Benjamin Schelling

Mittauksien ja hälytyksien visualisointi sähkönhallintaverkkosovelluksessa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

17.03.2022

Tiivistelmä

Tekijä:	Benjamin Schelling
Otsikko:	Mittauksien ja hälytyksien visualisointi sähkönhallintaverkkosovelluksessa
Sivumäärä:	31 sivua
Aika:	17.03.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Simo Silander Tuotepäällikkö Matti Kärenlampi

---

Tämän insinöörityön tarkoituksena on määritellä, suunnitella ja toteuttaa konsepti, jossa visualisoidaan mittauksia ja hälytyksiä pohjakartan päällä Hitachi Energyn sähkönhallintajärjestelmää varten. Työssä määritellään SIP (System Implementation Proposal), jonka pohjalta suunnitellaan, miten työ toteutetaan ja lopuksi luodaan tästä prototyyppi.

Verkkosovellukset ovat nykypäivää ja mahdollistavat sovelluksiin pääsyn monilla eri laitteilla. Tämä helpottaa tuotekehitystä, eikä sillä ole tarvetta luoda eri sovelluksia muun muassa mobiilikäyttöön. Mittausten ja hälytysten visualisointi on Hitachi Energy:n verkkosovellukseen tuleva lisäominaisuus, joka toteutetaan React-kirjastolla.

Työssä käydään läpi Hitachi Energyn tuotteita, niiden taustaa ja kuinka ne vaikuttavat kyseiseen insinöörityöhön ja tämän tuloksiin. Lisäksi kurkataan hieman tarkemmin työhön käytettyihin teknologioihin, toteutukseen sekä jatkokehitysideoihin.

Työn lopputuloksena on prototyyppi hälytysten esityksestä karttapohjaisessa verkkosovelluksessa, jonka avulla käyttäjä voi seurata kytkimien ja vianilmaisimien hälytystiloja yhdellä silmäyksellä. Lisäksi työssä tehtiin suunnittelutyö mittauksille mukaan lukien esityssuunnitelma käyttöliittymään. Lopuksi myös havaittiin jatkokehitystoimenpiteitä.

Avainsanat: React.js, Redux, verkkosovellus

Abstract

Author:	Benjamin Schelling
Title:	Visualization of Measurements and Alarms in Electrical Management Web Application
Number of Pages:	31 pages
Date:	17 March 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Simo Silander, Senior Lecturer Matti Kärenlampi, Product Manager

---

The purpose of this thesis is to define the design and implementation of a concept where measurements and alarms are visualized on a topographic base for Hitachi Energy’s electrical management system. During the study, a SIP (System Implementation Proposal) was defined to stand as the basis of the whole process of the implementation and planning. After the SIP, a prototype was created from it.

Nowadays web applications are a hot topic, and they allow access to applications on many different devices, and this facilitates product development as there is no need to create a different application e.g., for mobile use. Visualization of measurements and alarms is an additional feature made for Hitachi Energy's web application, implemented with a JavaScript library called React.

The study reviews Hitachi Energy's products, their background and how they affect this present study and its results. In addition, a closer look is taken at the technologies used here, their implementation and further development ideas.

The results for the study were the visualizations of the alarms in the map-based web application that allows users to see the alarm status of a switch or a fault detector immediately. In addition, design work was done for the measurements including planning for the user interface. Further development measures were also identified.

Keywords: React.js, Redux, Web application

Sisällys

Lyhenteet

AMQP: *Advanced Message Queuing Protocol* on avoin vakiosovellusta-soprotokolla viestikeskeiselle väliohjelmistolle, jonka määrittelemät ominaisuudet ovat viestin suunta, jonotus, reititys, luotettavuus ja suojaus.

API: *Application programming interface* eli ohjelmointirajapinta on raja moduulien tai komponenttien välillä ohjelmoitavassa järjestelmässä. API on määritelmä, jonka mukaan eri ohjelmat voivat keskustella keskenään tekemällä pyyntöjä ja vaihtamalla tietoa.

DMS600: *Distribution Management System* on Hitachi Energyn tuote, jolla voidaan hallita sähköjakelua karttapohjaisella sovelluksella.

DOM: *Document Object Model* on sovellusohjelmointirajapinta (API) HTML- ja XML-dokumenteille. Se määrittää asiakirjojen loogisen rakenteen ja tavan, jolla asiakirjaan päästään käsiksi ja voidaan manipuloida.

NE & WS: *Network Editor* ja *Workstation* ovat osa DMS600-järjestelmää, jolla editoidaan ja hallitaan verkkoa.

OPC: *Open Platform Communication* on joukko tiedonsiirtostandardeja, joka tukee teollisuuden automaatiosovelluksia. Tämän tarkoitus on antaa avointa liitettävyyttä avoimilla standardeilla, jotka määrittävät reaaliaikaisen tuotantotiedonvälityksen eri valmistajien automaatiojärjestelmissä.

ORD: *Object-relational database* eli oliorelaatiotietokanta on järjestelmä, joka on pohjimmiltaan hyvin samanlainen kuin relaatiotietokanta, mutta jossa on oliopohjainen tietokantamalli.

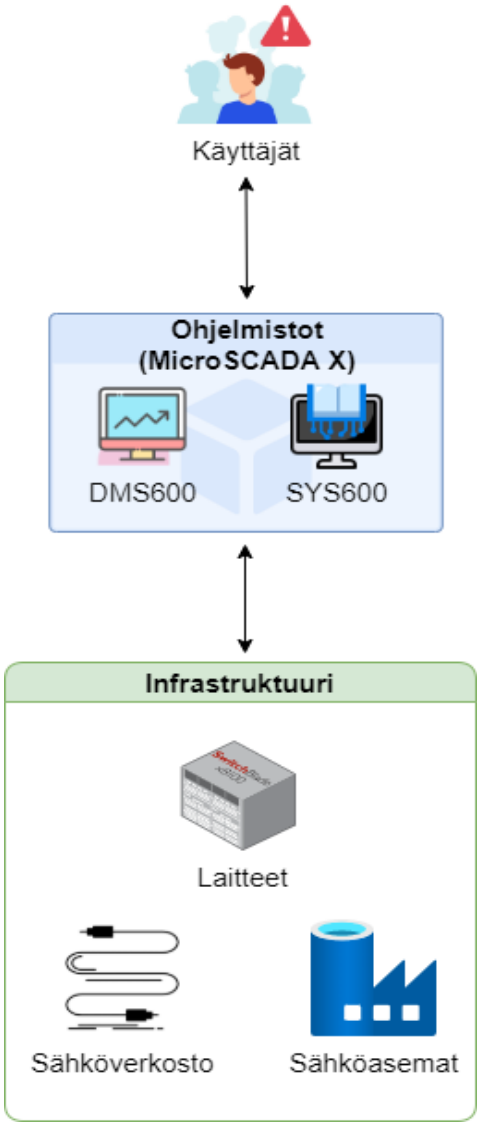
ORDBMS: *Object-relational database management system* on sama kuin ORD.

SYS600: Hitachi Energyn tuote, jolla hallitaan sähköaseman sisällä tapahtuvia kokonaisuuksia.

VCS: *Version Control System* eli versionhallintajärjestelmä hallitsee tiedoston tai tiedostojoukon tehtyjä muutoksia. Nämä muutokset tunnistetaan usein numeroiksi tai kirjainkoodeiksi, joita kutsutaan versionumeroiksi. Jokainen versio liittyy metatietoihin, kuten aikaleima ja tekijä. Versioita voidaan verrata, palauttaa tai yhdistää tarpeen mukaan.

1 Johdanto

Hitachi Energy (aikaisemmin Hitachi ABB Power Grids) on digitaalisten sähköverkkojen edelläkävijä, jonka tarkoituksena on edistää kestävän energian tulevaisuutta kaikille. He kehittävät omalta osaltaan maailman energiajärjestelmää kestävämmäksi, joustavammaksi sekä turvallisemmaksi. Yrityksellä on sähkönhallintajärjestelmä, jolla asiakkaat voivat omien tarpeidensa mukaan seurata ja hallita sähköjärjestelmiään (kuva 1), kuten metroverkoston hallinnointia.



Kuva 1. Hitachi Energyn sähkönhallintajärjestelmän yleiskuva.

Tämä insinöörityö on tehty Hitachi Energylle ja tarkoituksena esitellään kehitettyä prototyyppiä heidän luomaansa verkkosovellukseen. Itse tuote on maantieteellinen jakeluverkon hallintajärjestelmä, johon kuuluvat verkkokomponenttien hallinta ja verkkomallinnus, joka tarjoaa yleiskatsauksen verkosta ja visualisoi topologisen värityksen verkon tilasta. Tuote on suunniteltu auttamaan yrityksiä, joille sähköverkkojen hallinta ja seuraaminen ovat oleellinen osa yrityksen toimintaa, kuten Savon Voima Verkko Oy:llä ja Väylävirastolla (sähkönsyöttöverkko junille).

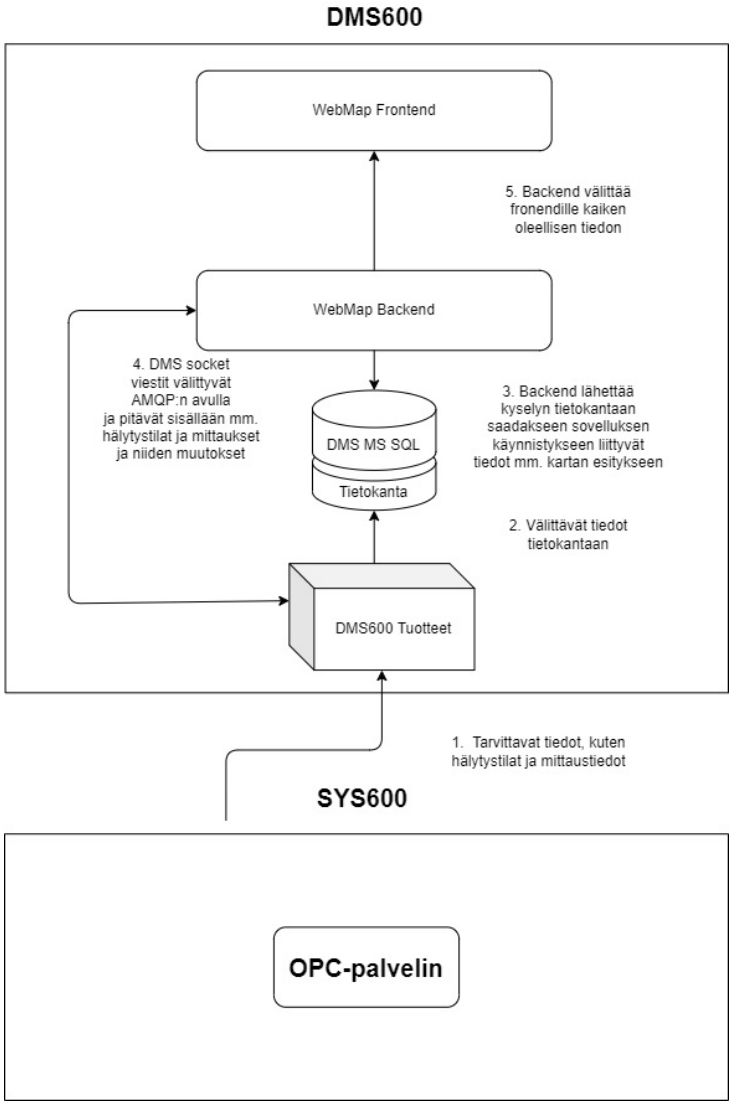
Dokumentissa keskitytään selainpohjaiseen sovellukseen nimeltä WebMap, johon tarkoituksena on visualisoida pohjakartan päälle hälytyksiä ja mittauksia. Lisäksi dokumentissa kuvataan myös projektin työskentelytapaa, kuten versionhallinnan käyttöä sekä tehtyä työtä ja jatkokehitysideoita.

2 Workplace X WebMap -verkkosovellus

MicroSCADA X on Hitachi Energyn tuoteperhe, joka pitää sisällään erilaisia työpöytäsovelluksia ja laitteita sähkönhallintaa varten. Yhtenä tutkimus- ja kehitystiimien tavoitteista on yksinkertaistaa tätä kokonaisuutta ja luoda yksi suuri selainpohjainen sovellus, jossa nämä kaikki eri työpöytäsovellukset kuten DMS600 ja SYS600 tuodaan yhteen. Tässä tulee kehiin Workplace X, joka mahdollistaa tämän kaiken, ja asiakkaille myydään lisenssejä eri sovellusosia varten heidän tarpeidensa mukaan. DMS600 WebMap on tämänhetkinen selainsovellus, joka tuo yhteen DMS600 WS:n (Workstation) ja DMS600 NE:n (Network Editor).

Sähkönhallinta eli sähköverkkojen tilojen seuranta ja säätely ovat olleet jo vuosia elintärkeä toimenpide, ja näiden tarpeellisuus kasvaa jatkuvasti yhä enemmän ja enemmän. Sähköä vaativien välineiden määrä on jyrkässä kasvussa ja niitä löytyy meillä Suomessa likimain kaikilta. Tämän valtavan kokonaisuuden hallintaan, kuten junien tai metrojen rataverkkojen hallinnointiin tarvitaan monenlaisia erilaisia järjestelmiä ja tässä tulee kuvaan Hitachi Energyn Workplace X -verkkosovellus (jonka osana tarkemmin WebMap).

Toteutuksessa ideana (kuva 2) on hakea olemassa olevalta SYS600:n OPC-palvelimelta (palvelin, joka täyttää yhteentoimivuusstandardit) tarvittava data ja lähettää se DMS600:aan, jonka jälkeen sovelluksen käynnistystä varten oleellinen data välitetään MS SQL -tietokantaan ja muun muassa hälytykset ja mittaukset DMS socket -viesteillä suoraan WebMap-backendiin. Tämän jälkeen backend hakee DMS600 SQL Server -tietokannasta oleellisen datan, ja lopulta välittää sen frontendille, jonka jälkeen data visualisoidaan käyttöliittymässä.



Kuva 2. Hälytysten ja mittausten visualisoinnin prosessikaavio.

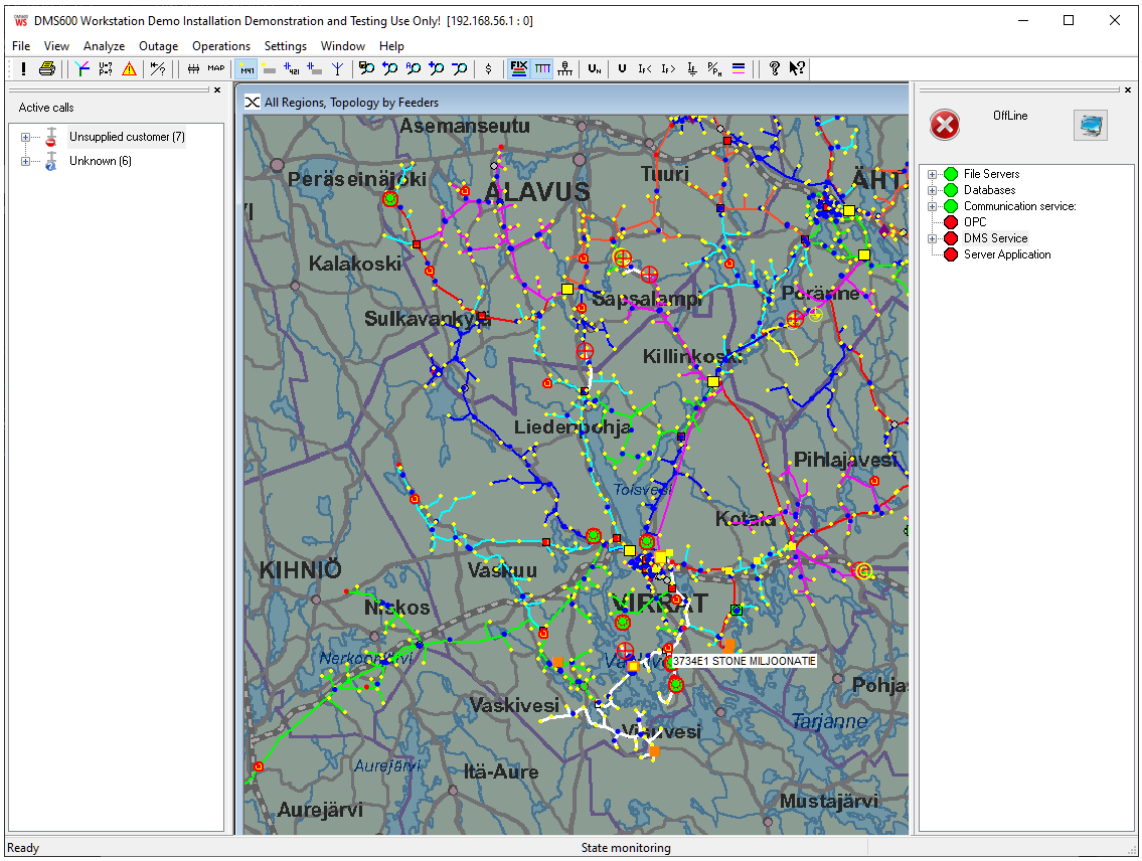
2.1 Työn tavoite ja tausta

Insinöörityössä on tavoite luoda toimiva visuaalinen esittely SYS600:n hälytyksistä ja mittauksista kartan päällä selainpohjaisessa sovelluksessa (kuvat 5–8). Tähän kokonaisuuteen kuuluvat määrittely, suunnittelu ja toteuttaminen. SYS600 toimii tyylipohjana, miten hälytykset ja mittaukset tulisi suurin piirtein näyttämään, sillä tarkoituksena on yhtenäistää DMS600- ja SYS600-tuotteet Workplace X:ssä.

2.1.1 DMS600

DMS600 on maantieteellinen jakeluverkon hallintajärjestelmä (kuva 3), joka on syvästi integroitu SYS600:aan. Ohjelmisto laajentaa perinteisiä sähkönhallinta-ominaisuuksia tarjoamalla maantieteellisesti perustuvia verkkonäkymiä. DMS600 sisältää verkkokomponenttien tietojen hallintaa ja verkon mallinnusta, joiden avulla saadaan yleiskuva verkosta ja sen tiloista. Ohjelmistossa on lisäksi monia valinnaisia moduuleja edistyneillä toiminnoilla.

DMS600 on suunniteltu auttamaan sähköyhtiöiden operatiivista henkilöstöä verkkojensa valvonnassa ja käytössä. Verkkotiedot ja taustakartat ovat käyttäjän määritettävissä ja lisäksi niissä on mahdollista käyttää useita erilaisia koordinaattijärjestelmiä. [1.]



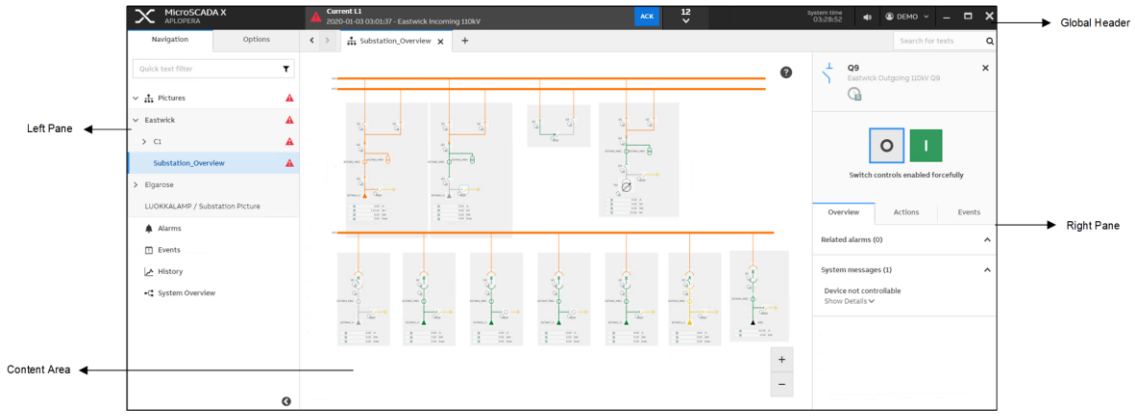
Kuva 3. DMS600 WS:n käyttöliittymä.

2.1.2 SYS600

SYS600 on modulaarinen ja skaalautuva automaatiotuote (kuva 4). Se on jäsennelty yleiseen sovelluksesta riippumattomaan alustaan ja sovellustoimintoon. SYS600 on suunniteltu pääasiassa sähköasemien automaatio- ja verkonhallintasovelluksiin, kuten DMS600:n. Yksi järjestelmän vahvuuksista on skaalautuvuus kapasiteetin, suorituskyvyn sekä toiminnallisuuden suhteen ja tämä mahdollistaa sopivan ratkaisun jokaiseen tarpeeseen. [2.]

SYS600 kerää tietoa prosesseista (yleisimmin sähköstä) eri tietoliikenneprotokollilla ja näitä ovat muun muassa mittaukset tai kytkinlaitteiden tilatiedot. Kyseisiä tietoja voidaan sitten esittää SYS600:n käyttöliittymän prosessikuvissa ja tietojen pohjalta voidaan luoda hälytyksiä ja tapahtumia tutkintaa sekä kontrolloin-

tia varten. DMS600 hyödyntää näitä tietoja ja esittää niitä käytännössä laajemmin, sillä tämä sisältää sähköasemien ulkopuolisia laitteita, joita SYS600 taas ei sisällä. SYS600:aa käytetään lyhyesti yleisimmin sähköasemien valvontaan ja hallintaan, kun taas DMS600:aa sähköverkkojen.

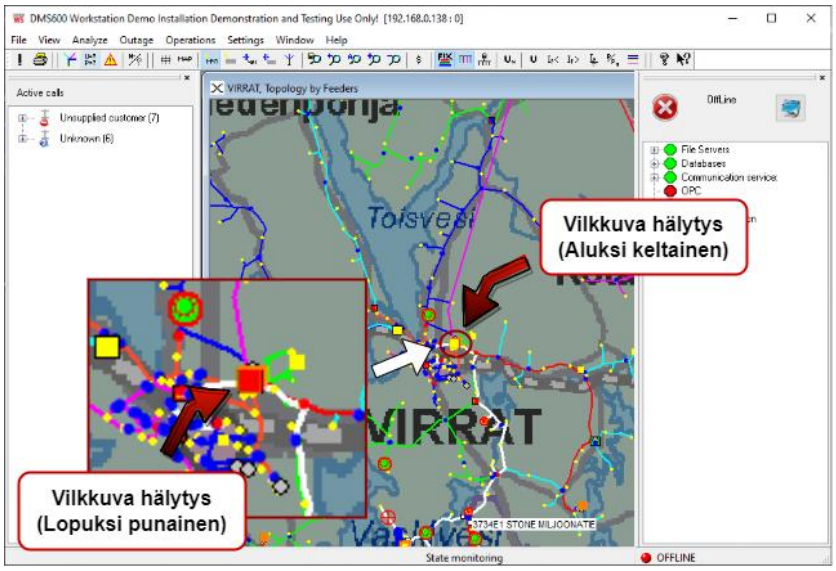


Kuva 4. SYS600-käyttöliittymä Workplace X:ssä

2.2 Työn suunnitelma

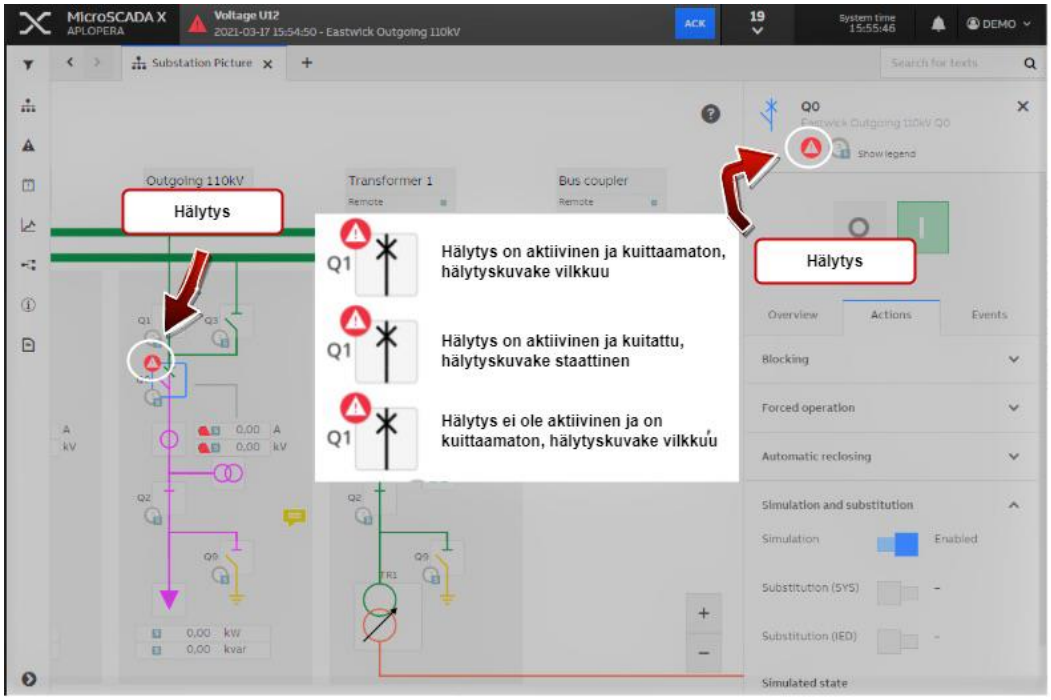
2.2.1 Hälytysten visualisoinnin suunnitelma

Hälytysten ensisijainen tarkoitus on ilmaista hälytettävä tilanne kytkimissä (sijaitsevat sähköverkostossa ja sähköasemissa), vianilmaisimissa sekä mittauksissa. Tähän käytetään pohjana DMS600 WS -sovelluksen periaatetta, jossa hälytys ilmaistaan puna-keltaisena vilkkuna sähköaseman päällä (kuva 5).



Kuva 5. Hälytysten visualisointi DMS600 WS -sovelluksessa, jossa vilkkuva komponentti kuvaa hälytystilaa.

SYS600:ssa on käytössä varoituskolmio (kuva 6), joka tulee olemaan Web-Map:ssa esitettävä komponentti kartassa. Varoituskolmio reagoi erilaisiin hälytettäviin tilanteisiin eri tavoin, kuten vakavimmillaan varoituskolmio välkkyy.

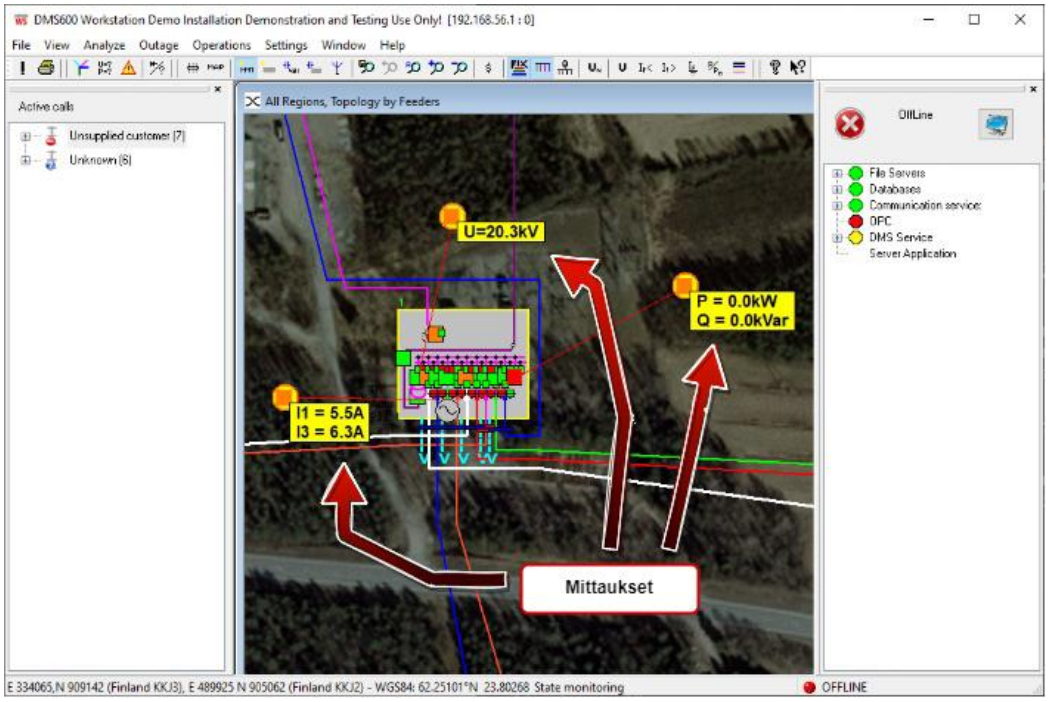


Kuva 6. Hälytykset SYS600:ssa on ilmaistu varoituskolmioilla.

WebMap:ssä on tarkoitus tehdä DMS600:sta ja SYS600:sta hyvin samannäköisiä. Tämä on syy, miksi suunnitelmana on tuoda varoituskolmio karttaan vilkkuvan laatikon sijaan. Yhtenäisen tuotteen luonti parantaa huomattavasti käyttäjäkokemusta sekä käyttöliittymässä navigointia ja sovelluksen nopeaa omaksumista.

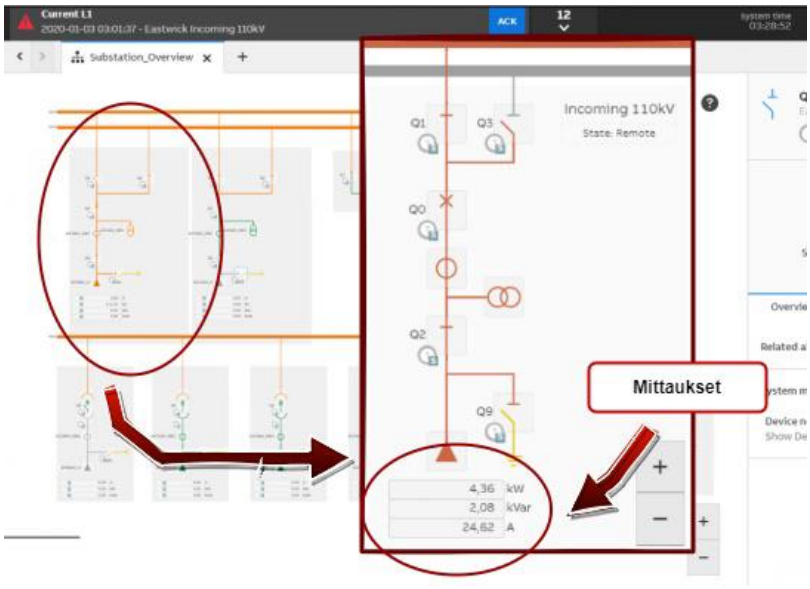
2.2.2 Mittausten visualisoinnin suunnitelma

Mittaukset esitetään DMS600:ssa keltaisissa laatikoissa (kuva 7). Mittausten visualisoinnin tarkoituksena on nopeasti näyttää sähköaseman mittaustiedot päältäpäin. Keltaisen värin etu on tämän erottuvuus pohjakartasta, joka tilanteen mukaan voi olla hyvä. Uudessa WebMap:ssä on koottu kaikki tärkeä ja oleellinen tieto sivupaneeliin, johon on helppo navigoida vain yhdellä painalluksella.



Kuva 7. Mittaukset visualisoituna DMS600 WS -sovelluksessa.

SYS600:ssa mittaukset on esitetty harmaissa laatikoissa (kuva 8), ja tämä tulee olemaan myös WebMap:ssä esitystapa. Mittauksien sijainti on määritelty sähköaseman sisään, jossa se indikoi kunkin sähkölinjan mittaustiedot, kuten virran määrän.



Kuva 8. Mittaukset esitettynä SYS600:ssa harmaissa laatikoissa.

3 Käytetyt teknologiat

3.1 Frontend

3.1.1 React.js

React on ilmainen avoimen lähdekoodin deklaratiivinen JavaScript-kirjasto, joka on tehty Facebookin (nykyisin Meta) toimesta käyttöliittymien rakentamiseen. Sen avulla voidaan luoda monimutkaisia käyttöliittymiä pienistä ja eristetyistä koodin osista, joita kutsutaan komponenteiksi. [3.] Reactia käytetään SPA-tarkoitukseen (single-page application), jossa kaikki toiminnallisuudet ja tiedot ladataan kerralla selaimeen, mutta ei välttämättä näytetä kerralla ja tämä nopeuttaa sivuston navigointia MPA:han (multi-page application) verrattuna huomattavasti.

React käsittelee lähinnä vain tilanhallintaa ja sen näyttämistä DOM:lle (dokumenttioliomalli). Tämän takia React-sovellukset vaativat usein lisäkirjastoja, joiden avulla toteutetaan muita toimenpiteitä, kuten tyylitystä tai reititystä. Suosituimpia komponenttikirjastoja ovat mm. Material UI ja React-Bootstrap, joiden avulla voidaan nopeasti implementoida erittäin muokattavia komponentteja, kuten painikkeita käyttöliittymässä.

Reactissa, kuten aikaisemmin mainittiin, manipuloidaan DOM:ia, jonka tarkoitus on kuvata dokumentin rakenne, kuten HTML- tai XML-puuna. Näistä sitten haetaan, tutkitaan ja manipuloidaan eri olioita, ja tämä onnistuu juuri esimerkiksi Reactilla. DOM-manipulaatio on hitaampaa kuin React-toiminnot ja tässä Reactin VDOM (virtuaalinen dokumenttioliomalli) tulee kehiin. Sen avulla voidaan nopeasti päivittää verkkosovelluksen näkymää ja ideana on kevyt kopio oikeasta DOM:sta, josta puuttuu kyky muuttaa suoraan näytön sisältö. Kun VDOM on päivitetty, React vertaa tämänhetkistä VDOM-tilannetta edeltävään VDOM:n tilannekuvaan ja päivittää vain vaihtuvan komponentin näkymässä sen sijaan, että esimerkiksi koko sivusto tai luettelo ladataan uudestaan. [4.]

```
/* status-objektiargumentti sisältää hälytysten tilat
VALID_AND_UNACKNOWLEDGED, INVALID_AND_UNACKNOWLEDGED,
VALID_AND_ACKNOWLEDGED, WARNING ja NO_ALARM */
export const Alarms = ({status}) => {

   // Ikonit
   const alarmSymbol = 'alarm_symbol.svg';
   const alarmAcknowledged = 'alarm_acknowledged.svg';
   const warning = 'warning.svg';
   const station = 'station.svg';

   const [type, setType] = useState(status);

   const handleClick = () => {
      setTypes("VALID_AND_ACKNOWLEDGED");
   };

   const alarmIcons = (type) => {
      switch (type) {
         case 'VALID_AND_UNACKNOWLEDGED':
            return alarmSymbol;
         case 'INVALID_AND_UNACKNOWLEDGED':
            return alarmSymbol;
         case 'VALID_AND_ACKNOWLEDGED'
            return alarmAcknowledged;
         case 'WARNING'
            return warning;
         case 'NO_ALARM':
            return station;
         default:
            return station;
      }
   }

   return (
     <div className='alarm'>
        {alarmIcons(type)}
        {type === "VALID_AND_UNACKNOWLWEDGED" ? (
           <button className="alarm-acknowledge" onClick={handleClick}>
              Acknowledge
           </button>
        ) : null}
     </div>
   );
}
```

Esimerkkikoodi 1. Hälytyksen esitys tämän tilan mukaan ja sen kuittaus.

Esimerkkikoodin 1 yksinkertaisessa funktiossa on tarkoituksena näyttää eri sähköasemaan liittyviä ikoneita tämän tilan mukaan. Kyseisessä esimerkissä on hälytystilat *VALID_AND_UNACKNOWLEDGED*, *INVALID_AND_UNACKNOWLEDGED*, *VALID_AND_ACKNOWLEDGED*, *WARNING* ja *NO_ALARM*, jotka esittävät kunkin tilan mukaan oman ikoninsa. Nämä hälytystilat välittyvät

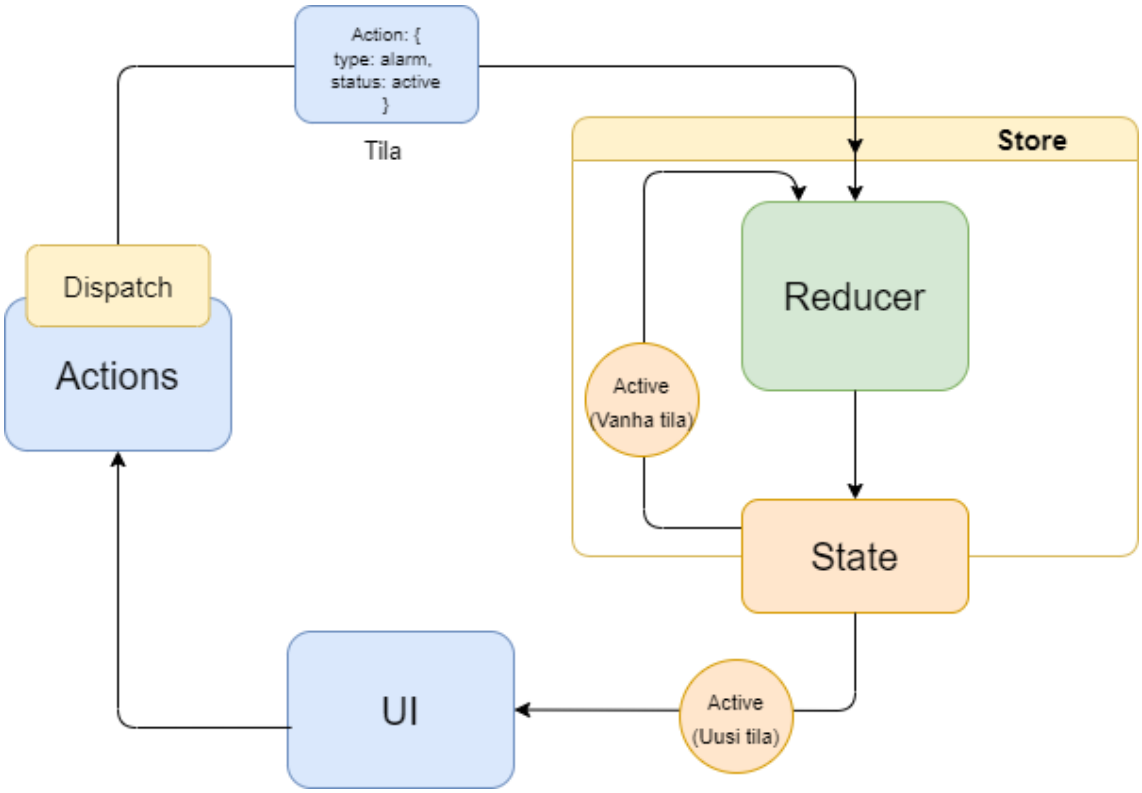
backendistä *status*-objektiargumentin (*properties*) avulla ja tämän objektiargumentin tila käsitellään *Redux*-kirjastoa käyttäen, ja se päivittyy automaattisesti DMS socket -viestien avulla.

Reactissa on koukkuja (*hooks*) ja yksi näistä on *useState* [5.], jolla voidaan hoitaa tilakäsittelyitä. Esimerkissä tarkoituksena on luoda muuttuja *type*, jolle voidaan asettaa *setType*:lla uusi arvo. Tässä tilanteessa *setType*:lla kuitataan hälytys, jonka jälkeen hälytyksen tila saa arvon *VALID_AND_AKNOWLEDGED* ja esittää tätä vastaavan ikonin käyttöliittymässä. *UseState* toimii siten, että ensimmäisenä parametrina annetaan alkutila ja toisena funktio, jolla tätä tilaa päivitetään.

3.1.2 Redux

Redux on kirjasto sovelluksien tilojen hallintaan ja päivittämiseen actioniksi kutsuttujen tapahtumien avulla. Se toimii keskitettynä tilavarastona, jota on käytettävä koko sovelluksessa ja tämän säännöillä varmistetaan, että tila voidaan päivittää vain silloin kuin halutaan. Reduxin tarjoamat mallit ja työkalut helpottavat sen ymmärtämistä, milloin, missä, miksi ja miten sovelluksen tilaa päivitetään ja kuinka sovelluslogiikka käyttäytyy näiden muutosten tapahtuessa. Redux auttaa käsittelemään jaettuja tiloja ja yleisesti sen hyödyt näkyvät vain, jos on suuria määriä sovellustiloja monissa eri paikoissa sovelluksessa. [6.]

WebMap:ssä sovellustiloja on hyvin monia ja tämä osoittautuu hälytyksille ja mittauksillekin lähestymistavaksi tilojen hallintaan. Hälytysten tilanteessa *state* pitää sisällään tiedon hälytyksestä, ja UI renderöi tämän alkutilan. Muutostilanteessa välittyy UI-muutos käsittelijälle, jossa lähetetään *action*, joka saapuu *store*en, jossa *reducer* käsittelee tämän ja välittää uuden tilan *statelle*. *Reducer* tarkastaa, onko tila muuttunut, ja *state* pitää muistissa edeltävän tilan. Joka kerta, kun uusi tila saapuu *storeen*, *state* palauttaa *reducerille* vanhan tilan. Jos tila on muuttunut *reducerin* vertailussa, tämä välittyy *staten* kautta UI:lle (kuva 9).



Kuva 9. Hälytyksen tilan käsittely Redux-kirjastoa hyödyntäen.

Action, jonka perusteella *storeen* lähetetään hälytystiedot, pitää sisällään tyypin ja datan. Tämän saapuessa *storeen*, *reducer* tekee tehtävänsä *staten* ja *actionin* avulla ja välittää *staten* kautta muutokset käyttöliittymälle. *Selectorin* avulla voimme sitten päästä kyseiseen hälytystietoon käsiksi tarpeen mukaan ja välittää tätä tietoa eri osiin käyttöliittymässä. Hälytystilan hallinta käyttäen Reduxia koostuu *Actionista*, *Statesta*, *Reducerista* ja *Selectorista* (esimerkkikoodi 2).

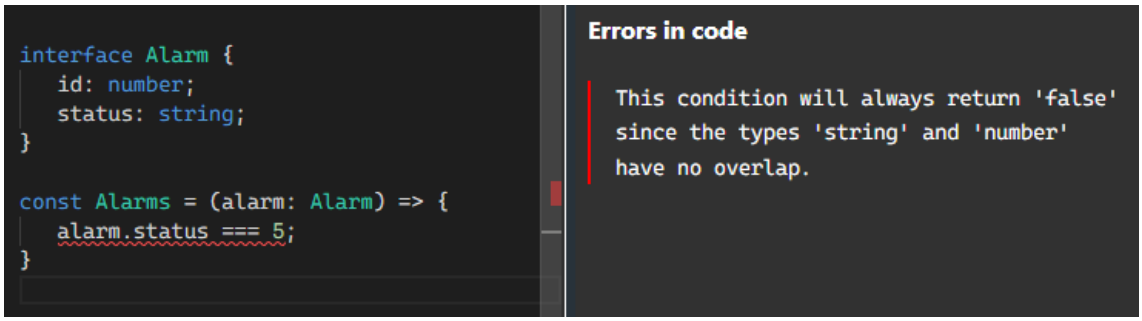
```
// Action
export interface AlarmStateRefreshAction {
     type: ActionTypes.ALARM_STATE;
     data: AlarmStateUpdateResponse;
}

// State
export interface AlarmState {
     id: number;
     status: AlarmStateType;
}


// Reducer
export const alarmReducer = (state: AlarmState[], action: Action):
AlarmState[] => {
     if (action.type === ActionTypes.ALARM_STATE) {
          if (action.data.alarm_states) {
               action.data.alarm_states.alarm_state_changes
               .forEach(alarm => {
                    state.push({
                         id: toNumber(alarm.change_id),
                         status: alarm.new_state
                    });
               });
          }
     }
     return state;
}

// Selector
export const alarmSelector = (state: State): AlarmState[] => {
     return state.alarms;
}
```

Esimerkkikoodi 2: Reduxin tilanhallintakomponentit.

3.1.3 TypeScript

TypeScript on pohjimmiltaan kuten JavaScript. Poikkeava ero on, että TypeScript, kuten nimestä voi päätellä, lisää JavaScriptiin tyypitykset, joiden avulla voidaan helpottaa koodin luettavuutta ja vähentää virhetilanteita. [7.]

Projektissa TypeScriptin tarkoitus on lisätä tyypitykset Reactiin, jotta vältetään turhia virheitä sekä ohjelmoitaessa voidaan ne kaapata heti. Hälytyksellä on tila *status*, jolla on eri merkkijonoarvoja. Tämä merkkijono on TypeScriptin antama tyypitys muuttujalle ja sen ansioista *status* ei voi ottaa muuta tyyppiä vastaan. Jos muuttujalle määrätään numero, palauttaa konsoli virheilmoituksen (kuva 10).



Kuva 10. TypeScript-tyypityksen tuottama virheilmoitus, kun vääränlainen arvo syötetty muuttujalle.

```
interface Alarm {
  id: number;
  status: string;  // status sisältää hälytysten tilat

}

export const Alarms(alarm: Alarm) {
   const alarmSymbol = 'alarm_symbol.svg';
   const alarmAcknowledged = 'alarm_acknowledged.svg';
   const warning = 'warning.svg';
   const station = 'station.svg';

  const [type, setType] = useState(alarm.status);
  const handleClick = () => {
    setType("VALID_AND_ACKNOWLEDGED");
  };

  const alarmIcons = (type: string) => {
    switch (type) {
         case 'VALID_AND_UNACKNOWLEDGED':
            return alarmSymbol;
         case 'INVALID_AND_UNACKNOWLEDGED':
            return alarmSymbol;
         case 'VALID_AND_ACKNOWLEDGED'
            return alarmAcknowledged;
         case 'WARNING'
            return warning;
         case 'NO_ALARM':
            return station;
         default:
            return station;
    }
  };

  return (
    <div className="App">
      {alarmIcons(type)}
      {type === "VALID_AND_UNACKNOWLEDGED" ? (
        <button className="alarm-acknowledge" onClick={handleClick}>
          Acknowledge
        </button>
      ) : null}
    </div>
  );
```

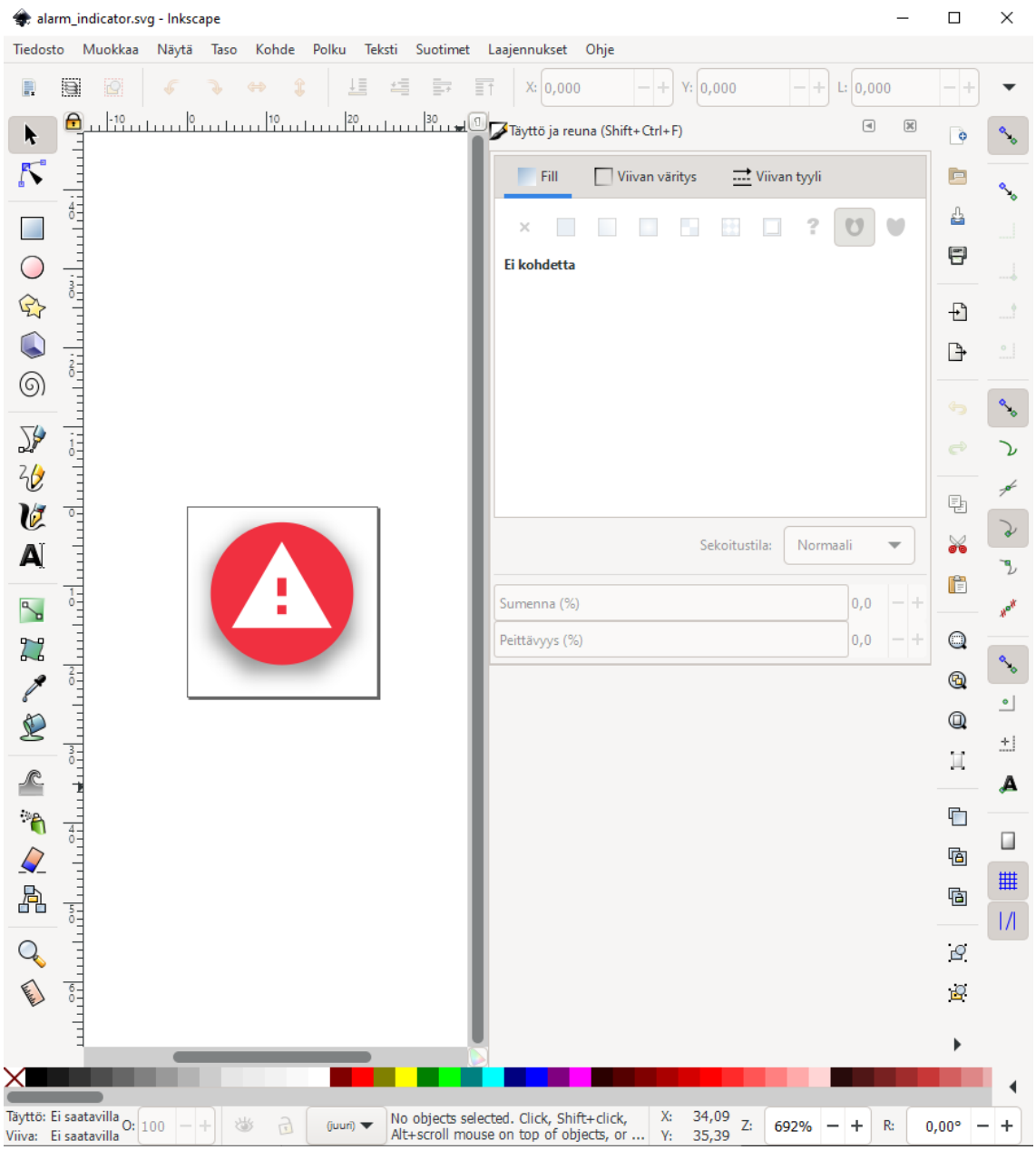
Esimerkkikoodi 3. Hälytyksien tyypitykset TypeScriptin avulla.

Esimerkkikoodin 3 funktio esittää saman tilanteen kuin ensimmäinen React-esimerkki, mutta tällä kertaa TypeScript-tyypitykset lisättynä. Tällä kertaa näemme *Alarm*-rajapinnan, jossa *status*-muuttuja on määritelty merkkijonotyyppiseksi. Objektiargumenttina annetaan sitten tämä rajapinta muuttujana, joka mahdollistaa rajapinnan käytön ja tämän arvojen saannin *Alarms*-funktioon.

3.1.4 Inkscape

Inkscape on ilmainen avoimen lähdekoodin vektorigrafiikkaeditori, jonka avulla voidaan kirjoittaa ja muokata kuvia interaktiivisesti ja tallentaa näitä mm. SVG- tai PNG-tiedostomuotoon. Vektorigrafiikan avulla voidaan luoda kuvitusta, kuten ikoneita, joissa pikselimäärä ei ole kiinteä eli ikonia voidaan venyttää vaikka kuinka paljon ja kuvanlaatu pysyy tarkkana. Inkscapessa on kattavat ominaisuudet, yksinkertainen käyttöliittymä ja on helposti laajennettava lisäosilla. [8.]

Tässä projektissa luotiin hälytysten ikonit käyttäen Inkscapea (kuva 11). Hälytykset perustuvat SYS600:n esitysmuotoon ja ovat kooltaan 24x24 px (pikseliä). Ikonit itsessään ovat hyvin yksinkertaisia ja näyttäviä, jotta ne erottuvat käyttöliittymästä hyvin ja nopeasti. Ikonien luontiprosessi on melko suoraviivainen. Ensin määritellään koko, jonka jälkeen Inkscape antaa raamit, joiden sisällä työskennellään. Raamien sisälle tässä tilanteessa luodaan ympyrä, joka värjätään punaiseksi ja tälle lisätään varjoa, joka luo hieman kolmiulotteista tuntumaa ikonille ja helpottaa tämän erottamista taustasta. Tämän jälkeen luodaan valkoinen kolmio ja tämän sisälle leikataan suorakulmio ja neliö huutomerkin esittämiseen.



Kuva 11. Hälytysikonin luonti.

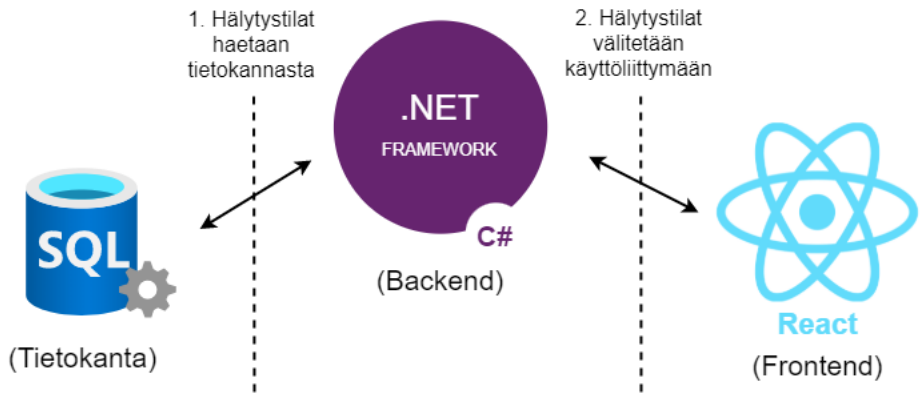
3.2 Backend

C# (C-Sharp) on Microsoftin luoma moderni, oliosuuntautunut ja tyyppiturvallinen ohjelmointikieli, jonka avulla voidaan rakentaa turvallisia ja kestäviä sovelluksia, jotka toimivat pääosin myös Microsoftin kehittämässä .NET:ssä (ohjelmistokomponenttikirjasto). Sen juuret tulevat C-kieliperheestä ja on ymmärrettä-

vissä C-, C++-, Java- ja JavaScript-ohjelmoijille. C# on hyvin tuettu ohjelmointikieli, joka on erittäin skaalautuva ja helposti ylläpidettävissä ja tästä syystä hyvin suosittu sekä pidetty kieli. Lisäksi sitä käytetään usein ammattimaisten sekä dynaamisten verkkosivujen kehittämiseen .NET-alustalla ja on tästä syystä valittu myös WebMap:in kehitykseen. [9.]

.NET on avoimen lähdekoodin kehittäjäalusta, joka toimii lähes kaikilla alustoilla ja on hyvin monipuolinen, sillä sitä voidaan käyttää lähes mihin vain, kuten työpöytäsovelluksiin ja verkkosovelluksiin. Se koostuu kehittäjätyökaluista, ohjelmointikielistä sekä eri sovelluksien rakentamiseen tarkoitetuista kirjastoista. [10.]

WebMap:ssä .NET toimii frontendin palvelimena, joka on kirjoitettu C#:lla. Tämä keskustelee tietokannan kanssa AMQP-viestien välityksellä ja välittää täältä tiedot frontendiin API-kutsuina ja lopulta tieto esitetään käyttöliittymässä, kuten tässä tilanteessa hälytysten tilatieto, johon frontend reagoi sille asetetuin toimenpitein (kuva 12).

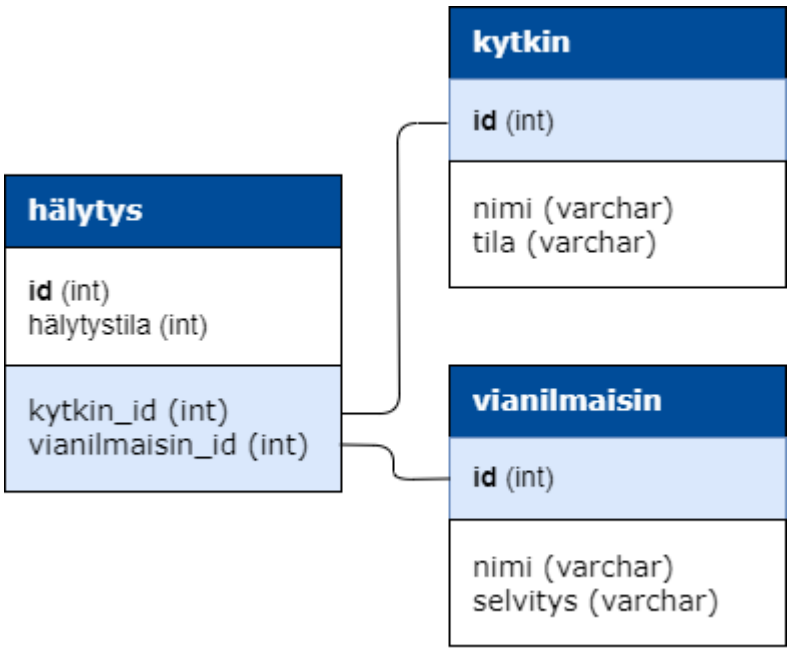


Kuva 12. C# .NET -backend toimii välikätenä frontendille ja tietokannalle.

3.3 Tietokanta

3.3.1 Microsoft SQL Server / Relaatiotietokanta

Microsoft SQL Server (MS SQL Server) on relaatiotietokannan hallintajärjestelmä, jonka tehtävänä on tallentaa ja noutaa tietoja muiden ohjelmistosovellusten pyynnöstä. Relaatiotietokannoissa on tapana tallentaa tiedot perinteiseen tietokantaan ja käsitellä tätä tietoa jollain kyselykielellä, kuten SQL:llä. Se esittää tietojoukot taulukkokokoelmina ja järjestää toisiinsa liittyvät tietopisteet (kuva 13). [11.] Kuten kuvassa nähdään, hälytykset ovat linkitetty kytkimiin ja vianilmaisimiin id:n perusteella, kuten relaatiotietokannoissa on tapana. Lisäksi näille annetaan tyypit, joiden perusteella tiedetään, mitä tyyppiä kukin taulukon kenttä pitää sisällään ja mitä sinne voidaan syöttää. Koska WebMap:ssa osa datasta tarvitaan olioina tallennettuna, toimii PostgreSQL-tietokanta välikätenä tässä.

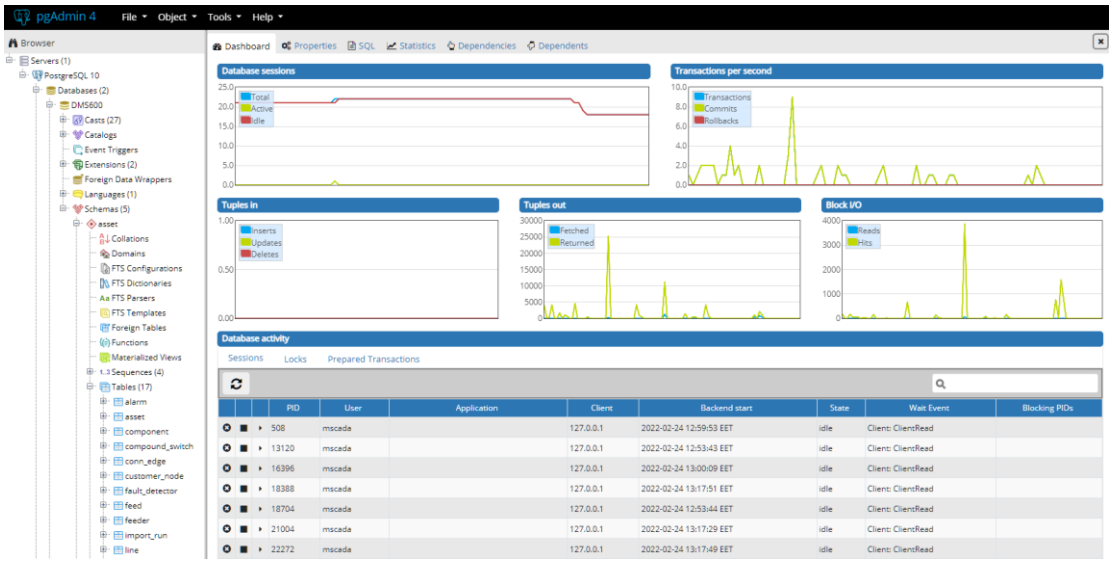


Kuva 13. Relaatiotietokannan rakenne.

3.3.2 PostgreSQL / Oliorelaatiotietokanta

PostgreSQL on tehokas, avoimen lähdekoodin oliorelaatiotietokanta (järjestelmä, joka on samanlainen kuin relaatiotietokanta, mutta jossa on oliopohjainen tietokantamalli, lyh. ORD), joka käyttää ja laajentaa SQL-kieltä yhdistettynä moniin ominaisuuksiin, jotka tallentavat ja skaalaavat turvallisesti jopa monimutkaisimmatkin tietokokonaisuudet. ORD (toiselta nimeltä objektirelaatiotietokantajärjestelmä, lyh. ORDBMS) on tietokannan hallintajärjestelmä, joka tukee minkä tahansa oliopohjaisen tietokantamallin peruskomponentteja, kuten olioita ja luokkia tavallisen relaatiotietokannan lisäksi. PostgreSQL sisältää monia ominaisuuksia, joiden tarkoituksena on auttaa kehittäjiä rakentamaan sovelluksia, järjestelmänvalvojia suojaamaan tietojen eheyttä ja rakentamaan vikasietoisia ympäristöjä. Lisäksi PostgreSQL on tehty auttamaan hallitsemaan tietoja riippumatta siitä, kuinka suuri tai pieni tietojoukko on kyseessä. [12.]

WebMap:ssä PosgreSQL-tietokantaa hallitaan pgAdminilla (kuva 14), joka on tehokas graafinen käyttöliittymätyökalu rakennettu Pythonilla (ohjelmointikieli) ja JavaScriptillä. Tämä on hyvä työkalu tiedon kulun tarkasteluun. Samalla nähdään, mitä tietokantaan tulee. Lisäksi pgAdminin tarjoama graafinen käyttöliittymä yksinkertaistaa tietokantaobjektien luomista, ylläpitoa ja käyttöä.



Kuva 14. PostgreSQL:n graafinen käyttöliittymä pgAdmin.

3.4 Versionhallinta

Verkkosovelluksen versionhallinta on toteutettu käyttäen Azure DevOpsia sekä Gittiä. Microsoftin Azure DevOps toimii projektikoodin tallennuspaikkana, joka mahdollistaa muutosten seurannan ja helpon yhteistyön muiden kehittäjien välillä, sillä koodia voidaan hallinnoida helposti ja nopeasti. Lisäksi se tarjoaa hyviä kehittäjäpalveluita, joilla suunnitellaan työtä ja seurataan tämän kulkua. [13.] Azure DevOpsin tarjoama tikettisysteemi on monipuolinen työkalu, jolla seurataan tuotteen kehitysjonoa (*backlog*). Kehitysjono koostuu erilaisista kehitettävistä asioista (*backlog items*), joita jaetaan lähinnä uusiin ominaisuuksiin ja bugikorjauksiin. Nämä jaetaan sitten eri sprintteihin asiakastarpeiden mukaan prioriteettijärjestykseen, joissa lopulta tehtävät jaetaan kehittäjille. Kehitettävät asiat koostuvat tehtävistä (*tasks*) ja pitävät sisällään eri tiloja riippuen, missä vaiheessa tehtävä työ on. Näistä yleisimpiä ovat hyväksytty, sitoutunut, valmis testattavaksi ja valmis.

Koodikatselmointi on myös suuri osa AzureDevopsia, ja tässä muut kehittäjät katselmoivat toistensa PR:iä (*pull requests*) eli tehdyn tehtävän koodia. Koodit ovat omissa haaroissaan, joissa niitä työstetään. Kun tehtävä on valmis, luodaan PR ja lopulta liitetään päähaaraan. PR-katsauksessa muut kehittäjät voivat muun muassa kommentoida koodiin ehdotuksia tai korjauksia, jonka jälkeen PR hyväksytään ja yhdistetään päähaaraan, kuten aiemmin mainittiin.

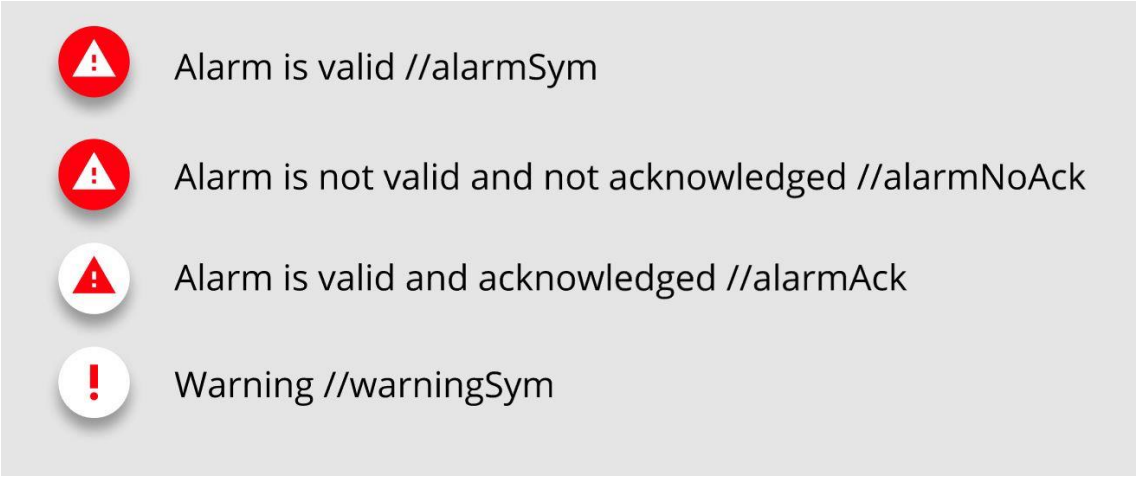
DMS600-kehityksessä Azure DevOpsin koodivarastot (*repositories*) toimivat Gittiä käyttäen, joka on vahva VCS (versiohallintajärjestelmä). Gitin avulla jokainen käyttäjä voi työstää paikallisesti omaa kopiota sovelluksesta. Kehittäjät voivat siis paikallisesti ohjelmoida vapaasti tehtäviään omissa haaroissaan ja liittää valmiit koodit PR:ien välityksellä päähaaraan. Git-komentoja on useita, ja niiden käyttö on hyvin nopeaa komentorivillä. *Git commit* -komennolla tallennetaan muutokset paikalliseen koodivarastoon, kun taas *git push* vie nämä samat muutokset verkossa olevaan koodivarastoon. G*it merge* -komennolla yhdistetään muutokset päähaaraan, *git pull* päivittää oman paikallisen koodivaraston. Jos paikallisessa koodivarastossa on muutoksia, tulee sen sijaan käyttää *git rebase*

-komentoa, jonka avulla yhdistetään nämä paikalliset muutokset verkossa olevan sovelluksen muutoksien kanssa. [14.]

4 Hälytykset ja mittaukset

4.1 Tulokset

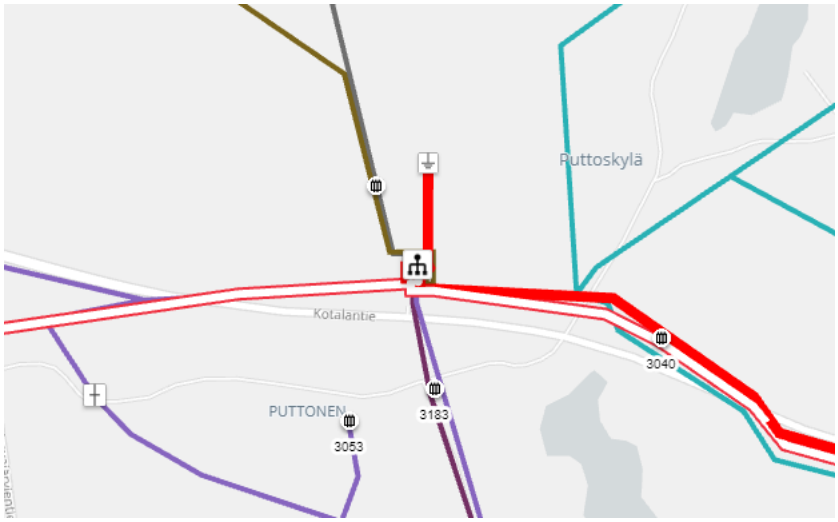
Hälytysten lähtökohtana oli luoda yhtenäinen visuaalinen esitystapa DMS600:n ja SYS600:n välillä WebMap:iin ja täten päädyttiin yksinkertaiseen ja selkeään lopputulokseen ikonien suhteen (kuva 15). Hälytysten värien valinta perustuu yleiseen varoitusväriin punaiseen, joka helpottaa hälytysten havaitsemista kartalla. WebMap on täynnä erilaisia värejä ja punainen on käytössä lähes aina, kun on kyseessä jonkinlainen vikatilanne. Kun esimerkiksi sähkölinja on poikki, linja on väriltään punainen (kuva 16).



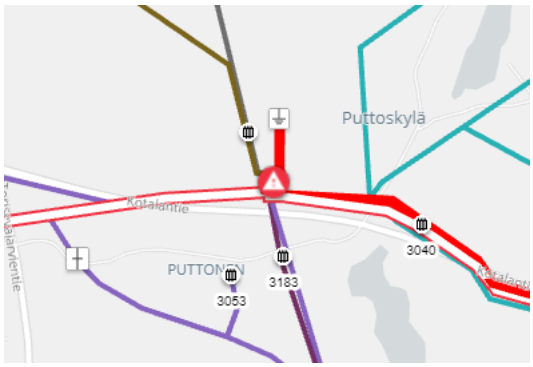
Kuva 15. Eri hälytystilojen ikonit

Hälytykset esiintyvät kytkimissä ja vianilmaisimissa sekä tulevaisuudessa mittauksissa. Frontend käsittelee hälytykset siten, että kun backend välittää tietokannasta tulevan hälytystaulukon, se vertaa tämän taulukon arvoja kytkin- ja vianilmaisintaulukoiden kanssa ja vaihtaa näitä vastaavien id-arvojen ikonit hälytystilan mukaan. Koska kyseessä on hälytystila, näytetään tämä hälytys kar-

talla suuremmalla prioriteetilla kuin esimerkiksi tavallinen kytkin. Prioriteetti vaikuttaa tilanteen mukaan karttakomponentin esitysjärjestykseen. Hyvä esimerkki tästä on, kun karttaa katsotaan loitommalta (kartalla näkyy avarampi alue) ja kyseisessä alueessa on useita erilaisia karttakomponentteja, kuten kytkimiä, näkyisi suurimalla prioriteetilla oleva karttakomponentti yksinään tai päällimmäisimpänä. Jos näillä karttakomponentilla on tarpeeksi etäisyyttä toisistaan, tilanne on eri. Esimerkkitilanne, jossa tämä tulee ilmi, on sähköasemissa. Jotta niiden erilliset karttakomponentit näkyisivät yksittäisinä, tulee karttaa lähentää (kartalla näkyy pienempi alue, mutta lähempää katsottuna). Tällaisessa tilanteessa sähköasemassa on kytkimiä, sähkönsyöttölaitteita ja muuntajia sekä tilanteen mukaan hälytyksiä (kuva 19). Jos kartta on hyvin loitonnettu, normaalisti kartassa näkyy sähköaseman ikoni (kuva 16), mutta hälytystilan ollessa päällä näkyisi sähköaseman kohdalla hälytysikoni (kuva 17).

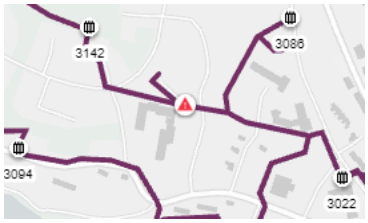


Kuva 16. Sähköaseman ikoni, kun kartta on loitonnettuna

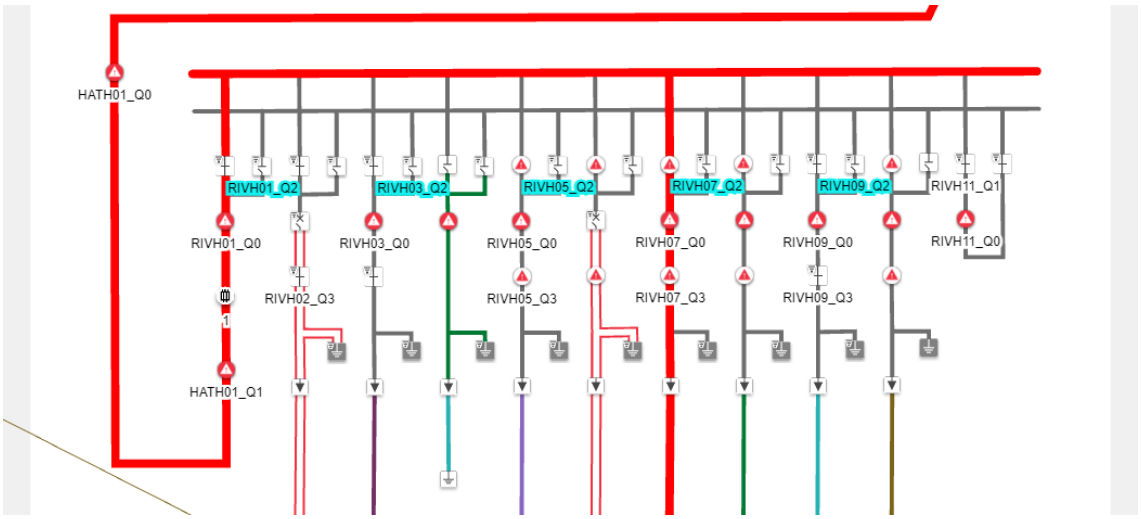


Kuva 17. Hälytys kytkimessä, joka sijaitsee sähköaseman sisällä

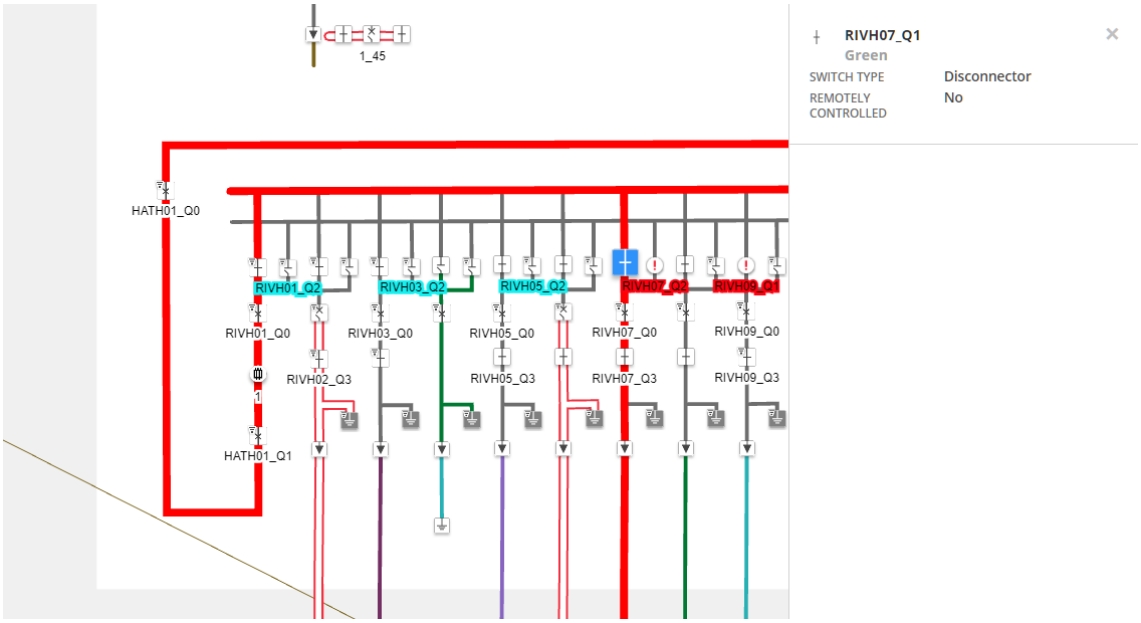
Käyttöliittymässä hälytykset esiintyvät selkeästi kartassa sähköverkkojen ja -asemien varrella (kuvat 18 ja 19). Kun kyseistä hälytystä painetaan, avautuu sivupaneeli, jossa saadaan kytkimeen tai vianilmaisimeen liittyvät tiedot (kuva 20). Kooltaan hälytykset ovat samankokoisia kuin muut karttakomponentit, kuten kytkimet, vianilmaisimet sekä sähköasemat. Sähköasemat ovat kooltaan suurempia kuin esimerkiksi kytkimet ja vianilmaisimet. Tämän takia hälytyksen korvatessa sähköaseman, se on myös kooltaan suurempi. Toisin sanoen hälytysikonin koko on riippuvainen siitä ikonista, jota se korvaa.



Kuva 18. Kuitattu hälytys kartalla osana sähköverkostoa.



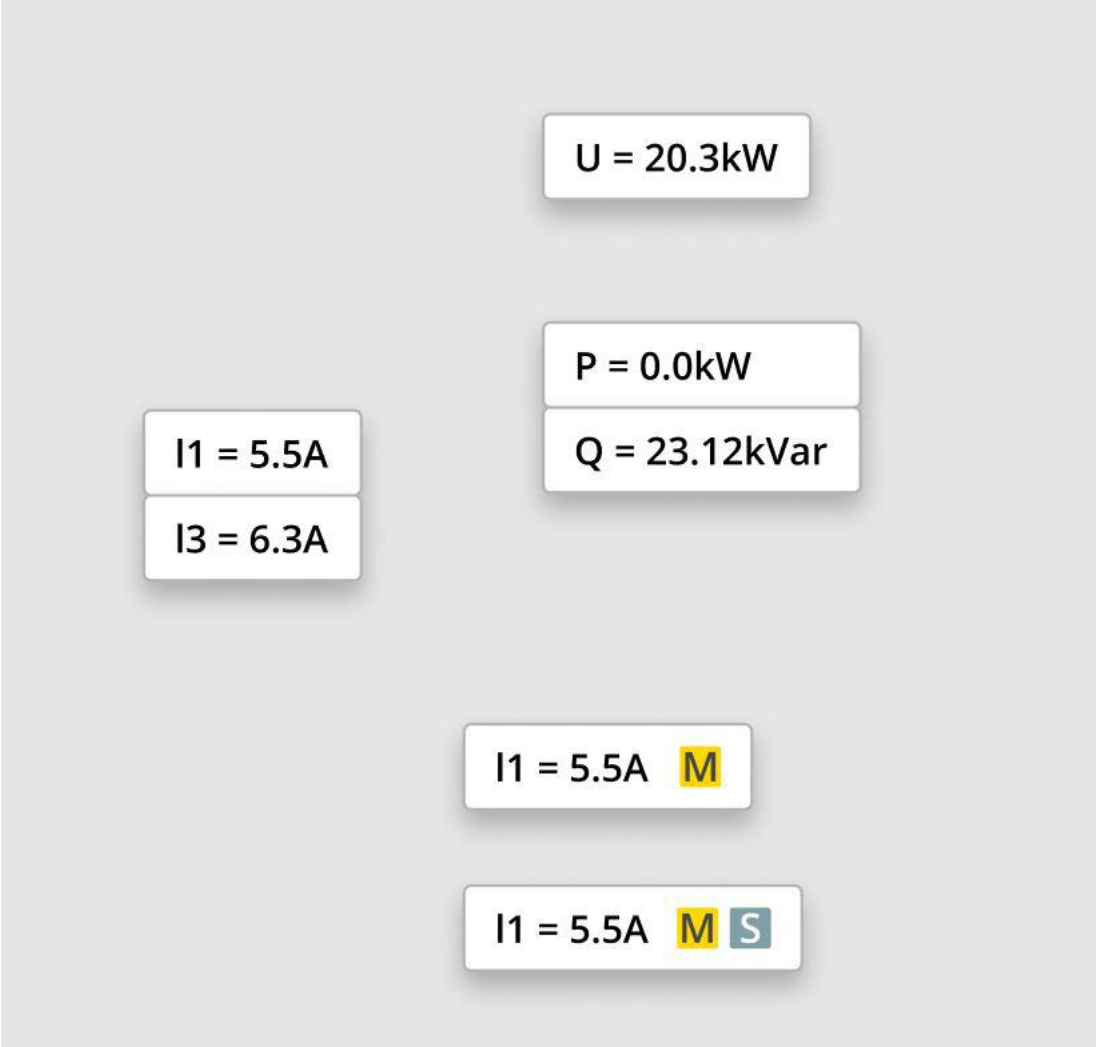
Kuva 19. Hälytyksiä sähköasemassa.



Kuva 20. Hälytystä painettaessa sivupaneeli avautuu ja näyttää painetun kartta-komponentin tiedot, joka tässä tilanteessa on kytkin.

Mittausten esityssuunnitelma on toteutuksessa yksinkertainen suorakulmio, jossa esiintyvät mittauslaitteen eri tiedot (kuva 21), kuten jännite (U), sähkövirta (I), sähköteho, (P) ja loisteho (Q). Mittaukset esiintyvät kartassa sähköasemien

sisällä, joissa tämä tieto on tarpeellista saada nopealla katsauksella. Tarkka sijoitus mittauksille on vielä suunnitteilla, mutta kuvassa 5 nähdään suurin piirtein, miten ne voisivat sijoittua käyttöliittymään.



Kuva 21. Mittausten esityssuunnitelma

4.2 Jatkokehitys

Hälytysten jatkokehitysideoita on useita ja uusia varmasti ilmenee vielä tuotteistusvaiheessa. Lähtökohtaisesti hälytysten tuonti sivupaneeliin on yksi kehitettävä toimenpide. Tilanne, jossa käyttöliittymässä on painettu kytkintä tai vianilmaisinta, avautuu sivupaneeliin myös tieto hälytystilasta. Tässä tilanteessa backendille lähetetään ilmoitus hälytystietojen hakuun, jonka jälkeen backend

lähettää tietokantakyselyn ja välittää tämän tiedot frontendille. Lisäksi hälytystilan muokkaus WebMap:ssa on tarpeen, sillä tällä hetkellä se on mahdollista vain työpöytäsovelluksessa. Hälytyksiä tulisi tällöin pystyä muun muassa kuittaamaan. SYS600:ssa hälytystilojen eroavaisuuksia on ilmaistu muun muassa vilkkumisella tietyissä tilanteissa ja tämä olisi myös tervetullut lisäys WebMap:iin.

Mittaukset vaativat jatkokehityspuolella backend-kyselyiden toteuttamisen ja niiden välittämisen frontendille, jotta saatavat tiedot voidaan esittää käyttöliittymässä. Lisäksi mittaustietojen sijoitus sähköaseman sisällä jää suunniteltavaksi asiaksi, kuten sijoitetaanko ne ympäri sähköasemaa niin kuin DMS600-työpöytäsovelluksessa vai pelkästään yhteen samaan paikkaan esimerkiksi vasempaan alakulmaan. Mittauksilla on lisäksi tarkoitus toteuttaa hälytystilat mittausarvojen perusteella. Kun arvo on liian korkea tai alhainen, aiheuttaa se hälytystilan, ja tämä indikoituu hälytysikonilla. Tämäkin vaatii erillisen käyttöliittymäsuunnitelman esimerkiksi tulisiko hälytysikoni mittausarvoon pitävään suorakulmioon vai aivan muualle.

5 Yhteenveto

Tavoitteena oli edistää verkkosovellusta, jonka tarkoitus on mahdollistaa sähkönjakeluverkon seurantaa ja hallintaa. Tarkoitus oli luoda prototyyppi hälytysten ja mittausten visualisointiin, jonka avulla tarvittavat ominaisuudet voidaan tuotteistaa myöhemmin tulevaisuudessa. Projektityöskentelyn aikana löytyi suuri määrä ratkaistavia ongelmia ja jatkokehitysideoita, jotka helpottavat tulevaisuudessa tuotteistusprosessia.

Hälytysten visualisointi toteutettiin taustakartan päälle kytkimien ja vianilmaisimien kohdalle näiden hälytystilan mukaan. Hälytystilan ollessa aktiivinen esiintyy kartalle hälytysikoni, jonka perusteella voidaan havaita, missä kytkimessä tai vianilmaisimessa on ongelmatilanne, ja tehdä tarpeelliset toimenpiteet sen kannalta tilan korjaamista varten.

Mittauksien visualisoinnin toteutus sisälsi lähinnä käyttöliittymäsuunnitelman ja komponentin rakentamisen Reactilla. Jatkokehitystoimenpiteiksi jää laajalti yhdistäminen tietokantaan ja backend-toteutuksen teko sekä hälytysten integrointi mittauksiin.

Kuten elämässä, joskus tulee vastoinkäymisiä, mutta niistä päästään takaisin jaloille ja etenemään. Työssä esiintyi monenlaisia erilaisia haasteita, mutta askel kerrallaan niistä päästiin yli ja lopuksi saatiin toimiva kokonaisuus hyvillä jatkokehitysideoilla.

Lähteet

1 MicroSCADA X DMS600 System Overview. 2021. Yrityksen sisäinen dokumentti. Hitachi Energy. Luettu 29.11.2021.

2 MicroSCADA X SYS600 Operation Manual 2021. Yrityksen sisäinen dokumentti. Hitachi Energy. Luettu 03.12.2021.

3 Intro to React. 2022. Verkkoaineisto. ReactJS.org. <https://reactjs.org/tutorial/tutorial.html>. Luettu 12.01.2022.

4 React: The Virtual DOM. Verkkoaineisto. CodeAcademy.com. <https://www.codecademy.com/article/react-virtual-dom>. Luettu 15.02.2022.

5 Hook at a Glance. 2022- Verkkoaineisto. ReactJS.org. <https://reactjs.org/docs/hooks-overview.html>. Luettu 20.01.2022.

6 Redux Fundamentals. 2022. Verkkoaineisto. Redux.js.org. <https://redux.js.org/tutorials/fundamentals/part-1-overview>. Luettu 25.01.2022.

7 What is TypeScript. 2021. Video. Microsoft.com. <https://docs.microsoft.com/en-us/shows/Web-Wednesday/What-is-TypeScript>. Katsottu 29.01.2022.

8 Inkscape Overview. 2022. Verkkoaineisto. Inkscape.org. <https://inkscape.org/about/>. Luettu 2.2.2022.

9 A tour of the C” language. 2021. Verkkoaineisto. Microsoft.com. <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>. Luettu 16.02.2022.

10 What is.NET Framework. 2022. Verkkoaineisto. Microsoft.com. <https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework>. Luettu 28.02.2022.

11 Relational Database. 2022. Verkkoaineisto. Omnisci.com. <https://www.omnisci.com/technical-glossary/relational-database>. Luettu 26.02.2022.

12 About PostgreSQL. 2022. Verkkoaineisto. PostgreSQL.org. <https://www.postgresql.org/about/>. Luettu 17.02.2022.

13 Azure DevOps. 2022. Verkkoaineisto. Microsoft.com. <https://azure.microsoft.com/en-us/services/devops/#overview>. Luettu 17.02.2022.

14 Git. 2022. Verkkoaineisto. Linux.fi. <https://www.linux.fi/wiki/Git>. Luettu 25.02.2022.