jamk.fi

GEMA - Settings Tool

Laatutarkistukset

Taneli Sormunen

Opinnäytetyö
Huhtikuu 2022
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Sormunen, Taneli	Opinnäytetyö, AMK	Huhtikuu 2022
	Sivumäärä 29	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
GEMA - Settings Tool
Laatutarkistukset

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Pasi Manninen, Kari Niemi

Toimeksiantaja(t)
Pinja Operational Excellence Oy

Tiivistelmä

Opinnäytetyön toimeksiantajana toimi Pinja Operational Excellence Oy, joka on ohjelmistojen suunnitteluun ja valmistukseen erikoistunut yritys. Pinjan valmistamiin tuotteisiin lukeutuu mm. tuotannonohjausjärjestelmä GEMA, jonka pohjalla on moderni koneseurantajärjestelmä, joka on mahdollista laajentaa täysimittaiseksi MES-järjestelmäksi.

Opinnäytetyön tehtävänä oli luoda GEMA:n laatutarkistuksien asetustyökalun prototyyppi Microsoftin kehittämällä Blazor-käyttöliittymäviitekehyksellä ja arvioida Blazorin kannattavuus tulevaisuudessa sekä verrata Blazor-toteutusta nykyiseen työkaluun, joka oli tehty React-kirjastolla.

Tehtävänantoa lähdettiin toteuttamaan toimeksiantajan interaktiivisen rautalankamallin pohjalta ja ainoana vaatimuksena oli käyttää pääteknologiana Blazor-käyttöliittymäviitekehystä. Työ oli käytännössä frontend-kehitystä, sillä tarvittavat REST-rajapinnat prototyypin tekemiseen oli jo valmiiksi olemassa. Aluksi uuden teknologian käyttö oli hieman kömpelöä ja haastavaa, mutta syvemmän perehtymisen jälkeen kehitys alkoi tuntua luontevalta.

Tuloksena syntyi karkea prototyyppi laatutarkistuksien asetustyökalusta uudella visuaalisella ilmeellä sekä saatiin parempi käsitys Blazorin sen hetkisen käytön kannattavuudesta. Työn loppupuolella alettiin kallistua vahvasti johtopäätöksiin, että Blazor on vielä liian nuori teknologia käytettäväksi laajemmin. Dokumentaation puute tekee kehityksestä hankalaa ja on järkevämpää antaa teknologian vielä kypsyä aikansa. Lisäksi katsottiin, että ei ole kannattavaa ylläpidon kannalta alkaa sekoittamaan kahta teknologiaa keskenään ja täten päätettiin jatkaa laatutarkistustyökalun kehitystä Reactilla.

Avainsanat (asiasanat)
Blazor, C#, Laadunvarmistus, REST, Web-kehitys, Web-sovellus

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Descripton

Author(s)	Type of publication	Date
Sormunen, Taneli	Bachelor's thesis	April 2022
		Language of publication: Finnish
	Number of pages: 29	Permission for web publication: x

Title of publication
GEMA - Settings Tool
Quality Checks

Degree programme
Information and Communication Technology

Supervisor(s)
Pasi Manninen, Kari Niemi

Assigned by
Pinja Operational Excellence Oy

Abstract

Thesis assignment was done in co-operation with the client company Pinja Operational Excellence which specializes in designing and developing software's on different platforms. Products made by Pinja include for example production management GEMA which is based on modern machine monitoring system that can be expanded to a full-scale MES system.

Assignment of the thesis was to create new prototype of a GEMA's quality control -module using Microsoft's Blazor framework and to evaluate the future viability of Blazor framework and to compare Blazor implementation with the current module made with the JavaScript library React.

The assignment was based on client's interactive wireframe model and the only requirement was to use Microsoft's Blazor UI framework as the main technology. The work was mainly in practice a frontend development since the REST interfaces necessary for this prototype assignment already existed. At first using the new technology was little clumsy and challenging but after gaining a deeper understanding of the framework, the development began to feel more natural.

The result was a rough prototype of a quality check setting tool with an upgraded visual look and a better understanding of Blazor's current viability in its current state. In the end with client company, we decided that Blazor is still too young technology to be used more widely in the GEMA project. Lack of documentation makes developing with Blazor awkward and it would be wise to let the technology mature its time. It was considered unviable in terms of maintenance to mix the two technologies with each other and thus it was decided to continue the development of the quality check setting tool with React.

Keywords/tags (subjects)
Blazor, C#, Quality assurance, REST, Web-application, Web-development

Miscellaneous (Confidential information)

Sisältö

Kuviot

Taulukot

Sanasto

.Net Framework	.Net Framework on Microsoftin kehittämä olioperusteinen ohjelmointiympäristö Windows-sovelluksien ja verkkopalveluiden tuottamiseen.
API	Application programming interface eli ohjelmointirajapinta.
Create-React-App	Npx-työkalun komento, jolla voidaan luoda React-sovelluksen runko.
HTTP	HTTP (Hypertext Transfer Protocol) on selaimien ja WWW-palvelimien tiedonsiirroissa käyttämä protokolla.
IDE	IDE (Integrated Development Environment) tarkoittaa ohjelmointiympäristöä, jolla ohjelmoija pystyy tuottamaan lähdekoodia tekstieditorilla sekä kääntämään koodin.
JavaScript	JavaScript on dynaaminen komentosarjakieli, jolla pääosin tuotetaan selaimissa toimivia verkkosovelluksia.
KPI	Key Performance Indicator eli mitattava tunnusluku.
MarvelApp	MarvelApp on selainpohjainen työkalu sovelluksien interaktiivisten rautalankamallien toteuttamiseen.
MES	Manufacturing Execution System on tuotannonohjausjärjestelmä, joka on syntynyt käytännön vaatimuksista.
Microsoft	Microsoft Corporation on yhdysvaltalainen ohjelmistoalan yritys, joka on valmistanut mm. Windows-käyttöjärjestelmät.
OPC	Open Platform Communications on joukko standardeja, joita käytetään teollisuudessa automaatiosovelluksien ja ohjelmoitavien logiikoiden välisessä keskustelussa.
PLC	Programmable Logic Controller eli ohjelmoitava logiikka on pienikokoinen tietokone, joka mahdollistaa automaatioprosessin ohjauksen sekä mittaustietojen keräämisen.
REST	Representational State Transfer on arkkitehtuurimalli, jonka tarkoitus on parantaa rajapintojen toimintaa.
Session Storage	Selaimen istuntokohtainen välimuisti.
SQL	Structured Query Language on kyselykieli, jolla kommunikoidaan relaatiotietokannan kanssa.
Task	Asynkroninen tehtävä C#-kielessä.
URI	Uniform Resource Identifier on merkkijono, jolla kerrotaan tietoverkossa sijaitsevan tiedon nimi, sijainti tai molemmat.
WASM	WebAssembly on web-ympäristöille kehitteillä oleva ohjelmointikieli ja vaihtoehto JavaScriptille.

1 Johdanto

1.1 Toimeksiantaja

ARROW Engineering Oy (myöhemmin pelkkä ARROW) on vuonna 1993 Suomessa perustettu yritys, jolla on yli 500 asiakasta 30:ssä eri maassa. ARROW valmistaa erilaisia järjestelmäratkaisuja erityisesti valmistavalle teollisuudelle tuotannon ja kunnossapidon operatiiviseen johtamiseen ja kehittämiseen. ARROW:n verkkosivuilla kerrotaan heidän Operational Excellence -konseptinsa keskittyvän tuottavuuden ja päivittäisjohtamisen parantamiseen, tiedon digitalisoinnin ja visuaalisen johtamisen avulla (Yritys. n.d.). Vuonna 2018 elokuussa ARROW:sta tuli Protacon-konsernin tytäryhtiö tapahtuneen yrityskaupan myötä. (Uudistuksia ARROW:n johtoon. n.d.)

Maaliskuussa vuonna 2020 Protacon ja useat sen alla toimineet yritykset esim. ARROW, SWD ja Powen yhdistyivät yhteisen brändin alle nimeltä Pinja ja ARROW:sta tuli Pinja Operational Excellence Oy. Yhteinen brändi selkeyttää yrityksen rakennetta ja toimintaa asiakkaille ja helpottaa entisestään asiakkaiden palvelemista kattavammin eri toimialoilla ja suuremmissa kokonaisuuksissa. (Protacon on nyt Pinja - teollisuuden uudistamisella ja digitalisaatiolla vahvaa kasvua. n.d.)

1.2 Tavoite

Opinnäytetyön tavoitteena oli kehittää toimivampi ratkaisu Pinjan kehittämään tuotannonseurantajärjestelmä GEMA:n laatutarkistusmoduuliin, jolla käyttäjät voivat luoda, muokata ja poistaa erilaisia tuotantoa valvovia tietyn väliajoin toistuvia laatutarkistuksia ja huoltotehtäviä. Yksi opinnäytetyön keskeisistä tehtävistä onkin tuoda esille, mitä puutteita ja kehitystarpeita nykyisessä laatutarkistustyökalussa on. Kehitysnäkökulmaa pohditaan ominaisuuksien sekä teknologiavalintojen kautta.

Samalla oli tarkoitus tutkia ja testata Microsoftin kehittämää Blazor-käyttöliittymäviitekehystä ja sen käytön kannattavuutta lähitulevaisuudessa sekä verrata sitä nyt käytössä olevaan Facebookin kehittämään React.js-kirjastoon.

2 GEMA

2.1 Yleisesti

Pinjan kehittämiin tuotteisiin kuuluu muun muassa GEMA, joka on MES-tason järjestelmäratkaisu tuotannon seurantaan ja tehostamiseen. GEMA:n ominaisuuksiin lukeutuu mm. koneseuranta, jolla voidaan reaaliaikaisesti seurata järjestelmään kytkettyjen koneiden ja työpisteiden tiloja, raportointityökalu, jonka avulla järjestelmästä voidaan luoda erilaisia raportteja mm. tuotannon tiloista ja suorituskykymittareista (KPI) sekä hallintatyökalu, jonka avulla käyttäjä voi konfiguroida ja hallita järjestelmäkohtaisia asetuksia, hallita järjestelmän käyttäjiä, työaikoja sekä muita ominaisuuksia. (ARROW GEMA. n.d.)

2.2 Arkkitehtuuri

GEMA:n toiminta perustuu tuotantolaitoksen tuotantolinjoilta ja työpisteiltä kerätyn raakadatan rikastamiseen, jonka jälkeen se tuodaan loppukäyttäjälle näkyviin dashboardille ja raportteihin halutulla tavalla. Raakadata kerätään tuotannosta erilaisten antureiden avulla, jotka ovat yhteydessä ohjelmoitavaan logiikkaan eli PLC:hen. PLC:ltä raakadata kerätään tiedonkeruupalvelimelle eli OPC-palvelimelle, josta GEMA:n keruusovellus (harvester services) kerää raakadatan GEMA:n SQL-tietokantaan. Tämän jälkeen GEMA:n laskentapalvelu laskee raakadatasta KPI-lukuja ja muita haluttuja tietoja dashboardille ja raportointiin. Opinnäytetyön toimeksianto käsittelee tarkemmin GEMA:n arkkitehtuurissa loppukäyttäjälle näkyvää frontend-verkkosovellusta.

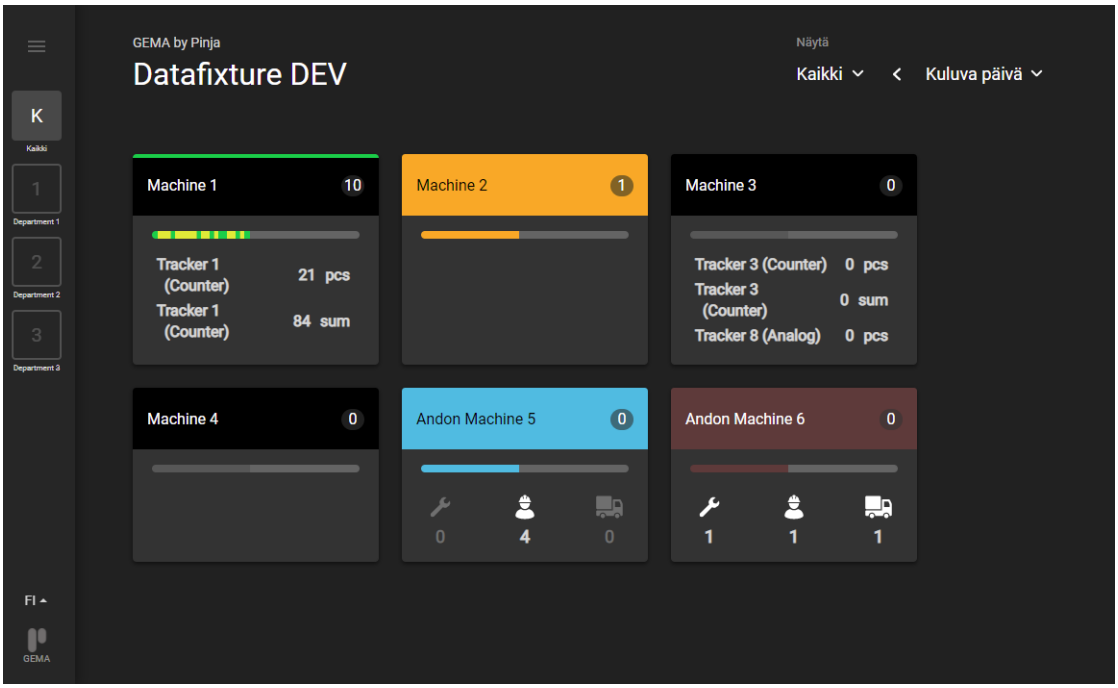
2.3 Dashboard

GEMA:n oleellisin toiminnallisuus on koneseuranta. GEMA:ssa on helppo seurata tuotantolaitoksen eri koneiden tai linjojen tuotantotilat yhdellä silmäyksellä tehdasnäytöltä. Tehdasnäytölle voidaan asettaa kone- ja linjakohtaisia KPI-mittareita, joista on helppo nopeasti silmäillä koneiden tai linjojen tiloja ja tuottavuutta. Lisäksi tehdasnäytölle voidaan muodostaa konekohtainen aikajana, jolla voidaan visualisoida käyttäjälle koneen ajotilat eri ajankohtina; ajotiloina voi olla määriteltynä esimerkiksi:

- Tuotanto (vihreä), kone toimii normaalisti ja on tuotannossa.
- Odotus (keltainen), kone on päällä, mutta ei tuotannossa.
- Häiriö (punainen), koneella on sattunut, jokin tuotantoa hidastava tai kokonaan pysäyttävä tilanne.
- Tauko (sininen), koneella suunniteltu aikataulun mukainen tauko.

Tilamäärittelyt ovat helposti muokattavissa asiakkaan tarpeiden mukaan, erilaisia tuotantotiloja voi olla hyvinkin paljon, joiden avulla käyttäjälle saadaan tuotua hyvinkin tarkkaa tietoa tuotannosta. GEMA:n yksi tärkeimmistä ominaisuuksista on odotus- ja häiriösyiden kuittaus, jolla operaattori pystyy tarkentamaan logiikalta tulleen odotuksen tai häiriön syitä etukäteen määriteltyjen syypuiden kautta sekä kirjoittamalla vapaamuotoisia kommentteja eri tilojen yhteyteen juuri silloin, kun häiriö tai pysähdys tapahtuu. Kuittausten avulla saadaan huomattavasti tarkempaa tietoa häiriön lähteestä, koska juuri käyttäjällä on usein paras näkemys tilanteesta ja ne helpottavat toistuvien pysähdyksien seurantaa ja ongelmatilanteiden korjaamista.

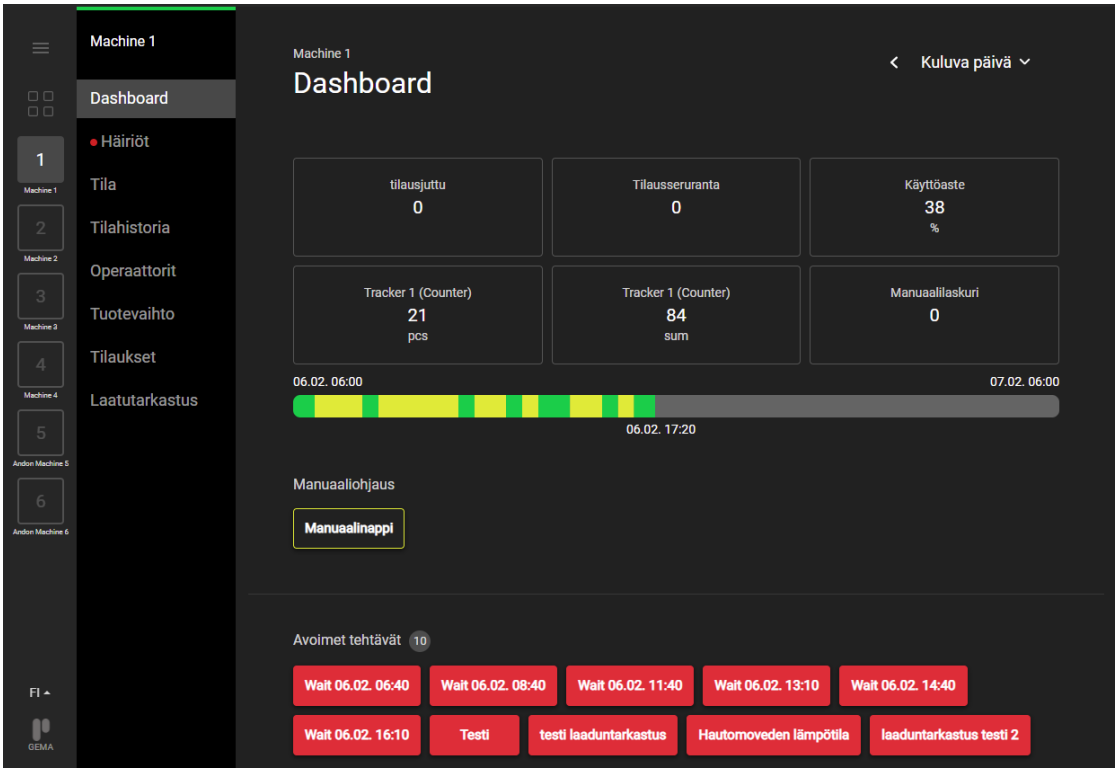
Tehdasnäytöltä (ks. Kuvio 1) on myös mahdollista siirtyä yksittäisen koneen dashboardille, mihin kerätään yleensä tehdasnäyttöä tarkempaa tietoa koneen tuotannosta. Esimerkiksi valmistuneiden tai viallisten tuotteiden kappalemääriä, käytösuhdetta, ajonopeuksia, tilavuuksia, lämpötiloja ja kaikkea, mitä koneisiin liitetyillä logiikkaohjaimilla ja erilaisilla antureilla on mahdollista kerätä ja laskea.



Kuvio 1 GEMA:n tehdasnäyttö, josta on helppo katsoa tehtaan yleistilanne yhdellä silmäyksellä.

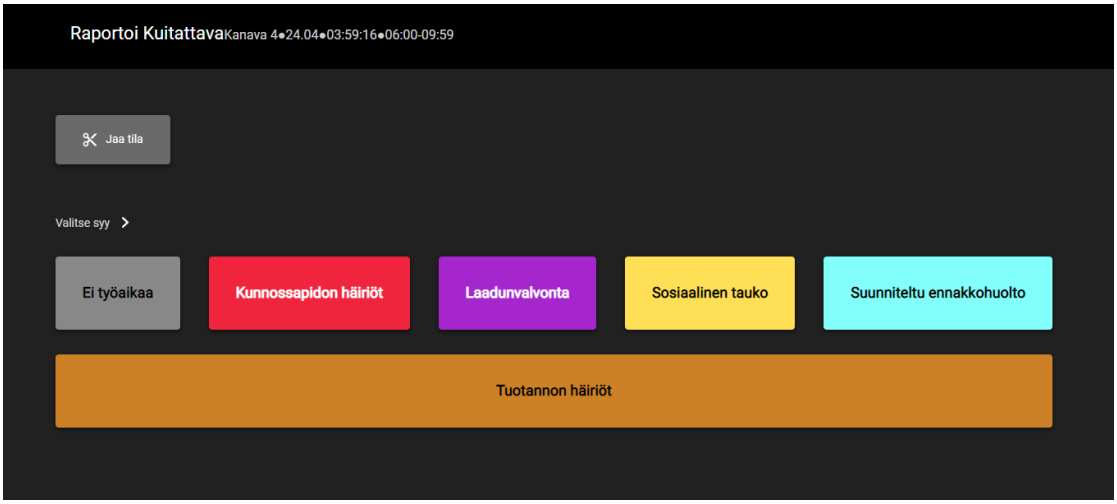
2.4 Häiriöt

Tuotannossa häiriön tapahtuessa tai laatutarkistuksen lauetessa suunnitelman mukaan koneen dashboardille ilmestyy punainen painike ”Avoimet tehtävät” -osioon, jota painamalla käyttäjä pääsee kuittaamaan tapahtuneen häiriön ja syöttämään tarkempaa tietoa häiriöstä tai täyttämään laatutarkistuksen (ks. Kuvio 2).



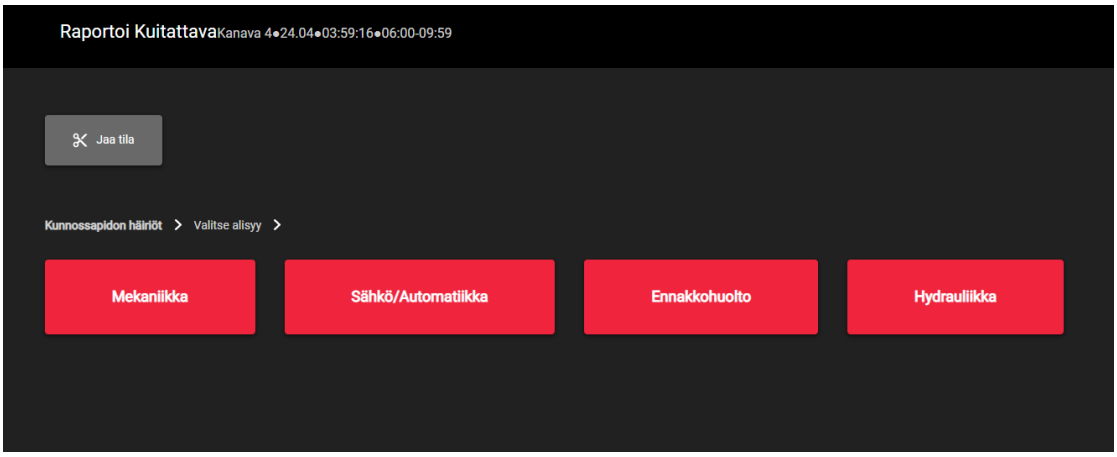
Kuvio 2. Yksittäisen koneen tarkemmat tiedot dashboardilla.

Kun käyttäjä valitsee kuitattavan häiriön, ohjataan käyttäjä uuteen näkymään, jossa valitaan järjestelmään syötetyistä syypuista häiriölle pääsyy (ks. Kuvio 3).



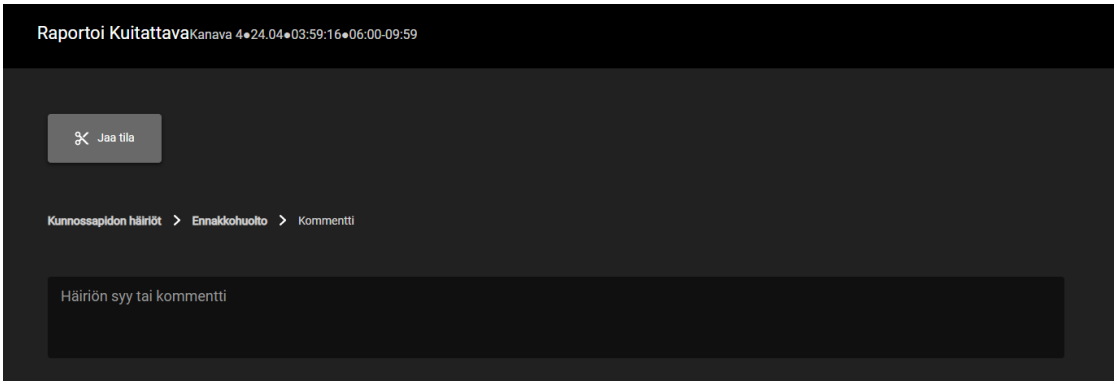
Kuvio 3. Kuitattavan häiriön määritetyt pääsyyt.

Jos pääsyille on asetettu alisyitä voi käyttäjä vielä tarkentaa sattuneen häiriön syytä valitsemalla sopivan alisyyn (ks. Kuvio 4).



Kuvio 4. Kuitattavan häiriön pääsyylle määritetyt alisyyt.

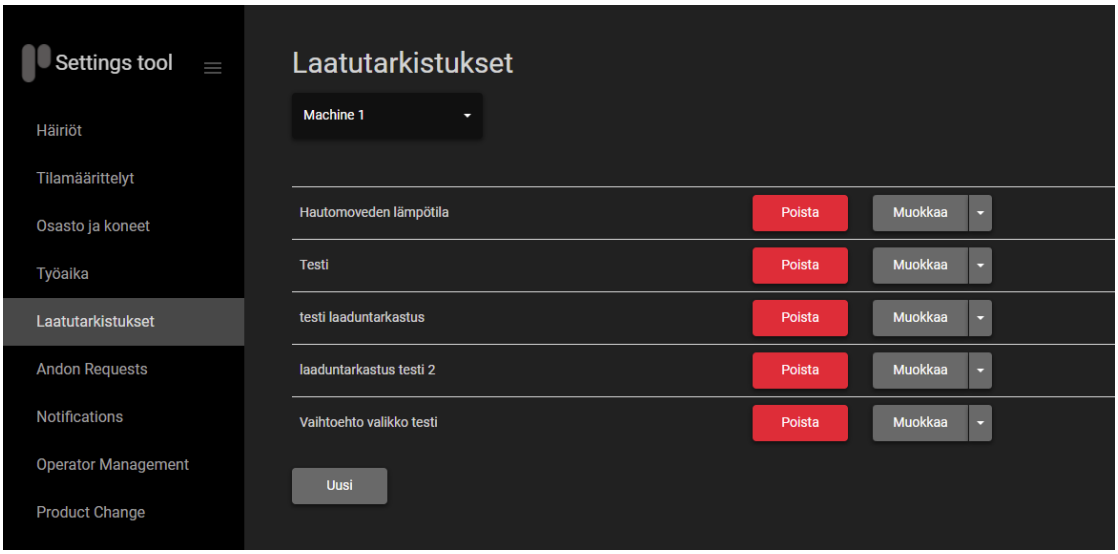
Lisäksi häiriölle on mahdollista kirjata vapaamuotoinen sanallinen selitys mitä tapahtui ja miten häiriö ratkaistiin (ks. Kuvio 5). Tämä mahdollistaa sen, että tulevaisuudessa on helpompi puuttua vastaavanlaisiin häiriöihin ja etsiä ne juurisyyt, jotka aiheuttavat häiriöt ja ne saataisiin kokonaan kuriin.



Kuvio 5. Kuittauksen kommenttikenttä, johon käyttäjä voi syöttää tarkempaa tietoa häiriöstä.

2.5 Settings Tool

GEMA:ssa on erillinen Settings tool -moduuli, jonka avulla käyttäjä voi säädellä GEMA:n asetuksia ja määrityksiä. Settings toolin kautta käyttäjä voi esimerkiksi luoda dashboardille ilmestyvien häiriöiden syypuut, luoda tehtaan työaikamallit, hallita käyttäjiä sekä laatutarkistuksia (ks. Kuvio 6).



Kuvio 6. GEMA:n asetuksien hallintanäkymä.

2.6 Laatutarkistukset

2.6.1 Yleisesti

Laatutarkistukset ovat tärkeä osa valmistavaa teollisuutta, sillä säännöllisesti tehtävät laatutarkistukset auttavat parantamaan laatua sekä tehostamaan tuotantoa, joka ylläpitää asiakastyytyväisyyttä. (MITTAUS, TESTAUS JA LAADUNVARMISTUS - Konsultointi, järjestelmät ja ylläpito - kasvata kilpailukykyäsi panostamalla laatuun. n.d.). Asiakkaat laativat laatutarkistukset omiin tarpeisiinsa eri tuotannon vaiheille ja tuotteille. Tietyillä teollisuudenaloilla kuten esimerkiksi elintarviketeollisuudessa voidaan olla hyvinkin tarkkoja laaduntarkistuksesta ja lisäksi erilaiset maakohtaiset lainsäädännöt voivat vaikuttaa siihen, kuinka useasti laatutarkistuksia tulee tehdä ja kuinka kattavia laatutarkistuksien tulee olla. Lisäksi GEMA:ssa on mahdollista rajoittaa laatutarkistuksien mittaus PIN-koodin taakse, jotta vain sertifioidut henkilöt voivat suorittaa laatutarkistuksia.

GEMA:ssa laatutarkistukset koostuvat viidestä komponentista: perustiedot, kysymykset, suunnitelmat, ohjeet ja validaatiot. Näitä käsitellään seuraavissa kappaleissa.

2.6.2 Perustiedot

Perustiedot kattavat lähinnä yksinkertaisia tietoja laatutarkistuksesta esimerkiksi nimi, mahdollinen alanimi sekä laatutarkistuksen yrityksien määrä eli operaattorilla on esimerkiksi kolme yritystä ja jos kaikki menevät pieleen on laatutarkistus kokonaisuudessaan hylätty. Perustietojen myös asetetaan laatutarkistus aktiiviseksi tai halutessaan pois päältä sekä mikäli laatutarkistus on vain sertifioidun henkilön tehtävissä, voidaan asettaa PIN-koodi vaatimus päälle. (Ks. Taulukko 1, josta käy ilmi perustietojen rakenne.)

Taulukko 1. Laatutarkistuksen perustiedot.

Title	Laatutarkistuksen nimi ja otsikko.
Subtitle	Laatutarkistuksen alaotsikko.
Retry count	Säätää kuinka usein laatutarkistuksen saa tehdä, jos se ei mene läpi.
Fail message	Jos laatutarkistus ei mene läpi näytetään käyttöliittymässä tämä virheviesti.
Active	Säätää onko laatutarkistus aktiivinen vai ei.
Require PIN code	Säätää vaatiiko laatutarkistuksen tekeminen PIN-koodin.

2.6.3 Kysymykset

Kysymykset muodostavat laatutarkistuksen lomakkeen, jonka operaattori täyttää laatutarkistusta tehdessään. Kysymyksille on mahdollista säätää erilaisia tyyppejä, jos tiedetään, että vastaukset ovat kiinteitä voidaan kysymyksille asettaa tietyt vastaukset, joista operaattori valitsee sopivimman. Laatutarkistuksen tyyppi voi olla myös vapaa tekstikenttä, jolloin operaattori kirjaa vapaasti vastauksen tai vaihtoehtoisesti myös sekoitus molempia aiemmin mainitsemaa vaihtoehtoa. Mikäli laatutarkistuksen kysymyksien vastaukset tulee olla numeraalisia, on laatutarkistukselle mahdollista asettaa tietyt raja-arvot, joidenka väliin vastauksen tulee osua tai laatutarkistus ei mene hyväksytysti läpi. Kysymyksien raja-arvojen tarkastelut toimivat yksinkertaisissa tapauksissa, joissa tiedetään kiinteät raja-arvot (ks. Taulukko 2, josta käy ilmi Kysymyksien rakenne).

Taulukko 2. Laatutarkistuksen kysymyksien rakenne.

Title	Kysymyksen otsikko.
Type	Minkä tyyppinen kysymys on kyseessä. Vaikuttaa loppukäyttäjälle siten, että miten tähän kysymykseen tulee vastata. • Text - vapaa tekstikenttä • Select - syötetään vaihtoehtoja, joista käyttäjä valitsee sopivimman. • Select or text – käyttäjä valitsee vaihtoehdoista sopivimman ja lisää vapaan selitteen teksti muodossa.
Row title	Rivin otsikko määrittää, jos halutaan rivittää kysymyksiä tietyn otsikon alle. Esim. Otsikko on "Mitat" ja sen alle voitaisiin ryhmittää kysymyksiä → "leveys", "pituus", "paksuus"
Row	Monesko rivi lomakkeella.
Position	Sijainti rivillä.
Lower limit	Kysymyksen alamitta, jonka alle käyttäjän syöttämä arvo ei saisi mennä.
Lower limit message	Virheviesti jos syötetty arvo on alle alamitan.
Upper limit	Kysymyksen ylämitta, jonka yli käyttäjän syöttämä arvo ei saisi mennä.
Upper limit message	Virheviesti jos syötetty arvo on yli ylimitan.
Minimum character count	Raja-arvo jos halutaan säätää vastaukselle jokin tietty minimi pituus → voidaan karsia "Ok" tyyppiset kuittaukset pois.
Minimum character count error	Tämä virheviesti näytetään, jos vastaus oli liian lyhyt.
Required	Onko kysymys pakollinen laatutarkistusta tehdessä.

2.6.4 Suunnitelmat

Laatutarkistuksien suunnitelmat ohjaavat sitä, kuinka useasti ja millaisissa tilanteissa jokin tietty laatutarkistus tulee tehdä. Suunnittelemalla laatutarkistukset hyvin voidaan myös vahtia laitteiden ja työvälineiden kuntoa, että mahdolliset ennakkohuollot voidaan suorittaa hyvissä ajoin ja vältyttäisiin suuremmilta vahingoilta. GEMA:ssa laa-

tutarkistuksia on mahdollista aktivoida, kun operaattori kirjautuu järjestelmään sisään ja on aloittamassa oman työpisteensä työtehtäviä esim. alkutarkistukset, että kaikki on työpisteellä niin kuin pitääkin tai valmistettavan tuotteen vaihtuessa, että onhan tarvittavat toimenpiteet suoritettu ennen kuin uutta tuotetta aletaan valmistamaan. (Ks. Taulukko 3, josta käy ilmi suunnitelmien rakenne.)

Taulukko 3. Laatutarkistuksen suunnitelmien rakenne.

Type	Sääntö milloin laatutarkistuksen täytyy aktivoitua.
Value	Arvo johon laatutarkistuksen tyyppi on sidottu. Esimerkiksi intervallin minuutti määrä.

2.6.5 Ohjeet

Ohjeiden avulla ohjataan operaattoria tekemään laatutarkistus oikein. Ohjeisiin on mahdollista liittää erilaisia dokumentteja esimerkiksi kuvia tai pdf-tiedostoja, muutkin tiedostomuodot ovat käytettävissä, mikäli käytetty päätelaite vain niitä tukee. (Ks. Taulukko 4, josta käy ilmi suunnitelmien rakenne.)

Taulukko 4. Laatutarkistuksen ohjeiden rakenne.

Text	Tekstimuodossa selitystä/ohjeistusta miten laatutarkistus tulisi tehdä.
Documents	Tähän voi laittaa liitteenä esim. ohjeistusta kuvina tai muina tiedostoina laatutarkistuksen tekemiseen.

2.6.6 Validaatiot

Validaatioiden avulla laatutarkistukselle voidaan asettaa monimutkaisempia tarkasteluita kuin vain kiinteät raja-arvot; näitä käytetään, kun valmistetaan muuttuvaa tuotetta esimerkiksi vaneria. Validaatioissa voidaan käyttää järjestelmään sisäänrakennettuja avainsanoja, jotka vastaavat laatutarkistukselle asetettuja kysymyksiä ja tätem kysymyksille syötettyjä vastauksia on mahdollista käyttää validaation kaavassa. (Ks. Taulukko 5, josta käy ilmi validaatioiden rakenne.)

Taulukko 5. Laatutarkistuksen validaatioiden rakenne.

Formula	Laskukaava validaatiolle.
Upper limit	Raja-arvo minkä yli tulos ei saa mennä.
Upper limit error	Raja-arvon ylittyessä näytetään tämä virheviesti.
Lower limit	Raja-arvo minkä ali tulos ei saa jäädä.
Lower limit error	Raja-arvon alittuessa näytetään tämä virheviesti.

3 Käytetyt teknologiat

3.1 Blazor

3.1.1 Yleistä

Blazor on saanut alkunsa Microsoftilla työskentelevän Steve Sandersonin henkilökohtaisena projektina ja hän esitteli sen NDC-konferenssissa Oslossa vuonna 2017. Blazor on ilmainen avoimen lähdekoodin .NET Frameworkia hyödyntävä verkkopohjainen sovelluskehys, jonka avulla kehittäjät voivat luoda selaimessa toimivia reaaliaikaisia sovelluksia. Blazorilla kehitys tapahtuu yhdistäen C#-ohjelmointikieltä, Razor-syntaksia, HTML-kieltä, mutta mahdollistaen myös JavaScriptin käytön lisänä sisäänrakennetun JavaScript Interopin kautta. (What is Blazor and why is it so exciting. 24.3.2018.)

Blazor hyödyntää WebAssemblyä, joka mahdollistaa lähes natiivin esim. C-, C++- ja C# -koodin ajamisen selainympäristöissä. Tämä mahdollistaa sen, että kehittäjät, joille C-kielet ovat tuttuja pystyvät tuottamaan myös omalla tutulla kielellään sovelluksia selainympäristöihin. (WebAssembly Concepts. 10.9.2020.)

3.1.2 Komponentit

Blazor-sovelluksen komponentit ovat razor-tiedostopäätteisiä tiedostoja, jotka koostuvat pääosin HTML-osiosta sekä koodilohkosta. Koodilohko erotellaan HTML-osiosta

kirjoittamalla esim. tiedoston loppuun @code {…}, jonka sisälle varsinainen logiikka ja komponentin funktiot kirjoitetaan. Koodilohkon sisään määriteltyjä funktioita on mahdollista kutsua erilaisten HTML-eventtien avulla. Alla olevassa NavMenu.razor esimerkissä kutsutaan ToggleNavMenu-funktiota, joka piilottaa tai näyttää kyseisen komponentin onclick-eventin avulla.

```
<div class="top-row pl-4 navbar navbar-dark">
  <a class="navbar-brand" href="">HelloWorldApplication</a>
  <button class="navbar-toggler" @onclick="ToggleNavMenu">
      <span class="navbar-toggler-icon"></span>
  </button>
</div>

<div class="@NavMenuCssClass" @onclick="ToggleNavMenu">
  <ul class="nav flex-column">
      <li class="nav-item px-3">
          <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
            <span class="oi oi-home" aria-hidden="true"></span> Home
          </NavLink>
      </li>
      <li class="nav-item px-3">
          <NavLink class="nav-link" href="counter">
            <span class="oi oi-plus" aria-hidden="true"></span> Counter
          </NavLink>
      </li>
      <li class="nav-item px-3">
          <NavLink class="nav-link" href="fetchdata">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
          </NavLink>
      </li>
  </ul>
</div>

@code {
    private bool collapseNavMenu = true;
    private string NavMenuCssClass => collapseNavMenu ? "collapse" : null;
    private void ToggleNavMenu()
    {
        collapseNavMenu = !collapseNavMenu;
    }
}
```

3.1.3 Pages

Oletuksena Blazor-projektin juureen luodaan Pages-kansio, jonka tarkoitus on erotella sovelluksen tiedostoista eri näkymät omaan kansioonsa. Varsinaisesti sovelluksen sivut ovat razor-tiedostoja, mutta erona on, että tiedostoon määritellään @page -direktiivillä missä osoitteessa kyseinen sivu näytetään. Alla esimerkki Hello World -sovelluksen etusivun rakenteesta:

```
@page "/"
<h1>Hello, world!</h1>
Welcome to your new app.
<SurveyPrompt Title="How is Blazor working for you?" />
```

3.1.4 Imports

Blazor-sovelluksessa komponenteille on mahdollista antaa lisätoiminnallisuuksia muista nimiavaruuksista sisällyttämällä `using`-avainsanan avulla, jotka sisällytetään yleisesti razor-tiedostojen alkuun esimerkiksi `using System.Net.Http;`

Blazor-sovelluksen kansiorakenteen juureen on myös mahdollista luoda `_imports.razor` -tiedosto, jonka avulla voidaan tehdä riippuvuusinjektioita sekä sisällyttää nimiavaruuksia kaikkiin projektin razor-tiedostoihin. Tämä mahdollistaa sen, että jos useassa sovelluksen komponentissa käytetään samoja nimiavaruuksia tai riippuvuuksia apuna, voidaan ne määrittää vain kerran kyseiseen tiedostoon mikä pitää sovelluksen koodin siistimpänä.

3.2 React

Facebookin ylläpitämä React on avoimen lähdekoodin JavaScript-kirjasto sovelluksien käyttöliittymien rakentamiseen. Reactia ei tule sekoittaa virheellisesti sovelluskehykseksi, sillä React hoitaa pelkästään datan esittämisen sovelluksessa ja siksi Reactin kanssa käytetään yleisesti myös muita kirjastoja esimerkiksi sovelluksen tilan hallitsemiseen ja erillistä palvelinpuolen koodia, joka hoitaa sovelluksen logiikan. Reactilla rakennetut käyttöliittymät koostuvat erillisistä komponenteista, joista kootaan isompia kokonaisuuksia. Esimerkiksi komponentteja voivat olla erilaiset painikkeet, tekstikentät sekä taulukot. Komponentit koostuvat komponentin omasta logiikasta huolehtivasta koodista sekä HTML-rakenteesta, joka renderöidään loppukäyttäjälle.

Alla esimerkki create-react-app-komennolla tehdyn esimerkkisovelluksen App-komponentista.

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

Kyseinen App-komponentti voidaan näyttää ruudulla esimerkiksi kutsumalla sitä index.js -tiedostossa ReactDOM.render(…) funktion sisällä HTML-elementin tapaan:

```
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

3.3 REST

3.3.1 Periaatteet

REpresentational State Transfer eli lyhennettynä REST on Roy Fieldingin kehittämä arkkitehtuurimalli, jonka avulla toteutetaan ohjelmointirajapintoja, jotka mahdollistavat sovelluksien kommunikaation verkon yli keskenään. Yleensä REST-rajapinnat kommunikoivat keskenään lähettämällä tietoa erilaisten HTTP-kutsujen välityksellä.

REST-arkkitehtuurissa asiakas (client) lähettää pyynnön palvelimelle (server) kun halutaan esimerkiksi hakea, muokata tai poistaa tietoa ja palvelin lähettää sitten vastauksen clientille. (What is REST. n.d.)

3.3.2 REST-arkkitehtuurin hyvät puolet

REST-arkkitehtuuri mahdollistaa erilaisten web-sovelluksien keskustelun keskenään ja luo raamit ohjelmistorajapintojen kehitykselle. REST-arkkitehtuurin tavoitteena on parantaa rajapintojen suorituskykyä, skaalautuvuutta, yksinkertaisuutta, muunneltavuutta, siirettävyyttä ja luotettavuutta. (What is REST. n.d.)

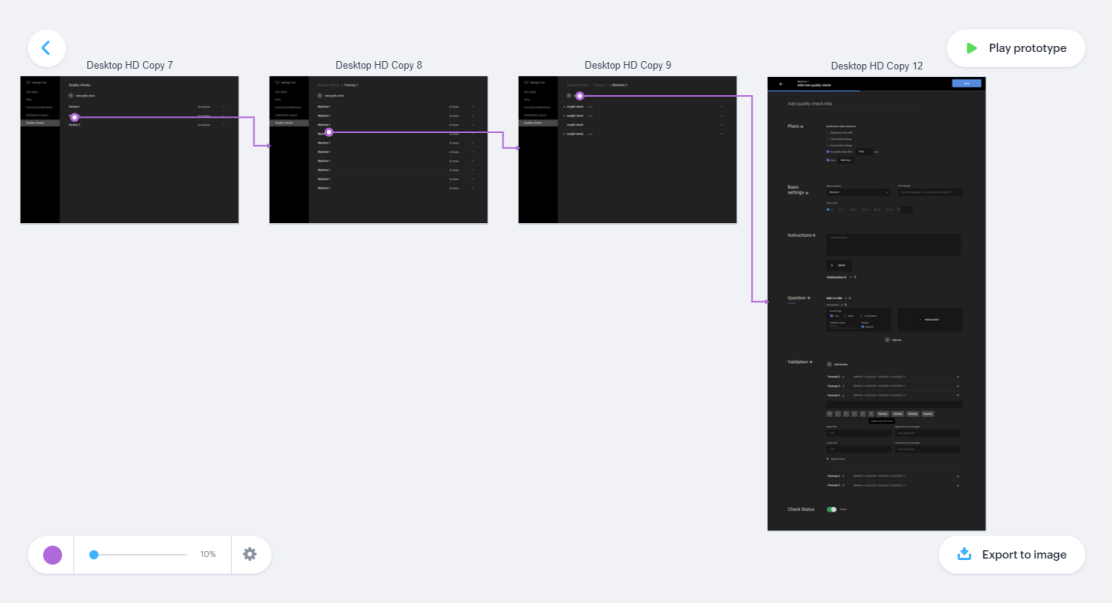
3.3.3 HttpClient

HttpClient on luokkarakenne, jonka avulla ohjelmoija pystyy lähettämään sekä vastaanottamaan erilaisia http-pyyntöjä esimerkiksi REST-rajapintaa vasten. HttpClient on yleisesti käytössä nykyaikaisissa .NET Framework -versioissa. (HttpClient Class. n.d.)

4 Laatutarkistuksien asetustyökalu

4.1 Toteutusmäärittely

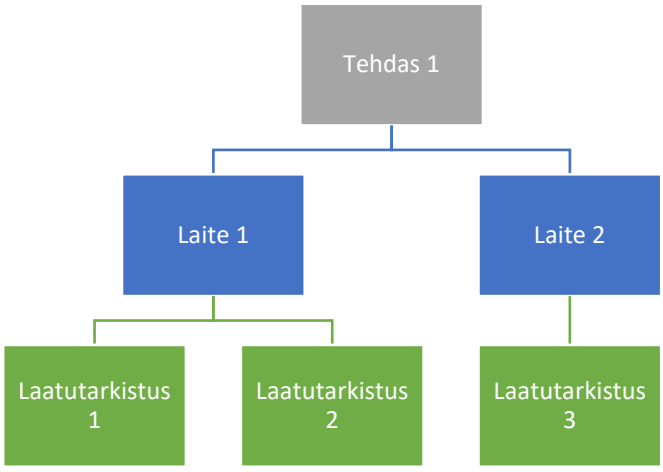
Laatutarkistuksien asetustyökalua lähdettiin toteuttamaan toimeksiantajan MarvelApp-rautalankamallin pohjalta (ks. Kuvio 7). Rautalankamallia ei noudatettu aivan täydellisesti ja toimeksiannon edetessä päädyttiin yhdessä toimeksiantajan kanssa poikkeamaan suunnitelmasta joissakin osa-alueissa. Toiminnaltaan uuden työkalun tuli täyttää samat toiminnot kuin alkuperäinen työkalu, mutta tavoitteena oli saada käytettävyydestä selkeämpi ja intuitiivisempi. Suurimmat erot vanhan työkalun käyttöliittymän ja MarvelApp-rautalankamallin välillä olivat siinä, että uudessa käyttöliittymässä laatutarkistuksen eri asetukset olisivat säädettävissä yhdessä näkymässä, eikä useassa eri näkymässä.



Kuvio 7. Userflows-näkymä Marvelissa, josta käy ilmi mistä näkymästä pääsee minnekkin.

4.2 Toteutus

Tuloksena syntyneen sovelluksen toteutus aloitetiin toteuttamalla listaus järjestelmään tallennetuista tehtaista, jotka ovat käyttöliittymän hierarkian ensimmäinen askel. (Ks. Kuvio 8, Kuvastaa tehdas-laite-laatutarkistus hierarkiaa.)



Kuvio 8. Käyttöliittymän Tehdas-Laite-Laatutarkistus hierarkia.

Tehtaiden listaukseen käytettäviä tietoja haetaan GEMA:n tietokannasta rajapinnan kautta ja tallennetaan selaimen istunnon muistiin (session storage) ja haetaan sieltä tarvittaessa.

Esimerkki sovelluksessa käytetystä Task:sta, joka hakee tehdastiedot ja asettaa ne selaimen välimuistiin. URI johon GET-pyyntö lähetetään, koostetaan tämän sovelluksen tapauksessa muuttujasta gemeBaseUrl ja API:n konfiguroidusta endpointista ”/api/opview/getFactories”. Laitetiedot ja laatutarkistuksien tiedot haetaan samaan tapaan, mutta vaihtamalla Http.GetJsonAsync(string requestUri) vaatiman requestUri:n loppuosa.

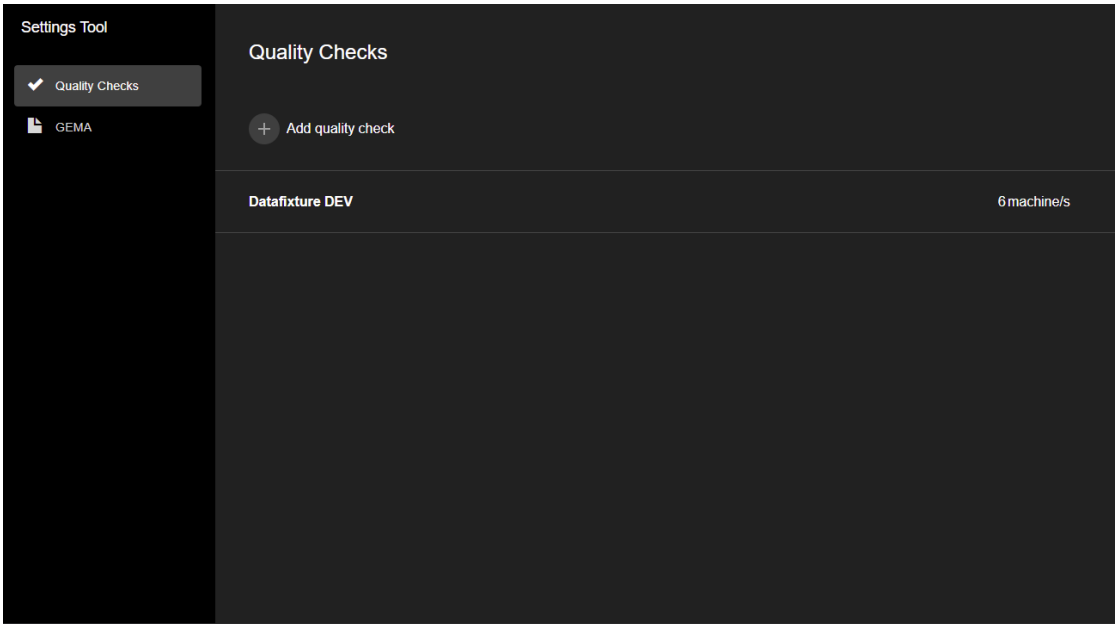
```
public async Task<List<Types.Factory>> getAllFactories(ISessionStorageService stor-
age)
    {
        var sessionStorage = storage;
        Factories = await Http.GetJsonAsync<List<Types.Factory>>(gemaBaseUrl +
"api/opview/getFactories");
        await sessionStorage.SetItemAsync("Factories", Factories);

        return Factories;
    }
```

Kun tehdastiedot on saatu haettua tietokannasta ja navigaation tilan ollessa ”factory”-tilassa generoidaan tehdaslistauksen html-rakenne alla olevalla koodilla.

```
<table class="generic-table">
    <tbody>
         @switch (status)
         {
             case "factory":
                 bool factoriesFound = false;
                 foreach (var factory in Factories)
                 {
                     factoriesFound = true;
                     int machineCount = 0;
                     <tr class="generic-table-tr" @onclick="@(() =>
                     setStatus(factory.Id, factory.Name, null))">
                         <td class="generic-table-first-td">
                         <strong>@factory.Name</strong></td>
                         @foreach (var machine in Machines)
                         {
                             if(machine.FactoryId == factory.Id)
                             {
                                 machineCount++;
                             }
                         }
                         <td>@machineCount</td>
                         <td>machine/s</td>
                     </tr>
                 }
```

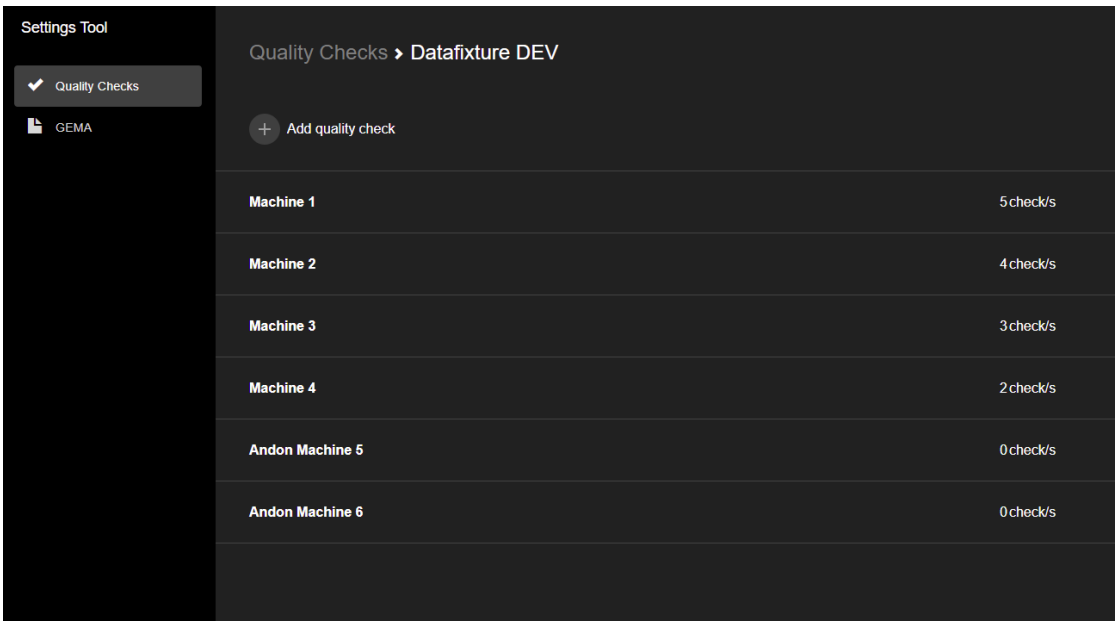
```
            if (!factoriesFound)
            {
                <tr class="generic-table-first-td">
                    <td class="generic-table-first-td">
                    Fetching factories...</td>
                </tr>
            }
            break;
…
```

Tehdastietojen perusteella luodaan järjestelmän hierarkian ensimmäinen näkymä. Näkymään listataan kaikki järjestelmään tallennetut tehdasympäristöt ja lisäksi näytetään tieto montako laitetta kyseiselle tehtaalle, on tallennettu (ks. Kuvio 9, tehdaslistaus ja laitteiden määrä). Lisäksi kaikissa näkymissä on näkyvillä yhteisenä painike ”Add quality check”, jota painamalla voidaan siirtä uuden laatutarkistuksen lisäysnäkymään (ks. Kuvio 12, Uuden laatutarkistuksen lisäysnäkymä).



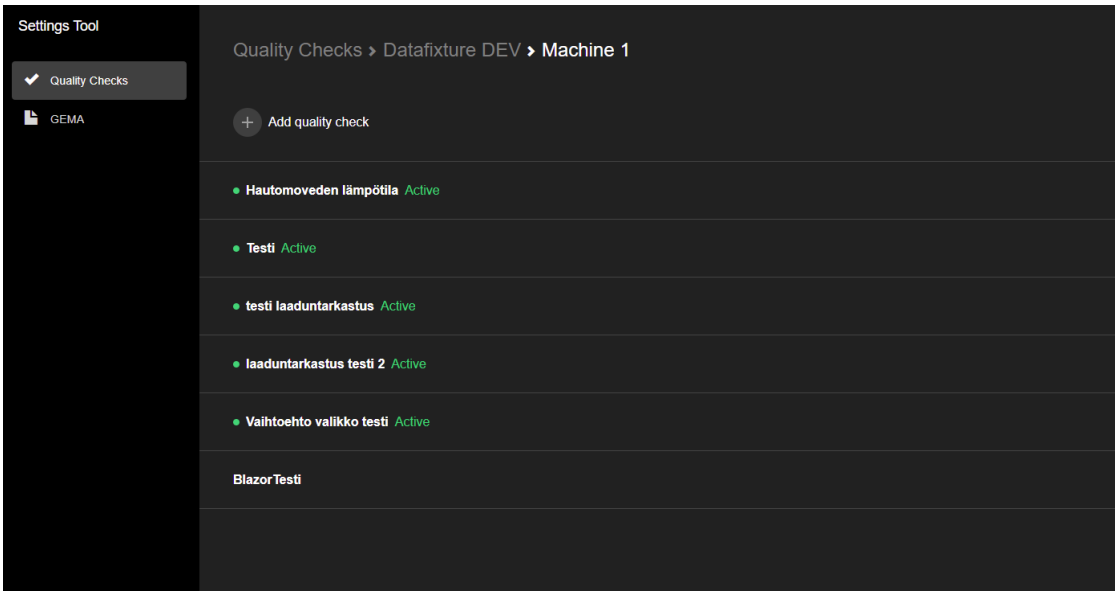
Kuvio 9. Laatutarkistuksien asetustyökalun tehdaslista.

Käyttäjän klikattua jotakin tehdasta esimerkiksi tässä tapauksessa ”Datafixture DEV” päästään näkymähierarkiassa taso syvemmälle, jossa listataan kyseisen tehtaan kaikki laitteet ja lisäksi näytetään kyseisille laitteille tallennettujen laatuarkistuksien (ks. Kuvio 10, laitelistaus ja laatutarkistuksien määrä).



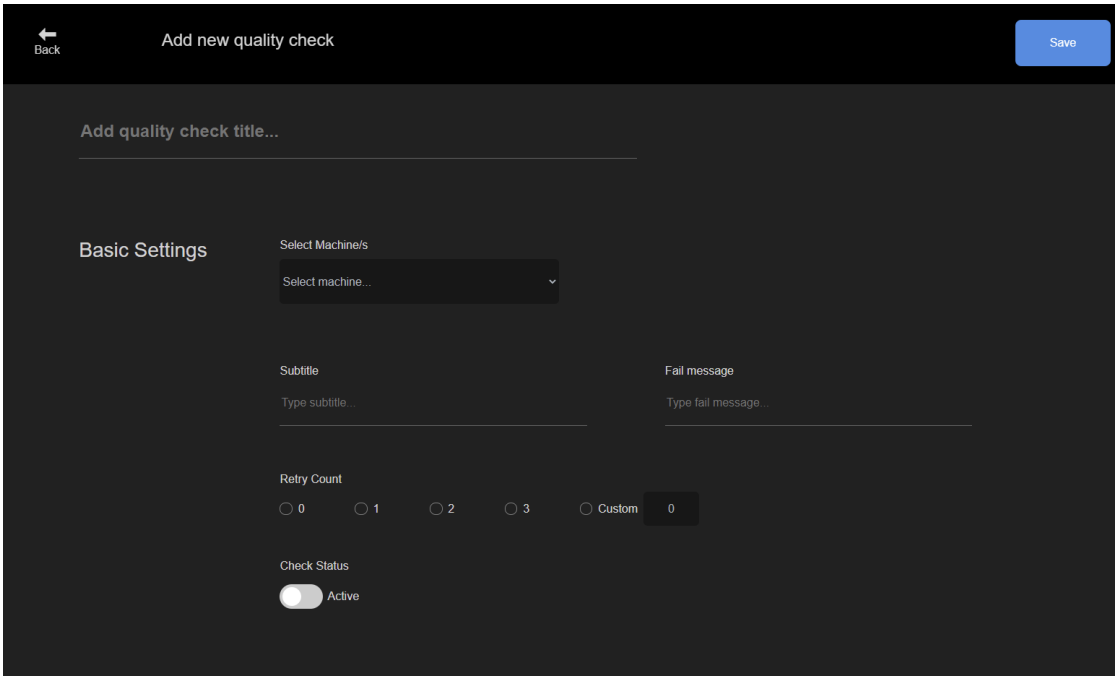
Kuvio 10. Laatutarkistuksien asetustyökalun laitelistaus.

Laitelistauksesta taas voidaan mennä askel syvemmälle tarkastelemaan kyseiselle laitteelle tallennettuja laatutarkistuksia. Näkymässä on käytetty apuna värikoodausta, jotta käyttäjän olisi helppo ja nopea nähdä yhdellä silmäyksellä, mitkä laatutarkistukset ovat aktiivisina (ks. Kuvio 11, laatutarkistuslista).



Kuvio 11. Laatutarkistuksien asetustyökalun laatutarkistuslista.

Yksi opinnäytetyön olennaisista osista oli myös testata uutta käyttöliittymää laatutarkastuksien lisäämiseen ja muokkauksen osalta. Näkymä koostuu useammasta pienemmästä kokonaisuudesta mm. perustiedot, kysymykset, suunnitelmat, ohjeet sekä validaatiot (ks. Kuvio12, jossa näkyy perustietojen osio).



Kuvio 12. Laatutarkistuksien lisäysnäkymän perustiedot.

Käyttökokemuksen kannalta näkymä, jossa on mahdollista piilottaa ei haluttuja elementtejä todettiin paremmaksi kuin vanha pieniin näkymiin jaettu ratkaisu. Vanhan käyttöliittymän käyttö koettiin hankalaksi, koska eri näkymien välillä piti hyppiä ja ei ollut mahdollista nähdä laatutarkastusta kokonaisuudessaan.

5 Tulokset

5.1 Blazor vs React

Opinnäytetyön edetessä pidemmälle todettiin toimeksiantajan kanssa, että Blazor ei ole vielä tässä vaiheessa kannattava teknologia näin suureen projektiin sekä, että Blazorin ja Reactin sekoittaminen keskenään vaikeuttaisi yleisesti projektin ylläpidettävyyttä ja tekisi sovelluksen rakenteesta epäselvän. Sovellus ei tullut aivan valmiiksi opinnäytetyön puitteissa, mutta eteni kuitenkin tarpeeksi pitkälle, että voitiin tehdä johtopäätökset siitä, että Blazoria ei oteta projektissa laajemmin käyttöön vaan jatketaan kehitystä Reactilla. Blazor voisi kuitenkin toimia hyvin pienemmissä projekteissa ja tiimeissä frontend-kehitykseen, joissa JavaScript ei ole laajalti kehittäjillä hallussa ja omaavat enemmän osaamista C#:n parissa. Blazor-projektin aloittaminen on suhteellisen suoraviivasta, jos käytössä on Visual Studio ja käyttää siinä olevia valmiita pohjia.

Blazor muistuttaa kehitysperiaatteiltaan paljon muita komponenttipohjaisia frontend-teknologioita kuten esimerkiksi Reactia. Mikäli kehittäjällä on aikaisempaa kokemusta komponenttipohjaisista frontend-teknologioista, pitäisi Blazor-sovelluksen rakenne olla helpohko ymmärtää ja yksinkertaisten komponenttien sekä sivujen toteuttaminen helppoa.

Reactin etuna Blazoriin verrattuna täytyy nostaa esille dokumentaatioiden määrä ja löydettävyys. Blazoria käytettäessä dokumentaation löytäminen on hieman hankalampaa ja usein löytyy vanhentunutta tietoa.

5.2 Käytettävyys

Käytettävyyden kannalta todettiin yhden laajemman näkymän olevan parempi vaihtoehto sille, että eri osiot olisivat valikon alla valittavissa omissa sivuissaan. Yhdistämällä näkymät loppukäyttäjän on helpompi saada kokonaiskuva laatutarkistukselle

asetetuista kysymyksistä, suunnitelmista, ohjeista ja validaatioista. Lisäksi laatutarkistuksien tekeminen yhdessä näkymässä tuntuu intuitiivisemmalta ja käyttäjältä kuluu vähemmän aikaa laatutarkistuksien luomiseen, koska ei tarvitse siirtyä eri näkymien välillä.

6 Pohdinta

Opinnäytetyön tavoitteena oli vertailla kahta erilaista frontend-teknologiaa keskenään samalla testaten miten erilainen käyttöliittymä toimisi GEMA:n laatutarkistuksien asetustyökalussa käyttäen Blazor-käyttöliittymäviitekehystä. Käyttöliittymän pohjana käytettiin toimeksiantajan valmista rautalankamallia. Samalla pohdittiin Blazorin käyttöönottoa GEMA:n osalta laajemmaltikin, mutta todettiin sen vievän liian paljon resursseja sekä, että useamman frontend-teknologian yhdistäminen yhteen sovellukseen hankaloittaisi sovelluksen ylläpidettävyyttä.

Opinnäytetyön tuloksena syntyi Blazorilla toteutettu sovellus, joka hakee GEMA:n rajapinnasta laatutarkistuksiin liittyviä tietoja ja näyttää niitä erilaisissa näkymissä sekä tallentamaan laatutarkistuksien runkoja. Lisäksi tuloksiin voidaan laskea myös itse opinnäytetyö sillä sen osia ja sisältöä voidaan mahdollisesti käyttää toimeksiantajan laatutarkistuksia käsittelevän dokumentaation sisältönä.

Opinnäytetyöprosessissa syntyneen sovelluksen jatkokehitys olisi pakollista, koska se ei nykyisellään olisi riittävä kattava tuotantokäyttöön. Sovellus ei kykenisi täyttämään vielä kaikkia ominaisuuksia, jotka ovat jo toteutettu vanhassa React-pohjaisessa toteutuksessa. Sovelluksessa on puutteita esimerkiksi uuden laatutarkistuksen eri osioiden tallentamisessa sekä olemassa olevien laatutarkistuksien muokkauksessa ja poistamisessa.

Uusien asiakkaiden ottaessa GEMA:n laatutarkistusmoduulin käyttöön voi heiltä nousta esiin paljonkin uudenlaisia määrityksiä sekä erilaisia vaatimuksia, jotka GEMA:ssa täytyisi pystyä ottamaan huomioon. Myös vanhoilta asiakkailta voi nousta tulevaisuudessa uudenlaisia vaatimuksia ominaisuuksiin.

Lähteet

ARROW GEMA. N.d. GEMA:n esittely ARROWin verkkosivuilla. Viitattu 15.11.2019. https://www.arroweng.fi/ratkaisut/gema/.

GEMA arkkitehtuuri. N.d. Kuva GEMA:n arkkitehtuurista kertovasta PowerPoint-tiedostosta. Viitattu 25.11.2020. https://confluence.pinja.com/pages/viewpage.action?pageId=3936908&preview=/3936908/3937164/Mets%C3%A4%20Wood%20-%20P%C3%A4rnu%20GEMA%20arkkitehtuuri.pptx.

HttpClient Class. N.d. Microsoftin HttpClient luokan dokumentaatio Microsoftin sivuilla. Viitattu 4.3.2020. https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=netframework-4.8.

MITTAUS, TESTAUS JA LAADUNVARMISTUS - Konsultointi, järjestelmät ja ylläpito - kasvata kilpailukykyäsi panostamalla laatuun. N.d. Info-sivu tuotannon laadunvarmistamisesta Pinjan verkkosivuilla. Viitattu 23.1.2021. https://www.pinja.com/teollisuuden-palvelut/mittaus-testaus-ja-laadunvarmistus/

Protacon on nyt Pinja – teollisuuden uudistamisella ja digitalisaatiolla vahvaa kasvua. 26.3.2020. Uutinen Pinjan kotisivuilla. Viitattu 6.10.2020. https://www.pinja.com/uutiset/protacon-on-nyt-pinja-teollisuuden-uudistamisella-ja-digitalisaatiolla-vahvaa-kasvua/.

Uudistuksia ARROW:n johtoon. 2019. Uutinen ARROW Engineering Oy:n verkkosivuilla. Viitattu 15.11.2019. https://www.arroweng.fi/uutiset/uudistuksia-arrown-johtoon/.

WebAssembly Concepts. 10.9.2020. Mozillan dokumentaatiota WebAssembylstä. Viitattu 13.10.2020. https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts.

What is Blazor and why is it so exciting. 24.3.2018. Verkkoartikkeli chrissanity.com:issa. Viitattu 3.3.2020. https://chrissainty.com/what-is-blazor-and-why-is-it-so-exciting/.

What is REST. N.d. Codeacademyn artikkeli REST-rajapinnoista. Viitattu 30.10.2020. https://www.codecademy.com/articles/what-is-rest

Yritys. N.d. ARROW Engineering Oy:n esittely ARROW Engineering Oy:n verkkosivuilla. Viitattu 15.11.2019. https://www.arroweng.fi/yritys/