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

Stefan Stojaković E5-35-2019

**ASSET MANAGEMENT SYSTEM**

Dokumentacija projektnog zadatka- Master akademske studije -  
  
  
  
  
  
  
  
  
  
  
  
  
-Novi Sad, 2020

Sadržaj:

[1. Uvod 1](#_Toc41295316)

[2. Opis rešavanog problema 2](#_Toc41295317)

[3. Arhitektura 3](#_Toc41295318)

[3.1 Simulator 3](#_Toc41295319)

[3.2 Scada 3](#_Toc41295320)

[3.3 Calculation Engine (CE) 3](#_Toc41295321)

[3.4 Network Model Service (NMS) 4](#_Toc41295322)

[3.5 UI Adapter 4](#_Toc41295323)

[3.6 Publish-Subscribe mehanizam (PubSub) 4](#_Toc41295324)

[3.7 Transaction Service 4](#_Toc41295325)

[3.8 Korisnički interfejs (UI) 4](#_Toc41295326)

[4. Standardi i modeliranje elektroenergetskih sistema 5](#_Toc41295327)

[4.1 Postupak kreiranja CIM Profila 5](#_Toc41295328)

[5. Simulacija elektroenergetskih sistema sa kritičnom misijom 8](#_Toc41295329)

[5.1 Modbus protkol 8](#_Toc41295330)

[5.2 Modbus TCP 10](#_Toc41295331)

[6. Održavanje i kontrola kvaliteta elektroenergetskog softvera 12](#_Toc41295332)

[6.1 Microservisna arhitektura vs. Monolitička arhitektura 12](#_Toc41295333)

[6.2 Prednosti i mane Mikroservisne arhitekture 13](#_Toc41295334)

[6.3 Reliable Services 13](#_Toc41295335)

[7. Napredni računarski sistemi sa kritičnom misijom u elektroenergetici 16](#_Toc41295336)

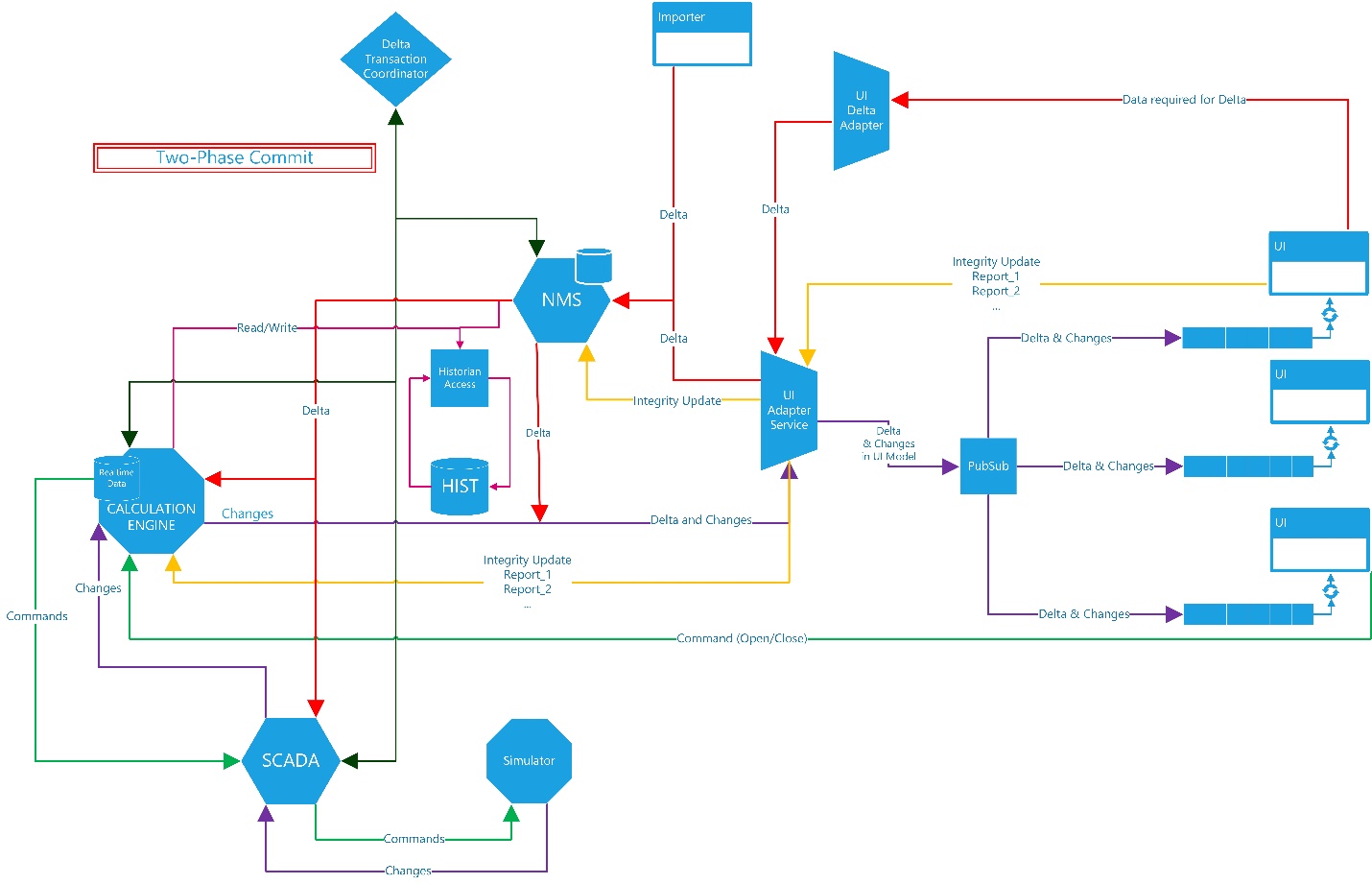
[7.1 Reportovi predikcije 16](#_Toc41295337)

[7.2 Remont prekidača 17](#_Toc41295338)

[8. Sigurnost i bezbednost u SmartGrid sistemima 19](#_Toc41295339)

[8.1 Prikaz elektroenergetske opreme koja je nabavljena u poslednjih mesec/kvartal/godina 19](#_Toc41295340)

# Uvod



Slika 1. Arhitektura projekta

# 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 2.)



Slika 2. 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.

## 3.1 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.

## 3.2 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.

## 3.3 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.

## 

## 3.4 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.

## 3.5 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.

## 3.6 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.

## 3.7 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.

## 3.8 Korisnički interfejs (UI)

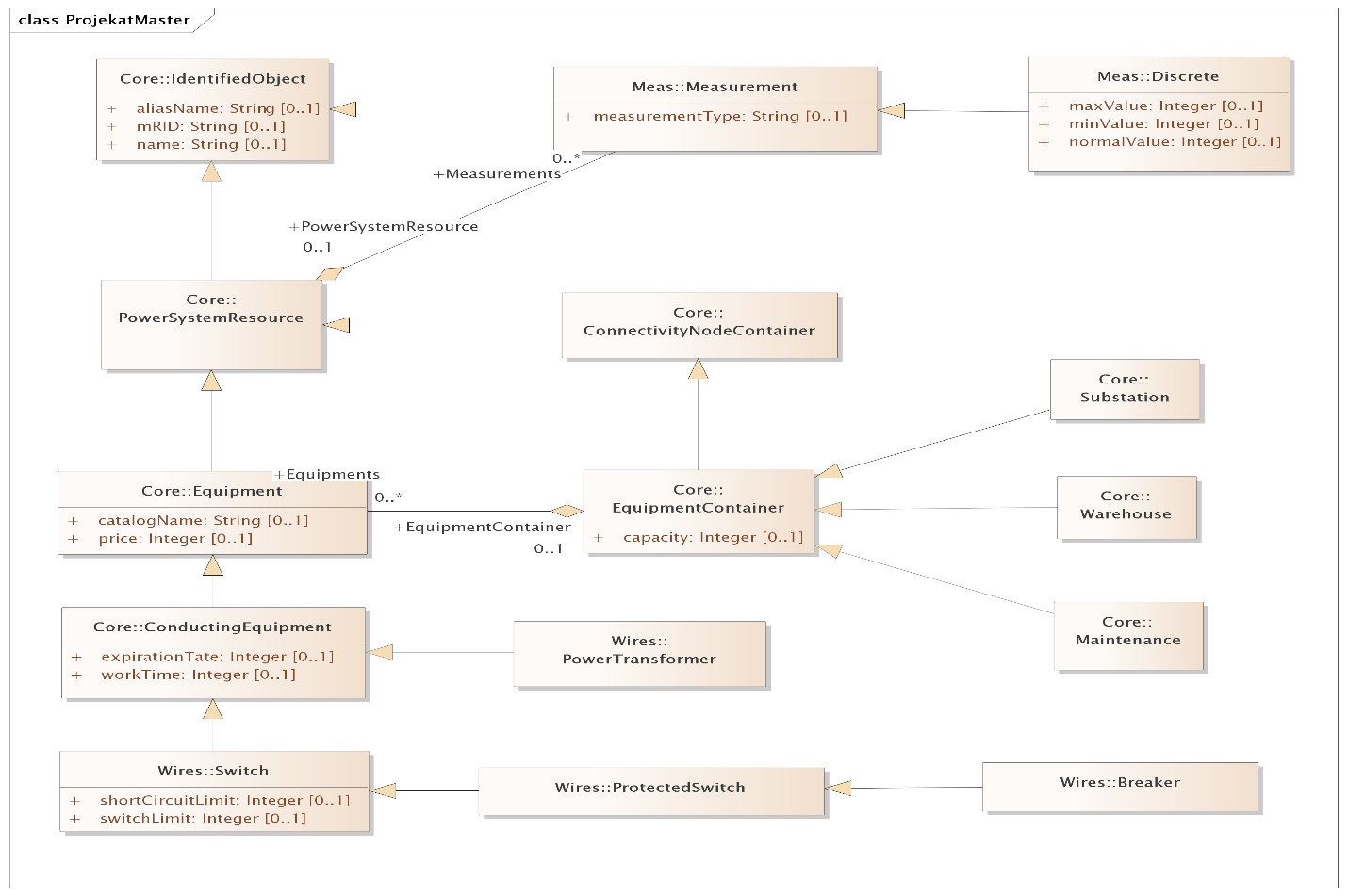
Korisnik za interakciju sa sistemom ima na raspolaganju korisnički interfejs. Funkcionalnosti mu omogućavaju manipulaciju uređajima koji se nalaze u sistemu, njihovo prebacivanje u funkciju i van funkcije, prikaz mnoštva izveštaja, popravku.

# Standardi i modeliranje elektroenergetskih sistema

**CIM** (Common Information Model) predstavlja apstraktni informacioni model koji se koristi u svrhe modelovanja električne mreže i modelovanja opreme koja se koristi u električnim mrežama. Upotreba modela omogućava smanjenje troškova i usmeravanje resursa ka povećanju funkcionalnosti i optimizaciji električnih sistema. Za potrebe modeliranja električnih mreža koristimo deo CIM-a, **CIM profil**, koji sadrži podskup klasa i atributa koji sa svojim međusobnim vezama daju smisaonu celinu. Profil je kreiran u skladu sa specifičnim situacijama u skladu sa kojim su birani i objekti tog profila. Za odgovarajući sistem CIM profil definiše odgovarajuću ontologiju (domensko znanje). Zadavanje CIM profila moguće je u nekoliko oblika: RDFS, XSD, kao tekstualni dokument ili HTML dokument.

## Postupak kreiranja CIM Profila

Upotrebom UML (Unified Modeling Language) jezika za modeliranje u Enterprise Arhitect alatu, na osnovu CIM IEC standarda definiše se odgovarajući klasni dijagram koji zadovoljava potrebe sistema definisanog u tekstu zadatka. Dodaju se klase, atributi klasa, veze između klasa (asocijacija, generalizacija). Takođe se dodaju sve apstaktne klase koje konkretne klase Discrete, Breaker, Substation, Warehouse, Maintenance i PowerTransfromer nasleđuju. Kao rezultat dobija se dijagram prikazan na slici 3.

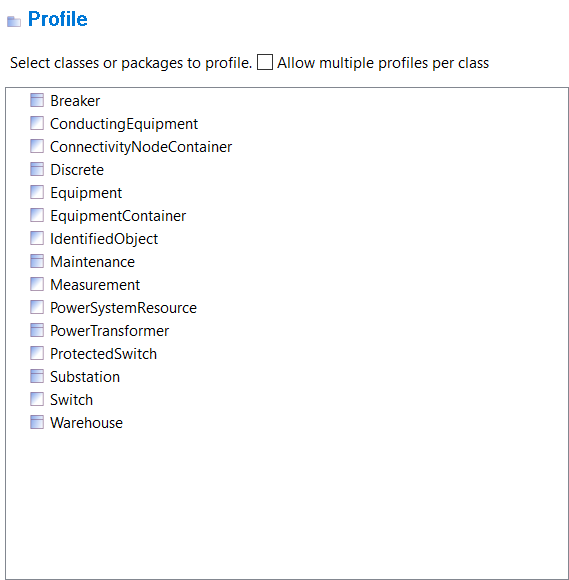


Slika 3. Klasni dijagram - ProjekatMaster

Transformator (**PowerTransformer**) je električna mašina za transformaciju napona uz poštovanje principa očuvanja prenesene snage. Prekidač (**Breaker**) ,deo rasklopne opreme, je uređaj za uključenje i isključenje električnih kola u svim mogućim režimima. U sistemu postoje samo diskretna merenja, odnosno merenja predstavljena diskretnim vrednostima (**Discrete**).

Osim opreme, prekidača i transformatora, definisani su i kontejneri za opremu **WAREHOUSE** – skladište koje čuva kupljenu i popravljenu opremu, **MAINTENANCE** – laboratorija u koju se šalje oprema koja treba da bude popravljena kao i **SUBSTATION** koji predstavlja trafostanicu (razvodno postrojenje) u kojem se nalazi oprema koja je u upotrebi.

Uz pomoć opcije Export Model to XMI u Enterprise Architect-u eksportuje se model definisan preko klasnog dijagrama u XMI format. XMI predstavlja standard za razmenu meta podataka. Definiše odgovarajuće objekte i njihove atribute, a postoji i mogućnost povezivanja objekata u istom ili različitim fajlovima. XMI model se koristi u daljem procesu modeliranja. Za kreiranje CIM profila na osnovu šeme modela date u XMI formatu koristi se opensource alat CIMTool koji je implementiran kao plugin za Eclipse platformu. Prvi korak je kreiranje novog CIMTool projekta. Zatim se importuje šema definisana u XMI formatu. Sledeći korak je kreiranje CIM profila i odabir formata u kojem će CIM profil biti zapisan. U konkretnom slučaju to je treća opcija *Builder for legacy-rdfs*. Potom se kreira CIM profil dodavanjem klasa na osnovu UML dijagrama definisanog pomoću Enterprise Architect-a. U CIM profil se dodaju sledeće klase: *IdentifiedObject, PowerSystemResource, ConnectivityNodeContainer, Equipment, EquipmentContainer, Substation, Warehouse, Maintenance, ConductingEquipment* iz *Core* paketa, *Measurement* i *Discrete* iz *Measurement* paketa i *PowerTransformer, Switch, ProtectedSwitch i Breaker* iz *Wires* paketa*.* Takođe dodaju se i atributi svake klase koji su definisani u tekstu zadatka kako bi sistem funkcionisao. Rezultat opisanog postupka je CIM profil definisan pomoću RDF-a u obliku RDFS fajla (RDF Scheme). Na slici 4. prikazan je profil kreiran pomoću CIMTool alata.



Slika 4. Profil iz CIMTool-a

Nakon dobijenog CIM profila u obliku RDFS file potrebno je parsirati dobijeni fajl kako bi se dobio Profil u memoriji (model in memory). Parsiranje RDFS file-ova se radi pomoću RDFS SAX parsera. SAX (Simple API for XML) parser koji se koristi za parsiranje XML dokumenata. Parser je *event driver* – kada se pročita predodređeni element XML-a, pozivaju se ogovarajuće metode za obradu.

Za čitanje XML dokumenata koristi .NET XmlReader. RDFS SAX parser se koristi radi programske manipulacije definisanog CIM profila u RDFS file-u. Rezultat parsiranja je objekat tipa *Profile* koji u sebi sadrži sve potrebne informacije CIM profila definisanog RDFS-om, prikazan na slici 5.

Implementacija RDFS SAX Parsera obuhvata: *StartDocument()* gde se inicijalizuju promenljive za kreiranje. *StartElement()* gde se pre poziva metode prikupljaju svi atributi odgovarajućeg resursa. Poziva se metoda i prikupljaju podaci za kreiranje objekata koji opisuju resurse. Zatim *EndElement()* gde se na osnovu prikupljenih informacija u metodi StartElement() kreiraju objekti koji opisuju RDFS resurse i dodeljuju profilu. Obrada je podeljena na nekoliko celina: Prvi koji obradjuje resurse definisane RDFS-om, određen je delom koda ograničenog uslovom ***if (qName.Equals(rdfProfileElement))***. Ova celina je dalje podeljena na obradu resursa različitog tipa:

* Paket: ***if (string.Compare(ExtractSimpleNameFromResourceURI(type), "ClassCategory") == 0)***
* Klasa: ***else if (string.Compare(ExtractSimpleNameFromResourceURI(type), "Class") == 0)***
* Atribut: ***else if (string.Compare(ExtractSimpleNameFromResourceURI(type), "Property") == 0)***

Ostatak metode obrađuje podatke vezane za:

* Labele: ***else if (qName.Equals(rdfsLabel))***
* Komentare: ***else if (qName.Equals(rdfsComment))***

Metoda Characters() koja prikuplja definisane vrednosti i na kraju EndDocument() koja oslobađa resursa i vrši dodatno procesiranje objekta tipa *Profile*.

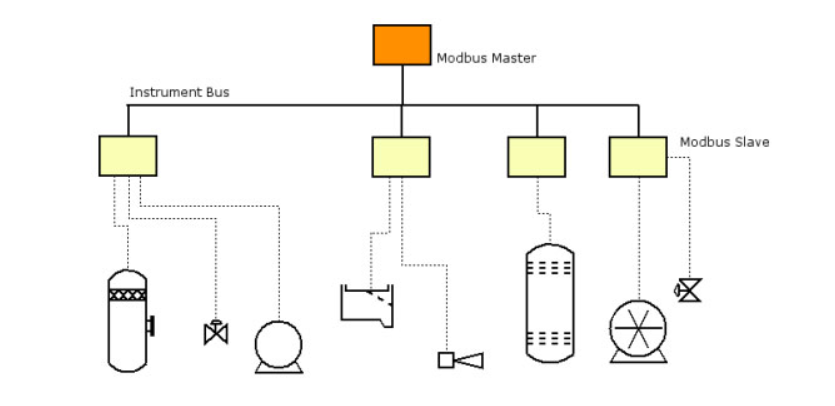


Slika 5. Dobijanje CIM profila

# Simulacija elektroenergetskih sistema sa kritičnom misijom

## 5.1 Modbus protkol

Modbus je nebalansirani, serijski komunikacioni protkol namenjen komunikaciji između programabilnih logičkih kontrolera – PLC-ova. Karakterišu ga jednostavnost i robusnost. Razvijen je za komunikaciju preko asinhrone serijske magistrale. Podržava jedan master i više slave uređaja u jednoj modbus mreži. Ahitektura Modbus protokola može se videti na slici 6.



AUB

PLC

Slika 6. Arhitektura Modbus protkola

Protkol definiše sastav grupe poslatih bajtova kao i skup funkcijskih kodova koje te poruke mogu prenositi. Koriste se tri tipa Modbus prenosa:

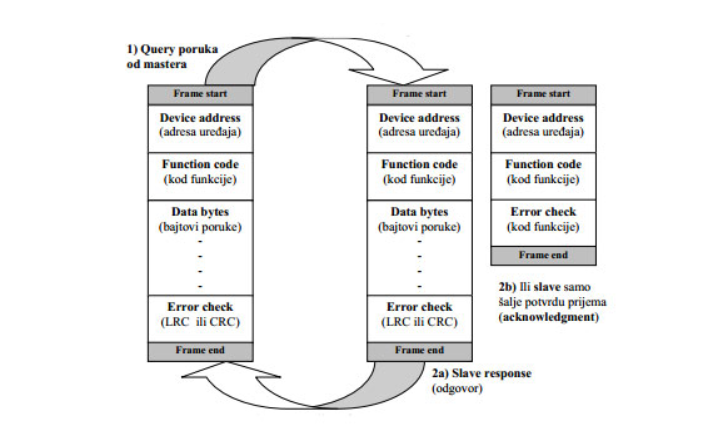
* **ASCII** – jedan byte poruke se sastavlja od dva ASCII karaktera
* **RTU** – poruke se sastoje od binarnih bajtova
* **Modbus/TCP** – modbus poruka se ugrađuje u standardni okvir TCP/IP poruke

Modbus poruka se sastoji od četiri osnovna elementa, koji se šalju po istom rasporedu. Poruka se sastoji od:

1. **Device address** – adrese primaoca
2. **Function code** – koda funkcije koju treba izvršiti
3. **Data** – bajtovi poruke
4. **Error check** – polja za proveru greške

Početak i kraj poruka se prepoznaju preko **Frame start** i **Frame end** karaktera ili signala. Ti signali zavise od tipa prenosa. Modbus komunikaciju uvek počinje **master** jedinica koja šalje poruku koja se zove master query i koja se uobičajeno obraća jednoj **slave** jedinici. Jedna slave jedinica se prepoznaje na osnovu svoje adrese koja se slaže sa **Device address** poljem i sprema odgovor.

Odgovor može biti puna Modbus poruka, koja sadrži podatke koje master traži, a može biti i samo potvrda (**ACK – acknowledgement**) koja se sastoji od 1. 2. i 4. elementa Modbus poruke. Šematski prikaz tipične Modbus komunikacije ptikazan je slici 7.

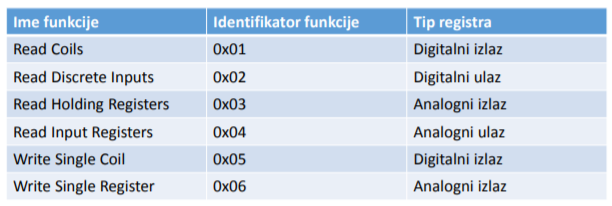


Slika 7. Modbus komunikacija

Tipovi Modbus poruke mogu biti:

* Čitanje digitalnih izlaza
* Čitanje digitalnih ulaza
* Čitanje analognih izlaza
* Čitanje analognih ulaza
* Pisanje digitalnih izlaza
* Pisanje analognih izlaza

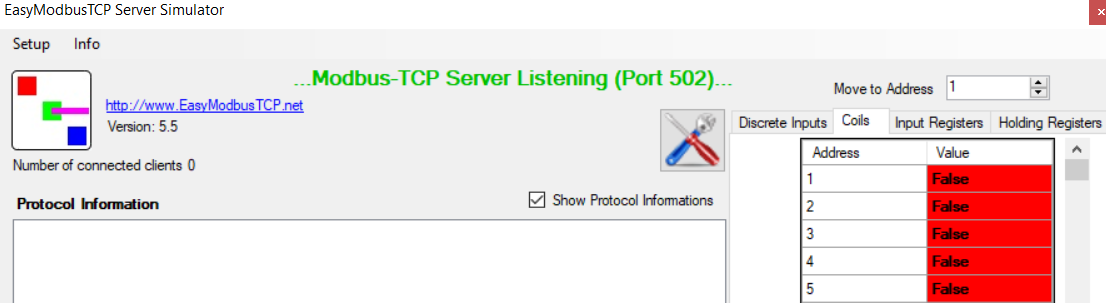
Na osnovu **Funtion code – kod funkcije** se zna o kojem tipu poruke se radi. Na slici 8. prikazani su svi kodovi funkcije koji se koriste u Modbus poruci.



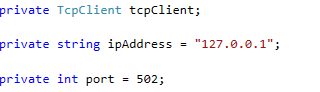
Slika 8. Function codes

## 5.2 Modbus TCP

Omogućena je kontrola i akvizicija podataka na daljinu preko interneta. Modbus/TCP je internet protkol i može da se koristi za razmenu podataka između dva uređaja bilo gde u svetu. Master (client) šalje poruku do specifirane IP adrese I porta. Svali slave uređaj (server) ima svoju specifičnu IP adresu. Po konvenciji mnogi slave uređaji koriste Ethernet port 502 kao standard. Kao što je slučaj i kod EasyModbusTCP simulatora koji je korišćen u izradi projekta (Slika 9.) i čiji je klijent podešen na port 502 i IP adresu. (Slika 10.)



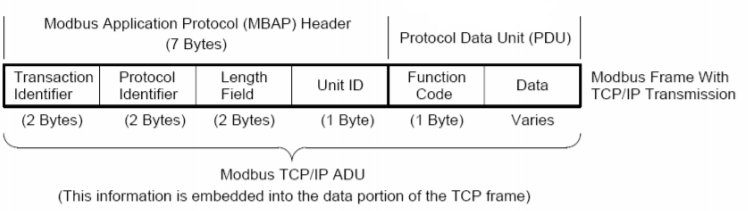
Slika 9. EasyModbus



Slika 10. ModbusClient

Primenom ModbusTCP korisnik može od kuće da dobija podatke iz fabrike i/ili da vrši kontrolu rada fabričkih prenosa. Modbus TCP/IP koristi TCP/IP internet protokol koji omogućava pouzdan prenos podataka preko interneta. Svaki uređaj koji koristi Ethernet fizičku vezu može da komunicira preko Modbus/TCP. Performance Modbus TCP/IP zavise od korišćenog hardware-a, ali i od samog saobraćaja na internetu. Modbus/TCP treba koristiti za održavanje, traženje kvarova, i sporu superviziju sistema.

Modbus/TCP poruka se razlikuje od klasične Modbus poruke po tome što je polja **Device address** i **Error check** izbačeni, a ostatak ugrađen posle zaglavlja Modbus Application Protokola. Adresa je izbačena iz očiglednih razloga što se komunikacija sada odvija preko interneta pa se i adrese drugačije ponašaju u odnosu na adrese kod klasične Modbus poruke. Takođe i provera greške je izbačena iz razloga što Ethernet TCP/IP sloj brine o tome i garantuje intergritet podataka. (Slika 11.)

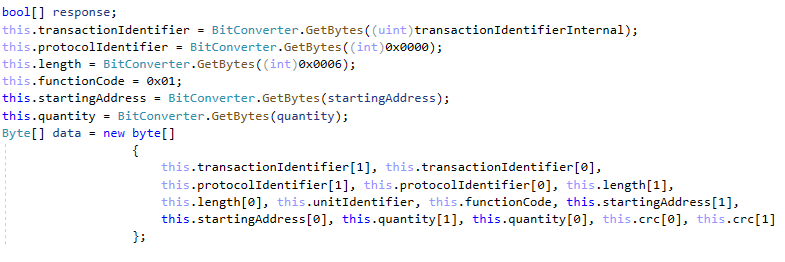


Slika 11. Modbus/TCP poruka

U projektu je korišćen je Modbus TCP/IP protokol i konekcija se uspostavlja pomoću klase ModbusClient i tu se takođe nalaze funkcije za čitanje i upis bajtova sa i na simulator. U klasi ModbusProtocol nalaze se svi parametri koji su potrebni za konfigurisanje jedne modbus poruke: *transactionId* – Id transakcije koji se svaki put povećava za jedan, *protocolId* – Id protokola, uvek je 0 jer to označava MODBUS protokol, *length* – dužina poruke, *unitId* – slave adresa i *functionCode* – kod funkcije koja označava o kojem tipu poruke se radi. Kao I *startingAddress* koja definiše adresu od koje počinje čitanje i atribut *quantity* koji govori koliko bajtova se želi pročitati.

Komunikacija počinje tako što SCADA servis prilikom podizanja ažurira svoj model sa modelom CalculationEngin-a tako što poziva metodu *CheckIntegrity()* proveravajući integritet baza SCADA servisa i CalculationEngine servisa. Zatim poziva *WriteSingleCoil()* kako bi se upisala sve vrednosti digitalnih izlaza na master servisu. Prilikom pokretanja SCADA servisa poziva metodu *PollContinuously()* i time počinje da čita nadolazeće promene od ModbusTCP klijenta pomoću *ReadCoils()* što označava da čita digitalne izlaze i ukoliko uspe da dobije odgovor, ažurira svoj model na najnovije promene.

Sa druge strane ModbusClient pakuje odgovor u funkciji *ReadCoils()* po pravilima za ModbusTCP poruke sa slike 9. Izgled odgovora ModbusClient-u u kodu vidi se na slici 12.

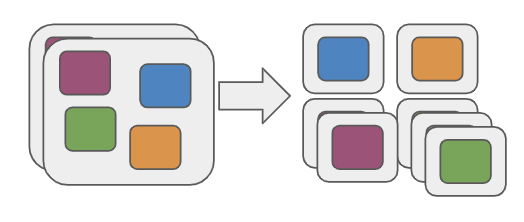


Slika 12. Odgovor Modbus Client-a

# Održavanje i kontrola kvaliteta elektroenergetskog softvera

Kada su zahtevi aplikacije mali i minimalni, većina kompanija će se odlučiti pre za monolitičku arhitekturu aplikacije iz razloga što je lakša za implementaciju i više primenljiva. Što je i logično s obzirom da time ima manje komponenti u igri. Ali s obzirom da razvojem tehnologije i dodavanjem više funkcionalnosti monolitičke aplikacije su postajale sve veće i komplikovanije. Podrška aplikacijama je postala skuplja sa strane vremena i novca.

*Microservice Arhitecture* opisuje određeni način dizajniranja softverskih aplikacija kao skup nezavisno razvijenih servisa koji se izvršavaju na *Cloud*-u. Mikroservisi predstavljaju način podele velikih softverskih rešenja na manje, nezavisne servise (Slika 13.).



Slika 13. Monolithic vs Microservices

## 6.1 Microservisna arhitektura vs. Monolitička arhitektura

Monolitičke aplikacije su dobro rešenje ukoliko je cilj samo napraviti malu aplikaciju za in-house funkcije ili za kancelarijske poslove. Monolitičku aplikacija se lakše i brže razvija i ukoliko nema veći broj dodatnih funkcija. Ali u koliko aplikacija treba da izrži veći broj korisnika i preuzme veći broj novih funkcija i usluga tada se prelazi na mikroservise.

Sa mikroservisnim aplikacijama može se skratiti vreme potrebno za objavljivanje *update*-ova. Može se koristiti više usluga nezavisno jedne od drugih. Takođe mogu se skalirati pojedine funkcije u zavisnosti od potražnje. I na kraju moguće je izbeći kaskadne kvarove na mikroservisima. Ako određeni servis padne, neće uzrokovati pad čitave aplikacije.

## 6.2 Prednosti i mane Mikroservisne arhitekture

Prednosti Mikroservisne arhitekture su:

* **Jednostavnije komponente** – Podelom NMS i CalculationEngine servisa, kao i UI Adaptera na mikroservise, servisi su manji i lakši za razumevanje čime je smanjena kompleksnost softvera.
* **Održavanje i nadogradnja softvera** - Softver je lakši za izgradnju i održavanje kada se podeli u set manjih servisa.
* **Produktivnost i brzina** – Mogućnost rada na različitim komponentama istovremeno bez čekanja na ostale delove softvera daje bolju produktivnost i brzinu.
* **Testiranje** – Svaki mikroservis unutar aplikacije se može pojedinačno testirati i mogu se komponente testirati koje su razvijene dok se radi na razvoju drugih komponenti.

Mane Mikroservisne arhitekture sistema su:

* **Skupo održavanje** – Izbor tehnologija prilikom izgradnje komponenti prouzrokuje problem nejednakog dizajna što povećava cenu održavanja softvera.
* **Povećana potreba za resursima** – Potreba za više memorije i veća jačina procesora iz razloga sto se sve komponente pokreću nezavisno.
* **Povećana mrežna komunikacija** – Nezavisno pokretanje komponenti koje međusobno komuniciraju preko mreže zahteva pouzdane i brze mrežne veze.
* **Različita interpretacija podataka** – Svaki servis može da ima svoju interpretaciju podataka, a time je povećan zahtev za obradu podataka.
* **Mrežna sigurnost** – Iz razloga što je osnova mikroservisne arhitekture komunikacija između servisa, servisi postaju sigurnosno ranjivi.

## 6.3 Reliable Services

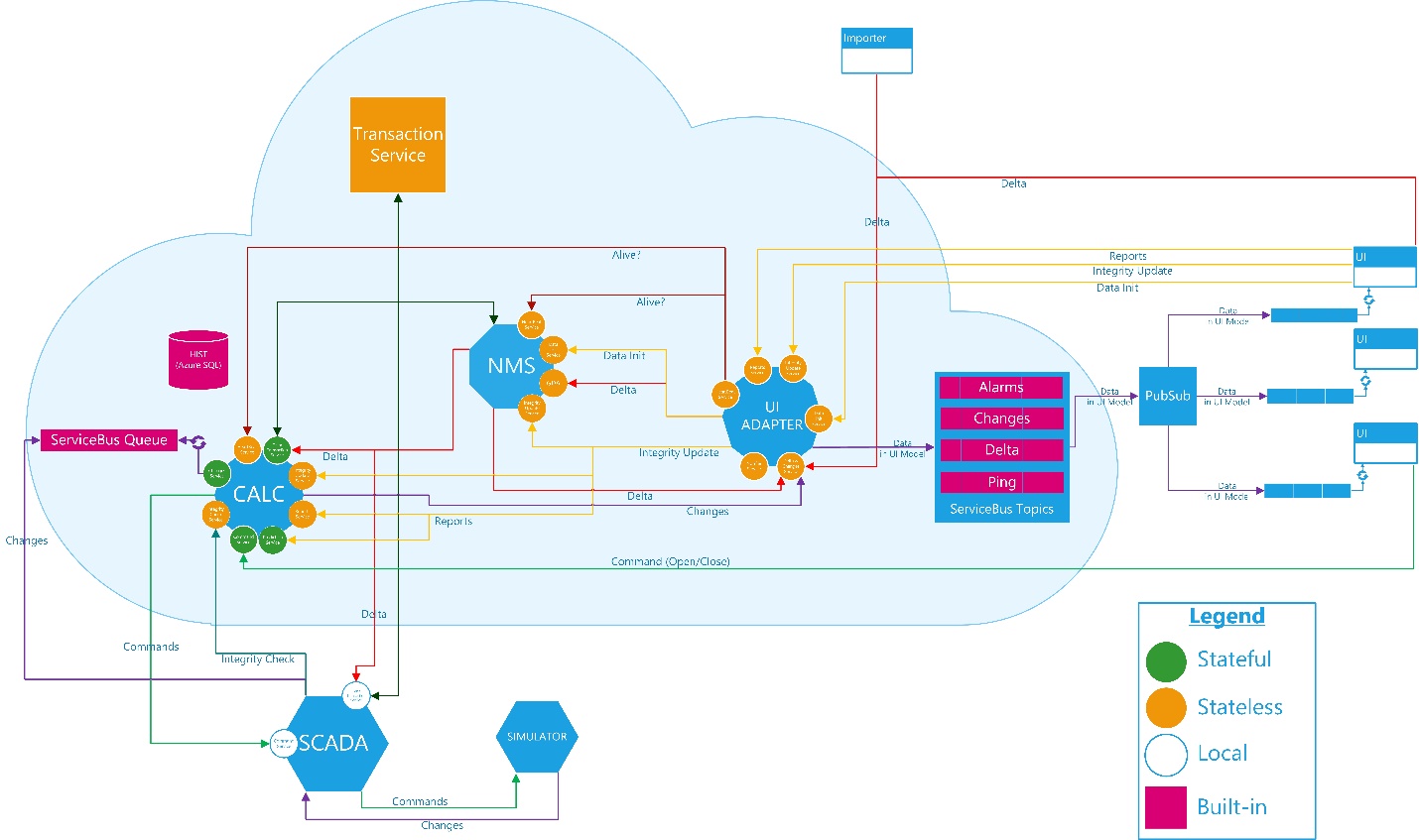
Velika prednost Mikroservisne arhitekture je upotreba *Reliable Service*-a, jednog od programskih modela dostupnih na *Service Fabric*-u. Reliable Services korisnicima pruža model koji je jednostavan za programiranje, lako uklopljiv model, podržava sve vrste HTTP, TCP komunikacije, kao i druge protokole. Reliable Services između ostalog omogućavaju:

* **Pouzdanost** – Servis uvek ostaje dostupan čak i ako je u nepouzdanom okruženju.
* **Dostupnost** – Servis je konstantno dostupan. *Service Fabric* održava željeni broj instanci koji se zadaje u konfiguraciji servisa.
* **Skalabilnost** –Servisi su odvojeni od specifičnog hardvera i mogu se povećavati ili skanjivati po potrebi u zavisnosti od dodavanja ili uklanjanja hardvera i drugih resursa. Servisi se lako mogu dodavati i brisati dinamički putem koda ili komandne linije tako omogućavajući više instanci po potrebi.
* **Konzistentnost** – Garantuje se konzistentnost svih podataka koji se čuvaju u servisu.

U vezi konzistentnosti postoje dve vrste servisa:

* **Stateless servisi** – koji ne čuvaju podatke na eksternim memorijama ili unutar servisa
* **Stateful servisi** – koji čuvaju podatke na eksternim memorijama ili unutar servisa

Na slici 14. koja prikazuje arhitekturu sistema može se videti da su *Transaction Service*, *NMS*, *Calculation Engine* i *UIAdapter* prebačeni na mikroservisnu arhitekturu iz razloga što se na njima vrši najviše proračuna/formatiranja sa akcentom na *CalculationEngine* servis. Takođe i *PubSub* je proširen *ServiceBus*-om kako bi koristio *topic*-e koji su definisani i nalaze se na *Cloud*-u. *SCADA* servis nije prebačen na mikroservisnu arhitekturu iz razloga što bi time bile smanjene performanese *SCADA* servisa koji mora da bude blizu polja, lokalno.



Slika 14. Cloud arhitektura

Sa slike 14. se vidi i da su mikroservisi kod *NMS* servisa, *UIAdapter*-a kao i *Transaction* servisa svi stateless dok u slučaju *CalculationEngine*-a sa slike 15. se vidi da su servisi *DeltaTransactionService*, *ChangesService*, *CommandService*, *PredictionService i ExpiryCheckService* stateful mikroservisi jer oni čuvaju podatke u svojim reliable kolekcijama i takođe rade sve proračune, dok se drugi servisi koji su stateless obraćaju njima ukoliko im je potreban neki podatak i komuniciraju sa ostalim komponentama sistema. Suština Stateful servisa je da se u slučaju gašenja primarnog čvora aktivan postane jedan od sekundarnih.



Slika 15. CalculationEngine

Rešenje je implementirano tako da svaki od stateful servisa ima svoj *CalculationModel*, koristeći *RealiableDictionary* kolekcije za čuvanje podatke. S obzirom da u sistemu učestvuju *Breaker*, *PowerTransformer* i *Discrete*, servisi u sebi imaju neke od sledeće tri kolekcije u zavisnosti da li taj servis radi sa tim elementima:



Slika 16. Reliable kolekcije

Dodatno su napisani i posebni modeli za *Breker*, *PowerTransformer* i *Discrete* jer u slučaju da neki od tih servisa takođe i upisuju neke podatke u bazu kako bi se sprečilo dupliranje koda jer bi u tom slučaju svi servisi imali sve modele sa svim propertijima, čak i sa onim sa kojim ne rade i koji im nisu od značaja.

Prednosti *Realiable* kolekcija u odnosu na druge kolekcije su:

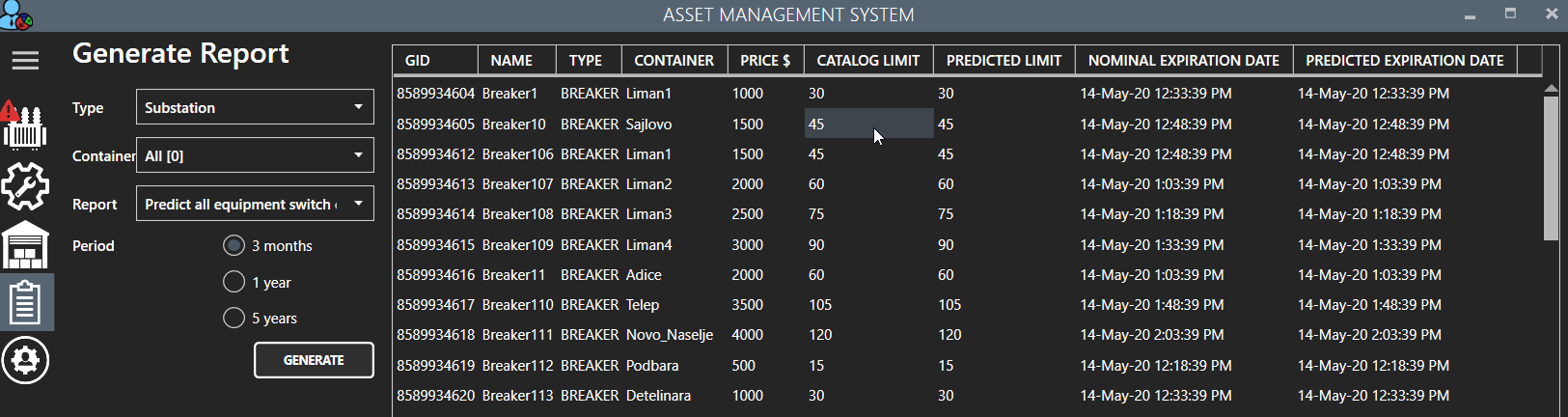
* **Replikacija** – promena stanja se repliciraju i time se postiže velika dostupnost.
* **Asinhroni API** – API-ji su asinhroni tako da se osigurava da thread-ovi ne budu blokirani prilikom IO operacija.
* **Transakcija** – Lako se može upravljati sa više Reliable kolekcija unutar servisa.
* **Postojanost** – Podaci se čuvaju na disku radi trajnosti čime se takođe i sprečavaju kvarovi npr. nestanak struje datacentra.

# Napredni računarski sistemi sa kritičnom misijom u elektroenergetici

## 7.1 Reportovi predikcije

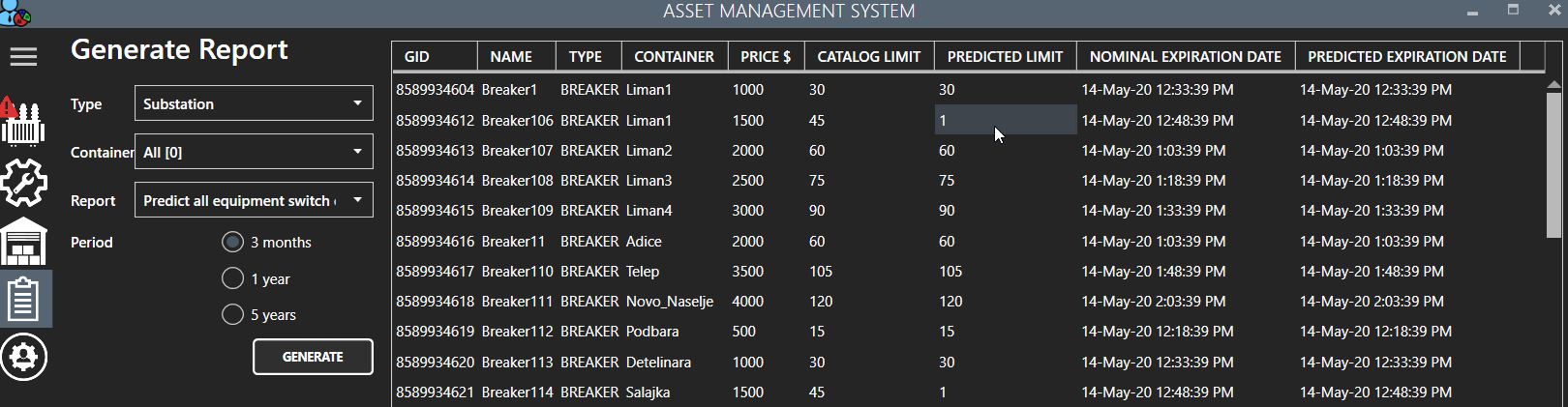
Report ili izveštaj je dokument koji predstavlja informacije koje su organizovane u formatu za određenu svrhu. Iako se sažeci izveštaja mogu dostavljati usmeno, potpuni izveštaji su gotovo uvek u obliku pisanih dokumenata. Reportovi za predikciju u sistemu iz teksta zadatka prvenstveno služe kako bi se unapred odredilo koji od elemenata (prekidača) će prestati sa radom u nekom narednom periodu i takođe koji je maksimalan broj manipulacija pre nego se prekidač šalje na remont. Prekidači koji pripadaju istom katalogu imaju definisan isti ServiceTime (vreme posle kojeg prekidači moraju da isti na servis), WorkTime (vreme ukupnog rada prekidača) kao i SwicthLimit (nominalni broj manipulacija prekidača) i ShortCircuitLimit (nominalan broj kratkih spojeva). Na osnovu toga se vidi da prekidači koji pripadaju istom katalogu utiču jedni na druge.

Ukoliko se komanduje jednim prekidačem (Breaker10 sa slike 17.) i zatim se prekidač prebaci u Maintenance (simulira se da je prekidač prekinuo svoj rad i prebačen na remont), računanjem predikcije dobija se nova prediktovana vrednost SwitchLimit-a za ostale prekidače koji pripadaju istom katalogu.



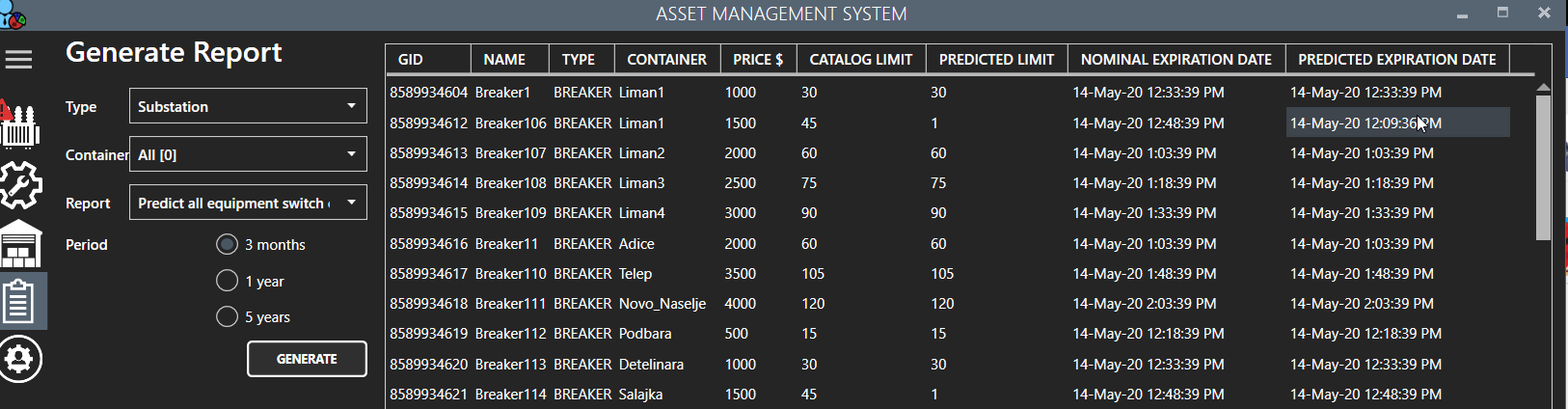
Slika 17. Prediction report #1

Sa slike 18. može se videti da prekidači koji pripadaju istom katalogu kao i Breaker10 imaju novi PREDICTED LIMIT 1 iz razloga što je Breker10 simulirano prestao da radi sa radom nakon samo jedne manipulacijje.



Slika 18. Prediction report #2

Zatim ukoliko se uzme neki drugi prekidač (Breaker106 sa slike 19.) koji takođe pripada istom katalogu, izvrši se jedna manipulacija, videće se i novo prekiktovano vreme za taj prekidač. Vidi se da je PREDICTED EXPIRATION DATE za Breaker106 manji nego NOMINAL EXPIRATION DATE.

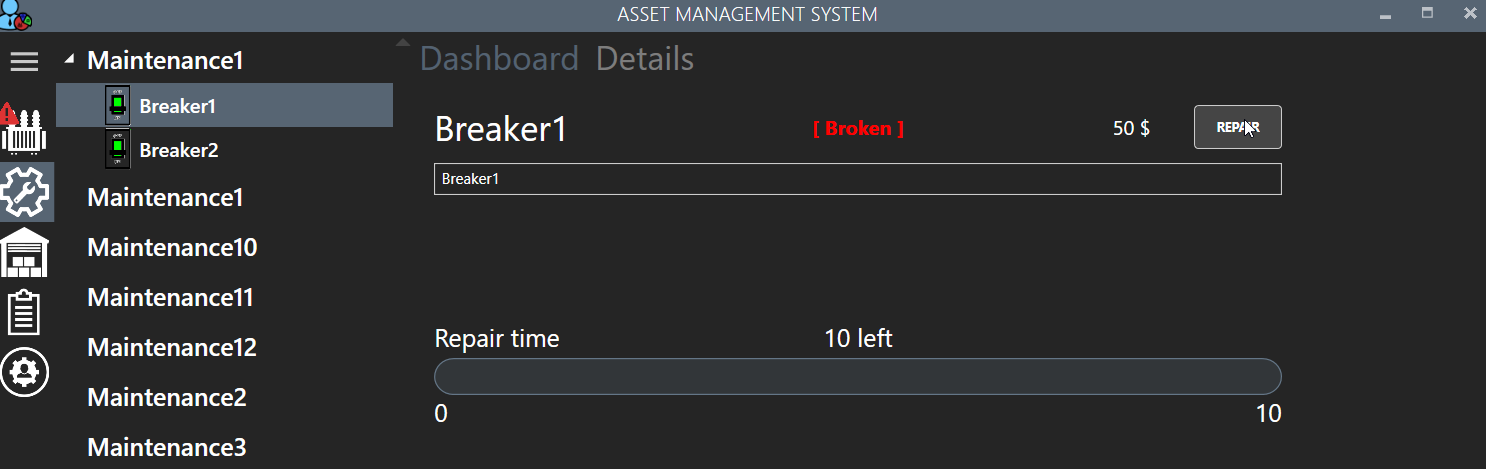


Slika 19. Prediction report #3

Predikcija se vrši na osnovu dve metode. Prva metoda *CalculateHISTLimit*() je zadužena za računanje SwitchLimit nakon prebacivanja Breaker-a u Maintenance. Metoda računa SwitchLimit za sve prekidače iz jednog kataloga tako što traži srednju vrednost *Mu* za sve SwichCount-e. Na osnovu toga dobijamo novi PREDICTED LIMIT za sve Breaker-e iz tog istog kataloga. Zatim uz pomoć druge metode koja se nalazi u CalculationModel-u *PredictNextTime()* računa predikciju za opremu kojoj prosleđuje sve istorijske vrednosti za SwicthCount, koji u bazi ima vreme kada je doslo do promene SwitchCounta kao i *long* vrednost koliko je SwitchCount tada bio. Pored toga šalje i HistoricalLimit tog breaker-a za koji se računa predikcija. Povratna vrednost funkcije je novo vreme *predictedTime.* Tako se dobija nova vrednost PREDICTED EXPIRATION DATE-a sa slike.

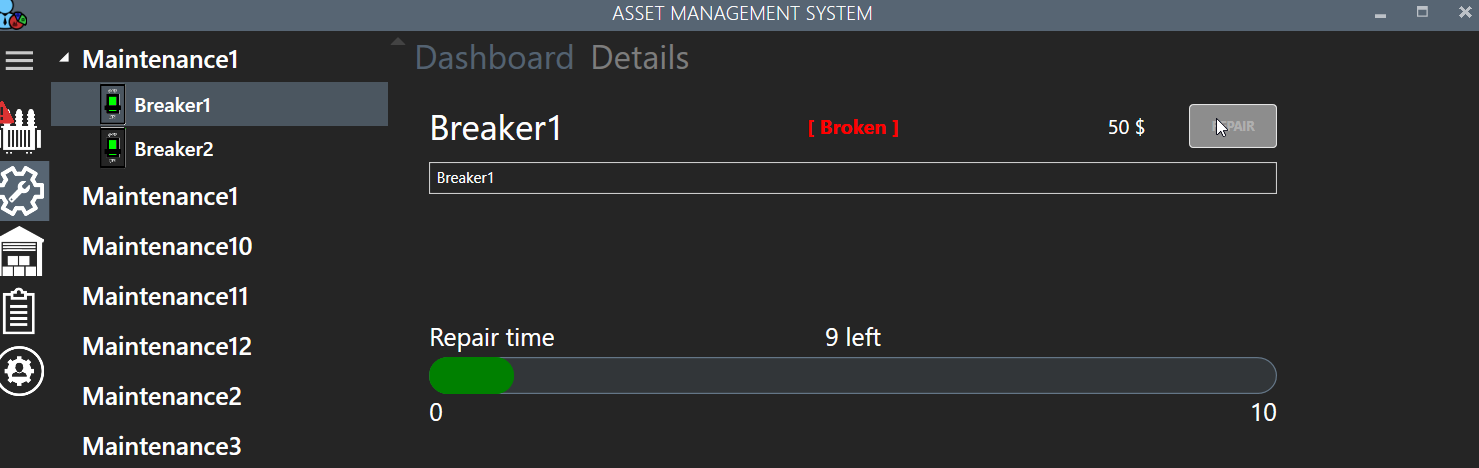
## 7.2 Remont prekidača

Prekidači koji dostignu maksimalan broj manipulacija šalju se na remont unutar Maintenance kontejnera. Remont prekidača predstavlja popravku prekidača na taj način što mu se vrednost SwichCount resetuje na 0. Klikom na REPAIR dugme započinje proces popravke Breakera. Breaker ima unapred definisano vreme popravke koje u slučaju sa slike iznosi 10 sekundi.



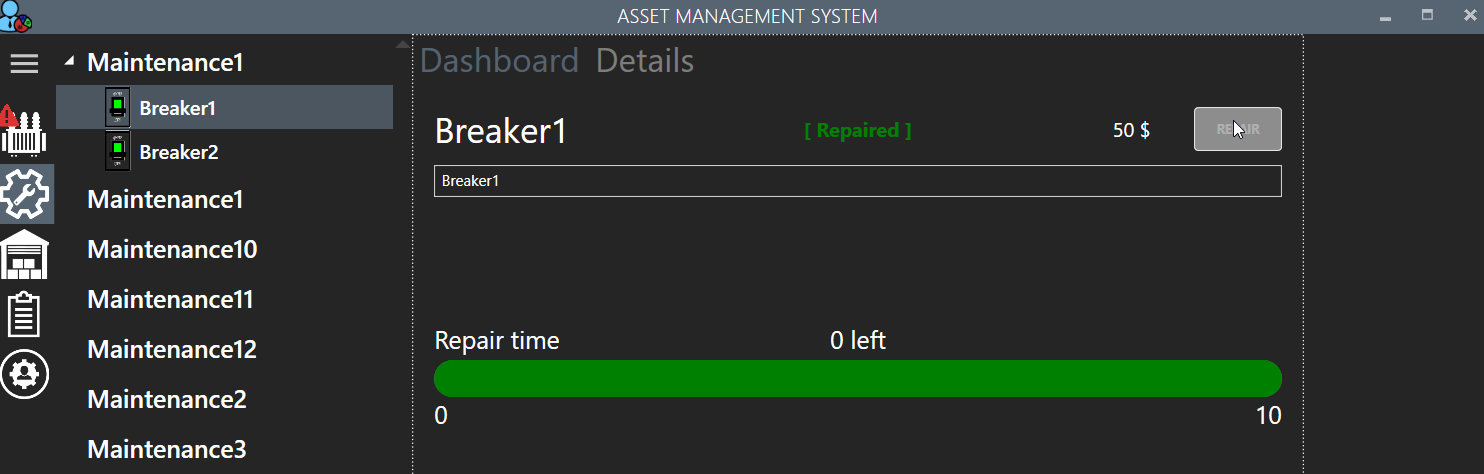
Slika 20. Remont prekidača #1

Breaker je trenutno u stanju Broken kao što se vidi na slike 20. i definisana je cena popravke koja iznosi 10 posto od cene nabavke Breakera.



Slika 21. Remont prekidača #2

Nakon isteka predefinisanog vremena Breaker se nalazi u stanju Repaired i kao takav može se ponovo pustiti u pogon uz pomoć Asset Managera (Slika 22.).



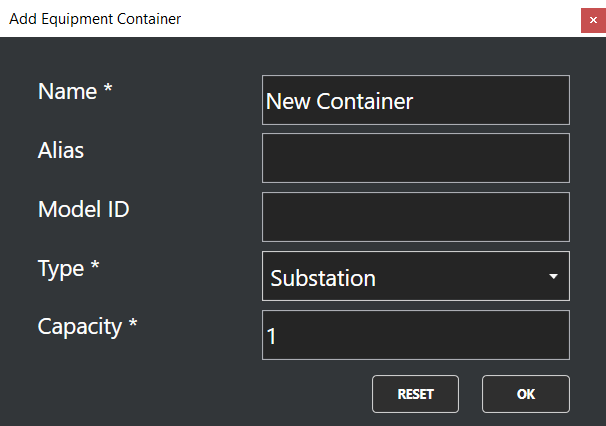
Slika 22. Remont prekidača #3

# Sigurnost i bezbednost u SmartGrid sistemima

## 8.1 Prikaz elektroenergetske opreme koja je nabavljena u poslednjih mesec/kvartal/godina

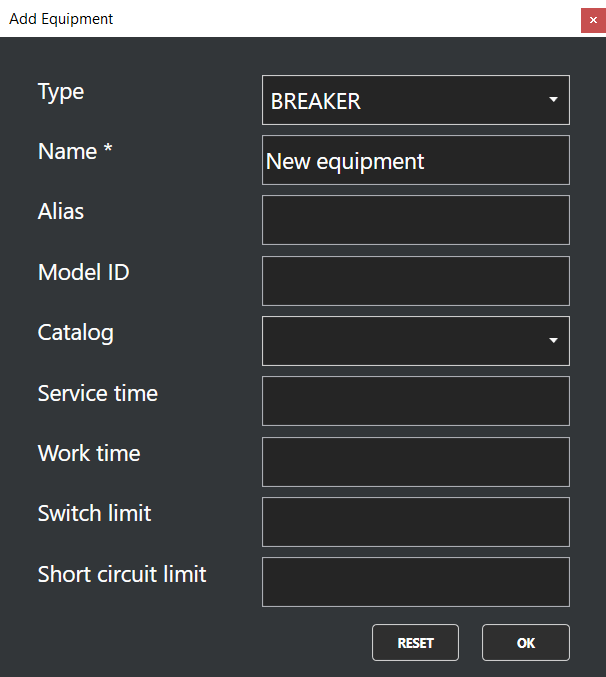
Opremu sistema Asset Management predstavljaju prekidači (Breaker) i transformatori (PowerTransformer). Opremu se ubacuje u sistem na dva načina. Prvi način je preko Importer servisa i ModelAps aplikacije u kojoj se unosi bulk kontejnera (substation, warehouse, maintenance) kao i bulk opreme (breaker, powertransformer) i merenja/signala (discrete) koji su definisani u CIMXML file-u. Na taj način, primenom Delte, unose se podaci o opremi i kontejenerima u sistem. Oprema koja bude unešena u kontejner WAREHOUSE, na ovaj način biva obeležena kao PURCHASED (kupljena/nabavljena). Drugi način ubacivanja opreme u sistem je preko AssetManager aplikacije.

Unutar AssetManager aplikacije postoji opcija dodavanja novog kontejnera u kojem se navodi Name (ime), Alias, ModelID, Type (Substation, Maintenance ili Warehouse), kao i Capacity (kapacitet) (Slika 23.) .



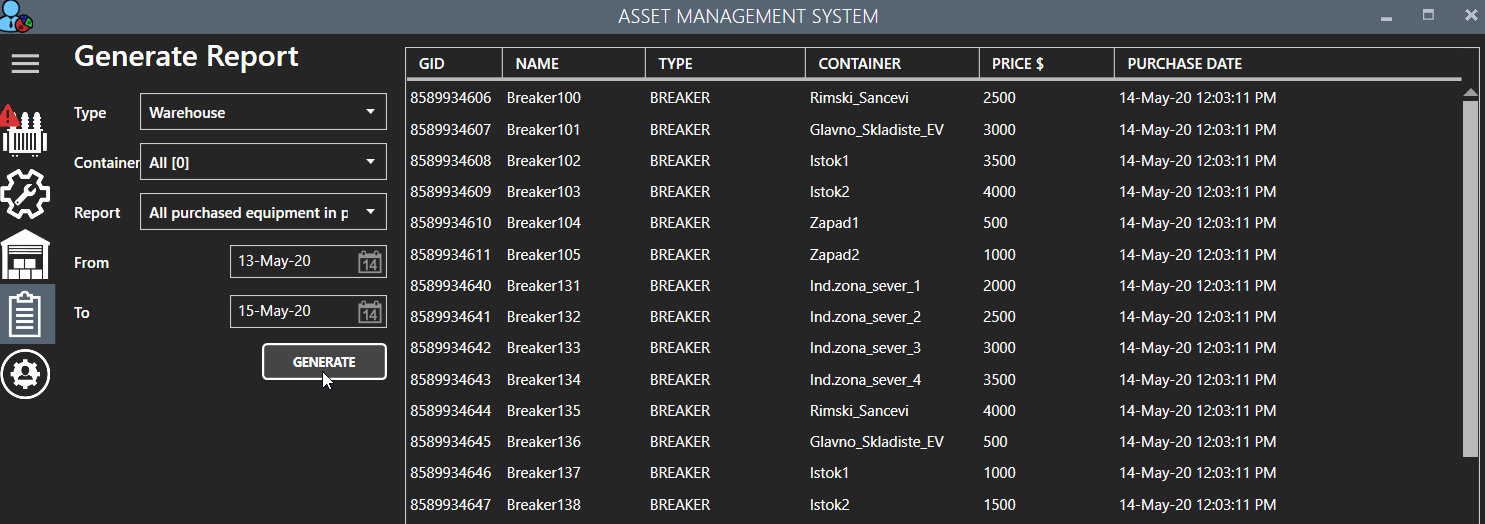
Slika 23. Dodavanje novog kontejnera za opremu

Takodje popstoji i opcija dodavanja nove opreme gde se u novom prozoru navodi Type (Breaker ili PowerTransfromer), Name (ime), Alias, Model ID, bira se katalog za koji će biti vezana oprema na osnovu čega se popunjavaju ServiceTime i WorkTime, SwitchLimit i ShortCircuitLimit. (Slika 24.)



Slika 24. Dodavanje nove opreme

U Reportu *Oprema koja je nabavljena u poslednjih mesec/kvartal/godina* (izveštaju) koji je vezan isključivo za WAREHOUSE tip kontejnera jer samo u tom kontejneru se može nalaziti oprema koja je kupljena navodi se i datum za koji se izveštaj traži (From -> To). Postoji i opcija odabira određenog WAREHOUSE kontejnera ili odabir svih WAREHOUSE kontejnera. Klikom na Generate na prikazu reporta vide se: GID opreme (jedinstveni identifikator), NAME (ime opreme), TYPE (Breaker ili PowerTransformer), zatim CONTAINER (kontejner kojem pripada nabavljena oprema), PRICE (cena nabavke opreme) i PURCHASE DATE (datum kada je kupljena oprema). (Slika 25.)



Slika 25. Report sve opreme koja je nabavljena u poslednjih mesec/kvartal/godina

Proces dobijanja reporta *Oprema koja je nabavljena u poslednjih mesec/kvartal/godina* nakon klika na Generate počinje tako što UI poziva metodu *WarehouseInsertsForPeriod()* na ReportsClientu ako je izabran određen WAREHOUSE kontejner Ili ukoliko su izabrani svi kontejneri poziva se metoda *AllWarehouseInsertsForPeriod().*

Ukoliko je pozvana metoda koja treba da vrati report za određen WAREHOUSE kontejner, *ReportsGenerator* klasa koja se nalazi na *CalculationEngine* servisu pronalazi zadani WAREHOUSE po warehouseGid-u i za njega uzima svu opremu koja je insertovana i koja upada u zadani period from-> to. A ukoliko je pozvana metoda koja treba da vrati opremu za sve WAREHOUSE kontejnere metoda na u klasi ReportsGenerator na CalculationEnginu-u uzima svu insertovanu opremu od svih WAREHOUSE kontejnera koja upada u period from -> to. Na taj način se dobijaju reportovi za *Oprema koja je nabavljena u poslednjih mesec/kvartal/godina* bilo da je reč o određenom WAREHOUSE kontejneru ili da je reč o svim WAREHOUSE kontejnerima.