KOMUNIKACIJA NEOVISNIH DISTRIBUIRANIH USLUGA KORIŠTENJEM SERVICE BUS-a

Prtenjača, Daniel

VISOKO UČILIŠTE ALGEBRA

DIPLOMSKI RAD

KOMUNIKACIJA NEOVISNIH DISTRIBUIRANIH USLUGA KORIŠTENJEM SERVICE BUS-a

Daniel Prtenjača

Zagreb, kolovoz 2018.

Predgovor

Koristim priliku da se zahvalim mom mentoru prof. Andreju Lackoviću na korisnim savjetima, iskazanom povjerenju u moje ideje, te nesebičnoj podršci pri izradi diplomskog rada.

Također zahvala i Visokoj školi za primijenjeno računarstvo - Visoko učilište Algebra bez kojeg moje studiranje ne bi bilo moguće.

Zahvale i svim profesorima, asistentima i kolegama koji su mi pomogli tijekom mog studiranja.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme diplomskog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Tema diplomskog rada je implementacija neovisnih distribuirani h sustava koji međusobno komuniciraju uz pomoć Service Bus-a. Distribuirani sustavi sastoje se od izvora podataka, tj. aplikacije koja sadrži podatke potrebne drugim neovisnim sustavima i dva sustava koja očekuju nekakve podatke. Service Bus aplikacija omogućuje razmjenu poruka od izvorišne aplikacije do krajnjih aplikacija. Nakon izvedbe aplikacija, prikazana je razmjena poruka između neovisnih sustava.

Ključne riječi: Service Bus, sustavi, razmjena

Abstract

The topic of the graduate thesis is the implementation of independent distribution systems that communicate with each other through the service bus. Distilled systems consist of a data source, ie an application that contains data required by other independent systems and two systems that expect some data. The Service Bus application lets you exchange messages from source applications to end-applications. After the application is running, the message exchange between independent systems is displayed.

Keywords: Service Bus, systems, exchange

Sadržaj

1. Uvod

U posljednje vrijeme zabilježen je veliki porast komunikacije između neovisnih distribuiranih sustava. Svaki sustav se na različite načine pokušava zaštiti od drugih sustava, ali potreba za međusobnim komuniciranjem stvara velike probleme mnogim velikim tvrtkama i vlasnicima aplikacija. Direktan pristup drugoj aplikaciji i dohvaćanje podataka narušava integritet aplikacija, što može proizvesti pristup podacima ili krađu podataka. Zbog toga se sve više implementiraju sustavi koji prenose poruke s jednog sustava na drugi.

U ovom diplomskom radu opisuju se principi i metode rada Service Bus-a. U prvom dijelu, opisuje se pojam distribuirani sustavi, servisno orijentirana arhitektura, mikroservisi i uloga Service Busa u distribuiranim sustavima. U drugom dijelu opisuje se Service Bus, njegove funkcije, prednosti i mane. Nakon toga pojašnjava se izvedba aplikacije koja je izvor podataka i aplikacija koje očekuju poruke od izvora, U zadnjem dijelu detaljno se opisuje rad Service Busa i njegovih značajki.

2. Distribuirane usluge

Pojam distribuirani sustav rijetko se koristi, a pridonosi velike značajke prilikom komuniciranja računala. Svaki distribuirani sustav ima komponente koje čine sustav, a te iste komponente da bi mogle međusobno komunicirati i koordinirati jedna drugu, moraju biti na umreženim računalima. Zbog toga komponente međusobno komuniciraju kako bi postigle zajednički cilj.

Distribuirani sustav se sastoji od:

- Računalni program(distribuirani program)
- Distribuirani programski proces

Distribuirani program se odnosi na bilo koji računali program koji se izvodi unutar distribuiranog sustava, a distribuirani programski proces je proces pisanja takvih programa. Danas postoji mnogo vrsta implementacija za mehanizam za prosljeđivanje poruka, kao što su redovi poruka, HTTP i mnogi drugi.

Svako računalo može imati neki problem, zbog toga u distribuiranim sustavima postoji i distribuirana računanja s kojima se rješavaju računalni problemi. Ako u distribuiranom sustavu, tj. u mreži postoji više računala koji međusobno komuniciraju, svaki od njih će rješavati jedan dio zadatka koji je definiran u distribuiranom računalstvu.

“Ne postoji jedinstvena definicija distribuiranog sustava, da bi se lakše definirao najlakše je objasniti tako da postoji nekoliko autonomnih računskih entiteta (računala ili čvorova), od kojih svaki ima svoju lokalnu memoriju. Subjekti komuniciraju jedni s drugima slanjem poruka. Distribuirani sustav može imati zajednički cilj, kao što je rješavanje velikog računalnog problema, korisnik tada percipira prikupljanje autonomnih procesora kao jedinice. Svako računalo može imati vlastiti korisnik s individualnim potrebama, a svrha

distribuiranog sustava je koordiniranje korištenja dijeljenih resursa ili pružanje komunikacijskih usluga korisnicima."[1]

.

Tipična svojstva distribuiranih sustava uključuju da :

- Sustav mora tolerirati kvarove na pojedinim računalima.
- Struktura sustava (topologija mreže, latencija mreže, broj računala) nije unaprijed poznata, sustav se može sastojati od različitih vrsta računala i mrežnih veza, a sustav se može promijeniti tijekom izvršenja distribuiranog programa.
- Svako računalo ima samo ograničeni, nepotpuni prikaz sustava.

Svako računalo može znati samo jedan dio unosa

2.1. Funkcije i primjena distribuiranih usluga

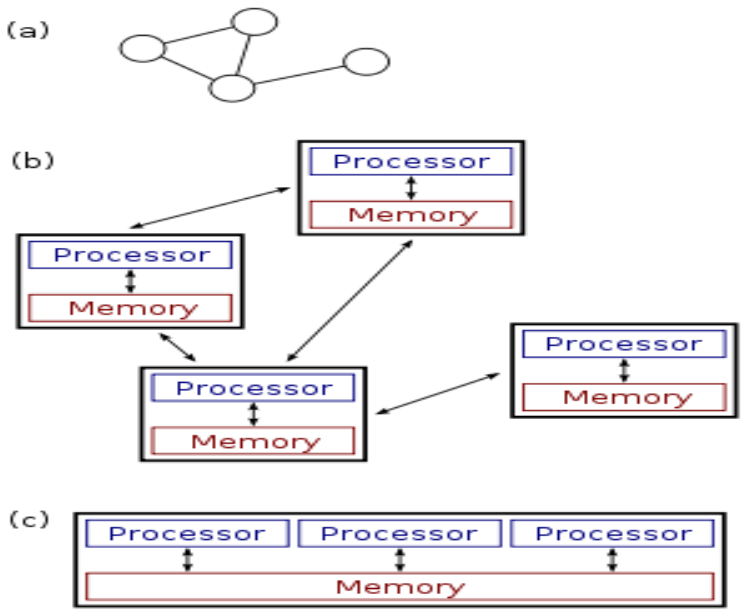
Kako distribuirane sustave čine niz umreženih računala koja imaju isti cilj, a to je prenošenje poruka, razlikujemo nekoliko izraza:

- Istodobno računanje,
- Paralelno računanje,
- Distribuirano računanje

Poznato je da se informacije razmjenjuju između računalnih procesora. Tako sva ova tri gore navedena izraza su vrlo slična. Među njima nema neke jasne razlike, zbog toga što isti sustav može biti karakteriziran kao paralelni, istodobni i distribuirani. Paralelno računanje može se promatrati kao poseban čvrsto povezani oblik distribuiranog računanja, a distribuirani računalni sustav može se promatrati kao labavo spojeni oblik paralelnog računanja. Ipak, moguće je grubo klasificirati usporedne sustave kao "paralelne" ili "distribuirane".

[1] Izvor: https://en.wikipedia.org/wiki/Distributed_computing

Kod paralelnog računanja svi procesori mogu imati pristup zajedničkoj memoriji za razmjenu informacija između procesora, au distribuiranom računalstvu, svaki procesor ima svoju privatnu memoriju.



Slika 1.Razlika između distribuiranih i paralelnih sustava

Slika prikazuje razliku između distribuiranih i paralelnih sustava.

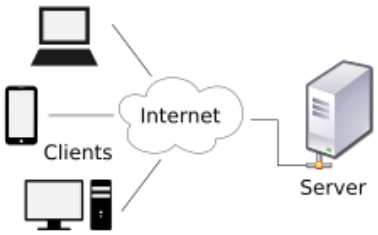
- Slika (a) je shematski prikaz tipičnog distribuiranog sustava; sustav je predstavljen kao topologija mreže u kojoj je svaki čvor računalo, a svaka linija koja povezuje čvorove je komunikacijska veza.
- Slika (b) prikazuje isti distribuirani sustav s više pojedinosti: svako računalo ima svoju lokalnu memoriju, a informacije se mogu razmjenjivati samo davanjem poruka iz jednog čvora na drugi pomoću dostupnih komunikacijskih veza.
- Slika (c) prikazuje paralelni sustav u kojem svaki procesor ima izravan pristup zajedničkoj memoriji.

Različite arhitekture hardvera i softvera koriste se za distribuirani računalstvo. Potrebno je međusobno povezati više procesora s nekom vrstom mreže, bez obzira na to je li ta mreža tiskana na ploču ploče ili načinjena od labavo povezanih uređaja i kabela. Na višoj razini, potrebno je međusobno povezivati procese koji se izvode na tim procesorima s nekim komunikacijskim sustavom.

Distribuirani program obično spada u jednu od nekoliko osnovnih arhitektura:

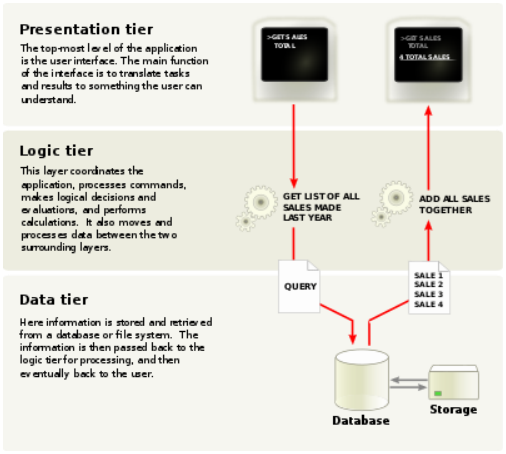
- Client–server (klijent-poslužitelj)
- Three-tier (troslojni),
- n-tier
- peer-to-peer.

Client-server:arhitektura u kojoj klijenti kontaktiraju poslužitelj za dohvat podataka, a nakon toga obrađuju i prikazuju korisnicima.



Slika 2. Client-server

Three-tier (Tri razine): arhitektura u kojoj se dodaje srednji sloj koji implementira logiku potrebnu klijentima za korištenje. Zbog toga se u web aplikacijama uvodi ova arhitektura za tri razine koja omogućuje jednostavniju implementaciju aplikacija.



Slika 3. Three-tier

N-tier: arhitektura omogućuje web aplikacijama slanje zahtjeva na druge sustave.

Peer-to-peer: arhitekture gdje nema posebnih strojeva koji pružaju uslugu ili upravljaju mrežnim resursima.Peers može poslužiti i kao klijenti i kao poslužitelji.

Slika 4. Peer-to peer

Osim navedenih aspekata postoji još jedan ne manje bitan, koji opisuje komunikaciju i koordinaciju rada između istovremenih procesa.Kroz različite protokole za prolaz poruke, procesi mogu izravno međusobno komunicirati. Arhitektura koja je bazirana na bazi podataka omogućuje zajedničko dijeljene podataka koristeći zajedničku bazu bez ikakve izravne komunikacije među njima

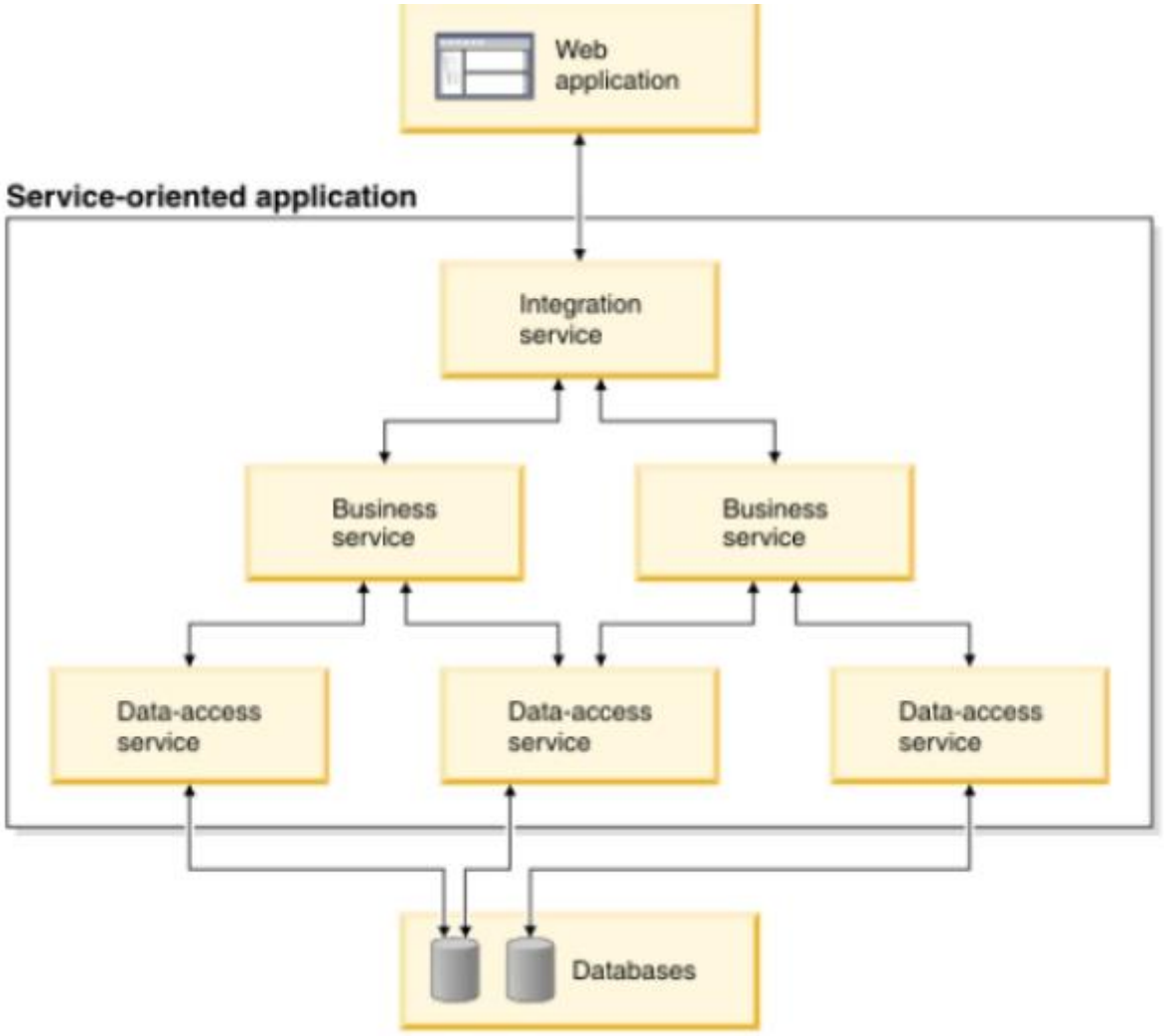
2.2. SOA(Service oriented architecture)

SOA uključuje implementaciju usluga koje se izvode u mreži a imaju brojne karakteristike. Najvažnije karakteristike SOA-e su:

- Upravljanje poslovnim procesom kao što je izračunavanje citata osiguranja ili distribucija e-pošte;
- Obrađivanje tehničkih zadataka kao što je pristup bazi podataka;
- Pružanje poslovnih podataka i tehničkih detalja za izradu grafičkog sučelja;
- Uz odgovarajuću tehnologiju runtime može pristupiti tradicionalnom programu i odgovoriti na različite vrste podnositelja zahtjeva kao što su web aplikacije;
- Relativno je neovisan o drugim softverima jer promjene zahtijevaju male ili nikakve promjene u samoj usluzi i unutarnjoj logici usluge - ovakva relativna neovisnost usluge i drugog softvera naziva se i 'labav spoj';
- Usluga može rukovati interakcijama unutar same tvrtke te između tvrtke i njezinih npr. dobavljača, partnera i kupaca.[2]

SOA je stil razvoja koji se fokusira na poslovanje kao cjelinu i na modularnost, te ponovnu upotrebu. SOA je aplikacija čije su usluge najčešće u hijerarhiji kao što je prikazano na *Slika 5. SOA*

[2] https://www.ibm.com/support/knowledgecenter/en/SSMQ79_9.5.1/com.ibm.egl.pg.doc/topics/pegl_serv_overview.html



Slika 5. SOA

Najviša razina u hijerarhiji sadrži jednu ili više integracijskih usluga, od kojih svaka kontrolira tijek aktivnosti od kojih svaka poziva jednu ili više poslovnih usluga.

Druga razina sastoji se od usluga od kojih svaka ispunjava relativno nisku razinu poslovnog zadatka. Na primjer, usluga integracije može se pozvati na niz poslovnih usluga za potvrdu pojedinosti, te ako se vrate vrijednosti koje se smatraju "izdavanjem pravila" - usluga integracije poziva drugu poslovnu uslugu.

Najniža razina sastoji se od usluga pristupa podacima, od kojih svaka upravlja tehničkim zadatkom čitanja i pisanja u području pohrane podataka u baze podataka. Usluga pristupa podacima najčešće se poziva iz poslovnog sloja, ali jednostavan pristup uslugama omogućuje različite namjene.

Središnja točka je fleksibilnost gdje integracijske usluge pružaju različite operacije različitim podnositeljima zahtjeva i/ili se pozivaju na druge integracijske usluge i pristupa različitim vrstama usluga unutar aplikacije orijentirane prema usluzi, a podnositelj zahtjeva može pristupiti integracijskoj usluzi u jednom trenutku te poslovnoj službi u drugoj.

2.2.1. Poslovne implikacije SOA-e

SOA utječe na poslovanje tvrtke na način kada je svaka komponenta relativno samostalna jedinica gdje vaša tvrtka može brže reagirati na poslovne ili tehnološke promjene i uz manje troškove.

Dobro oblikovana SOA povećava agilnu sposobnost tvrtke da brzo reagira i dobro se mijenja tijekom vremena.

SOA također utječe na suradnju ljudi jer dobro napisana usluga je ona usluga kada ljudi mogu razumjeti svrhu čak i uz minimalno znanje o samom softveru gdje poslovni analitičari mogu razmjenjivati informacije i uključivati korisnike u ranim raspravama o opsegu svake usluge te razumjeti implikacije promjene poslovnog postupka.

SOA može tvrtki pomoći da izbjegne dodatne troškove tako da se izbjegne razvoj novog softvera, te mogu imati koristi od ponovne uporabe povećanjem pouzdanosti softvera

tijekom vremena. Možete obaviti manje opsežne testove ako se postojeći servis stavi u novu aplikaciju, u usporedbi s testiranjem potrebnim za implementaciju softvera koji je napisan ispočetka.

SOA se možete upotrijebiti kako bi poslovni procesi i podaci bili dostupniji.

2.3. Mikroservisi

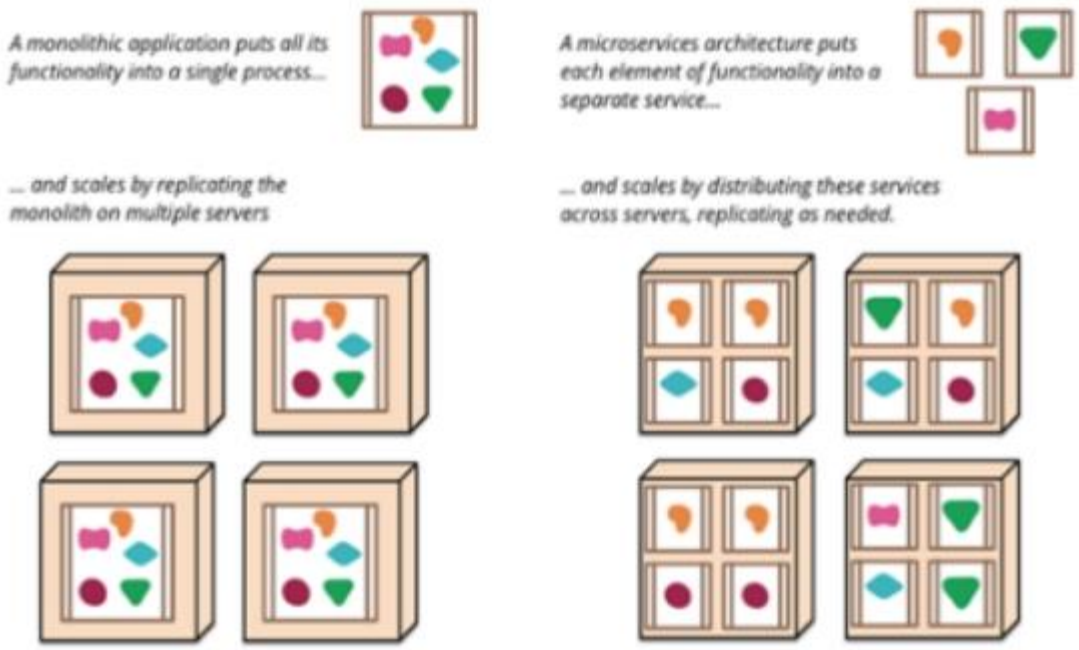
Mikroservisi su oblik pristupa distribuiranim sustavima koji promiču upotrebu servisa vlastitim životnim ciklusom koji međusobno surađuju. Riječ je o arhitekturi u kojoj se servisi lako zamjenjuju, organiziraju na temelju poslovnih sposobnosti, a mogu se implementirati različitim programskim jezicima i u različitim hardverskim i softverskim okruženjima.
Mikroservisi se uglavnom temelje na poslovnim domenama te se njima izbjegava problem tradicionalno ustrojenih arhitektura te također obuhvaćaju nove tehnologije i tehnike koje su se pojavile u posljednjih deset godina, čime se izbjegavaju mogući problemi implementacije arhitekture usmjerene na usluge.

Mikroservisi stvaraju različite komponente softverskog dizajna i smještene su kao pojedinačne, izolirane usluge. Svaki se distribuira odvojeno i komuniciraju putem dobro definiranih sučelja na mreži te se zbog njihove izolacije i zahtjeva za komunikacijom sprječavaju potencijalni problemi.

"Mikroservisne arhitekture predstavljaju u velikoj mjeri kontrast tradicionalnom monolitnom arhitekturalnom stilu. Osnovna značajka monolita je da je kompletna poslovna logika sustava pakirana u jedinstvenu izvedbenu jedinicu. Druga važna značajka monolita je da promjena u implementaciji bilo kojeg dijela programske logike izaziva potrebu ponovne izgradnje, pakiranja i instalacije cijele aplikacije. To postaje problem kad imamo velike sustave koji su implementirani kao monoliti, i kad njihovi procesi izgradnje, pakiranja i instalacije nisu adekvatno automatizirani. Također, dodatni je problem i

činjenica da monolitne sustave skaliramo tako da pokrećemo njihove dodatne instance (skaliramo kompletnu aplikaciju, umjesto da skaliramo kritične poslovne funkcije, i time kvalitetnije koristimo sistemske resurse). "[3]

Mikroservisi također definiraju ponašanje razvojnih timova, grade izolirane poslovne funkcije i upravljaju njihovim životnim ciklusima.



Slika 6. Monoliti i mikroservisi.

[3] https://medium.freecodecamp.org/microservices-from-idea-to-starting-line-ae5317a6ff02

Mikroservisi i SOA stavljaju u fokus usluge kao glavnu komponentu. Razlikuju se u pogledu karakteristika usluga. Mikroservisi su logična evolucija SOA-e i podržavaju suvremene slučajeve poslovanja. SOA je pogodnija za velike i složene poslovne aplikacijske i okruženja koja zahtijevaju integraciju s mnogim aplikacijama. Aplikacije temeljene na tijeku rada koje imaju dobro definirane protoke informacija teže je implementirati pomoću SOA obrasca. Male aplikacije također nisu prikladne za SOA jer ne trebaju komponente za razmjenu poruka. Mikroservisi su dobro prilagođeni za manje i dobro podijeljene web sustave.

Mikroservisi razvojnim programerima daju veću kontrolu kod razvoja aplikacija.Kod orkestracije niza poslovnih procesa , SOA pruža bolji skup alata. U ranoj fazi poslovanja, mikroservisi se mogu pokazati kao dobar izbor ali kod povećanja obujma poslovanja i rasta tvrtke potrebna je pretvorbe kompleksnih zahtjeva i heterogenih integracija sustava. U takvim situacijama vjerojatno se možete obratiti obrascu SOA kako biste zamijenili MSA. SOA i MSA su isti skup standarda koji se koriste u različitim slojevima poduzeća.

2.4. Uloga ESB u distribuiranim sustavima

2.4.1. ESB vs SOA

ESB omogućuje tvrtkama da usvajaju načela SOA-e bez potrebe za zamjenom cjelokupne infrastrukture. Budući da se samostalni ESB-ovi obično grade prema otvorenim standardima, tvrtke daju fleksibilnost u integraciji širokog spektra sustava. Za razliku od prethodnih SOA inicijativa, ESB ne nameću dobavljača zaključavanje ili arhitektonski izbor. ESB tvrtkama omogućuje da postignu ono što SOA inicijativa obećava.

2.4.2. ESB vs mikroservisi

ESB mora ispuniti funkciju integracije, koordinacije, usmjeravanja i praćenja poslovne djelatnosti. Razumijevanje ESB-a tako možemo izgraditi aplikacije putem usluga ili mikro usluga kako bismo riješili zahtjeve i potrebe tvrtke. Usluge moraju biti tretirane na individualiziran način sa standardiziranim sučeljem na platformu s automatski skalabilnim vremenom izvršenja. Mikro-servisi ne znače smrt ESB-ova ako se ta sredstva koriste na prikladan način već se koncentriraju na arhitekturu u kojoj usluge imaju glavnu ulogu.

3. Enterprise Service Bus (ESB)

Enterprise Service BUS (ESB) je alat koji se koristi za distribuciju zadataka između povezanih komponenti aplikacije. ESB-ovi funkcioniraju tako da omogućuju jedinstven način pomicanja rada i nude aplikacijama mogućnost povezivanja i pretplatu na poruke. Koriste jednostavna strukturna i poslovna pravila.

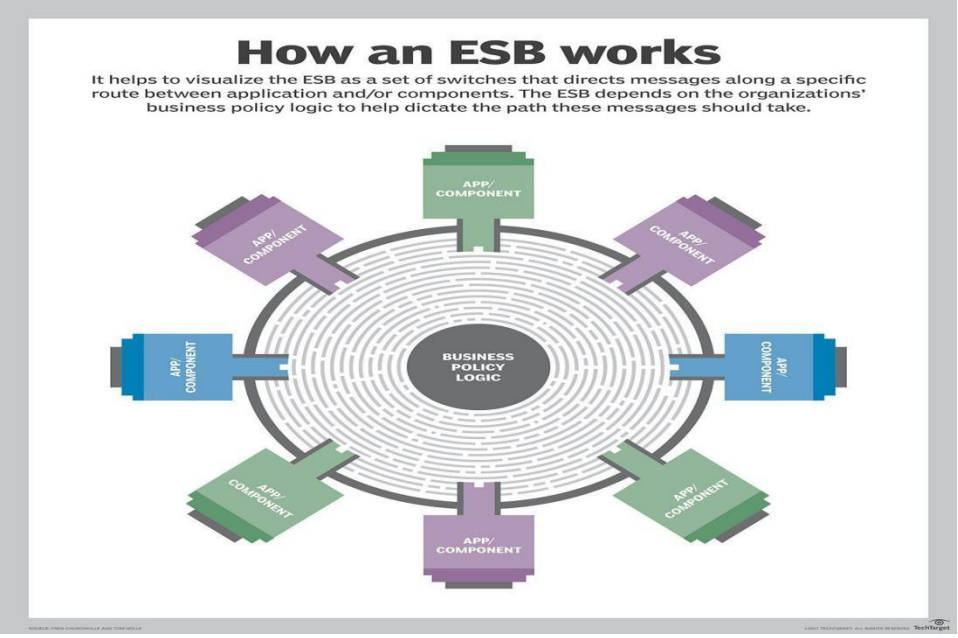
ESB alat koji koristi i distribuiranu računalnu integraciju i komponentu. Možemo ga opisati kao skup prekidača koji mogu usmjeravati poruku na određenoj ruti između komponenti aplikacije na temelju sadržaja poruke i implementacije ili poslovne politike.

ESB je središte aplikacijskog tijeka rada. To je zapravo red čekanja za poruke koji obrađuje razmjenu informacija kroz aplikaciju te ne diktira jesu li komponente koje koriste lokalne ili udaljene, niti provode posebne zahtjeve za programske jezike. ESB pomaže ujediniti različite načine na koje komponente mogu primati ili slati informacije drugim elementima aplikacije.

Integracija aplikacija i podataka smatra se jednim od najvećih izazova današnjeg poslovanja. Izgradnjom Enterprise Service Bus-a je najbrži i najisplativiji način odgovora na taj izazov.

Korištenje Web Servisa, preko Interneta ili intraneta, u arhitekturi informacijskog sustava stvara veliki broj pristupnih točaka. Njihovo upravljanje i održavanje, kao i održavanje velikog broja pristupnih točaka je izrazito teško. Organizacije koje koriste veliki broj servisa, te SOA moraju osigurati pouzdanu komunikacijsku infrastrukturu, inteligentno preusmjeravanje, i međusobnu interakciju među servisima. Aplikacijski poslužitelji omogućavaju keširanje veza (connection pooling), upravljanje transakcijama, upravljanje životnim ciklusom objekata, te tako olakšavaju razvoj novih aplikacija. EBS osigurava standardnu komunikaciju i integraciju. EBS koristi industrijske standarde za većinu funkcionalnosti i tako osigurava neovisnost o platformi i interoperabilnost različitih sustava. ESB nije novi softverski proizvod već put za integraciju aplikacija, koordinaciju

resursa i upravljanje informacijama. ESB za razliku od drugih pristupa integriranju, interoperabilnosti distribuiranih aplikacijskih sustava (npr. RPC) povezuje aplikacije koje se izvršavaju paralelno na različitim platformama, napisane različitim programskim jezicima korištenjem različitih programskih modela.[4]



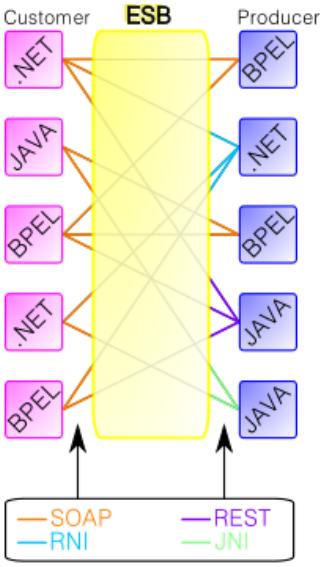
Slika 7. Rad ESB

3.1. Funkcije i primjena ESB-a

Enterprise Service Bus (ESB) implementira komunikacijski sustav između međusobno interaktivnih softverskih aplikacija u uslužno orijentiranoj arhitekturi (SOA). Ona

[4] http://www.infodom.hr/default.aspx?id=32

implementira softversku arhitekturu kao što je prikazano na slici. Budući da implementira distribuiranu računalnu arhitekturu, implementira posebnu varijantu modela klijent-poslužitelj gdje svaka aplikacija koja koristi ESB može koristiti kao poslužitelj ili klijent. ESB potiče agilnost i fleksibilnost s obzirom na komunikaciju visokih razina protokola između aplikacija. Primarni cilj komunikacije protokola visoke razine jest integracija poslovnih aplikacija.



Slika 8. Sve usluge korisnicima komuniciraju na isti način s ESB-om: ESB prevodi poruku u ispravnu vrstu poruke i šalje poruku ispravnoj službi za korisnike.

ESB primjenjuje koncept dizajna koji radi u mrežama različitih i nezavisnih računala unutar suvremenih operacijskih sustava. ESB pruža usluge robe uz usvajanje, prevođenje i usmjeravanje zahtjeva klijenata odgovarajućim uslugama odgovaranja.

Primarne dužnosti ESB-a su:

- Slanje poruka rutom između usluga
- Praćenje i kontroliranje usmjeravanja razmjene poruka između usluga
- Rješavanje sukoba između komunikacijskih komponenti usluge
- Kontrola implementacije i verzije usluga

- Pružanje usluga kao što su rukovanje događajima, preoblikovanje i mapiranje podataka, čekanje i redoslijed poruka i događaja, sekvencioniranje sigurnosti ili iznimka, pretvorba protokola i provođenje odgovarajuće kvalitete komunikacijske usluge

ESB se koristi u softveru koji djeluje između poslovnih aplikacija i omogućuje međusobnu komunikaciju među njima. ESB bi trebao moći zamijeniti sve izravne kontakte s aplikacijama na ESB-u na tako da se sva komunikacija odvija preko ES

postigao, ESB mora obuhvatiti funkcionalnost koju nudi njegova komponenta aplikacija na smislen način upotrebom modela poslovnih poruka. Model poruke definira standardni skup poruka koje ESB prenosi i prima. Kada ESB primi poruku, usmjerava poruku odgovarajućoj aplikaciji uz transformaciju poruke u format koji aplikacija može tumačiti.

ESB-ovi se oslanjaju na precizno konstruiranje modela poslovnih poruka i ispravno oblikovanje funkcionalnosti koju nude aplikacije. Ako model poruke u potpunosti ne obuhvaća funkcionalnost aplikacije, onda ostale aplikacije moraju zaobići sabirnicu i pozivati izravno neusklađene aplikacije.

Funkcije ESB-a:

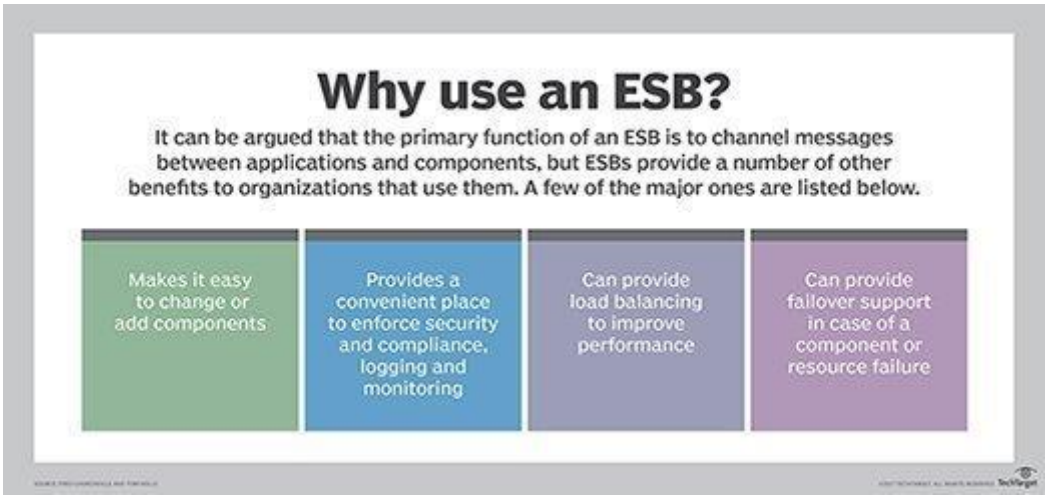
- Pozivanje sinkronih i asinkronih transportnih protokola,
- Mapiranje usluga (lociranje i vezivanje)
- Usmjeravanje na temelju sadržaja i usmjeravanje na temelju pravila
- Obrada poruke i pojačavanje poruka
- Procesna koreografija i implementacija složenih poslovnih procesa
- Složeni događaj obrade događaja - interpretacija, korelacija, podudaranje uzoraka
- Ostala kvaliteta sigurnosti usluga (šifriranje i potpisivanje), pouzdana isporuka, upravljanje transakcijama
- Sigurnost standardiziranog sigurnosnog modela za autorizaciju, provjeru autentičnosti i reviziju korištenja ESB-a

- Pretvorba koja olakšava transformaciju formata podataka i vrijednosti, uključujući usluge transformacije (često preko XSLT ili XQuery) između formata aplikacije za slanje i primanje poruka
- Provjera valjanosti prema shemama za slanje i primanje poruka
- Upravljanje sposobnošću primjene jedinstvenih pravila poslovanja
- Razdvajanje i kombiniranje višestrukih poruka i rukovanje iznimkama

3.2. Prednosti korištenja ESB

Budući da ESB arhitektura upravlja načinom rada, olakšava promjenu komponenti ili dodavanje dodatnih komponenti aplikaciji. Budući da ESB vidi sve, to također čini prikladno mjesto za provođenje zahtjeva za sigurnost i sukladnost ili iznimke, a čak i upravljanje praćenjem izvedbe transakcija.

Također može često pružiti podršku za neuspjeh ako komponenta ili njezini resursi ne uspiju.



Slika 9. Zašto koristiti ESB

3.3. Mane korištenja ESB

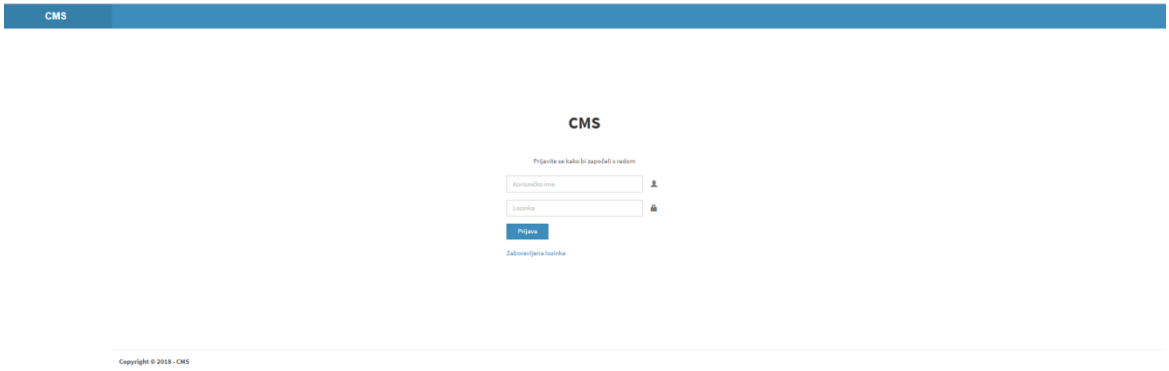
Izazov u ESB konceptu ne postoji niti jedan standard prihvaćen za značajke ili ponašanje. Iako se može tvrditi da je primarna funkcija ESB-a da djeluje kao sabirnica poruka koja usmjerava poruke između aplikacija ili komponenti prema jeziku pravila, tijekom vremena se termin koriti za opisivanje svega što na neki način podržava tijek rada.

4. Praktična izvedba

U ovom poglavlju će biti opisana programska implementacija Service Bus-a.

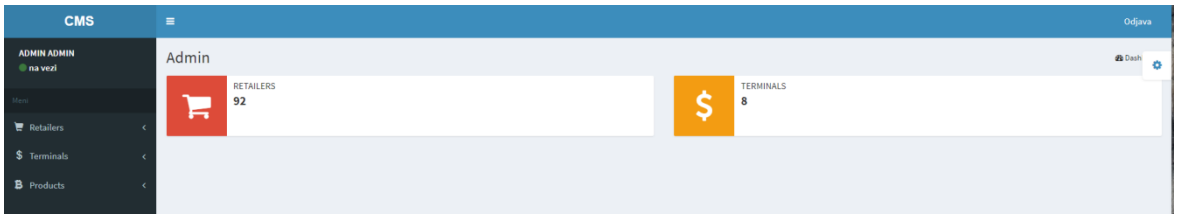
4.1. Izvorna aplikacija (CMS aplikacija)

Izvorna aplikacija sadrži podatke o prodajnim mjestima i terminalima. Prodajno mjesto je definirano nazivom partnera, mjestom, gradom, adresom i produktima. Svaki produkt se definira prilikom kreiranja prodajnog mjesta. Terminali se kreiraju zasebno od prodajnog mjesta, a jedno prodajno mjesto može sadržavati više terminala. Isto tako terminali poprimaju produkte koji su definirani na prodajnom mjestu.



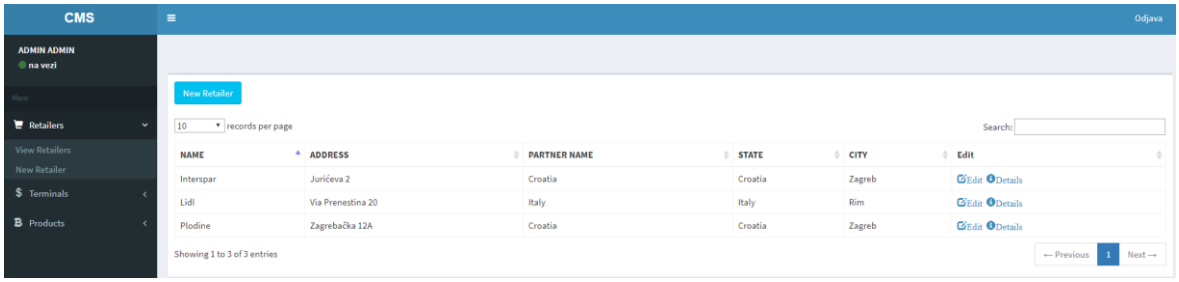
Slika 10. CMS – prijava

Početna stranica prikazuje ukupni broj prodajnih mjesta i terminala. Admin sustava ima pravo uređivanja prodajnih mjesta, terminala i dodavanje novih produkata.



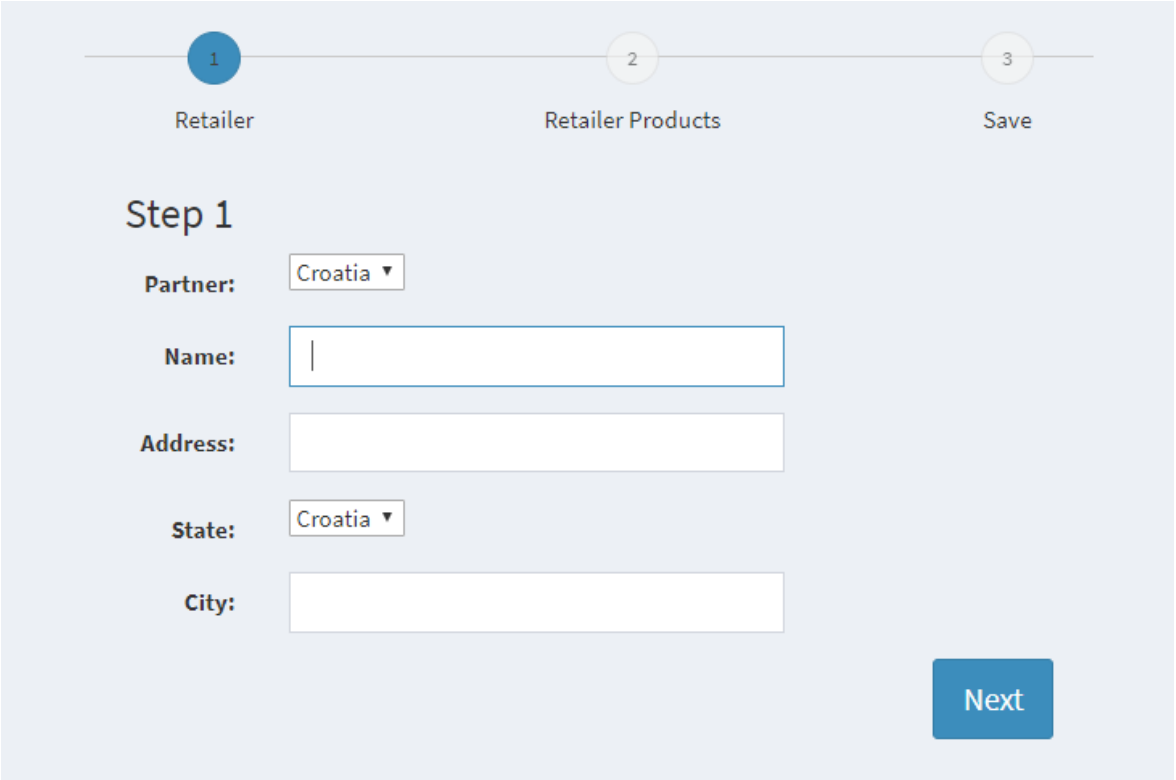
Slika 11. CMS – početna stranica

Pregled prodajnih mjesta prikazan je tablicom. Za svako prodajno mjesto se može vidjeti opis i opcija za ažuriranje podataka..



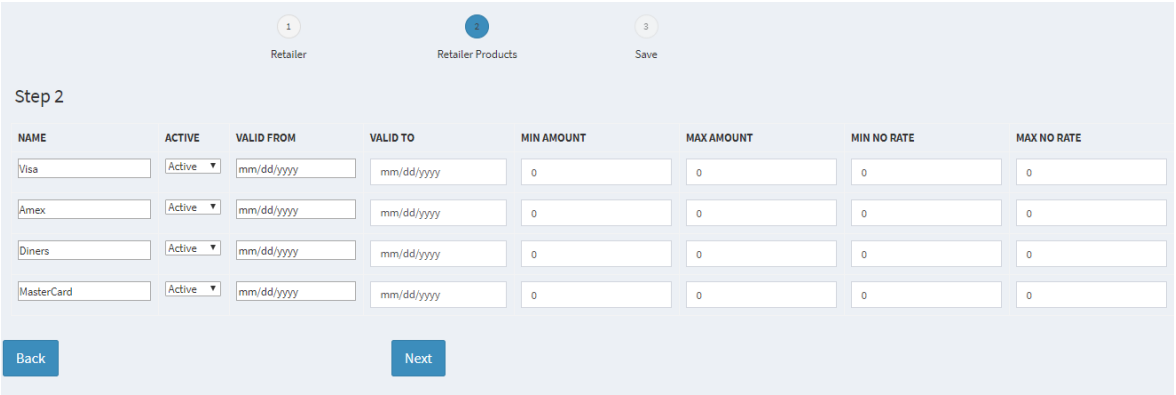
Slika 12. CMS – pregled prodajnih mjesta

Kreiranje prodajnog mjesta se sastoji od tri koraka. U prvom koraku upisuje se naziv prodajnog, partner, grad i adresa.



Slika 13. CMS – dodavanje novog prodajno mjesta

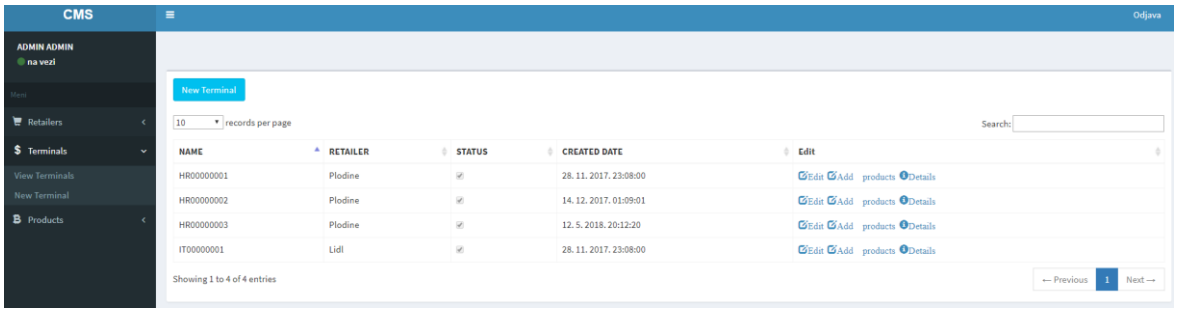
Drugi korak sadržava sve produkte koje prodajno mjesto može imati. Postavljanje produkta kao aktivnog uz minimalni i maksimalni broj rata, te minimalnog iznosa definiramo da to prodajno mjesto im aktivan produkt kojeg smo aktivirali.



Slika 14. CMS – dodavanje novog prodajno

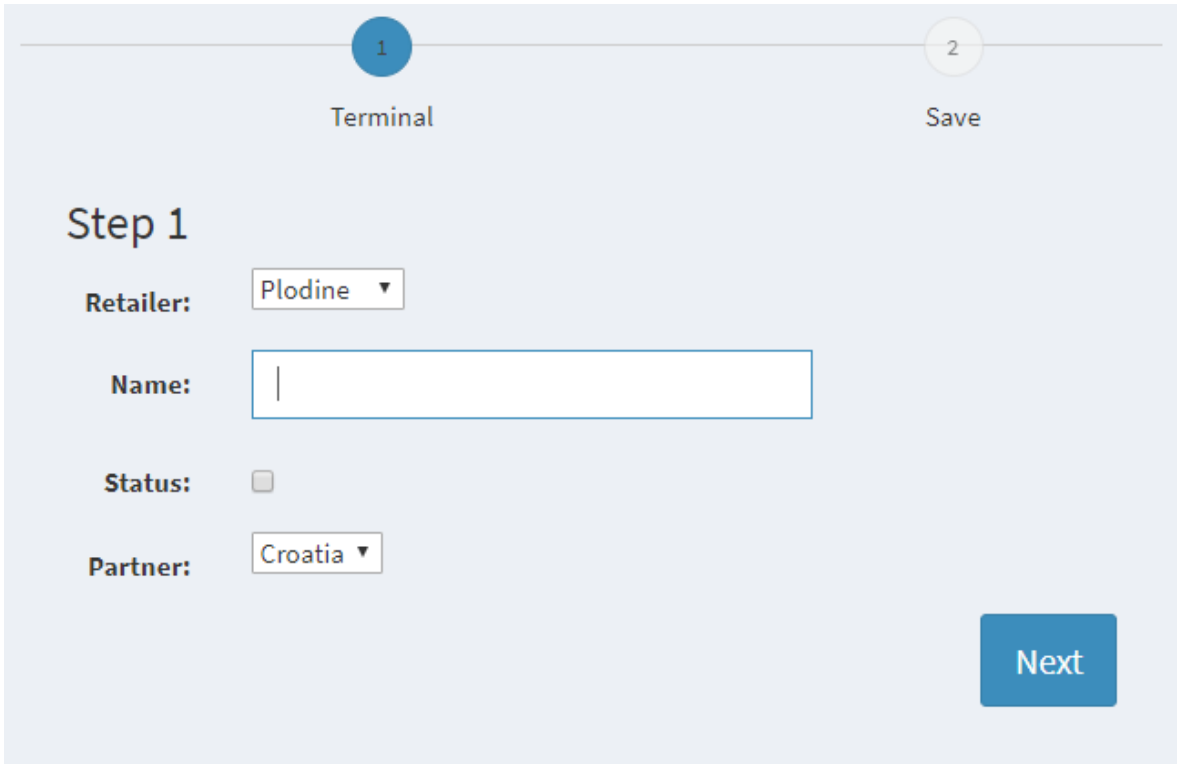
Treći korak je spremanje podataka u bazu.

Terminali su prikazani tablicom. Za svaki terminal se može vidjeti detaljan opis terminal i opcija za ažuriranje terminala.



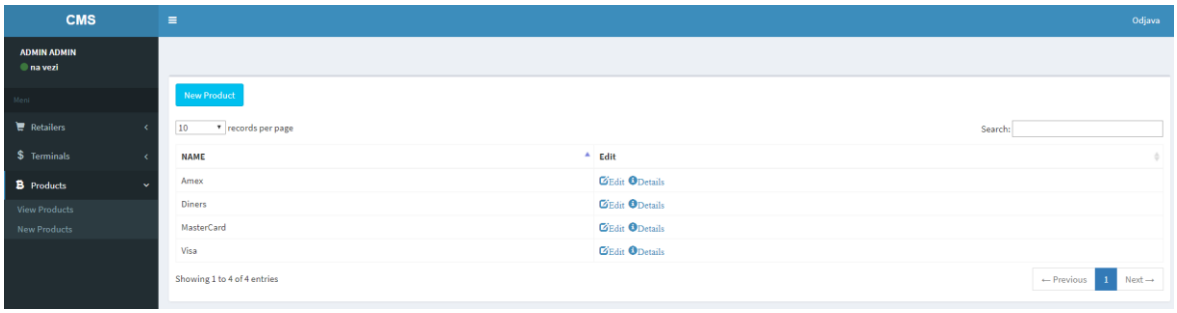
Slika 15. CMS – pregled terminala

Dodavanje novog terminala obuhvaća odabir prodajnog mjesta, naziv terminala, status i partnera.



Slika 16. CMS – dodavanje novog terminal

Pregled proizvoda može vidjeti samo admin sustava, i on je jedini koji može dodavati/uklanjati proizvode.



Slika 17. CMS – pregled proizvoda

4.2. Destinacijske aplikacije (CCA i ICA aplikacije)

Za primanje poruka implementirane su dvije aplikacije CCA(*eng. Croatian Card Authorization*) i ICA(*eng. Italy Card Authorization*). Obe aplikacije su izvedene kao windows aplikacije i to jedna sadrži podatke o Hrvatskim prodajnim mjestima, a druga sadrži podatke o Talijanskim prodajnim mjestima. Njihova namjena je samo prikaz poruka, tj. podataka koje prime od Service Bus-a.

Aplikacije se sastoje od jedne forme sa sljedećim funkcionalnostima:

- Pretraga terminala/prodajnih mjesta
- Pregled terminala/prodajnih mjesta
  - Naziv
  - Adresa
  - Grad
  - Država
  - Broj prodajnog
  - Broj terminala
  - Minimalni broj rata (amex,visa,mastercard)

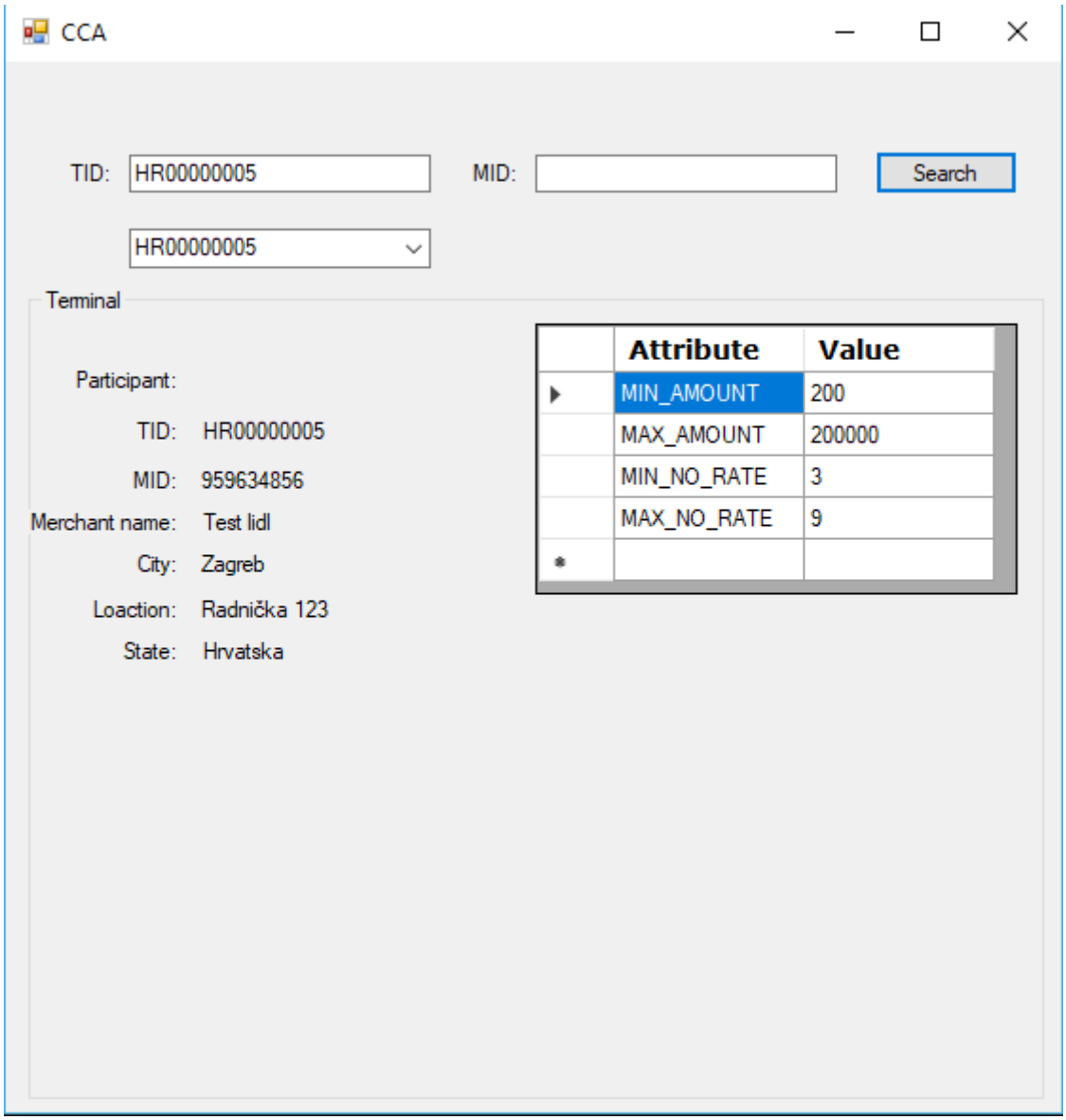
- Maksimalni broj rata (amex,visa,mastercard)
- Minimalni iznos
- Maksimalni iznos

Service Bus-a šalje poruke putem web servisa koji su definirani na klijentima(CCA i ICA). Metoda za primanje poruka na klijentima zove se *ReceiveMessage(string xml).*

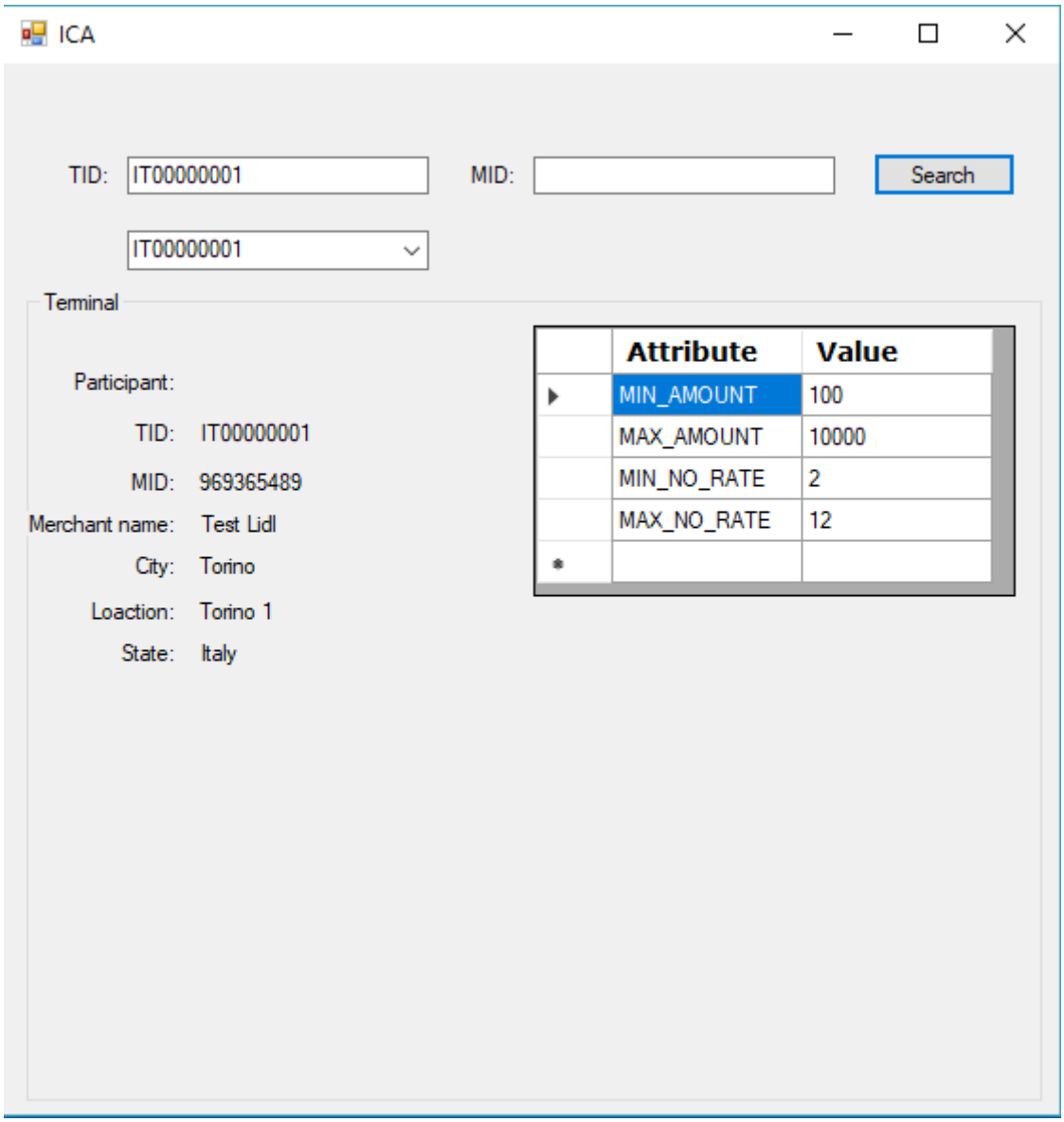
CCA sustav prima podatke isključivo za terminale koji imaju aktivan Amex produkt na CMS-u, a ICA sustav samo podatke za terminale koji imaju aktivan Visa produkt na CMS-u.

```
[WebMethod]
[XmlInclude(typeof(Merchant))]
public Status ReceiveMessage(string xml)
{
MerchantRepository _repo = new MerchantRepository();
XmlSerializer serializer = new XmlSerializer(typeof(Merchant));
Merchant merchant;
using (StringReader reader = new StringReader(xml))
{
merchant = (Merchant)(serializer.Deserialize(reader));
}
Status status = _repo.Insert(merchant);
return status;
}
```

Kôd 1 Web metoda za slanje poruka



Slika 18. CCA– izgled aplikacije



Slika 19. ICA– izgled aplikacije

4.3. Service Bus

U ovom poglavlju opisana je implementacija Service Busa. Projekt je razdvojen u dva segmenta:

- Klijenti
- Server

U segmentu “Klijenti” su definirani sustavi kojima Service Bus može pristupiti. U ovom slučaju to su CMS(izvorna aplikacija), CCA i ICA aplikacija. Sve tri aplikacije komuniciraju putem Service Busa. Na samim klijentima implementirani su web servisi preko kojih Service Bus dohvaća poruke i šalje poruke.

Segment “Server” sastoji se od implementacije Service Busa i to:

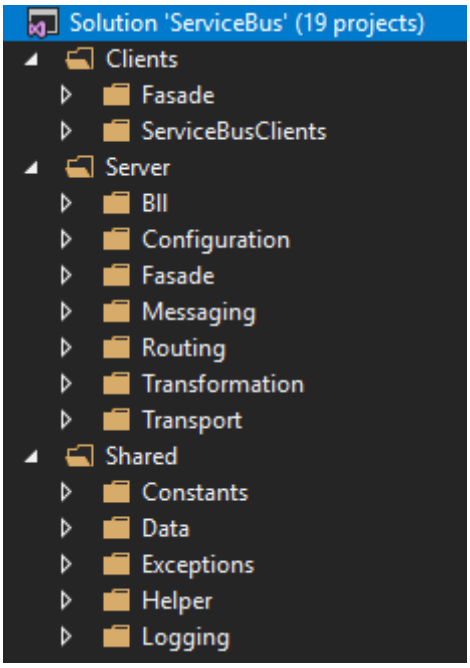
- Messaging - Kreiranje ulazne i izlazne poruke
- Transformation - Pretvaranje ulazne poruke u izlaznu (ulazna i izlazna poruka definirane su xsd-om(za svakog partnera drugačije))
- Transport - Slanje poruke

Service Bus će se izvršavati kao windows servis koji će bit instaliran na lokalnom računalu. Klijent i server čine zajedno Service Bus i oba segmenta će biti podignuta na istom windows servisu.

Funkcionalnosti:

- Komunikacija s izvornom aplikacijom (CMS) putem web servisa
- Dohvat work ordera + podaci o prodajnim mjestima i terminalima na temelju work ordera
- Generiranje ulazne poruke
- Transformiranje ulazne poruke u izlaznu poruku (izlazne poruke su definirane xsd-om za svakog partnera
- Slanje poruka prema destinacijskim sustavima (CCA i ICA)

- Zatvaranje work ordera ako je od destinacijskih sustava dobivena povratna informacija da su poruke uspješno zaprimljene
- Ponavljanje procesa svakih 15 minuta



Slika 20. Service Bus

Service bus ima svoju bazu s 5 tablica:

- SB_MESSAGE - sadrži podatke o ulaznim i izlaznim porukama
- SB_MESSAGE_TRANSFORMATION - sadrži podatke o transformaciji ulaznih u izlazne poruke i njihovih validacijsku shemu.
- SB_MESSAGE_TYPE - sadrži podatke o kakvom je tipu poruke riječ
- SB_USER - podaci o userima koji mogu koristiti Service bus
- SB_USER_PERMISSION - dodavanja prava userima na određeni tip poruke

Slika 21. Service Bus diagram baze

4.3.1. Klijenti

Klijentska strana Service Busa je definirana s klijentima (CMS, CCA, ICA). Service Bus komunicira s njima putem web servisa. Svaki klijent ima implementirane web metode kojima Service Bus ima pristup.

CMS sustav s podacima pruža Service Bus-u 4 metode kojima može pokupiti podatke:

- GetWorkOrders - dohvat work ordera
- GetRetailerData - dohvat podataka o prodajnim mjestima
- GetTerminalData - dohvat podataka o terminalima
- GetTidMidProducts - dohvat podatka o produktima koji su vezani na prodajno mjesto i na terminal

CCA i ICA sustav omogućuju Service Bus-u pristup samo jednoj metodi za primanje poruka:

- ReceiveMessage - primanje poruka u obliku xml-a

4.3.1.1 Web service klijenti

Ako se dogodilo ažuriranje podataka, kreiranje novog prodajnog mjesta ili terminala, tj. svaka promjena nad matičnim podacima bilo riječ o prodajnom mjestu ili terminalu automatski rezultira generiranjem work ordera u statusu "0". Nakon toga Service Bus dohvaća work ordere i kupi informacije koje su mu potrebne, a u ovom slučaju to je "Terminal ID" koji je zapisan u work orderu.

```
[WebMethod]
[XmlInclude(typeof(WorkOrder))]
public Status GetWorkOrders(int partnerId, DateTime dateTime, out
List<WorkOrder> wos)
  {
     wos = new List<WorkOrder>();
Status status=
CmsService.GetInstance().GetWorkOrderByFilter(partnerId,dateTime, out
wos);
 if (status.IsOk())
     return status;
 return new Status(Enums.StatusEnumType.NOK);
 }
```

Kôd 2 Web metoda za dohvat work ordera

Sljedeći korak Service Bus-u je dohvat podataka o terminalu. Za dohvat podataka o terminalu koristimo "Terminal ID" kojeg smo dobili iz work ordera.

```
[WebMethod]
        [XmlInclude(typeof(Terminal))]
```

```
        public Status GetTerminalData(int partner, int terminalId, out
Terminal terminal)
        {
            terminal = new Terminal();
            Status status =
CmsService.GetInstance().GetTerminalByFilter(partner, terminalId, out
terminal);
            if (status.IsOk())
                return status;
            return new Status(Enums.StatusEnumType.NOK);
        }
```

Kôd 3 Web metoda za dohvat podataka o terminalu

Kada smo dobili podatke o terminalu, iz njega izvlačimo “Retailer ID” da bismo dobili podatke o prodajnom mjestu.

```
[WebMethod]
[XmlInclude(typeof(Retailer))]
public Status GetRetailerData(int partner, int retailerId, out Retailer
retailer)
  {
            retailer = new Retailer();
            Status status =
CmsService.GetInstance().GetRetailerByFilter(partner, retailerId, out
retailer);
            if (status.IsOk())
                return status;
            return new Status(Enums.StatusEnumType.NOK);
        }
```

Kôd 4 Web metoda za dohvat podataka o prodajnom

Prilikom kreiranja prodajnog mjesta korisnik je mogao unijeti podatke o produktima s kojima definira koje prodajno mjesto ima pravo korištenja produkata na terminalima. Dakle, terminali koji imaju aktivne produkte, daju mogućnost kupcima da mogu koristiti produkte na njihovim terminalima.

```
        [WebMethod]
        [XmlInclude(typeof(Product))]
        public Status GetTidMidProducts(int terminalId, int retailerId,
out Product product)
        {
            product = new Product();
            Status status =
CmsService.GetInstance().GetTidMidProducts(terminalId, retailerId, out
product);
            if (status.IsOk())
                return status;
            return new Status(Enums.StatusEnumType.NOK);
        }
```

Kôd 5 Web metoda za dohvat podataka o produktima na prodajnom i terminalu

CCA i ICA sustavi imaju implementiranu samo jednu metodu za primanje poruka. S obzirom na to da CMS sustav sadrži podatke bitne njima, CCA sustav očekuje podatke samo za terminale koji imaju Amex produkt aktivan na CMS-u, a ICA sustav očekuje samo podatke za terminale koji imaju Visa produkt aktivan na CMS-u.

```
[WebMethod]
        [XmlInclude(typeof(Merchant))]
        public Status ReceiveMessage(string xml)
        {
            MerchantRepository _repo = new MerchantRepository();
            XmlSerializer serializer = new
XmlSerializer(typeof(Merchant));
            Merchant merchant;
            using (StringReader reader = new StringReader(xml))
            {
               merchant = (Merchant)(serializer.Deserialize(reader));
            }
            Status status = _repo.Insert(merchant);

            return status;
        }
```

Kôd 6 Web metoda za primanje poruka

4.3.2. Server

Segment "Server" je zadužen za kreiranje poruka, transformiranje poruka i slanje poruka na destinacijske sustave. Od mnogobrojnih funkcionalnosti koje Service Bus posjeduje u ovom poglavlju izdvojene su jedne od važnijih funkcionalnosti Service Bus-a, a to su:

- Kreiranje poruka (ulazne/izlazne)
- Validacija poruka
- Transformiranje poruka

- Slanje poruka (transport)

4.3.2.1 Poruke

Kod kreiranja poruka razlikujemo ulazne poruke od izlaznih. Ulazne poruke su sirovi podaci dohvaćeni s CMS sustava, a izlazne poruke su definirane od strane partnera i zapisane su u obliku xsd-a. Temeljem xsd-a radi se validacija izlazne poruke. Destinacijski sustavi CCA i ICA očekuju xml poruku koja je definirana xsd-om zapisanim u bazi podataka.

```
<xs:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="Merchant">

      <xs:complexType>

            <xs:sequence>

                  <xs:element type="xs:string" name="TID"/>

                  <xs:element type="xs:string" name="MID"/>

                  <xs:element type="xs:string" name="MERCHANT_NAME"/>

                  <xs:element type="xs:string" name="CITY"/>

                  <xs:element type="xs:string" name="ADDRESS"/>

                  <xs:element type="xs:string" name="STATE"/>

                  <xs:element type="xs:string" name="ZIP_CODE"/>

                  <xs:element type="xs:string" name="MIN_AMOUNT"/>

                  <xs:element type="xs:string" name="MAX_AMOUNT"/>

                  <xs:element type="xs:string" name="MIN_NO_RATE"/>

                  <xs:element type="xs:string" name="MAX_NO_RATE"/>

                  <xs:element type="xs:string" name="STATUS"/>

            </xs:sequence>

      </xs:complexType>

</xs:element>
```

```
</xs:schema>
```

Kôd 7 XSD za izlaznu poruku

Validacija poruka se izvršava nakon kreiranja ulazne poruke i nakon kreiranja izlazne poruke. Ako validacija nije valjana Service Bus prekida obradu za work order koji je u procesiranju, zapisuje grešku u log i nastavlja s daljnjim procesiranjem sljedećeg work ordera.

```
      public static bool ValidateXml(string xml, string xsd,
out List<string> errors)
      {
            errors = null;
            XmlValidation validation = new XmlValidation();
            XmlSchemaSet shemas = new XmlSchemaSet();
            shemas.Add(
                  XmlSchema.Read(new
StringReader(xsd),validation.eventHandler));
            XDocument.Parse(xml).Validate(shemas,validation.e
ventHandler);
            if(!valdiation.p_valid){
                  errors= validation.Errors;
            }
            return valdiation.p_valid;
      }
```

Kôd 8 Validacija ulaznih poruka

4.3.2.2 Transformacija

Transformiranjem poruke ulazne u izlaznu daje do znanja da je izlazna poruka uspješno generirana i da je spremna za slanje prema sustavima. Kad se poruka ne transformira je znak da je greška i ili u ulaznim podacima ili je nastala promjena na definiciji klase(OutputMessage).

```
public interface IXmlTransformator
{
	string Transform(string xml);
}
```

Kôd 9 Interface IXmlTransformator

Primjer transformiranja ulazne poruke za CCA sustav:

```
public class TransformCroatia : IXmlTransformator
{
	public string Transform(string xml)
	{
		XmlSerializer serializer = new XmlSerializer(typeof(InputMessage));
		InputMessage input = (InputMessage)serializer.Deserialize(new StringReader(xml));
		OutputCroatiaMessage output = new OutputCroatiaMessage()
		{
			TID = input.TerminalDetails.Name,
			MID = input.RetailerDetails.Name,
			MERCHANT_NAME = input.RetailerDetails.MerchantName,
			CITY = input.RetailerDetails.City,
```

```
                ADDRESS = input.RetailerDetails.Address,
                STATE = input.RetailerDetails.State,
                ZIP_CODE = input.RetailerDetails.ZipCode,
                PRODUCT  = input.RetailerDetails.ProductName,
                MIN_AMOUNT = input.RetailerDetails.MinAmnt,
                MAX_AMOUNT = input.RetailerDetails.MaxAmnt,
                MIN_NO_RATE = input.RetailerDetails.MinNoRate,
                MAX_NO_RATE = input.RetailerDetails.MaxNoRate,
                STATUS = input.TerminalDetails.Status,
            };
            serializer = new XmlSerializer(typeof(OutputCroatiaMessage));
            StringWriter writer = new StringWriter();
            serializer.Serialize(writer,output);
            return writer.ToString();
        }
}
```

Kôd 10 . Transform Croatia

4.3.2.3 Transport

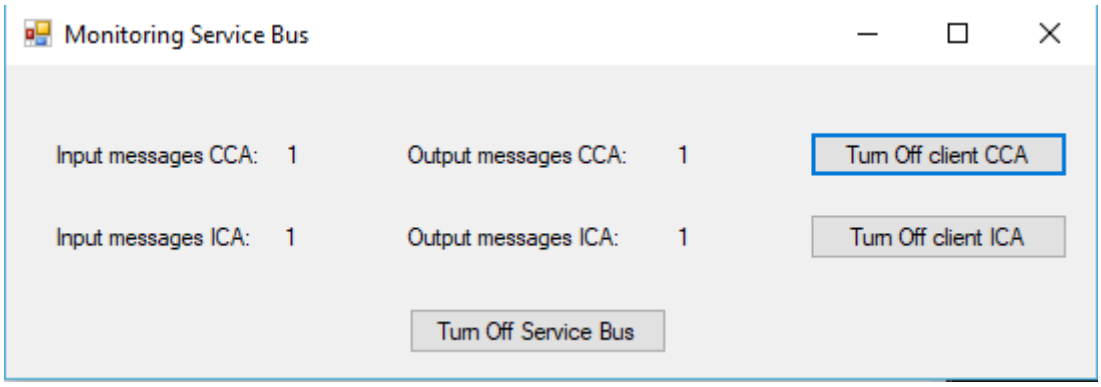
Slanje poruka putem web servisa opisano je u prethodnim poglavljima. Metoda *ReceiveMessage(string xml)* definirana je na klijentima. Service Bus prilikom kreiranja izlaznih poruka šalje te iste poruke prema destinacijskim sustavima. Kada destinacijski sustav primi poruku i uspješno je obradi, vraća status(“OK”/”NOK”) prema Service Bus-u. Ako je status “OK” Service Bus putem web servisa zatvara work order na CMS-u u status “C”, da se kod sljedećeg procesiranja ne kreira opet ista poruka za isti work order. Ako je status “NOK” Service Bus će prilikom sljedećeg procesiranja pokušati ponovno poslati poruku prema destinacijskom sustavu.

4.4. Monitoriranje sustava

Monitoring sustava prikazuje u stvarnom vremenu broj kreiranih ulaznih i izlaznih poruka. Isto tako ima mogućnost gasiti i paliti klijente, tj. Stopirati kreiranje i slanje poruka prema destinacijskim aplikacijama. Aplikacija ima pristup bazi Service Bus-a i pravo na dohvat broja kreirani ulaznih i izlaznih poruka.

Monitoriranje sustava sastoji se od:

- Praćenje ulazno/izlaznih poruka za svakog partnera
- Gašenje/Paljenje klijenata
- Gašenje/Paljenje Service Bus-a



Slika 22. Monitoring Service Bus

Zaključak

Komunikacija između distribuirani h sustava bez Service Bus-a je moguća, ali uz jako veliki rizik od krađe podatka, što može uzrokovati velike probleme za opstojanje tvrtki. Prolazeći kroz ovakav način komunikacije, gdje jedino Service Bus prenosi informacije može se zaključiti da je takav koncept isplativ. Service Bus u prikazanom radu testiran je na maloj količini podataka i pokazao se vrlo uspješnim. Dohvat podataka s izvorišnog sustava se odmah prilikom pokretanja kešira u memoriju što znatno ubrzava rad aplikacije jer nakon toga svaka igra s podacima je jednostavna jer ih imate na jednom mjestu. Definiranjem izlaznih poruka u shemi (xsd) omogućuju upravljanje i dodavanje novih uzoraka bilo da je riječ o novom sustavu ili novoj poruci za postojeći sustav. Sva ta konfiguracija je zapisana u bazi Service Busa. Činjenica je da su mali podaci male brige, veliki podaci velike brige. Iz toga se postavlja pitanje: Koliko je isplativo nadograđivati postojeći sustav koji radi s velikom količinom podataka? Baš zbog takvih pitanja Service Bus je idealno rješenje za velike tvrtke, jer dodatno usporavanje velikih aplikacija/sustava prouzrokuje gubitak novca.

Popis kratica

CCA	*Croatia Card Authorization*	aplikacija za autoriziranje transakcija
ICA	*Italy Card Authorization*	aplikacija za autoriziranje transakcija
CMS	*Content management system*	aplikacija koja sadrži podatke
XSD	*XML Schema Definition*	opisivanje elemenata u XML-u
ESB	*Enterprise Service Bus*	služi za razmjenjivanje informacija
SOA	*Service Oriented Architecture*	stil razvoja poslovanja
HTTP	*Hyper Text Transfer Protocol*	metoda prijenosa informacija

Popis slika

Popis kôdova

Pogreška! Za dodavanje Table of Figures tekstu koji želite da se ovdje pojavi koristite karticu Polazno.

Literatura

[1] Distribuirani sustavi, https://en.wikipedia.org/wiki/Distributed_computing, srpanj 2018

[2] Mikroservisi, https://medium.freecodecamp.org/microservices-from-idea-to-starting-line-ae5317a6ff02, srpanj 2018

[3] Enterprise Service Bus, https://www.mulesoft.com/resources/esb/what-esb, srpanj 2018

[4] Esb vs microservices, https://blogs.mulesoft.com/dev/microservices-dev/microservices-versus-esb/, srpanj 2018

[5] SOA, https://www.ibm.com/support/knowledgecenter/en/SSMQ79_9.5.1/com.ibm.egl.pg.doc/topics/pegl_serv_overview.html, srpanj 2018

[6] ESB vs SOA, https://blogs.mulesoft.com/dev/connectivity-dev/esb-vs-soa/, srpanj 2018

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, 15.Kolovoza.2018.