LAB University of Applied Sciences

Palautekanavan toteutus web -palveluun

Case: Kemppi Oy

LAB-ammattikorkeakoulu
Insinööri (AMK), Tieto- ja viestintätekniikka
2023
Saku Väisänen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Väisänen, Saku	Opinnäytetyö, AMK	2023
	Sivumäärä	
	31	

Työn nimi

Palautekanavan toteutus web-palveluun

Case: Kemppi Oy

Tutkinto

Insinööri (AMK), Tieto- ja viestintätekniikka

Toimeksiantajan nimi, titteli ja organisaatio

Kemppi Oy

Tiivistelmä

Opinnäytetyössä toteutettiin palautekanava toimeksiantajan web -palveluun. Palautekanavan kautta käyttävät pystyvät ilmoittamaan ohjelmistovirheistä, lähettämään ehdotuksia, kysymään kysymyksiä ja antamaan yleistä palautetta. Palautekanavan lisääminen parantaa kommunikaatioväylää palvelun käyttäjien ja kehittäjien välillä. Toimeksiantajana toimi Kemppi Oy.

Palautekanavan käyttöliittymä toteutettiin käyttämällä Angular kehystä ja taustajärjestelmä toteutettiin käyttämällä Amazonin pilvipalveluita, kuten AWS Lambda ja S3, taustajärjestelmän puolella. Koodi kirjoitettiin TypeScript kielellä.

Lopputuloksena oli onnistunut palautekanava, joka sisältää kaikki ominaisuudet, joita toimeksiantaja alun perin toivoi ja mitä lisättiin kehityksen aikana. Käyttäjäkokemuksia palautekanavasta ei vielä ole ehditty keräämään.

Asiasanat

Angular, TypeScript, Käyttöliittymä

Abstract

Author(s)	Type of Publication	Published
Väisänen, Saku	Thesis, UAS	2023
	Number of Pages	
	31	

Title of Publication

Implementing a feedback channel to a web service

Case: Kemppi Oy

Name of Degree

Bachelor of Engineering, Information and Communications Technology

Name, title and organization of the client

Kemppi Oy

Abstract

The goal of this thesis was to design and implement a feedback channel to an existing web service run and operated by the client. The goal of the feedback channel was to enable users of the web service to send bug reports, suggestions, ask questions or to give out general feedback. The motivation to add the feedback channel was to improve the communication channel between users and developers. The client for this thesis was Kemppi Oy.

The front end of the feedback channel was implemented using Angular. The back end was done using Amazon's cloud services like AWS Lambdas and S3. The programming language of choice was TypeScript.

The result of the thesis is a functional feedback channel with all the features that were requested originally or later. Users have not had the chance to use the feedback channel yet due to prolonged development.

Keywords

Angular, TypeScript, Front end

Sisällys

REST	Arkkitehtuurityyli verkko-ohjelmistoille
JSON	Tiedostomuoto JavaScript objekteille
Screenshot	Ruudunkaappaus
URL	Verkko-osoite
Template	Sivupohjatiedosto
Metodi	Funktio, joka on osa luokkaa
Funktio	Koodi segmentti, jota voidaan kutsua
Asynkroninen	Ei-reaaliaikainen
Back end	Taustajärjestelmä
Front end	Käyttöliittymä

1 Johdanto

Tämän työn tavoitteena on toteuttaa palautekanava Kemppi Oy:n WeldEye -palvelulle. Palautekanavan avulla WeldEye -palvelun käyttäjät pystyvät palvelusta suoraan ilmoittamaan ohjelmistovirheistä, ehdottamaan parannuksia tai antamaan yleisesti palautetta palvelusta. Palautekanavan tarvitsee olla helppokäyttöinen ja selkeä. Työn toimeksiantaja on Kemppi Oy.

Kemppi Oy on kaarihitsausteollisuudessa toimiva suomalainen yritys. Kemppi toimittaa hitsauslaitteita, hitsaukseen liittyviä lisä- ja suojavarusteita, hitsaushallintaohjelmistoratkaisuja sekä asiantuntijapalveluita. Kemppi on perustettu vuonna 1949. Kemppi työllistää lähes 800 asiantuntijaa ja operoi 16 eri maassa. Kempin pääkonttori on Lahdessa.

WeldEye on Kemppi Oy:n kehittämä hitsaustuotannonhallinta ohjelmisto. WeldEye ominaisuuksiin kuuluu prosessienhallinta, hitsaajien ja tarkastajien pätevyyksien hallinta, hitsausstandardien hallinta, hitsauksen dokumentointi ja raportointi.

Tämä työ kattaa niin käyttöliittymän kuin taustajärjestelmän, mutta tämä raportti painottuu käyttöliittymän toteutukseen. Taustajärjestelmää käsitellään pääosin vain suunnittelutasolla ja vain niillä osin kuin se on oleellinen käyttöliittymän toteutuksen selittämiseen.

Tietopohja osuudessa käydään läpi projektille olennaisia asioita, kuten käytettäviä kehyksiä, kirjastoja ja muita teknologioita. Suunnittelu osuudessa käydään läpi projektille olennaista tietoa, ja pohditaan toteutukselle oleellisia asioita. Toteutusosuudessa keskitytään käyttöliittymän toteutukseen, siinä kohdattuihin haasteisiin ja lopputuotokseen. Yhteenvedossa kerrataan mitä tehtiin, miten onnistuttiin ja miten voitaisiin jatkaa.

2 Tietopohja

2.1 Typescript

TypeScript on JavaScript yhteensopiva ohjelmointikieli. Tarkemmin ilmaistuna se on JavaScriptin yläjoukko (engl. superset), eli TypeScript sisältää JavaScriptin ominaisuudet. TypeScriptiä ei yleensä ajeta suoraan sellaisenaan, vaan se käännetään takaisin JavaScriptiksi. (Lease 2018.)

TypeScriptin etu JavaScriptiin nähden on parannettu tyypitys. TypeScriptissä muuttujat voidaan staattisesti tyypittää ja luoda rajapintoja (engl. interface). Staattinen tyypittäminen parantaa koodin huollettavuutta ja vähentää mahdollisuuksia virheisiin, etenkin suuremmissa projekteissa. (Lease 2018.)

2.2 Angular

Angular on verkkosovelluskehys, jonka tavoite on mahdollistaa tehokkaiden ja kehittyneiden yksisivuisten verkkosovelluksien luominen. Angular sovellusten kehittämiseen käytetään HTML ja TypeScriptiä. (Thakur 2021.)

Suoritetun kyselyn mukaan Angular on verkkosovelluskehyksistä ja verkkosovellusteknologioista viidenneksi suosituin kyselyn vastaajien keskuudessa, 17.4% suosiolla. Rajattaessa kyselyn vastanneet ammatikseen ohjelmoiviin, on Angular sijalla neljä, 19.89% suosiolla. (StackOverflow 2023.)

2.2.1 Historia

Angular sai alkunsa vuonna 2009 Miško Heveryn sivuprojektina. Angularin tavoite oli yksinkertaistaa verkkosovellusten kehitystä. Myöhemmin vuonna 2010 Angular, silloin nimellä AngularJS, julkaistiin avoimena lähdekoodina. (Gudelli 2019; Gavigan 2018.)

Vuonna 2016 Angularista julkaistiin täysin uudelleenkirjoitettu versio, Angular 2, joka nykyään tunnetaan nimellä Angular. AngularJS kehyksellä luotuja verkkosovelluksia ei ole mahdollista päivittää Angular 2 kehykselle. (Gavigan 2018.)

2.2.2 Arkkitehtuuri

Angularin arkkitehtuurin voidaan jakaa muutamaan pääosaan, moduulit, palvelut ja komponentit. Moduulit toimivat säiliöinä muille arkkitehtuurin osille, jotta Angular projektin voidaan jakaa useampaan helpommin huollettavaan osaan. Jokaisessa projektissa on vähintään yksi moduuli, niin kutsuttu ”root” -moduuli, yleensä nimellä ”AppModule”. Jos moduu-

lin sisältämiä osia halutaan käyttää jossakin toisessa moduulissa, täytyy se moduuli sisällyttää siihen moduuliin missä osia halutaan käyttää. (Thakur 2021.)

Palvelut ovat luokkia, jotka suorittavat logiikka tai sisältävät dataa, joka ei suoranaisesti liity mihinkään yhteen komponenttiin tai jonka halutaan olla jaettavissa usean eri komponentin välillä (Garg 2019). Palvelut ovat Angularissa oletuksena ”ainokaisia” (engl. singleton), eli luokasta on sovelluksen elinkaaren aikana olemassa vain yksi instanssi (BairesDev). Palveluita käytetään usein komponenttien väliseen kommunikointiin. Palvelut tuodaan komponentteihin käyttämällä riippuvuusinjektiota (engl. dependency injection) (BairesDev).

Angular-komponentit koostuvat kolmesta osasta, komponenttiluokka, template- ja tyylitiedosto (Sharma 2021). Yhdessä nämä kolme osaa muodostavat näkymän, joka esitetään käyttäjälle. Komponenttiluokka sisältää näkymän logiikan ja datan, kun taas template on näkymän pohja, johon data sidotaan. Tyylitiedosto sisältää tyylittelyn samaan tapaan kuten perinteisillä verkkosivuilla. Templatet ovat HTML-tiedostoja, joissa on lisänä Angularin template-syntaksi, jonka avulla voidaan sitoa data komponenttiluokasta templateen. (Thakur 2021.)

Komponentti voi olla itsessään kokonainen sivu tai vain yksi elementti sivulla. Komponentin näkymä ja logiikka ovat oletuksena kapseloitu, jotta komponentit olisivat uudelleen käytettävissä. Kapseloinnilla varmistetaan, että tyylittely, logiikka tai data eivät komponenttien välillä vaikuta toisiinsa arvaamattomalla tavalla. (Sharma 2021).

Datan sidonta mahdollistaa kommunikaation templaten ja komponentin välillä. Datan sidontaa on useaa eri tyyppiä, interpolaatio, ominaisuussidonta, tapahtumasidonta ja kaksisuuntainensidonta. Interpolaatio sitoo jonkin komponentin muuttujan arvon suoraan templateen, nimensä mukaisesti arvo interpoloidaan templateen sisään. Ominaisuus- ja tapahtumasidontaa käytetään elementtien väliseen datan välitykseen. Kaksisuuntainensidonta on yhdistelmä tapahtuma- ja ominaisuussidontoja, sillä voidaan kaksisuuntaisesti sitoa muuttujan arvo elementtien välillä. (Yewale 2020.)

2.2.3 Angular Material

Material Design on Googlen kehittämä suunnittelukieli. Ensimmäinen versio Material Designista julkaistiin vuonna 2014. Sen tarkoitus on parantaa ja yhtenäistää käyttökokemusta eri laitteiden välillä. (Chapman.)

Angular Material on käyttöliittymä komponentti kirjasto Angularille. Angular Material on implementaatio Googlen kehittämästä Material Design tyylistä. Angular Material tarjoaa

laajan valikoiman valmiita komponentteja, jotka on helppo lisätä projektiin. Angular Materialille voi määrittää väriteeman, jota kaikki sen komponentit seuraavat. (Salazar 2020.)

2.2.4 Angular CDK

Angular CDK (Component Development Kit) sai alkunsa Angular Materialin kehityksen aikana. Materialin kehittäjät huomasivat, että eri komponentit jakavat samanlaisia käyttäymismalleja keskenään. Lopulta nämä yhteiset käyttäytymismallit muokattiin helpommin uudelleenkäytettäviksi ja refaktorointiin omaan komponentti kirjastoonsa. (Elbourn 2018.)

Angular CDK pitää sisällään alipaketteina eri tarpeisiin liittyviä käyttäytymismalleja. Näitä ovat mm. a11y, joka keskittyy saavutettavuuden parantamiseen, overlay, joka tarjoaa rajapinnan päälle piirrettäviin elementteihin kuten dialogeihin sekä layout, joka tarkoitettu skaalautuvuuden parantamiseen eri näyttökoille. (Elbourn 2018.)

2.3 RxJS

RxJS on kirjasto, joka implementoi ReactiveX'n JavaScriptille. RxJS on sisällytetty Angulariin yhtenä ydinkirjastoista. RxJS käytetään asynkroonisien ja tapahtuma (engl. event) pohjaisten ohjelmien luomiseen. (Slack 2021a.)

RxJS:n ydin konseptit ovat tarkkailtava, subjekti ja tilaus. Tarkkailtava on kokoelma arvoja ajassa säiliönä asynkronisesti haetulle datalle. Tarkkailtavilla voidaan korvata JavaScriptin promise -rakenteet. Tarkkailtava ei itsessään tee mitään, ennen kuin se tilataan. Tilattaessa tarkkailtavalle annetaan takaisinkutsufunktio, jota tarkkailtava kutsuu saadessaan uuden arvon. Tarkkailtavia voidaan luoda monin eri tavoin, yksi niistä tavoista on subjekti. Subjekti on kuin tarkkailtava, mutta sillä on 3 lisämetodia, next, error ja complete. Next metodilla voidaan subjektille antaa uusi arvo, jolla sitten subjekti kutsuu tilauksia. Subjekti voidaan tilata suoraan tai muuntaa tarkkailtavaksi. (Slack 2021b.)

2.4 HTML5 Canvas

Canvas (suom. piirtoalue) on HTML elementti grafiikan, staattisen sekä interaktiivisen, esittämiseen verkkosivulla. Canvas elementillä on oma sisäinen resoluutionsa, joka on määritettävissä elementin rajapinnan kautta. Elementin rajapinta tarjoaa eri kontekstitiloja, joiden avulla elementtiin voidaan piirtää. Näitä ovat mm. 2D, WebGL ja WebGPU. (WHATWG.)

2D kontekstitila on rasterigrafiikka rajapinta, jota voidaan käyttää JavaScriptin avulla grafiikan piirtämiseen. Canvas elementti toimii välittömässä tilassa. Tässä tilassa canvas

elementtiin piirrettyjä pikseleitä ei tallenneta, vaan jokainen kehys piirretään aina alusta. (Fulton & Fulton 2011.)

2.5 MediaDevices API

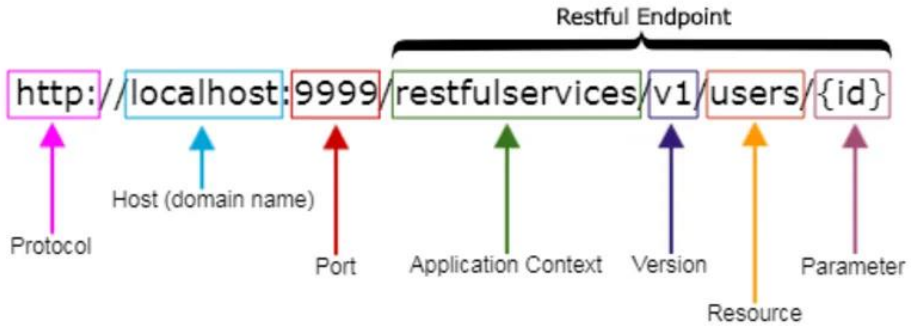
MediaDevices on HTML5 rajapinta medialähteiden käsittelyyn JavaScriptissä. Sen avulla voidaan (esimerkiksi) piirtää web kameran syöte HTML5 canvas elementtiin. MediaDevices rajapinta vaatii HTTPS yhteyden selaimen ja web-palvelimen välille. (Bidelman & Dutton.)

Ennen MediaDevices rajapintaa medialähteiden käsittely oli haastavaa. Kehittäjät joutuivat turvautumaan selainliitännäisiin reaaliaikaisen medialähteiden käsittelyn kanssa. Median kaappaus oli mahdollista ennenkin, mutta se oli rajattu median kaappaamiseen ennen kuin, sitä oli mahdollista käsitellä. (Bidelman & Dutton.)

2.6 RESTful APIs

REST on arkkitehtuurityyli verkko-ohjelmistoille, ja se tulee sanoista REpresentational State Transfer (suom. edustuksellinen tilasiirto). REST on tilaton (stateless), eli se ei tallenna tilaa pyyntöjen välissä, vaan tila välitetään jokaisen pyynnön yhteydessä. REST tyylin mukaisesti, taustapalvelun objekteja käsitellään resursseina, joita voidaan luoda (Create), lukea (Read), päivittää (Update) ja poistaa (Delete). Näistä perustoiminnoista puhutaan joskus lyhenteellä CRUD. (Kumar 2019.)

Kuvassa (Kuva 1) näkyy REST tyylin mukainen nimeämisskeema. Resurssi osuus url -osoitetta täytyy olla monikossa REST tyyliä seuratessa, ja parametrin avulla voidaan url -osoite kohdistaa tiettyyn objektiin. Esimerkiksi, käyttäjiin (engl. users) liittyvät toiminnot tapahtuisivat /api/users osoitteen kautta, kun taas yhteen käyttäjään, jonka id on 5, kohdistuva toiminto olisi saatavilla /api/users/5 osoitteen kautta. (Pethiyagoda 2022.)



Kuva 1. REST Nimeämisskeema (Pethiyagoda 2022)

2.7 S3

Amazon Simple Storage Service, tai yleisemmin S3, on tallennuspalvelu objektien säilyttämiseen. S3:ssa objektit tallennetaan ämpäreihin (engl. Bucket). Ämpäri on säiliö objekteille samaan tapaan kuin kansiot Windowsissa. (Venati 2019.)

Objektit ovat rakenteetonta dataa. Rakenteettomalla datalla tarkoitetaan dataa, joka ei seuraa mitään data mallia tai skeemaa. Kuvat, dokumentit, multimedia ja teksti ovat vain muutama esimerkki, mitä rakenteeton data voi olla. Rakenteeton data tallennetaan yleensä sen natiivissa tiedostotyypissään. (Mongodb.com; Elastic.co.)

2.8 Elasticsearch

Elastisearch on hakumoottori, jolla voidaan suorittaa sana hakuja nopeasti datan määrästä huolimatta. Tämä on mahdollista, koska Elasticsearch indeksoi datan tallennushetkellä. Datan indeksidata tallennetaan JSON muodossa. Elasticsearch toimii rakenteellisen, puolirakenteettoman tai täysin rakenteettoman datan kanssa. (Velotio Technologies 2019.)

Jotta data voidaan indeksoida, täytyy se ensin kartoittaa (engl. mapping). Kartoittamisella tarkoitetaan datan kuvaamista Elasticseachin käyttämillä datatyypeillä. Kartoittamalla datan voi ohjelmoija kertoa Elasticsearchille mitä data kenttiä indeksoida ja minkälaista dataa kentät sisältävät. Jos kartoitusta ei anneta, Elasticsearch yrittää kartoittaa datan dynaamisesti. (Velotio Technologies 2019)

2.9 AWS Lambda

AWS (Amazon Web Services) Lambda on Amazonin versio FaaS (Function as a Service; suom. Funktio palveluna) arkkitehtuuri mallista. Ajatus FaaS mallin takana on infrastruktuurin yksinkertaistaminen kehittäjän näkökulmasta. Kehittäjän ei tarvitse huolehtia palvelimen ylläpidosta tai palvelinohjelmiston arkkitehtuurista. Kehittäjä luo vain tarvitsemansa toiminnon, ja koodi suoritetaan pilvessä kuten kehitysympäristössä. (Rehemägi 2021.)

Lambdat suoritetaan mikro-virtuaalikoneissa Firecracker nimisen virtualisointialustan (engl. Hypervisor) alla. Firecracker on AWS:än kehittämä avoimen lähdekoodin virtualisointiteknologia FaaS mallia varten. (Rehemägi 2021.)

3 Suunnittelu

3.1 Vaatimukset

Toimeksiantaja on asettanut ominaisuusvaatimuksia, jotka palautekanavan tulee täyttää. Osa vaatimuksista ovat yleisiä ja hyvinkin tyypillisiä palautekanaville tai ohjelmistovirheilmoitus -työkaluille, kuten käyttäjän kirjoittama vapaamuotoinen teksti tai kuvan liittäminen palautteeseen. Toiset vaatimukset ovat hieman monimutkaisempia, kuten ruudunkaappaus selaimesta käsin. Palautekanavan toivotaan olevan modaali, eli dialogi -tyyppinen. Palautekanavan dialogissa on oltava mahdollisuus lähettää uusia palautteita, sekä tarkastella vanhoja palautteita. Käyttäjä voi tarkastella oman organisaationsa sisällä lähetettyjä palautteita vapaasti. Järjestelmän ylläpitäjä voi tarkastella kaikkia lähetettyjä palautteita ja muuttaa palautteiden tilaa.

Palautekanavan taustapalveluilta koskevat vaatimukset ovat avoimemmat. Taustapalveluissa on käytettävä toimeksiantajan jo käyttämiä teknologioita, lisäksi uusista palautteista on saatava ilmoitus Slack -viestintäpalvelun kanavalle, sekä palautteen tilan muuttuessa palautteen antajalle on lähetettävä sähköposti-ilmoitus muutoksesta, jos palautteen antaja on näin valinnut.

3.2 Sisäiset komponentit

Käytettävissä on valikoima sisäisiä komponentteja sekä palveluita. Komponentit suurimmalta osin pohjautuvat Angular Material UI komponentteihin tai Angular CDK käyttäytymismalleihin. Näitä sisäisiä komponentteja kutsutaan KemppiShared komponenteiksi, tästä eteenpäin referoituna KS-komponentit.

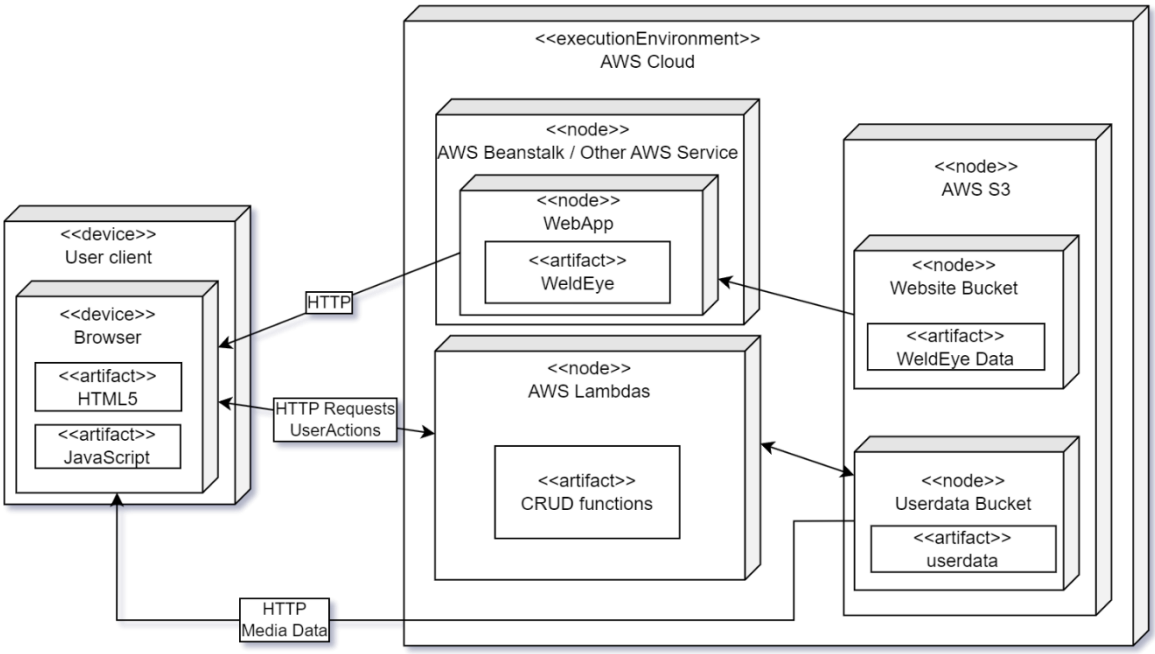
Datan syöttöön tarkoitetuissa KS-komponenteissa on read-only (suom. vain luku) tila, joka korvaa datan syöttö kentän puhtaalla tekstillä. Kentän korvaaminen puhtaalla tekstillä sallii helpomman tyylittelyn sekä tulostaessa luotettavamman käyttäytymisen.

WeldEyessa on olemassa olevia palveluita, joita voidaan hyödyntää. Esimerkiksi, tiedoston tai kuvan lähettämiselle on olemassa palvelu ja taustajärjestelmän palvelu, joten niitä ei tarvitse toteuttaa uusiksi.

3.3 Taustajärjestelmä

WeldEye on Amazonin pilvessä, joten taustajärjestelmä on rakennettava käyttäen Amazonin tarjoamia palveluita. Taustajärjestelmää rakentaessa on otettava huomioon WeldEyen olemassa oleva arkkitehtuuri ja toiminnot.

Kuviossa (Kuvio 1) kuvataan WeldEyen perustoimintojen suoritusarkkitehtuuri. Kuvio on huomattavasti yksinkertaistettu. Palautekanavan taustajärjestelmän data on tallennettava Userdata S3 ämpäriin. Taustajärjestelmän toiminnot on toteutettava AWS Lambda funktioina.



Kuvio 1. Yksinkertaistettu käyttöönottokaavio WeldEyesta

Toiminnot hajautetaan omiin lambdoihinsa. Palautekanavan vähimmäistoimivuus vaatii vain ns. CRUD (Create, Read, Update, Delete) toiminnot, näiden perustoimintojen lisäksi sähköpostin lähetys hajautetaan omaan lambdaansa. Tikettien hakeminen, read, toteutetaan myös useammalla lambdalla. Syynä tähän on se, että ylläpitäjän ja käyttäjän oikeustasolla suoritettavat lambdat halutaan erottaa toisistaan. Semanttisesti, lambdat noudattavat RESTful ideologiaa. Esimerkiksi datan hakeminen suoritetaan käyttämällä GET pyyntöä, ja lambdan osoite muodostetaan käsiteltävän asian substantiivista. Hakulambdan pyyntö olisi GET /api/feedback-tickets haettaessa useita tikettejä. Jos haetaan vain yhtä tikettiä ID:n perusteella, pyyntö olisi muodossa GET /api/feedback-tickets/TICKET-XXXXX

Hakulambdan on pystyttävä suodattamaan tuloksia eri kriteerien perusteella, esimerkiksi annetun aikavälin tai tiketin statuksen perusteella. Jotta suodatus voidaan toteuttaa tehokkaasti, on tiketit indeksoitava Elasticsearchiin, sillä Elasticsearch pystyy suodattamaan ja järjestelemään tiketit paljon tehokkaammin kuin JavaScriptillä toteutettu suodatin. Suodattimien välitys lambdalle onnistuu helpoiten lähettämällä ne samassa muodossa kuin niitä käytetään taustajärjestelmän puolella. Suodattimet välitetään taustajärjestelmään

merkkijonoksi muutettuna JSON objektina url-osoitteen ”query string” (suom. kysely merkkijono) osassa.

Tiketit tallennetaan S3 ämpäriin käyttäjän organisaation hakemistoon. Ennen ämpäriin lisäämistä, lambdan on täydennettävä puuttuvat tiedot tikettiin, kuten ID, aikaleima sekä status. Määrittämällä tietoja vasta taustajärjestelmän puolella voidaan varmistaa, että nämä tiedot ovat oikein ja estää vahinkoja, joissa saattaisi tallentua päällekkäisiä tietoja.

Tikettien päivittämiseen voidaan käyttää joko PUT tai PATCH pyyntöä. Semanttisesti ero näiden kahden välillä on yleensä se, että PUT pyynnön mukana lähetetään koko päivitettävä objekti, kun taas PATCH pyynnön mukana lähetetään vain muuttunut data. Molemmissa tavoissa on etunsa ja haittansa. Tikettien päivittämisessä on huomioitava myös tarve kirjoittaa eri organisaatioiden välillä, sillä järjestelmän ylläpitäjät ovat omassa organisaatiossaan, joten lambda on suojattava väärinkäytöltä tarkastamalla, onko lambdan kutsujalla järjestelmän ylläpitäjän roolia.

Sähköpostit lähetetään HTML muodossa. Jotta HTML koodi saadaan generoitua jokaiselle tiketille yksilöllisesti, on hyödynnettävä templating kirjastoa. Lyhyesti selitettynä templating kirjasto täyttää HTML tiedoston dynaamisesti oikealla datalla.

3.4 Käyttöliittymä

Käyttöliittymää suunniteltaessa on huomioitava käyttöliittymän selkeys ja käytettävyys. Käytettävyyteen vaikuttaa esitetyn tiedon ja valintojen määrä sekä näiden asettelu. Palautteen lähettämisestä on tehtävä helppoa sekä kitkatonta, jotta käyttäjä kokee palautteen jättämisen aikansa arvoiseksi. Palautteiden selaamisen kannalta palautteiden suodatettavuus on tärkeä. Palautteiden lukemisen osalta etenkin asettelu on tärkeää, sillä informaatiotiheys vaikuttaa tiedon sisäistämiseen. Käyttöliittymä on syytä hajauttaa useisiin komponentteihin, jotta koodi pysyy huollettavana.

Käyttöliittymä on jaettavissa kahteen selkeään toiminnallisuuteen. Palautteiden luominen sekä palautteiden selaaminen ja lukeminen. Molemmat toiminnallisuudet voidaan toteuttaa saman dialogin kautta jakamalla toiminnallisuudet omille välilehdilleen. Vaikkakin luominen ja lukeminen ovat eri toiminnallisuuksia voidaan ne yhdistää yhteen komponenttiin. Palautteiden selaaminen tarvitsee oman komponenttinsa.

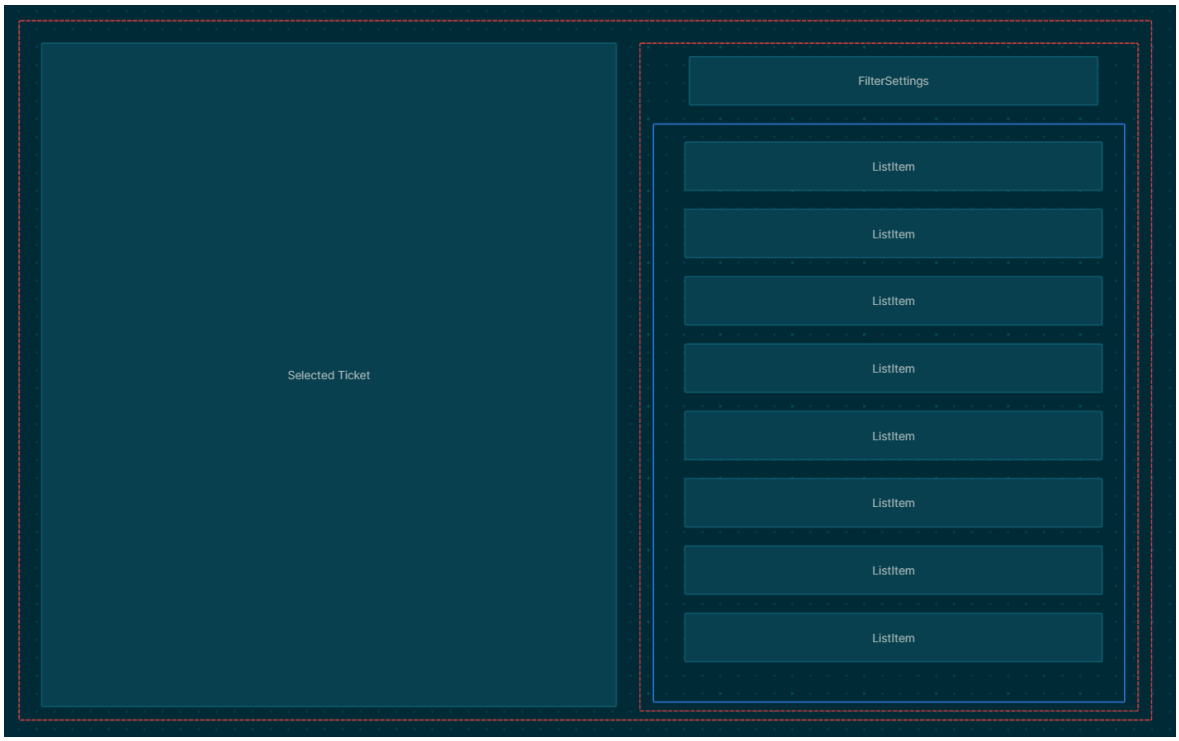
3.4.1 Asettelu

Kuvasta (Kuva 2) nähdään palautelomakkeen suunnittelun lähtöpiste, joka on hyvin yksinkertainen. Punaisella katkoviivalla merkityt alueet ovat suunniteltuja div-elementtejä tai

vastaavia. Kiinteät nelikulmiot ovat näkyviä elementtejä. Ladattujen kuvien säiliö on suunniteltu olemaan horisontaalisesti skrollattava, jotta käyttäjä voi lisätä haluamansa määrän kuvia. Lomakkeessa ei ole itsessään lähetys painiketta, sillä se tulee sijaitsemaan dialogi ikkunan ”actions” alueella.

Kuva 2. Palautelomakkeen suunnittelu

Kuvasta (Kuva 3) nähdään, palautteiden historia välilehti on suunniteltu listaksi. Listasta klikkaamalla aukeaa valittu palautetiketti listan vierelle. Listaa voidaan suodattaa, ja suodatus valinnat tulevat sijoittumaan listan yläpuolelle.



Kuva 3. Palautehistoria elementti

3.4.2 Tietosuoja

Palautteen luomisen yhteydessä kerätään automaattisesti tietoja, kuten miltä sivulta käyttäjä raportoi ongelmasta, ja mitä lisenssejä käyttäjällä on raportoinnin hetkellä. Keräämällä tietoja ongelmat ja ohjelmistovirheet voidaan helpommin jäljittää. Tietoja kerättäessä on tärkeää ottaa huomioon tietosuojaan ja tietojen käsittelyyn liittyvät seikat.

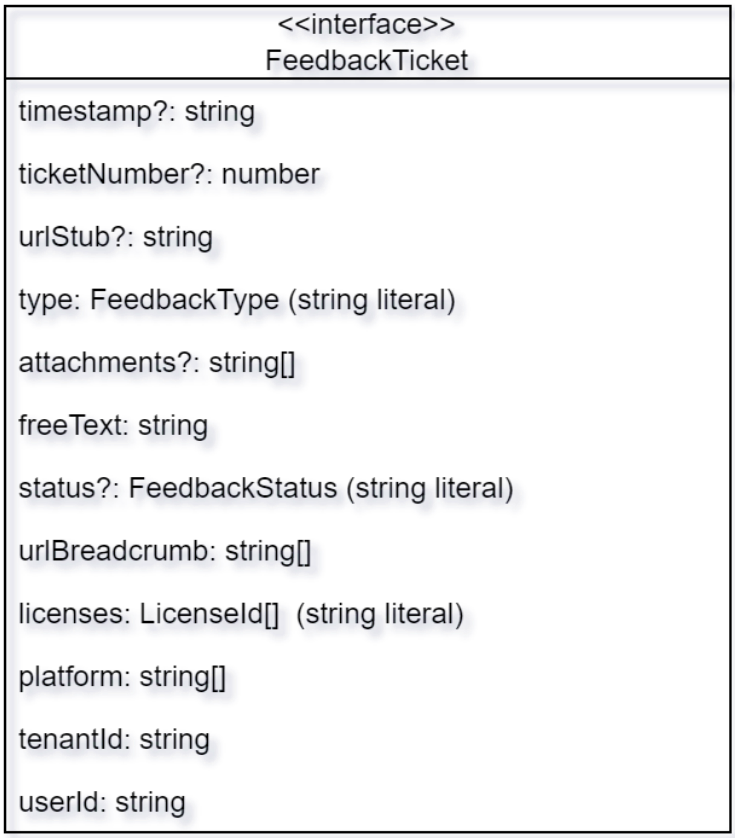
Tietojen käsittelyyn on löydyttävä oikeusperuste. Tietoja voidaan kerätä ja käsitellä oikeutetun edun perusteella, mutta se vaatii tarkkaa harkintaa, jossa käyttäjän oikeuksia on punnittava tarkoin rekisterinpitäjän etuja vastaan. (tietosuoja.fi.)

Toinen vaihtoehto on pyytää käyttäjän suostumusta. Suostumuksen vaatimukset ovat, että se on yksilöity, tietoinen, vapaaehtoinen ja yksiselitteinen tahdonilmaisu. Tietoisella tarkoitetaan, että käyttäjä ymmärtää miten ja mihin hänen tietojansa käytetään. Vapaaehtoinen merkitsee sitä, että käyttäjä ei voi kieltäytyessään olla huonommassa asemassa. Yksiselitteisellä tahdonilmauksella tarkoitetaan, että käyttäjä ei voi vahingossa antaa suostumusta, eli käytännön kannalta suostumus ei saa olla valmiiksi rastitettu ruutu. Suostumuksen pyytäminen on oleellista silloin kun käyttäjän tietojen käsittelyn tarkoitus tai tapa on muuttunut aikaisemmasta suostumuksesta. (tietosuoja.fi.)

3.5 Tietomalli

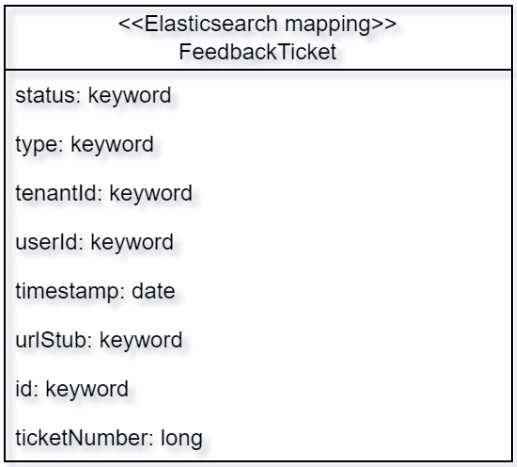
Tietomallia suunniteltaessa on tärkeää ottaa huomioon kaikki datan ominaisuudet pitäen mielessä tallennusjärjestelmän tietotyypit. Palautekanavan tapauksessa data tallennetaan S3 ämpäriin JSON objektina, joten data voitaisiin tallentaa sellaisenaan. On kumminkin huomioitava, että data on oltava myös indeksoitavissa Elasticsearchiin, joten indeksoitavien ominaisuuksien osalta rajoituksena on Elasticsearchin tukemat tietotyypit.

Kuviossa (Kuvio 2) nähdään suunniteltu tietomalli palautetiketeille. Se sisältää vaaditut kentät palautekanavan toteuttamiselle. Tyyppi, status ja lisenssi kentät ovat ”string literal” tyyppejä, eli kenttien tieto on string muodossa, mutta niiden hyväksyttävät arvot on rajattu. Palautetiketit numeroidaan lineaarisesti kaikkien organisaatioiden kesken. Tikettiin on tarkoitus myös tallentaa sivun osoite mistä se lähetetään urlBreadcrumb kenttään, tallentaminen suoritetaan segmentoimalla url-osoite osiin ja tallentamalla osat taulukkona. UrlStub kenttään tallennetaan urlBreadcrumb kentän ensimmäinen alkio, jotta se voidaan indeksoida. Tikettiin lisätyt kuvat voidaan tallentaa omaan hakemistoonsa, joten vain kuvien tiedostonimet ovat tarpeellisia sisällyttää tikettiin. Kuvien tiedostonimet tallennetaan attachments kenttään string taulukkoon.



Kuvio 2. FeedbackTicket tietomalli

Kuviossa (Kuvio 3) on nähtävillä suunniteltu kartoitus Elasticsearch indeksointia varten. Keyword (suom. avainsana) on Elasticsearchissa tekstityyppinen arvo, joka arvioidaan aina kokonaisuudessaan, joten keyword on oikea valinta kentille, jotka ovat uniikkeja, kuten id-kentät, tai sisältävät ennalta määritettyjä arvoja, kuten status ja type kentät. Date-tyyppi hyväksyy, joko unix-aikaleiman tai ISO 8601 muotoisen aika-arvon. TicketNumber kenttä on parasta määrittää numeeriseen muotoon, jotta voidaan hyödyntää numerokentille tarkoitettuja Elasticsearchin ominaisuuksia.



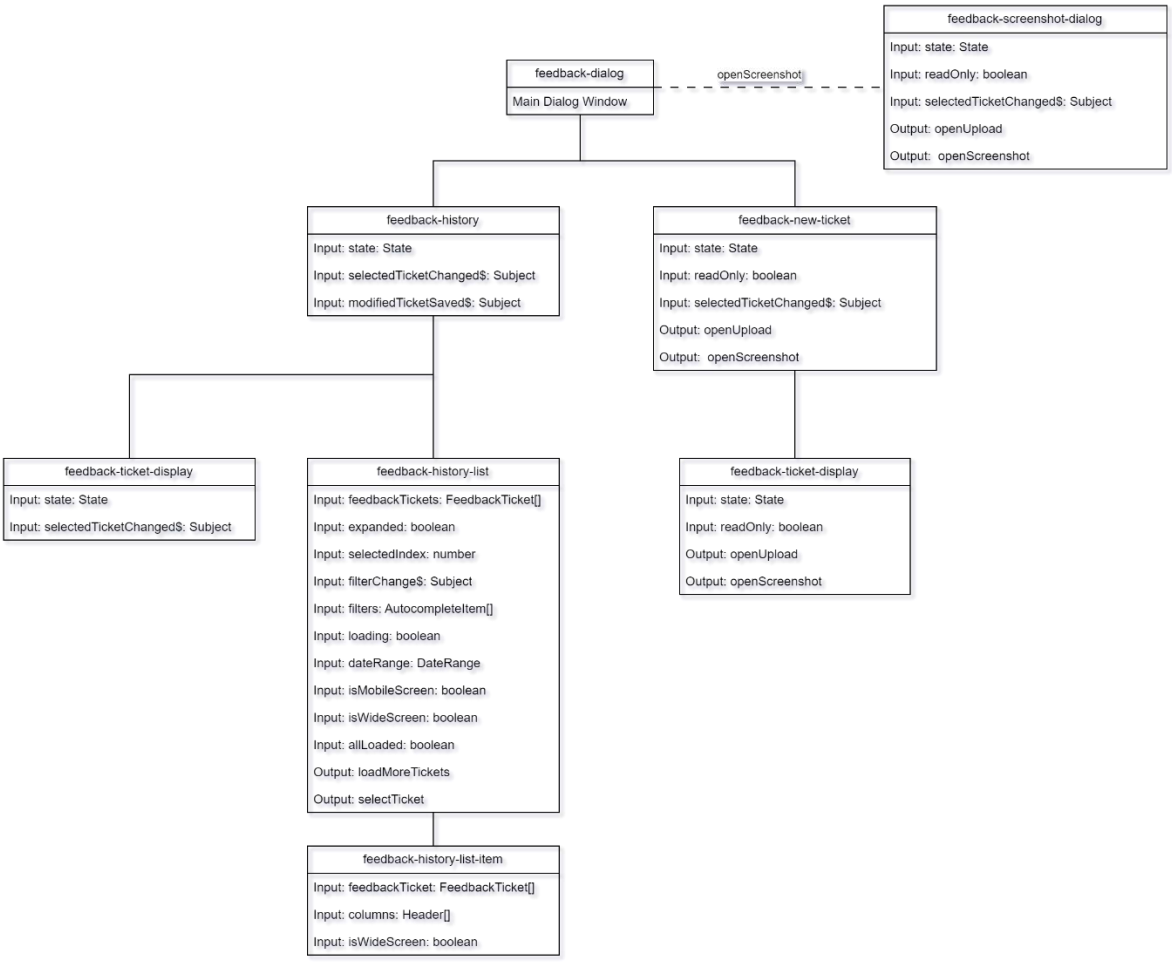
Kuvio 3. Elasticsearch indeksikartoitus

4 Toteutus

4.1 Yleiskatsaus

Toteutettu käyttöliittymä koostuu useasta komponentista. Näitä ovat mm. palautelomake komponentti, jota käytetään myös palautteiden lukemiseen, palautelista ja sen lista elementit, sekä ruudunkaappaus työkalu. Komponenttien implementoinnissa pyrittiin tasapainottamaan komponenttien määrä sekä huollettavuus.

Seuraavassa kuviossa (Kuvio 4) on kuvattuna implementoitujen komponenttien rakenne yksinkertaistettuna. Kuviosta on jätetty pois Angular Material komponentit, sekä KS komponentit. Palaute dialogi voidaan avata WeldEyen sisällä mistä tahansa. State muuttuja pitää sisällään toiminnallisuudelle tärkeimmät ja yleisimmät muuttujat. Komponenttien keskeisessä tiedonvälityksessä on käytetty Input ja Output datasidoksia, sekä subjekteja, kun on ollut tarpeellista välittää dataa usean komponentin kesken tai usean komponentin läpi. Feedback-screenshot-dialog, eli ruudunkaappausdialogi on yhdistetty katkoviivalla kuvaamaan sitä, että se ei ole minkään elementin suoranainen lapsielementti. Ruudunkaappausdialogi avataan klikkaamalla painiketta feedback-ticket-display komponentissa, mutta dialogin avaus suoritetaan feedback-dialog komponentin sisällä. Painikkeen klikkaustapahtuma välitetään tapahtumasidonnan kautta ylöspäin isäntäelementeille.



Kuvio 4. Implementoitujen komponenttien rakenne

Käyttöliittymän toteutus onnistui ilman suurempia ongelmia. Kehityksen aikana tuli vastaan pienempiä haasteita, kuten responsiivinen asettelu eri komponenteissa sekä dialogi ikkunoiden piilottaminen ruudunkaappauksen aikana. Nämä pienemmät haasteet onnistuttiin ratkaisemaan ilman suurempia viiveitä palautekanavan kehitykselle.

4.2 Ruudunkaappaus

Tutkittaessa näytönkaappaus työkalun toteuttamistapaa selvisi kaksi eri tapaa, HTML-DOMin (Document Object Model) uudelleen luominen canvas elementin sisällä tai getDisplayMedian kautta näytönjakamisen hyödyntäminen.

Ensimmäistä vaihtoehtoa voisi kutsua pseudo-näytönkaappaukseksi, sillä siinä ei ruudun sisältöä käsitellä ollenkaan. Käyttötarkoitus mielessä pitäen ensimmäinen vaihtoehto ei ollut hyväksyttävä, sillä HTML DOMin uudelleen luominen HTML canvas elementin sisällä ei ole aivan yksi yhteen mahdollista, johtuen canvas elementin toteutuksesta.

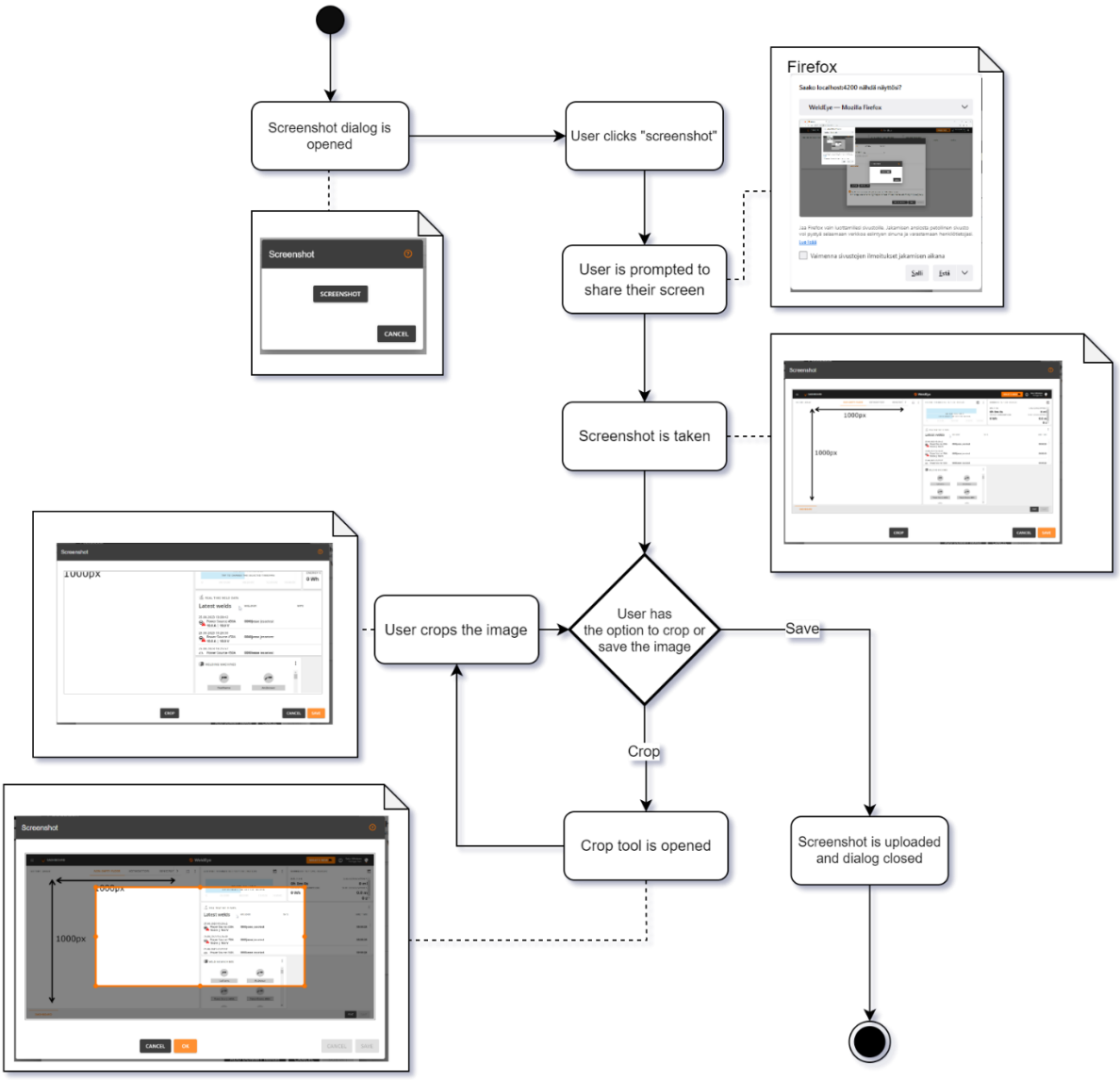
Toinen vaihtoehto on hyödyntää MediaDevices rajapintaa ja pyytää käyttäjältä näytön jakamista. Näytön jakamisen videosignaali piirretään canvas elementtiin, jonka jälkeen näytön jakaminen katkaistaan. Canvas elementin sisältö voidaan konvertoida jpeg tiedostoksi tai blob objektiksi, joka voidaan sitten lähettää palvelimelle.

Chromium pohjaisilla selaimilla, näytön jakamiseen on implementoitu optio, joka mahdollistaa verkkosivun pyytämään omaa välilehteään lähteeksi näytön jakamiselle, mutta koska tämä toiminto ei ole standardisoitu, sama ei toimi muissa selaimissa, joten käyttäjän tarvitsee itse valita oikea ikkuna tai näyttö jaettavaksi. Tästä syystä on oleellista, että näytönkaappaus työkalu tukee ruudunkaappauksen rajaamista jälkikäteen.

4.2.1 Työnkulku

Ruudunkaappaustyökalun toteutuksessa oli tavoitteena pitää työnkulku yksinkertaisena ja helposti ymmärrettävänä käyttäjälle, sillä ruudunkaappaustyökalun tarkoitus on toimia helpompana vaihtoehtona normaaliin ruudunkaappaukseen. Ruudunkaappaustyökalussa on oikealla yläreunassa pieni kysymysmerkki, jota klikattaessa antaa tilanteeseen oleellisia ohjeita.

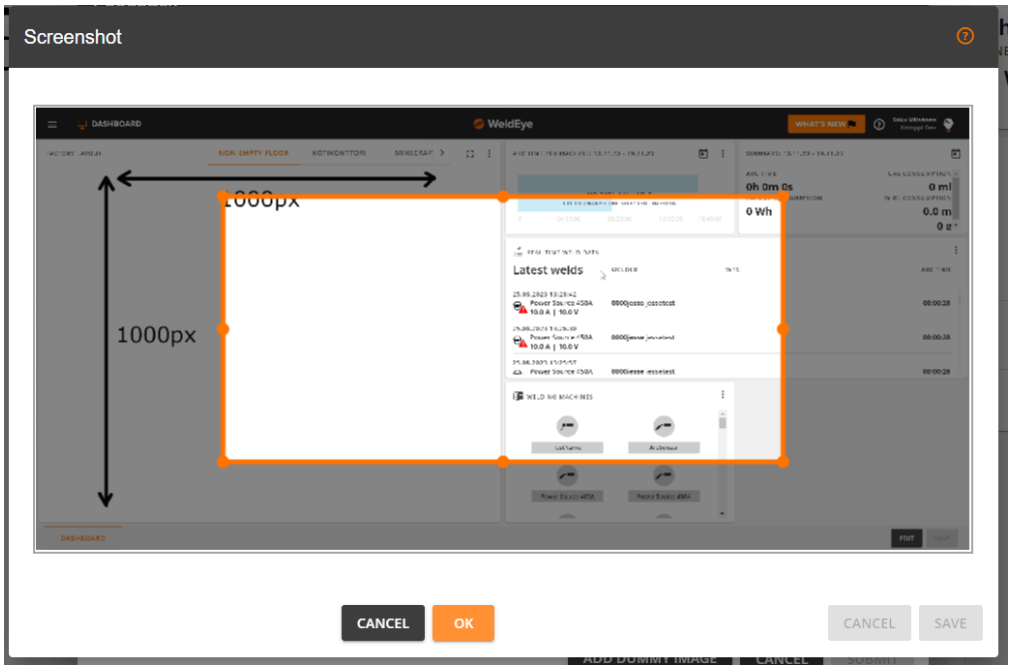
Kuviossa (Kuvio 5) on nähtävissä ruudunkaappaustyökalun työnkulku. Käyttäjän avatessa ruudunkaappaustyökalu nähtävissä on vain yksi painike, ”screenshot”. Tämä toteutettiin näin, jotta voidaan tarvittaessa lisätä ohjeistusta näytönjakamisen sallimiseen selaimessa. Klikatessaan painiketta selain pyytää käyttäjää valitsemaan jaettavan kohteen ja, joko sallimaan tai estämään näytönjakamisen verkkosivulle. Käyttäjän salliessa näytönjakaminen, voi ruudunkaappaus prosessi edetä. Näytönjako alkaa ja dialogi ikkunat piilotetaan määrittämällä niille tyyliluokka. Näytönjako kestää muutaman sekuntin. Tämä on tarpeen, sillä selaimet käyttäytyvät eritavoin näytönjakamisen yhteydessä. Jotkin selaimet lisäävät palkin näytönjakamisesta selaimen ylälaitaan aiheuttaen katseluikkunan koon muutoksen, kun taas toiset selaimet kaappaavat ensimmäisen kehyksen ennen kuin dialogi ikkunat on piilotettu. Näytönjakamisen jälkeen käyttäjä voi esikatsella ruudunkaappausta ja voi halutessaan suorittaa rajaustoiminnon.



Kuvio 5. Ruudunkaappauksen työnkulku

4.2.2 Rajaaminen

Kuvassa (Kuva 4) nähdään ruudunkaappauksen rajaaminen. Käyttäjä voi rajata haluamansa alueen siirtämällä suorakulmiota. Siirtäminen tapahtuu klikkaamalla ja raahaamalla suorakulmiota kahvoista. Ulosjäävä alue on himmennetty. Käyttäjä voi hyväksyä rajauksen klikkaamalla OK.



Kuva 4. Ruudunkaappauksen rajaaminen

Rajauksen valintaan käytetään omaa canvas elementtiä, joka on asetettu olemaan päällekkäin ruudunkaappaus canvaksen kanssa. Rajauksen valinnan voisi toteuttaa samassa canvas elementissä esikatselun kanssa, mutta se vaatisi jokaisella päivityksellä ruudunkaappauksen kopioinnin esikatselu canvakseen. Ruudunkaappauksen rajaus suoritetaan piirtämällä alkuperäisestä canvaksesta valittu alue uudelle canvakselle, jonka jälkeen se piirretään takaisin alkuperäiseen canvakseen.

4.3 Palautelomake

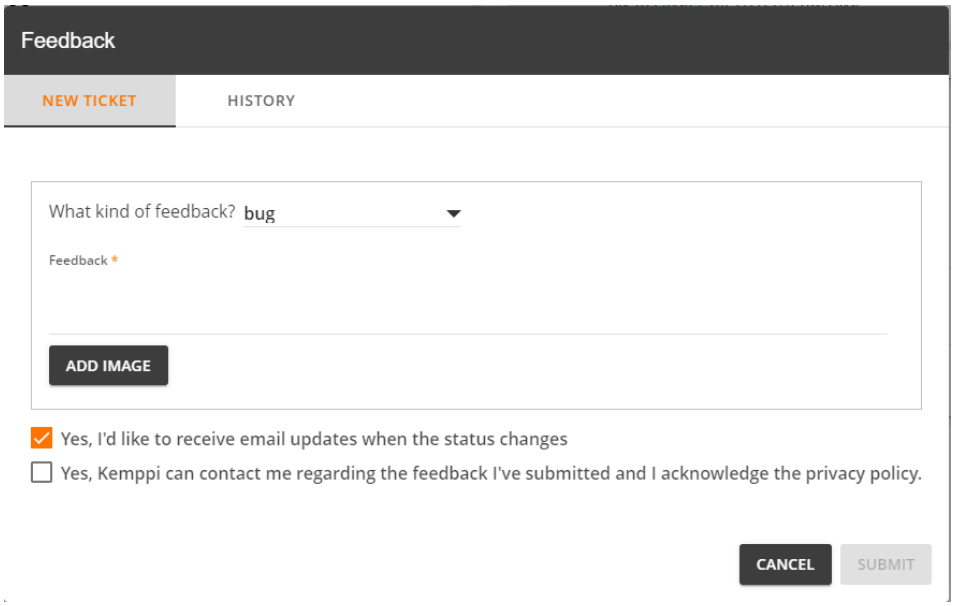
Palautelomakekomponentti on toteutettu siten, että sitä voidaan käyttää myös palautteiden lukemiseen. Kaiken kaikkiaan, lomakkeessa on vain 2 pakollista kenttää, palautteen tyyppi, joka valitaan pudotusvalikolla, sekä tekstikenttä, johon käyttäjä voi kirjoittaa vapaamuotoisen palautteen. Käyttäjä voi myös lisätä kuvia, joko lataamalla ne palveluun tai ottamalla ruudunkaappauksen.

Palautelomakekomponentin toteutuksen aikana ei kohdattu ongelmia, mutta palautehistoriaa implementoitaessa kohdattiin haasteita palautelomakekomponentin kanssa. Ongelmaksi muodostui se, miten palautelomaketta käytetään uudelleen palautteiden lukemiseen ja miten palautteiden data säilötään feedback-dialog komponentissa. Ongelma ilmeni siirtyessä historia välilehdeltä takaisin palautelomake välilehdelle, viimeksi luetun tiketin data pysyi lomakkeessa hetken aikaa tai jos siirtymisen ajoitti oikein, data jäi pysyvästi.

Ratkaisu ongelmalle löytyi käyttämällä tiketin vaihtamiseen subjektia. Näytettävän tiketin data lähetetään subjektin kautta. Subjekti luodaan feedback-dialog komponentissa, josta se välitetään datan sidonnan kautta lapsielementeille. Subjektin tilaus sijaitsee myös feedback-dialog komponentissa, ennen tilausta subjektin tarkkailtava putkitetaan switchMap operaattorin läpi. SwitchMap operaattori mahdollistaa tilauksen callback funktion keskeyttämisen, jos SwitchMap operaattori saa uuden arvon kesken funktion suorittamisen. Käytännössä tämä tarkoittaa sitä, että jos tiketin vaihtamisen aikana vaihdetaan tikettiä uudestaan tai siirrytään takaisin lomake välilehdelle, voidaan vanhentunut vaihto-operaatio keskeyttää ja täten estää väärän tiketin datan näyttämisen. Koodissa tähän subjektiin referoidaan nimellä selectedTicketChanged$.

4.3.1 Käyttöliittymä

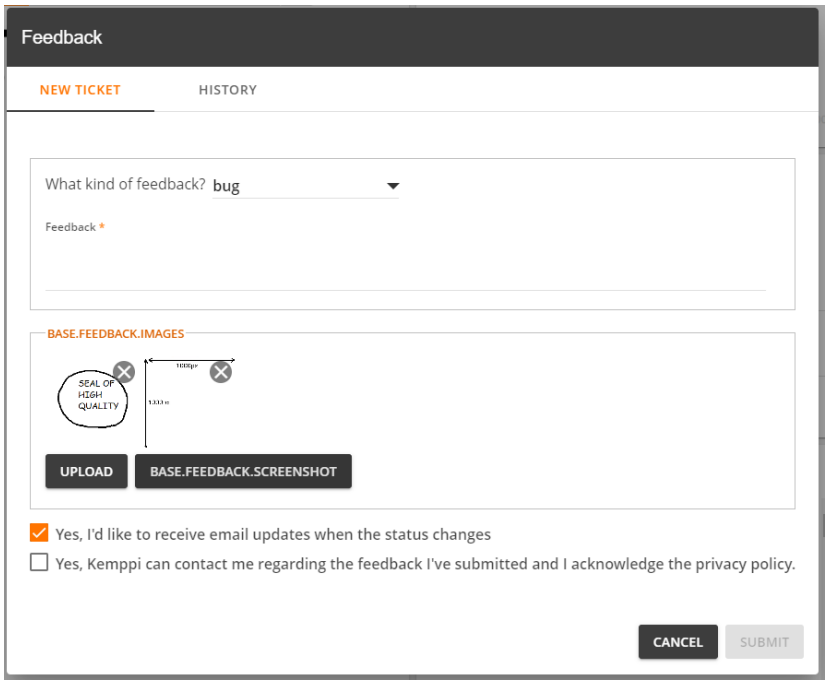
Kuvasta (Kuva 5) nähdään palautelomake tyhjillään. Tyhjilläänkin tekstikentälle on määritetty vähimmäiskorkeudeksi 3 riviä. Vähimmäiskorkeus on määritetty vihjeenä käyttäjälle, että kenttään voi kirjoittaa enemmän tekstiä. Palautelomakkeen tekstikenttänä toimii ks-input-textarea, joka on Angular Material matInput komponenttiin pohjautuva tyylitelty tekstikenttäkomponentti.



Kuva 5. Toteutettu lomake

Kuvien lähettämiseen on käytetty ks-upload dialogi komponenttia, joka avataan käyttämällä mat legacy dialog palvelua. Ks-upload tukee toistaiseksi vain yhden kuvan lähettämistä kerrallaan, mutta suunnitelmissa on päivittää ks-upload tukemaan usean kuvan lähettämistä. Ruudunkaappaus tapahtuu käyttämällä projektia varten kehitettyä ruudunkaappaus komponenttia.

Kuvasta (Kuva 6) nähdään tilanne, jossa käyttäjä on lisännyt kuvia. Kuvia ja ruudunkaappauksia ei erotella. Kuva-alue on skrollattava, joten käyttäjä voi lisätä enemmän kuvia kuin dialogilla on leveyttä.



Kuva 6. Lisättyjä kuvia

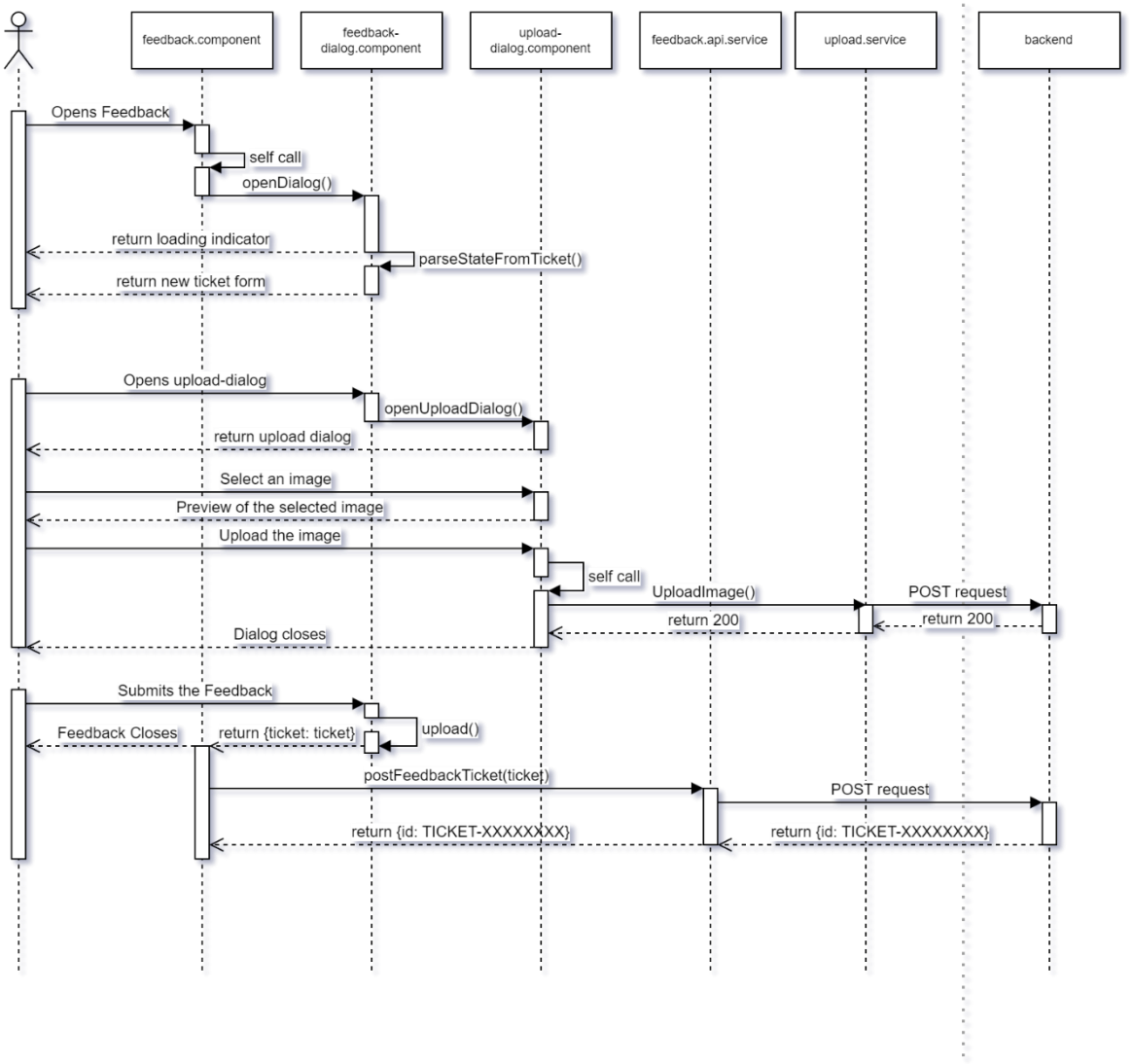
Pudotusvalikko on toteutettu käyttämällä ks-select komponenttia, ja sen valinnat on rajattu 3 erityyppiseen palautteeseen bug, improvement ja other. Palautteen lähettämisprosessi ei eroa palautetyyppien välillä.

4.3.2 Toiminta

Lomakkeen kentät tallentuvat state muuttujaan. State muuttuja luodaan käyttämällä parseStateFromTicket metodia. Kyseinen metodi on toteutettu siten, että jos sitä kutsutaan ”undefined” arvolla, se palauttaa tyhjän State muuttujan. Palautelomake puoli on käärittyfeedback-new-ticket komponenttiin, jonka tehtävä on initialisoituessa välittää tiketin vaihtosubjektille (selectedTicketChanged$) ”undefined” arvo.

Kuvio 6 sisältää sekvenssidiagrammin palautteen lähettämisestä. Kun käyttäjä avaa palautedialogin, feedback-dialog komponentti aukeaa ja ensimmäiseksi käyttäjälle esitetään latausindikaattori, kunnes uusi state muuttuja on luotu. Kuvien lisäämiseen käytetty upload-dialog avataan feedback-dialog komponentin sisältä, eli feedback-dialog sisältää referenssin upload-dialogiin. Käyttäjän lisätessä ja lähettäessään kuvan, upload-dialog sulkeutuu ja kuvan tiedot tallennetaan state muuttujaan. Tässä vaiheessa itse kuva on jo tallennettu taustajärjestelmään, ja tikettiin liitetään mukaan vain kuvan tiedostonimi. Käyt-

täjän lähettäessä palautelomake, feedback-dialog komponentti kutsuu metodia, jonka tehtävä on muodostaa state muuttujasta FeedbackTicket tyyppinen objekti. Tämän jälkeen, feedback-dialog sulkeutuu palauttaen valmiin palautetiketin feedback.component luokalle, joka vuorostaan lähettää tiketin taustajärjestelmään käyttämällä feedback.api palvelua. Taustajärjestelmä palauttaa uuden tiketin ID:n. Toistaiseksi tätä palautettua arvoa ei kumminkaan hyödynnetä käyttöliittymän puolella.



Kuvio 6. Palautteen lähettäminen

4.4 Palautehistoria

Palautehistorian käyttöliittymä ei aiheuttanut suuria ongelmia, vaikkakin pieniin haasteisiin törmättiin paikoittain. Yksi esimerkki näistä pienistä haasteista oli saada palautelista ja sen eri sarakkeet asettumaan oikein eri resoluutioilla.

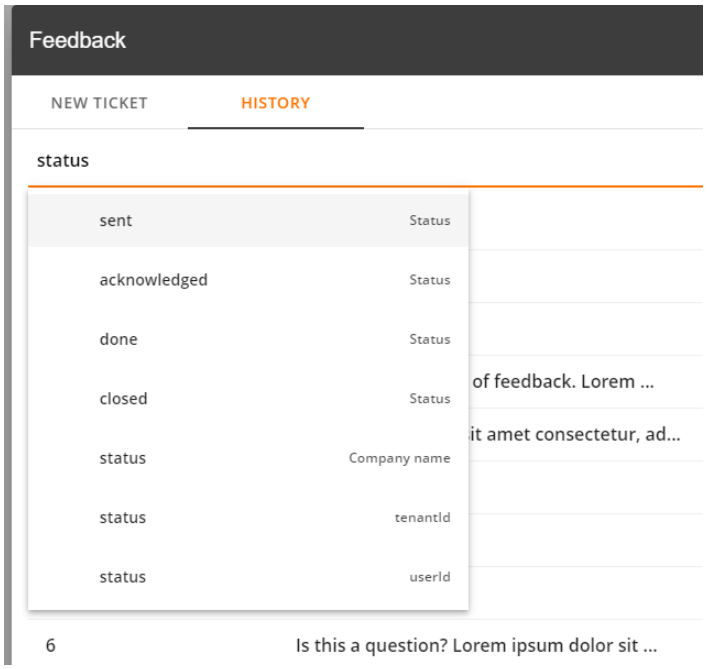
Sarakkeiden asettelu ongelman syy on monijakoinen. Sarakkeille ei aseteta mitään kiinteää leveyttä, vaan sarakkeet asettautuvat CSS flexboxilla, teoriassa tämä mahdollistaa listan paremman horisontaalisen skaalautuvuuden. Käytännössä flexboxin hyödyntäminen toimii niin kauan kuin listan elementit eivät sisällä ylileveitä datakenttiä. Datakenttien sisältö ei tosin olisi ongelma, ellei samaan aikaan haluttaisi myös hyödyntää CSS text-overflow: ellipsis tyylittelyä. Text-overflow: ellipsis piilottaa ylivuotavan tekstin ja päättää näkyvän osuuden pistekolmikkoon (…), mutta koska se vaatii white-space: nowrap tyylittelyn toimiakseen tarkoittaa se sitä, että datakenttä arvot muuttuvat yksi rivisiksi ja usein pitkät arvot vuotavat yli.

Ainut ratkaisu tähän ongelmaan on rajoittaa datakenttä arvojen pituutta, jotta ne eivät vuoda leveyden osalta yli. Ratkaisu on luoda yksinkertainen putki, joka ottaa luvun ja lyhentää sisällön luvun pituiseksi päättäen sisällön pistekolmikkoon.

4.4.1 Suodatus

Palautteiden suodatuksen käyttöliittymä toteutettiin ensiksi painikkeina, mutta mielipide kierroksen jälkeen päädyttiin vaihtamaan painikkeet tag-input elementiksi. Tag-input on ulkonäöltään hakukenttä, johon tekstiä syöttäessä saadaan ehdotuksia avainsanahauille.

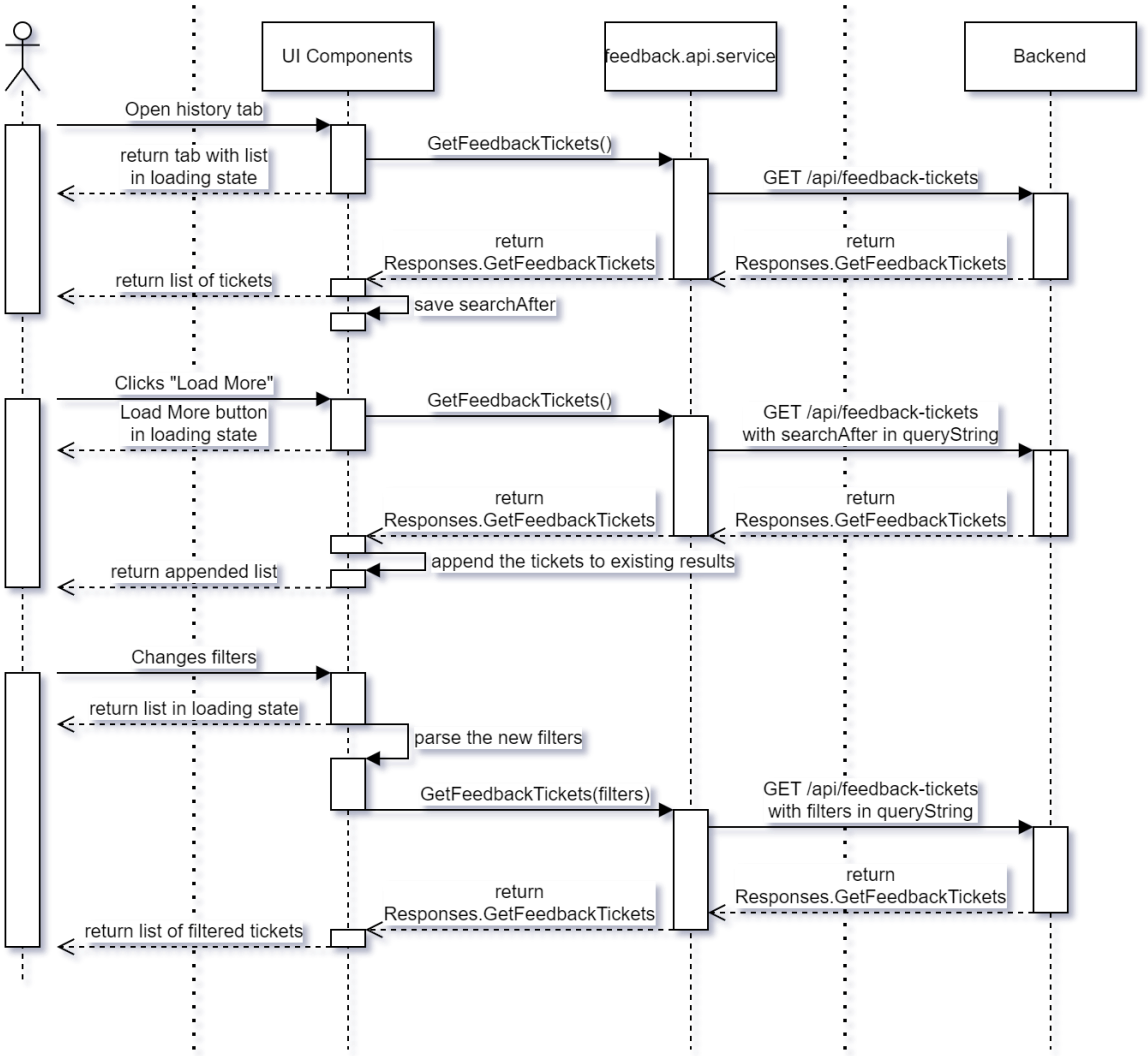
Kuvassa (Kuva 7) nähdään tag-input elementti ehdotuksineen. Ehdotukset, jotka sisältävät vain kiinteitä arvoja, pääasiassa status ja type kentät, määritetään ennakkoon feedback-history-list komponentin initialisoituessa. Näiden ennalta määritettyjen ehdotuksien kohdalla, tag-input ehdottaa myös kentän nimen perusteella, kuten kuvassa nähdään.



Kuva 7. Tag-input elementti ja ehdotukset

4.4.2 Toiminta

Kuviossa (Kuvio 7) kuvatut kutsut ovat asynkronisia, ja ne on toteutettu hyödyntämällä RxJS:n subjekteja ja tarkkailtavia. Käyttäjän avattaessa historia välilehden haetaan uusimmat palautetiketit, jotka on luotu käyttäjän organisaation sisällä. Jos käyttäjä on järjestelmän ylläpitäjä, haetaan uusimmat palautetiketit välittämättä siitä missä organisaatiossa ne on luotu. Käyttäjän muuttaessa suodatin asetuksia, haetaan suodatuksen mukaiset palvelutiketit. Tikettien haussa käytetään feedback.api palvelua.



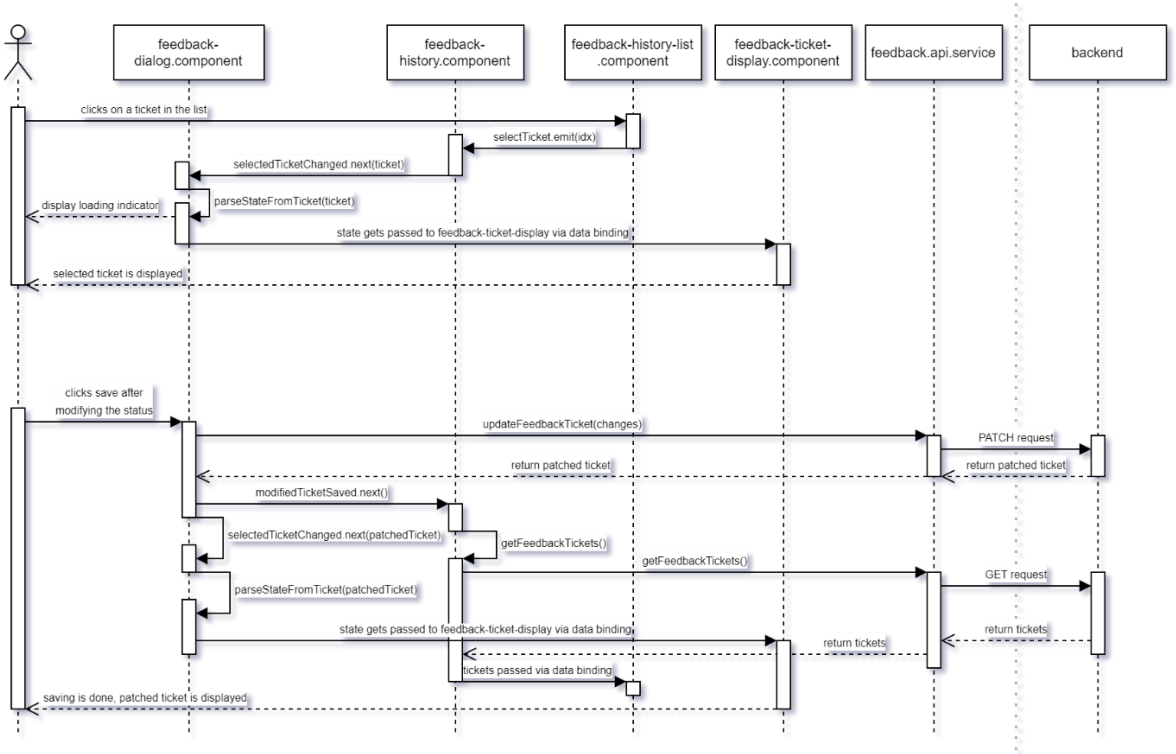
Kuvio 7. Selaus ja suodatus sekvenssikaavio

Http kutsut taustajärjestelmään käyttävät Angularin HttpClienttiä, joka palauttaa tarkkailtavan. Käyttäjän aktivoidessa tikettien uudelleenhaku, esimerkiksi muuttamalla suodatinasetuksia, käyttäjälle viestitään latauksesta esittämällä latausindikaattori. Latausindikaattori sijaitsee joko tikettilistan ylälaidassa tai ”load more” painikkeessa, riippuen siitä onko listassa elementtejä.

4.5 Palautteen lukeminen ja päivittäminen

Palautteen lukeminen ja päivittäminen toteutettiin käyttämällä palautelomakekomponenttia read-only tilassa. Palautelomakekomponentin ollessa read-only tilassa, se paljastaa kenttiä, jotka ovat piilotettu *ngIf direktiiviä hyödyntämällä. Näitä ovat mm. palautetiketin referenssi numero, palautetiketin status ja jos lukeva käyttäjä on ylläpitäjä, näytetään tiketin lähetyksen yhteydessä kerätyt tiedot.

Kuviossa (Kuvio 8) nähdään tiketin päivittämisen prosessi sekvenssidiagrammin muodossa. Kun käyttäjä klikkaa tikettiä, feedback-history-list komponentti lähettää data sidonnan kautta viestin, joka sisältää klikatun tiketin alkionumeron taulukossa. Feedback-history komponentin vastaanotettua viestin Feedback-history komponentti välittää selectedTicketChanged$ subjektille valitun tiketin sisällön ja avaa tikettinäkymän. Käyttäjälle esitetään latausindikaattori, kunnes state on valmiina näytettäväksi. State välittyy feedback-ticket-display komponentille data sidonnan kautta useamman elementin lävitse. Käyttäjän ollessa järjestelmän ylläpitäjä, hän voi muokata ja tallentaa muokatun tiketin. Tiketin tallennusprosessi alkaa feedback-dialog komponentissa, jossa verrataan muokattua tikettiä alkuperäiseen. Jos tikettiä on muokattu, lähetetään muutokset taustajärjestelmään hyödyntäen feedback.api palvelua. Taustajärjestelmä palauttaa muokatun tiketin, joka välitetään selectedTicketChanged$ subjektille, joka hoitaa esitetyn tiketin vaihtamisen eteenpäin. Samanaikaisesti kutsutaan modifiedTicketSaved$ subjektia, jota käytetään viestin välittämiseen feedback-history komponenttiin. Saadessaan viestin modifiedTicketSaved subjektilta feedback-history komponentti hakee tiketit uudestaan.



Kuvio 8. Tiketin päivittämisen diagrammi

4.6 Muutoksia

Projektin lähestyessä loppua pidettiin palaveri, jossa käytiin palautekanavaa läpi. Vastaanotto palautekanavalle oli pääasiassa positiivinen, mutta todettiin, että jotkin asiat kai-

paisivat vielä hiomista. Palaverissä päätettiin muuttaa joitakin asioita ja parantaa palautekanavan toiminnallisuutta lisäämällä uusia ominaisuuksia.

Palautelomakkeeseen päätettiin lisätä mahdollisuus nimetä ja järjestellä uudelleen lisättyjä kuvia, sen lisäksi kuvat numeroidaan. Motivaatio muutokselle on kuvien referoimisen helpottaminen käyttäjän kirjottaessa lomaketta.

Palautetiketin päivitykseen päätettiin lisätä järjestelmän ylläpitäjille mahdollisuus sisällyttää viesti, joka toimitetaan päivitysilmoitussähköpostissa. Päivitysviestin lisäksi haluttiin tiketteihin lisätä päivityshistoria, joka tallentaa mitä on päivitetty, milloin päivitys on tehty sekä päivitykseen liitetyn viestin. Päivityshistoria on nähtävissä tiketin lukemisen yhteydessä kaikille käyttäjille. Palautteen tyyppi haluttiin tehdä muokattavaksi järjestelmän ylläpitäjille

Palautehistoriaan haluttiin enemmän vaihtoehtoja suodattamiselle. Haluttiin mahdollisuus suodattaa tikettejä mm. käyttäjän sähköpostin tai organisaation avulla. Kaikki halutut lisäsuodattimet ovat hyödyllisiä vain järjestelmän ylläpitäjille.

4.6.1 Suodatin muutokset

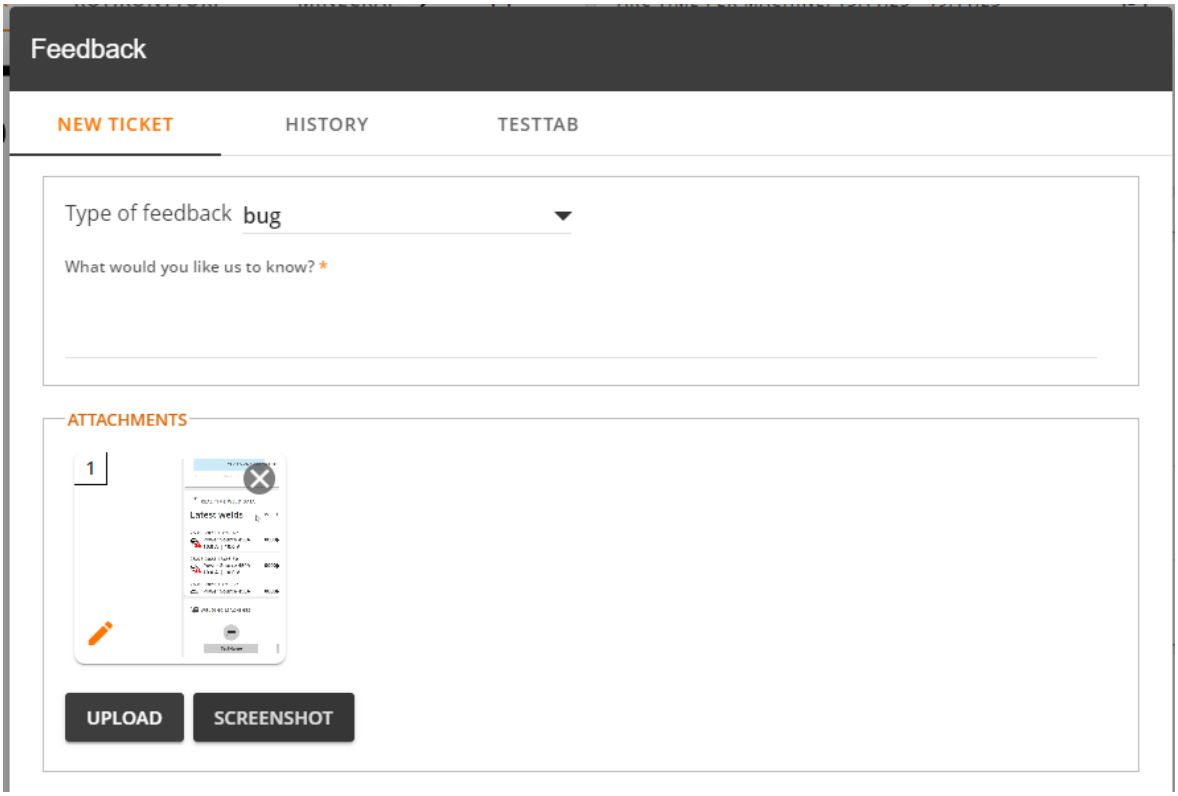
Palautehistorian suodattimien lisääminen vaati muutoksia niin käyttöliittymä puolella kuin taustajärjestelmän puolella. Osa halutuista suodatin vaihtoehdoista vaati Elasticsearchin indeksin muokkaamista, sillä suodatukseen haluttuja kenttiä ei ollut vielä indeksoitu. Indeksointia on mahdollista muokata jälkikäteen, mutta jos indeksistä ei oteta varmuuskopiota, voi ihmiserheen tapahtuessa menettää osan indeksidatasta.

Käyttöliittymän puolella suodattimien lisääminen ei vaatinut suurempia toimenpiteitä. Uudet suodattimet saatiin lisättyä muokkaamalla vain tag-input-menu elementille generoitavaa ehdotuslistaa. Uudet suodattimet lukittiin järjestelmän ylläpitäjä roolin taakse, sillä normaali käyttäjä ei niitä tarvitse.

4.6.2 Kuviin liittyvät muutokset

Kuvien uudelleen järjestämisen implementointiin käytettiin Angular Materialin CDK:sta löytyvää drag and drop (suom. raahaa ja pudota) toimintoa. Drag and drop toiminnon avulla kuvien uudelleen järjestelyn toteuttaminen oli yksinkertaista. Kooditasolla toiminnon implementointi vaati template tiedostossa direktiivien asettamisen oleellisille elementeille, joita ovat raahattavat elementit ja elementtien säiliö. Komponentin puolella uudelleen järjestely vaati funktion, joka siirtää elementin taulukossa paikasta toiseen. Kuvien nimeäminen vaati enemmän muutoksia, sillä palautetiketin tietomalli ei kuvan nimiä vielä tukenut.

Kuvassa (Kuva 8) on kuvien lisäämisen käyttöliittymä muutoksien jälkeen. Kuvasta Kuva 8 on nähtävissä kuvan numeroinnin visuaalinen toteutus. Kuvat numeroidaan laittamalla vasempaan ylälaitaan kuvan indeksinumero taulukossa + 1. Klikkaamalla kuvan vasemmassa alalaidassa sijaitsevaa oranssia kynää, voi käyttäjä lisätä kuvalle otsikon. Otsikkokenttä on toteutettu normaalina input elementtinä.

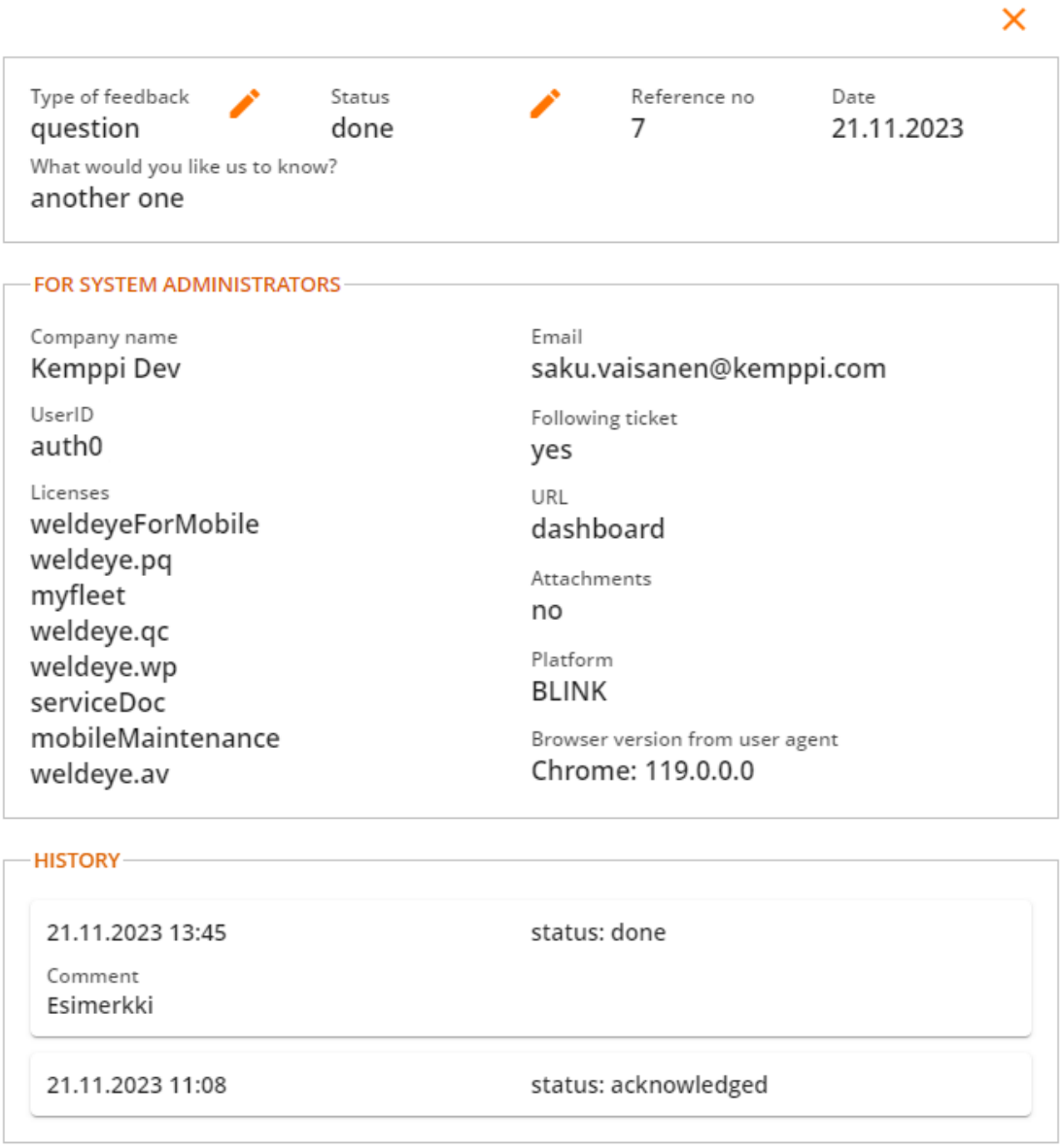


Kuva 8. Päivitetty kuvien lisääminen

4.6.3 Palautteen päivitys

Päivityshistorian implementointi vaati muutoksia palautetiketin tietomalliin. Päivityshistoria lisättiin tietomalliin valinnaisena kenttä, joka on tietotyypiltään taulukko päivitysmuutoksia. Päivitysmuutoksen tietomalliin kuuluu 4 kenttää, aikaleima, muuttuneen arvon avain, uusi arvo ja valinnainen viesti.

Kuvassa (Kuva 9) nähdään miltä tiketin päivityshistoria näyttää käyttöliittymässä. päivityshistoria lisättiin sarjana mat-card elementtejä käänteisessä aikajärjestyksessä. Jokainen elementti sisältää päiväyksen muuttuneen arvon avaimen sekä uuden arvon sekä viestin, jos muutos sisältää viestin.



Kuva 9. Tiketin päivityshistoria

Palvelutikettien päivitystä varten luotiin uusi dialogi, joka sisältää muokattavan arvon sekä viestikentän. Sulkiessa päivitysdialogin save-painikkeen kautta, tallennetaan tiketin muutokset välittömästi.

5 Yhteenveto

Työn tavoitteena oli toteuttaa palautekanava Kemppi Oy:n WeldEye palveluun. Palautekanavan lisääminen WeldEyehin helpottaa palautteen antamista ja mahdollistaa suoraviivaistaa kommunikaatiota käyttäjien ja kehittäjien välillä. Palautekanavaan haluttu toiminnallisuus sisälsi alun perin palautelomakkeen, historianäkymän lähetettyjen palautteiden selaamiseen, sekä mahdollisuuden lukea ja päivittää palautteiden statusta. Palautteisiin piti myös pystyä lisäämään kuvia, sekä mahdollisuus ottaa ruudunkaappaus suoraan selaimesta. Työ kattoi niin käyttöliittymän kuin myös taustajärjestelmän.

Työn toteutus kesti arvioitua kauemmin, mutta säilyi Kempin asettaman aikarajan sisällä. Toteutuksen venymiseen on monta syytä, pienet odottamattomat bugit siellä täällä, arvioidun keston aliarviointi, ja kehityksen aikana tapahtuva toivottujen ominaisuuksien tarkennus, muokkaus sekä lisääminen. Kestoa lukuun ottamatta, toteutusprosessissa onnistuttiin hyvin.

Palautekanavaa voidaan arvioida kahdesta näkökulmasta, ylläpitäjän / kehittäjän tai tavallisen käyttäjän. Ensimmäisestä näkökulmasta katsottuna palautekanava täyttää kaikki vaaditut, sekä toivotut ominaisuudet, joten siltä kantilta työn tuotoksessa on onnistuttu. Käyttäjän näkökulmasta katsottuna palvelukanavaa ei vielä voida arvioida sillä tuotos ei ole vielä tuotannossa asti, joten käyttäjäkokemuksia palautekanavalla ei vielä ole. Suunnitteilla on, että palautekanavaan annettaisiin pääsy tietyille asiakkaille koemielessä.

Jatkokehitysideoita on jo useita. Yksi niistä on toive integroida palautekanava tehtävienhallintaohjelmisto Jiraan. Tätä mahdollisuutta on jo alustavasti tutkittu. Idea integraatiossa olisi se, että slack kanavalle lähetetty viesti sisältäisi painikkeen, jota klikkaamalla voitaisiin palautetiketistä luoda Jira -tiketti. Tiketin luomisen lisäksi, voitaisiin Jira automaatiota hyödyntämällä linkittää myös Jira -tiketin sulkeminen palautetiketin sulkemiseen, eli kun Jira -tiketti suljettaisiin, sulkeutuisi myös palautetiketti. Tämä vähentäisi työtä ja työtaakkaa, jota palautekanavan ylläpitäminen lisää.

Toinen jatkokehitysidea olisi mahdollistaa tavallisten käyttäjien päivittää tikettejä, esimerkiksi lisäämällä niihin kuvia. Jos tätä ideaa haluaa viedä pidemmälle, voitaisiin mahdollistaa kaksisuuntainen viestintä suoraan palautekanavan kautta. Tämä keskittäisi asiakas-ylläpitäjä viestinnän yhteen paikkaan. Muita jatkokehitysideoita on esimerkiksi näytöntallennus lyhyeksi videoksi tai .gif -tiedostoksi tai tikettistatistiikan kerääminen ja hyödyntäminen sisäisesti.

Lähteet

BairesDev. Haettu 2023. A Comprehensive Guide to Angular Services. Saatavissa https://www.bairesdev.com/blog/angular-services/

Bidelman. E. & Dutton. S. Haettu 2023. Capture audio and video in HTML5. Saatavissa https://web.dev/articles/getusermedia-intro

Chapman. C. Haettu 2023. Why use Material Design? Saatavissa https://www.toptal.com/designers/ui/why-use-material-design

Elastic.co. Haettu 2023. What is unstructured Data? Saatavissa https://www.elastic.co/what-is/unstructured-data

Elbourn. J. 2018. A Component Dev Kit for Angular. Saatavissa https://blog.angular.io/a-component-dev-kit-for-angular-9f06e3b4b3b4

Fulton. S. & Fulton J. 2011 O'Reilly Media, inc. HTML 5 Canvas. Saatavissa https://www.oreilly.com/library/view/html5-canvas/9781449308032/ch01.html

Garg. B. 2019. Angular Architecture Overview. Saatavissa https://medium.com/@bhavikagarg8/angular-architecture-overview-1e7cc7483a0

Gavigan. D. 2018. The History of Angular. Saatavissa https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7

Gudelli. A. 2019. History of AngularJS. Saatavissa https://www.angularjswiki.com/angular/history-of-angularjs/

Kumar. S. 2019. RESTful API. Saatavissa https://medium.com/@ksarthak4ever/restful-api-1a49417729a8

Lease. D. 2018. TypeScript: What is it & when is it useful? Saatavissa https://medium.com/front-end-weekly/typescript-what-is-it-when-is-it-useful-c4c41b5c4ae7

Mongodb.com. Haettu 2023. Unstructured Data. Saatavissa https://www.mongodb.com/unstructured-data

Pal. S. 2019. A Full overview of HTML Canvas. Saatavissa https://www.freecodecamp.org/news/full-overview-of-the-html-canvas-6354216fba8d/

Pethiyagoda. N. 2022. REST API Naming Conventions and Best Practices. Saatavissa https://medium.com/@nadinCodeHat/rest-api-naming-conventions-and-best-practices-1c4e781eb6a5

Rehemägi. T. 2021. How to Get Started With AWS Lambda and Node.js: A guide for the Uninitiated. Saatavissa https://aws.plainenglish.io/tutorial-getting-started-with-aws-lambda-and-node-js-4ad50d9e4cc4

Salazar. M. 2020. To Angular Material or not to Angular Material?. Saatavissa https://marasalazar.medium.com/to-angular-material-or-not-to-angular-material-940a616ac22d

Sharma. V. 2021. What is ViewEncapsulation in Angular? Saatavissa https://vibhas1892.medium.com/what-is-viewencapsulation-in-angular-58d31901901c

Slack. E. 2021a. RxJS for Beginner Angular Developers – Part1: Introduction. Saatavissa https://medium.com/ngconf/rxjs-for-beginner-angular-developers-introduction-2d26ffa364af

Slack. E. 2021b. RxJS for Beginner Angular Developers: Fundamentals Saatavissa https://medium.com/ngconf/rxjs-for-beginner-angular-developers-part-2-fundamentals-535b939378d5

StackOverflow. Developer Survey 2023. Saatavissa https://survey.stackoverflow.co/2023/

Thakur. I. 2021. Understanding Angular. Saatavissa https://medium.com/analytics-vidhya/understanding-angular-2775383eac99

Tietosuoja.fi. Haettu 2023. Milloin henkilötietoja saa käsitellä? Saatavissa https://tietosuoja.fi/kasittelyperusteet

Velotio Technologies. 2019. Elastisearch 101: Fundamentals & Core Components. Saatavissa https://medium.com/velotio-perspectives/elasticsearch-101-fundamentals-core-components-a1fdc6090a5e

Venati. R. 2019. Introduction to Amazon S3. Saatavissa https://medium.com/@rahulvenati/day-5-introduction-to-amazon-s3-5d47d2962a3

WHATWG. Haettu 2023. HTML Living Standard – The Canvas Element. Saatavissa https://html.spec.whatwg.org/multipage/canvas.html#the-canvas-element

Yewale. S. 2020. Data Binding with Angular. Saatavissa https://medium.com/angularjs-front-end-web-framework/data-binding-with-angularjs-83a0845d3209