriste se interfejsi *IReplicationService* i *IReplicationServiceCallback*. Komunikacija između replikacionog servisa i klijenta, što je u slučaju ovog projekta posrednik, koriste se interfejsi *IReplicationClient* i *IReplicationClientCallback*.

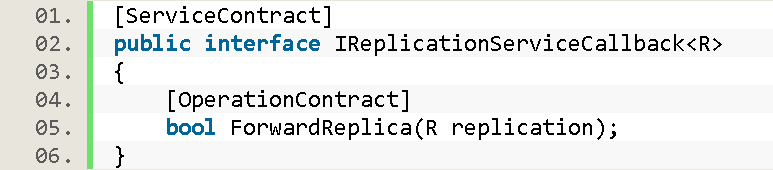
Replikacioni servis čuva *Callback* metodu klijenta kako bi mogao da mu isporuči repliku podataka ako je u *StandBy* stanju, ili kako bi mogao da mu prosledi zahtev za poravnanjem podataka ako je u *Hot* stanju. Takođe čuva *Callback* partnerskog replikacionog servisa kako bi mogao da mu prosledi repliku ili podatke na zahtev za poravnanje podataka ako je u *Hot* stanju. Ako je replikacioni servis u *StandBy* stanju tada se kreira komunikacioni kanal prema partneru koji je u *Hot* stanju kako bi mogao da se registruje ili da zatraži zahtev za poravnanjem podataka.



Listing 4.16 - IReplicationService interfejs

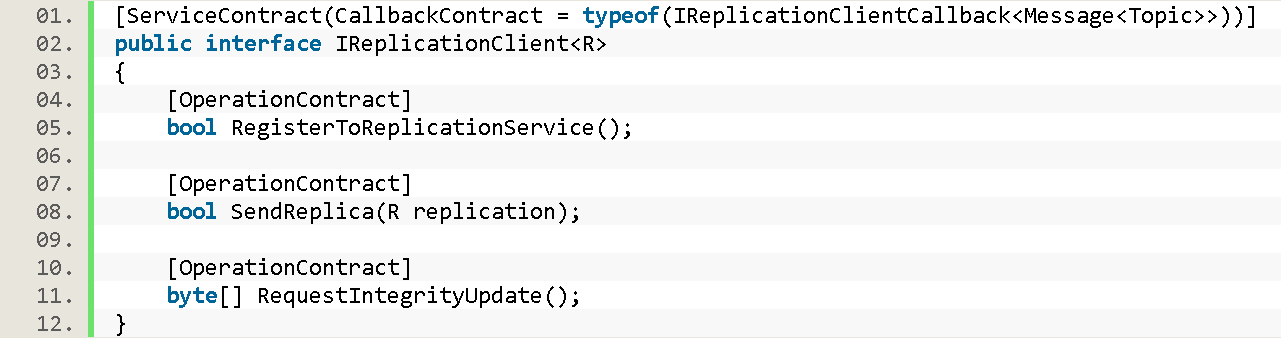
Listing 4.16 prikazuje *IReplicationService* interfejs koji se sastoji od dve metode:

* *ForwardIntegrityUpdate* – ovu metodu poziva partnerski replikacioni servis kada klijent koji je u *StandBy* stanju želi da izvrši poravnanje podataka (*Integrity Update*). U okviru ove metode se pomoću klijentskog *Callback-a* zahtevaju potrebni podaci koji se vraćaju kao njena povratna vrednost
* *RegisterToPartner* – partnerski replikacioni servis poziva ovu metodu odmah nakon kreiranja komunikacionog kanala. Unutar ove metode se čuva *Callback* partnera uz pomoć *Callback* *Handler-a* kako bi kasnije mogli da se prosleđuju neophodni podaci.



Listing 4.17 - IReplicationServiceCallback interfejs

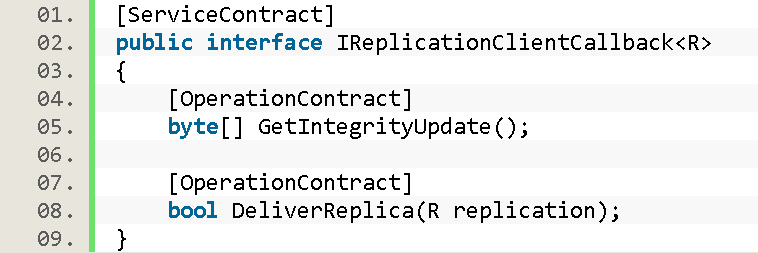
Listing 4.17 prikazuje *IReplicationServiceCallback* interfejs. U sklopu pomenutog interfejs se nalazi metoda *ForwardReplica*. Pomoću ove metode se preko *Callback-a* replika prosleđuje partnerskom replikacionom servisu koji je dalje prosleđuje odgovarajućem klijentu.



Listing 4.18 - IReplicationClient interfejs

*IReplicationClient* interfjes (Listing 4.18) čine tri metode:

* *RegisterToReplicationService* – metoda koju klijent poziva nakon kreiranja komunikacionogkanala prema replikacionom servisu. Unutar ove metode se uz pomoć *Callback* *Handler-a* čuva *Callback* klijenta. Sačuvani *Callback* se kasnije koristi da bi se klijentu prosledila replika ili zahtev za poravnanjem podataka.
* *SendReplica* – klijent poziva ovu metu kada je primio neke podatke, ili ih izgenerisao, i želi da ih replicira. U okviru ove metode replikacioni servis prosleđuje partneru repliku pomoću *Callback-a* parnera koji je prethodno sačuvao.
* *RequestIntegrityUpdate* – ovu metodu poziva klijent kada je u *StandBy* stanju. Unutar ove metode replikacioni servis prosleđuje zahtev za poravnanjem podataka putem komunikacionogkanala koji je prethodno kreirao.



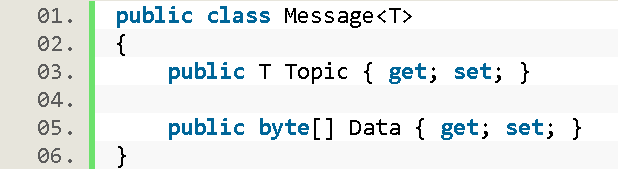
Listing 4.19 - IReplicationClientCallback interfejs

Listing 4.19 prikazuje *IReplicationClientCallback* interfejs. Ovaj interfejs implementira klijentska aplikacija (u ovom projektu posrednik) i sastoji se iz sledećih metoda:

* *GetIntegrityUpdate* – svrha ove metode je da replikacioni servis nakon zatraženog zahteva za poravnanjem podataka od klijenta dobavi potrebne podatke. Replikacioni servis poziva ovu metodu preko *Callback-a* koji je prethodno sačuvao.
* *DeliverReplica* – metoda koja prosleđuje replicirane podatke sa replikacionog servisa na klijenta. Takođe se poziva preko *Callback-a* koji je sačuvan na replikacionom servisu.

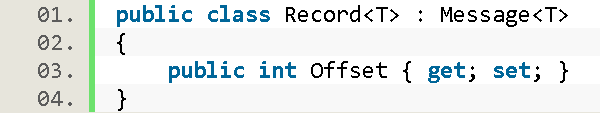
## Struktura podataka

U ovom poglavlju će biti predstavljen format poruka koji se koristi u procesu razmene podataka u prethodno opisanom implementiranom rešenju kao i format u kojem se čuvaj podaci pristigli na *Broker* komponentu.



Listing 4.20 - Klasa Message

Klasa *Message* (Listing 4.20) se sastoji od generičkog tipa *Topic* i niza bajta *Data*. Upotrebljava se prilikom razmene podataka između komponenti. Generički tip *Topic* predstavlja temu na koju se podaci objavljuju. Prednosti generičke teme jeste što može da se prilagodi različitim implementacijama sistema. Tema može biti bilo koji tip podatka ili novi objekat koji sami kreiramo. Niz bajta *Data* služi za smeštanje podataka koje želimo da pošaljemo. Pre slanja podaci se serijalizuju u niz bajta i smeštaju se u polje *Data*. Kada se poruka primi, polje *Data* se jednostavno deserijalizuje u potreban tip podatka i imamo sve informacije koje su nam od značaja.



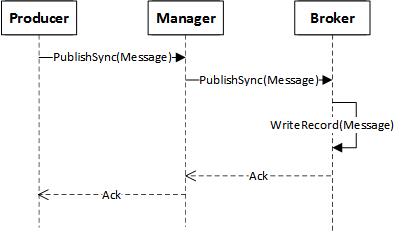
Listing 4.21 - Klasa Record

Listing 4.21 prikazuje klasu *Record*. Ona se upotrebljava prilikom čuvanja podataka u okviru *Broker* komponente. Nasleđuje klasu *Message*. Uz to sadrži polje *Offset* koje nam govori na kom mestu u particiji se nalazi traženi podatak. Vrednosti polja *Offset* se postavlja neposredno pre smeštanja u listu.

# Dijagram sekvence

U ovom poglavlju ćemo opisati tok izvršavanja akcija prilikom sinhronog i asinhronog slanja podataka. Takođe će biti opisana raplikacija podataka kao i izvršavanje zahteva za poravnanjem podataka.

## Sinhrono slanje podataka



Slika 5.1 - Sinhrono slanje podataka

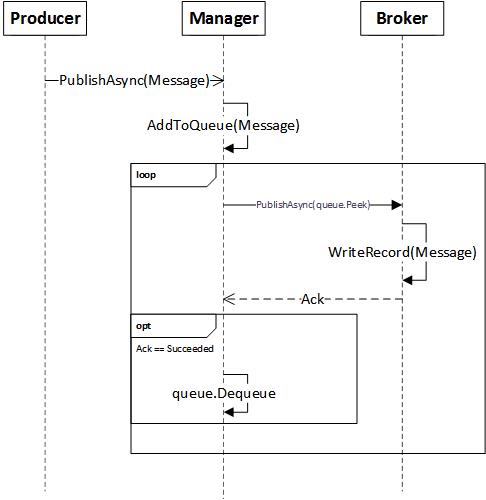
Sinhrono slanje podataka (Slika 5.1) se izvršava na sledeći način:

* Proizvođač podatke šalje menadžeru pomoću metode *PublishSync* i čeka na potvrdu da li su se podaci uspešno isporučili.
* Menadžer dobijene podatke prosleđuje posredniku i takođe čeka potvrdu.
* Posrednik dobijene podatke upisuje u odgovarajuću bazu podataka i kao odgovor vraća menadžeru da li je operacija uspešno izvršena.
* Menadžer tu potvrdu prosleđuje dalje do proizvođaču koji može da nastavi sa radom i u zavisnosti od odgovora vrši izbor daljih koraka.

## Asinhrono slanje podataka

Slika 5.2 prikazuje korake izvršavanja prilikom asinhronog slanja podataka. tok izvršavanja je sledeći;

* Proizvođač šalje podatke menadžeru pozivom funkcije *PublishAsync*. Ne čeka na odgovor i može da nastavi sa izvršavanjem.
* Menadžer prima podatke od proizvođača i smešta iz u red za asinhronu komunikaciju (*asyncQueue*)
* Menadžer u posebnoj niti konstantno proverava da li postoje podataka u redu za asinhronu komunikaciju. Ukoliko ima podataka za slanje, isti se čitaju iz reda i šalju se posredniku.
* Posrednik prima podatke i upisuje ih u bazu podataka. Nakon toga obaveštava menadžera da li je operacija uspešno izvršena
* Menadžer na osnovu odgovora odlučuje o narednim koracima. Ukoliko je odgovor pozitivan (*Succeeded*) menadžer skida prethodno poslatu poruku sa rade i šalje sledeću. ukoliko odgovor nije pozitivan, menadžer ponovo šalje istu poruku posredniku.

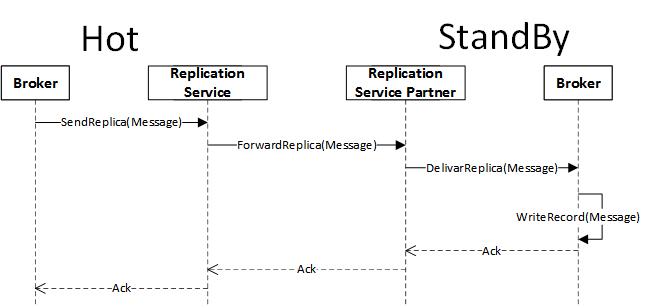


Slika 5.2 - Asinhrono slanje podataka

## Replikacija podataka

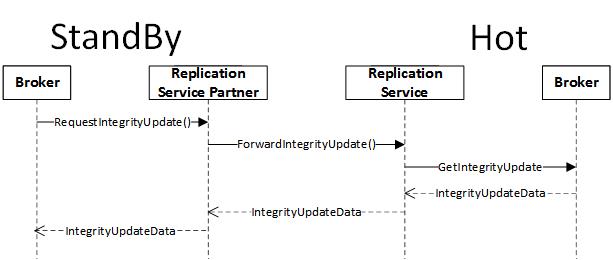
Proces replikacije podataka (Slika 5.3) izvršava se na sledeći način:

* Nakon prijema podataka od menadžera posrednik primljene podatke šalje replikacionom servisu pozivom funkcije *SendReplica*.
* Replikacioni servis dobijene podatke od posrednika prosleđuje partnerskom replikacionom servisu pozivom funkcije *ForwardReplica*.
* Partnerski replikacioni servis prosleđuje podatke do posrednika koji se nalazi u *StandBy* stanju.



Slika 5.3 - Replikacija podataka

## Poravnanje podataka (*Integrity Update*)



Slika 5.4 - Koraci prilikom poravnanja podataka (Integrity Update)

Slika 5.4 prikazuje korake prilikom zahteva za poravnanjem podataka:

* Posrednik koji je u *StandBy* stanju šalje zahtev za poravnanjem podataka replikacionom servisu pozivom funkcije *RequestIntegrityUpdate*
* Partnerski replikacioni servis prosleđuje taj zahtev replikacionom servisu koji se nalazi u *Hot* stanju pozivom funkcije *ForwardIntegrityUpdate*
* Zatim replikacioni servis pozivom funkcije *GetIntegirtyUpdate* prosleđuje zahtev do posrednika koji je u *Hot* stanju koji kao povratnu vrednost vraća podatke potrebne za poravnanje.
* Replikacioni servis tako dobijene podatke vraća partnerskom replikacionom servisu koji ih dalje prosleđuje do posrednikakoji je zatražio zahtev za poravnanje podataka.

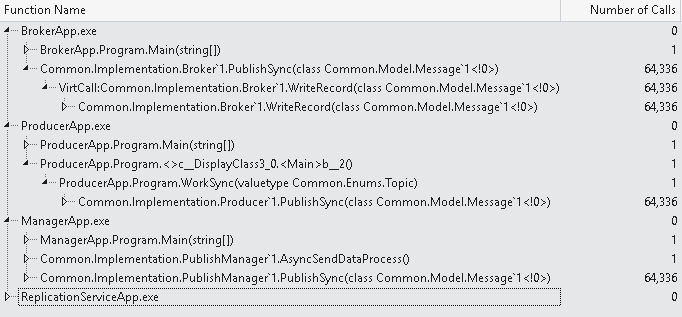
# Performanse

U okviru poglavlja 6 će biti predstavljeni rezultati merenje propusnosti sinhronog i asinhronog načina slanja podataka kao i opterećenje procesora u toku pomenutog procesa. Rezultati će biti predstavljeni pomoću *Performance Profiler-a* koji se nalazi u sklopu *Visual Studio-a* (3.3).

Karakteristike okruženja koje se koristi za merenje performansi su sledeće:

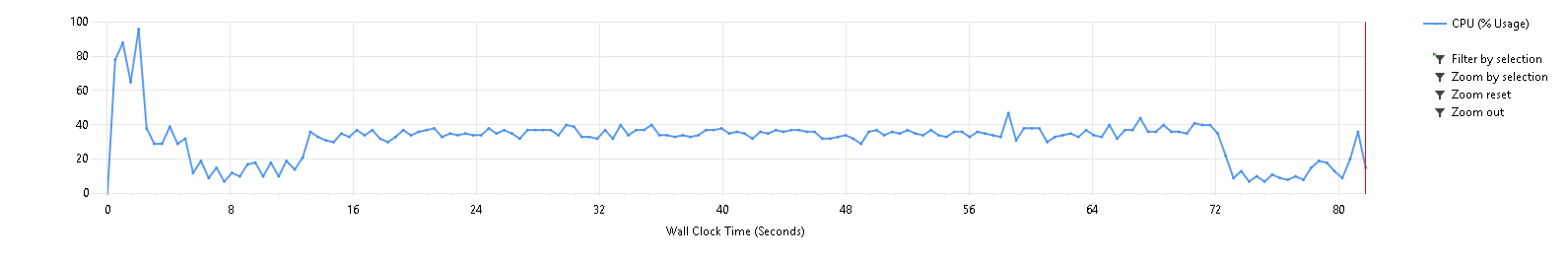
* *Windows* 10 operativni sistem
* 8 GB RAM memorije
* Intel(R) Core i7-4500U

## Performanse kod sinhronog slanja podataka



Slika 6.1 - Propusnost kod sinhronog slanja podataka

Slika 6.1 prikazuje propusnost (broj poruka koje se pošalju u određenom vremenskom intervalu) kod sinhronog slanja podataka. Vremenski period u okviru kog se merila propusna moć jeste jedan minut. Za jedna minut je poslato 64,336 poruka, što se može videti u *Number of Calls* koloni. Za sinhrono slanje poruka poziva se *PublishSync* metoda kod proizvođača. Kod posrednika se poziva metoda *WriteRecord* u okviru koje se čuvaju podaci. Test pokazuje da je i ona pozvana 64,336, što znači da su svi podaci uspešno isporučeni.

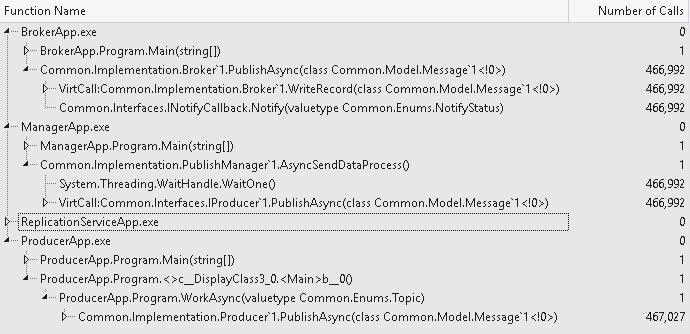


Slika 6.2 - Opterećenje procesora kod sinhronog slanja podataka

Kada je reč o opterećenju procesora kod sinhronog slanja podataka (Slika 6.2) vidimo da je reč o 40% od punog kapaciteta procesora. Procesor je slabije opterećen je sinhrona komunikacija zahteva odgovor kao potvrdu da bi se nastavilo dalje sa radom.

Kod sinhrone razmene podataka trpi propusnost na račun konzistentnosti podataka.

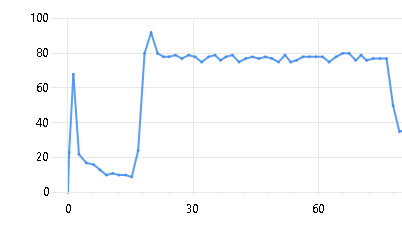
## Performanse kod asinhronog slanja podataka



Slika 6.3 - Propusnost kod asinhronog slanja podataka

Propusnost asinhronog slanja podataka (Slika 6.3) je očekivano veća nego propusnost kod sinhronog slanja. Vremenski period za koji se merila propusnost je takođe jedan minut. Za asinhrono slanje podataka se koristi funkcija *PublishAsync* kod proizvođača. Sa slike (Slika 6.2) možemo videti da je ta funkcija pozvana 466,992 put. Kod posrednika je funkcija *WriteRecord* pozvana takođe 466,992 puta, što pokazuje da su svi podaci uspešno isporučeni. Za isti vremenski period se prilikom asinhronog slanja pošalje oko sedam puta više podataka nego kod sinhronog slanja..

Kod asinhronog slanja podataka konzistentnost trpi na račun performansi.



Slika 6.4 - Opterećenje procesora kod asinhronog slanja podataka

Slika 6.4 prikazuje koliko je procesor opterećen prilikom asinhronog slanja podataka. Možemo da vidimo da opterećenje varira oko 80%. Opterećenje procesora je veće nego kod sinhronog slanja jer se ne čeka odgovor nego se nastavlja dalje sa slanjem podataka.

# Zaključak

Razmena podataka u distribuiranim sistemima predstavlja osnovni zahtev. Povećanjem broja čvorova povećava se i broj zahteva koje je neophodno obraditi. Teži se za rešenjem koje će davati dobre performanse u procesu razmene podataka.

Cilj ovog rada je razvoj programskog rešenja za sinhronu i asinhronu razmenu podataka. Implementirano rešenje predstavlja jedan tip mehanizma za razmenu podataka u distribuiranim sistemima. Implementirano rešenje omogućava čuvanje podataka nakon slanja i klijenti mogu u svakom trenutku da zahtevaju podatke koji su im potrebni.

Razvijena je posebna komponenta koja jeste i centralna komponenta ovako razvijenog rešenja. Tu se skladište poslati podaci i vrše se obrade zahteva za podacima. Ukoliko klijenti ne uspeju da obrade podatke ili im zatraženi podaci nisu dostavljeni, mogu ponovo da pošalju isti zahtev jer su svi podaci sačuvani na jednom mestu.

Još jedna od prednosti jeste replikacija podataka na partnerske komponente. Samim tim se smanjuje šansa od gubitka podataka prilikom otkaza jedne od komponenti sistema. Implementirano rešenje podržava replikaciju podataka na samo jednu partnersku komponentu.

Dalji razvoj rešenja bi mogao da teče u sledećim pravcima:

* Omogućiti replikaciju podataka na više partnerskih komponenti istovremeno. Time bi podigli sigurnost podataka na još viši nivo. Podaci bi bili replicirani na nekoliko geografski udaljenih lokacija.
* Implementirati logiku za prelazak sa *StandBy* na *Hot* i nastavak nesmetanog rada sistema (*Failover*). Ukoliko komponenta koja se nalazi u *Hot* stanju padne i postane neaktivna, komponenta koja se nalazi u *StandBy* stanju treba da preuzme zadatke i nastavi sa izvršavanjem zahteva tamo gde je stala komponenta koja je prethodno bila u *Hot* stanju.
* Klijentima su omogućeni samo sinhroni zahtevi za podacima. Implementacija asinhronih zahteva bi omogućila klijentima da ne čekaju na odgovor ukoliko im podaci nisu potrebni za dalje izvršavanje.

# Literatura

1. Andrew S. Tanenbaum, *Distributed Systems Principles and Paradigmas*, accessed 03.09.2019.
2. Wikipedia, *Apache Kafka*, accessed 03.09.2019,<https://en.wikipedia.org/wiki/Apache\_Kafka>
3. Free Code Camp, *A Thorough Introduction To Distributed Systems*, accessed 03.09.2019, <https://www.freecodecamp.org/news/a-thorough-introduction-to-distributed-systems-3b91562c9b3c/>
4. The New Stack, *Apache Kafka: A Primer*, accessed 05.09.2019, <https://thenewstack.io/apache-kafka-primer/>
5. Kafka Apache, *Introduction*, accessed 05.09.2019, <https://kafka.apache.org/intro.html>
6. Microsoft, *What is .NET framework?*, accessed 10.09.2019, <https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework>
7. Stack Overflow*, What is unit testing?*, accessed 10.09.2019, <https://stackoverflow.com/questions/1383/what-is-unit-testing>
8. Tech Target, *C Sharp*, accessed 10.09.2019, <https://searchwindevelopment.techtarget.com/definition/C>
9. Lifewire, *Data Consistency and Its Role Within Transactions*, accessed 11.09.2019, https://www.lifewire.com/database-consistency-definition-1019249
10. Cisco, *Cisco Visual Networking Index*, accessed 18.09.2019, <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html>
11. Tech Target, *Failover*, accessed 20.09.2019, <https://searchstorage.techtarget.com/definition/failover>
12. Techopedia, *Failover*, accessed 20.09.2019, <https://www.techopedia.com/definition/1202/failover>
13. Wikipedia, *Windows Communication Foundation*, accessed 20.09.2019, <https://sr.wikipedia.org/sr-el/Windows\_Communication\_Foundation>
14. Microsoft, *Whait Is Windows Communication Foundation*, accessed 20.09.2019, <https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf>
15. Tutorials Teacher, *Generics in C#*, accessed 20.09.2019, <https://www.tutorialsteacher.com/csharp/csharp-generics>
16. Microsoft, *Generics (C# Programming Guide)*, accessed 20.09.2019, <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/>
17. Png All, *Networking png*, accessed 22.09.2019, <http://www.pngall.com/networking-png/download/13339>

# Podaci o kandidatu

Kandidat Kristijan Salaji rođen je 02.01.1995. godine u Novom Sadu. Osnovnu školu “Aleksa Šantić“ u Vajskoj završio je 2010. godine. Završio je srednju elektrotehničku školu “9. Maj“ 2014. godine u Bačkoj Palanci. Iste godine je upisao osnovne akademske studije na Fakultetu Tehničkih Nauka u Novom Sadu, smer Elektroenergetski Softverski Inženjering, koje je završio 2018. godine sa prosečnom ocenom 8.50. Odmah nakon završenih osnovnih akademskih studija upisuje master akademske studije na Fakultetu Tehničkih Nauka, smer Primenjeno Softversko Inženjerstvo. Ispunio je sve obaveze i položio je sve predmete predviđene studijskim programom.