|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU  **FAKULTET TEHNIČKIH NAUKA U NOVOM SADU** |  |
| Predrag Kaljević E5-37/2019 | | |
| **Asset Management** | | |
|  |  |  |
| Dokumentacija projektnog zadatka | | |
| - Master akademske studije - | | |
|  | | |
| Novi Sad, 2020. | | |

Sadržaj

[1 Uvod 1](#_Toc41298809)

[2 Opis rešavanog problema 2](#_Toc41298810)

[3 Arhitektura 3](#_Toc41298811)

[3.1 Simulator 3](#_Toc41298812)

[3.2 Scada 3](#_Toc41298813)

[3.3 Calculation Engine (CE) 4](#_Toc41298814)

[3.4 Network Model Service (NMS) 4](#_Toc41298815)

[3.5 UI Adapter 4](#_Toc41298816)

[3.6 Publish-Subscribe mehanizam (PubSub) 4](#_Toc41298817)

[3.7 Transaction Service 4](#_Toc41298818)

[3.8 Korisnički interfejs (UI) 5](#_Toc41298819)

[4 Standardi i modeliranje EES (Višekorisnički klijentski rad) 6](#_Toc41298820)

[*4.1* *Integrity Update* prilikom pokretanja novog *UI* klijenta 6](#_Toc41298821)

[4.2 *Integrity* *Update* za već aktivne *UI* klijente 7](#_Toc41298822)

[5 Simulacija EES – *SCADA* (Ulazni i izlazni signali) 9](#_Toc41298823)

[*5.1* *Modbus* 9](#_Toc41298824)

[5.2 Signali 10](#_Toc41298825)

[5.2.1 Razlike između analognih i digitalnih signala 10](#_Toc41298826)

[5.2.2 Ulazni i izlazni signali na *Modbus*-u 11](#_Toc41298827)

[5.2.3 Signali u *Asset* *Management* aplikaciji 11](#_Toc41298828)

[6 Održavanje i kontrola kvaliteta EES – *Cloud* (*PUB/SUB* tok podataka) 13](#_Toc41298829)

[*6.1* *Queue* 13](#_Toc41298830)

[*6.2* *Publish/Subscribe* 13](#_Toc41298831)

[6.2.1 Kreiranje *topic*-a i *subscription*-a 14](#_Toc41298832)

[6.2.2 *Publish* poruke 16](#_Toc41298833)

[6.2.3 Čitanje poruka sa *subscription*-a 16](#_Toc41298834)

[7 Napredni računarski sistemi – Upravljanje i optimizacija (Reportovi alarma) 17](#_Toc41298835)

[7.1 Prikaz alarma na *UI* klijentu 17](#_Toc41298836)

[7.2 Reportovi 18](#_Toc41298837)

[8 Sigurnost i bezbednost – DMS (Ugrađena oprema u periodu i promena tema) 20](#_Toc41298838)

[8.1 Ugrađena oprema 20](#_Toc41298839)

[8.2 Promena tema 22](#_Toc41298840)

# Uvod

Cilj svake uspešne kompanije jeste da smanjenjem troškova poveća svoj profit. To se može uraditi na više načina, ali jedan od najefikasnijih i najbržih jeste korišćenjem softverskog rešenja. Upravo razvoj jedne takve aplikacije, čijim korišćenjem se efikasno mogu smanjiti kapitalni troškovi i jeste cilj našeg projekta.

Aplikacija bi trebalo da obuhvati tok kretanja opreme (prekidači i transformatori) kroz elektoenergetski sistem, a samim tim aplikacija se može koristiti u više timova u okviru jedna firme. Osim toga aplikacija treba da podrži komandovanje nad opremom, praćenje i čuvanje tih akcija, kao i generisanje raznih izveštaja. Čuvanje tih akcija nam omogućava jednu od najbitnijih funkcionalosti aplikacije, a to je predikcija datuma kad je potrebno poslati opremu na popravku. Na ovaj način se unapred može odrediti i isplanirati remont, što značajno smanjuje mogućnost dolaska do havarije i gubitaka napajanja kod velikog dela korisnika.

# Opis rešavanog problema

Napraviti aplikaciju za podršku poslovima održavanja u elektrodistributivnoj mreži, namenjenu unosu podataka o tehničkim karakteristikama i listama održavanja elektroenergetske opreme, u bazu podataka, kao i za pregled i obradu podataka iz baze i za pravljenje raznovrsnih izveštaja.

# Arhitektura

Arhitektura projekta je podeljena u tri sloja (**slika 1**):



Slika 1: Arhitektura

* Prezentacioni sloj predstavlja korisnički interfejs kroz koji korisnik ima uvid u dešavanja u sistemu i vrši interakciju sa istim.
* Biznis logiku obuhvataju servisi: *PubSub*, *CE*, *Scada*, *NMS*, *UI Adapter*. Logika je sakrivena od korisnika i može da bude distribuirana jer je komunikacija između svih servisa preko mreže.
* Sloj podataka je *Azure SQL Server* kome servisi pristupaju koristeći razvijenu biblioteku za komunikaciju sa istorijskom bazom podataka.

**Slika 1** je slikovita reprezentacija arhitekture celokupnog sistema.

## Simulator

Simulator predstavlja *third-party* aplikaciju koja služi za simuliranje vrednosti elemenata u polju. Vrednosti mogu da se zadaju ručno kliktanjem po korisničkom interfejsu ili nekom aplikacijom koja komunicira sa njim i šalje komande. Za komunikaciju se koristi *TCP*, a format poruke mora da poštuje *MODBUS* protokol.

## Scada

*Scada* je servis koji vrši periodičnu akviziciju stanja vrednosti na simulatoru koristeći *MODBUS* protokol. Kada primi trenutno stanje, šalje se razlika trenutnog i prethodnog stanja na *Calculation Engine* servis na interpretaciju.

## Calculation Engine (CE)

Servis koji prima podatke sa *Scada*-e i interpretira njihovo značenje. Dužan je da proveri da li se neki od uređaja nalazi u alarmnom stanju i da zapisuje podatke u *Azure SQL* bazu podataka (na slici *HIST*). Za to vreme se pristigli podaci proslede na *UI Adapter*. Ako se utvrdi da je neki uređaj u alarmnom stanju, dodatne informacije se šalju na *UI Adapter* da bi se korisnički interfejs informisao o tom stanju. Na zahtev *UI Adapter*-a generiše izveštaje i šalje ih nazad.

## Network Model Service (NMS)

Servis koji čuva konektivnost mreže. Podatke o konektivnosti može da primi od *Importer*aplikacije i sa korisničkog interfejsa. Pristigle podatke validira i primenjuje ako su validni, u suprotnom informiše pošiljaoca o tome šta je bila greška u validaciji. Nakon primenjenog modela, ostali servisi mogu po potrebi da šalju zahteve i traže informacije o statičkom modelu.

## UI Adapter

Specifičnost ovog servisa je u tome da služi kao most između korisničkog interfejsa i ostatka sistema. Korisnički interfejs šalje jedinstvene zahteve i dužnost ovog servisa je da koordiniše zahteve na jedan ili više servisa da bi u potpunosti odgovorio na zahtev korisničkog interfejsa. Tako korisnički interfejs i ostatak sistema nemaju deljene modele podataka.

## Publish-Subscribe mehanizam (PubSub)

Da bi moglo da se podrži više od jednog korisničkog interfejsa i da to bude nezavisno od celog sistema koristi se *publish-subscribe* mehanizam koji će pristigle podatke da demultipleksira na svaki od korisničkih interfejsa tako da svi budu u sinhronizaciji.

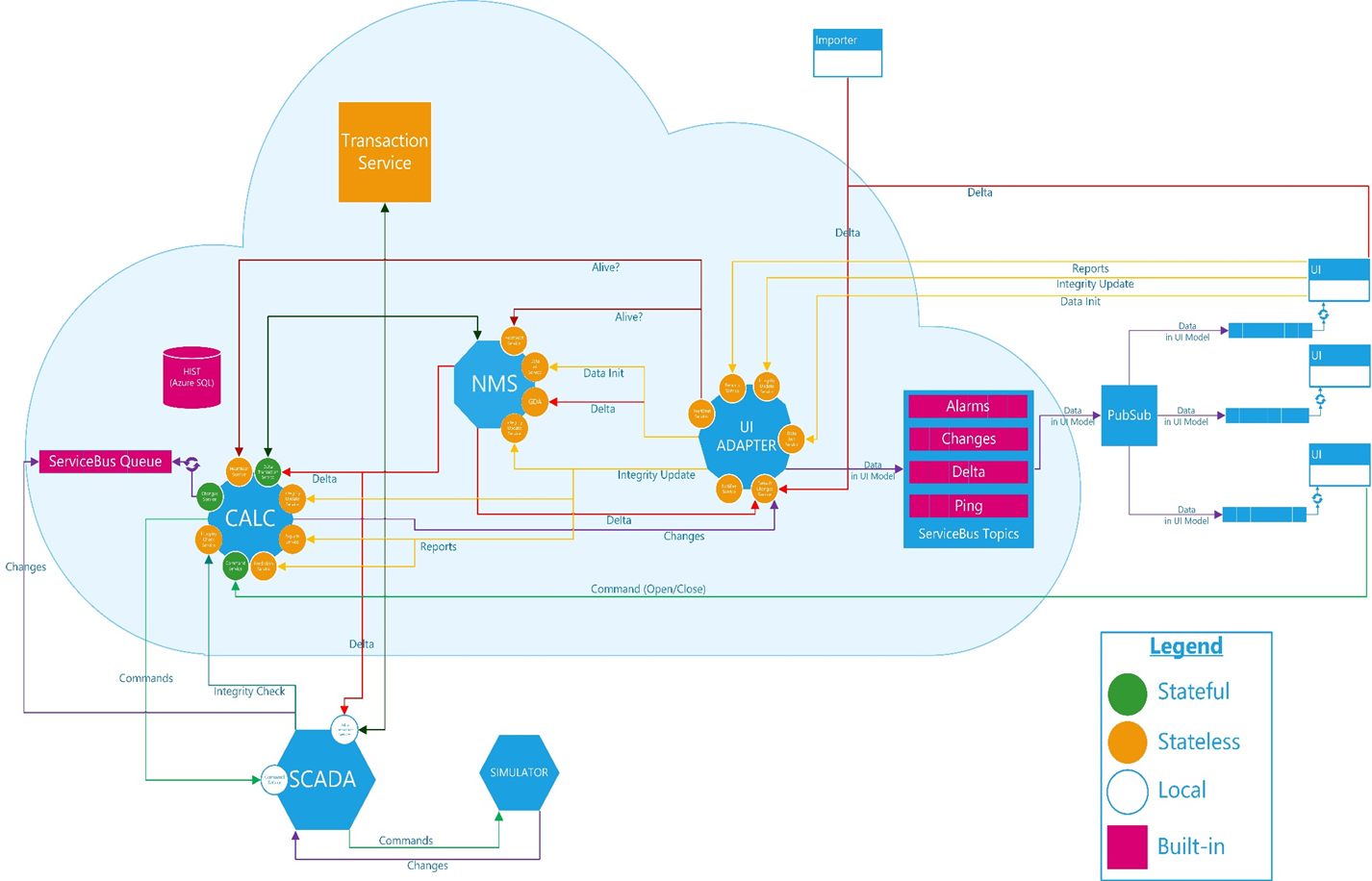
## Transaction Service

Funkcionalnost ovog servisa je koordinacija distribuirane transakcije kroz sistem. Kada podaci stignu na *NMS*i uspešno prođu validacije proslede se ostalim servisima. Nakon toga se servisi od interesa prijavljuju transakcionom servisu i kada se svi prijave *NMS*obavesti transakcioni servis da distribuirana transakcija može da se pokrene.

## Korisnički interfejs (UI)

Korisnik za interakciju sa sistemom ima na raspolaganju korisnički interfejs. Funkcionalnosti mu omogućavaju manipulaciju uređajima koji se nalaze u sistemu, njihovo prebacivanje u funkciju i van funkcije, prikaz mnoštva izveštaja, popravku.

Arhitektura celokupnog sistema može se videti na **slici 2**:



Slika 2: Arhitektura

# Standardi i modeliranje EES (Višekorisnički klijentski rad)

Aplikacija podržava sinhronizovani rad više korisnika, promena na jednom *UI* klijentu se šalje na sve ostale aktivne *UI* klijente. Prilikom pokretanja novog klijenta, uz pomoć *integrity update*, na *UI* se dovode najsvežiji podaci.

Implementirana je distribuirana transakcija sa *2 Phase Commit* protokolom. Kako ime govori, protokol se sastoji iz dve faze. Prva je *commit-request* faza u kojoj *TransactionCoordinator* upravlja svim resursima transakcije da izvrše *commit* ili *rollback*. Druga faza je *commit-phase* u kojoj donosi odluku da završi operaciju tako što izvršava *commit* (upis u bazu) ili *rollback* (odbacivanje svih izmena koje se nalaze u transakciji i vraćanje na prethodno stanje).

Distribuirana transakcija podrazumeva da svaki resurs uključen u nju izvršava *enlist*. Nakon toga, prva faza je realizovana tako što resursi pozivaju *Prepare()* metodu. Ova metoda zahteva od svakog resursa da prosledi da li je uspešno izvrsio *prepare* ili ne. *TransactionCoordinator* odlučuje da izvrši *commit* samo ako su svi resursi u transakciji uspešno izvršili *prepare*. U suprotnom, izvršava se *rollback*. Na kraju, resursi su obavešteni da je transakcija izvršena (uspešno ili neuspešno).

Distribuirana transakcija, kao i njena konkretna implementacija u ovoj aplikaciji, biće detaljnije objašnjena u drugoj dokumentaciji.

## *Integrity Update* prilikom pokretanja novog *UI* klijenta

*Integrity Update* je odgovoran za sinhronizaciju podataka prilikom pokretanja novog *UI* klijenta. *UI* klijent sadrži klasu *IntegrityDataGetter* koja vrši komunikaciju sa *UIAdapterService* vezanu za dobavljanje najsvežijih podataka.

*IntegrityDataGetter* sadrži sledeće metode:

* *GetRootEquipmentData(ContainerType type)* – dobavlja osnovne podatke o svim *container*-ima određenog tipa (*maintenance*, *substation* ili *warehouse*) unutar aplikacije na osnovu prosleđenog tipa.
* *GetAllEquipmentData(long ContainerId, bool GetLiveData = false)* – na osnovu prosleđenog *gid*-a dobavlja detaljne podatke o *container*-u, kao i osnovne podatke sve opreme koja se nalazi unutar njega.
* *GetBreakerData(long breakerGid)* – dobavlja detaljne podatke o konkretnom *breaker*-u na osnovu prosleđenog *gid*-a.
* *GetPTData(long ptGid)* - dobavlja detaljne podatke o konkretnom *transformer*-u na osnovu prosleđenog *gid*-a.
* *GetAlarms()* – *UI* klijentu vraća sve alarme trenutno aktuelne u sistemu.

*UIAdapterService* dalje prosleđuje zahteve ostalim servisima kako bi dobavio statičke i dinamičke podatke neophodne za *integrity* *update*. *NMS* (*Network* *Model* *Service*) sadrži statičke, a CE (*Calculation* *Engine)* dinamičke podatke. Klasa *IntegrityUpdate* je zadužena za to.

*NMS*-u se obraća za podatke u sledećim metodama:

* *GetRootEquipmentContainers(ContainerType containerType)* – svi *container*-i koji pripadaju određenom tipu (*maintenance*, *substation* ili *warehouse*).
* *GetEquipmentsForContainer(long gid)* – statički podaci *container*-a čiji je *gid* prosleđen, kao i *gid*-ovi sve opreme koja se u njemu nalazi.
* *GetAllMeasurementsForBreaker(long gid)* – statički podaci vezani za *breaker* čiji je *gid* prosleđen.
* *GetAllContainersOfType(ContainerType containerType)* – vraća samo nazive svih container-a koji pripadaju određenom tipu.

*CE*-u u sledećim:

* *GetBreakerData(long gid)* – dobavlja dinamičke podatke (*discrete*-ovi, *service time*, *work time*...) vezane za *breaker* čiji je gid prosleđen.
* *GetPTData(long gid)* – dinamički podaci za *power* *transformer*.
* *GetAlarms()* – zahteva od *calculation* *engine*-a sve trenutno aktuelne alarme.
* *GetEquipmentContainerChanges(DMSType type)* – dinamički podaci vezani za trenutni *container*. Što se ovoga tiče, jedini dinamički podaci su oni vezani za istoriju popunjenosti *container*-a i iscrtavanje grafika te istorije.

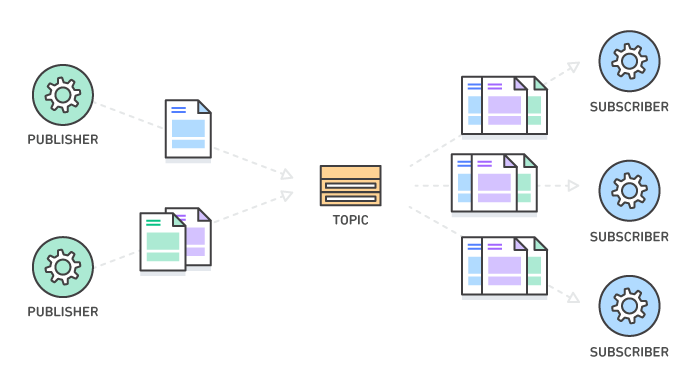
## *Integrity* *Update* za već aktivne *UI* klijente

Kada se sa jednog klijenta pošalje neka izmena modela, u distribuiranoj transakciji se vodi računa da su *NMS*, *CE* i *SCADA* uspešno odradili *prepare*. Ukoliko bar jedan od njih nije, poziva se *rollback* (odbacuju se sve izmene) i ne dolazi do *integrity* *update*. Sa druge strane, ako je *prepare* uspešan kod svih i svi uspešno izvrše *commit*, šalju se ažurirani podaci na sve postojeće klijente preko *Pub/Sub*-a.

*Pub/Sub* (*Publish/Subscribe*), prikazan na ***slici 1***, je servis za asinhrono slanje poruka. Svaka poruka koju *publisher* pošalje na određeni *topic* je istovremeno primljena od strane svih *subscriber*-a na tom *topic*-u. *Topic* je imenovani resurs na koji *publisher*-i šalju poruke. *Subscription* predstavlja tok poruka sa jednog određenog *topic*-a koje treba da budu dostavljene *subscribe*-ovanom klijentu. Poruka predstavlja podatke koje *publisher* šalje na *topic* da budu dostavljene *subscriber*-ima. Komunikacija može biti *one-to-one* (jedan *publisher* i jedan *subscriber* na topic-u), one-to-many(jedan subscriber, više publisher-a ili obrnuto) ili *many-to-many*(više *subscriber*-a i više *publisher*-a).

Vezano za *integrity* *update*, koriste se četiri *topic*-a:

* *Delta* – *Topic* za slanje svih izmena samog modela tj. statičkih podataka.
* *Changes* – na ovom *topic*-u stižu dinamičke izmene (promena stanja *breaker*-a, *service* *time*, *work* *time*...).
* *Alarms* – nakon pojave novog alarma u sistemu, šalje se na ovaj *topic*.
* *Repair* – kada se izvrši popravka neke opreme, svim aktivnim *UI* klijentima se na ovaj *topic* šalje informacija o promeni stanja te opreme.



Slika 3: *Publish/Subscribe*

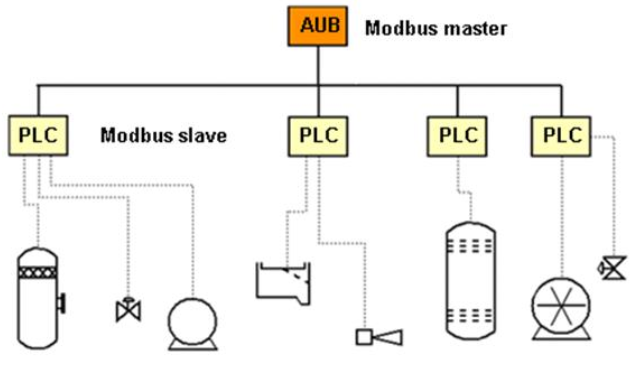
Prilikom pokretanja svakog *UI* klijenta, instancira se *subscriber* na svim *topic*-ima koji su gore navedeni. Nakon toga se poziva metoda *StartReceivingMessages()* kako bi svaki *subscriber* počeo da prima sve poruke koje dolaze na njegov odgovarajući *topic*. Definisani su *ReceivedAction* *task*-ovi za obradu podataka koji stižu sa svakog *topic*-a.

# Simulacija EES – *SCADA* (Ulazni i izlazni signali)

## *Modbus*

*Modbus* industrijski komunikacioni protokol je bio razvijen 1979. godine od strane *Modicon*-a (danas *Schneider* *Electric*). Od tada je postao industrijski standard za prenos diskretnih i analognih *I/O* informacija i podataka registara između akviziciono upravljačkog bloga SCADA stanice (eng. FEP – Front End Processor) i procesnih kontrolera (eng. *RTU – Remote Terminal Unit*, *PLC – Programmable Logic Controller*). Popularnost je stekao usled jasnog modela podataka koji garantuje laku programsku implementaciju i visoku fleksibilnost u primeni.

U *Modbus* sistemu(**slika 2**) funkcioniše *master-slave* (*client-server*) komunikacija u kojoj samo jedan uređaj (*master/client*) može da inicira transakcije (upite). Drugi uređaji (*slaves/servers*) odgovaraju tako što šalju zahtevane podatke *master*-u, ili izvršavanjem akcije zahtevane u upitu. *Slave* je bilo koji periferni uređaj (*I/O* pretvarač, ventil, ili drugi uređaji za merenje) koji obrađuje informacije i šalje rezultat *master*-u preko *Modbus*-a.



Slika 4: *Modbus* sistem sa jednim *master* uređajem i više *slave* uređaja

*Master* može da se obrati individualnim *slave* uređajima, ili može da pošalje *broadcast* poruku koju prima svaki *slave*. *Slave* uređaji šalju odgovor na sve upite koji su upućeni njima individualno, ali ne odgovaraju na *broadcast* upite. *Slave* uređaji ne iniciraju komunikaciju prvi, već samo odgovaraju na upite od strane *master*-a.

*Master*-ov upit se sastoji iz:

* adrese *slave*-a,
* koda funkcije koji definiše zahtevanu akciju,
* propratnih podataka i
* *error* *check* polja.

*Slave*-ov odgovor se sastoji iz:

* polja za potvrdu izvršene akcije,
* podataka koje treba vratiti na *master*-ov zahtev i
* *error* *check* polja.

Ukoliko ne dođe do greške, odgovor *slave* uređaja sadrži zahtevane podatke. Ako dođe do greške u primljenom upitu, ili *slave* nije u stanju da izvrši zahtevanu akciju, vratiće *exception* poruku kao odgovor.

*Modbus TCP/IP* predstavlja *Modbus RTU* protokol sa *TCP* interfejsom koji je prilagođen za rad na lokalnoj mreži. *TCP/IP* omogućava razmenu blokova binarnih podataka između uređaja. Primarna funkcija *TCP* je da obezbedi da svi paketi podataka budu ispravno primljeni, dok *IP* vodi računa da su poruke pravilno adresirane i preusmerene.

## Signali

Signal predstavlja elektromagnetnu ili električnu struju koja se koristi za prenos podataka od jednog sistema ili mreže do drugog. Signal je funkcija koja prenosi informaciju o nekoj pojavi.

Analogni signal je neprekidni signal u kojem jedna vremenski-različita količina predstavlja drugu vremenski-baziranu promenljivu. Digitalni signal se koristi za predstavljanje podataka kao sekvence odvojenih vrednosti u trenutku u vremenu. Može da ima samo jednu od fiksnog broja vrednosti.

### Razlike između analognih i digitalnih signala

|  |  |
| --- | --- |
| **Analogni** | **Digitalni** |
| Analogni signal je neprekidni signal koji predstavlja fizička merenja. | Digitalni signali su vremenski-razdvojeni signali generisani uz pomoć digitalne modulacije. |
| Označen je sinusoidom | Označen je kvadratnim talasom |
| Koristi neprekidni raspon vrednosti za predstavljanje informacija. | Koristi diskretne 0 i 1 da predstavi informacije. |
| Merenja senzora temperature, svetlosnih senzora, radio signali | Otvoreno/zatvoreno, uključeno/isključeno |
| Analogni signali gube kvalitet usled šuma kroz prenos, kao i tokom *read/write* ciklusa. | Digitalni signali su relativno imuni na šum i gubljenje kvaliteta tokom prenosa i *read/write*. |
| Nizak protok | Visok protok |
| Analogna oprema nema fleksibilnu implementaciju | Digitalna oprema ima fleksibilnu implementaciju |
| Obrada može biti u realnom vremenu i troši manji protok od digitalnog signala. | Nikada ne garantuje da obrada digitalnog signala može biti u realnom vremenu. |
| Analogni signali nemaju ograničeni raspon. | Digitalni signal ima ograničeni broj, npr. 0 i 1. |

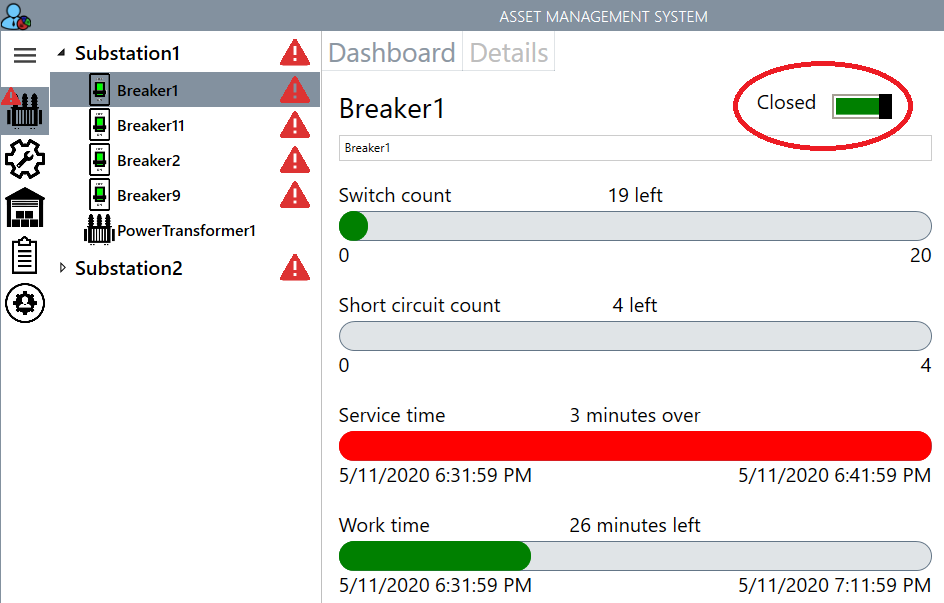
### Ulazni i izlazni signali na *Modbus*-u

*Modbus* registarska mapa predstavlja četiri grupe podataka u *RTU/PLC* adresnom prostoru. Reprezentuju najvažnije podatke, procesne ulaze i izlaze. Analogni ulazi i izlazi su dužine 16 bita, a digitalni 1 bit. Dve osnovne operacije su čitanje (*read*) za sve tipove i upis (*write*) za izlaze.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Adresa | | Oznaka | Dužina | Pristup | Opis |
| 00001 | 10000 | *Discrete Outputs (Coils)* | 1 bit | *Read/Write* | Digitalni izlazi |
| 10001 | 20000 | *Discrete Inputs* | 1 bit | *Read* | Digitalni ulazi |
| 30001 | 40000 | *Input Registers* | 16 bit | *Read* | Analogni ulazi i Brojači |
| 40001 | 50000 | *Holding Registers* | 16 bit | *Read/Write* | Analogni izlazi |

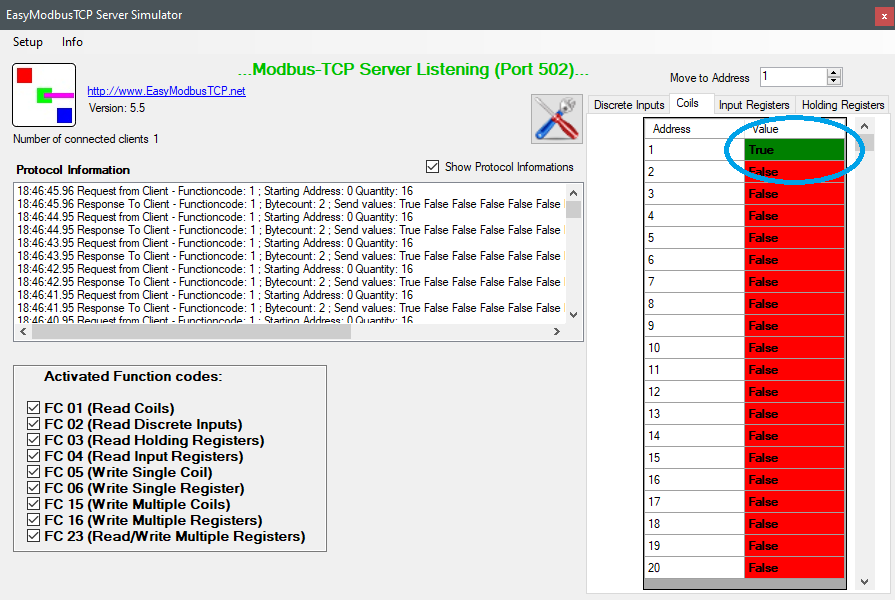
### Signali u *Asset* *Management* aplikaciji

Što se tiče signala u *Asset Management* aplikaciji, postoje isključivo digitalni. Osnovni ulazni digitalni signal predstavlja stanje prekidača. Na **slici 3** se može videti zatvaranje prekidača preko *UI klijenta*:



Slika 5: Zatvaranje prekidača preko *UI* klijenta

Izdavanjem komande zatvaranja preko *UI* klijenta, menja se stanje prekidača “*Breaker1*”u “*Closed*”. Stanje 1 (*true*) znači da je prekidač zatvoren, a 0 (*false*) otvoren. *SCADA* komanduje da se taj prekidač zatvori na simulatoru (prikazano na **slici 4**). Ovo predstavlja izlazni digitalni signal.



Slika 6: Prikaz zatvorenog prekidača na *EasyModbus*-u

Drugi digitalni signal predstavlja kratak spoj na prekidaču. Kada se desi kratak spoj, dolazi do promene stanja prekidača. Stanje 0 (*false*) označava da prekidač nije u kratkom spoju, a 1 (*true*) da jeste.

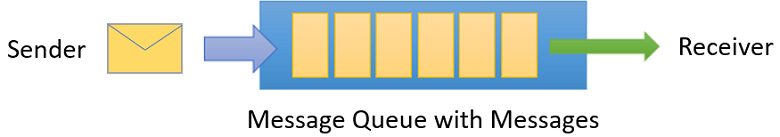
# Održavanje i kontrola kvaliteta EES – *Cloud* (*PUB/SUB* tok podataka)

*Azure Service Bus* predstavlja servis za razmenu poruka na *cloud*-u. Koristi se za povezivanje aplikacija, uređaja i servisa na *cloud*-u sa drugim aplikacijama i servisima. Omogućava pouzdanu i bezbednu platformu za asinhroni prenos podataka i stanja. Podaci se razmenjuju među različitim aplikacijama i servisima uz pomoć poruka. Poruka je u binarnom formatu i može sadržati *JSON*, *XML* ili običan tekst.

*Namespace* je *container* svih komponenti vezanih za razmenu poruka. Više *queue*-ova i *topic*-a mogu biti unutar jednog *namespace*-a, i oni često služe kao *container*-i aplikacija.

## *Queue*

*Queue*(**slika 5**) ima *FIFO* (*First* *In*, *First* *Out*) isporuku ka jednom ili više primalaca. Poruke su primljene i obrađene po redosledu dodavanja u *queue*, i samo jedan primalac može da primi i obradi jednu poruku.



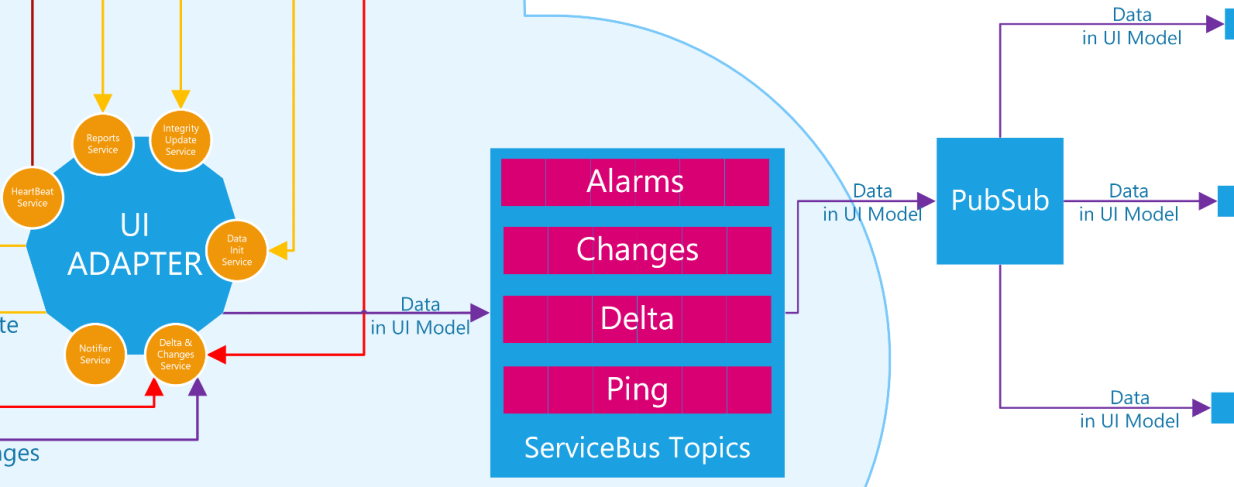
Slika 7: *Service Bus Queue*

Glavna prednost korišćenja *queue*-a je da se postigne “*temporal decoupling*” komponenata aplikacije. Drugim rečima, pošiljaoci i primaoci ne moraju da šalju i primaju poruke u isto vreme, zato što se poruke trajno čuvaju unutar *queue*-a. Takođe, pošiljaoc ne mora da čeka na odgovor od strane primaoca da bi nastavio da obrađuje i šalje poruke.

## *Publish/Subscribe*

Za razliku od *queue*-a, gde je svaka poruka obrađena od strane jednog primaoca, *topic*-i i *subscription*-i omogućavaju *one-to-many* oblik komunikacije, u vidu *publish/subscribe*. Poruke su poslate na *topic* na isti način kao kod *queue*-a, ali nisu primljene od strane *topic*-a direktno. Umesto toga, njih primaju *subscription*-i. *Subscription* izgleda kao *queue* koji prima kopije poruka poslatih na *topic*. Poruke se primaju sa *subscription*-a identično kako se primaju sa *queue*-a. Svaka poruka je dostupna svakom *subscription*-u koji je registrovan na njenom *topic*-u.

Na **slici 6** se može videti komunikacija sa *Service* *Bus*-om u *Asset Management*-u. Postoje četiri *topic*-a: *Alarms*, *Changes*, *Delta* i *Ping*. Mikroservis *DeltaAndChangesService* na *UIAdapter*-u je odgovoran za slanje izmena statičkih i dinamičkih podataka na svoje odgovarajuće *topic*-e na *Service* *Bus*-u.



Slika 8: *Service Bus Topic*-i u arhitekturi *Asset* *Management*-a

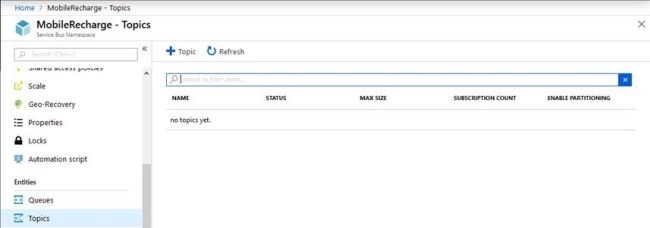
*K*ada se desi izmena statičkih (izmena modela) ili dinamičkih (*discrete*-ovi, *service* *time*, *work* *time*) podataka, *DeltaAndChangesService* šalje informacije na *topic*-e *Delta* (statički) i *Changes* (dinamički), koji se opet prosleđuju na *subscription* lokalnog *Pub/Sub*-a.

Slično, kada deo opreme pređe u stanje alarma, *Notifier* servis je odgovoran za slanje informacije na *topic* *Alarms*. Dalje se prosleđuje na *subscription* ka lokalnom *Pub/Sub*-u, koji prosleđuju informacije o alarmu svim *UI* klijentima sa kojima je povezan.

*HeartBeatService* na intervalima šalje na *Ping* *topic* informacije da li su podignuti *NetworkModelService* i *CalculationEngine* servisi, koji su neophodni za obavljanje *integrity* *update*-a.

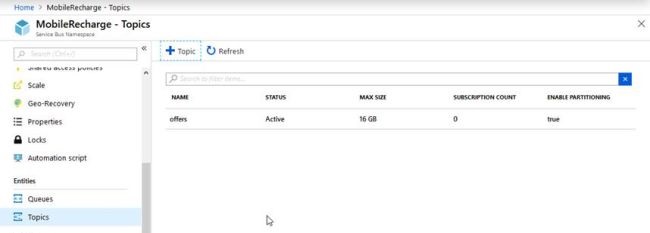
### Kreiranje *topic*-a i *subscription*-a

Nakon logovanja na *Azure* *Portal* (*portal.azure.com*), unutar *namespace*-a, klikom na dugme *“+ Topic*” dodaje se novi *topic* (**slika 7**).



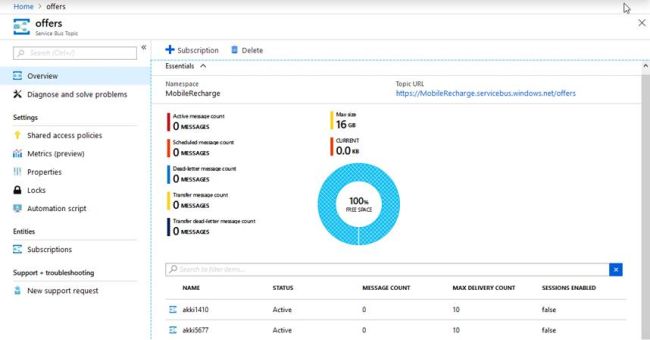
Slika 9: Dodavanje novog *topic*-a

Nakon popunjavanja naziva i ostalih polja, klikom na “*Create*” dodaje se novi *topic* unutar *namespace*-a, koji se može videti na **slici 8** ispod:



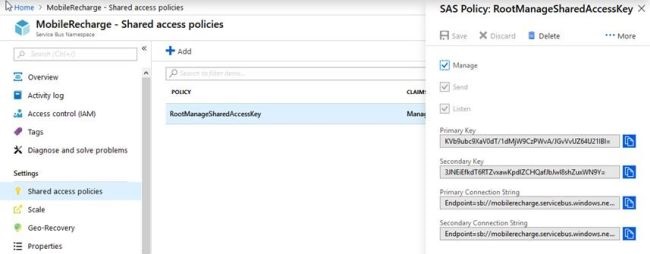
Slika 10: *Topic* dodat

Sledeći korak je dodavanje *subscription*-a. Otvaranjem prethodno kreiranog *topic*-a, možemo videti njegovu listu *subscription*-a (trenutno je prazna). Klikom na dugme “*+ Subscription*” otvara se dijalog za dodavanje novog. Nakon popunjavanja naziva i ostalih polja, na prikazu *topic*-a možemo videti da sada ima jedan *subscription* (**slika 9**).



Slika 11: *Subscription* dodat

Nakon toga, prelaskom na “*Shared access policies*” sekciju, unutar “*RootManageSharedAccessKey*” mogu se videti ključevi i *connection string*-ovi koji se koriste za konektovanje (**slika 10**).



Slika 12: *Shared access policies*

### *Publish* poruke

Postupak *publish*-ovanja poruke:

1. Unutar aplikacije napraviti 2 promenljive, jednu za *connection* *string* (kopiran iz “*Shared* *access* *policies*” sekcije), a drugu za naziv *topic*-a.
2. Dodati *Microsoft.Azure.ServiceBus* u *NuGet package manager*.
3. Kreirati *topic* klijenta i proslediti mu *connection* *string* i naziv *topic*-a.
4. Konvertovati *string* poruku u *Azure Service Bus* poruku.
5. Na *topic* klijentu pozvati metodu *SendAsync()* i proslediti joj poruku.
6. Na kraju, pozvati *CloseAsync()* kako bi se zatvorila otvorena konekcija.

### Čitanje poruka sa *subscription*-a

Postupak čitanja poruke:

1. Unutar aplikacije napraviti 2 promenljive, jednu za *connection* *string* (kopiran iz “*Shared* *access* *policies*” sekcije), a drugu za naziv *topic*-a.
2. Kreirati *topic* klijenta i proslediti mu *connection* *string* i naziv *topic*-a.
3. Na *topic* klijentu pozvati *RegisterMessageHandler()* koji služi za neprekidno primanje poruka. Registruje *message* *handler* i otvara novi *thread* za primanje poruka. Ovaj *handler* se čeka svaki put kada primalac primi novu poruku.
4. Unutar *message* *handler*-a pozvati *CompleteAsync()* koji završava poruku uz pomoć svog *lock* *token*-a i briše je iz *queue*-a.

# Napredni računarski sistemi – Upravljanje i optimizacija (Reportovi alarma)

Alarm označava događaj u sistemu usled približavanja određenog dela opreme svojoj granici upotrebljivosti po nekom svojstvu. Ta svojstva su sledeća:

* Isticanje ili približavanje isticanju propisanog roka od prethodnog servisiranja nekog dela opreme (*service* *time*).
* Isticanje ili približavanje isticanju ukupnog radnog veka dela opreme od dana nabavke (*work* *time*).
* Prelazak ili približavanje prelasku propisanog ograničenja broja manipulacija na prekidaču, pre nego što treba biti poslat na servisiranje (*switch* *count*).
* Prelazak ili približavanje prelasku propisanog ograničenja broja kratkih spojeva koji se može desiti na prekidaču pre nego što treba biti poslat na servisiranje (*short* *circuit* *count*).

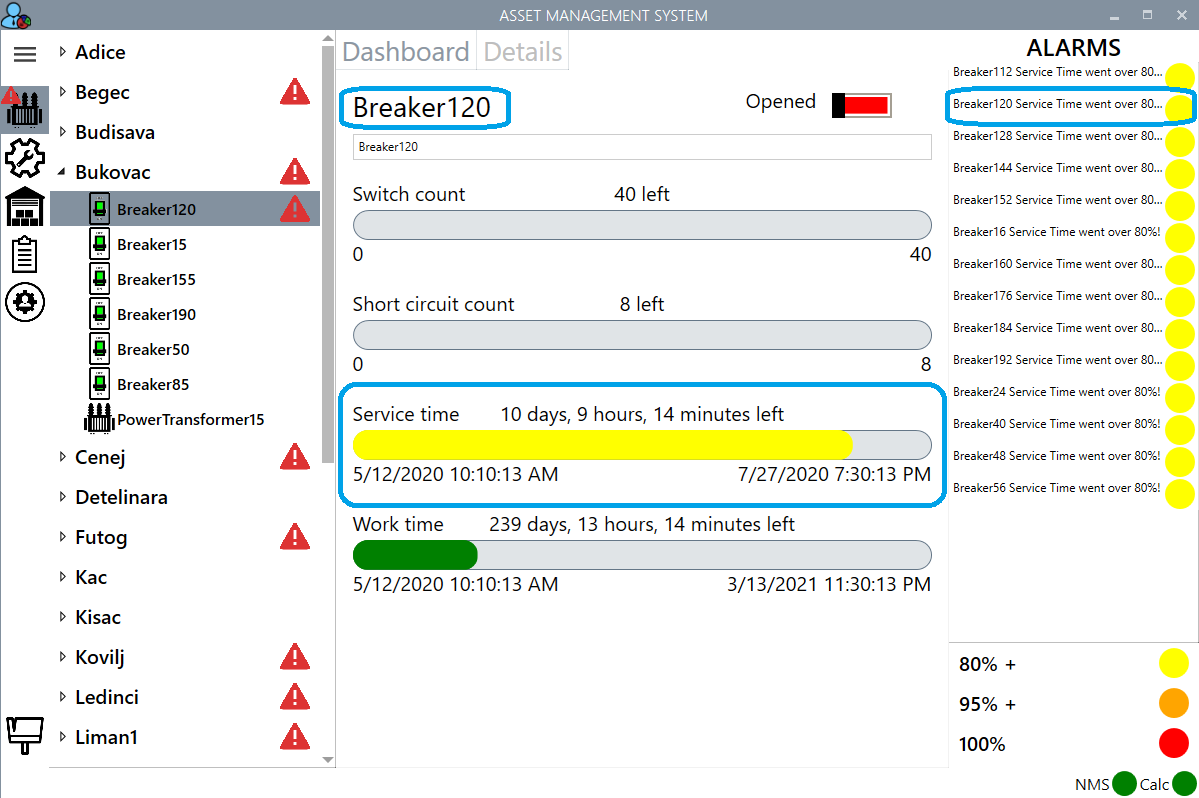
Svaki alarm može da ima jedno od četiri stanja: *Normal* (nije prikazano na *UI*), *High*, *HighHigh* i *Overused*.

*CalculationEngine* intervalno proverava promene vezane za *SCADA* *discrete* signale. Nakon što neka promena izazove pojavu alarma na delu opreme, informacije o alarmu upisuju se u *HIST bazu* zajedno sa *timestamp*-om trenutka njegove pojave. Zatim, šalju se *Notifier* servisu na *UIAdapter*-u, koji preko *Pub/Sub*-a prosleđuje alarm svim aktivnim *UI* klijentima.

U nastavku, biće prikazan izgled alarma u trenutku pojave na *UI* klijentu, kao i generisanje svih reportova vezanih za njih. Detaljnije objašnjenje vremenskih alarma i alarma vezanih za manipulaciju nad opremom biće navedeno u drugim dokumentacijama.

## Prikaz alarma na *UI* klijentu

Na **slici 11** vidi se prikaz alarma koji su trenutno aktivni u sistemu. *Dashboard* trenutno otvorenog prekidača(“*Breaker120*”) pokazuje da je prošlo preko 80% vremenskog roka od njegovog prethodnog servisiranja. Nakon što je *CalculationEngine* pokupio tu informaciju, alarm je prosleđen preko *Notifier*-a na *Pub/Sub*, i preko njega na sve aktivne *UI* klijente. Svaki klijent posle primljenog alarma, na desnoj strani, dodaje alarm sa odgovarajućom bojom i obaveštenjem da je prekidaču “*Breaker120*” isteklo 80% *service* *time*-a.

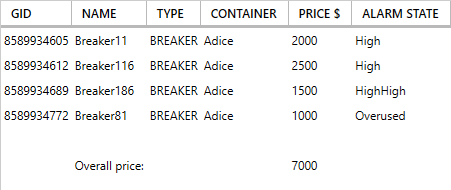


Slika 13: *High* *service* *time* alarm

## Reportovi

Report predstavlja dokument za prezentovanje informacija zahtevanih od strane korisnika u organizovanom formatu. Može biti generisan za više različitih tipova informacija, a u ovom poglavlju tema su reportovi vezani za alarme koji se javljaju u *Asset* *Management* aplikaciji.

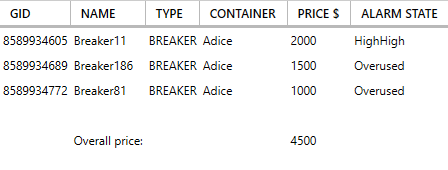
Prvi tip reporta je prikaz sve opreme sa alarmima u *High* stanju, uključujući i sva viša stanja. Može se generisati za opremu unutar jedne konkretne trafostanice ili za svu opremu trenutno u pogonu. **Slika 12** prikazuje takav report za trafostanicu “Adice”.



Slika 14: Alarmi u *High* *state*-u

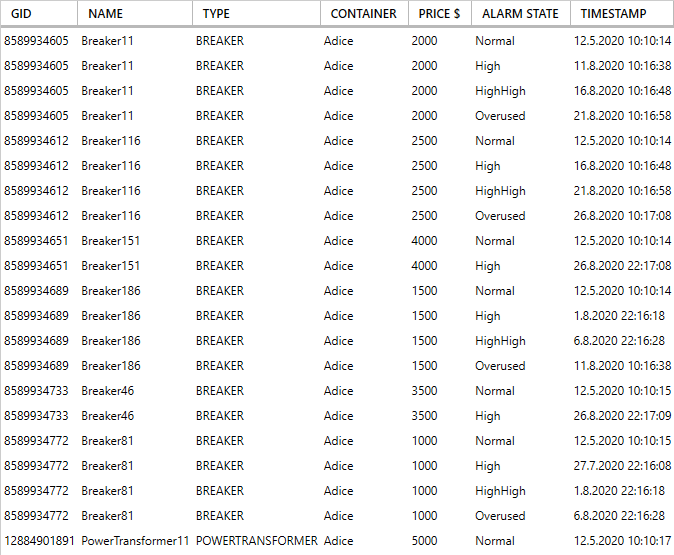
Na reportu se može videti *GID* opreme, njegov naziv, tip (prekidač ili transformator), trafostanica kojoj pripada, njegova nabavna cena, kao i u kom stanju alarma se nalazi. Dodatno, na dnu reporta je prikazana ukupna cena sve prikazane opreme, kao i na svim ostalim reportovima.

Drugi report (**slika 13**) funkcioniše na isti način, sa jedinom razlikom da prikazuje samo opremu sa alarmima u *HighHigh* (ili više) stanju.



Slika 15: Alarmi u *HighHigh* *state*-u

Treći tip predstavlja istoriju alarma. Takođe se može generisati za opremu unutar jedne trafostanice ili svu opremu u pogonu. Na **slici 14** prikazana je istorija promena stanja alarma za svu opremu unutar stanice “Adice”. Za svaki deo opreme, u vremenskom redosledu, prikazana su sva stanja alarma u kojima je bio od trenutka ugradnje, kao i *timestamp*-ovi kada je ušao u svako od stanja.



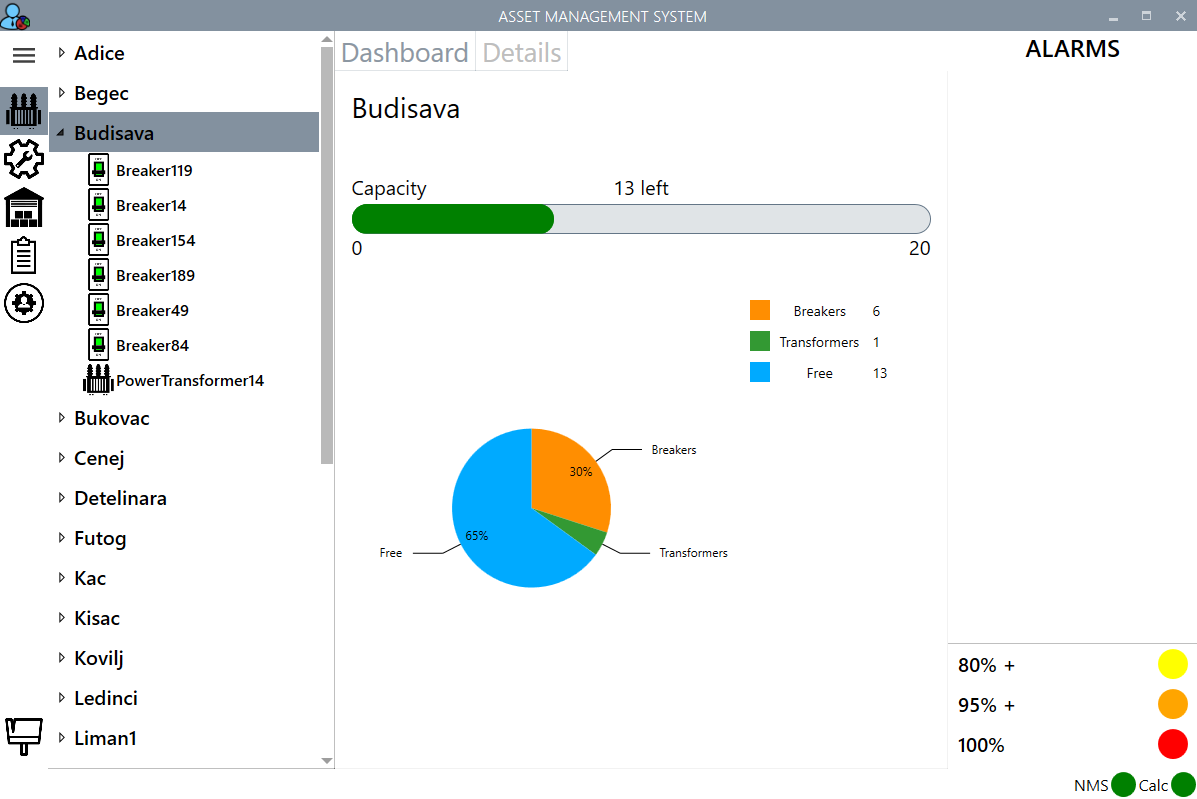
Slika 16: Istorija alarma

# Sigurnost i bezbednost – DMS (Ugrađena oprema u periodu i promena tema)

Kao što je već pomenuto u prethodnom poglavlju, report je dokument za prezentovanje informacija zahtevanih od strane korisnika u organizovanom formatu. U ovom poglavlju će biti analiziran report vezan za opremu ugrađenu u periodu, dok će ostali reportovi biti tema drugih dokumentacija. Dodatno, na kraju poglavlja, biće prikazana personalizacija *UI* teme.

## Ugrađena oprema

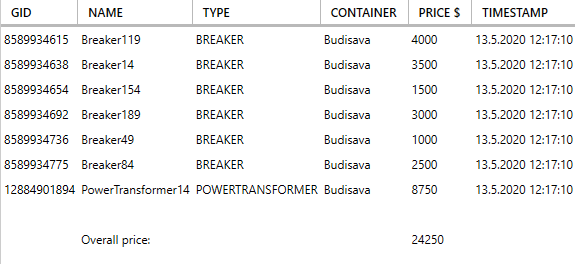
*Dashboard* trafostanice (**slika 15**) sadrži *piechart* popunjenosti njenog kapaciteta. Prikazuje broj prekidača i transformatora koji se nalaze u njoj, kao i preostali broj slobodnih mesta do popunjavanja punog kapaciteta. Sa druge strane, ovaj prikaz ne pokazuje korisniku istoriju svake konkretne opreme u pogonu unutar trafostanice. Takođe, grafik istorije popunjenosti na *tab*-u *Details* jedino prikazuje promenu popunjenosti stanice kroz vreme.



Slika 17: Prikaz popunjenosti kapaciteta trafostanice

Report ugrađene opreme (“*Installed* *Equipment*“) je namenjen da prikaže korisniku te informacije koje nedostaju.

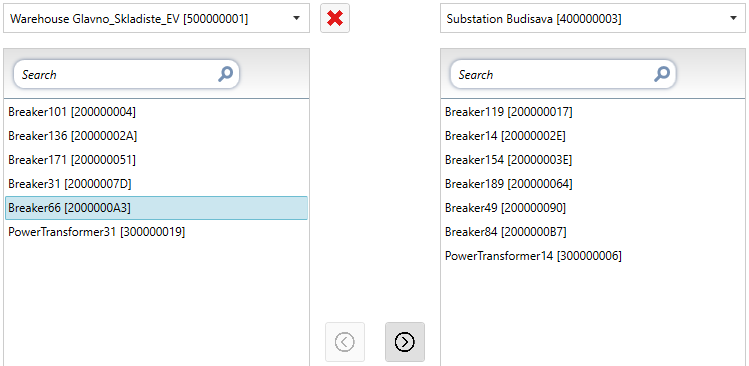
Report (**slika 16**) prikazuje jedinstveni *GID* svakog dela opreme u pogonu, njen naziv, tip (prekidač ili transformator), naziv trafostanice kojoj pripada, cenu i *timestamp* trenutka kada je ugrađena. Na dnu reporta nalazi se ukupna cena sve opreme unutar te stanice. Sva prikazana oprema ima isti *timestamp* što govori da je sve u pogonu od dana montaže trafostanice.



Slika 18: Report ugrađene opreme

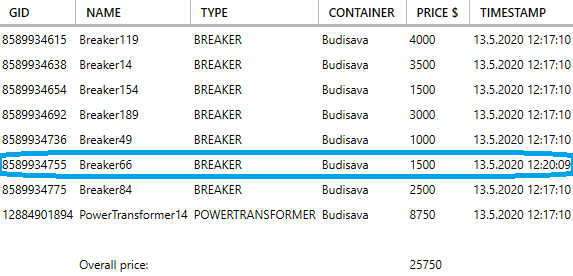
Postoji mogućnost i generisanja reporta sve opreme koja je u pogonu, umesto samo za jednu konkretnu trafostanicu. Medjutim, problem kod toga je veliki broj prikazanih stavki, stoga je implementirano filtriranje ugrađene opreme prema vremenu (mesec/kvartal/godina).

Instalacija opreme se obavlja preko *Asset* *Manager*-a, gde korisnik odabirom opreme iz skladišta može da je prebaci u odabranu trafostanicu. Na **slici 17** je prikazana postavka prekidača “Breaker66” u trafostanicu “Budisava”.



Slika 19: *Asset* *manager* - instalacija prekidača u trafostanicu

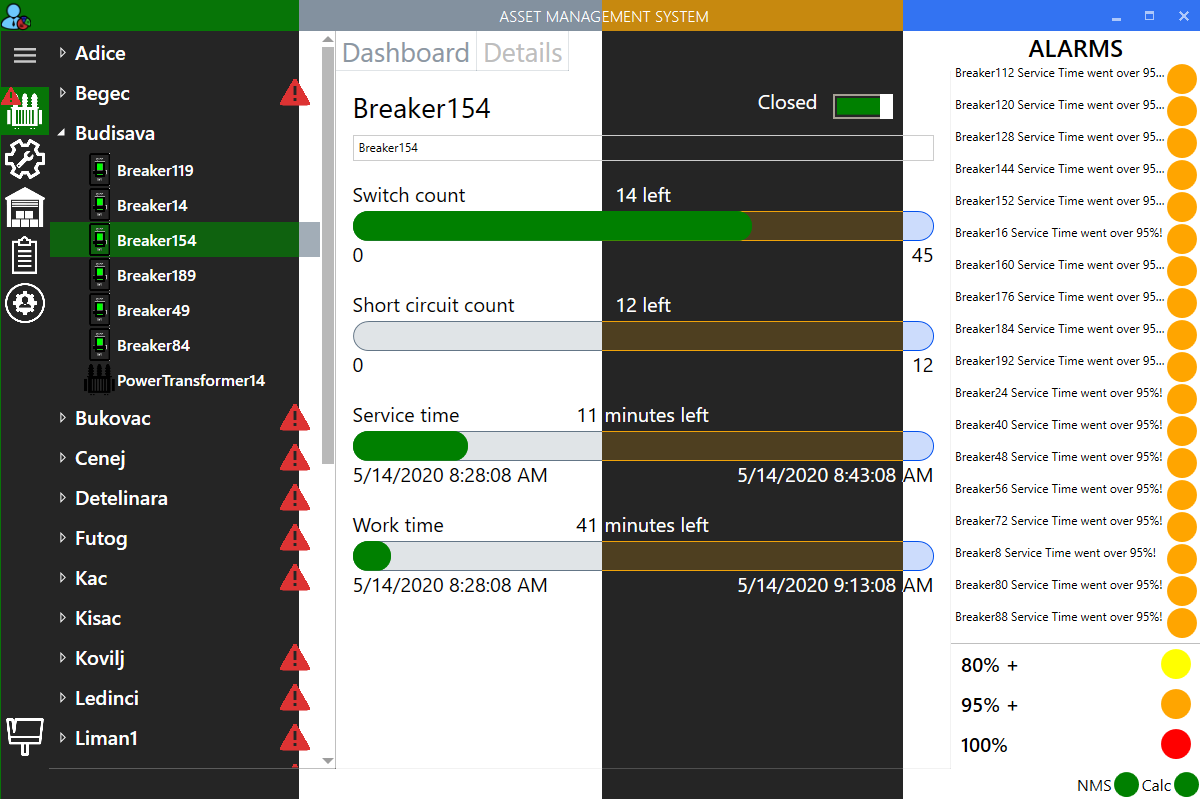
Nakon toga, novo generisanje reporta (**slika 18**) za odabranu trafostanicu pokazuje novi ugrađeni prekidač, čije se vreme instalacije razlikuje od ostalih.



Slika 20: Report posle instalacije

## Promena tema

*Asset* *Management* aplikacija podržava personalizaciju *UI* izgleda po želji. Umesto par individualnih tema, korisnik sam može da pravi kombinacije između 2 boje pozadine (crna i bela) i 6 boja za sve ostale *UI* elemente. Na **slici 19** se vidi kako izgleda UI aplikacije kroz četiri različite kombinacije tema.



Slika 21: Različite kombinacije *UI* tema