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

Đorđe Urankar E5-32-2019

**Asset Management**

Dokumentacija projektnog zadatka

- Master akademske studije-

-Novi Sad, 2020-

**Sadržaj**

[1 Uvod 1](#_Toc41317016)

[1.1 Opis rešavanog problema 1](#_Toc41317017)

[1.2 Arhitektura 1](#_Toc41317018)

[1.2.1 Simulator 2](#_Toc41317019)

[1.2.2 Scada 2](#_Toc41317020)

[1.2.3 Calculation Engine (CE) 2](#_Toc41317021)

[1.2.4 NetworkModel Service (NMS) 2](#_Toc41317022)

[1.2.5 UI Adapter 2](#_Toc41317023)

[1.2.6 Publish-Subscribe mehanizam (PubSub) 2](#_Toc41317024)

[1.2.7 Transaction Service 2](#_Toc41317025)

[1.2.8 Korisnički interfejs (UI) 3](#_Toc41317026)

[2 SCADA 5](#_Toc41317027)

[2.1 Modbus Protocol 5](#_Toc41317028)

[3 NMS 7](#_Toc41317029)

[3.1 Validacija modela 7](#_Toc41317030)

[4 Napredni racunarski sistemi 9](#_Toc41317031)

[4.1 Predikcija i metod najmanjih kvadrata odstupanja 9](#_Toc41317032)

[5 Azure – Service Fabric 12](#_Toc41317033)

[5.1 Diagnostic Events 12](#_Toc41317034)

[6 Sigurnost i bezbednost(DMS) 14](#_Toc41317035)

[6.1 Planiranje remonta na osnovu radnog veka 14](#_Toc41317036)

### 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.

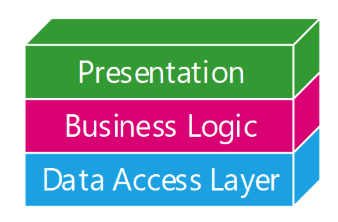
Aplikacija bi trebalo da obuhvati tok kretanja opreme(prekidači i transformatori) kroz elektroenergetski sistem, a samim tim aplikacija se može iskoristiti od više timova u okviru jedne firme. Osim toga aplikacija treba da podrži komandovanje nad opremom, praćenje i čuvanje tih akcija, kao i generisanje raznih izveštaja. Čuvanje tih akcija nam omogućava jednu od najbitnijih funkcionalnosti 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 gubitka 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 1.2.1)



*Slika 1.2.1 – Slojevi aplikacije*

* 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 1.2.8.1 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.

### NetworkModel 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)

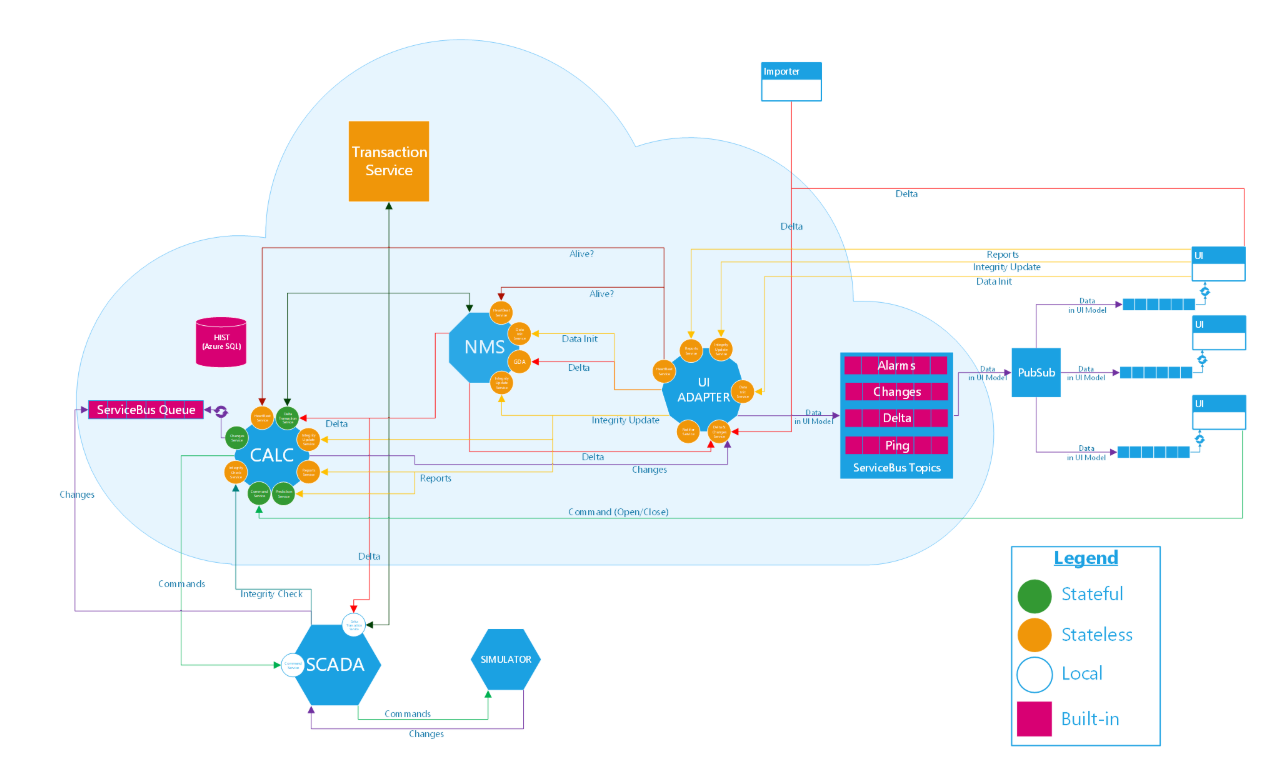
Da bi moglo da se podrži više od jednog korisničkog interfejsa i da to bude nezavisno od celog sistema koristi se publish-subscribe mehanizam koji će pristigle podatke da demultipleksira na svaki od korisničkih interfejsa tako da svi budu u sinhronizaciji.

### 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.

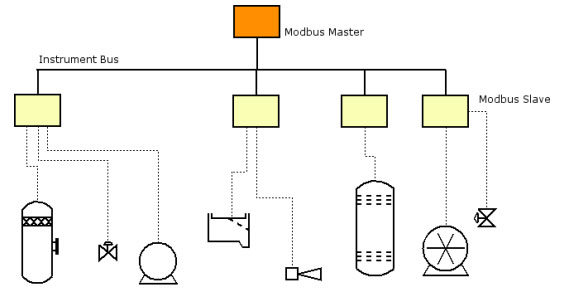


*Slika 1.2.8.1 – Arhitektura celog zadatka*

# SCADA

## Modbus Protocol

Modbus je serijski komunikacioni protokol predstavljen 1979. godine od strane kompanije *Modicon*, namenjen komunikaciji između programabilnih logičkih kontolera – PLC-ova. Karakterišu ga robusnost i jednostavnost. Ovaj protokol je *master/slave* protokol iz grupe Fieldbus protokola. Modbus protokol je zasnovan na serijskoj komunikaciji između *master* jedinice i jedne ili više(do 247) *slave* jedinica spojenih u istu mrežu. Primer ovakve arhitekture prikazan je na slici 2.1.1.



*Slika 2.1.1 – Arhitektura master/slave protokola*

*Master* uređaj može biti računar ili SCADA sistem. *Slave* uređaji se nazivaju RTU(Remote Terminal Unit). To su elektronski uređaji koje kontroliše mikroprocesor, koji određene parametre iz fizičkog sveta skupljaju i putem određenog sistema prosleđuju *master* uređaju.

Modbus komunikacije se odvija preko poruka. Struktura svake poruke je identična i sastoji se od:

1. Adresa uređaja za koji je poruka namenjena, odnosno adresa *slave* uređaja
2. Kod za funkciju koju uređaj treba da izvrši
3. Blok sa podacima nad kojima se primenjuje funkcija
4. Blok podataka za proveru greške

Razmenu ovih poruka inicira *master* uređaj, a u zavisnosti od *slave* uređaja navedenog u samoj poruci, *slave* uređaj odgovara. Konekcija se može izvršiti preko dva bazična moda za transmisiju i kodovanje poruke, *ASCII* i *RTU*. *Modbus/ASCII* salje poruku koja se može čitati u *ASCII* formatu, odnosno čitljiva je za čoveka. *Modbus/RTU* šalje poruku u binarnom formatu koji nije čitljiv za čoveka. *Modbus/TCP* je nova, treća vrsta modbus transfera. Predstavlja varijantu *Modbus/RTU*  transfera koja je prilagođena internet okruženju, i koja primenjuje *TCP/IP* protokol.

U našem projektu iskorišćen je *Modbus TCP/IP* protokol i konekcija se ostvaruje kroz klasu ModbusClient. U ovoj klasi se takođe nalaze funkcije za čitanje i upis bajtova na simulator. U klasi ModbusProtocol definisani su parametri neophodni za konfigurisanje jedne poruke:

* *TransactionIdentifier* – identifikator transakcije koji se autoinkrementuje
* *ProtocolIdentifier* – identifikator protokola, uvek je 0 jer to označava Modbus protocol
* *Length* – dužina poruke
* *unitIdentifier* – *slave* adresa
* *functionCode* – kod funkcije

Kod nas u sistemu pomoću SCADA-e potrebno je komandovati prekidačima. Za ovu svrhu dovoljno je omogućiti funkcionisanje digitalnih signala. To je zahtevalo korišćenje funkcije *WriteSingleCoil*, koja se koristi za upis digitalnih izlaza, kao i korišćenje funkcije *ReadCoils* koja se koristi za čitanje digitalnih izlaza.

# NMS

## Validacija modela

NMS(*Network Model Service*) servis je jedan od glavnih servisa, čija uloga je da grupiše podatke u model i da skladišti podatke u bazi.

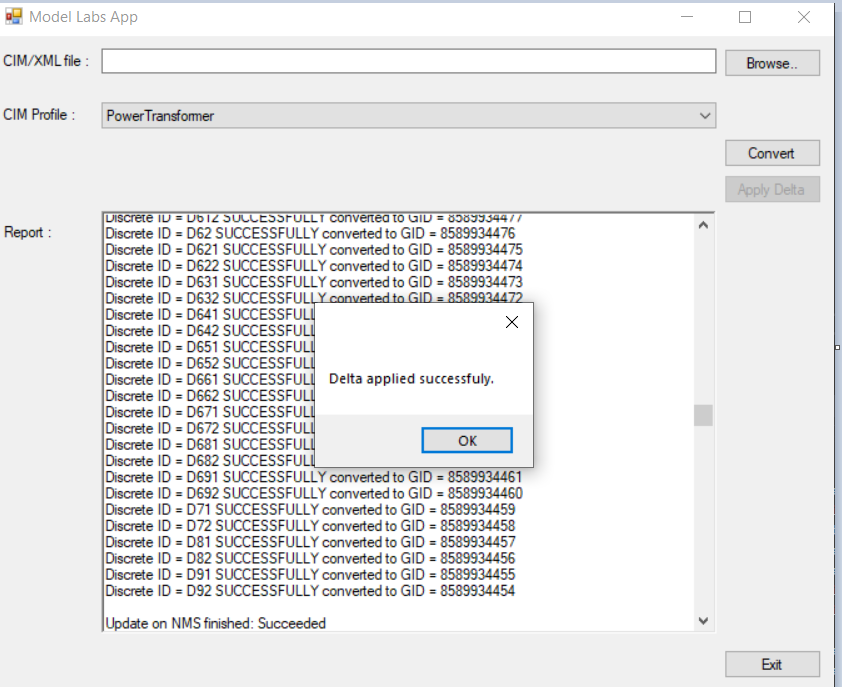
Validacije na NMS-u su izuzetno bitne jer se pomoću njih izbegava primena tzv. “loše delte”, odnosno delte koja ne zadovoljava određene uslove. Da ne postoje validacije ne bi sa sigurnošću mogli da tvrdimo da u svakom trenutku imamo prihvatljivo stanje modela. Delta se u našem sistemu može kreirati na dva načina: pomoću Importera(uvlačenjem odgovarajućeg xml fajla) ili na UI od strane korisnika. Način kreiranja u ovom poglavlju ne igra veliku ulogu, jer se kod oba načina validacija radi na identičan način.

Validacija se radi nad svim afektovanim entitetima(insertovanim ili izmenjenim) pozivanjem metode *Validate()*, poziv ove metode je prikazan na listingu 3.1.1.



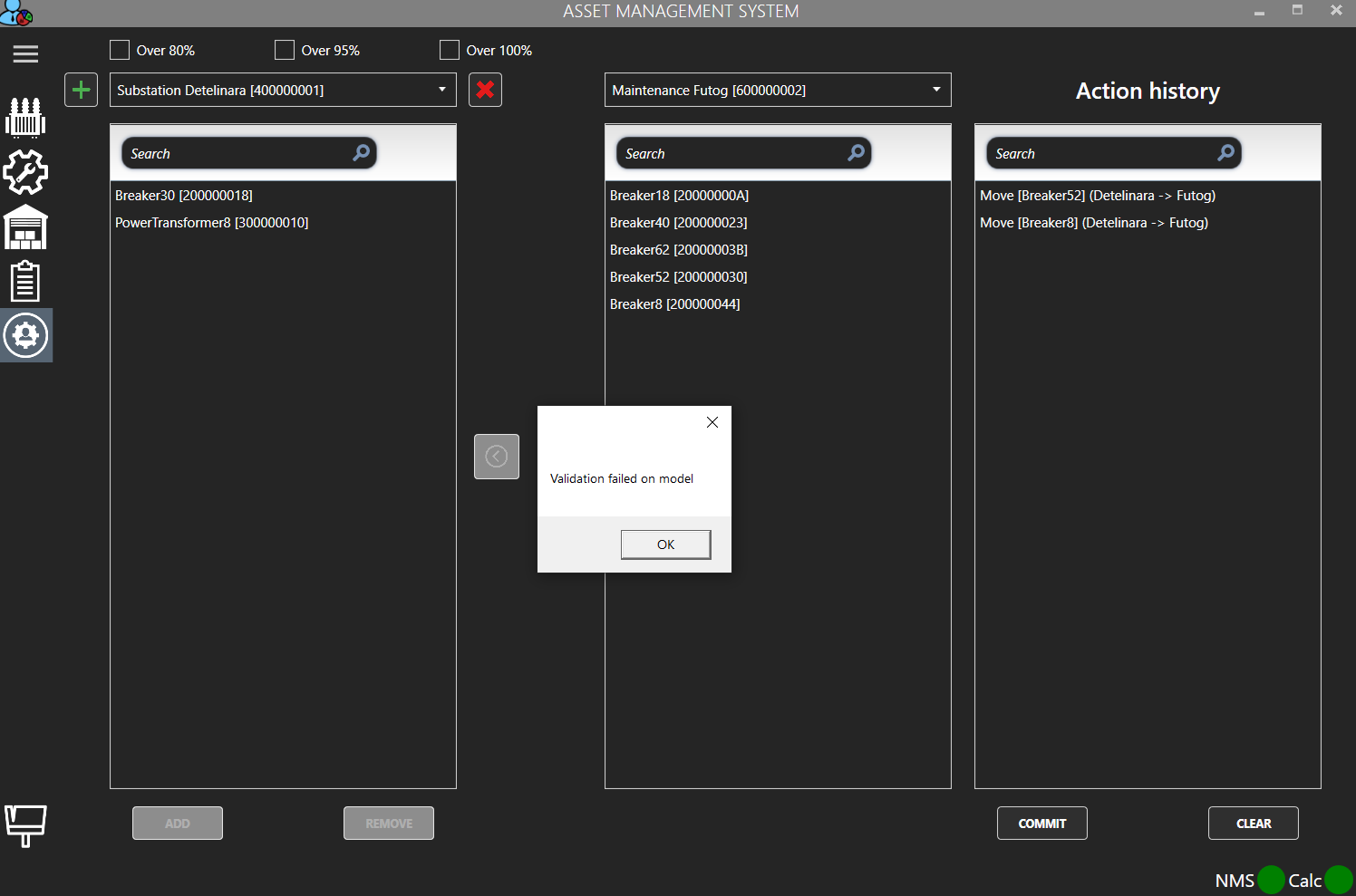
*Listing 3.1.1 - Validacija afektovanih entiteta*

Metoda *Validate()* se implementira u svakoj od klasa iz modela krenuvši od dna proveravajući redom i nadklase. Jedan primer validacije bi bio npr. provera da li su *serviceTime* i *expirationTime* opreme veći od nule,kao i da li je *expirationTime* veći od *serviceTime*. Ukoliko su validacije uspešne može početi proces distribuirane transakcije. Na slici 3.1.1 dat je prikaz uspešne validacije i uspešne primene delte putem Importera.



*Slika 3.1.1 – Uspešna primena delte*

Validacije su prilagodljive zahtevima klijenta. U našem sistemu je realizovana jedna takva “dodatna” validacija. Na početku realizacije dogovoreno je da jedna radna stanica(*Substation*) mora sadržati minimum dva prekidača i jedan transformator kako bi bila ispravna. U tu svrhu je odrađena validacija da se oprema iz određene stanice ne može slati na popravku ukoliko nije ispunjen taj uslov(mora se prethodno dodati nova oprema u stanicu). Primer ovakve validacije sa porukom za korisnika prikazan je na slici 3.1.2.



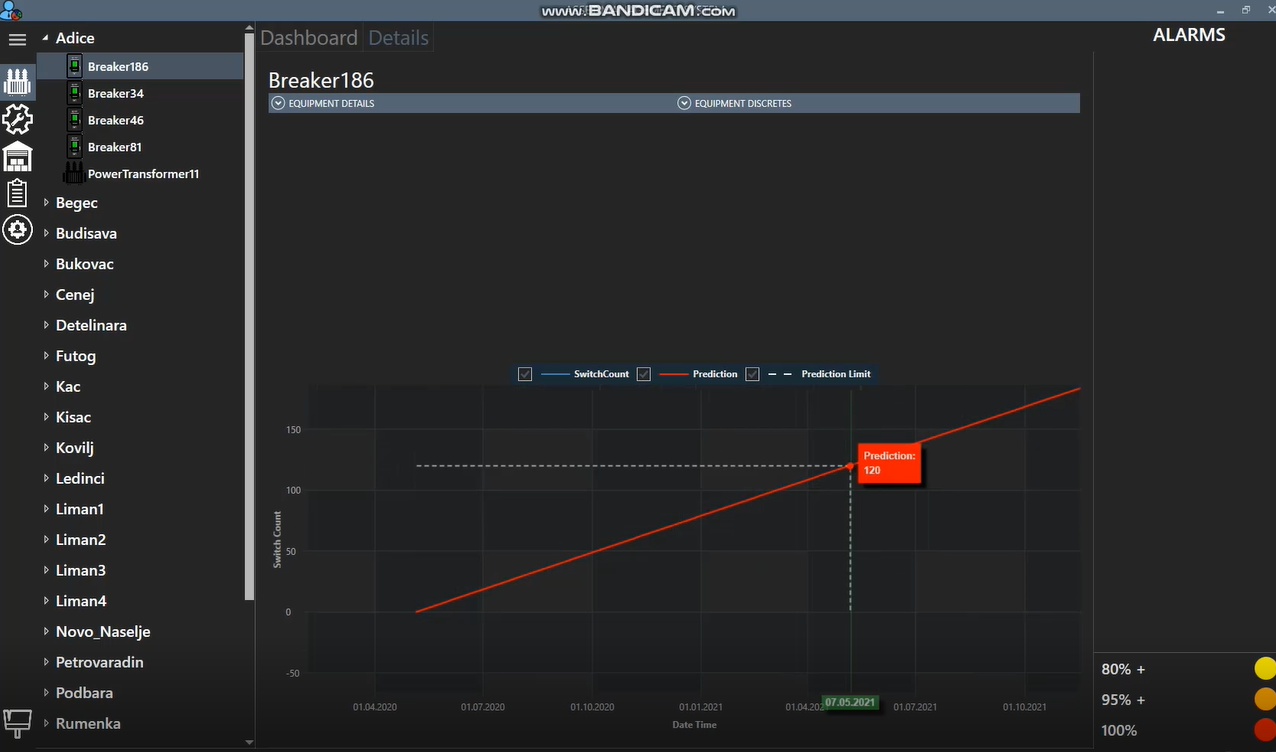
*Slika 3.1.2. – Neuspešna primena delte*

# Napredni racunarski sistemi

## Predikcija i metod najmanjih kvadrata odstupanja

Na *CalculationEngineService-u* u okviru implementirana je klasa *SmallestSquaresRegressionCalculator* koja sadrži metodu za predikciju vremena *PredictNextTime()*, kao i pomoćne funkcije.

Zavisnost znači da vrednost jedne veličine potpuno određuje vrednost druge veličine, odnosno možemo reći da svaka vrednost veličine *x* uslovljava tačno jednu vrednost veličine *y*. To kraće možemo zapisati kao *y = f (x)*, gde *y* predstavlja pravilo ko kome *y* zavisi od *x.* Postoji dosta pravila zavisnosti, ali mi ćemo u našoj aplikaciji iskoristiti linearnu zavisnost, koja se zapisuje kao *y=ax+b*, a njen grafički prikaz jeste prava(slika 4.1.1). S obzirom da se očekuje da naš sistem radi sa dosta podataka(očekuje se i veći broj tačaka), jasno je da je nemoguće povući pravu koja će proći kroz sve tačke. Iz tog razloga koristimo metodu najmanjeg kvadrata odstupanja, uz pomoć koje se određuju parametri *a* i *b* takvi da će odrediti pravu koja će proći što bliže tačkama.

**

*Slika 4.1.1 – Grafički prikaz pravca pre predikcije*

Metoda najmanjih kvadrata odstupanja zasniva se na principu da su najbolji oni parametri *a* i *b* za koje je suma kvadrata razlika između merenih vrednosti *yi* , i = 1,2,…,n i izračunatih vrednosti *f(xi,a,b)* minimalna. Detaljan postupak za određivanje parametara će biti objašnjen u nastavku.

Odstupanje predstavlja razliku između merene vrednosti *y* i izračunatih vrednosti *f(xi,a,b).* Označićemo *i*-to odstupanje kao *Di*, i ono je jednako:

*Di = yi - f(xi,a,b) (1)*

Prema metodi minimalnih kvadrata, parametre određujemo tako da suma *D12 + D22 +…+ Dn2* bude minimalna. Taj izraz zavisi od parametara *a* i *b*, pa možemo reći sledeće:

*F(a,b) = [y1-f(x1,a,b)]2 + [y2-f(x2,a,b)]2 + … + [yn-f(xn,a,b)]2 (2)*

Odnosno kraće zapisano:

*(3)*

Dalje, pomoću parcijalnih izvoda definišu se uslovi lokalnog ekstrema:

= 0 i = 0 ,

Odnosno ukoliko to uvrstimo u nasu funkciju (3) dobijamo sledeće:

*(4)*

*(5)*

Korišćenjem osobina izvoda prethodna dva izraza se mogu zapisati kao:

*(6)*

*(7)*

Nakon daljeg sređivanja oda dva izraza dobijamo:

*(8)*

*(9)*

S obzirom da znamo da je *f(xi,a,b) = axi+b* možemo izračunati parcijalne izvode te funkcije, odnosno izvode po *a* i po *b:*

*(10)*

*(11)*

Sada, ako te izraze ubacimo u *(8)* i *(9)* dobijamo:

*(12)*

*(13)*

Nakon raspisivanja prethodnih izraza dobijamo:

*(14)*

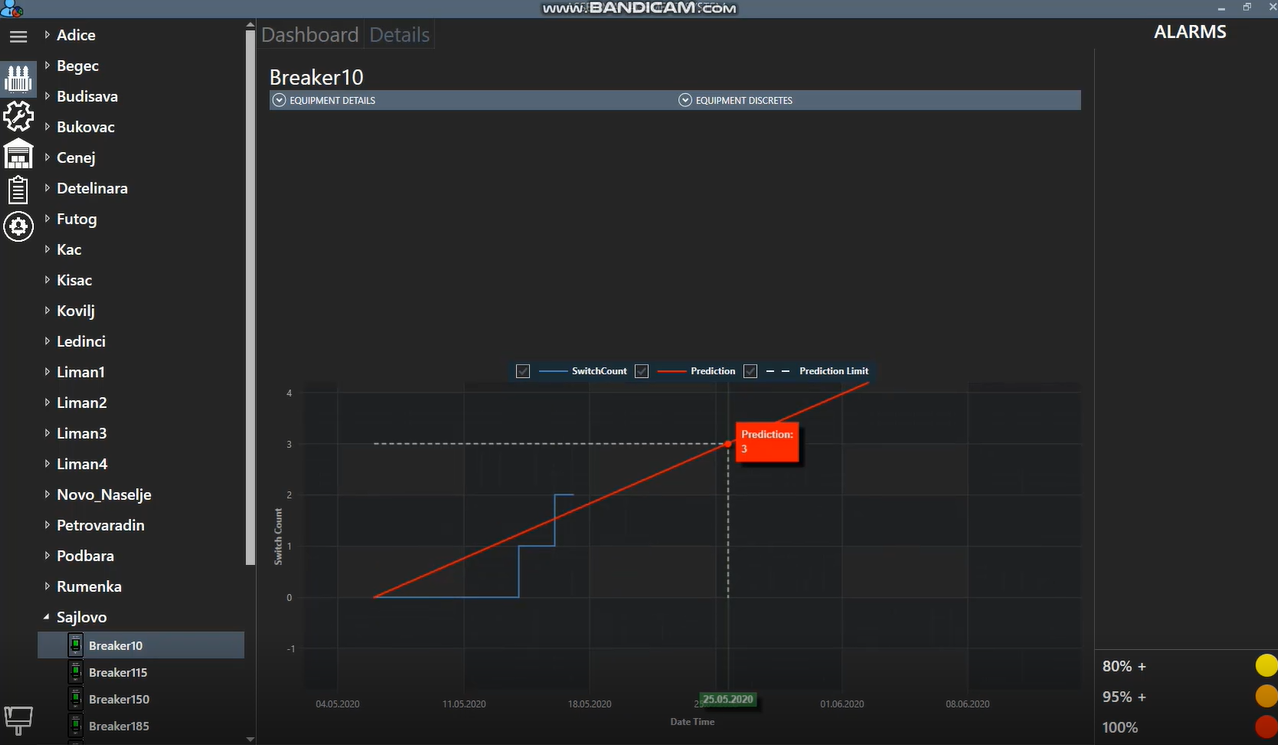
*(15)*

Sredjivanjem ovih izraza konačno možemo izračunati parametre *a* i *b*:

*(16)*

*(17)*

Na osnovnu ovih parametara definišemo pravac koji je određen jednačinom *y = ax+b.*

**

*Slika 4.1.2 – Primenjen metod najmanjih kvadrata odstupanja nad više tačaka*

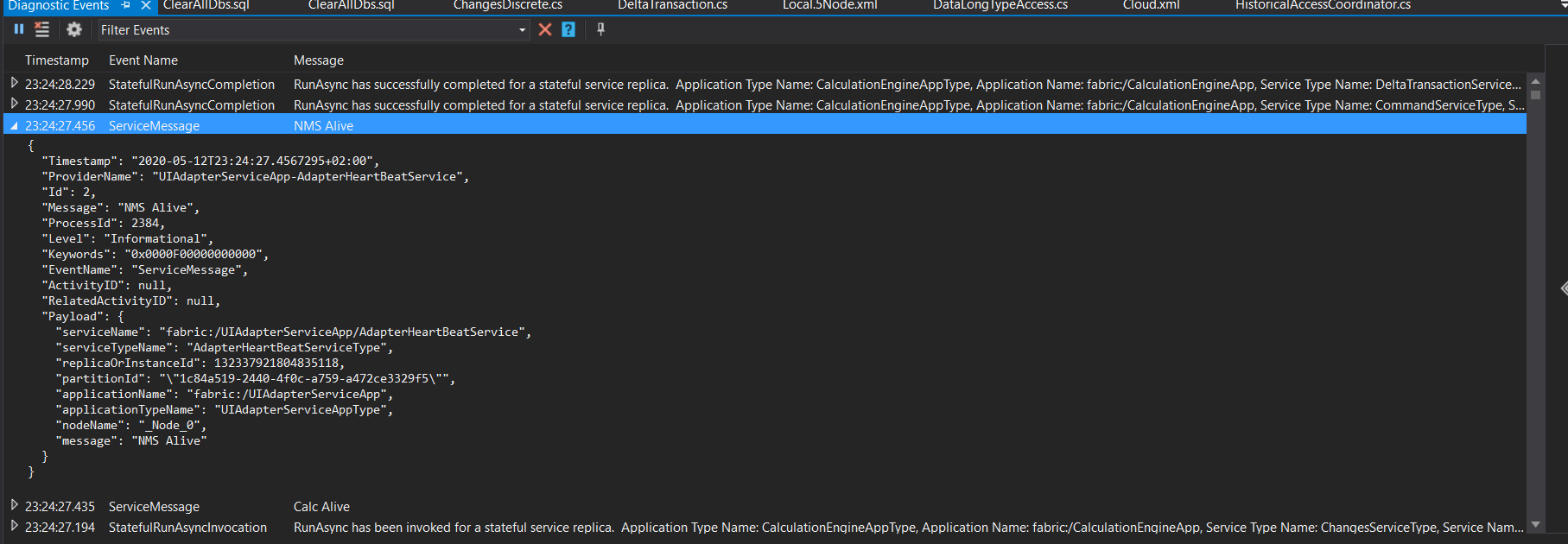
Upotreba metode najmanjih kvadrata odstupanja omogućila nam je efikasnu predikciju kvara opreme u sistemu(prekidača), što je jedna od ključnih funkcionalnosti naše aplikacije. Praćenjem predikcije u mogućnosti smo da efikasno isplaniramo kada ćemo opremu poslati na remont i na taj način možemo znatno smanjiti vreme koje je klijent bio bez napajanja usled kvara zbog prekomernog rada opreme.

# Azure – Service Fabric

## Diagnostic Events

Azure Service Fabric je distribuirana platforma koja olakšava rad sa mikroservisima i kontejnerima. Takođe rešava značajne izazove u razvoju i upravljanju aplikacijama u oblaku. Konkretno kod nas u projektu iskorišćen je ETW(*Event Tracing for Windows*), što je jedna od stvari koje omogućava Service Fabric.

Za praćenje poruka koristi se Diagnostic Events prozor u okviru Visual Studia. Ukoliko se ne otvori odmah pri pokretanju aplikacije moguće ga je ručno otvoriti *View > Other Windows* > *Diagnostic Events.* Prikaz ovog prozora je prikazan na slici 5.1.1.



*Slika 5.1.1 – Prikaz Diagnostic Event prozora*

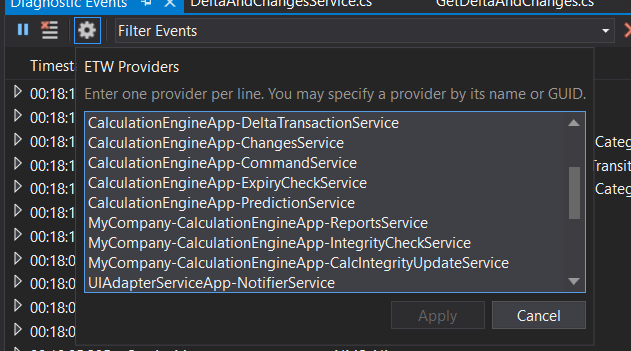
Pomoću *Diagnostic Events* prozora možemo na neki način ispratiti i logove, jer možemo direktno videti kada i kako se nešto u našoj aplikaciji izvršava. Takođe pomoću dijagnostike možemo ispratiti koje funkcionalnosti korisnik najčešće koristi i na osnovu toga možemo isplanirati šta bi i koliko mogli naplaćivati korisniku.

Za ispis poruka potrebno je iskoristiti klasu *ServiceEventSource,* čija je upotreba prikazana na sledećem listingu:



*Listing 5.1.1 - Ispis poruke*

U velikim sistemima očekivano je da se ispisuje veliki broj poruka, samim tim je gotovo nemoguće ispratiti sve od interesa. U tu svrhu može se iskoristiti opcija *Filter Events*. Potrebno je proslediti ime *EventSource*-a koji nas interesuje, što se može videti na dva načina. Prvi je da u okviru *Diagnostic Events* prozora kliknemo na točkić, nakon toga nam se prikazuje prozor sa slike 5.1.2.



*Slika 5.1.2 – prikaz svih EventSource-a*

Jednostavnim kopiranjem određenog imena servisa i pritiskom na *Enter* možemo primeniti filter.

Drugi način je da pogledamo atribut u klasi *ServiceEventSource* servisa koji nas interesuje. Prikaz ovog atributa dat je na listingu 5.1.2.



*Listing 5.1.2 – Atribut za imenovanje EventSource-a*

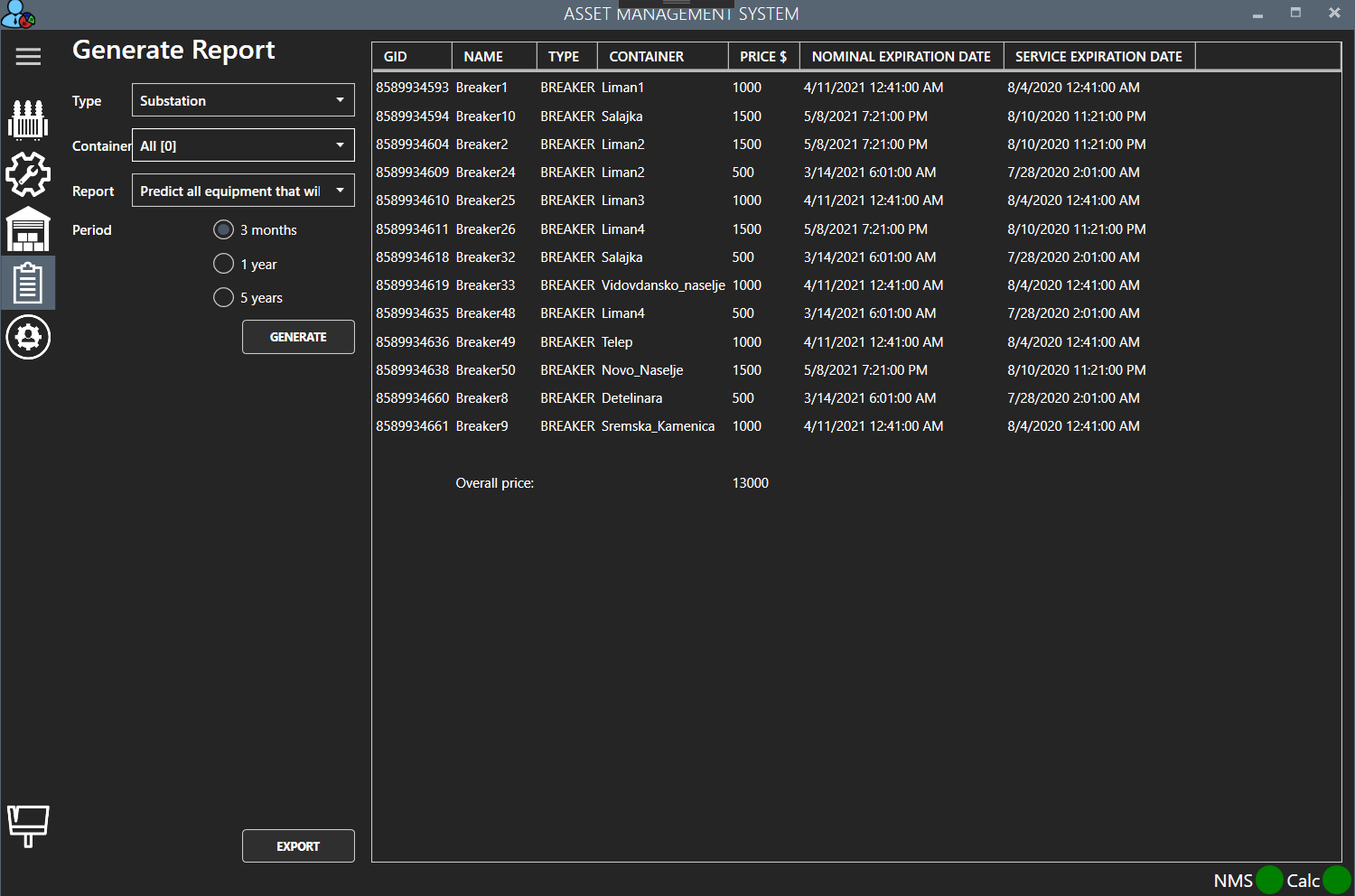
Posmatrano iz poslovnog ugla, korisniku će verovatno često biti potrebna funkcionalnost predikcije vremena kada je potrebno prekidač poslati na popravku. Iz tog razloga ovu funkcionalnost je svakako potrebno dodati na spisak stvari koje ćemo ispratiti, kako bi na osnovu broju poziva mogli detaljno isplanirati kako ćemo korisniku naplatiti.

# Sigurnost i bezbednost(DMS)

## Planiranje remonta na osnovu radnog veka

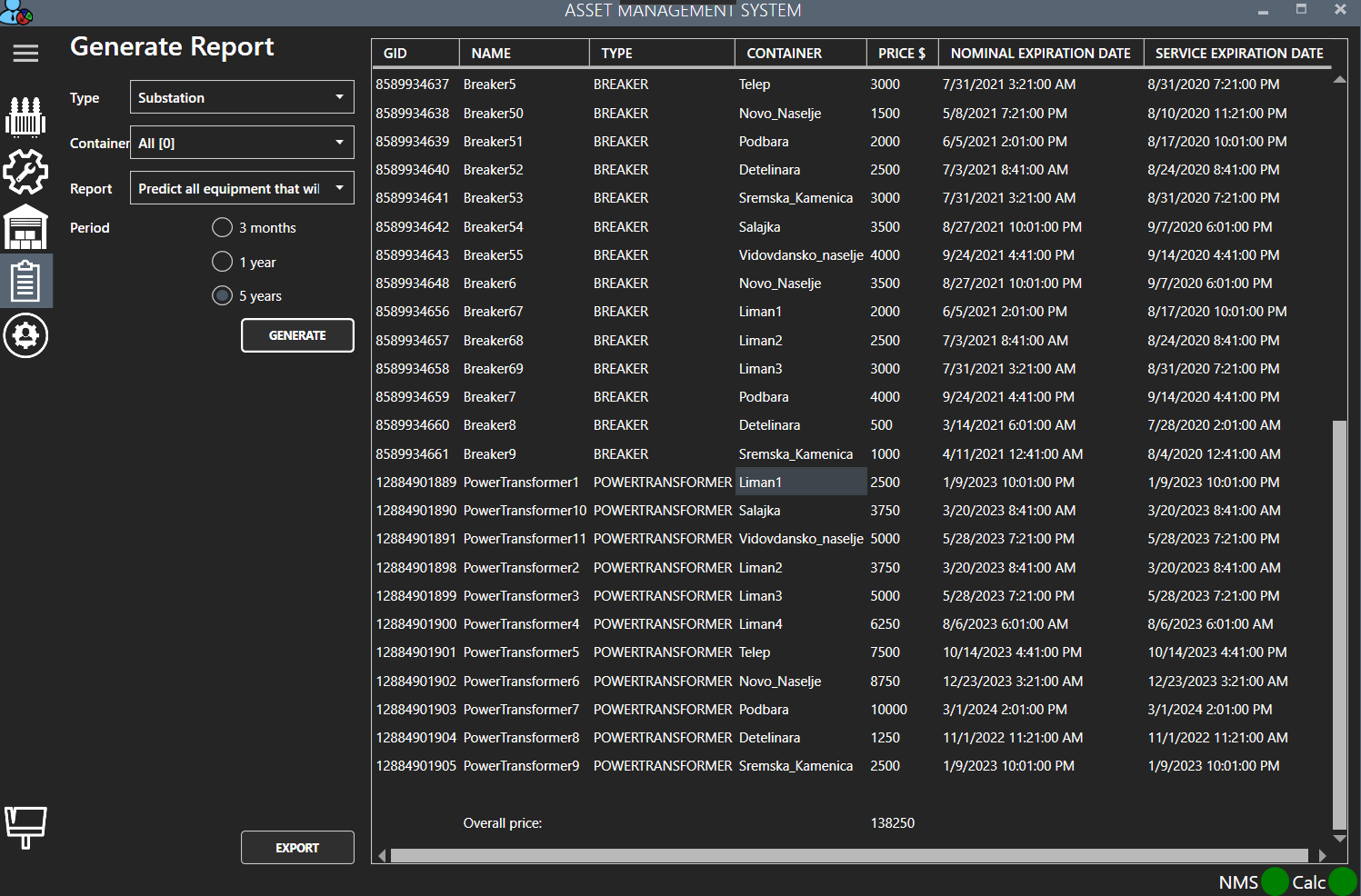
Planiranje remonta igra važnu ulogu u svakom sistemu. Efikasnim planiranjem kod nas se konkretno može poboljšati efikasnost elektroenergetskog sistema. Unapred možemo videti kada koju opremu treba poslati na popravku i na taj način se isplanirati popravku, tako da krajnji korisnik najmanje trpi, odnosno da ima minimalan nestanak napajanja. S druge strane efikasnim planiranjem remonta oprema se šalje na remont u okviru predviđenog radnog veka(ili ranije) i na taj način se znatno smanjuje šansa da dođe do veće havarije ili štete usled prekomernog rada opreme.

Prikaz ovog plana remonta u našoj aplikacija prikazan je u vidu izveštaja. Primer jednog ovakvog izveštaja dat je na slici 6.1.1.



*Slika 6.1.1 – Plan remonta u okviru 3 meseca*

U izveštajima su prikazani: GID opreme, naziv i tip opreme, stanica u kojoj se oprema nalazi, cena, rok trajanja opreme, kao i vreme kada je potrebno opremu poslati na remont. Korisniku je omogućeno da odabere vremenski period za koji želi dobiti izveštaj(3 meseca, 1 godina, 5 godina). Primer izveštaja opreme koju bi trebalo poslati na popravku u narednih 5 godina prikazan je na slici 6.1.2.



*Slika 6.1.2 – Plan remonta u narednih 5 godina*

Ovaj izveštaj(kao i svi ostali) moguće je eksportovati u excel ili .csv fajl radi dodatnih operacija nad podacima.