SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Paskal Šimec

ANALIZA METODA KOMUNIKACIJA IZMEĐU PROGRAMSKIH SUSTAVA

Varaždin, 2020.

*Ovaj rad izrađen u Laboratoriju za Web arhitekture, tehnologije, servise i sučelja na Fakultetu organizacije i informatike u Varaždinu pod vodstvom prof. dr. sc. Dragutina Kermeka i predan je na natječaj za dodjelu Rektorove nagrade u akademskoj godini 2019./2020.*

# Sadržaj

[1. Uvod 1](#_Toc48467109)

[1.1. Opći i specifični ciljevi 1](#_Toc48467110)

[2. Metode komunikacija programskih sustava 3](#_Toc48467111)

[2.1. Protokoli za razmjenu poruka 3](#_Toc48467112)

[2.1.1. MQTT 3](#_Toc48467113)

[2.1.2. WebSocket 4](#_Toc48467114)

[2.2. Asinkrono komuniciranje 6](#_Toc48467115)

[2.2.1. Ajax 6](#_Toc48467116)

[2.2.2. JMS 8](#_Toc48467117)

[2.3. Web servisi 10](#_Toc48467118)

[2.3.1. Soap 11](#_Toc48467119)

[2.3.2. RESTful 15](#_Toc48467120)

[2.3.3. Usporedba web servisa 17](#_Toc48467121)

[3. Usporedna analiza metoda komunikacije 20](#_Toc48467122)

[4. Implementacija sustava 23](#_Toc48467123)

[4.1. Korištena tehnologija i alati 23](#_Toc48467124)

[4.1.1. Razvojno okruženje 23](#_Toc48467125)

[4.2. Arhitektura sustava 25](#_Toc48467126)

[4.2.1. Struktura projekta 25](#_Toc48467127)

[4.2.2. Baza podataka 26](#_Toc48467128)

[4.2.3. Web servisi sustava 27](#_Toc48467129)

[4.2.4. Konfigurabilnosti i lokalizacija sustava 29](#_Toc48467130)

[4.3. Metode komunikacije unutar sustava 30](#_Toc48467131)

[4.4. Primjer korištenja sustava 36](#_Toc48467132)

[5. Rasprava 41](#_Toc48467133)

[6. Zaključak 43](#_Toc48467134)

[Popis literature 44](#_Toc48467135)

[Popis slika 46](#_Toc48467136)

[Popis tablica 47](#_Toc48467137)

[Sažetak 48](#_Toc48467138)

[Summary 49](#_Toc48467139)

1. Uvod

U današnje vrijeme razvoj programskih sustava postao je glavna djelatnost IT sektora. Gotovo svaki posao, nevezano za zanimanje, ima vlastiti ili koristi tuđi programski sustav kako bi pojednostavio i unaprijedio svoje poslovanje. Baš iz tog razloga, programski se sustavi ubrzano razvijaju i međusobno integriraju. Kako bi se ostvario jedan veliki sustav koji se sastoji od više manjih komponenti nužnih za održavanje poslovanja, potrebno je na pravilan način integrirati i povezati postojeće sustave. Najčešći način međusobnog povezivanja jest komunikacija između više različitih sustava.

U ovom radu prikazani su neki od najpoznatijih načina povezivanja sustava metodama komunikacije te su na praktičnom primjeru predočene razne mogućnosti integriranja više manjih sustava u jedan veći s jedinstvenim ciljem.

Rad je podijeljen u pet glavnih poglavlja: **(1) Uvod** u kojem je prezentiran kratak uvid u temu rada te su prikazani opći i specifični ciljeva rada, **(2) Metode komunikacija programskih sustava** gdje su predstavljene i opisane metode komunikacije između programskih sustava, **(3) implementacija sustava** koji sadrži arhitekturni i grafički prikaz izrađene aplikacije koja se temelji na ranije opisanim metodama komunikacije sustava te sadrži isječke programskog koda, **(4) rasprava** gdje su dana dodatna razmišljanja o proširenju implementiranog sustava, povećanju stupnja integriranosti i produkcijskoj uporabi sustava, te analiza osobina na bazi slanja i prijema većeg broja poruka i **(5) zaključak** kao posljednje poglavlje.

* + - 1. Opći i specifični ciljevi

Razvoj web aplikacija češće se koristi od starih stolnih (engl. *Desktop*) aplikacija. Najveći razlog tome je veća dostupnost i lakša implementacija web aplikacija u odnosu na desktop aplikacije. Kako bi web aplikacije bile u toku s današnjom tehnologijom i ne bi zaostajale, kao što se to desilo se desktop aplikacijama, moraju imati određenu mogućnost prilagođavanja i ponovnog korištenja. Uzmimo za primjer da se u skorije vrijeme razvije nova popularna platforma koja zahtjeva posve novu implementaciju i nema sličnih elemenata kao što su imale dosadašnje web aplikacije: na takvu platformu će se moći brzo i jednostavno prilagoditi samo one aplikacije koje imaju visoku stopu prilagodljivosti koja se postiže brojnim komponentama koje međusobno djeluju i komuniciraju i koje su podržane s jediničnim testovima.

Cilj ovog rada je analiza načina povezivanja više programskih sustava s ciljem zajedničkog komuniciranja i integracije u veći sustav. Rad obuhvaća teorijsku definiciju popularnih principa komunikacije, razvojnih okvira i protokola, te njihovu usporedbu i mogućnosti integracije. Kako bi se pokazalo da se baš odabrani aspekti mogu međusobno integrirani, izrađen je sustav koji se temelji na principu ponovne iskoristivosti, a integrira više manjih komponenti u jedan relativno složen sustav s jedinstvenim ciljem.

1. Metode komunikacija programskih sustava

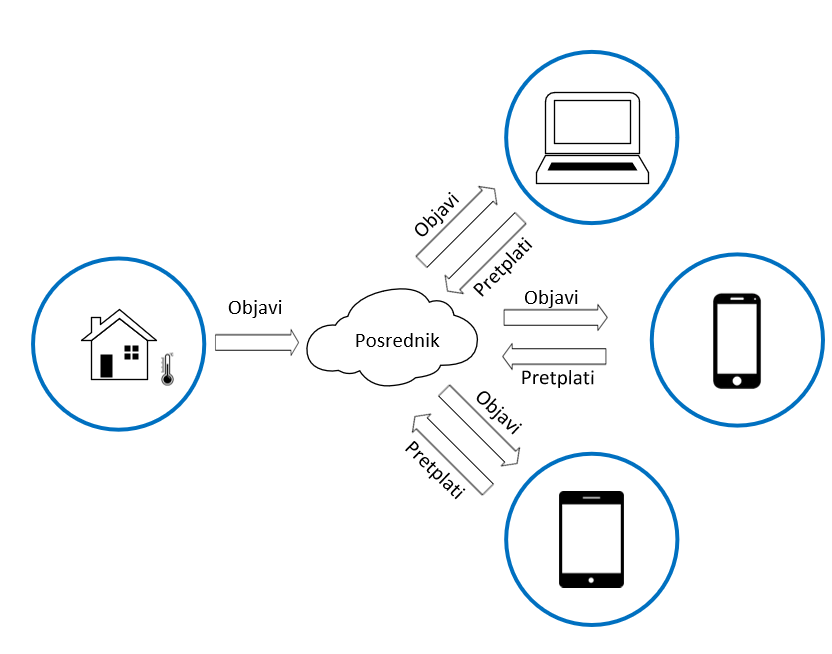
U ovom poglavlju opisane su poznatije metode za komunikaciju programskih sustava, način rada svake metode te usporedba određenih metoda. Svaka navedena metoda je kasnije primijenjena unutar implementiranog sustava.

* 1. Protokoli za razmjenu poruka
     1. MQTT

MQTT (engl. *Message Queuing Telemetry Transport*) je lagan, otvoren i jednostavan klijent-server protokol principa Objavi/Pretplati (engl. *Publish/Subscribe*), dizajniran da se lako implementira. Sama činjenica da je protokol lagan i jednostavan čini ga izvrsnim za upotrebu kod komunikacije tipa M2M (engl. *Machine to Machine*) ili u danas sve popularnijem IoT-u (engl. *Internet of Things*). MQTT djeluje iznad TCP/IP protokola. [1]

MQTT je lakši od HTTP 1.1. protokola te je veoma interesantna opcija kada se podaci moraju slati i primati u gotovo stvarnom vremenu pomoću Objavi/Pretplati modela. Osim navedene primjene u IoT i M2M projektima, u današnje vrijeme MQTT zbog svojih povoljnih karakteristika ima sve češću primjenu u web i mobilnim aplikacijama koje zahtijevaju sigurnu razmjenu i učinkovitu distribuciju poruka.

Već ranije navedeni model Objavi/Pretplati sastoji se od posrednika (engl. *Broker*) s kojim svi klijenti moraju ostvariti vezu. Klijent koji je ostvario vezu s posrednikom i šalje mu određene podatke još se naziva i izdavač (engl. *Publisher*). Zadaća posrednika je da filtrira dolazne poruke i distribuira ih klijentima koji su zainteresirani za takav tip poruke. Klijente koje zanima određeni tip poruke i pretplate se na njih nazivaju se subscribers. Kako bi se komunikacija odvijala bez prepreka, *publisher* i *subscriber* moraju ostvariti vezu prema posrednikom. [2] Slika 1. prikazuje opisani model Objavi/Pretplati.



Slika : MQTT Objavi/Pretplati model (autorski rad)

Ranije spomenuti posrednik odnosno poslužitelj sastoji se od relativno komplicirane logike i implementacije. Iz tog razloga, preporuča se korištenje nekih od već implementiranih poslužitelja koje podržavaju MQTT poruke, kao što su: Apache ActiveMQ Apollo, Mosquitto, Bevywise MQTTRoute i RabbitMQ.

* + 1. WebSocket

Kod tradicionalnog HTTP zahtjeva, klijent traži resurse, a poslužitelj ih pruža. Uz ograničenja da razmjenu uvijek mora inicirati klijent, dok poslužitelj ne može poslati nikakve podatke bez da ga klijent prvi zatraži. Ovakav opisani model bio je prihvatljiv dok su klijenti slali zahtjeve za sadržajima koji se rijetko mijenjaju. U današnje vrijeme takav je pristup sve rjeđi, jer klijenti očekuju interaktivne i dinamičke sadržaje koji automatski podrazumijevaju potražnju za većim brojem zahtjeva za sadržajem koji se često mijenja.

WebSocket protokol rješava taj problem pružajući full-duplex komunikacijski kanal između klijenta i poslužitelja. U kombinaciji s drugim tehnologijama, kao što su JavaScript i HTML5, WebSocket omogućuje web aplikacijama pružanje bogatijeg korisničkog iskustva. [3]

U aplikacijama koje se temelje na WebSocket-u, poslužitelj objavljuje krajnju točku WebSocket-a, a klijent koristi točku za povezivanje s poslužiteljem. Sam protokol je simetričan što omogućuje da nakon uspostavljanja veze, klijent i poslužitelj mogu međusobno slati poruke u bilo kojem trenutku dok je veza otvorena, a vezu mogu zatvoriti kada im to odgovara. Klijenti se obično povezuju samo na jednog poslužitelja, a poslužitelji prihvaćaju vezu s više klijenata. [3]

WebSocket protokol je protokol temeljen na TCP-u. Jedina poveznica prema HTTP-u je početno uspostavljanje veze, tzv. rukovanje (engl. *Handshake*). U pravilu, WebSocket protokol koristi port 80 za regularne veze, a port 443 za TLS (engl. *Transport Layer Security*). [4]

WebSocket protokol sastoji se od rukovanja i prijenosa podataka. Klijent inicira rukovanje slanjem zahtjeva krajnjoj točki WebSocket-a koristeći njen URI. Primjer rukovanja od klijenta ovako izgleda:

GET /path/to/websocket/endpoint HTTP/1.1

Host: localhost

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: xqBt3ImNzJbYqRINxEFlkg==

Origin: http://localhost

Sec-WebSocket-Version: 13

Primjer rukovanja od poslužitelja kao odgovor na zahtjev klijenta :

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: K7DJLdLooIwIG/MOpvWFB3y3FE8=

Platforma Java EE (engl. *Java Enterprise Edition*) uključuje Java API za WebSocket koji omogućuje stvaranje, konfiguriranje i raspoređivanje krajnjih točaka WebSocket u web aplikacijama. WebSocket API također nudi i pristup udaljenim krajnjim točkama WebSocket-a iz bilo koje Java aplikacije što pruža brojne mogućnosti prilikom implementacije web aplikacije.

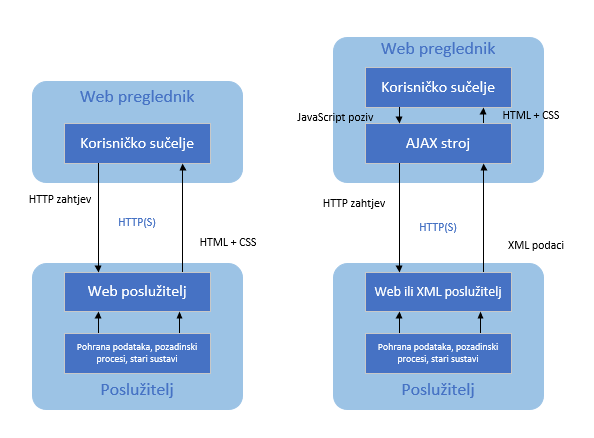
* 1. Asinkrono komuniciranje

Asinkrono komuniciranje predstavlja princip u kojem pružatelj može isporučiti poruke klijentu kako one stižu, dok s druge strane klijent ne treba zahtijevati poruke kako bi ih primio. Na opisanom principu rade brojni razvojni okviri te se temelje razne aplikacije. Jedna od glavnih osobina asinkronog komuniciranja je dodatna sloboda klijenta i povećanje korisničkog iskustva. U nastavku su opisani AJAX razvojni okvir i JMS, koji svoje funkcionalnosti temelje na asinkronom komuniciranju.

* + 1. Ajax

U klasičnom modelu web aplikacija korisničke radnje pokreću HTTP zahtjev prema web serveru, koji odgovara traženom HTML stranicom. Ovakav je pristup logičan s tehničke strane, ali ne i s korisničke. Kako bi korisnik vidio nove podatke, odnosno pristigli HTML odgovor, mora ručno osvježiti stranicu što zasigurno ne utječe dobro na korisničko iskustvo. Jedan od odgovora na objašnjenu problematiku je AJAX.

AJAX (engl. *Asynchronous JavaScript and XML*) nije programski jezik, već način korištenja postojećih standarda kao što su JavaScript, XML, DHTML i DOM. Radi na principu, slanja zahtjeva na poslužiteljsku stranu koju zahtjeva određeni klijent, a potom obrađuje i vraća podatka korisniku putem JavaScript-a. Ovakav način rada uvelike smanjuje opterećenje na strani poslužitelja i vrijeme čekanja na strani klijenta te u konačnici poboljšava korisničko iskustvo.



Slika : Usporedba tradicionalnog i AJAX poziva [5]

Slika 2 dio a) prikazuje usporedbu tradicionalnog i AJAX poziva kod web aplikacija. Kod tradicionalnih web aplikacija, kao što je i ranije opisano, korisnik akcijom pokreće HTTP zahtjev web serveru, koji nakon procesiranja zahtjeva vraća HTTP odgovor koji će korisniku biti vidljiv tek nakon akcije osvježavanja ekrana (engl. *refresh*). Dio slike s oznakom b) prikazuje novi način rada – AJAX poziv. Kod AJAX-a na strani klijenta nalazi se i JavaScript podrška koja se izvršava unutar preglednika. JavaScript upravlja podacima koje korisnik unese, prikazuje potreban sadržaj i upravlja svim ostalim zahtjevima na strani klijenta. U slučaju da treba dohvatiti nove podatke, šalje asinkroni HTTP zahtjev za web poslužitelj. Web poslužitelj obrađuje zahtjev i vraća XML dokument koji sadrži rezultat koji JavaScript procesira i prikazuje korisniku [5]. U današnje vrijeme dokument u XML formatu se često zna zamijeniti sJSON formatom. JSON (engl. *JavaScript Object Notation*) je lagani format za razmjenu podataka koji se jednostavnije generira i interpretira od XML-a što omogućuje AJAX aplikacijama jednostavniju prezentaciju podataka. Trenutno je JSON najpopularniji format za razmjenu podataka. Tablica 1 prikazuje usporedbu vremena slanja poruka u XML i JSON formatu, iz koje je vidljivo da su vremena za JSON format znatno manja u odnosu na XML [6].

Tablica : Usporedba vremena prijenos podataka JSON i XML [6]

|  |  |  |
| --- | --- | --- |
| Količina podataka | Vrijeme (ms) | |
| XML | JSON |
| 100 | 23.16 | 16.07 |
| 200 | 24.45 | 16.65 |
| 500 | 40.14 | 27.55 |
| 800 | 44.36 | 33.08 |
| 1000 | 69.35 | 47.25 |
| 2000 | 120.15 | 78.45 |

AJAX nudi 3 osnovna principa asinkrone komunikacije. Prvi princip, „polling“ omogućuje web klijentima da šalju zahtjeve u pravilnim intervalima, a poslužitelj po primitku zahtjeva, odmah šalje odgovor i zatvara vezu. Drugi princip „long-polling“ se razlikuje po tome što poslužitelj ostavlja otvorenu vezu na dulji period. Zadnji princip „streaming“, u kojem poslužitelj ostavlja otvorenu vezu prema klijentu sve tok ju on sam ne zatvori. [7]

Nakon detaljnog opisa AJAX tehnologije vidi se da dominiraju samo njegove dobre strane. Naravno, kao i s drugim razvojnim okvirima, postoje i loše strane. Glavni nedostatak AJAX-a je to što aplikaciji u koju je integriran daje dodatnu složenost. AJAX je sam po sebi kompliciran i integracijom sa složenom aplikacijom kompleksnost dodatno raste. Drugi nedostatak je razlika u implementaciji JavaScripta na različitim preglednicima te problem s preglednicima koji ne podržavaju AJAX [5].

* + 1. JMS

JMS (engl. *Java Message Service*) API omogućava aplikacijama da kreiraju, šalju, primaju i čitaju poruku uz pouzdanu, asinkronu i slabo povezano komunikaciju. JMS minimizira skup pojmova koji programer treba naučiti kako bi koristio proizvode za razmjenu poruka, ali pruža dovoljno mogućnosti za podršku sofisticiranih aplikacija za razmjenu poruka. [8]

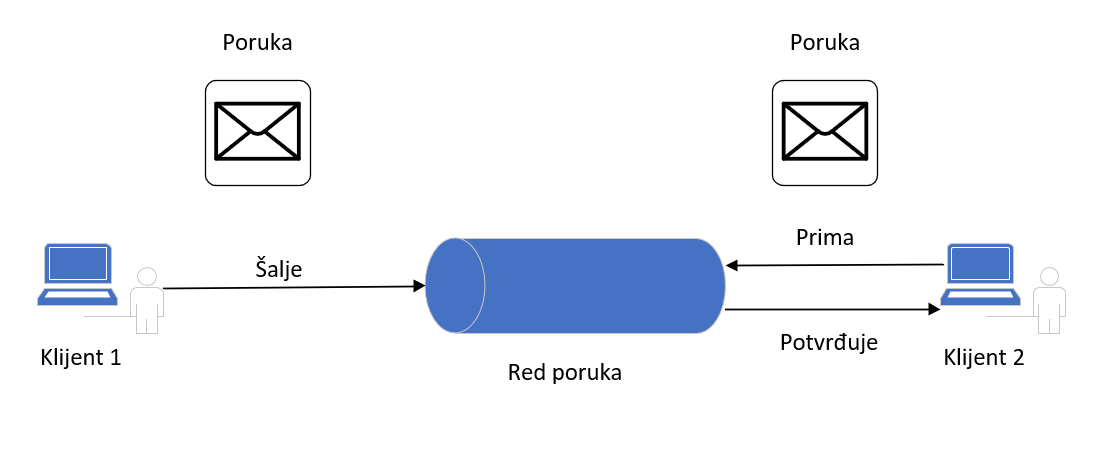
JMS je dio je JEE (engl. *Java Platform Enterprise Edition*), koja omogućuje stvaranje rješenja za razvoj, upotrebu i upravljanje višeslojnim aplikacijama usmjerenim na poslužitelje. JEE koristi Java platformu, standardno izdanje za širenje cjelovite, skalabilne, stabilne, sigurne i brze Java platforme na razini poduzeća. Sama činjenica da je JMS dio JEE znači da se aplikacije koje koriste JMS mogu izvoditi na bilo kojoj platformi koja ima Java virtualni stroj što otvara šire mogućnosti za aplikacije [9]. U novije vrijeme JMS mijenja naziv u *Jakarta Messaging*, dok se JEE popularno naziv Jakarta EE.

JMS aplikacija svoju arhitekturu temelji na četiri osnovne komponente:

* JMS poslužitelj – sustav za razmjenu poruka koji implementira JMS sučelja i pruža administrativne i kontrolne osobine
* JMS klijent – su programi ili komponente, napisane u programskom jeziku Java, koje primaju i šalju poruke
* Poruke – objekti koji komuniciraju informacijama između JMS klijenata
* JMS objekti ili anotacije- služe za bolju integraciju i implementaciju JMS destinacija i veza prema klijentima

Prije postojanja samog JMS API-ja, većina srodnih proizvoda za razmjenu poruka podržavala je ili PTP (engl. *point-to-point*) ili Objavi/Pretplati stil. Kako bi JMS API bio konkurentan drugim proizvodima za razmjenu poruka, JMS implementira oba spomenuta stila. Također, omogućuje dodatnu mogućnost korištenja istog programskog koda za oba navedena stila. Ta mogućnost aplikacijama osigurava dodatnu fleksibilnost i višekratno korištenje. [8]

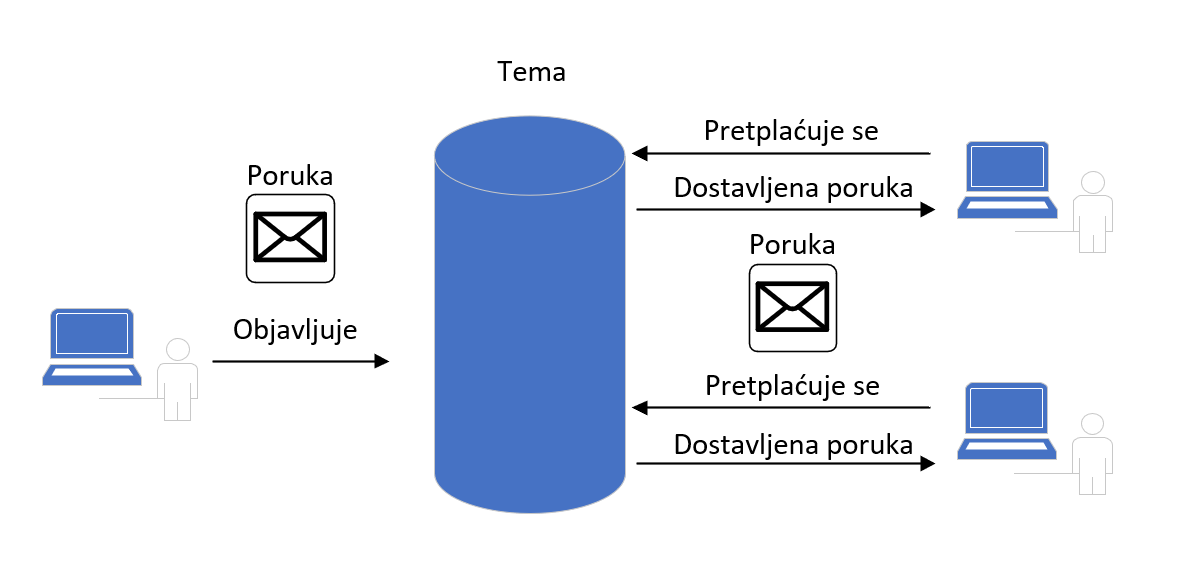
Stil PTP temelji se na pošiljatelju, primatelju te na redu poruka. Svaka poruka adresirana je na specifičan red, a klijent koji primaju poruke izdvajaju poruke iz reda koji je postavljen da čuva njihove poruke. Arhitektura P2P stila prikaza na je slici 3.



Slika : P2P pristup [8]

Osobine PTP stila su da svaka poruka ima samo jednog primatelja, dok primatelj može dohvatiti poruku bez obzira je li radio kada je pošiljatelj poslao poruku ili nije.

Drugi stil naziva Objavi/Pretplati temelji se na pristupu adresiranja poruka za temu, koja djeluje poput oglasne ploče. Izdavači i pretplatnici mogu dinamički objavljivati ​​ili se pretplatiti na željenu temu. Sustav brine o distribuciji poruka koje stižu od više izdavača prema više pretplatnika. Na slici ispod nalazi se arhitektura pub/sub stila.

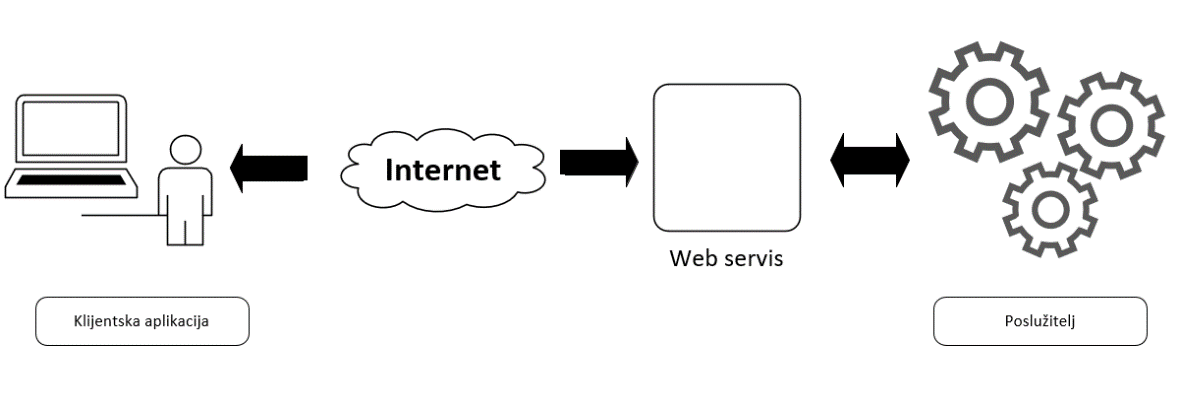


Slika 5: Objavi/Pretplati pristup [8]

Glavne osobine stila su da svaka poruka može imati više primatelja, dok klijent koji se pretplati na temu može konzumirati samo one poruke koje su poslane nakon što se klijent pretplatio te mora nastaviti biti aktivan kako bi mogao konzumirati poruke. [8]

* 1. Web servisi

Prije upoznavanja sa SOAP i RESTful web servisima potrebno je objasniti pojam i značenje web servisa te njihovu primjenu. Web servis je mrežno sučelje izgrađeno korištenjem standardnih web tehnologija. Drugim riječima, ako se aplikacija može koristiti putem interneta pomoću protokola kao što su HTTP, XML, SMTP onda se radi o web servisu. [10]



Slika : Tok komunikacije s web servisom [10]

Kao što je i vidljivo iz slike 6, web servis je komponenta na internetu koja se nalazi između korisnika interneta te same logike. Moglo bi se reći da web servis skriva kompliciranu poslovnu logiku i nudi korisniku jednostavan način njenog korištenja.

Jedna od najbitnijih osobina, koja je i razlog upoznavanja sa servisima, je mogućnost komuniciranja pomoću poruka. Njihova glavna, a moglo bi se reći i jedina zadaća je slanje i primanje poruka preko interneta poštujući određene protokole. Primanje i slanje poruka je jasno definirano, svaki servis ima definirane ulazne i izlazne argumente. [10]

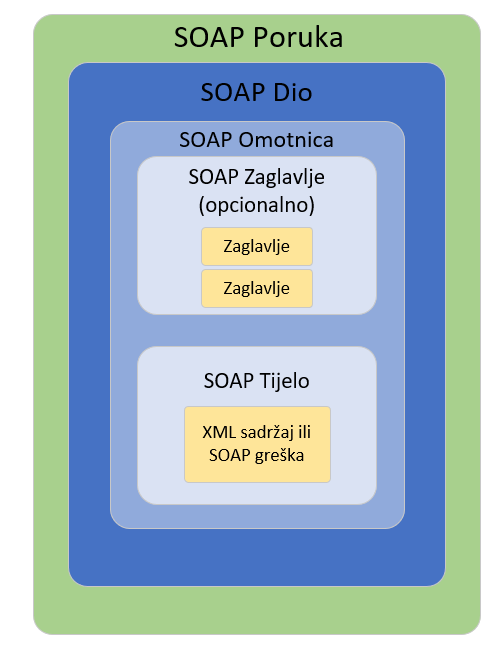
Nakon opisa web servisa i njihove glavne zadaće, postavlja se pitanje kako krajnji korisnici mogu saznati, odnosno doći do servisa kojeg trebaju. Kako si se moglo pristupiti informacijama servisa pa tako i samom servisu, servis treba biti objavljen u nekom od javno dostupnih registara. Jedan od popularnijih je UDDI registar.

*Universal Description, Discovery, and Integration* (UDDI) je mehanizam za registriranje i pretraživanje javno dostupnih servisa. IBM i Microsoft u razvili *Web Services Inspection Language* (WSInspection) koja je alternativa UDDI. [10]

* + 1. Soap

SOAP (engl. *Simple Object Access Protocol*) je lagani protokol namijenjen za razmjenu strukturiranih informacija u decentraliziranom, udaljenom okruženju. Službene stranice uz SOAP servis povezuju i pojmove: pouzdanost, sigurnost, povezanost, usmjeravanje, jednostavnost. [11]

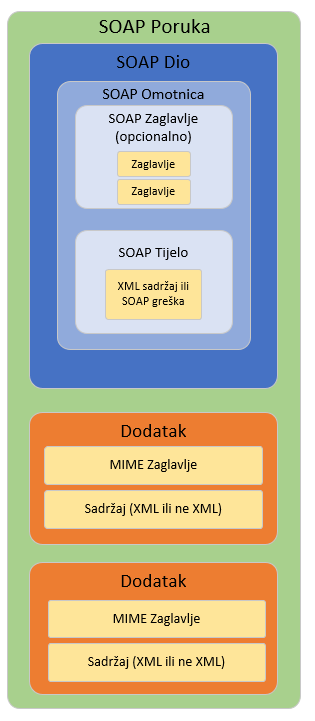
Glavni dio SOAP-a je SOAP poruka koja je prikazana na slici ispod.



Slika : SOAP poruka *[[1]](#footnote-1)*

Kao što je vidljivo iz slike, SOAP poruka je zapravo dobro strukturiran XML dokument. Takav dokument sastoji se od omotnice (engl. *Envelope*) koja u sebi sadrži opcionalno zaglavlje i obavezno tijelo poruke. Zaglavlja (engl. *Header*) su primarni mehanizmi proširivosti SOAP-a, koji omogućuju jednostavno dodavanje novih protokola, autentifikacija i autorizacija, upravljanja transakcijama, obrada plaćanja, praćenje i revizija, i drugo [12]. Element tijela poruke je glavni dio SOAP poruke. On sadrži jednostavan mehanizam za razmjenu obaveznih podataka namijenjenih krajnjem primatelju poruke. Glavna poruka sadržana je u XML formatu. Element tijela poruke sadrži strukturirane ulazne i izlazne argumente ili opis mogućih grešaka prilikom izvođenja.

Postoji, također i mogućnost slanja SOAP poruka koje sadrže dodatke. Takve poruke koje sadrže odgovore koji nisu u XML obliku moraju biti definirani unutar dijela za dodatke (engl. *AttachmentPart*)*.* Primjer takve omotnice nalazi se na slici 7.

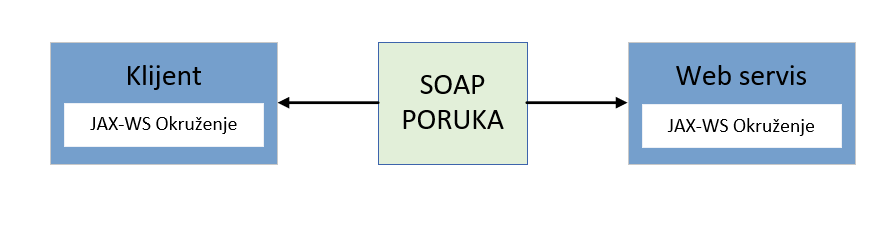


Slika : SOAP poruka s dodatkom [[2]](#footnote-2)

Kao što je već ranije opisano, SOAP je samo protokol koji definira razmjenu poruka. Kako bi se odvijala komunikacija pomoću SOAP poruka potreban je razvojni okvir koji to podržava.

JAX-WS je specifikacija za izradu web servisa koji komuniciraju XML baziranim protokolom kao što je SOAP. Pošto su SOAP poruke kompleksne JAX-WS API sakriva složenu logiku i pruža jednostavno sučelje programerima.

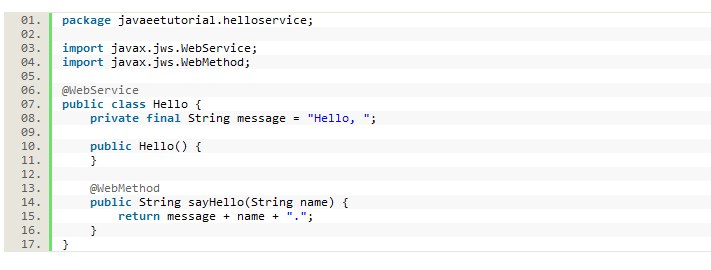
Na strani poslužitelja programer definira operacije web servisa pomoću metoda napisanih u programskom jeziku Java. Klijentske je programe također lako pisati. Klijent kreira surogat (engl. *Proxy*), zamjensku klasu, a zatim jednostavno poziva metode na proxy-ju. JAX-WS omogućuje automatsko generiranje SOAP poruka iz programskog koda. Najveća prednost JAX WS klijenata je da su neovisni od platforme Java programskog jezika. Druga velika prednost je fleksibilnost koja je postignuta korištenjem W3C tehnologija: HTTP, SOAP i WSDL. [13] Slika 9 prikazuje ranije opisanu komunikaciju JAX-WS servisa i klijenta.



Slika : Komunikacija između JAX-WS servisa i klijenta [[3]](#footnote-3)

WSDL (engl. *Web Service Description Language*), također vrlo bitan pojam za SOAP poruke, on specificira XML format za opis usluge kao skup krajnjih točaka koje djeluju na poruke. Prilikom razvoja SOAP web servisa postoje dva pristupa: prvi se naziva *Contract-First* u kojemu se najprije piše opis web servisa, odnosno WSDL te pomoću nekog od javno dostupnih razvojnog okvira kao što je Spring generira odgovarajući programski kod. Drugi je pristup *Code-First* u kojem se, kao što mu i ime kaže, prvo piše programski kod web servisa iz kojeg se automatski generira WSDL. Postoje brojne rasprave i studije o tome koji je pristup bolji i efikasniji, ali ipak prevladava mišljenje da ako se radi o servisu koji ima široku primjenu bolje je primijeniti pristup Contract-First jer se svakom izmjenom koda neće automatski mijenjati WSDL nego će se mijenjati samo na zahtjev klijenata. [14]

U nastavku slijedi kratak programski isječak kojim se definira jednostavan web servis.



Slika : Jednostavan SOAP web servis [[4]](#footnote-4)

Kao što je vidljivo iz programskog koda, opisan je jedan vrlo jednostavan web servis naziva „Hello“ koji se sastoji od svega jedne metode. Jedina uloga metode je ispis pozdrava pozivatelju. JAX-WS omogućuje kreiranje web servisa pomoću anotacija koje su prisutne u programskom jeziku Java od verzije 7.

* + 1. RESTful

RESTful (engl*. Representational State Transfer*) ili skraćeno REST web servise definirao je 2000. godine Roy Fielding u svojoj doktorskoj disertaciji pod naslovom “*Architectural Styles and the Design of Network-based Software Architectures*”. Kao i SOAP web servisi, REST servisi su arhitekturni stil aplikacije tipa klijent-poslužitelj koja je usredotočena na prijenos podataka prema principu zahtjev-odgovor. [15]

REST servisi u većini slučajeva koriste HTTP protokol koji je uz ostale autore također specificirao Roy Fielding. HTTP je sinkroni aplikacijski protokol temeljen na zahtjevu-odgovoru, koji se koristi za distribuirane, kolaborativne sustave temeljene na dokumentima. Velika prednost HTTP-a je u tome što tijelo poruke može sadržavati bilo koji format poruke. Neki od najpopularnijih su HTML, XML, JSON, običan tekst i binaran tip podatka. [16]

Korištenje REST servisa je specifično jer se temeljni na HTTP metodama te mora biti u skladu s definicijom HTTP-a. Na taj se način ostvaruju osnovne CRUD (engl. *Create Read Update Delete*) operacije:

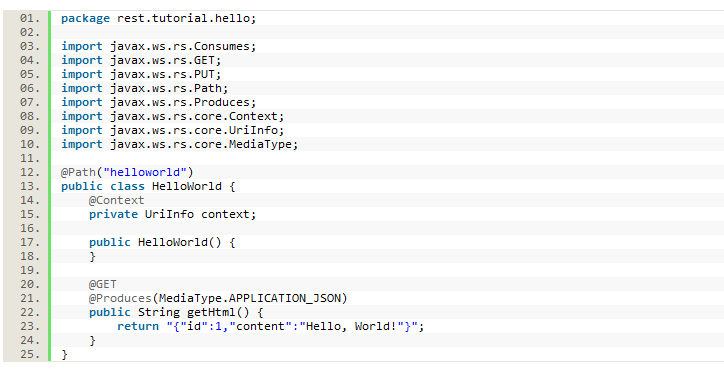
* POST – služi za kreiranje resursa
* GET– služi za dohvat resursa
* PUT – služi za promjenu stanja resursa
* DELETE – služi za uklanjanje resursa

Neke od glavnih značajki i razloga korištenja REST web servisa su: identifikacija resursa putem URI, jednostavno upravljanje resursima pomoću HTTP metoda, samo-opisne poruke u različitim formatima, jednostavna isporuka i integracija servisa kao i manipuliranje pomoću putanja servisa.

JAX-RS je specifikacija za programski jezik Java koja olakšava kreiranje web servisa koji koriste REST arhitekturu. Anotacije su uvedene kako bi se olakšalo i ubrzalo kreiranje servisom. Neke od najvažnije i najkorištenijih anotacija [16] su:

* @Path – klasa REST servisa koja daje relativnu URI putanja koja određuje mjesto na kojem će biti smještene Java klase
* @GET,@POST,@PUT,@DELETE – HTTP metode
* @PathParam – URI parametri putanje izdvajaju se iz URI zahtjeva i nazivi parametara korespondiraju nazivima varijabli predloška URI putanje iz @Path anotacije na razini klase
* @QueryParam – tip parametra koji se može izdvojiti za korištenje u klasi resursa. Parametri upita izdvajaju se iz parametara URI zahtjeva
* @Consumes – određuje MIME tip reprezentacije medija koji resurs može konzumirati kada se šalje klijentu
* @Produces – određuje MIME tip reprezentacije medija koji resursa stvara i vraća korisniku

U nastavku slijedi kratak isječak programskog koda kojim je definiran jednostavan REST web servis. Web servis je naziv Helloworld i ima jednu metodu koja je tipa GET i korisniku vraća „Hello, World!“ pozdrav u JSON obliku. Na prikazanom primjeru vidljive su neke od ranije spomenutih pojmova koji su karakteristični za REST web servise: HTTP metode, JAX-RS anotacije, različiti tipovi odgovora – JSON.



Slika : Jednostavan REST web servis

* + 1. Usporedba web servisa

Iako se na prvi pogled dvije ranije opisane vrste web servisa, SOAP i REST, čine sličnima, mnogi bi programeri i stručnjaci u ovom području pružili brojne protuargumente. Jedina stvar oko koje se svi mogu složiti da je namjena ista: komunikacija sustava i razmjena podataka.

U nastavku slijedi tablica u kojoj su uspoređene dvije vrste servisa:

Tablica 2: Usporedba karakteristika SOAP i REST servisa [17]

|  |  |
| --- | --- |
| SOAP | REST |
| Dobro poznata, tradicionalna tehnologija | Relativno nova tehnologija u usporedbi sa SOAP |
| Unutar poduzeća i u B2B scenarijima, SOAP je i dalje vrlo atraktivan. | Česta primjena REST servisa u bankarstvu |
| U SOAP-u je interakcija klijent-poslužitelj čvrsto povezana. | U REST-u interakcija klijent-poslužitelj je slabo povezana. |
| Postoje brojni razvojni okviri za SOAP | REST razvojni programeri tvrde da SOAP-u nedostaje fleksibilnosti |
| Promjene na strani poslužitelja često uzrokuje promjene na strani klijenta – ponovno kompiliranje programskog koda | Promjene na strani poslužitelja ne uzrokuju promjene na strani klijenta |
| SOAP ima veliko opterećenje u odnosu na REST | REST je lagan, jer je namijenjen laganom prijenosu podataka preko najčešće poznatog sučelja (URI) |
| Zahtjeva binaran zapis podataka | Podržava sve tipove podataka |
| SOAP servisi uvijek vraćaju XML tip podataka | REST servisi su fleksibilni u pogledu povratnog tipa podataka |
| Dizajnirani za korištenje kod udaljenih okruženja | Pretpostavlja komunikaciju od točke do točke  model – nije predviđen za udaljena okruženja |
| Težak za razvoj, zahtjeva dodatne alate | Puno jednostavniji za razvoj u odnosu na SOAP |
| Kriva predodžba da je SOAP sigurniji | REST pretpostavlja prijenos preko HTTP ili HTTPS i dodatno raspoloživi ugrađeni protokoli |
| Trenutno se više koristi i nudi bolju integraciju s drugim alatima i podršku dobavljača | Nedostatak podrške za sigurnosne standarde i poruke |

Unutar istraživanja „Web Services Based On SOAP and REST Principles“, napravljena je usporedba performansi REST i SOAP servisa na složenom projektu [19]. Projekt naziva „Multimedijska konferencija API“, pruža mogućnosti kreiranja i uklanjanja konferencije, te manipuliranje sudionicima. Svaki od dva implementirana web servisa pruža iste metode s jednakom implementacijom. Spomenute metode su: kreiranje konferencije, dohvat informacija konferencije, dodavanje sudionika, uklanjanje sudionika, dohvat svih sudionika, dohvat informacija o sudioniku i završavanje konferencije. Rezultati istraživanja prikazani su u tablici broj 3.

Tablica 3: Usporedba rada SOAP i REST servisa [18]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| MULTIMEDIJSKA KONFERENCIJA API | SOAP | | | REST | | |
| **Vrijeme u udaljenom okruženju (u ms)** | **Vrijeme na lokalnom okruženju (u ms)** | **Prijenos podataka (byte)** | **Vrijeme u udaljenom okruženju (u ms)** | **Vrijeme na lokalnom okruženju (u ms)** | **Prijenos podataka (byte)** |
| Kreiraj konferenciju | 848.4 | 381.7 | 767 | 171.4 | 102.7 | 273 |
| Dohvati informacije konferencije | 818.6 | 335.3 | 546 | 172.3 | 98.6 | 177 |
| Dodaj sudionika | 1325.3 | 334.2 | 578 | 368.8 | 103.3 | 200 |
| Ukloni sudionika | 1322.3 | 357 | 588 | 382.9 | 107.2 | 195 |
| Dohvati sudionike | 787.1 | 342.7 | 615 | 197.8 | 104.8 | 195 |
| Dohvati informacije o sudioniku | 766.2 | 346.7 | 619 | 169.8 | 105 | 204 |
| Završi konferenciju | 1508.4 | 341.4 | 500 | 556.6 | 105.3 | 204 |

Kao što je vidljivo iz tablice, REST servis je nedvojbeno ostvario bolje rezultate. Bez obzira je li mijerenje obavljeno na udaljenom poslužitelju ili lokalno REST servis je u nekim slučajevima ostvario i do četiri puta bolje vrijeme posluživanja od SOAP servisa. Prijenos podataka je također znatno manji od REST servisa.

1. Usporedna analiza metoda komunikacije

U ovom poglavlju napravljena je usporedna analiza ranije definiranih metoda komunikacije. Cilj poglavlja je sažeti ranije opisane metode komunikacije i opisati scenarije u kojim je najbolje koristiti određenu metodu komunikacije.

Tablica 4 prikazuje neke od najbitnijih karakteristika metoda te njihove prednosti i mane. Zadnji stupac predstavlja kvalitativnu vrijednost, broj poruka koju je određena metoda uspjela proizvesti unutar vremenske jedinice od jedne sekunde. Mjerenje je provedeno na način da je svaka metoda slala istu poruku u JSON formatu veličine 258 byte-ova. Mjerenje je provedeno kroz tri ponavljanja u razmaku od 5 minuta. Svaka iteracija mjerenja tijekom koje su slane poruke za svaku metodu zasebno trajala je 60 sekundi. Predočeni rezultati predstavljaju srednju vrijednost mjerenja na bazi vremena od 1 sekunde.

Tablica 4: Usporedna analiza metoda komunikacija

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Kada koristiti? | Prednosti | Mane | Broj poruka u sekundi |
| SOAP | * Ako aplikacija treba imati garantiranu razinu sigurnosti i pouzdanosti * Ako se želi pružiti sinkrona obrada i pozivi * Ako se obje strane (davatelj usluga i potrošač) moraju dogovoriti o formatu poruka koje se razmjenjuju | * Podržava opis servisa pomoću kojeg je lakše razumjeti operacije i namjenu servisa * Jasno definiran izgled SOAP poruke i omotnice * Podržava HTTPS | * Podržava samo XML format * Teško upravljanje greškama prilikom izvođenja operacija * Relativno stara vrsta servisa, postepeno se prelazi na REST servise * Poruke su velike što uzrokuje spori prijenosom | 16.3 |
| REST | * Ako sustav zahtjeva CRUD operacije na resursima * Ako se želi koristi *Cache* memorije za informacije koje se prenose * Ako je potrebna podrška za sve web preglednike | * Podržava JSON/XML/TEXT format * Jednostavno upravljanje greškama prilikom izvođenja operacija * Jednostavna implementacija servisa * Nova vrsta web servisa * Ne troši puno CPU resursa * Podržava HTTPS | * Usko povezan za HTTP protokol * Često pogrešno korištenje CRUD metode, osobito GET metode | 63.1 |
| MQTT | * Ako se želi povezati veći broj korisnik sa što manjom međusobnom ovisnošću * Ako se žele povezati IoT uređaji s aplikativnim sustavom * Ako se žele slati poruke male veličine, vrlo visokom brzinom | * Asinkroni protokol, primijenjen praćenju rada IoT senzora * Dobra podrška za lošu internet vezu * Jednostavan za instalaciju i korištenje * Veliki broj javno dostupnih poslužitelja | * Radi se o vrlo laganom protokolu koji ne podržava velike poruke * Ne podržava slanje slika i video zapisa | 87.6 |
| JMS | * Ako se želi poslati ista poruka većem broju zainteresiranih korisnika * Ako se žele slati poruke s vrlo visokim stupnjem pouzdanosti * Ako se žele slati obavijesti unutar istog ili udaljenog sustava | * Osigurava apstrakciju između klijenta i servera * Objavi/Pretplati princip rada * Visoka razina pouzdanosti | * Komplicirana implementacija * Dodavanjem JMS-a unutar sustava znatno se povećava složenost sustava * Nema podrške za obavještavanje i nadgledanje reda poruka * REST web servisi razmjenjuju JMS | 37.6 |
| AJAX | * Ako se želi osvježiti web stranica bez ponovnog učitavanja stranice * Ako se žele zatražiti podaci od poslužitelja nakon učitavanja stranice * Ako se pozadinski žele slati podaci do poslužitelja | * Povećano korisničko iskustvo * Smanjeno vrijeme čekanja, ukupno ubrzanje sustava * Povećanje razine povezanosti sustava – moguće integracije s vanjskim sustavima * Manja razina korištenja internet prometa * Jednostavnija navigacija kroz web aplikaciju | * Problemi podrške kod određenih web preglednika * Smanjenje sigurnosti aplikacije i povećanje kompleksnosti programskog koda * Povećanje opterećenja na poslužiteljskoj stani | 43.2 |
| Web Socket | * Ako se želi postići sustav s velikom brzinom reaktivnosti * Ako se želi uspostaviti komunikacija sustava ili klijenata u realnom vremenom * Ako se žele koristi poruke visoke frekvencije, a male veličine | * Klijent i poslužitelj mogu slati poruke jedan drugom bez ikakve ovisnosti * Visoka razina sigurnosti * Mogućnost povezivanja više odvojenih sustava | * Postoje web poslužitelji koji nemaju dobru podršku za Web socket-e * Nije moguće pratiti Web socket poruke | 58.9 |

1. Implementacija sustava

Nakon predstavljanja metoda komunikacije između programskih sustava, u ovom poglavlju bit će prikazan implementirani sustav koji objedinjuje opisane metode, a svrha mu je nadgledanje letova aviona. Sustav se sastoji od tri odvojene aplikacije koje međusobno komuniciraju i tvore jednu cjelinu. Prva aplikacija, orijentirana korisnicima i bazi podataka, prikazuje registrirane korisnike, njihove aktivnosti unutar sustava i pruža sučelje prema bazi podataka. Druga aplikacije zadužena je za prikaz aviona i aerodroma. Aplikacija kroz različite grafičke poglede prikazuje letove avione, aerodrome te udaljenosti između pojedinih aerodroma. Posljednja aplikacija je zamišljena kao administratorska aplikacija, unutar koje administrator može upravljati ostalim aplikacijama i nadzirati aktivnosti unutar sustava.

* 1. Korištena tehnologija i alati

Sustav za upravljanje aerodromima napravljen je unutar Netbeans razvojnog okruženja i koristi programski jezik Java. Kako se radi o kompleksnom projektu, integrirane su brojne tehnologije, programi te programski alati koji su opisan unutar ovog poglavlja.

* + 1. Razvojno okruženje

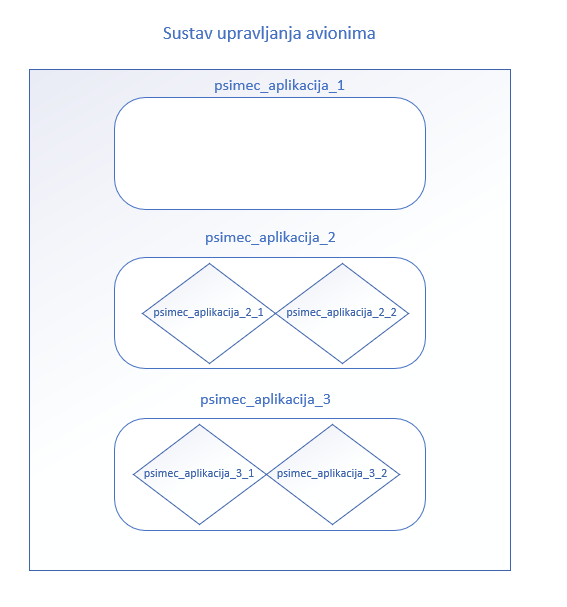
Kao što je već opisano cijeli sustav je izrađen pomoću programskog jezika Java, točnije Java EE. Kako se radi o web aplikacijama i web servisima, potreban je poslužitelj koji omogućuje isporuku (engl. *deploy*) aplikacije i servisa. Kao poslužitelj je odabran Glassfish 5.5. Radi se o poslužitelj otvorenoga koda (engl. *Open Source*), napisan u programskom jeziku Java, a namijenjen isporuci gotovih aplikacija [19]. Za pohranu podataka korištene su dvije baze podataka MySQL [21] i Apache Derby [22], unutar kojih su pohranjeni podaci vezani za korisnike, aerodrome i avione. Neke od poznatijih Java biblioteka i razvojnih okvira koji su također korišteni unutar projekta:

* Gson [23] – Java biblioteka namijenjena transformaciji Java objekata u JSON objekte i obrnuto
* Eclipse Jersey [24] – REST razvojni okvir koji pruža JAX-RS implementaciju
* JSF (engl. *JavaServer Faces*) [25] – Java razvojni okvir namijenjen za izradu korisničkog sučelja
* PrimeFaces [26] – besplatan razvojni okvir za JSF koji sadrži brojne gotove komponente prikladne za izradu korisničko sučelja
* JAX-WS [27] – razvojni okvir koji pruža alate i infrastrukturu za implementaciju web servisa

Osim navedenih razvojnih okvira i Java biblioteka projekt sadrži klijente vanjskih web servisa, i to:

* LocationIQ [28] – javno dostupan API brojnih mogućnosti primarno orijentiran na pružanje geografska lokacije. Unutar projekta korišten za određivanje točne lokacije aerodroma i udaljenosti između dvije geografske točke
* OpenWeather [29] - javno dostupan API koji pruža informacije vezane za vremensku prognozu. Unutar projekta korišten za prikaz prognoze za pojedini aerodrom
* OpenSkyNetwork [30] – javno dostupan API koji pruža informacije o letovima aviona. Unutar projekta je integrirana dretva (engl. *Thread*) koja u zadanim intervalima pomoću OpenSkyNetwork API-a prikuplja informacije o letovima te ih sprema u bazu podataka kako bi kasnije bili dostupni korisniku.
  1. Arhitektura sustava
     1. Struktura projekta

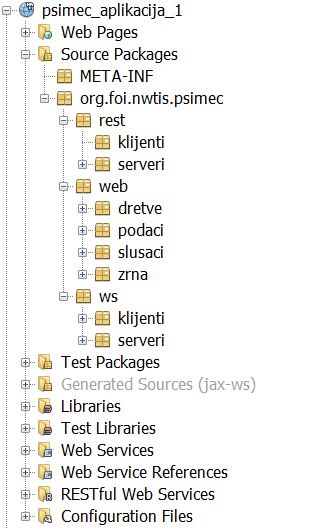
Slika 11 prikazuje projekt organiziran u tri odvojene aplikacije. Svaka aplikacija je zasebna komponenta koja komunicira s ostalim aplikacijama.



Slika : Prikaz aplikacija

Kao što je vidljivo iz slike 11, aplikacije 2 i 3 sastoje se od dodatnih modula. Aplikacije koje se sastoje od dodanih modula nazivaju se Java EE aplikacije. Prvi modul je tipa EJB (engl. *Enterprise JavaBean*) koji sadrži poslovnu logiku aplikacije, dok drugi modul naziva *Web projekt*, koristi poslovnu logiku EJB-a te generira ekrane za prikaz podataka korisniku.

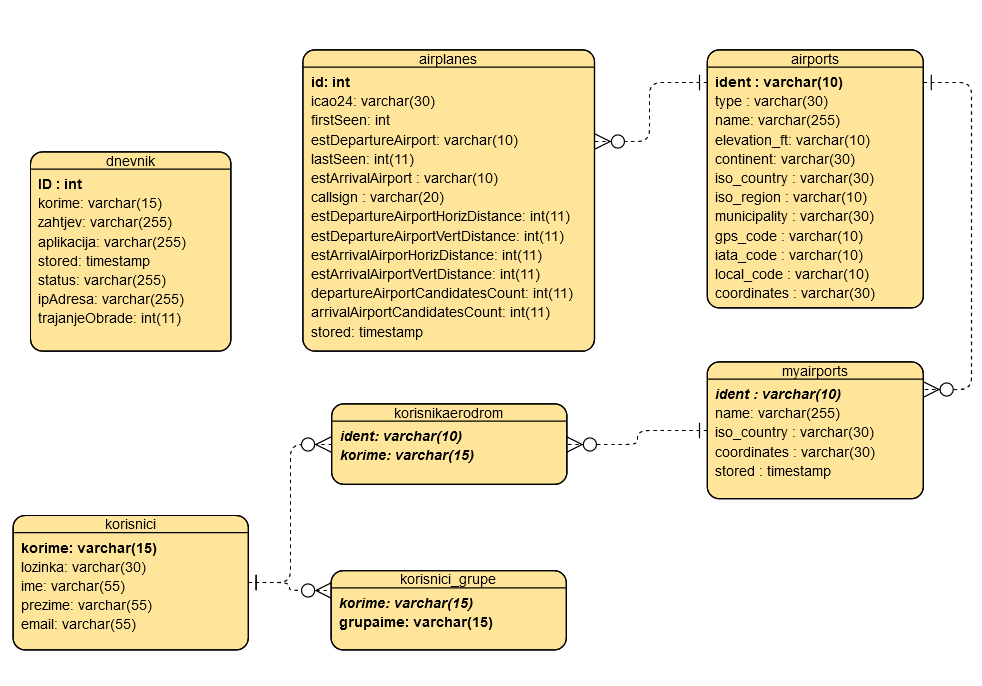
Organizacija podataka i klasa unutar web aplikacije prikaza je na slici 12. Kako najbolje prakse programskog jezika Jave nalažu, sve klase organizirane su u odvojene pakete. Tako postoje paketi koji sadrže REST i SOAP servise te paket koji sadrži sve klase povezane na grafički dio aplikacije. Iz prikazane slike vidljivo je kako paketi koji sadrže web servise imaju dodatnu podjelu na klijente i servere. Klijenti predstavljaju vanjske web servise koji su samo integrirani unutar aplikacije, dok se u paketu naziva serveri nalazi implementacija vlastitih web servisa. Također, web paket se sastoji od dodatnih paketa u svrhu bolje organizacije projekta.



Slika : Organizacija klasa web projekta

* + 1. Baza podataka

Sustav za upravljanje aerodromima sastoji se od dvije baze podatka. Prva baza, ujedno i glavna baza podataka, je *MySQL* baza podataka koja se sastoji od sedam tablica. Sve glavne funkcionalnosti podržane u sustavu odvijaju se preko baze podataka. Kako bi se osigurala što veća stopa neovisnosti, komunikaciju s prvom bazom podataka ostvaruje isključivo prva aplikacija, točnije web servisi aplikacije. Ostale aplikacije vezu prema bazi podataka ostvaruju pomoću dostupnih web servisa. Unutar baze nalaze se tablice koje sadrže podatke o korisnicima te akcijama koje određeni korisnici mogu napraviti. Tablica *airports* sadrži sve aerodrome koje su dostupni u sustavu, dok tablica *myairports* ostvaruje vezu između korisnika i aerodroma. Tablica *airplanes* sadrži sve podatke koje su vezani za jedan let aviona. Unutar tablice *dnevnik* nalaze se sve korisničke akcije koje su napravljene u sustavu, ta tablica ima nadzornu svrhu. Struktura baze podataka prikazana je na slici 13.



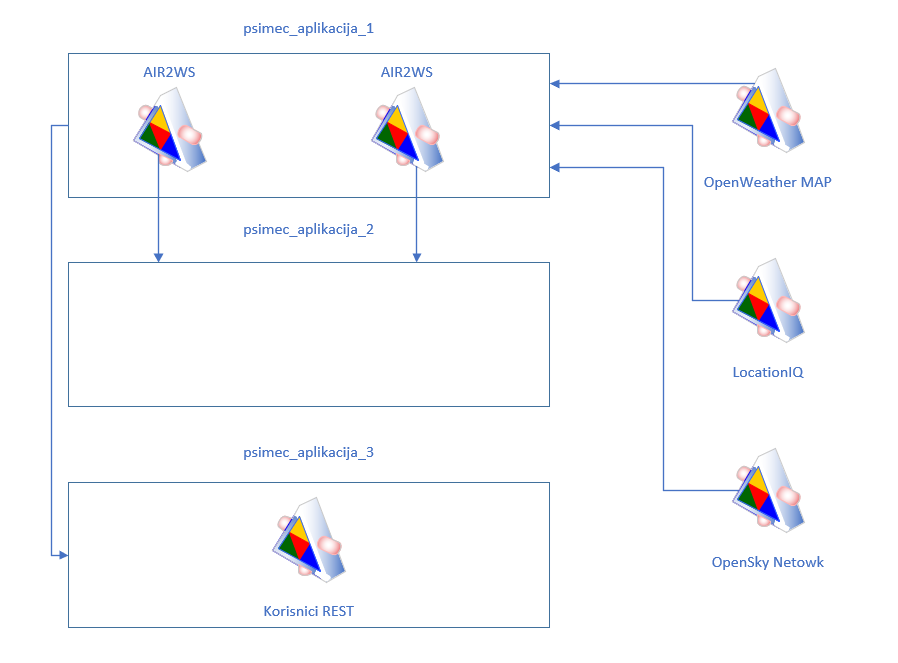
Slika : ERA model baze podatka

Druga baza podataka bitno je jednostavnija, sastoji se svega od dvije tablice: *dnevnik* i *mqtt\_poruke*. Tablica *dnevnik* razlikuje se od tablice istog imena u prvoj bazi podataka po pohranjenim podacima. Ova tablica sadrži adrese stranica koje je određeni korisnik posjetio, dok druga tablica *dnevnik* sadrži sve akcije kao što su pozivi web servisa, upravljanje MQTT porukama i slično. Tablica *mqtt\_poruke* kao što joj i ime govori, sadrži sve primljene MQTT poruke.

* + 1. Web servisi sustava

Cijeli programski sustav sastoji se od više povezanih komponenti. Komponenta koja se najviše koristi i ima najveću ulogu u cijelom sustavu jest web servis. Ukupno sustav koristi sedam web servisa, od kojih su tri vanjska. Vanjski servisi su *OpenWeather Map*, *LocationIQ* i *OpenSky* Network koji su ranije opisani. Navedene servise koristi isključivo prva aplikacija te dobivene informacije dalje prosljeđuje dijelovima sustava koje ih zahtjevaju.

Prva aplikacija implementira dva web servisa. Prvi servis, naziva *AIR2WS*, jedina je poveznica prema *MySQL* bazi podataka. Servis pruža CRUD operacije nad korisnicima sustava, omogućuje korištenje MQTT poruka ostvarivanjem veze prema MQTT poslužitelju i implementira metodu za izračun udaljenosti između dva aerodroma. Drugi web servis, naziva AIR2REST, orijentiran je na aerodrome. Servis implementira sve metode koje su potrebne za dohvat i upravljanje aerodromima. Druga aplikacija ima klijente prve aplikacije te prikazuje korisniku podatke koje dobiva preko servisa. Kako bi se ostvarila određena sigurnost, oba servisa obavljaju autentifikaciju prije izvršenja same funkcionalnosti metode. Zadnji web servis, naziva Korisnici REST, dio je treće aplikacije i isključivo se u njoj koristi. Njegova zadaća je autentifikacija, koja se ostvaruje pozivom mrežne utičnice (engl. *Server Socket*) smještene unutar prve aplikacije. Mrežna utičnica prve aplikacije definira skup operacija koje je moguće pozvati izvan aplikacije, a odnose se na autentifikaciju korisnika i upravljanje tokom MQTT poruka. Slika 14. prikazuje opisane servise i njihovo korištenje.



Slika : Web servisi sustava

* + 1. Konfigurabilnosti i lokalizacija sustava

Implementirani sustav podržava visoku stopu konfigurabilnosti. Svaka vrijednosti koja se koristi u aplikacije zapisana je unutar konfiguracije. Takvim principom rada omogućava se lagana izmjena načina rada aplikacija, pa čak i promjena cijele baze podataka bez izmjena vezanih na programski kod. Slika 15. predstavlja konfiguraciju prve aplikacije.

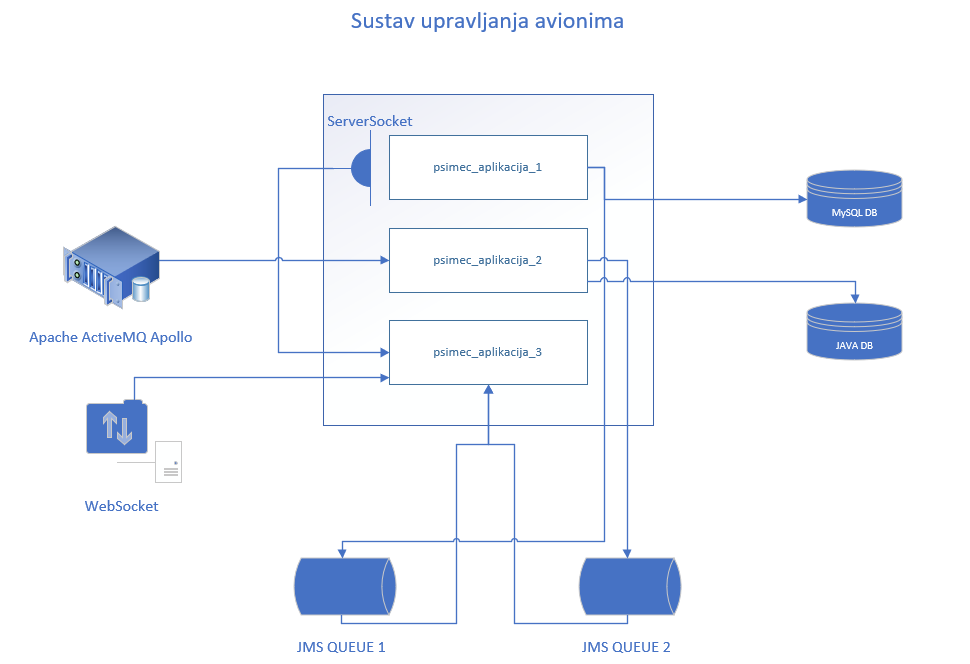


Slika : Konfiguracija aplikacije

Osim konfiguracije, sustav podržava i lokalizaciju, odnosno promjenu jezika. Trenutno je implementirana podrška za hrvatski i engleski jezik ali postoji mogućnost dodavanja više jezika prema potrebi. Lokalizacija se ostvaruje na način da postoje zasebne datoteke (engl. *file*) s ključnim riječima i opisima, koji se pozivaju s korisničke strane (engl. *front-end*) te se u konačnici prikazuju korisniku. Lokalizacija daje dodanu vrijednost sustavu i omogućuje korištenje na više govornih područja.

* 1. Metode komunikacije unutar sustava

U teorijskom dijelu ovog rada opisane su neke od metoda komunikacije između programskih sustava. Opisane metode implementirane su u sklopu sustava upravljanja avionima, kao što prikazuje slika 16. U nastavku ovog poglavlja prikazane su programske implementacije metoda komunikacije unutar sustava.



Slika : Metode komunikacije implementiranog sustava

Kao što je i vidljivo iz slike 16 sustav se sastoji od dva JMS reda poruka u koje prva i druga aplikacija šalju poruke, dok treća aplikacija samo koristi odnosno prikazuje poruke korisniku.

S programske strane potrebno je implementirati dvije klase za upravljanje porukama. Prva klasa, prikazana na slici 17, zadužena je za slanje poruka u JMS red. Kao što je vidljivo iz slike, definiran je red u koji se poruka šalje te jednostavnim pozivom metode send poruka je poslana u željeni red.



Slika 17: Implementacija - Slanje JMS poruka

Druga klasa vezana za JMS poruke prikazana je na slici 18, a sadrži funkcionalnosti primanja poruka. Klasa implementira sučelje MessageListener koje omogućuje da prilikom dolaska nove poruke u ranije definirani red, metoda onMessage bude automatski pozvana. Na opisan način implementirano je preuzimanje poruka iz oba JMS reda.

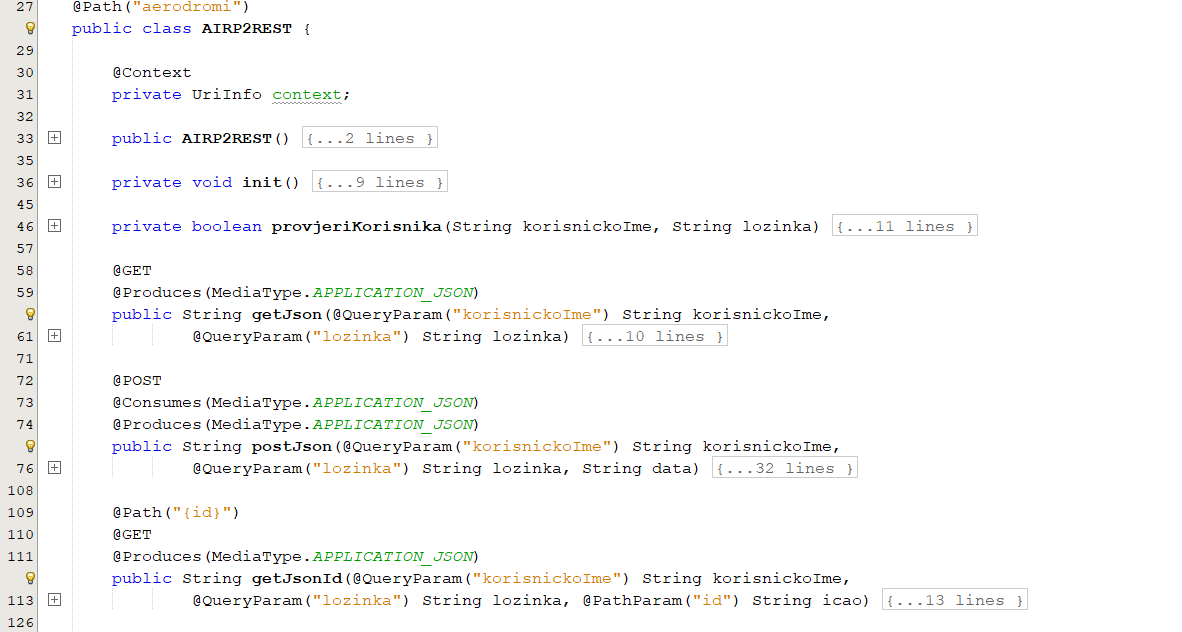


Slika 18: Implementacija - Primanje JMS poruka

Web servisi čine glavni način prijenosa informacija unutar implementiranog sustava. Njihova implementacija različita je ovisno o tipu servisa. Na slikama 19 i 20 prikazana je implementacija servisa s time da slika 19 predstavlja SOAP servis, dok se na slici 20 nalazi REST servis.



Slika 9: Implementacija - SOAP web servis



Slika 0: Implementacija - REST web servis

Web servisi su razvijeni korištenjem anotacija koje bitno olakšavaju implementaciju. Anotacije SOAP servisa odnose se na ime metoda servisa te ulazne parametre. Nešto složenije anotacije za REST servis, opisane su unutar teorijskog dijela ovog rada, a na slici 20 može se vidjeti njihova primjena.

MQTT poruke upravljane su pomoću vanjskog sustava, a njihov dohvat moguć je isključivo unutar implementiranog sustava. Slika 16 prikazuje Apache ActiveMQ Apollo poslužitelj koji je zadužen za slanje MQTT poruka. Unutar sustava primanje MQTT poruka implementirano je korištenjem dretve, prikazane na slici 21. Ona ostvaruje vezu prema poslužitelju i prima poruke. Nakon što poslužitelj pošalje poruku poziva se metoda onPublish koja sprema poruku u bazu podataka kako bi se kasnije mogla prikazati korisniku.



Slika 21: Implementacija - Primanje MQTT poruka

Slično kao JMS poruke, implementacija WebSocket-a sastoji se od dva odvojena dijela. Prva klasa, prikazana na slici 22, predstavlja točku pristupa (engl. *endpoint*), koja je zadužena za slanje poruka svim otvorenim sjednicama. Sjednica (engl. *session*) predstavlja aktivnu aplikaciju kojom se služi korisnik. Nakon što se dogodi određena akcija unutar sustava, klase obavještavaju točku pristupa da pošalje poruku svim otvorenim sjednicama.

Drugi dio WebSocket-a je sama utičnica (engl. *Socket*). Utičnica je implementirana na korisničkoj strani i omogućuje prikazivanje informacija pozivom određene metode. Konkretno, kada se korisnik prijavi u sustav, administratoru će putem WebSocket-a automatski doći obavijest o novoj radnji unutar sustava.



Slika 22: Implementacija - WebSocket točka pristupa

Sljedeća metoda komunikacije prema implementaciji razlikuje se od ranije objašnjenih. AJAX pozivi implementirani su na korisničkoj strani, dok su ostale metode implementirane na serverskoj strani. Slika 23 prikazuje implementaciju forme za filtrirani prikaz letova aviona unutar koje se nalazi AJAX poziv.



Slika 23: Implementacija - AJAX poziv

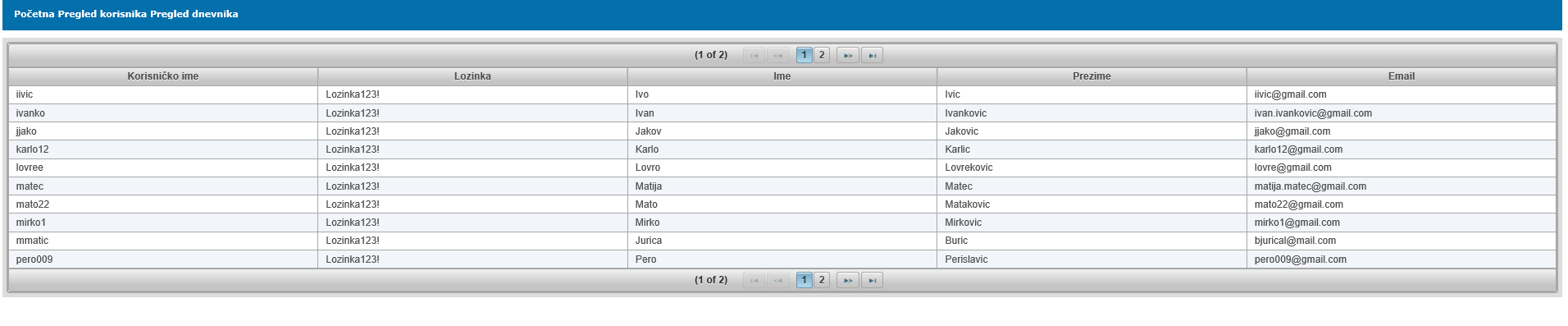
Posljednja metoda komunikacije je pomoću mrežne utičnice. Mrežna utičnica je implementirana korištenjem dretve (engl. *Thread*) koja konstantno radi te osluškuje definirani vrata (engl. *Port*). Kada korisnik pošalje određeni zahtjev na mrežnu utičnicu, zahtjev se prihvati, provjeri mu se sintaksa te u slučaju da zadovoljava sve uvijete koje zahtjeva aplikacija, zahtjev se obrađuje. Slika 24 prikazuje programsku implementaciju prihvaćanja poruke koja se prijenosi preko mrežne utičnice i obradu korisničkog zahtjeva.



Slika 24: Implementacija – Mrežna utičnica

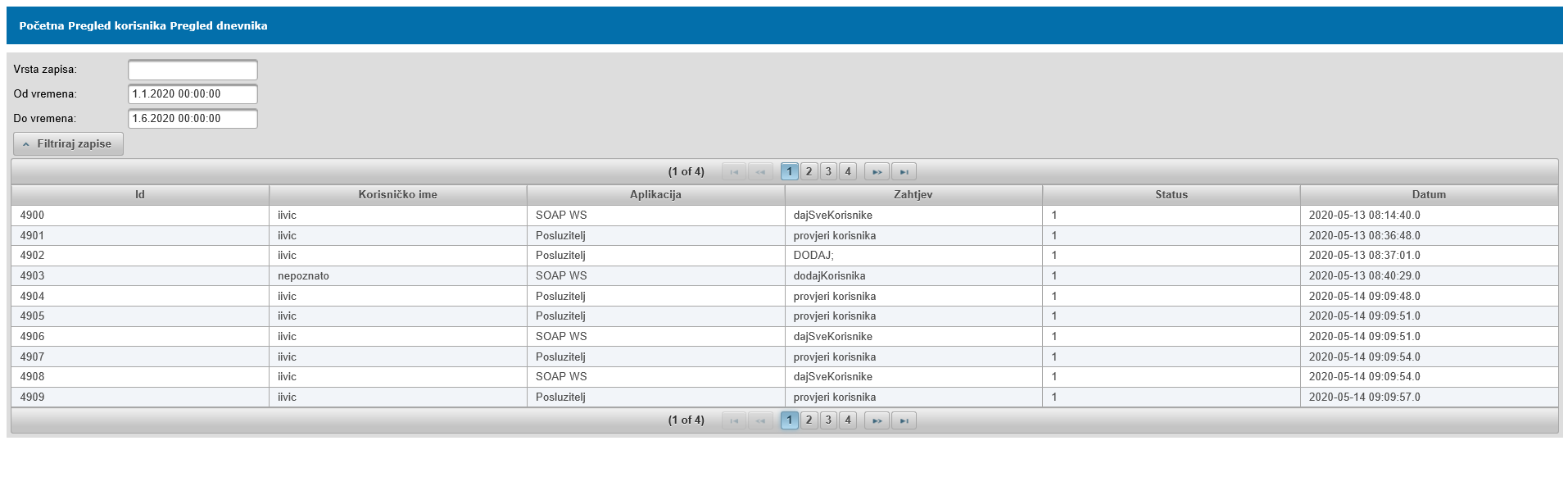
* 1. Primjer korištenja sustava

Nakon detaljnog opisa sustava s teorijske, arhitekturne i programske strane, unutar ovog poglavalja bit će prikazani grafički dio aplikacije. Svaka od tri aplikacije sastoji se od svoje zasebne web stranice koja korisniku pruža različite informacije. Tako se web stranica prve aplikacije sastoji od dva ekrana. Prvi ekran, naziva „Pregled korisnika“, prikazuje sve korisnike koji su se uspješno registrirali unutar sustava. Prikazani su svi dostupni korisnički podaci što je vidljivo i na slici 25.



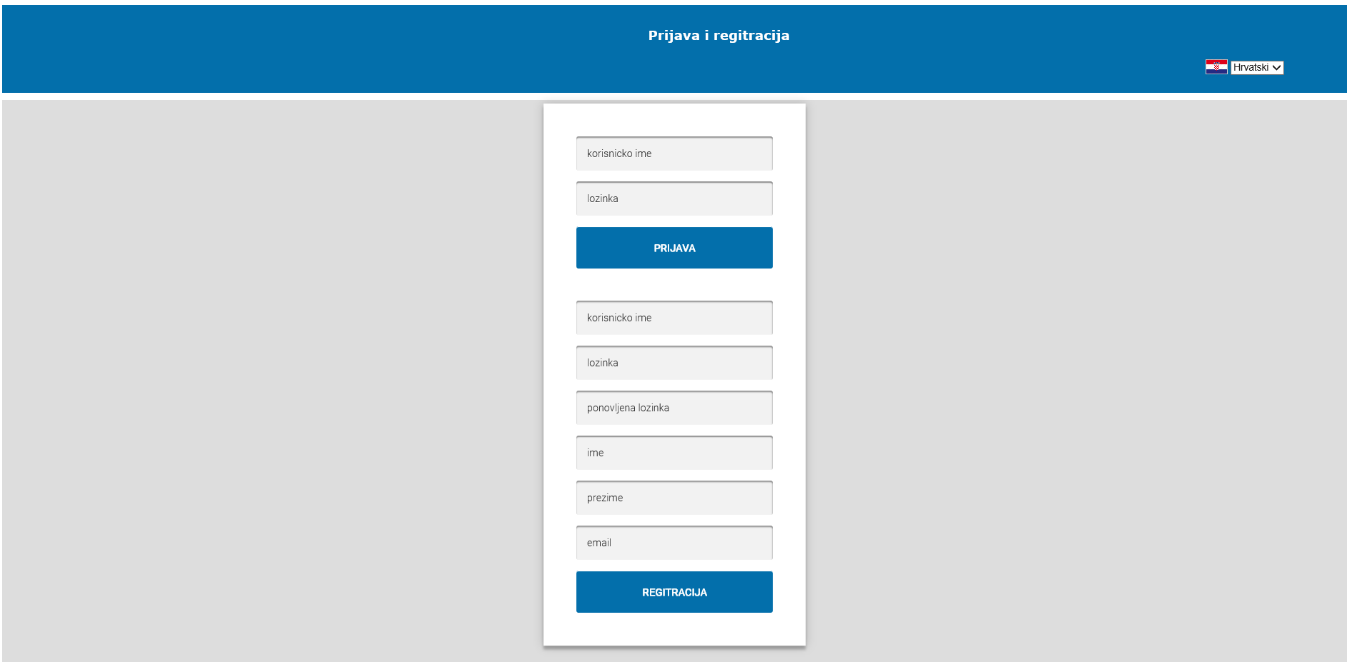
Slika 25: Stranica sustava - pregled korisnika

Drugi ekran, prikazan na slici 26, daje uvid u dnevničke zapise. Svaki zahtjev koji korisnik napravi unutar prve aplikacije zapisuje se u bazu podataka te se prikazuje korisniku. Ovakav pristup korištenja dnevnika omogućuje određeni nadzor nad sustavom.



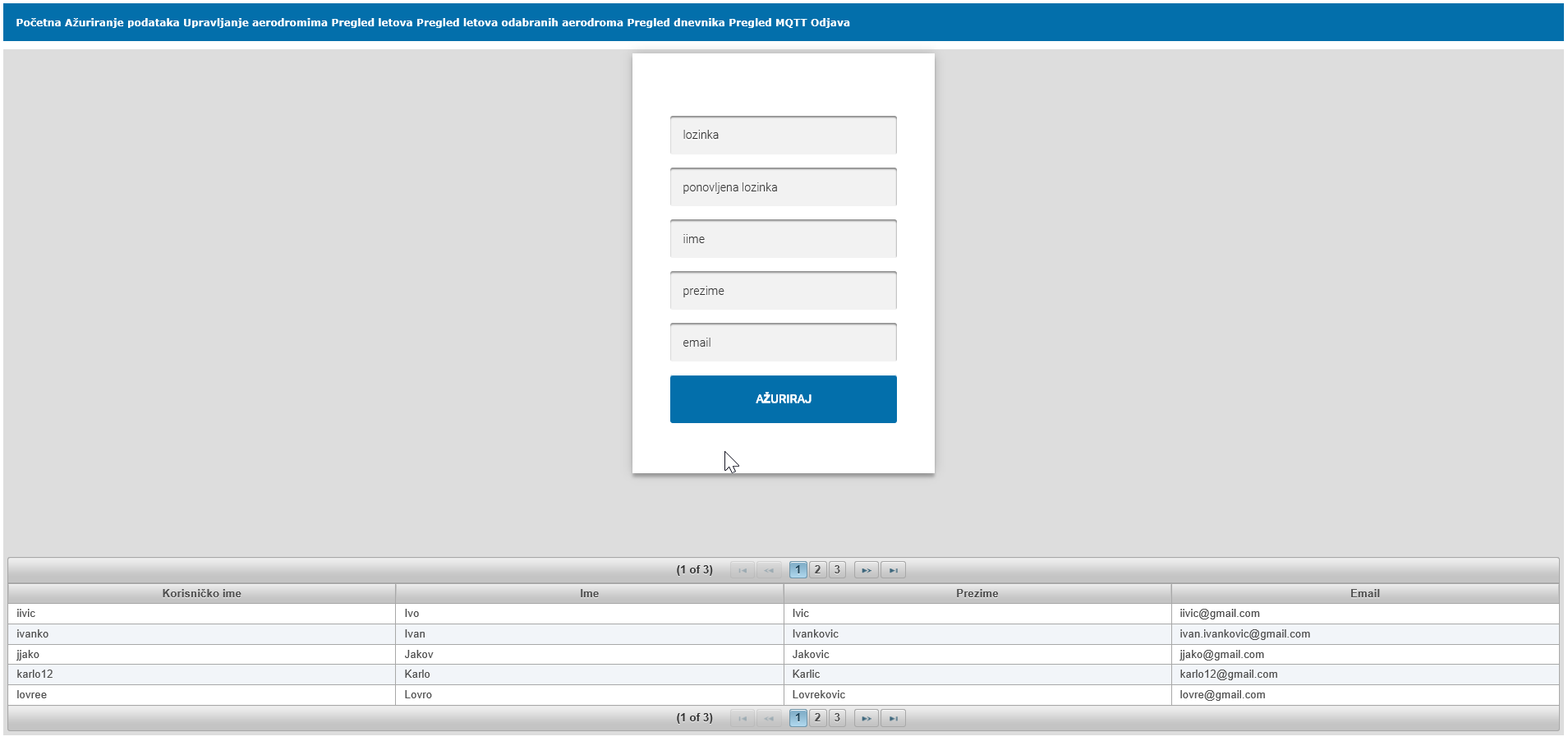
Slika 26: Stranica sustava - pregled dnevnika

Za razliku od prve, druga aplikacija sastoji se od zaslona za prijavu i registraciju koji su prikazani na slici 27. Aplikacija u pozadini poziva servise prve aplikacije pomoću kojih se izvršava većina funkcionalnosti ove aplikacije.



Slika 27 Stranica aplikacije - prijava i registracija

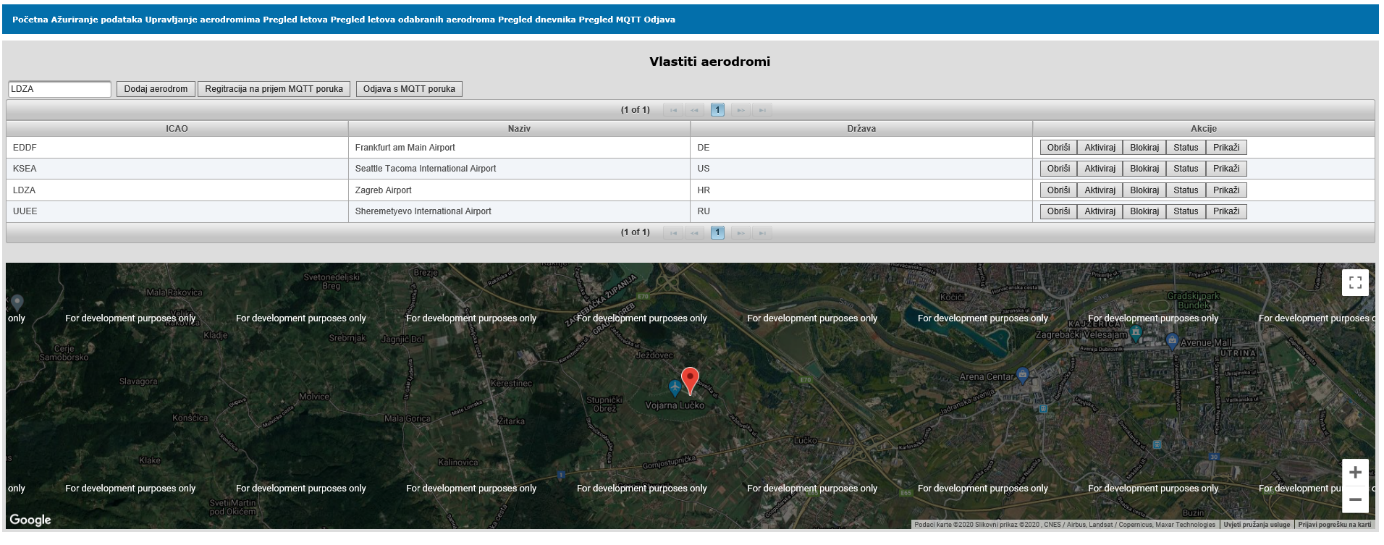
Druga mogućnost koju nudi je ažuriranje korisničkog profila. Korisnik prema potrebi može promijeniti svoje podatke i nastaviti koristiti sustav s novim podacima. Ažuriranje korisničkog profila prikazano je na slici 28.



Slika 28: Stranica aplikacije - ažuriranje korisnika

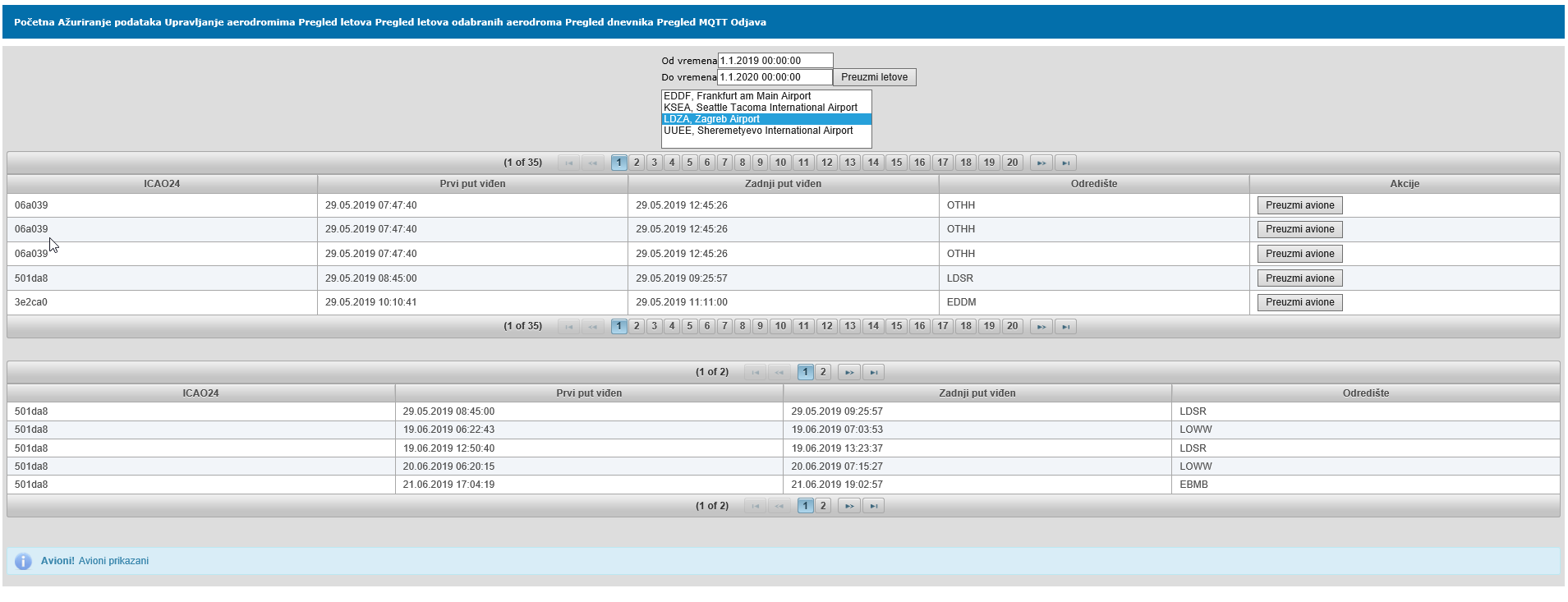
Kao što je i vidljivo iz slike, korisnik ne može promijeniti svoje korisničko ime jer je to njegova jedinstvena oznaka. Ispod forme za ažuriranje korisnik ima uvid u ostale korisničke profile s podacima, ali ovaj puta bez prikazanih lozinki.

Druga dostupna stranica treće aplikacije je „Upravljanje aerodromima“. „Upravljanje aerodromima“ je prva stranica koja prikazuje samo tematiku cijelog sustava. Na njoj je moguće pridružiti korisniku aerodrom za koji se interesira te prijaviti i odjaviti se s prijama MQTT poruka. Također, stranica korisniku nudi uvid u njegove aerodrome te određene akcije i prikaz aerodroma na *Google maps* karti. Opisana stranica nalazi se na slici 29.



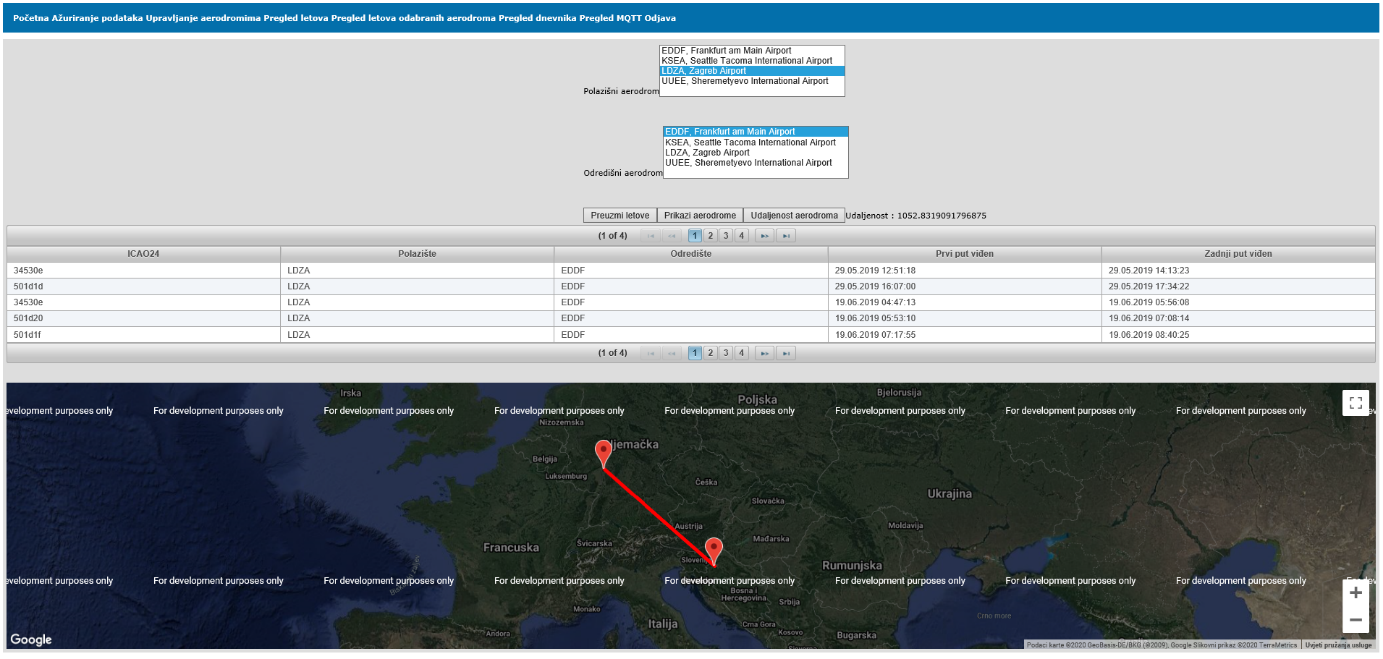
Slika 29: Stranica aplikacije - upravljanje aerodromima

Sljedeća stranica, naziva „Pregled letova“, omogućuje korisniku da za aerodrome koje je na prethodnoj stranici označio da ga zanimaju, pregleda letove koji su poletjeli s određenog aerodroma u definiranom vremenskom intervalu. Nakon što mu se prikažu letovi, korisnik može pogledati koji avioni su letjeli tim specifičnim putem. Opisana stranica prikazana je na slici 30.



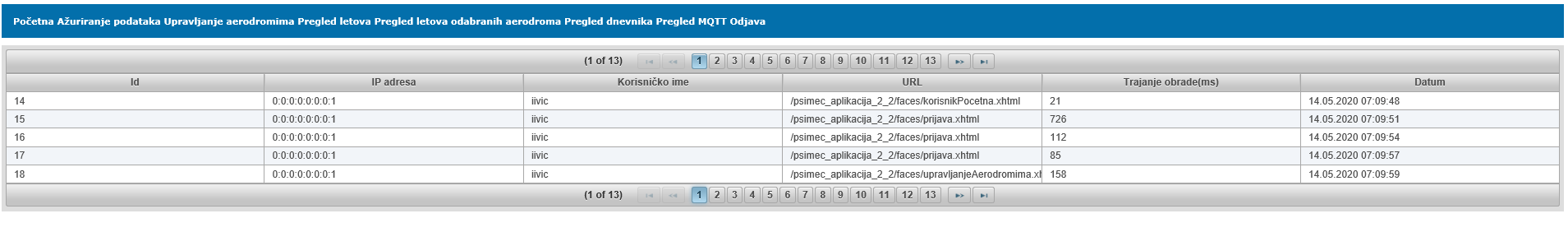
Slika 30: Stranica aplikacije - pregled letova

Druga stranice je „Pregled letova odabranih aerodroma“. Korisnik na stranici odabire polazišni i odredišni aerodrom te mu aplikacije prikazuje: sve dostupne letove između dva odabrana aerodroma, udaljenost aerodroma u kilometrima i prikaz aerodroma na karti.



Slika 31 Stranica aplikacije - pregled letova odabranih aerodroma

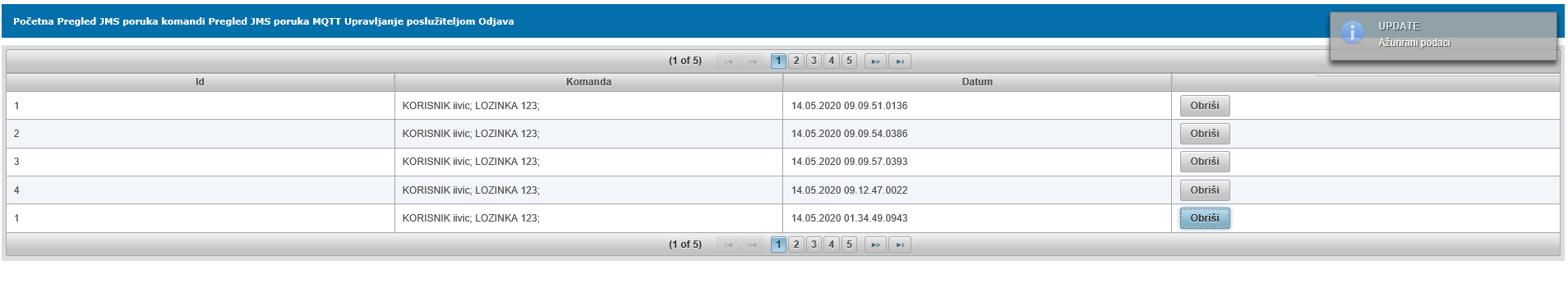
Zadnje dvije stranice koje su dostupne u sklopu druge aplikacije nisu vezane za aerodrome već za korisničke aktivnosti. Stranica „Pregled dnevnika“ daje uvid u sve stranice kojima je korisnik pristupio, IP adresu s koje je pristupio, vrijeme pristupa i koliko dugo je trajalo učitavanje stranice. Druga stranica, naziva „Pregled MQTT“, omogućuje korisniku pregled svih pristiglih MQTT poruka. Na slici 32 nalaze se dnevnički zapisi prijavljenog korisnika.



Slika 32: Stranica aplikacije - pregled dnevnika

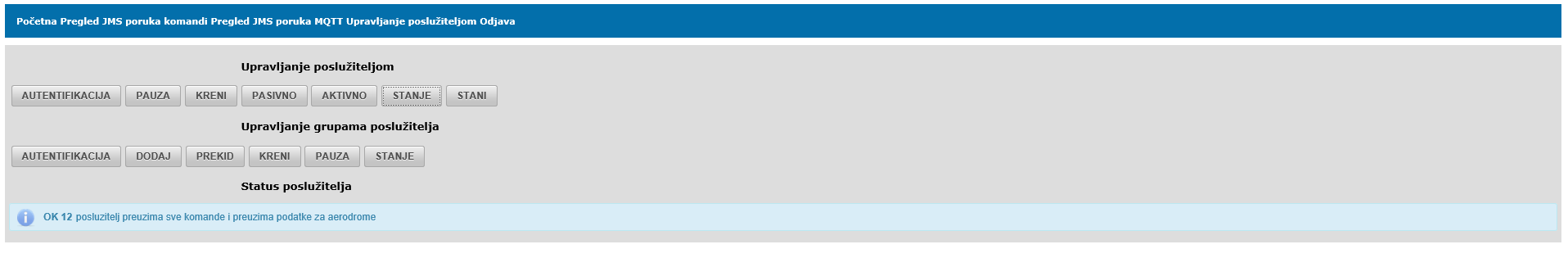
Treća aplikacije sastoji se od stranice za prijavu koja je vrlo slična prijavi na drugu aplikaciju. Samu autentifikaciju vrši REST servis treće aplikacije koji je već ranije opisan.

Svaka akcija na mrežnoj utičnici prve aplikacije šalje se kao JMS poruka u red poruka. Treća aplikacija dohvaća poruke i prikazuje ih korisniku. Dodatna funkcionalnost koja je implementirana kod prikaza poruka je WebSocket ažuriranje. Tako svaka poruka koja se dohvati iz reda, automatski bude prikazana korisniku bez ikakve akcije, kao što je i vidljivo na slici 33.



Slika 33: Stranica aplikacije - JMS poruke

Stranica „Pregled JMS poruka MQTT“, kao što joj ime kaže, prikazuje MQTT poruke proslijeđene iz druge aplikacije kao JMS poruke. Zadnja korisnička stranica koja je dostupna u trećoj aplikaciji je „Upravljanje poslužiteljem“. Na navedenoj stranici moguće je upravljati mrežnom utičnicom prve aplikacije, a podržane akcije su: pauza, kreni, stani, pasivno i aktivno. Također, moguće je upravljati tokom MQTT poruka pomoću akcija: dodaj, prekid, kreni, pauza i stanje. Opisana stranica nalazi se na slici 34.



Slika 34: Stranica aplikacije - Upravljanje poslužiteljem

1. Rasprava

Razvijeni sustav je odgovarajući za ovo istraživanje jer pokriva sve teorijski obrađene metode komunikacije i prikazuje kako se u praksi mogu implementirati opisane metode. U nastavku prikaz implementiranih metoda unutar razvijenog sustava:

* JMS poruke korištene su kao sredstvo komunikacije između tri odvojene aplikacije. Prva i druga aplikacija su prosljeđivale poruke u red, dok je treća koristila poruke, obrađivala i u konačnici ih prikazivala korisniku
* MQTT poruke dobivene od vanjskog poslužitelja pohranjivale su se u bazu podataka i slale iz aplikacije u aplikaciju pomoću JMS poruka
* AJAX je korišten kod korisničkih ekrana kako bi se dinamički prikazivale nove informacije bez korisničkih akcija
* WebSocket je korišten u kombinaciji s JMS porukama. Svaka dobivena poruka bi se prikazala svim prijavljenim korisnicima u isti čas
* Web servisi predstavljaju glavno sredstvo prijenosa podataka. Korišteni su vlastiti, ali i vanjski servisi kako bi se prikupile i distribuirale informacije vezane za aerodrome i avione te se u konačnici prikazale korisniku

Sustav za upravljanje aerodromima sastoji se od više odvojenih komponenti koje međusobno komuniciraju različitim metodama. Korištene metode su veoma popularne i često korištene kod komercijalnih aplikacija i sustava. Način implementacije pojedinih metoda je također primjeren i izrađen prema najboljoj praksi opisanoj u pojedinoj dokumentaciji metode.

Iako na prvi pogled sustav za upravljanje aerodromima izgleda kao koristan i prihvatljiv sustav, on to u stvarnosti nije. Moglo bi se reći da je razvijeni sustav „demo sustav“ koji implementira različite metode komunikacija u svrhu demonstracije njihovih funkcionalnosti i međusobne interakcije. Unutar sustava su na različit način povezane aplikacije koje međusobno razmjenjuju podatke te je sama implementacija odgovarajuća, ali podaci koji se šalju u nekim slučajevima nemaju previše smisla ili su implementirani isključivo kako bi se pokazao način rada određene metode komuniciranja. Kako bi implementirani sustav bolje konkurirao trenutnim sustavima na tržištu trebao bi se smanjiti stupanj povezanosti sustava u svrhu povećanja brzine sustava te organizirati podatke na način da korisniku budu prihvatljiviji.

Grafički dio aplikacije zasigurno zahtjeva doradu, ukoliko bi se koristio u komercijalne svrhe. Današnje web aplikacije su postavile vrlo visoku ljestvicu odgovarajućeg web dizajna i ako se želi biti u koraku s njima potrebno je izraditi inovativnu, jednostavnu aplikaciju koja prati današnje trendove, što sustav za upravljanje avionima nije. Druga stvar koju treba uzeti u obzir je sigurnost. Unutar aplikacije koristi se autentifikacija kod poziva metoda web servisa i kod prijave u aplikacije, što također u današnje vrijeme ne predstavlja dovoljno visoku razinu sigurnosti.

1. Zaključak

Temeljem proučavanja metoda komunikacije programskih sustava dolazimo do sljedećeg zaključka. Jedna od temeljnih vrijednosti sustava je njegova podložnost promjenama, koja se postiže odvajanjem sustava u manje programske cjeline. Razdvajanje programskog sustava na manje komponente je prema brojnima izvorima jedan od najboljih načina povećanje kvalitete cijelog sustava. Nadalje, postoje i brojni uzorci dizajna koji prikazuju dobre prakse odvajanja sustava u manje komponente s ciljem smanjenja povezanosti sustava. Takve odvojene komponente međusobno razmjenjuju informacije različitim metodama komunikacije. Cilj ovog rada bio je upoznati se s takvim metodama, pružiti najbitnije informacije i u konačnici prikazati primjer u kojem su obrađene metode komunikacije implementirane.

Razvijeni Sustav za upravljanjem aerodromima sadrži brojne metode komuniciranja te prikazuje njihovu namjenu i široku upotrebu. Opisani sustav nije prikladan za produkcijsku namjenu, ali može služiti kao pokazni projekt koji ističe pojedine metode komunikacije i služi ostalim razvojnim programerima kao primjer dobre prakse.

Popis literature

1. “MQTT Version 5.0.” [Online]. Dostupno na: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html. [Accessed: 10-May-2020].
2. Gaston C. Hillar. MQTT Essentials - A Lightweight IoT Protocol. January 2017.
3. “Introduction to WebSocket.” [Online]. Available: https://javaee.github.io/tutorial/websocket001.html#BABDABHF. [Accessed: 09-May-2020].
4. “RFC 6455 - The WebSocket Protocol.” [Online]. Dostupno na: https://tools.ietf.org/html/rfc6455. [Accessed: 09-May-2020].
5. Paulson, L. D. (2005). Building rich Web applications with Ajax. Computer, 38(10), 14–17. <https://doi.org/10.1109/MC.2005.330>
6. B. Lin, Y. Chen, X. Chen and Y. Yu, "Comparison between JSON and XML in Applications Based on AJAX," 2012 International Conference on Computer Science and Service System, Nanjing, 2012, pp. 1174-1177, doi: 10.1109/CSSS.2012.297.
7. Puranik, D. G., Feiock, D. C., & Hill, J. H. (2013). Real-time monitoring using AJAX and WebSockets. Proceedings of the International Symposium and Workshop on Engineering of Computer Based Systems, 110–118. <https://doi.org/10.1109/ECBS.2013.10>
8. “Basic JMS API Concepts.” [Online]. Dostupno na: https://javaee.github.io/tutorial/jms-concepts002.html. [Accessed: 09-May-2020].
9. Tarak Modi (2002). Practical Java Message Service An Introduction to concepts and a guide to developing applications
10. James Snell, Doug Tidwell, and Pavel Kulchenko. 2002. Programming Web services with SOAP. O’Reilly & Associates, Inc., USA.
11. “SOAP Specifications.” [Online]. Dostupno na: https://www.w3.org/TR/soap/. [Accessed: 03-May-2020].
12. Graham, Steve & Davis, Doug & Simeonov, Simeon & Daniels, Glen & Brittenham, Peter & Nakamura, Yuichi & Fremantle, Paul & König, Dieter & Zentner, Claudia. (2002). Building Web services with Java: Making sense of XML, SOAP, WSDL, and UDDI.
13. “Overview of Java API for XML Web Services.” [Online]. Dostupno na: https://javaee.github.io/tutorial/jaxws001.html#A1250966. [Accessed: 03-May-2020].
14. T. Karthikeyan and J. Geetha, “Contract First Design: The Best Approach to Design of Web Services,” vol. 5, no. 1, pp. 338–339, 2014.
15. “What Are RESTful Web Services?” [Online]. Dostupno na: https://javaee.github.io/tutorial/jaxrs001.html#GIJQY. [Accessed: 03-May-2020].
16. B. Burke (2014). RESTful Java with JAX-RS 2.0, 2nd Edition Designing and Developing Distributed Web Services
17. “Creating a RESTful Root Resource Class.” [Online]. Dostupno na: https://javaee.github.io/tutorial/jaxrs002.html. [Accessed: 03-May-2020].
18. Wagh, K., & Thool, R. (2012). A Comparative Study of SOAP Vs REST Web Services Provisioning Techniques for Mobile Host. Journal of Information Engineering and Applications, 2(5), 12–16. <http://www.iiste.org/Journals/index.php/JIEA/article/view/2063>
19. Mumbaikar, S., & Padiya, P. (2013). Web Services Based On SOAP and REST Principles. International Journal of Scientific and Research Publications, 3(5), 3–6.
20. GlassFish dokumentacija. Dostupno na <https://javaee.github.io/glassfish/documentation>
21. MySQL dokumentacija. Dostupno na <https://dev.mysql.com/doc/>
22. Apache Derby dokumentacija. Dostupno na <https://db.apache.org/derby/manuals/index.html>
23. GSON dokumentacija. Dostupno na <https://github.com/google/gson>
24. Eclipse Jersey dokumentacija. Dostupno na <https://eclipse-ee4j.github.io/jersey/>
25. JSP dokumentacija. Dostupno na <http://www.javaserverfaces.org/>
26. PrimeFaces dokumentacija. Dostupno na <https://www.primefaces.org/documentation/>
27. JAX-WS dokumentacija. Dostupno na https://javaee.github.io/metro-jax-ws/
28. LocationIQ dokumentacija. Dostupno na <https://locationiq.com/docs>
29. Weather API dokumentacija. Dostupno na <https://openweathermap.org/current>
30. OpenSky Network dokumentacija. Dostupno na https://opensky-network.org/apidoc/

Popis slika

[Slika 1: MQTT Objavi/Pretplati model (autorski rad) 4](#_Toc48849828)

[Slika 2: Usporedba tradicionalnog i AJAX poziva [5] 6](#_Toc48849829)

[Slika 4: P2P pristup [8] 9](#_Toc48849830)

[Slika 5: Objavi/Pretplati pristup [8] 10](#_Toc48849831)

[Slika 6: Tok komunikacije s web servisom [10] 10](#_Toc48849832)

[Slika 7: SOAP poruka 12](#_Toc48849833)

[Slika 8: SOAP poruka s dodatkom 13](#_Toc48849834)

[Slika 9: Komunikacija između JAX-WS servisa i klijenta 14](#_Toc48849835)

[Slika 10: Jednostavan SOAP web servis 15](#_Toc48849836)

[Slika 11: Jednostavan REST web servis 17](#_Toc48849837)

[Slika 12: Prikaz aplikacija 25](#_Toc48849838)

[Slika 13: Organizacija klasa web projekta 26](#_Toc48849839)

[Slika 14: ERA model baze podatka 27](#_Toc48849840)

[Slika 15: Web servisi sustava 28](#_Toc48849841)

[Slika 16: Konfiguracija aplikacije 29](#_Toc48849842)

[Slika 17: Metode komunikacije implementiranog sustava 30](#_Toc48849843)

[Slika 17: Implementacija - Slanje JMS poruka 31](#_Toc48849844)

[Slika 18: Implementacija - Primanje JMS poruka 31](#_Toc48849845)

[Slika 19: Implementacija - SOAP web servis 32](#_Toc48849846)

[Slika 20: Implementacija - REST web servis 32](#_Toc48849847)

[Slika 21: Implementacija - Primanje MQTT poruka 33](#_Toc48849848)

[Slika 22: Implementacija - WebSocket točka pristupa 34](#_Toc48849849)

[Slika 23: Implementacija - AJAX poziv 34](#_Toc48849850)

[Slika 24: Implementacija – Mrežna utičnica 35](#_Toc48849851)

[Slika 25: Stranica sustava - pregled korisnika 36](#_Toc48849852)

[Slika 26: Stranica sustava - pregled dnevnika 36](#_Toc48849853)

[Slika 27 Stranica aplikacije - prijava i registracija 37](#_Toc48849854)

[Slika 28: Stranica aplikacije - ažuriranje korisnika 37](#_Toc48849855)

[Slika 29: Stranica aplikacije - upravljanje aerodromima 38](#_Toc48849856)

[Slika 30: Stranica aplikacije - pregled letova 39](#_Toc48849857)

[Slika 31 Stranica aplikacije - pregled letova odabranih aerodroma 39](#_Toc48849858)

[Slika 32: Stranica aplikacije - pregled dnevnika 39](#_Toc48849859)

[Slika 33: Stranica aplikacije - JMS poruke 40](#_Toc48849860)

[Slika 34: Stranica aplikacije - Upravljanje poslužiteljem 40](#_Toc48849861)

Popis tablica

[Tablica 1: Usporedba vremena prijenos podataka JSON i XML [6] 7](#_Toc48849862)

[Tablica 2: Usporedba karakteristika SOAP i REST servisa [17] 17](#_Toc48849863)

[Tablica 3: Usporedba rada SOAP i REST servisa [18] 19](#_Toc48849864)

[Tablica 4: Usporedna analiza metoda komunikacija 20](#_Toc48849865)

Sažetak

**Ime autora**: Paskal Šimec

**Naslov rada**: Analiza metoda komunikacija između programskih sustava

U ovom radu predstavljene su neke od poznatijih metoda komunikacije između programskih sustava, i to su: JMS, MQTT, WebSocket, AJAX te SOAP i REST web servisi. Svaka od navedenih metoda detaljno je opisana te su prikazane najbolje prakse korištenja pojedine metode. Rad sadrži i usporedbu metoda iz koje čitatelj može zaključiti u kojem scenariju je koja metoda bolje primjenjiva. Kako bi se pokazao način implementacije pojedine metode, izrađen je kompleksan programski sustav. Izrađeni programski sustav, naziva „Sustav za upravljanjem aerodromima“, koji se sastoji od više manjih komponenti koje međusobno komuniciraju te objedinjuje ranije opisane metode komunikacije i pokazuje način njihove primjene. Izrađeni sustav nije primijenjen za produkcijsku namjenu već isključivo kao pokazni projekt s objedinjenim dobrim praksama implementacije pojedinih metoda komunikacije između programskih sustava.

Ključne riječi: metode komunikacije, programski sustavi, povezanost sustava, komunikacija udaljenih sustava

Summary

**Author**: Paskal Šimec

**Title**: Analysis of communication methods between software systems

This paper presents some of the better-known methods of communication between software systems. The communication methods that this paper presents are JMS, MQTT, WebSocket, AJAX and SOAP and REST web services. Each of these methods is described in detail and the best practices of using each method are presented. The paper also contains a comparison of methods from which the reader can conclude in specific scenario which method is better applicable. In order to show how to implement a particular method, a complex software system was developed. The developed software system called "Airport Management System", consists of several smaller components that communicate with each other and combines the previously described methods of communication and shows how to apply them. The developed system is not used for production purposes, but exclusively as a demonstration project with integrated good implementation practices of individual methods of communication between software systems.

Key words: methods of communication, software systems, system connectivity, communication of remote systems

1. https://docs.oracle.com/cd/E19879-01/819-3669/bnbhj/index.html [↑](#footnote-ref-1)
2. https://docs.oracle.com/cd/E19879-01/819-3669/bnbhj/index.html [↑](#footnote-ref-2)
3. https://javaee.github.io/tutorial/jaxws002.html [↑](#footnote-ref-3)
4. https://javaee.github.io/tutorial/jaxws002.html [↑](#footnote-ref-4)