Udaljeno ispitivanje programske podrške za automobilsku industriju

Puntarić, Marko

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij računarstva

UDALJENO ISPITIVANJE PROGRAMSKE PODRŠKE ZA AUTOMOBILSKU INDUSTRIJU

Diplomski rad

Marko Puntarić

Osijek, 2022. godina.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 19.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Marko Puntarić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1154R, 22.10.2020.
OIB studenta:	69972252655
Mentor:	izv. prof. dr.sc. Josip Job
Sumentor:	,
Sumentor iz tvrtke:	Davor Kedačić
Predsjednik Povjerenstva:	Doc. dr. sc. Denis Vranješ
Član Povjerenstva 1:	izv. prof. dr.sc. Josip Job
Član Povjerenstva 2:	Prof. dr. sc. Marijan Herceg
Naslov diplomskog rada:	Udaljeno ispitivanje programske podrške za automobilsku industriju
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Hardver koji se koristi u automobilskoj industriji često je specifičan i ograničeno dostupan pa je samim time i otežano njegovo testiranje u smislu da je moguće provesti velik broj različitih testova u što kraćem vremenu na zadovoljavajući način. Ispitivanje programske podrške za takvu vrstu hardvera moguće je, uz određene preduvjete, provoditi s udaljene lokacije stoga je mjerenje vremena utrošenog na pojedine testove od velike važnosti za razvoj programske pordrške u takvim situacijama. Mjerenje vremena utrošenog na testove pruža uvid u zauzeće resursa, a što je preduvjet za prikupljanje znanja koje u konačnici omogućava lakše i preciznije planiranje i provedbu radnih procesa te optimiziranje korištenja resursa. Zadatak ovog rada je
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	19.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 28.09.2022.

Ime i prezime studenta:	Marko Puntarić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1154R, 22.10.2020.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Udaljeno ispitivanje programske podrške za automobilsku industriju**

izrađen pod vodstvom mentora izv. prof. dr.sc. Josip Job

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

U cilju razlikovanja od konkurenata, poboljšanja kvalitete proizvoda i smanjenja troškova, pojavljuje se sve veća potreba za izgradnjom pametnijih tvornica i radnih okruženja. Napori moraju biti usmjereni na kontrolu kako bi se produktivnost i profitabilnost maksimizirale. Ne samo veliki igrači iz automobilske industrije, već i mali dobavljači svakodnevno su pred izazovom da svojim kupcima pruže najbolji proizvod po najboljoj cijeni. U tom pogledu svaki pojedini detalj u cijelom lancu proizvodnje može igrati veliku ulogu. Iako su proizvodni procesi prva stvar koja padne na pamet, jednako kao i cijeli logistički lanac, uključujući sve od prijema sirovina do isporuke finalnog proizvoda, područja testiranja i ispitivanja te razvoja proizvoda također su vrlo važan dio ove slagalice. Tvornički informatički sustavi su ključni kako bi sve funkcioniralo kako treba. Danas u proizvodnim pogonima postoji puno elemenata koje bi se također trebali nadzirati, na primjer kao što su industrijska računala, programabilni logički kontroleri (engl. *programmable logic controller* - PLC), baze podataka, industrijske sabirnice podataka, besprekidna napajanja (engl. *uninterruptible power supply* - UPS) ili neki drugi mrežni elementi. Također, vrlo je važno razlikovati koje su najvažnije varijable koje daju dodanu vrijednost proizvodnom lancu i kupcu te ih staviti pod kontrolu. Povijesno gledano, tvrtke su se odlučile za implementaciju računalnih sustava za nadzor, mjerenje i upravljanje industrijskim sustavima u svojim proizvodnim procesima (engl. *Supervisory Control And Data Acquisition* - SCADA), u nastavku teksta SCADA. Iako je SCADA vrlo robustan sustav i dobro prihvaćen u industrijskoj zajednici, dolaskom novih rješenja poput industrijskog interneta objekata (engl. *Industrial internet of things* - IIoT), drugi distribuirani sustavi upravljanja i nadzora preuzeli su dio kolača, te su postojeći sustavi postali manje fleksibilni i znatno skuplji od danas postojećih rješenja. Pri implementaciji takvih alata za upravljanje i nadzor potrebna je određena količina opreza oko pitanja sigurnosti, protokola i spremanja podataka te cjelokupnog sustava. Ne radi se o količini, već o kvaliteti podataka, načinu na koji s njima postupamo i načinu na koji ih koristimo. U eri današnjice potrebno je razlikovati ono što je potrebno i ono što daje dodanu vrijednost od onoga što je dobro imati. Ako se pridržava tih načela, nadziranje i analiza podataka postaju vrlo moćan alat. U ovome radu opisana je ideja i izrada alata za udaljeni nadzor i praćenje parametara testnih stanica te princip rada cijeloga sustava koristeći neki od modernijih komunikacijskih protokola poput MQTT-a (engl. *Message Queuing Telemetry Transport*), u nastavku rada MQTT. MQTT protokol je već pronašao primjenu u automobilskoj industriji u nekim sličnim područjima, poput aplikacije za vezu između automobila koja se oslanja

na HiveMQ za pouzdanu povezanost [1]. Još jedan primjer u automobilskoj industriji prema [2] je EMQ koji pomaže SAIC Volkswagenu u izgradnji IoV platforme.

1.1. Zadatak diplomskog rada

Zadatak ovog diplomskog rada je omogućiti praćenje parametara stanja testnih stanica te njihovo spremanje u bazu podataka. Omogućiti izvršavanje udaljenih zadataka na testnim stanicama. Dizajnirati komunikacijske veze unutar sustava i odabrati komunikacijski protokol. Izraditi sučelje za prikaz podataka korisnicima zaduženima za nadzor. Primjena ovakvog sustava u automobilskoj industriji unaprijedila bi proces udaljenog ispitivanja programske podrške.

2. PREGLED PODRUČJA TEME

Tema ovoga rada dolazi u doticaje s nekim aktualnim sustavima i tehnologijama koje su navedene u sljedećim poglavljima. Navedene su usporedbe i prednosti te nedostaci prema autorovim spoznajama i iz navedenih izvora.

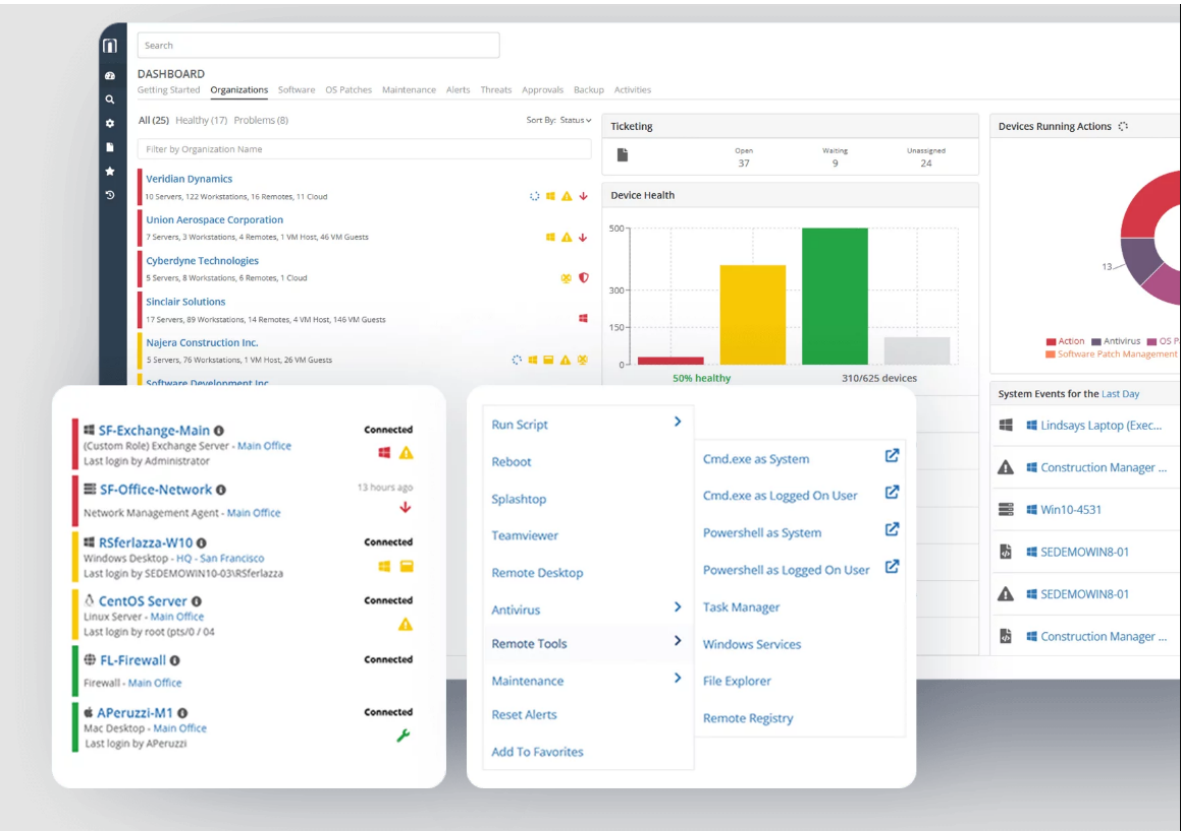
2.1. Stroj-stroj komunikacija

Komunikacija stroj-stroj (engl. *Machine-to-machine* - M2M), u nastavku teksta M2M, široka je oznaka koja se može koristiti za opisivanje bilo koje tehnologije koja omogućuje umreženim uređajima razmjenu informacija i izvođenje radnji bez ručne pomoći ljudi. Umjetna inteligencija (engl. Artificial intelligence - AI) i strojno učenje (engl. *machine learning* - ML) olakšavaju komunikaciju između sustava, dopuštajući im da donose vlastite autonomne odluke. M2M tehnologija je prvi put prihvaćena u proizvodnim i industrijskim pogonima, gdje su druge tehnologije, poput SCADA i daljinskog nadzora, pomogle u daljinskom upravljanju i kontroli podataka iz opreme i alata. M2M je od tada pronašao primjenu u različitim sektorima, poput automobilske industrije, zdravstva, poslovanja i osiguranja. Također je temelj za komunikaciju između interneta objekata (engl. *internet of things* - IoT). Za razliku od SCADA ili drugih alata za daljinski nadzor, M2M sustavi često koriste javne mreže i metode pristupa, na primjer, mobilnu ili Ethernet kako bi bili isplativiji. Takva komunikacija često se koristi za daljinski nadzor. Ovakvi sustavi unatoč brojnim prednostima suočavaju se s brojnim sigurnosnim problemima, od neovlaštenog pristupa preko bežičnog upada do hakiranja uređaja. Također se moraju uzeti u obzir fizička sigurnost, privatnost, prijevara i izloženost kritičnih aplikacija. Nekoliko ključnih M2M standarda pojavilo se zadnjih godina, poput uključujući CoAP, AMQP, STOMP, SMCP, SSI, DDS i MQTT [3]. MQTT je protokol za razmjenu poruka i podataka. Dizajniran je kao iznimno lagan prijenos poruka na principu objavljivanja i pretplate koji je idealan za povezivanje udaljenih uređaja s malim otiskom koda i minimalnom propusnošću mreže te dobrom podrškom. MQTT se danas koristi u raznim industrijama, kao što su automobilska industrija, proizvodnja, telekomunikacije, nafta i plin i brojnim drugima.

2.2. Alati za daljinsko praćenje i upravljanje

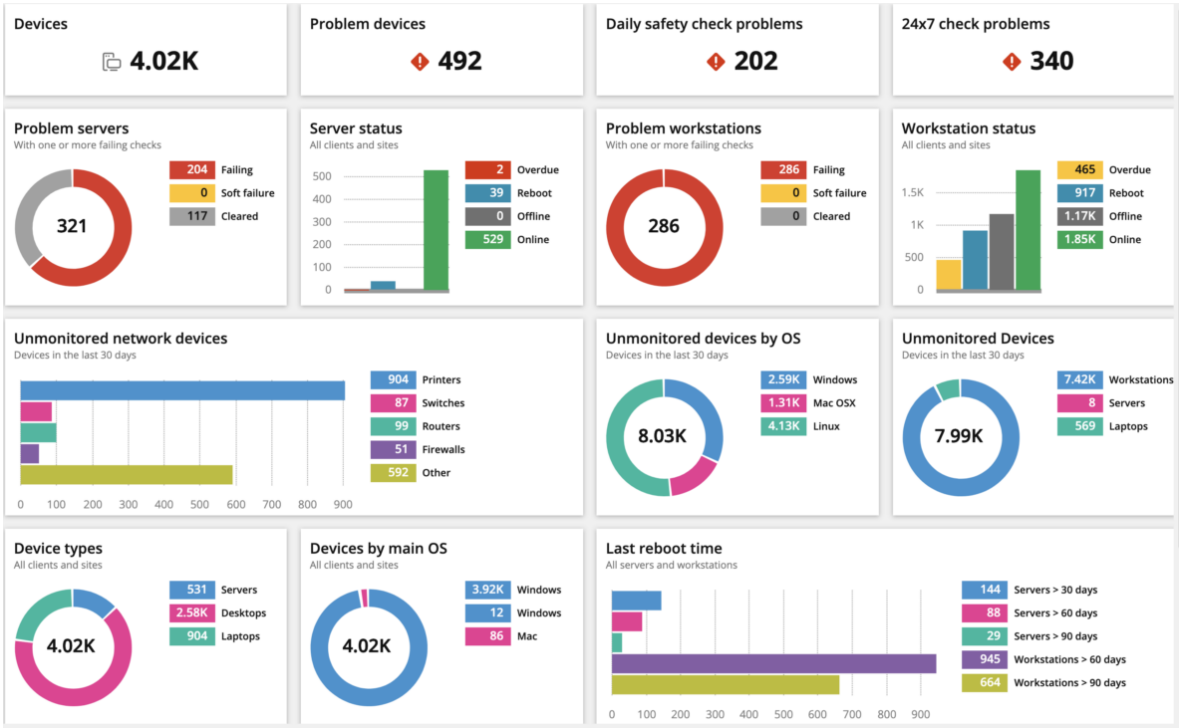
Alati za udaljeni nadzor i upravljanje (engl. *Remote Monitoring and Management* - RMM), u nastavku teksta RMM alati, mogu prikupljati korisne podatke o klijentskom softveru, hardveru i mrežama, pratiti stanje mreže i sustava te nadzirati više krajnjih točaka i klijenata. Mogu dostaviti izvješća o aktivnostima i vizualizirati podatke. U slučaju bilo kakvog problema, može generirati upozorenja i prijaviti problem. RMM alati su sve korišteniji jer donose brojne prednosti poput automatizacije procesa, poboljšanja produktivnosti i povećanje stupnja nadzora poduzeća i infrastrukture. Na izbor RMM alata utječu čimbenici poput mogućnosti i značajki alata, cijena, performanse i podrška. Tvrtke također često zbog već navedenih razloga ali i još nekih poput pitanja sigurnosti i potreba često izrađuju prilagođene vlastite alate za nadzor i upravljanje kako bi bili što prikladniji ovisno o potrebama i zahtjevima. Neki od najpoznatijih alata suvremenog vremena biti će objašnjeni u nastavku.

NinjaOne je jedan od najkorištenijih komercijalnih RMM alata. Automatizira IT operacije i rezultira nižim troškovima upravljanja, poboljšavajući učinkovitost. Nudi besplatnu obuku i usluge postavljanja, kao i dobru dokumentaciju za podršku. Korisnici dobivaju jedinstvenu upravljačku konzolu za Windows, Mac, Linux, SNMP i VMware uređaje. Međutim, značajke poput upravljanja zakrpama i verzijama mogu se automatizirati samo za Windows, Mac i Linux uređaje. Programeri mogu izraditi prilagođena upozorenja kako bi ograničili lažne rezultate i dobili analizu ispravnosti uređaja u stvarnom vremenu. Sadrži centraliziranu upravljačku konzolu te nudi prilagođena izvješća i obavijesti. Slika 2.1. prikazuje sučelje alata [4]. Za opcije kupovine i cijene potrebno je kontaktirati odjel za prodaju. Neki od potencijalnih nedostataka su česta nepotrebna upozorenja i veliki opseg značajki alata.



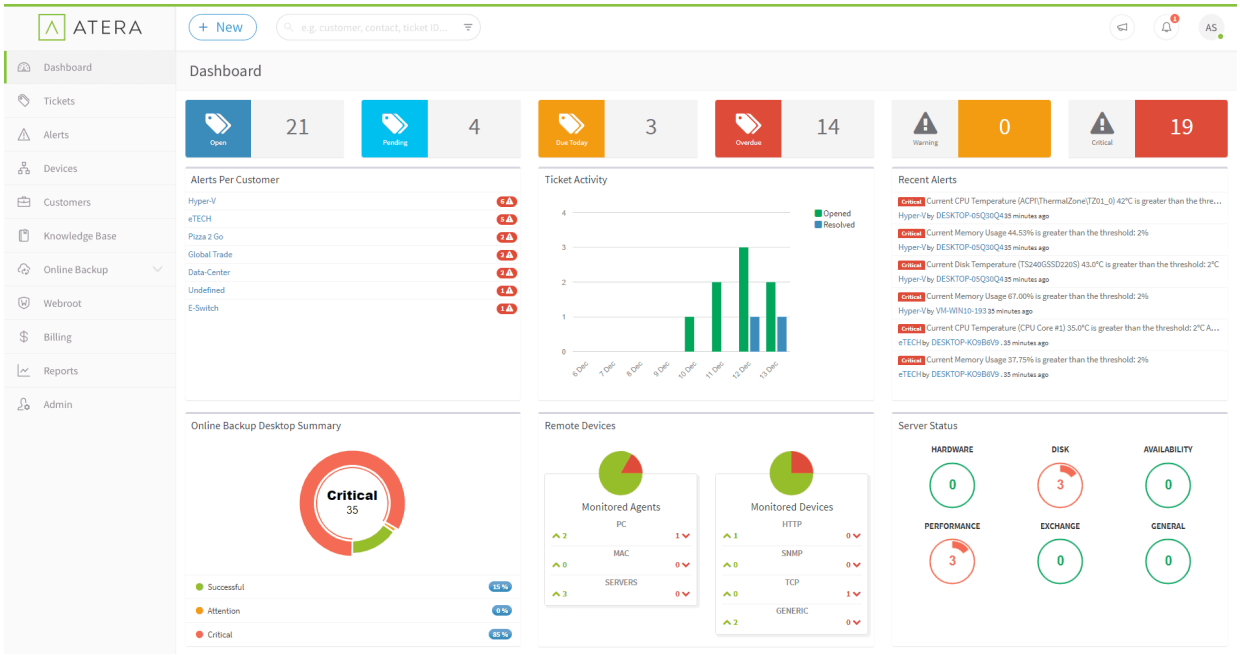
Slika 2.1. NinjaOne RMM alat.

N-Able RMM (prethodno SolarWinds) pruža automatizaciju bez potrebe pisanja prilagođenih skripti, što olakšava implementaciju platforme i početak upravljanja uređajima. Njegova slojevita sigurnosna rješenja blokiraju prijetnje koje se razvijaju, dok značajke vraćanja pomažu u zaštiti od opasnosti i pada uređaja. Korisnici dobivaju automatizirani popravak odmah po izlasku ili mogu prilagoditi pravila automatizacije sa unaprijed ugrađenim PowerShell skriptama. Platforma podržava Windows, Mac, Linux i Raspberry Pi uređaje i nudi automatiziranu podršku za uobičajene tipove aplikacija, uključujući Apple, Google, Java i Adobe. Slika 2.2. prikazuje nadzornu ploču alata [5]. Podaci o cijenama nisu dostupni na web stranici. Prednosti su jednostavnost za korištenje i implementaciju, korisna i dostupna podrška i vrlo dobra optimizacija alata tako da ne usporava krajnje uređaje.



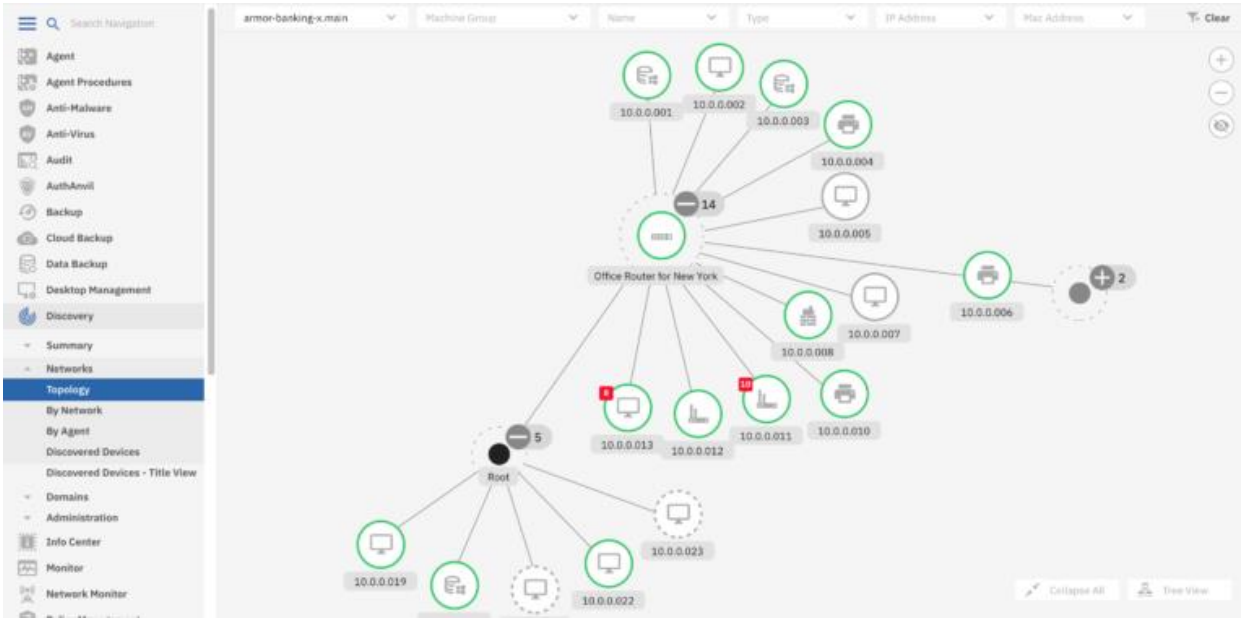
Slika 2.2. N-able RMM alat.

Atera uključuje mogućnost daljinskog pristupa, automatizaciju profesionalnih usluga, praćenje naplate i izvješća, što ga čini odličnom opcijom i za interne timove. Integrirana služba za podršku pruža dodatnu prednost. Postoje tri razine mjesečnih cijena koje tvrtke mogu birati, cijena se izražava po tehničaru. Slika 2.3. prikazuje izgled alata [6]. Tvrtke koje žele opciju live chat morat će nadograditi osnovni plan. Prednost koja nije izražena kod ostalih alata je dostupna biblioteka skripti koje su izradili korisnici.



Slika 2.3. Sučelje u Atera RMM alatu.

Kaseya VSA pruža sve u jednom (engl. *all-in-one*) rješenje za upravljanje krajnjim točkama tj. uređajima, automatizaciju i sigurnost za poboljšanje vidljivosti i operativne učinkovitosti. Korisnici mogu upravljati svim mrežnim uređajima, uključujući mobilne i uređaje internet objekata. Dodatno, sve funkcije krajnjih točaka dostupne su sa centralizirane upravljačke konzole, smanjujući vrijeme koje je tehničarima potrebno za rutinsko održavanje. Sigurnosne mjere uključuju dvofaktorsku autentifikaciju, otkrivanje prijetnji i automatsko upravljanje zakrpama i verzijama. Slika 2.4. prikazuje mrežu dodanih uređaja u alatu [7]. Kaseya se može pohvaliti jednostavnim modelom određivanja cijena ali informacije o cijenama nisu dostupne na web stranici. Jedna od prednosti je da pruža detaljnije informacije o uređajima od ostalih alata. Može biti nešto zahtjevniji za korištenje od ostalih sličnih alata.



Slika 2.4. Kaseya vsa - prikaz mreže uređaja.

3. UDALJENO ISPITIVANJE PROGRAMSKE PODRŠKE

Često puta u visoko tehnološkim industrijama poput automobilske oprema i resursi su skupi i ograničeni, također poželjno je da se ispitivanje na istima može izvršavati sa udaljenih lokacija. Zbog tih i još nekih razloga omogućeno je udaljeno ispitivanje.

3.1. Opis problema i definiranje zahtjeva

Dva glavna pristupa koja postoje u slučajevima kada dolazi do potrebe nabave softvera su kupovina komercijalno dostupnih softverskih rješenja i razvijanje vlastitog rješenja. Komercijalno dostupni softver može koristiti određenoj organizaciji jer je često povoljniji, lako dostupan te ima više funkcionalnosti. Takva rješenja najčešće ne mogu zadovoljiti specifičnosti radnih procesa neke tvrtke pa je izrada novog softvera rađenog po mjeri tvrtke često prihvatljivija opcija. Takvo rješenje može razviti vanjski dobavljač ili ga se može razviti unutar same tvrtke. Razvoj prilagođenog softvera omogućuje organizaciji u konačnici bolje i prilagođenije korištenje tehnologija za rješavanje specifičnih problema koje komercijalno dostupni alati najčešće ne rješavaju u potpunosti ili je potreban daleko veći trud kako bi ih se prilagodilo.

Zahtjevi koje je bilo potrebno ispuniti u ovome radu mogu se pronaći u nekim postojećim alatima koji su ranije navedeni. Glavni ciljevi koje je potrebno zadovoljiti su praćenje parametara stanja testnih stanica te njihovo spremanje u bazu podataka. Kako bi traženi ciljevi bili ispunjeni i kako alat ne bi sadržavao ostale nepotrebne funkcionalnosti izraditi će se novi alat i cjelokupni sustav za te potrebe. Takav pristup donosi prednosti i kod pitanja sigurnosti. Svi protokoli komunikacije, enkripcije, spremanja podataka te prava pristupa mogu se točno definirati prema vlastitim potrebama i željama. Također velika prednost je da se takav sustav može lakše i redovitije nadograđivati.

Potrebno je omogućiti **udaljeno praćenje** parametara testnih stanica što znači da će biti potrebno definirati strukturu sustava koji sadrži brojne testne stanice kao krajnje točke. Ne govoreći o formatu, testne stanice mogu biti stvarne jedinice ali isto tako i virtualne jedinice. Testne stanice biti će udaljene kako jedna od druge tako i od centralne kontrolne točke i korisnika koji će ih nadzirati preko korisničkog sučelja. Potrebno je odabrati komunikacijski protokol koji će to omogućiti unutar mreže organizacije neovisno o stvarnoj udaljenosti i lokaciji.

Parametri testnih stanica koje je potrebno nadzirati mogu se odrediti prema potrebama ovisno o njihovoj namjeni. Također prilikom izrade vlastitog alata ta značajka nije jedinstvena, što znači da

će na jednoj testnoj stanici biti moguće nadzirati na primjer parametre resursa operacijskog sustava poput zauzeća memorije, dok će na nekoj drugoj testnoj stanici biti potrebno nadzirati prijavljene korisnike ili pokrenute programe, prijavljene greške sustava i slično.

Spremanje podataka jedna je od obaveznih značajki ovoga sustava. Tim postupkom omogućiti će se prikaz podataka kroz povijest i detaljnija analiza događanja. Brojne su pogodnosti koje se mogu izvući i iskoristiti iz spremljenih podataka u bazi podataka. Potrebno je obratiti pažnju na podatke koji će se odabrati spremati te na količinu i čuvanje kroz povijest. Ispravno i sigurno postavljanje baze podataka je jedan od preduvjeta svakog ozbiljnijeg sustava koji obrađuje podatke.

Prikaz podataka korisnicima koji će biti zaduženi za nadzor potrebno je generirati koristeći sve navedeno do sada. Izraditi sučelje koje je vizualno prihvatljivo i lako za koristiti te služi glavnoj svrsi je krajnji prikaz cijeloga sustava pruženog korisnicima. Također za prikaz podataka već postoje gotovi alati koje je moguće integrirati s vlastitim podacima.

Kada svi navedeni zahtjevi budu ispunjeni takav sustav rezultirat će većom produktivnosti, profitabilnosti te boljom organizacijom tvrtke. Tehničarima će se omogućiti bolje i brže djelovanje na određeni događaj, skratiti vrijeme za određene akcije i analizu događaja koji su prethodili određenom slučaju. Također će donijeti bolje planiranje raspoređenosti i zauzeća testnih stanica menadžerima ili voditeljima projekta. U svakome trenutku imat će uvid u stanje svake testne stanice, njenu aktivnost te procese koji se odvijaju. Sustav je potrebno izraditi tako da bude lako proširiv i fleksibilan jer se broj testnih stanica može mijenjati tijekom rada sustava. Dodavanje novih testnih stanica u mrežu i komunikacijski kanal treba biti brzo i jednostavno.

3.2. Korišteni alati i tehnologije

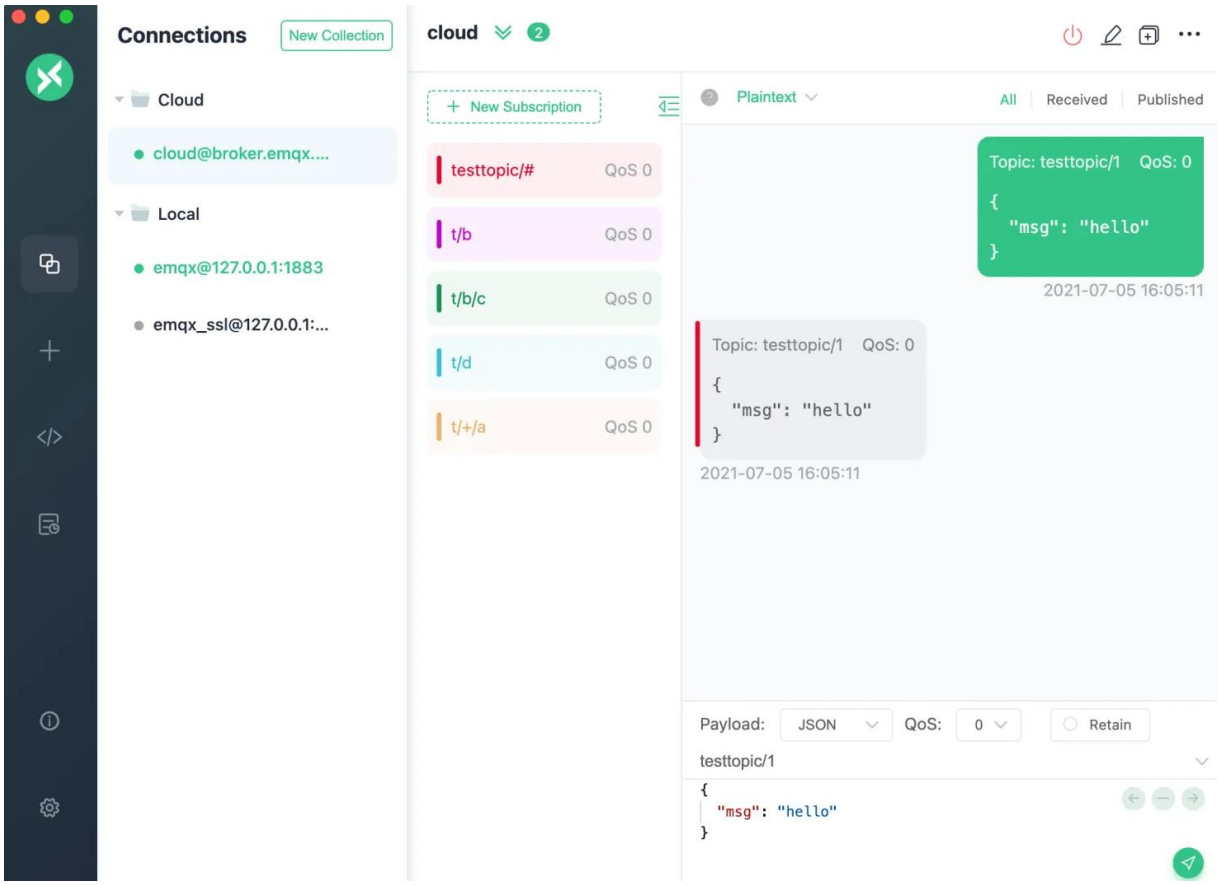
Za izradu prilagođenog alata odabran je programski jezik **Python.** Između ostaloga, omogućuje korisnicima da vide i prate status testne stanice u određenom trenutku, stvara datoteke zapisa bitnih događaja kroz vrijeme, šalje podatke u bazu podataka i ostalim jedinicama na mreži putem MQTT komunikacijskog protokola. Izrađeni python program mora biti pokrenut na testnim stanicama neovisno o njihovom operacijskom sustavu. Status može za primjer uključivati, ali nije ograničen na korištenje CPU-a, latenciju mreže, korištenje memorije, korištenje diska, prijavljenog korisnika, pokrenute procese te aktivnost. Python programski jezik pogodan je za ovu vrstu primjene jer nudi gotove biblioteke poput *psutil,* koje omogućuju dohvaćanje podataka o resursima hardvera i operacijskog sustava ali omogućuje isto tako i pisanje vlastitih. U slučaju proširenja

sustava pisanjem vlastitih prilagođenih python programa mogu se omogućiti naprednije funkcionalnosti za nadzor i upravljanje koje gotovi alati ne nude. Također odgovara za potrebe spremanje podataka i komunikaciju s bazom podataka te komunikacije putem MQTT protokola gdje postoji izvrsna učestala podrška i biblioteke za rad s navedenim.

MQTT je komunikacijski protokol za razmjenu poruka koji mrežnim klijentima sa ograničenim resursima pruža jednostavan način za distribuciju telemetrijskih informacija i u okruženjima niže propusnosti. Protokol, koji koristi komunikacijski obrazac objavljivanja i pretplate, koristi se za komunikaciju stroj-stroj. Stvoren kao protokol s niskim opterećenjem kako bi se prilagodio propusnosti i procesorskim ograničenjima, MQTT je dizajniran za rad u ugrađenom okruženju gdje može pružiti pouzdan, učinkovit način za komunikaciju. Prikladan je za povezivanje više raznih uređaja s malim otiskom koda, dobar je izbor i za bežične mreže koje imaju različite razine kašnjenja zbog povremenih ograničenja propusnosti ili nepouzdanih veza. MQTT protokol već danas ima primjenu u industrijama od automobilske preko energetske do telekomunikacija. Iako je MQTT započeo kao protokol koji se koristio za komunikaciju sa sustavima nadzorne kontrole i prikupljanja podataka poput SCADA u industriji nafte i plina, postao je popularan u području pametnih uređaja i danas je vodeći protokol otvorenog koda za povezivanje internet objekata i industrijskih internet objekata. **MQTT broker** djeluje kao posrednik između klijenata koji šalju poruke i pretplatnika koji te iste poruke primaju. U analogiji s poštanskim uredom, broker je sam poštanski ured. Sve poruke moraju proći kroz broker prije nego što se mogu isporučiti pretplatniku. Brokeri će možda morati rukovati milijunima istovremeno povezanih klijenata, pa bi ih tvrtke pri odabiru MQTT brokera trebala ocijeniti na temelju njihove skalabilnosti, integracije, nadzora i sposobnosti otpornosti na kvarove. Većina najpoznatijih brokera ima mogućnosti koje odgovaraju većim sustavima kada ih koriste za komunikaciju zbog opcija poput sigurnosti prilikom prijave klijenata gdje podržavaju više oblika autentikacije na temelju korisničkog imena, ID-a klijenta, HTTP-a, JWT-a, LDAP-a i raznih baza podataka kao što su MongoDB, MySQL, PostgreSQL te Redis. Isto tako postoji kontrola prava pristupa klijenata, točnije objavljivanja i pretplate na određene teme i komunikacijske puteve koje je moguće postaviti i kontrolirati istim navedenim tehnologijama. Protokol se neprestano poboljšava i sada podržava protokol WebSocket, koji omogućuje dvosmjernu komunikaciju između klijenata i brokera u stvarnom vremenu. Kao primjer nekih ažuriranja, verzija 5.0 uključivala je bolje izvješćivanje o pogreškama, uključivanje metapodataka u zaglavljima poruka, opcije dijeljenih pretplata, nestajanje poruka i sesija te druge novitete. Neki od najčešće korištenih brokera su Mosquitto, EMQ X, Cassandana, HiveMQ, AWS IoT Core MQTT.

Odabrani broker za potrebe MQTT protokola je EMQX, izrađen od strane tvrtke EMQ. EMQX je MQTT broker otvorenog koda s visokoučinkovitim mehanizmom za obradu poruka u stvarnom vremenu, koji pokreće strujanje događaja za IoT uređaje u velikim razmjerima. Kao najskalabilniji MQTT broker, EMQX omogućuje povezivanje bilo kojih uređaja, u bilo kojoj mjeri. Prednosti su mu velika skalabilnost, do 100 milijuna istodobnih MQTT klijentskih veza. Visoke performanse, slanje i obrada milijun MQTT poruka u sekundi na jednom brokeru. Niska latencija ispod milisekunde i isporuku poruka u stvarnom vremenu. Potpuno je usklađen s MQTT 5.0 i 3.x standardom za bolju skalabilnost, sigurnost i pouzdanost. Još jedna prednost je visoka dostupnost i skalabilnost kroz distribuiranu arhitekturu. Implementacija je jednostavna lokalno ili u oblaku uz Kubernetes Operator i Terraform. [8] Jedan od razloga odabira EMQX brokera su performanse analizirane u pronađenom radu koji uspoređuje nekoliko najpoznatijih brokera u različitim scenarijima [9].

Postoje brojni MQTT klijentski alati koji pomažu pri postavljanju i korištenju brokera i protokola. Jedan od njih je **MQTT X**, višeplatformski MQTT 5.0 klijentski alat koji je izradio EMQ, a koji se može koristiti na macOS-u, Linuxu i Windowsu. Podržava ujedno i kreiranje i formatiranje MQTT poruka [10], prikaz izgleda aplikacije prikazano je na slici 3.1. Pojednostavljuje testni rad uz pomoć poznatog sučelja nalik chatu. Lako je i brzo stvoriti višestruke, simultane veze MQTT klijenata i može testirati funkcije povezivanja, objavljivanja i pretplate MQTT/TCP, MQTT/TLS, MQTT/WebSocket kao i druge značajke MQTT protokola.



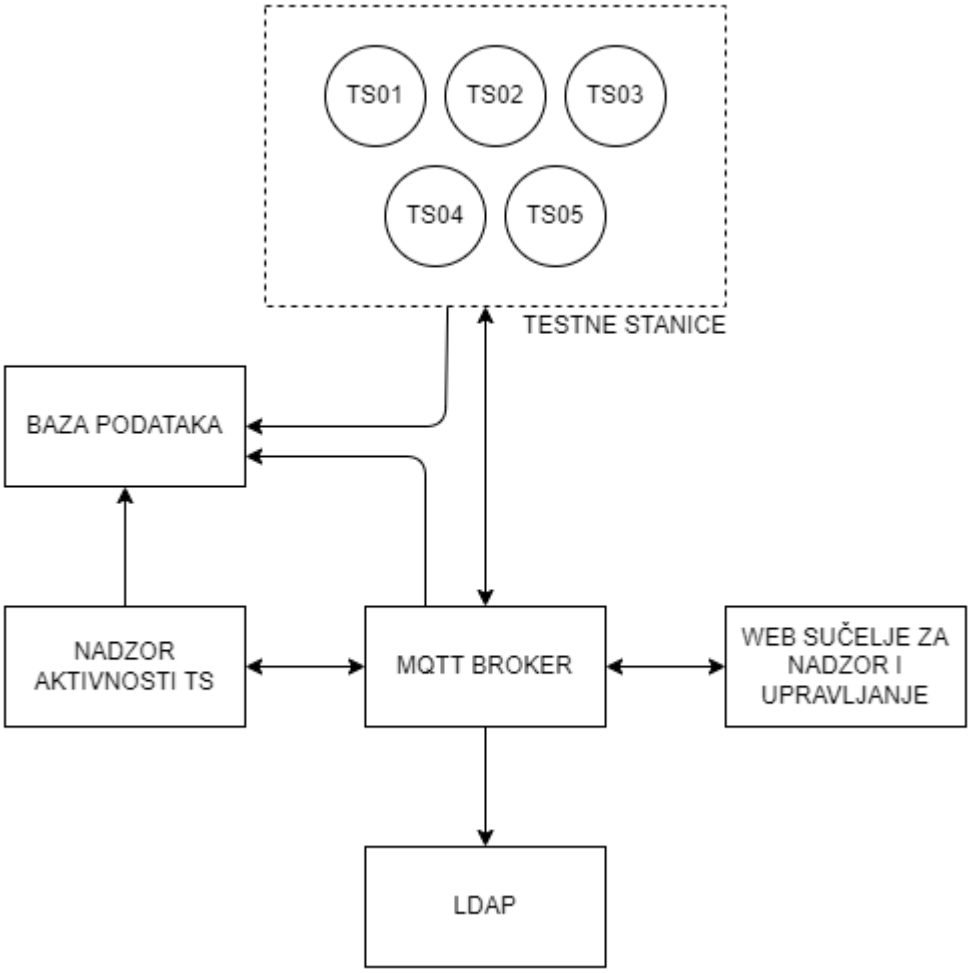
Slika 3.1. MQTTX.

Baza podataka je organizirani skup strukturiranih informacija ili podataka, obično elektronički pohranjenih u računalnom sustavu. Bazom podataka obično upravlja sustav za upravljanje bazom podataka. Podaci i sustav za upravljanje bazom podataka, zajedno s aplikacijama koje su s njima povezane, nazivaju se sustavom baze podataka, često skraćeno na samo baza podataka. Podaci unutar najčešćih vrsta baza podataka koje danas rade obično su modelirani u redovima i stupcima u nizu tablica kako bi obrada i upiti podataka bili brzi i učinkoviti. Podacima se tada može lako pristupiti, njima se može upravljati, mijenjati, ažurirati, kontrolirati i organizirati. Većina baza podataka koristi strukturirani jezik upita (engl. *sequential query language* - SQL) za pisanje i postavljanje upita podacima.

LDAP (engl. *Lightweight Directory Access Protocol)* je fleksibilan i dobro podržan mehanizam koji se temelji na standardima za interakciju s poslužiteljima. Često se koristi za provjeru autentičnosti i pohranjivanje informacija o korisnicima, grupama i aplikacijama, ali LDAP poslužitelj je pohrana podataka opće namjene i može se koristiti u velikom broju aplikacija.

Prikaz i analiza podataka kao pojam nema jedinstvenu definiciju ili alat. Često puta tvrtke i organizacije koriste komercijalno dostupne alate koji se nude za prikaz podataka i analizu koji odgovaraju njihovim potrebama. No isti ponekad ne odgovaraju za sve sustave i potrebe pa se podjednako često izrađuju vlastiti prilagođeni alati koristeći na primjer dostupne web tehnologije kao glavni alat zbog jednostavnosti izrade i pogodnosti za dani zadatak. Prilikom izrade vlastitog alata moguće je prilagoditi svaki prikaz prema potrebi i podacima tako da bude što prikladniji, također moguće je ukomponirati više različitih funkcionalnosti ili sustava u jedan alat i nadograđivati ga samostalno po potrebama sustava.

3.3. Prijedlog komunikacijskog protokola

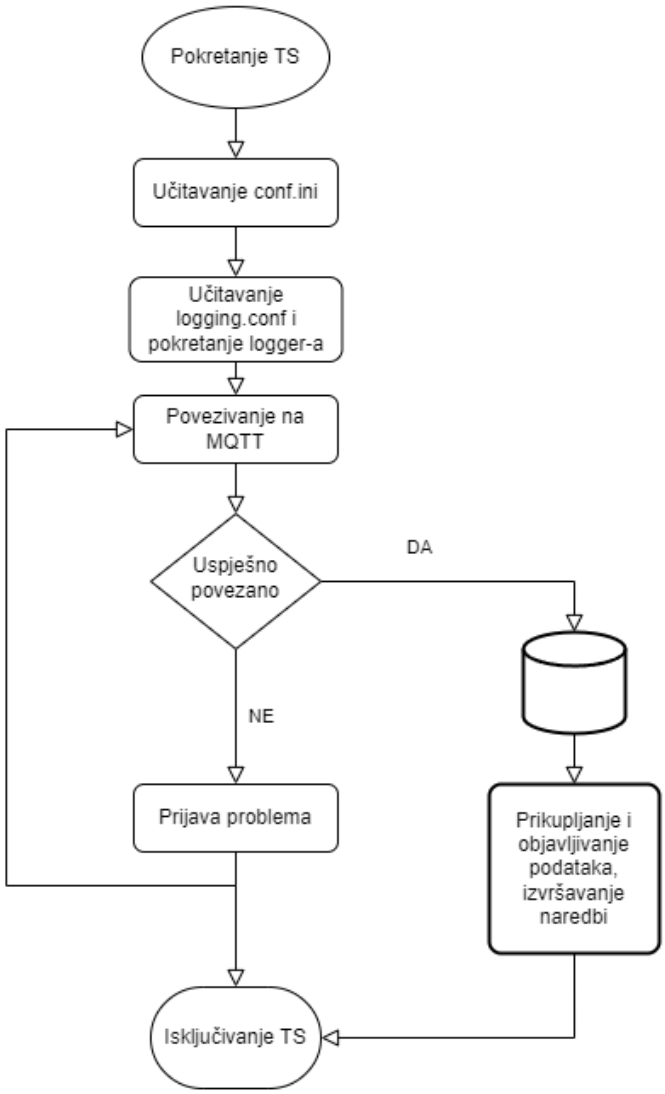


Slika 3.2. Blok dijagram toka komponenti i veza sustava.

Sustav se sastoji od međusobno povezanih manjih komponenti podsustava koje su prikazane na slici 3.2. Broj testnih stanica simbolično je prikazan te nije ograničen u stvarnoj primjeni, isto vrijedi i za broj web sučelja za nadzor i upravljanje. MQTT broker može predstavljati jednog postavljenog brokera ili više njih u grupi (engl. *cluster*). Prikazani su komunikacijski putevi između pojedinih podsustava koji mogu ostvariti razmjenu informacija ili podataka. U nekim slučajevima kao što je prikazano veze su jednosmjerne, no u nekima kao u slučajevima komunikacije s MQTT brokerom i dvosmjerne.

3.3.Implementacija rješenja

Kako bi se omogućio nadzor testnih stanica, u okviru ovog diplomskog rada, izrađen je python program zadužen za prikupljanje podataka o testnoj stanici, prijavu testne stanice kao MQTT klijenta, kako bi se omogućila komunikacija unutar sustava, objavljivanje prikupljenih podataka te spremanje aktivnosti i podataka u bazu podataka. Također kao dodatna funkcionalnost omogućeno je izvršavanje određenih zadataka na testnim stanicama putem web sučelja, poput pokretanja programa. Program je postavljen tako da se automatski pokrene pri svakom pokretanju testne stanice. Okvirna procedura i tijek algoritma testnih stanica prikazani su blok dijagramom na slici 3.3.



Slika 3.3. Blok dijagram toka Python programa testnih stanica.

Kao prvi korak algoritma potrebno je učitati konfiguracijsku (engl. *config*) datoteku i izdvojiti potrebne podatke. Ovaj korak je važan kako bi se testne stanice razlikovale jedna od druge te kako bi svaka imala svoje jedinstvene identifikatore i slično. Također config datoteka može biti velika prednost u budućnosti kada dođe do ažuriranja sustava ili dodavanja novih funkcionalnosti. Datoteka se sastoji od odjeljaka, od kojih svaki sadrži ključeve s vrijednostima. Biblioteka *configparser* omogućuje čitanje i pisanje takve datoteke.

```
[DEFAULT]
Version = 1.0.0
Update = http://20.30.20.51:5000/update

[MQTT]
Host = http://20.30.20.51/
Port = 1883
User = mqtt-RTS0012
Password = Passw0rd
Base_topic = CompanyName/ToolName/remote/

[TEST-STATION]
ID = RTS0012
Name = Remote-TS-0012
Password = Passw0rd

[UTILS]
API_base_url = http://20.30.20.51:5000/
```

Slika 3.4. Primjer sadržaja conf.ini datoteke.

Na slici 3.4. vide se neki od podataka koji se mogu nalaziti u config datoteci. Pojedini parametri poput korisničkog imena, lozinke ili verzije su jedinstveni za svaku testnu stanicu. Iz tog razloga config datoteke su jedinstvene za pojedinu testnu stanicu.

```
def config_handler(file):
    config = configparser.ConfigParser()
    config.read(file)

    username = config['MQTT']['User']
    password = config['MQTT']['Password']
```

Slika 3.5. Naredbe za čitanje konfiguracijske datoteke.

Učitavanje datoteke i čitanje vrijednosti ključeva User i Password pod odjeljkom MQTT prikazane su na slici 3.5.

```
[loggers]
keys=root,REMOTElogger

[handlers]
keys=consoleHandler,fileHandler

[formatters]
keys=form01,form01

[logger_root]
level=DEBUG
handlers=consoleHandler

[logger_REMOTElogger]
level=DEBUG
handlers=fileHandler
qualname=REMOTElogger
propagate=0

[handler_consoleHandler]
class=StreamHandler
level=DEBUG
formatter=form01
args=(sys.stdout,)

[handler_fileHandler]
class=handlers.RotatingFileHandler
level=DEBUG
args=('logs/REMOTE.log','a',1000*10*10,7)
formatter=form01

[formatter_form01]
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s

[formatter_form02]
format=%(levelname)s - %(message)s
```

Slika 3.6. Primjer sadržaja logging.conf datoteke.

Prilikom izrade sustava analizirani su ispisi u konzolu, prijave i greške programa koje su se pojavljivale, no kako bi se to pojednostavilo i poboljšalo u budućnosti implementiran je python zapisivač (engl. *logger*), dalje u tekstu logger. Logger je modul koji pruža puno funkcionalnosti i fleksibilnosti. Zadužen je za prikazivanje i zapisivanje određenih događaja tokom rada programa.

Brojni parametri i opcije mogu se postaviti kako bi prilagodili njegov rad, odredište, format i filtere. Na slici 3.6. prikazana je datoteka postavki dva kreirana loggera te različite formate zapisa.

```
import logging
import logging.config

def start_logger():
    logging.config.fileConfig('logging.conf')

    logger = logging.getLogger('REMOTElogger')

    return logger
```

Slika 3.7. Funkcije za inicijalizaciju Python logger-a.

Slika 3.7 prikazuje na koji način se u programu inicijalizira te pokreće logger. U isto vrijeme moguće je imati i više aktivnih. Jedan od slučajeva kada bi se koristila dva ili više aktivna je kada je potrebno istovremeno i zapisivati u datoteku ali i prikazivati te ispisivati u korisničkoj konzoli.

```
logger.debug('This message should go to the log file')
logger.info('So should this')
logger.warning('And this, too')
logger.error('And non-ASCII stuff, too, like Øresund and Malmö')
```

Slika 3.8. Primjer zapisivanja događaja različitih razina.

Primjeri zapisivanja bilješki različitih razina važnosti u datoteke mogu se vidjeti na slici 3.8. Dostupne razine ovisno o vrsti događaja su debug, info, warning i error. Funkcija *setLevel* postavlja prag razine za zapisivanje. Poruke zapisivanja koje su manje razine bit će zanemarene. Poruke zapisivanja koje imaju istu razinu ozbiljnosti ili višu će biti obrađene.

Nakon uspješnog pokretanja programa te čitanja konfiguracijskih datoteka potrebno je testnu stanicu povezati na MQTT tako da kontaktira MQTT broker s potrebnim argumentima (Slika 3.9.)

```
client = mqtt.client.Client(client_id = id, clean_session = True)
client.username_pw_set(username, password)
client.will_set("CompanyName/ToolName/TS/status", payload = username +
" gone Offline - last will msg", qos = 1, retain = False)
client.on_connect = on_connect
client.connect(broker, port)
```

Slika 3.9. Funkcija spajanja klijenta na MQTT.

Testna stanica, korištenjem MQTT protokola, šalje brokeru zahtjev za prijavom. Prijava funkcionira tako da klijent mora poslati *CONNECT* zahtjev za uspostavljanje MQTT veze. MQTT *CONNECT* zahtjev sastoji se od tri vrijednosti koje mora sadržavati: *clientID, cleanSession i keepAlive* i neobaveznih, kao što su *korisničko ime, lozinka, lastWillTopic, lastWillQos, lastWillMessage i lastWillRetain*. Postavljena je i *last will* poruka koja se čuva u mqtt brokeru, a šalje se kada dođe do odspajanja klijenta ili gubitka veze s klijentom.

```
def subscribe(client: mqtt_client, topic):
    def on_message(client, userdata, msg):
        print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")

        msg = json.loads(msg.payload.decode())
        task = msg['task']
        REMOTE_task_handler(client, task, msg)

    client.subscribe(topic, qos = 1)
    client.on_message = on_message
```

Slika 3.10.. Funkcija pretplate klijenta na temu.

```python
def publish(client, topic, msg):
    result = client.publish(topic, msg, qos=1)
    status = result[0]
    if status == 0:
        print(f"Send `{msg}` to topic `{topic}`")
    else:
        print(f"Failed to send message to topic {topic}")
    return msg
```

Slika 3.11. Funkcija objavljivanja poruke na temu.

Ovisno o predanim argumentima i postavkama broker obavlja provjere u LDAP-u i bazi podataka. Ako je klijent pružio ispravne i valjane podatke broker omogućuje prijavu. U programu se tada poziva *on_connect* funkcija i klijent ima prava pretplatiti se i objavljivati na teme (Slika 3.10, Slika 3.11.)

getTSSessionID	Prima argument testStationID - niz znakova. Sprema u bazu podataka i vraća pozivatelju generirani SessionID - niz znakova.
deleteTSSessionID	Prima argument testStationID - niz znakova. Obavlja radnju brisanja SessionID-a iz baze podataka za određenu testnu stanicu.
updateTSActivity	Prima argumente testStationID - niz znakova, status - cjelobrojna vrijednost 0,1 ili 2, vrijeme - niz znakova. Sprema u bazu podataka događaj o aktivnosti testne stanice ovisno o argumentu status koji primi.

Tablica 3.12. Pozivi prema bazi podataka.

U tablici 3.11. su prikazani neki od dostupnih poziva aplikacijsko - programskog sučelja. HTTP zahtjev *getTSSessionID* vraća pozivatelju generirani *SessionID*. Navedena vrijednost je tipa niza četrtdeset i osam znakova i generira se nasumično pomoću python naredbe *str(binascii.hexlify(os.urandom(24)))*. Prilikom svakog poziva obavlja se potrebna radnja i izmjene u bazi podataka. Postavljeni su preduvjeti za sve funkcije koji bi zaštitili iskorištavanje

ovih dijelova sustava tako da se na primjer *getTSSessionID* ne može izvršiti ako testna stanica nije aktivna ili je isključena.

```
def memory_treshold():
    mem = psutil.virtual_memory()
    THRESHOLD = 100 * 1024 * 1024  #100MB

    if mem.available <= THRESHOLD:
        return (True, mem.available)
    else:
        return (False, mem.available)
```

Slika 3.13. Funkcija za dohvaćanje podataka o memoriji.

Nakon dijela postavljanja i prijave algoritam ulazi u fazu rada koja traje do isključenja testne stanice. Radnje i funkcije koje izvršava ovise o tome što se od testne stanice zahtjeva putem sučelja za nadzor i upravljanje. Neke od dostupnih radnji su nadzor resursa operacijskog sustava, nadzor hardvera, informacije o prijavljenim korisnicima, pokretanje programa i naredbi i drugih. Dostupne funkcionalnosti na testnim stanicama lako je modificirati i ažurirati novim funkcijama. Testne stanice imaju mogućnost automatskog preuzimanja novijih verzija programa i konfiguracijskih datoteka. Primjerom na slici 3.14. naveden je slučaj kada korisnik zatraži informacije o određenom procesu poput naziva, proces id-a, vremena pokretanja. Također može dobiti informaciju o tome od testne stanice kada se proces prekine.

```
def on_process_start(process):
    for proc in psutil.process_iter():
        if(proc.name() == process):
            time = proc.create_time()
            formated = datetime.datetime.fromtimestamp(
                time).strftime("%H:%M:%S")
            return (formated, proc.pid, proc.name)

def on_terminate(proc):
    print("Process {} terminated".format(proc))
    process_terminated_handler(proc)
```

Slika 3.14. Funkcija za nadziranje pokrenutih procesa.

Unatoč brojnim dostupnim MQTT brokerima u oblaku odabrano je postavljanje vlastitoga na osobno računalo koje je služilo kao poslužitelj za sve servise sustava zbog sigurnosti i izrade zatvorenog sustava. Simulirano je okruženje servera koristeći operacijski sustavu Ubuntu 20.04 bez grafičkog sučelja na koji se povezivalo putem SSH. Broker je postavljen u Docker kontejner, pokretan je zajedno s ostalim servisima koristeći docker-compose.

```
services:
  emqx:
    image: emqx/emqx:latest
    container_name: emqx
    restart: always
    depends_on:
      - mysql
    ports:
      - 1883:1883
      - 18083:18083
      - 8083:8083
      - 8081:8081
    environment:
      - EMQX_NAME=wbt_emqx
      - EMQX_LOADED_PLUGINS=emqx_auth_mysql,emqx_recon,emqx_retainer,emqx_management,emqx_dashboard
      - EMQX_ALLOW_ANONYMOUS=false
      - EMQX_AUTH__MYSQL__SERVER=20.30.20.51:3306
      - EMQX_AUTH__MYSQL__USERNAME=mqtt
      - EMQX_AUTH__MYSQL__PASSWORD=1234
      - EMQX_AUTH__MYSQL__DATABASE=testdb
      - EMQX_AUTH__MYSQL__PASSWORD_HASH=sha256,salt
      - EMQX_AUTH__MYSQL__AUTH_QUERY="SELECT password,salt FROM mqtt_user WHERE
        username = '%u' LIMIT 1"
    volumes:
      - vol-emqx-data:/opt/emqx/data
      - vol-emqx-etc:/opt/emqx/etc
      - vol-emqx-log:/opt/emqx/log
```

Slika 3.15. EMQX docker-compose.

Na slici 3.15. prikazan je sadržaj datoteke docker-compose za servis EMQX. Potrebno je otvoriti portove za komunikaciju i za nadzornu ploču. Također je moguće definirati određene postavke poput nekih za bazu podataka izravno u docker-compose-u, no isto tako i u nadzornoj ploči naknadno. Postavljene su datoteke za spremanje podataka iz kontejnera kako bi se osigurala postojanosti podataka tijekom rada u kontejnerima.

```
# etc/plugins/emqx_auth_ldap.conf
auth.ldap.servers = 127.0.0.1
auth.ldap.port = 389
auth.ldap.pool = 8

auth.ldap.bind_dn = cn=root,dc=emqx,dc=io

auth.ldap.bind_password = public

auth.ldap.timeout = 30s

auth.ldap.device_dn = ou=device,dc=emqx,dc=io

auth.ldap.match_objectclass = mqttUser

auth.ldap.username.attributetype = uid

auth.ldap.password.attributetype = userPassword
```

Slika 3.16. Ldap postavke MQTT broker-a.

LDAP provjerava autentičnost koristeći vanjski LDAP poslužitelj kao izvor autentikacijskih podataka, koji može pohraniti veliku količinu podataka i olakšati integraciju s vanjskim sustavima upravljanja uređajima. Na slici 3.16. prikazana je integracija MQTT brokera sa LDAP serverom kako bi služio za potrebe autentifikacije korisnika.

```
ldap_server_admin:
  container_name: ldap_admin_php
  image: osixia/phpldapadmin:0.7.2
  ports:
    - 8092:80
  environment:
    PHPLDAPADMIN_LDAP_HOSTS: ldap_server
    PHPLDAPADMIN_HTTPS: 'false'
```

Slika 3.17. Servis LDAP admin u docker-compose.

Slika 3.17. prikazuje docker-compose za alat ldap admin php koji olakšava, vizualizira i omogućuje dodavanje novih korisnika, grupa, podataka o korisnicima u ldap server, to jest bazu korisnika.

```
# etc/plugins/emqx_auth_mysql.conf
auth.mysql.server = 20.30.20.51:3306

auth.mysql.pool = 8

auth.mysql.username = emqx

auth.mysql.password = public

auth.mysql.database = mqtt

auth.mysql.query_timeout = 5s
```

Slika 3.18. MySQL postavke MQTT brokera.

EMQX broker nudi integraciju s brojnim sustavima pa tako i s bazom podataka MySQL za potrebe autentifikacije korisnika i za provjeru prava pristupa korisnika. Na slici 3.18. prikazan je izgled datoteke potrebne za povezivanje MQTT brokera i baze podatka. Neki od podataka koji su potrebni brokeru kako bi ostvario uspješno povezivanje su mrežna adresa baze podataka, korisničko ime te lozinka korisnika koji je poželjno da bude stvoren samo za potrebe brokera te naziv baze u kojoj je tablica sa MQTT klijentskim podacima.

```
CREATE TABLE `mqtt_user` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(100) DEFAULT NULL,
  `password` varchar(100) DEFAULT NULL,
  `salt` varchar(35) DEFAULT NULL,
  `is_superuser` (1) DEFAULT 0,
  `created` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `mqtt_username` (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

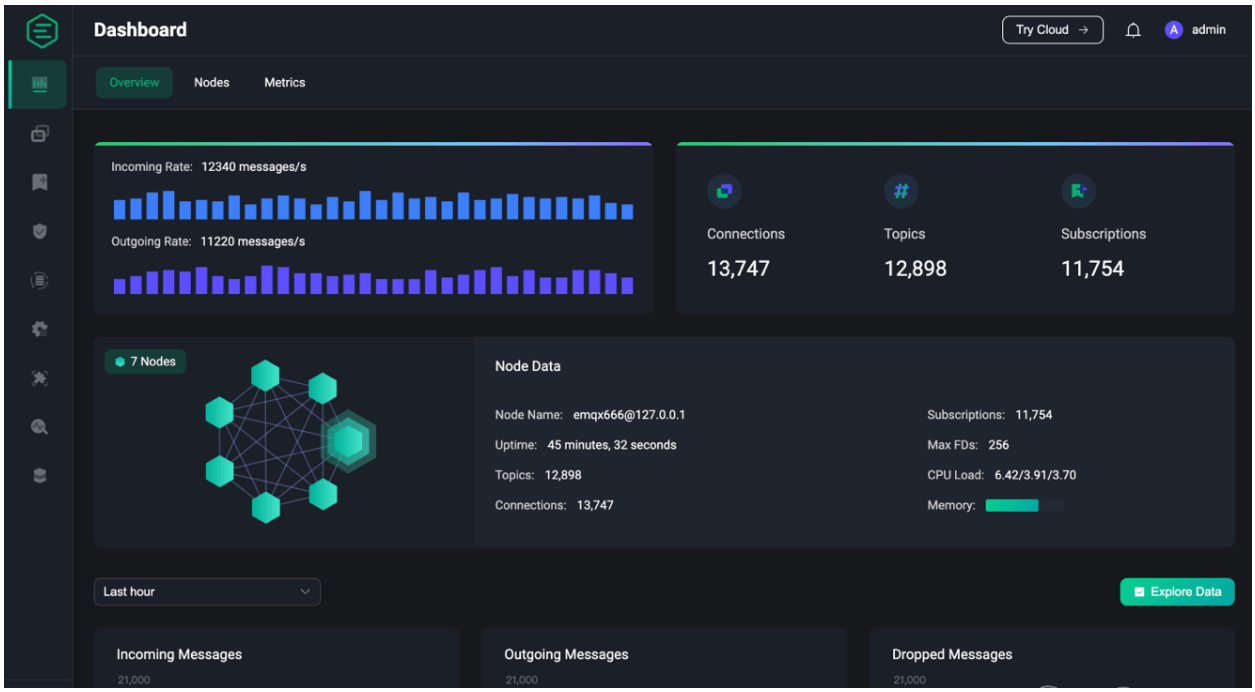
Slika 3.19. Struktura tablice MQTT klijenata u MySQL bazi podataka.

Prava pristupa služe kako bi se ograničilo i dopustilo određenim korisnicima pretplatu i objavljivanje na određene teme. Korisna su u hijerarhiji svake tvrtke jer nemaju svi zaposlenici ili klijenti iste razine prava te je to na ovaj način moguće kontrolirati. Slika 3.19. predstavlja strukturu tablice o MQTT klijentima u bazi podataka.

```
CREATE TABLE `mqtt_acl` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `allow` int(1) DEFAULT 1 COMMENT '0: deny, 1: allow',
  `ipaddr` varchar(60) DEFAULT NULL COMMENT 'IpAddress',
  `username` varchar(100) DEFAULT NULL COMMENT 'Username',
  `clientid` varchar(100) DEFAULT NULL COMMENT 'ClientId',
  `access` int(2) NOT NULL COMMENT '1: subscribe, 2: publish, 3: pubsub',
  `topic` varchar(100) NOT NULL DEFAULT '' COMMENT 'Topic Filter',
  PRIMARY KEY (`id`),
  INDEX (ipaddr),
  INDEX (username),
  INDEX (clientid)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Slika 3.20. Struktura tablice u MySQL bazi u kojoj su određena prava pristupa.

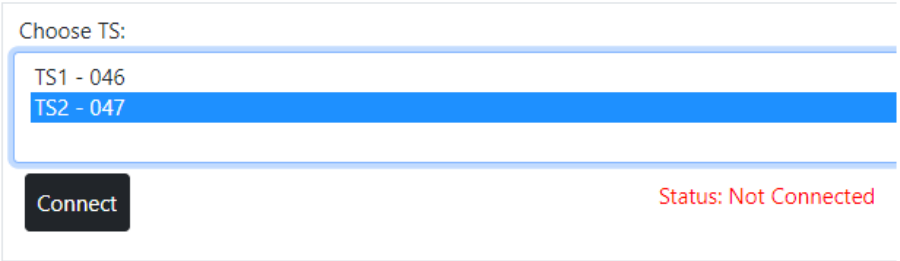
Spomenuta prava pristupa korisnika definiraju se također određenom strukturom u tablici baze podataka (Slika 3.20.) Prema ovoj strukturi broker šalje upit bazi podataka koja daje odgovor na temelju kojega broker odlučuje ima li klijent pravo na traženu radnju poput pretplate ili objave poruke.



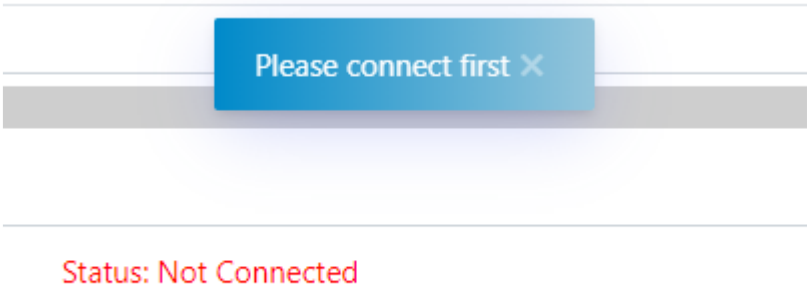
Slika 3.21. EMQX nadzorna ploča.

Jedna od prednosti EMQX-a je nadzorna ploča koju je moguće aktivirati i u kojoj se mogu mijenjati postavke rada brokera, nadzirati broker i klijenti te protok podataka (Slika 3.21.). Također moguće je ovim putem obaviti povezivanje s bazom podataka ili LDAP-om.

Web sučelje koje služi za udaljeni nadzor i upravljanje nije bilo primarni cilj ovoga rada no izrađene su određene forme za potrebe testiranja i prikaza zamišljenih funkcionalnosti. Web sučelje kreirano je koristeći python Django i pomoćne alate poput Bootstrap frameworka.

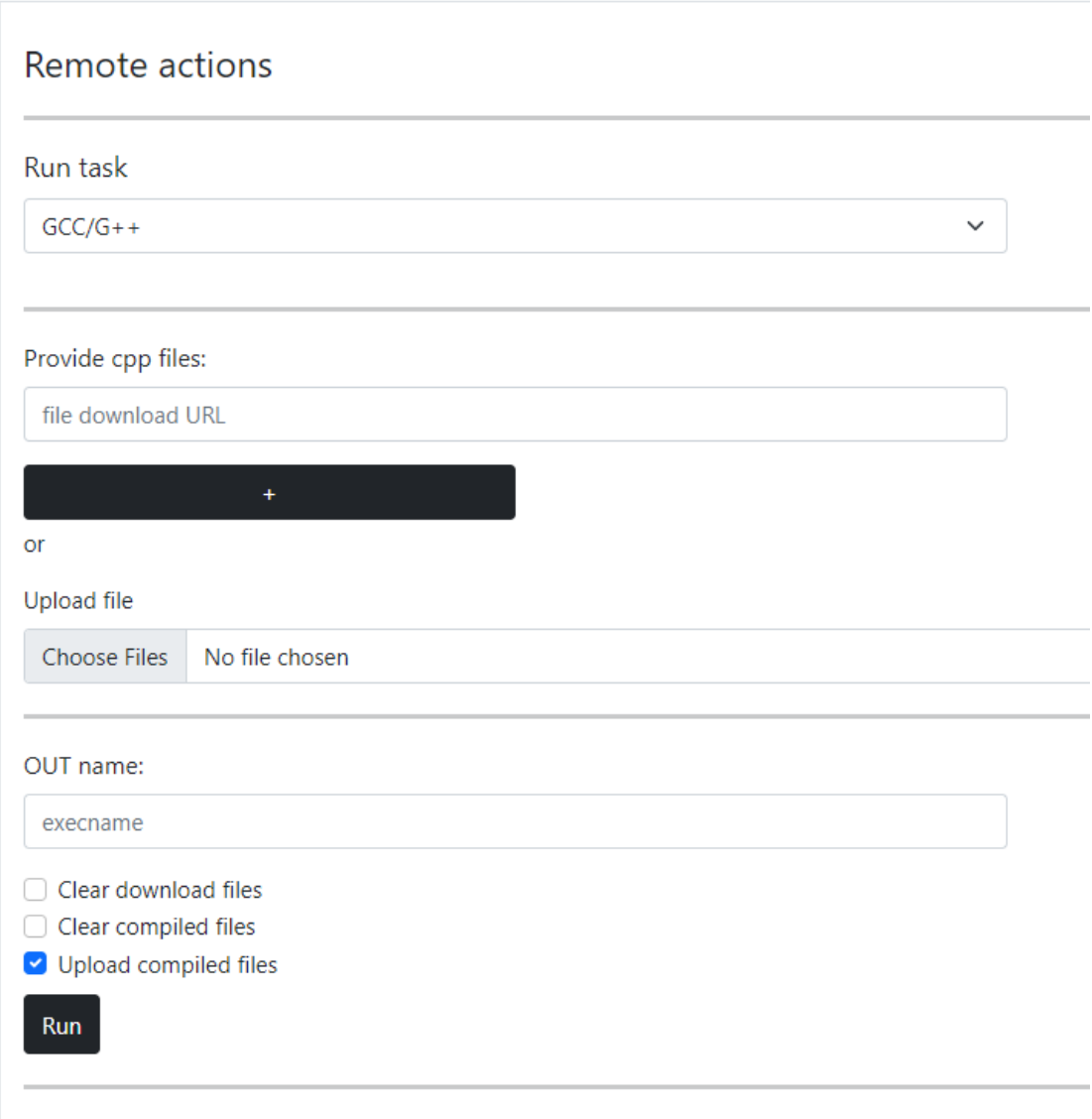


Slika 3.22. Forma za povezivanje na testnu stanicu.



Slika 3.23. Forma upozorenja.

Kako bi korisnik mogao obaviti radnje na željenoj testnoj stanici potrebno je povezati se na nju (Slika 3.22). Taj korak ne predstavlja izravno povezivanje na testnu stanicu već postavljanje komunikacijskih puteva i MQTT tema kako bi se omogućilo kontaktiranje te određene testne stanice. Cijelo sučelje omotano je uvjetima kako bi rad bio ispravan i ne bi dolazilo do grešaka u radu. Tako je na slici 3.23. prikazana obavijest ako korisnik pokuša obavljati radnje dok nije povezan na testnu stanicu. Također ako korisnik ne želi nadzirati samo jednu testnu stanicu ili obavljati dostupne udaljene akcije na njoj, stvoren je prikaz gdje se nalazi popis svih testnih stanica te njihova aktivnost.



Slika 3.24. Prikaz zahtjeva za izvršenje GCC zadatka na testnoj stanici.

Kako bi se ispitale mogućnosti sustava, u smislu složenijih funkcionalnosti, implementirana je funkcionalnost prevođenja datoteka C ili C++ pomoću prevoditelja XY.... Potrebno je priložiti željene datoteke ili url na kojem su dostupne za preuzeti, zadati naziv koji će kompajler koristiti pri generiranju exe datoteke. Također moguće je označiti određene mogućnosti poput čišćenja datoteka na testnoj stanici nakon završetka i vraćanje stvorene datoteke ako je uspješno stvorena. U slučaju da kompajler pronađe grešku korisnik će biti obaviješten o tome i dobit će ispis kompajlera. Na slici 3.24. prikaz je forme za navedenu udaljenu radnju. Treba spomenuti da je svaki zahtjev prema testnoj stanici jedinstven prema *requestID* parametru koji se šalje zajedno sa zahtjevom kako ne bi došlo do miješanja podataka.

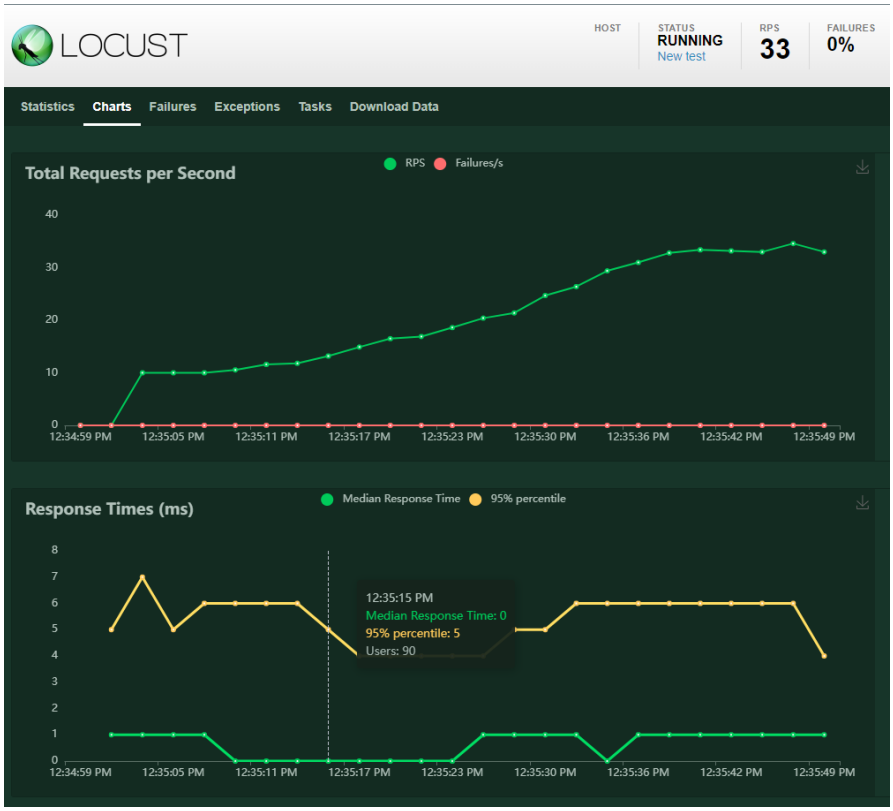
4. ISPITIVANJE MOGUĆNOSTI SUSTAVA

4.1. Simuliranje rada sustava

Kako bi se ispitao rad sustava korišteno je osam osobnih računala koje su simulirale po jednu testnu stanicu i jednog korisnika zaduženog za nadzor i upravljanje prijavljenog na web sučelje. Međusobno su bili povezani u više različitih struktura, prva je tako da svaka testna stanica bude spojena s jednim klijentom, zatim da na jednu testnu stanicu bude spojeno više korisnika za nadzor i istovremeno šalju zahtjeve. U oba slučaja sustav je radio bez grešaka, početna pretpostavka prije testiranja jest da je osam računala zanemarivo malo opterećenje za ovakav sustav te je u ovom slučaju jedino bilo moguće otkriti proceduralne greške i greške u radu algoritama ili komunikacije, no do toga nije došlo. Rezultat je da se sustav ponašao ispravno i zamišljeno.

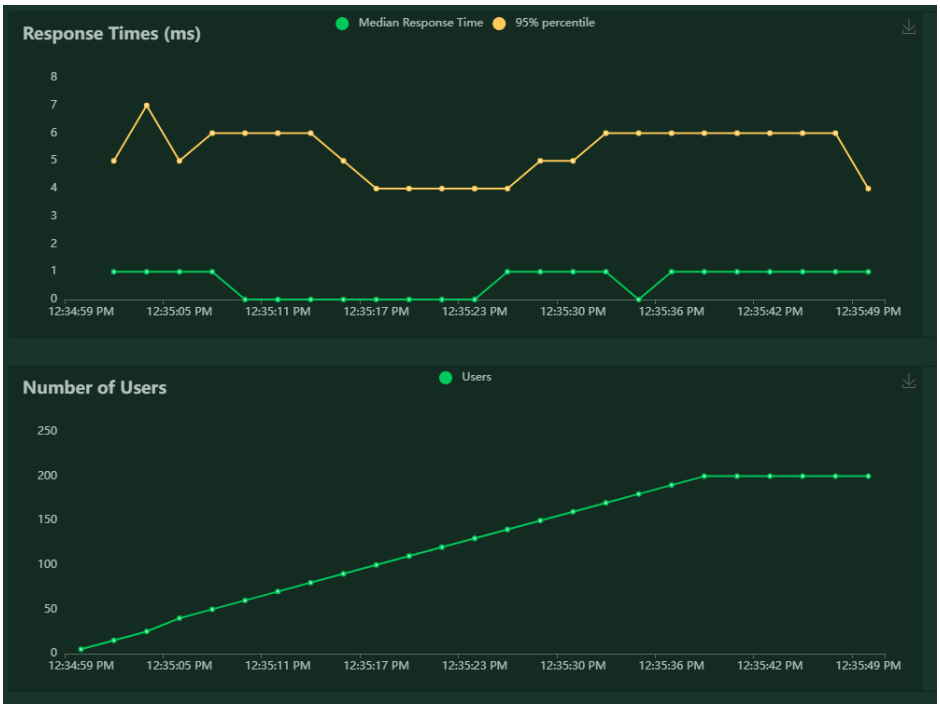
4.2. Ispitivanje rada sustava pod opterećenjem

Kako bi se testirao scenariji s velikim brojem klijenata iskorišten je alat Locust. Locust se temelji na programskom jeziku Python i pruža dodatnu kontrolu testiranja bez GUI sučelja.



Slika 4.1. Locust alat.

Ono što je najvažnije, uz Locust se može stvoriti roj ili velika grupa klijenata (tu je Locust dobio svoje ime). Roj se koristi za testiranje velikog broja korisnika ili uređaja koji imaju različito ponašanje. Na slikama 4.1. i 4.2. prikazano je sučelje za nadzor rezultata tijekom testiranja u postavljenim scenarijima.



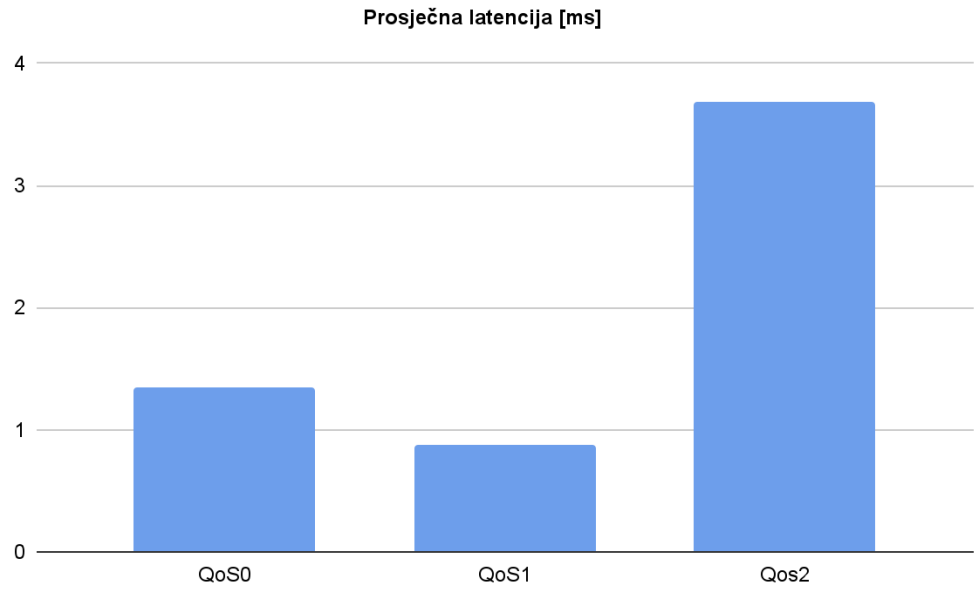
Slika 4.2. Locust grafovi ovisnosti broja korisnika i vremena odziva.

Ovakvim pristupom simulirano je samo opterećenje sustava, no ne i svih dijelova poput baze podataka ili LDAP-a. Simulacijom su prikupljeni okvirni podaci o najvećem protoku poruka, prosječnoj i predviđenoj upotrebi procesora te latenciji za EMQX broker. Prikazani su na tablici 4.3 i uspoređeni s ostalim brokerima prema [9]. Daljnjim povećanjem broja poruka i mjenjanjem kvalitete usluge došlo je do promjene vremena latencije. Računalo korišteno kao poslužitelj ima procesor i7 6700 3.40 GHz, 16GB RAM memorije. MQTT protokol nije pretjerano zahtjevan za procesor što je i jedna od prednosti, no kada bi računalo koje je služilo kao poslužitelj imalo bolje specifikacije i više resursa pretpostavka je da bi rezultati jedino mogli biti bolji.

	QoS0	QoS1	QoS2
Najveći protok poruka (poruka/s)	18034,00	4633,41	2627,31
Prosječna upotreba procesora sustava (%) pri višoj stopi poruka	98,71	96,82	95,54
Predviđena stopa obrade poruka pri 100% korištenju procesora sustava	18268,68	4785,59	2749,96
Prosječna latencija (ms)	1,34	0,87	3,68

Tablica 4.3. Predviđene stope obrade poruka pri 100%-tnoj upotrebi procesora.

Iz tablice 4.3 generiran je stupčasti graf na slici 4.4. koji prikazuje prosječnu latenciju iz podataka tablice. Kako je prikazano najmanja latencija je za poruke razine kvalitete usluge QoS1 (engl. *Quality of Service* - QoS.) Kako je ranije i spomenuto za većinu tema u sustavu korištena je razina kvalitete usluge QoS1 što je prednost kada govorimo o performansama sustava uspoređeno s ostalim razinama kvalitete usluge iz razloga što je najveći protok poruka dovoljan.



Slika 4.4. Prosječna latencija ovisno o razini kvalitete usluge.

5. ZAKLJUČAK

Ciljevi u zadatku rada postavljeni su planski kako bi se unaprijedilo sadašnje stanje udaljenog testiranja u automobilskoj industriji. Izrađen je sustav koji služi navedenom. Sustav je koncipiran tako da poboljša određene segmente u procesu udaljenog testiranja vezanog uz specifične testne stanice u automobilskoj industriji, a samim time i cjelokupni proizvodni proces. Uz pomoć odabranih tehnologija omogućen je nadzor i upravljanje udaljenim testnim stanicama. Prvi dio zadatka bio je realizirati nadziranje parametara testne stanice. To je napravljeno izradom algoritma koji se pokreće automatski na testnim stanicama koristeći Python programski jezik. Drugi dio zadatka bio je osmisliti i dizajnirati komunikacijske putove te odabrati protokol za komunikaciju. Za tu potrebu iskorišten je MQTT protokol koji je odgovarao primjeni. Idući zadatak obuhvaćao je analizu i prikaz podataka kroz povijest za što je bilo nužno implementirati bazu podataka u sustav. Sljedeće, mrežu stanica i klijenata te njihove podatke potrebno je prikazati korisnicima koji će obavljati nadzor. Kako bi se to realiziralo izrađeno je web sučelje. Moguće ga je dodatno unaprijediti izradom detaljnijih prikaza mreže testnih stanice i lakšem pristupu pojedinoj. U koracima simuliranja rada sustav je radio zadovoljavajuće te su ispunjeni zadani ciljevi. Omogućena je jednostavna daljnja nadogradnja funkcionalnosti koja će rezultirati još boljim radom i iskorištenosti sustava.

LITERATURA

[1] T. H. Team, „HiveMQ Case Study: BMW Mobility Services Car Sharing“., dostupno na: https://www.hivemq.com/case-studies/bmw-mobility-services/ [pristupljeno 14. rujan 2022.].

[2] E. T. C. Ltd, „EMQ helps SAIC Volkswagen building IoV platform“, dostupno na: https://www.emqx.com/en/blog/emqx-in-volkswagen-iov [pristupljeno 14. rujan 2022.].

[3] „What is MQTT and How Does it Work?“, *IoT Agenda*., dostupno na: https://www.techtarget.com/iotagenda/definition/MQTT-MQ-Telemetry-Transport [pristupljeno 14. rujan 2022.].

[4] „NinjaOne | Remote Monitoring & Management Software“, dostupno na: https://www.ninjaone.com/ [pristupljeno 14. rujan 2022.].

[5] „MSP + IT Management Software: RMM, Backup, Security“, *N-able*., dostupno na: https://www.n-able.com/ [pristupljeno 14. rujan 2022.].

[6] „RMM software for IT heroes | Atera“, *Atera - RMM software | PSA & Remote Access for MSPs*., dostupno na: https://www.atera.com/ [pristupljeno 14. rujan 2022.].

[7] „VSA“, *Kaseya*., dostupno na: https://www.kaseya.com/products/vsa/ [pristupljeno 14. rujan 2022.].

[8] „Introduction“, *EMQX 5.0 Documentation*., dostupno na: https://www.emqx.io/docs/en/v5.0/ [pristupljeno 14. rujan 2022.]

[9] „Stress-Testing MQTT Brokers: A Comparative Analysis of Performance Measurements“., dostupno na: https://mdpi-res.com/d_attachment/energies/energies-14-05817/article_deploy/energies-14-05817-v2.pdf [pristupljeno 14. rujan 2022.].

[10] E. T. C. Ltd, „MQTT X v1.7.2 Release Note“, *www.emqx.com*.,dostupno na: https://www.emqx.com/en/blog/mqttx-v-1-7-2-release-notes [pristupljeno 14. rujan 2022.].

SAŽETAK

U ovom radu dizajniran je, opisan i izrađen sustav za nadzor i upravljanje udaljenim testnim stanicama automobilske industrije. Omogućeno je praćenje parametara stanja testnih stanica te njihovo spremanje u bazu podataka. Uz navedeno pruženo je i korisničko sučelje za prikaz podataka i izvršavanje dostupnih funkcija upravljanja testnim stanicama. Korišten je Python programski jezik, Django programski okvir, EMQX MQTT broker, MySQL baza podataka, LDAP i neke druge tehnologije te pomoćni alati.

Ključne riječi: automobilska industrija, mqtt, nadzor, udaljeni pristup

ABSTRACT

Remote Software Testing in the Automotive Industry

In this paper, a system for monitoring and managing remote test stations of the automotive industry was designed, described and built. It is possible to monitor the state parameters of the test stations and save them in the database. In addition, a user interface is also provided for displaying data and executing the available remote test station tasks. Python programming language, Django programming framework, EMQX MQTT broker, MySQL database, LDAP and some other technologies and auxiliary tools were used.

Keywords: automotive, mqtt, monitoring, remote access

ŽIVOTOPIS

Autor ovog rada Marko Puntarić rođen je 17.02.1998. u Požegi. Nakon završene osnovne škole upisuje Matematičku gimnaziju u Požegi. Po završetku srednje škole upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer Računarstvo. Tijekom obrazovanja redovito sudjeluje na informatičkim natjecanjima i izrađuje vlastite projekte. Nakon uspješnog završetka preddiplomskog studija nastavlja školovanje i upisuje diplomski studij računarstva, smjer Robotika i umjetna inteligencija na istom fakultetu.

Potpis: _____