Ylläpitopalvelun ulosoton toteutus Inertialla

Kasper Ahlroth

Haaga-Helia ammattikorkeakoulu
Tradenomi
Opinnäytetyö
2023

Tiivistelmä

Tekijä(t)
Kasper Ahlroth

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Ylläpitopalvelun ulosoton toteutus Inertialla

Sivu- ja liitesivumäärä
30

Opinnäytetyössä toteutetaan ulosottojen käsittelyominaisuus toimeksiantajan uuteen ylläpitopalveluun. Toteutuksessa käytetään Vue ja Laravel sovelluskehyksiä sekä Inertia kirjastoa toteuttamaan sovelluskehyksien välistä keskustelua.

Inertiaa kuvaillaan modernina monoliittina eli uutena tapana kehittää ohjelmistoja monoliittisella arkkitehtuurilla. Monoliittisessa arkkitehtuurissa kaikki ohjelman toiminnot on rakennettu samaan projektiin. Mikropalveluarkkitehtuuriin verrattuna monoliittinen arkkitehtuuri voi olla helpompi kehittää, testata ja ottaa käyttöön ja lokitietojen kerääminen sekä suorituskyvyn seuraaminen on yksinkertaisempaa.

Inertia pystyy lataamaan täysiä JavaScript pohjaisia sivuja ja käyttämään reaktiivisuutta sekä osittaisia sivunlatauksia ilman mikropalveluita tai ohjelmointirajapintoja. Inertialla käytetään XHR protokollaa ja sivuolioita mahdollistamaan JavaScript sivun tietojen päivittämisen ilman täyttä sivulatausta. Inertia tarjoaa komponentteja ja apufunktioita mahdollistamaan reaktiivista datan päivittämistä komponenteissa.

Toteutusta varten toimeksiantaja laati kymmenen tehtäväkokonaisuutta, joiden avulla kaikki ulosottojen käsittelyyn vaaditut ominaisuudet toteutuisivat. Tehtäväkokonaisuudet rakentavat toteutuksessa tarvittavia toimintoja järjestyksessä, jossa seuraava kokonaisuus rakentaa edellisen kokonaisuuden päälle uusia toimintoja. Kaikki kokonaisuudet toteutuivat onnistuneesti ja ulosottojen käsittely ylläpitopalvelussa toimii odotetusti.

Lopuksi monoliittisen arkkitehtuurin ja Inertian hyötyjä tarkastetaan pohtimalla niiden vaikutusta projektin toteutuksen kannalta.

Asiasanat
Inertia, Monoliitti, Arkkitehtuuri, Ylläpito

Sisällys

1 Johdanto

Toimeksiantajayritys tarjoaa palvelutuotteen, joka tekee laskujen luomisesta ja lähettämisestä helppoa. Palvelutuotteen toimintoihin kuulu laskujen lähettämisen lisäksi lakisääteisten vero- ja eläkemaksujen ennakonpidätys ja maksaminen. Laskut menevät lähetyksen jälkeen toimeksiantajayrityksen taloushallinnon käsittelyyn, jota varten on käytössä ylläpitopalvelu. Ylläpitopalvelu mahdollistaa taloushallintoa muokkaamaan, hyväksymään ja hylkäämään laskuja ja muita käyttäjien tai viranomaisten lähettämiä lomakkeita. Toimeksiantaja on julkaissut uuden version palvelutuotteestaan, jossa on uusia ominaisuuksia. Ominaisuuksien toiminnan eheyden ja vastuullisen toiminnan kannalta, taloushallitsijoiden on pakko käsitellä uusien ominaisuuksien lähettämiä lomakkeita ylläpitopalvelussa. Ongelmana on, että käytössä oleva ylläpitopalvelu on sidoksissa vanhaan palvelutuotteeseen. Uusien ominaisuuksien lisääminen on tämän takia hankalaa eikä ole kannattavaa. Tämän johdosta tehtiin päätös, että uudelle palvelutuotteelle toteutettaisiin uusi ylläpitopalvelu.

Projektin aihe on toteuttaa rajattu aihealue toimeksiantajayrityksen ylläpitopalvelun korvausprojektissa. Uusi ylläpitopalvelu tarvitaan mahdollistamaan uusia ominaisuuksia toimeksiantajayrityksen palvelutuotteessa, helpottamaan palvelutuotteen kehitystä ja takaamaan ylläpitopalvelun jatkokehittämismahdollisuudet. Toimeksiantajan palvelutuote käsittelee käyttäjien raha-, vero- ja eläkeasioita. Ylläpitopalvelu on oleellinen palvelutuotteen toiminnan eheyden kannalta, sillä se mahdollistaa käyttäjien tietojen hallinnan ja käyttäjien täyttämien lomakkeiden hyväksymisen ja hylkäämisen. Tällöin toimeksiantajayritys pystyy vastuullisesti käsittelemään ja auttamaan käyttäjiä tärkeissä lakisääteisissä raha-, vero- ja eläkeasioissa.

Projektissa käytettävien teknologioiden valinnassa kiinnitettiin paljon huomiota ohjelmistokehityksen henkilöstöresursseihin. Toimeksiantajayrityksen palvelutuotteen kehityksessä oleva lisäominaisuus vei suurimman osan kehitysresursseista, joten ylläpitopalvelun korvausprojekti piti pystyä toteuttamaan rajatuilla resursseilla. Tämän perusteella teknologioiden valintakriteeriksi muodostui kehittämisen helppous, kuinka paljon backend- tai frontend-kehittäjä pystyy toteuttamaan sekä backend- että frontend-tehtäviä. Tämän avulla vähennetään riskiä, että ylläpitopalvelun toteutus viivästyy resurssipulan takia.

Ylläpitopalvelun korvausprojektin loppukäyttäjiä ovat toimeksiantajayrityksen sisäiset työntekijät. Suurimmat käyttäjäryhmät tulevat olemaan taloushallinto ja ohjelmistotestaajat. Palvelu tulee ainoastaan sisäiseen käyttöön ja projektin tyylivaatimusten ei tarvitse olla yhtä korkeat kuin toimeksiantajan tarjoamissa tuotteissa. Ohjelmistokehityksen UX standardien tai saavutettavuuskäytäntöjen seuraaminen ei ole projektin kannalta yhtä oleellista kuin ulkoisille käyttäjille tarjotuissa tuotteissa. Sisäisiä designresursseja käytettiin projektiin rajatusti, eivätkä

kaikki aihealueet saneet omia uusia ulkonäkösuunnitelmia. Projektissa käsiteltävä ylläpitopalvelun ulosottojen käsittely käytti vanhaa ulkonäköä mallina.

Vaatimusten ja rajausten laatimisen jälkeen luotiin ryhmä, jonka tehtävänä oli toteuttaa pieni proof of concept-projekti erilaisilla mahdollisilla teknologioilla. Proof of concept–projektin aikana työryhmä arvioi teknologioita vaatimuksiin ja rajauksiin peilaten. Lopputuloksena Inertia osoittautui parhaaksi vaihtoehdoksi. Toimeksiantajayrityksen palvelutuotteessa on käytössä Laravel- ja Vue-sovelluskehykset, jotka ovat molemmat Inertian tukemia teknologioita, joten ylläpitoprojektin toteutuksessa vähintään yksi käytössä olevista teknologioista on ennestään tuttu frontend ja backend kehittäjille. Inertia, Laravel ja Vue ovat kaikki aktiivisessa kehityksessä, joten ylläpitopalvelun päivittäminen ja jatkokehitys ovat mahdollisia. Inertia on teknologia, joka mahdollistaa täysien JavaScript sivujen lataamisen backend puolelta. Inertian Vue ja Laravel yhteentoimivuuden johdosta on mahdollista käyttää frontend-komponentteja ja backend-luokkia sekä funktioita palvelutuotteesta, joka vähentää kehitysmäärää.

Inertian arkkitehtuuri eroaa huomattavasti toimeksiantajayrityksen palvelutuotteesta. Palvelutuote on mikropalvelupohjainen verkkoapplikaatio, jossa on erilliset backend- ja frontend-ohjelmistosovellukset, joiden väliseen keskusteluun käytetään ohjelmointirajapinnan kutsuja ja mikropalveluja. Inertia sen sijaan kuvaa itseään modernina monoliittina, joka mahdollistaa monoliittisen ohjelmistoarkkitehtuurin käytön.

Ylläpitopalvelun korvausprojekti on laaja kokonaisuus ja on projektinhallinnallisista syistä päätetty jakaa pienempiin aihealueisiin. Opinnäytetyölle on korvausprojektista valittu rajattu aihealue, joka käsittelee käyttäjien ulosoton hallintaa ylläpitopalvelussa.

Ylläpitopalvelun ulosottojen käsittelyn ominaisuuskokonaisuuden tavoitteena on rakentaa Inertiaa, Laravelia ja Vuea käyttäen toimiva osa toimeksiantajan ylläpitopalvelua, joka mahdollistaa vastuullisen palvelutuotteen käyttäjien ulosottojen käsittelyyn. Toteutus katsotaan onnistuneeksi, jos toimeksiantajan taloushallinnoitsija pystyy tarkastelemaan kaikkia käsittelyä vaativia ulosottoja, pystyy muokkaamaan ulosottojen tietoja ja pystyy hyväksymään tai hylkäämään ulosoton.

Opinnäytetyön toteuttajan näkökulmasta tavoitteena halutaan rakentaa ymmärrystä ohjelmistokehityksen arkkitehtuurista, Inertian roolista monoliittisessa arkkitehtuurissa ja sen tuomista hyödyistä ja heikkouksista. Tavoite katsotaan onnistuneeksi riippuen toteutuksen onnistumisesta ja toteuttajan omista näkemyksistä Inertiasta ja monoliittisesta arkkitehtuurista.

2 Ohjelmistoarkkitehtuuri ja Inertia

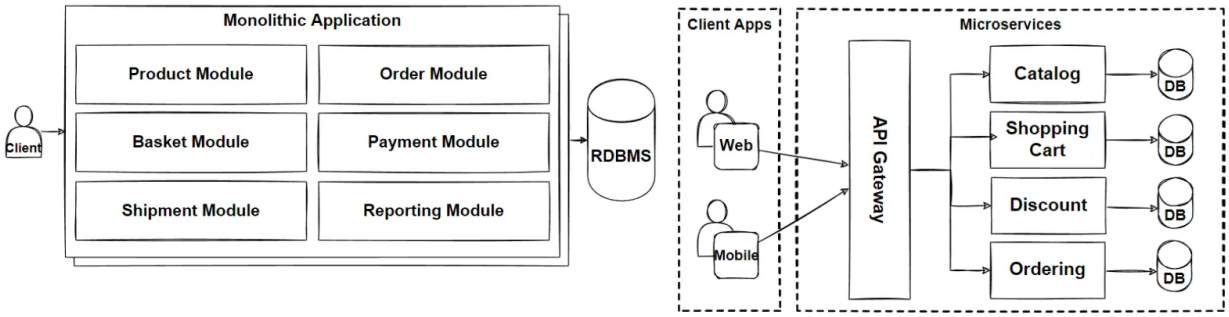
Kappaleessa halutaan selvittää mikä on monoliittinen arkkitehtuuri, ja mitkä ovat sen vahvuuksia ja heikkouksia. Inertiasta halutaan selvittää mikä se on ja mitä ominaisuuksia Inertia tuo projektin toteutukseen.

2.1 Monoliittinen arkkitehtuuri

Monoliittinen arkkitehtuuri ohjelmistokehityksessä tarkoittaa, että ohjelmiston kaikki osat on koottu yhteen projektiin (Doglio 2.2.2022). Bisneslogiikan ja käyttöliittymän koodit ovat samassa projektissa ja yleensä niin sidottuja toisiinsa, että niitä ei pysty ottamaan käyttöön osissa tai erikseen (Doglio 2.2.2022). Käyttöjärjestelmän näkökulmasta monoliittinen ohjelmisto toimii yhtenä prosessina palvelimessa (Blinowski, Ojdowska & Przybyłek 2022, 20358). Monoliittisen arkkitehtuurin suurin vahvuus on sen yksinkertaisuus verrattuna muihin esimerkiksi mikropalveluarkkitehtuureihin (Blinowski, Ojdowska & Przybyłek 2022, 20358).

Monoliittinen arkkitehtuuri on aiemmin ollut perinteinen tapa rakentaa ohjelmistoja suurissa yrityksissä (De Lauretis 2020, 94). Yritysohjelmistot on usein rakennettu kolmen tason mallilla. Käyttäjäpuolen taso kostuu HTML-sivuista ja JavaScript-metodeista. Palvelinpuolen taso käsittelee bisneslogiikkaa, HTTP-pyyntöjä, hakee ja päivittää dataa tietokannasta sekä palauttaa HTML-sivuja ja niiden sisältöä selaimelle (Blinowski, Ojdowska & Przybyłek 2022, 20358). Monoliittisessa arkkitehtuurissa kaikki toiminnot on rakennettu yhteen ohjelmaan ja se tuo monia hyötyjä yksinkertaisempiin ohjelmiin ja palveluihin (De Lauretis 2020, 94).

Monoliittiseen arkkitehtuuriin pohjautuvista ohjelmistoista ollaan nykyään siirtymässä mikropalveluarkkitehtuuriin. Mikropalveluarkkitehtuurissa kaikki toiminnot ja ominaisuudet on hajautettu useaan pienempään ohjelmaan, jotka kootaan yhteen ja näytetään yhtenä ohjelmana käyttäjälle. Monoliittista ja mikropalveluarkkitehtuuria verrattaan usein toisiinsa. Usein vertailu tehdään, kun aikomuksena on siirtyä monoliittisesta arkkitehtuurista mikropalveluarkkitehtuuriin. Kuva 1 näyttää kaavion monoliittisesta ja mikropalvelun arkkitehtuurista.



Kuva 1. Application Architecture (Ozkaya 2023)

Monoliittisen arkkitehtuurin hyötyihin ja vahvuuksiin mikropalveluarkkitehtuuriin verrattuna kuuluu:

- Yksinkertaisempi testaus ja bugien korjaus, sillä testejä ja korjauksia tarvitsee toteuttaa ainoastaan yhteen ohjelmaan (Doglio 2.2.2022). Monoliittisen arkkitehtuurin yksinkertaisemman rakenteen takia ohjelmiston toiminnan ymmärtäminen on helpompaa ja testauksen suunnittelu yksinkertaisempaa (Doglio 2.2.2022).
- Helpompi ohjelmiston lokitietojen kerääminen, suorituskyvyn seuraaminen ja välimuistin käsittely, koska kaikki ohjelmiston toiminnot tapahtuvat samassa projektissa (Doglio 2.2.2022).
- Helpompi ohjelmiston käyttöönotto, sillä koko ohjelmisto on yhdessä projektissa (Doglio 2.2.2022).
- Helpompi kehittäminen, koska datan käsittelyyn vaaditaan ainoastaan luokka- tai metodikutsuja (Doglio 2.2.2022).
- Parempi suorituskyky pienemmissä ohjelmissa (Blinowski, Ojdowska & Przybyłek 2022, 20358). Komponenttien välinen kommunikointi on nopeaa ja kevyttä suorituskyvyn näkökulmasta, sillä kommunikointi tapahtuu enimmäkseen funktiokutsujen kautta (Martin 2017, luku 18).

Mielipiteet monoliittisen arkkitehtuurin tuomista hyödyistä testauksen näkökulmasta vaihtelevat suuresti lähteestä riippuen. Esimerkiksi kirjassa Software Architecture: The Hard Part väitetään, että kerrosrakenteinen monoliittinen arkkitehtuuri kärsii huonosta testattavuudesta. Täyden kattavuuden saavuttaminen regressiotestauksella on haastavaa ohjelmistokokonaisuuden laajuden takia. Pitää ottaa huomioon, että täydellä testikattavuudella koodimuutosten toteuttaminen kestäisi huomattavasti kauemmin. Muutosten yhteydessä kaikki ohjelmiston testit ajettaisiin uudelleen riippuen testien määrästä ja se vaatisi paljon aikaa. Virhetilanteiden ilmetessä ongelman selvittäminen olisi haastavaa. (Ford, Richards, Sadalage & Dehghani 2021, luku 3.)

Ohjelmiston suhteellista ylläpidettävyyttä voidaan mitata tutkimalla:

1. Ohjelmistokomponenttien välistä kytkentää eli kuinka tietoisia komponentit ovat toisistaan (Ford, Richards, Sadalage & Dehghani 2021, luku 3).
2. Ohjelmistokomponenttien välistä kiinteyttä eli millä tavalla ja tasolla komponenttien toiminnat liittyvät toisiinsa (Ford, Richards, Sadalage & Dehghani 2021, luku 3).
3. Syklomaattista monimutkaisuutta (Ford, Richards, Sadalage & Dehghani 2021, luku 3), joka on tapa laskea ja määrittää komponentin monimutkaisuutta (IBM, 2021).
4. Ohjelmistokomponentin kokoa (Ford, Richards, Sadalage & Dehghani 2021, luku 3).

Suurilla monoliittisilla arkkitehtuureilla on yleisesti matala ylläpidettävyys kerrosrakenteen aiheuttaman toimintojen hajautuksen takia, korkea ohjelmistokomponenttien välinen kytkentä ja heikko komponenttien välinen kiinteys. Huonon ylläpidettävyyden takia uuden ominaisuuden lisääminen monoliittiseen ohjelmaan on suurempi kokonaisuus, koska muutokset vaikuttavat ohjelmistotasolla ja saattavat levittäytyä jokaiselle sovelluskerrokselle (Ford, Richards, Sadalage & Dehghani 2021, luku 3). Kehitystiimien kokoonpanosta riippuen toteutus saattaa vaatia yhteistyötä useiden kehitystiimien välillä, joka tekee toteutuksesta ja ylläpidosta haastavampaa (Ford, Richards, Sadalage & Dehghani 2021, luku 3). Monoliittisen ohjelman skaalaminen on vaikeampaa, mutta sitä voi helpottaa ajamalla useita instansseja ohjelmasta load balancering kanssa (Blinowski, Ojdowska & Przybyłek 2022, 20360).

Ohjelmiston kehittäminen monoliittisellä arkkitehtuurilla on kannattavaa tilanteissa, joissa vaatimuksia on rajatusti ja soveltamispiiri ei ole suuri. Monoliittisen arkkitehtuurin käyttöönotto kannattaa, jos ohjelman tai palvelun loppukäyttäjien määrä on alhainen (Doglio 2.2.2022). Hyvä esimerkki on toimeksiantajayrityksen sisäiset testaajat ja taloushallinnoitsijat. Toimeksiantajayrityksen ylläpitopalvelun pienen loppukäyttäjämäärän takia monoliittinen ohjelmistoarkkitehtuuri on sopiva toteutustapa korvausprojektille.

2.2 Inertian perusta

Inertia.js dokumentaatio kuvailee Inertiaa uutena tapana rakentaa perinteistä palvelinpohjaista verkkoapplikaatiota. Inertia toimii usealla backend sovelluskehyksellä, mutta se on tehty toimimaan parhaiten Laravelin kanssa. Inertia ei ole sovelluskehys, eikä se ole korvaus palvelinpuolen tai käyttäjäpuolen sovelluskehyksille. (Inertia. Introduction.)

Inertia ei käytä käyttäjäpuolen reititystä tai ohjelmointirajapintaa. Inertian kanssa sovelluksia rakennetaan palvelinpuolella normaaliin tapaan luomalla luokkia, funktioita ja reitityksiä. (Inertia. How it works.)

Inertia on pohjimmiltaan käyttäjäpuolen reitityskirjasto, joka mahdollistaa näkymien renderöinnin ja sivulatauksia ilman sivun täyttä uudelleen lataamista (Inertia. How it works). Palvelinpuolen näkymien ja sivujen renderöinti korvataan Inertian omalla palautus funktiolla, joka lataa täysiä JavaScript pohjaisia single-page application (SPA) sivuja. Yleisesti modernin JavaScript pohjaisen SPA sovelluksen käyttöön tarvitaan ohjelmointirajapinta tai mikropalvelu, jolloin keskeiset ominaisuudet kuten reaktiivisuus ja osittaiset sivunlataukset toimisivat. Inertia pystyy hahmottamaan reaktiivisia SPA sivuja ilman mikropalveluita ja sovellusrajapintoja. Normaalisti, jos haluaa näyttää ohjelman sivuja ilman mikropalveluita tai ohjelmointirajapintoja, palvelinpuolelta palautetaan näkymiä, eli palvelinpuolella rakennettuja sivuja (Inertia. How it works). Inertian tarjoama hyöty tulee siitä, että ohjelman sivujen näyttämisessä pystyy käyttämään esimerkiksi Vue-komponentteja ja sen kautta Vuen tarjoamia ominaisuuksia.

Normaalisti linkkiä painamalla selain tekee täyden sivunlatauksen, jolloin käyttäjäpuolen ohjelmistokehys käynnistää sivun ohjelman uudelleen. Inertia tekee sivulatauksia XHR protokollan kanssa. XHR tai XMLHttpRequest protokolla mahdollistaa datan hakemisen URL linkistä ilman täyttä sivulatausta, jolloin tapahtuu osittainen sivulataus, joka päivittää sivun tietoja (MDN Web Docs). XHR protokollan avulla palvelin pystyy tunnistamaan sivulatauksen Inertia sivulatauksena ja palvelin palauttaa sivun JSON tiedostona tyypillisen täyden HTML sivun sijaan. Inertia pystyy JSON tiedoston avulla päivittämään sivun komponentit ja selaimen välimuistin historian (Inertia. How it works).

Ensimmäinen kutsu Inertia applikaatiolle on täyden sivun selainpyyntö, joka palauttaa HTML dokumentin. HTML vastaukseen kuuluu sivun CSS ja JavaScript sisältö sekä JSON muotoinen sivuolio. JSON muotoisen sivuolion avulla Inertia käynnistää käyttäjäpuolen palvelukehyksen ja näyttää halutun sivun. (Inertia. The protocol.)

Inertia applikaation käynnistyksen jälkeen kaikki kutsut tehdään XHR protokollalla. Kutsun mukana lähetetään X-Inertia header-avain, jonka arvo on true. Tämän avulla palvelin tietää, että kyseessä on Inertia pyyntö ja palauttaa vastauksena ainoastaan JSON muotoisen sivuolion sen sijaan, että palautuksena olisi HTML dokumentti, joka sisältää sivuolion. (Inertia. The protocol.)

Sivuolio koostuu kaikesta tarvittavasta tiedosta sivun lataamista varten. Siihen kuuluu JavaScript sivun komponentti, sivun propsit, sivun URL ja resurssiversiot. Täydessä sivunlatauksessa sivuolio on osana HTML vastausta JSON muodossa ja Inertia sivunlatauksessa sivuolio palautetaan vastauksessa JSON muotoisena sisältönä. (Inertia. The protocol.)

Inertia käyttää resurssien versiointia seuraamaan sivulle tehtyjä muutoksia. Kun Inertia havaitsee muutoksen sivun resursseihin, Inertia tekee täyden sivulatauksen XHR sivunlatauksen sijasta.

Sivuolion sisältämä resurssiversio asetetaan palvelimen puolella ja sen asetettu arvo voi olla numero, string tai joku muu arvo, joka vastaa resurssin versiota. Inertia pyynnön yhteydessä nykyinen resurssiversio asetetaan pyynnön X-Inertia-Version header-tagin arvoksi. Palvelin vertaa pyynnön yhteydessä saatua resurssiversion arvoa nykyiseen resurssiversion arvoon ja päivittää sivua sen mukaan. Vastaavat arvot jatkavat pyyntöä normaalisti tai palauttavat 409 Conflict-virheen, jos arvoit eivät täsmää. Virhetilanteissa pyyntö palauttaa vastauksena X-Inertia-Location header-tagin, jonka arvo on URL, joka kertoo Inertialle pyynnön lopullisen osoitteen. (Inertia. The protocol.)

Inertia pyynnöt mahdollistavat osittaiset lataukset. Osittaisissa latauksissa Inertia lisää X-Inertia-Partial-Data ja X-Inertia-Partial-Component header-tagit Inertia pyyntöön. X-Inertia-Partial-Data sisältää propsit, jotka halutaan palauttaa ja X-Inertia-Partial-Component sisältää komponentit, jotka halutaan ladata. Osittaisilla latauksilla pystytään parantamaan sivun suorituskykyä. (Inertia. The protocol.)

2.3 Inertian ominaisuudet

Inertia tukee useita backend- ja frontend-sovelluskehyksiä. Koska ylläpitopalvelun korvausprojektissa käytetään Laravel- ja Vue 3-sovelluskehyksiä, Inertian ominaisuuksia tutkitaan niiden toiminnan näkökulmasta.

2.3.1 Inertian sivut ja Inertian palautusten lähettäminen

Inertian suurimpiin ominaisuuksiin kuluu kyky lähettää dataa JavaScript komponenttiin ilman ohjelmointirajapintaa. Tyypillisesti kehotetaan luomaan jokaiselle JavaScript sivulle oma kontrolleri luokka. Koska kontrolleri luokat luodaan jokaiselle sivulle erikseen, pystytään hakemaan, käsittelemään ja palauttamaan JavaScript sivuille ainoastaan data, joka on tarpeellinen sille sivulle.

Kontrolleri luokassa pitää luoda funktio, jossa on palautus. Inertian ominaisuuksia varten on käytettävä joko Inertia::render-metodia tai inertia()-apufunktiota, näiden avulla palautuksesta tehdään Inertia-palautus (Inertia. Responses). Molemmat keinot toimivat samalla tavalla. Inertia-palautukseen määritellään JavaScript sivun nimi, joka kertoo mille komponentille data ja näkymä ohjautuu. Jotta Inertia-palautuksen mukana saadaan lähetettyä dataa komponentille, palautukseen pitää määrittää attribuutteja. Attribuuteille asetetaan data, joka halutaan lähettää palautuksen mukana.

Vue-komponentin puolella dataa vastaanotetaan määrittämällä Inertia-palautuksen mukana lähetetyt attribuutit propseina (Inertia. Pages). Propsien määrittämisen jälkeen vastaanotettua

dataa pystytään käyttämään samalla tavalla, kuin jos propsit olisi vastaanotettu toisesta Vue-komponentista Inertia-palautuksen sijaan.

2.3.2 Inertian Link-komponentti

Linkkien luomiselle Inertia applikaatioissa on olemassa Inertian oma Link-komponentti. Komponentti on rakennettu siten, että se vastaanottaa painallustapahtumia ja estää sivunlatauksia (Inertia. Links). Link-komponentille voi syöttää useita attribuutteja, joiden avulla pystyy muokkaamaan komponentin toimintaa ja ulkonäköä. Komponentin href-attribuutti määrittelee mille sivulle käyttäjä ohjataan Link-komponenttia painamalla. Inertia palauttaa linkkejä oletuksena a-elementteinä ja niitä voi muokata as-attribuutin avulla. As-attribuutille pystyy asettamaan arvon button, joka lataa Link-komponentin button-elementin muodossa. Method-attribuutilla määritetään HTTP-pyynnön metodi. Oletuksena käytössä on GET-metodi, mutta method-attribuutilla sen voi vaihtaa POST-, PUT-, PATCH- tai DELETE-metodiksi. Link-komponenttiin pystyy määrittelemään PUT- ja POST-pyyntöihin ylimääräisen data-attribuutin, jos haluaa sisällyttää pyyntöön ylimääräistä dataa (Inertia. Links).

2.3.3 Reititys

Inertiaa käyttäen kaikki ohjelmiston reititykset määritetään palvelimen puolella ja reititykseen liittyvän pyynnön yhteydessä palautetaan Inertia-vastaus. Inertialla ei tarvitse tehdä reitityksiä frontend sovelluskehyksessä. Laravelin kanssa voi olla käytössä Ziggy kirjasto, joka lisää globaalin route() funktion helpottamaan reitityksien käyttöä (Inertia. Routing). Kirjaston route() funktio lisätään käyttäjäpuolen linkkeihin ja funktioon määritetään reitin nimi. Painalluksen yhteydessä käyttäjäpuolella, funktio aktivoi palvelinpuolella luodun reitin.

2.3.4 Inertian lomakkeet

Inertian kanssa ei suositella HTML lomakkeiden käyttämistä, koska ne aiheuttavat täysiä sivulatauksia. Inertialla ei tarvitse tarkastaa lomakkeita käyttäjäpuolella, koska palvelinpuolella reititys tai kontrolleri luokka lähettää uudelleenohjausvastauksen. Inertialla on olemassa useForm-apumetodi, jonka tarkoitus on vähentää lomakekäsittelyyn tarvittavaa koodia. Apumetodin kanssa lomakkeen lähettämiseen voi käyttää get- post-, put-, patch- ja delete-metodeja. Lomakkeen dataa voi muovata ennen lähettämistä transform()-metodin avulla. Lomakkeiden lähettäminen Inertialla toimii hyvin useimmissa tilanteissa, mutta jos tarvitset enemmän hallintaa lomakkeen lähettämisessä, kannattaa käyttää XHR- tai fetch-pyyntöjä. (Inertia. Forms.)

3 Ylläpitopalvelun ulosottojen toteutus

Ylläpitopalvelun korvausprojekti rakennetaan toimeksiantajayrityksen palvelutuotteen backend projektin päälle. Ylläpitopalvelun korvausprojektin suunnitteluvaiheessa käyttäjien ulosottojen käsittely eroteltiin omaksi aihealueeksi, jonka toteutus tapahtuu omassa kehityshaarassaan. Kehityshaaran perustana toimii toimeksiantajayrityksen palvelutuotteen kehityshaara. Tällä tavalla ylläpitopalvelun kehityksessä on käytössä palvelutuotteen luokkia, funktioita ja komponentteja. Lisäksi Vue-komponenttien toteutuksessa käytetään Vuetify-komponenttikirjastoa, joka tarjoaa kattavan kirjaston komponentteja yleisiin käyttötarpeisiin, esimerkiksi syöttökenttien luomiseen.

3.1 Ulosottojen käsittelyn tehtäväkokonaisuudet

Ylläpitopalvelun ulosottojen käsittelyn toteutusta varten toimeksiantajayritys oli määritellyt tehtäväkokonaisuudet. Tehtäväkokonaisuuksien avulla varmistetaan, että kaikki aihealueen ominaisuusvaatimukset toteutuisivat. Vanhan ulosottokäsittelyn ominaisuudet ja ulkonäkö toimi mallina uudelle versiolle. Ennen toteutuksen aloitusta pidettiin kokous, jossa toimeksiantaja esitteli tehtäväkokonaisuudet ja yhdessä arviointiin jokaisen tehtäväkokonaisuuden työmäärä. Tehtäviin ei kuulu testien kirjoittaminen luoduille luokille, funktioille, metodeille tai komponenteille. Tehtäväkokonaisuudet on järjestetty siten, että jokainen käyttää edellisen rakentamia toimintoja perustana ja rakentaa edellisen kokonaisuuden päälle uusia toimintoja.

Tehtäväkokonaisuudet ovat:

1. Ulosottojen listaussivun luonti, suodatustoiminnon, linkin lisääminen navigaatiopalkkiin ja listattavien ulosottojen suodattaminen ulosoton statuksen mukaan.
2. Muokkaussivun luonti, muokkaussivulle johtavan linkin lisääminen jokaiselle listatulle ulosotolle ja käsittelijän liittäminen käsiteltävälle ulosotolle.
3. Käsittelyjonon tietojen lisääminen ulosoton muokkaussivulle.
4. Linkkien lisääminen muokkaussivulle käyttäjän tarkastamiseen ja listaussivulle siirtymiseen.
5. Ulosoton tietojen palautus muokkaussivulle.
6. Huomioviestin näyttäminen muokkaussivulla, jos usealla käyttäjällä on sama henkilötunnus.
7. Ulosoton tietojen hakeminen xml muodossa.
8. Ulosoton tietojen palauttaminen xml muodossa muokkaussivulle.
9. Lomakkeen lisääminen ulosoton muokkaussivulle.
10. Ulosoton arvojen näyttäminen muokkaussivun syöttökenttien vieressä

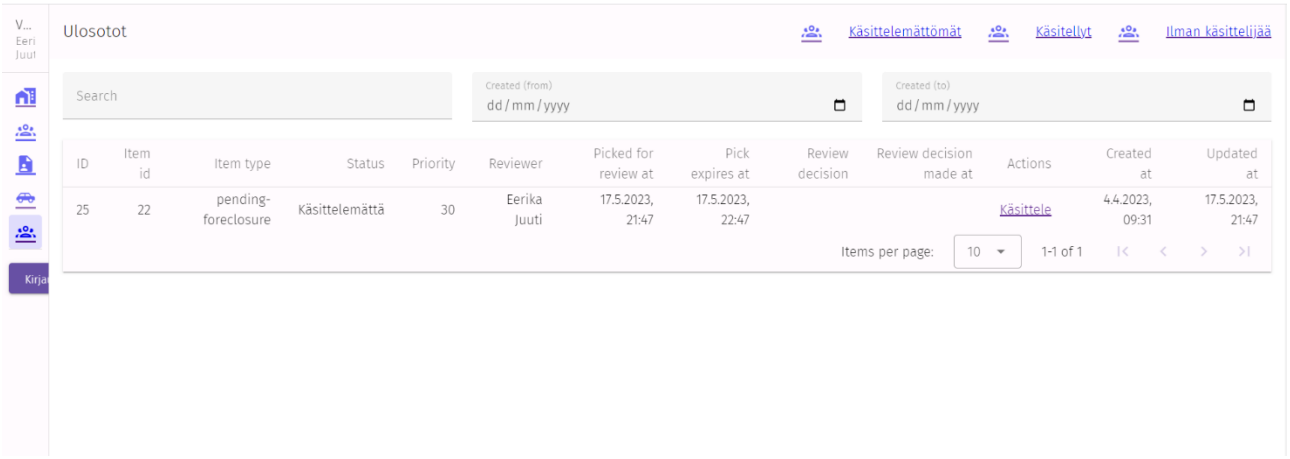
3.2 Tehtäväkokonaisuuksien toteuttaminen

3.2.1 Tehtäväkokonaisuus 1: Ulosottojen listaussivun luonti, suodatustoiminnon, linkin lisääminen navigaatiopalkkiin ja listattavien ulosottojen suodattaminen ulosoton statuksen mukaan.

Ensimmäisen tehtävän toteutuksessa käytin Inertian Link-komponenttia. Link-komponenttia halutaan käyttää, koska se estää täydet sivunlataukset. Muuten Inertia ei pystyisi käyttämään osittaisia sivunlatauksia. Ylläpitopalvelulle on erikseen luotu MainNavigation- omponentti, joka toimii ylläpitopalvelun navigointipalkkina. Navigointipalkkiin halutaan asettaa kaikki linkit, jotka ohjaavat taloushallinnoitsijoita ylläpitopalvelun eri sivuille. Link-komponentille asetetaan href-attribuutti, joka määrää mille sivulle linkki ohjaa. Href-attribuuttiin asetetaan route-apufunktio, joka on Ziggy-kirjaston lisäämä globaali funktio. Riippuen Laravel versiosta, Ziggy-kirjasto saattaa sisältyä Laraveliin (Inertia. Responses). Apufunktio antaa meidän käyttää Laravel-reitityksiä JavaScript-komponentissa ja määritellä reitin nimi ja parametreja. Linkin reitiksi asetetaan admin.review-queue, joka ohjaa linkin oikeaan reititys luokkaan. Lisäksi foreclosure- ja unprocessed-parametriarvot auttavat reititystä hakemaan ulosottoja, joita ei ole vielä käsitelty. Link-komponentin sisälle asetetaan Vuetifyn VListItem-komponentti, koska se muotoilee linkin luettelokappaleeksi, joka sopii tyylimuotoilultaan hyvin navigaatiopalkkiin. Luettelokappaleen tekstiksi asetetaan sopiva nimi foreclosure eli ulosotto.

Kuva 2. Ylläpitopalvelun navigointipalkki

Näiden avulla ylläpitopalvelun navigointipalkkiin ilmestyy kuvassa 2 näkyvä foreclosure niminen painike, jota painamalla tehdään kolme toimintoa. Ensin haetaan ulosottoja, joita ei ole käsitelty ja sen jälkeen ulosotot palautetaan ReviewQueueList-komponenttiin ja viimeiseksi käyttäjä ohjataan samaan komponenttiin. ReviewQueueList on listaussivu (kuva 3), joka asettaa ja näyttää reitityksen kautta tullutta dataa luettelona. Täältä käsin taloushallinnoitsijat tarkastelevat, suodattavat ja valitsevat käsiteltäviä ulosottoja.



Kuva 3. Ylläpitopalvelun ulosottojen listaussivu, jossa näkyy yksi ulosotto

Suodattamisominaisuutta varten luodaan ForeclosureNavigationkomponentti. ForeclosureNavigation toimii samalla tavalla kuin MainNavigation, eli se luo navigaatiopalkin, johon sijoitetaan linkkejä. Toisin kuin MainNavigation-komponentti, ForeclosureNavigation lähettää reititys kutsun samalle sivulle eri parametreillä. ForeclosureNavigation-komponentissa on 3 Link-komponentilla luotua painiketta. Painikkeet hakevat kaikki käsittelemättömät ulosotot, kaikki käsitellyt ulosotot ja kaikki ulosotot, joilla ei ole käsittelijää. Tällä tavalla ReviewQueueList-komponentti saadaan suodattamaan listattuja ulosottoja.

Navigaatiokomponentit ovat hyvä esimerkki, miten Inertian palautukset toimivat käytännössä. Esimerkiksi, jos Chrome selaimessa avaa konsolin ja tarkastaa Link-komponentin lähettämän pyynnön. Pyynnön tarkastuksessa on olemassa Content-Type-attribuutti, jonka arvo on application/json. Content-Type kertoo selaimelle missä muodossa data palautetaan. Inertia-palautuksissa dataa palautetaan JSON muodossa ja konsolin avulla näkee, että Inertian palautus toimii oikein.

3.2.2 Tehtäväkokonaisuus 2: Muokkaussivun luonti, muokkaussivulle johtavan linkin lisääminen jokaiselle listatulle ulosotolle ja käsittelijän liittäminen käsiteltävälle ulosotolle.

Tässä tehtäväkokonaisuudessa halutaan luoda ulosotoille muokkaussivu ja lisätä ulosottojen listaussivulle linkki, joka vie muokkaussivulle. Halutaan myös estää kahta tai useampaa taloushallinnoitsijaa käsittelemästä samaa ulosottoa. Tämän kokonaisuuden tärkeys on, että se antaa taloushallinoijan tarkastella kaikkia ulosottoja ja valita niistä mitä ulosottoa haluaa käsitellä muokkaussivulla. Samalla ulosottojen käsittelyn tietoeheyttä parannetaan, koska useammat käsittelijät eivät pysty muokkaamaan ulosoton dataa samaan aikaa.

Jokaiselle listatulle ulosotolle halutaan painike, jonka avulla taloushallinnoitisja pääsee käsittelemään kyseistä ulosottoa. Painike saadaan luotua lisäämällä uusi linkki ReviewQueueList-komponenttiin. Linkin kodalle lisätään v-if-ehto, joka lataa ja näyttää Link-komponentin ainoastaan silloin kun kyseessä on ulosotto. Jotta painike johtaisi ulosoton muokkaussivulle pitää luoda uusi reititys funktio admin-web-reititystiedostoon (Kuva 4), jossa kaikki ylläpitopalveluun liittyvät reitit määritellään. Reititys asetetaan ohjaamaan ulosottojen kontrolleri luokkaan.

```
Route::prefix('/foreclosure')
    ->name('foreclosure.')
    ->group(function () {
        Route::controller(PendingForeclosureAdminController::class)
            ->group(function () {
                Route::get('/{pending_foreclosure}', 'edit')
                    ->name('edit');
            });
    });
```

Kuva 4. Admin-web-reititystiedoston reitti, joka vie ulosoton muokkaussivulle

Muokkaussivua varten pitää luoda ulosotolle oma kontrolleri luokka PendingForeclosureAdminController, jotta pystytään luomaan ulosotoille omia funktioita ja Inertia-palautuksia, joiden avulla pystytään toteuttamaan haluttuja toimintoja ja ominaisuuksia. Luokkaan luodaan edit-funktio (Kuva 5), joka toimii pohjana ulosoton datan hakemiselle ja käsittelylle sekä datan lähettämiselle Vue-komponentteihin. Funktiossa käytetään myös PendingForeclosure model-luokkaa. Model-luokka toimii työkaluna tietokannan ja sovelluksen välisessä keskustelussa, jonka avulla voidaan määritellä mitä tietoa halutaan hakea tietokannasta.

```
public function edit(Request $request, PendingForeclosure $pendingForeclosure)
```

Kuva 5. PendingForeclosureAdminController-luokkaan luotu julkinen edit-funktio

Koska halutaan, että PendingForeclosureAdminController-luokka ohjaa ulosoton muokkaussivulle, pitää luoda palautus. Palautukseen käytetään Inertian inertia()-apufunktiota, koska se mahdollistaa Inertian ominaisuuksia kuten osittaiset sivunlataukset. Apufunktiolle pitää ainoastaan syöttää sen komponentin nimi, jolle halutaan, että palautus ohjaa. Tässä tilanteessa halutaan ohjata EditPendingForeclosurePage-komponenttiin. Jos haluaa lähettää dataa komponentille, route()-apufunktioon lisätään myös attribuutteja, joiden arvoksi asetetaan data, joka halutaan lähettää

(Kuva 6). Tässä vaiheessa pitää ainoastaan luoda EditPendingForeclosurePage-komponetti, johon lisätään hello world teksti ja nyt kontrolleri luokan palautus vie näkymään, jossa näkyy teksti.

```
return inertia('Foreclosures/EditPendingForeclosurePage', [
    'reviewQueueItem' => new ReviewQueueResource($reviewQueueItem),
    'user' => new UserResource($pendingForeclosure->user),
    'activeForeclosure' => $pendingForeclosure->user->getActiveForeclosure(),
    'duplicateUserSsn' => $duplicateUserSsn,
    'pendingForeclosureXml' => $xml,
    'pendingForeclosure' => new PendingForeclosureResource($pendingForeclosure),
    'calculationMethods' => $calculationMethods,
]);
```

Kuva 6. PendingForeclosureAdminController-luokan Inertia-palautus

Nyt on päästy tilanteeseen, jossa jokaisella listatulla ulosotolla on linkki (Kuva 7), jota painamalla näkymä siirtyy ulosoton muokkaussivulle.

Item type	Status	Priority	Reviewer	Picked for review at	Pick expires at	Review decision	Review decision made at	Actions
pending-foreclosure	Käsittelemättä	30	Eerika Juuti	17.5.2023, 21:47	17.5.2023, 22:47			Käsittele

Kuva 7. Ulosotto, jossa on käsittele niminen linkki, joka vie muokkaussivulle

Koska halutaan estää useampaa taloushallinnoitsijaa käsittelemästä samaa ulosottoa samaan aikaan, pitää käsittelyjonosta hakea tietoja siitä kuka käsittelee ulosottoa. Sitä varten luodaan $reviewQueueItem-muuttuja, joka hakee kaiken ulosoton käsittelyyn liittyvät tiedot käsittelyjonosta, esimerkiksi milloin ulosoton käsittely alkoi. Muuttujassa käytetään ReviewQueueResource-resurssiluokkaa, joka muuttaa haetun datan sopivaksi taulukkorakenteeksi Vue-komponentteja varten.

Muuttujan avulla pystytään luomaan if-ehto (Kuva 8). Ehto tarkistaa onko ulosotolla käsittelijää ja ehdon täyttyessä ulosotolle asetetaan käsittelevä taloushallinnoitsija, joka samalla estää muita taloushallinnoitsijoita käsittelemästä kyseistä ulosottoa tunnin ajaksi. Jos tunti on kulunut, ehto asettaa seuraavan taloushallinnoitsijan, joka yrittää käsitellä ulosottoa, uudeksi ulosoton käsittelijäksi. Käsittelyn lukitseminen parantaa tietoeheyttä ja vastuullista käyttäjätietojen hallintaa.

```
if (! $reviewQueueItem->reviewer_id) {
    $this->reviewQueueService->assign($request->user(), $reviewQueueItem);
} elseif (! $reviewQueueItem->review_decision && $reviewQueueItem->pick_expires_at->lessThan(now())) {
    $this->reviewQueueService->reassign(User::find($reviewQueueItem->reviewer_id), $reviewQueueItem, $request->user());
}
```

Kuva 8. If-ehto, joka asettaa käsittelijän ulosotolle

Tehtäväkokonaisuudessa käy hyvin ilmi monoliittisen arkkitehtuurin hyödyt jopa suuremmissa projekteissa. Koska ylläpitopalvelu rakennetaan palvelutuotteen projektin päälle, ylläpitopalvelu pystyy käyttämään luokkia ja funktioita palvelutuotteesta.

3.2.3 Tehtäväkokonaisuus 3: Käsittelyjonon tietojen lisääminen ulosoton muokkaussivulle

Käsittelyjonon tietoja halutaan näyttää muokkaussivulla, koska silloin tiedetään, kuka käsittelee ulosottoa, milloin käsittelijä aloitti ja kuinka kauan hän on käsitellyt ulosottoa. Tiedot pystytään helposti näyttämään käyttämällä ReveiwQueueInformation-komponenttia, joka sisältää myös painikkeen, jonka avulla käsittelijä irrotetaan ulosoton käsittelystä ja lopettaa ulosoton tunnin kestävän käsittely eston.

Käsittelyjonon tiedot kertovat kuka on asetettu ulosoton käsittelijäksi tällä hetkellä, milloin käsittely alkoi ja kuinka kauan on jäljellä, kunnes ulosoton käsittely ei ole enää lukittuna tiettyyn käsittelijään. Kaiken tiedon näyttämiseen käytetään ReveiwQueueInformation-komponenttia, koska sillä on valmiiksi muotoiltu ulkoasu ja siihen tarvitsee ainoastaan syöttää dataa käsittelyjonosta. Käsittelijän irrottaminen ulosoton käsittelystä kuuluu myös komponentin toimintoihin.

Käsittelyjonon data saadaan aiemmin luodusta $reviewQueueItem-muuttujasta. Muuttujan data saadaan lähetettyä muokkaussivulle lisäämällä se Inertia-palautukseen PendingForeclosureAdminController-luokassa. Muuttuja lisätään Inertia-palautukseen, koska tällä tavalla Inertia lähettää dataa Vue-komponentteihin monoliittisen arkkitehtuurin takia.

EditPendingForeclosurePage-muokkaussivulla Inertian palautuksen kautta lähetetty data vastaanotetaan komponentissa rekisteröimällä data propsina (Kuva 9). ReveiwQueueInformation-

komponentti lisätään muokkaussivulle ja siihen syötetään reviewQueueItem-propsi. Muuta ei tarvitse tehdä, koska ReveiwQueueInformation suorittaa kaikki toiminnot.

```
const props = defineProps({
  reviewQueueItem: {
    type: Object,
    required: true
  },
```

Kuva 9. Esimerkki kuinka Vue-komponentissa vastaanotettu data rekisteröidään propsina

Nyt muokkaussivulla näkyy, kuka käsittelee ulosottoa, milloin käsittelijä aloitti ja kuinka kauan hän on käsitellyt ulosottoa (Kuva 10). Painike, joka irrottaa käsittelevän taloushallinnoitsijan ulosoton käsittelystä toimii ja antaa muiden taloushallinnoitsijoiden käsitellä kyseistä ulosottoa, vaikka tunnin kestävä esto olisi ollut vielä voimassa.

Kuva 10. ReveiwQueueInformation-komponentti näyttää vastaanotettua dataa

Tehtäväkokonaisuus 3 toimii yksinkertaisena esimerkkinä siitä, kuinka Inertian kautta lähetetään dataa ja vastaanotetaan Vue-komponentissa.

3.2.4 Tehtäväkokonaisuus 4: Linkkien lisääminen muokkaussivulle käyttäjän tarkastamiseen ja listaussivulle siirtymiseen.

Käyttäjän tarkastaminen on tärkeä ominaisuus, koska se antaa ulosottoa käsittelevälle taloushallinnoitsijalle kontekstia ulosotosta. Sieltä näkee, jos käyttäjällä on aktiivisia ulosottoja.

Toiminnallisuuden toteuttaminen on PendingForeclosureAdminController-luokan puolella yksinkertaista. Käytetään resurssiluokkaa hakemaan ulosoton käyttäjän tietoja, jotka lisätään kontrollerin Inertia-palautukseen.

EditPendingForeclosurePage-komponentin template-elementtiin lisätään kolme Inertia Link-komponenttia (Kuva 11).

```
<Link
  :href="
    $page.props.config.v1_url +
    '/view-user-foreclosures/id/' +
    user.id
  "
>
  Käyttäjän ulosotot
</Link>
```

Kuva 11. Esimerkki Inertian Link-komponentin käytöstä

Ensimmäisen Link-komponentin osoitteeksi asetetaan polku, joka vie käyttäjätietojen sivulle. Toinen Link-komponentti asetetaan viemään sivulle, jossa näytetään käyttäjän kaikki ulosotot ja kolmas Link-komponentti vie takaisin ulosottojen listaussivulle (Kuva 12).

Kuva 12. EditPendingForeclosurePage-komponentille lisätyt kolme linkkiä

3.2.5 Tehtäväkokonaisuus 5: Ulosoton tietojen palautus muokkaussivulle.

PendingForeclosureAdminController-luokassa Inertia-palautukseen lisätään activeForeclosure-attribuutti, jonka arvoksi asetetaan valitun ulosoton tiedot (Kuva 13).

```
return inertia('Foreclosures/EditPendingForeclosurePage', [
    'reviewQueueItem' => new ReviewQueueResource($reviewQueueItem),
    'user' => new UserResource($pendingForeclosure->user),
    'activeForeclosure' => $pendingForeclosure->user->getActiveForeclosure(),
]);
```

Kuva 13. Inertia-palautukseen lisätään activeForeclosure

3.2.6 Tehtäväkokonaisuus 6: Huomioviestin näyttäminen muokkaussivulla, jos usealla käyttäjällä on sama henkilötunnus.

Seuraavaksi halutaan muokkaussivulle näkyviin huomioviesti, jos ulosoton käyttäjällä ja joillain muilla käyttäjillä on sama henkilötunnus. Huomioviestissä on linkki kaikkiin käyttäjiin, joilla on sama henkilötunnus. Tämän avulla ulosottoa käsittelevää taloushallinnoitsijaa autetaan varmistamaan, että ulosotto ei mene vanhalle tilille.

Huomioviestin näyttäminen kannattaa toteuttaa EditPendingForeclosurePage-komponentissa ja henkilötunnusten hakeminen ja vertaaminen toisiinsa onnistuu helposti PendingForeclosureAdminController-luokassa.

PendingForeclosureAdminController-luokkaan luodaan $duplicateUserSsn-muuttuja. Muuttuja hakee SocialSecurityNumber-luokan avulla tietokannasta kaikki käyttäjät, joiden henkilötunnukset ovat samat kuin ulosoton käyttäjän henkilötunnus. Käyttäjät poimitaan tietokannasta yksitellen, jonka takia $duplicateUserSsn-muuttujasta tulee taulukkomuuttuja. Tämä on hyödyksi huomioviestin näyttämisessä Vue-komponentissa, koska pystytään tarkastamaan, onko taulukossa käyttäjiä ja huomioviesti näytetään silloin, kun taulukossa löytyy enemmän kuin yksi käyttäjä. Taulukon tarkastuksessa halutaan löytää enemmän kuin yksi käyttäjä, koska tietokannasta poimitaan aina ulosoton käyttäjä. Tämä tapahtuu, koska ulosoton käyttäjän henkilötunnusta verrataan kaikkiin tietokannan henkilötunnuksiin, johon kuuluu myös ulosoton käyttäjä.

Huomioviestiä varten käytetään Vuetifyn VAlert-komponenttia, koska sillä on valmiiksi sopiva ulkonäkö huomioviestille. Huomioviestissä listataan kaikkien käyttäjien id, jossa on linkki käyttäjien tietojen tarkistukseen.

3.2.7 Tehtäväkokonaisuus 7: Ulosoton tietojen hakeminen xml muodossa.

Tehtäväkokonaisuudessa 7 ja 8 halutaan hakea ja näyttää ulosoton tietoja muokkaussivulla. Tietoja haetaan ja käsitellään XML muodossa, koska tiedot tulevat XML muodossa suoraan ulosottoviranomaisilta. Jokaista ulosottoa kohti vastaanotetaan oma XML tiedosto ulosottoviranomaisilta. Tiedosto tallennetaan jokaiselle ulosotolle toimeksiantajan Amazon Web Service (AWS) tallennustilaan.

Tiedon hakemiseen tietokannasta on olemassa Storage-luokka, jonka avulla haetaan tietoa toimeksiantajan AWS tallennustilasta (Kuva 14). Haettu tieto asetetaan muuttujaan, joka lähetetään PendingForeclosureAdminController-luokasta EditPendingForeclosurePage-komponenttiin Inertia-palautuksella.

```
$xml = Storage::get('pending-foreclosures-documents/' . $pendingForeclosure->source_file);
```

Kuva 14. Storage-luokka hakee XML tiedoston toimeksiantajan AWS tallennustilasta

3.2.8 Tehtäväkokonaisuus 8: Ulosoton tietojen palauttaminen XML muodossa muokkaussivulle.

Vastaanotettu data rekisteröidään propsina EditPendingForeclosurePage-komponentissa. Jotta XML tieto muotoutuisi oikein muokkaussivulla, on käytettävä pre-tagiä. Pre-tagi on HTML-elementti, joka säilyttää syötetyn tekstin välilyönnit ja rivimuutokset. PendingForeclosureXml asetetaan aaltosulkuihin pre-tagin sisälle komponentin template-elementtiin (Kuva 15).

```
<pre class="p-4">{{ pendingForeclosureXml }}</pre>
```

Kuva 15. HTML pre-tagi, joka tulostaa pendingForeclosureXml-propsin

Ulosoton tietojen näyttäminen muokkaussivulla on tärkeää tietoeheyden ja vastuullisuuden kannalta. Tiedot vastaanotetaan suoraan ulosottoviranomaisilta ja niiden näyttäminen muokkaussivulla toimii varmistamaan, että tietokannasta saadut ja muokkaussivulla näytettävät tiedot vastaavat ulosottoviranomaisilta saatuja tietoja.

3.2.9 Tehtäväkokonaisuus 9: Lomakkeen lisääminen ulosoton muokkaussivulle.

Lomakkeen lisääminen muokkaussivulle on suuri kokonaisuus, sillä se vaatii useita syöttökenttiä, datan käsittelyä, useita komponentteja ja lomakkeiden datan tallennuspyynnön lähettämistä. Muokkaussivulle tarvittavat syöttökentät ovat:

- valintakenttä ulosoton laskentatavalle
- syöttökenttä maksukiellon numerolle
- syöttökenttä maksuviitteelle
- ulosoton alkamispäivä
- ulosoton lopetuspäivä
- ulosotettava summa yhteensä
- enintään ulosotettava summa kuukaudessa
- ulosotettava summa per palkka
- suojattu osuus per päivä
- jätettävä summa
- muu tulo per kuukausi
- ulosottoprosentti
- ylläpidon käsittelyviesti

Lisäksi luodaan lomakkeen hyväksymis- ja hylkäystoiminnot, jotka hyväksyvät tai hylkäävät käsittelyssä olevan ulosoton. Lisäksi luodaan lista, joka näyttää mitkä ulosoton kuukaudet ovat olleet maksuvapaita.

Ominaisuuksia varten pitää tuoda PendingForeclosureResource- ja ForeclosureCalculationMethod-luokat PendingForeclosureAdminController-luokkaan. Kontrolleriin luodaan $calculationMethods-muuttuja, joka käyttää collect-apufunktiota palauttamaan taulukkona kaikki ulosoton laskentatavat ja niiden avaimet sekä arvot. Inertia-palautukseen (Kuva 5) lisätään calculationMethod-attribuutti, jonka arvo on ulosottojen laskentatavat taulukkomuuttujana. Inertia-palautukseen lisätään myös pendingForeclosure-attribuutti, joka sisältää muokattavan ulosoton tietoja.

EditPendingForeclosurePage-komponentissa calculationMethod ja pendingForeclosure rekisteröidään propsina. Lomaketta varten luodaan PendingForeclosureDetails-komponentti, joka rekisteröidään EditPendingForeclosurePage-komponentin script-elementissä ja lisätään komponentti-tagin muodossa template-elementtiin (Kuva 16). PendingForeclosureDetails-komponentille syötetään calculationMethod-, pendingForeclosure- ja reviewQueueItem-propsit.

```
<PendingForeclosureDetailsVue
  :review-queue-item="reviewQueueItem"
  :pending-foreclosure="pendingForeclosure"
  :calculation-methods="calculationMethods"
/>
```

Kuva 16. PendingForeclosureDetails-komponenttiin syötetään propseja

Syöttökenttiä varten luodaan pendingForeclosureForm-lomake PendingForeclosureDetails-komponentin script-elementtiin. Lomake on tarpeellinen vastaanottamaan PendingForeclosureAdminController-luokan palauttamaa ulosottodataa. Lomakkeeseen määritellään arvoja, jotka liitetään jokaiseen syöttökenttään Vuen v-model-ominaisuudella. Lomakkeen arvoksi määritellään Vuen reactive-olio, jonka sisälle määritellään attribuutteja. Jokaiselle attribuutille määritellään arvo, joka haetaan pendingForeclosure-propsista. Osa arvoista on raha-arvoja, jotka on tallennettu sentteinä. Raha-arvoja tallennettaan tietokannassa sentteinä backend bisneslogiikan käsittelyä varten. Raha-arvot muutetaan euroiksi fromMoney-apumetodin avulla, jotta ulosottoa käsittelevällä henkilöllä olisi helpompi hahmottaa rahasummia.

PendingForeclosureDetails-komponenttiin luodaan valintakenttä ulosoton laskentatavalle. Siihen käytetään Vuetifyn VSelect-komponenttia, jolle pystyy määrittelemään valinnat syöttämällä

taulukko item-propsin kautta. Item-propsiin asetetaan calculationMethod-taulukko. Valintakenttään liitetään v-model-arvo pendingForeclosureForm.calculation_method. Nyt valintakentällä pystyy valitsemaan ulosoton laskentatavan vaihtoehtojen välillä.

Syöttökenttien luominen on yksinkertaista ja siihen käytetään Vuetifyn VTextField-komponenttia. VTextField-komponentti luo yhden rivin syöttökentän. Komponenttiin määritellään type-attribuutti, jonka arvo on input- ja v-model-argumentti, jonka arvo on esimerkiksi pendingForeclosureForm.withholding_note_number. Suurin osa syöttökentistä käyttää samaa komponenttia ja rakennetta.

Aika-arvoja käsittelevät syöttökentät käyttävät type-attribuutin arvona date, joka muuttaa syöttökentän käsittelemään päivämääriä.

Ylläpidon käsittelyviestin syöttökentälle käytetään Vuetifyn VTextarea-komponenttia. Komponentti luo syöttökentäksi tekstiruudun, joka muuten toimii samalla tavalla kuin VTextField-komponentti. V-model-argumenttiin asetetaan arvo pendingForeclosureForm.admin_notes.

Maksuvapaiden ulosottokuukausien listaamista varten käytetään Vuen v-for-argumenttia, joka listaa yksitellen valittua tietoa argumentille syötetystä taulukosta. Tässä tapauksessa v-for argumentille syötetään pendingForeclosure.free_months-taulukko ja valitun tiedon key-avaimeksi asetetaan months. Näiden avulla v-for-argumentti erottelee taulukon months-olioihin ja olioiden sisältö pystytään näyttämään. Tässä tapauksessa lisäämällä p-tagin sisälle months.month ottaa jokaisen months-olion ja etsii sieltä month-arvon ja näyttää sen tekstinä.

Hyväksymis- ja hylkäämistoiminnoille luodaan oma komponentti nimeltä PendingForeclosureActions, joka lisätään PendingForeclosureDetails-komponentin sisälle. Tällä tavalla PendingForeclosureDetails-komponentti pysyy siistimpänä ja helpommin luettavana, joka helpottaa ylläpidettävyyttä.

Hyväksymistoimintoa varten pitää luoda itse hyväksymispainike ja asynkroninen hyväksymisfunktio. Funktiosta tehdään asynkroninen, koska se tulee lähettämään kaksi pyyntöä. Ensimmäinen pyyntö tallentaa lomakkeen tiedot siltä varalta, että ulosoton käsittelijä on muuttanut tietoja. Toinen pyyntö hyväksyy ulosoton. Asynkronisuus takaa, että lomakkeen tiedot tallentuvat ennen kuin ulosotto hyväksytään. PendingForeclosureActions-komponentin script-elementtiin luodaan asynkroninen approve-funktio. Funktioon luodaan uusi pendingForeclosureData-lomake käyttämällä Inertian useForm-apumetodia. UseForm-apumetodi auttaa vähentämään vaadittavaa koodia tallennus- ja lähetyspyyntöjen yhteydessä. Uusi lomake pitää luoda koska vanha PendingForeclosureDetails-komponentin pendingForeclosureForm-lomake käsitteli raha arvoja euroina. Raha arvot pitää muuttaa takaisin senteiksi backend bisneslogiikan käsittelyä varten.

Siihen käytetään toMoney()-apumetodia, joka ottaa sille syötetyn euromäärän ja muuttaa sen senteiksi. Lomakkeen tietojen tallentamista varten lomake pitää muuttaa sopivampaan muotoon. Approve funktioon luodaan input-muuttuja (Kuva 17), joka käyttää Object.assign()-metodia keräämään pendingForeclosureData-lomakkeen arvot yhteen olioon.

```
const input = Object.assign({}, pendingForeclosureData)
```

Kuva 17. Input-muuttuja luodaan pendingForeclosureData-lomakkeen sisällöstä

Ulosoton datan päivittämistä varten on luotava reititysfunktio, joka ohjaa päivityspyynnön oikeaan kontrolleri funktioon. Admin-web-reittitiedostoon luodaan uusi reitti, joka lähettää pyynnön PendingForeclosureController-luokan change-funktioon. Funktiossa on toimintoja, jotka vastaanottavat lähetetyn datan ja tallentavat sen tietokantaan ulosoton ja käyttäjän id:n perusteella.

Approve-funktioon luodaan päivityspyyntö (Kuva 18), jolle syötetään reitti, jossa on ulosoton id ja käyttäjän id. Pyyntöön syötetään input-muuttuja, joka sisältää ulosoton tiedot, jotka halutaan tallentaa. Pyyntö ottaa yhteyttä admin-web-tiedoston ulosoton päivitysreititykseen (Kuva 19) ja ohjaa pyynnön change-tallennusfunktioon.

```
router.patch(
  route('admin.pending-foreclosure.change', {
    pending_foreclosure: props.reviewQueueItem.item_id,
    user: router.page.props.pendingForeclosure.user.id
  }),
  input
)
```

Kuva 18. Approve-funktion päivityspyyntö

```
// === Pending foreclosures
Route::prefix('/users/{user}/pending-foreclosures')
    ->group(function () {
        Route::patch('/{pending_foreclosure}', [PendingForeclosureController::class, 'change'])
            ->name('pending-foreclosure.change');
    });
```

Kuva 19. Ulosoton päivityksen reititys

Ulosoton tietojen päivittämisen jälkeen approve-funktio lähettää päivityspyynnön admin-web-reititystiedoston ulosoton hyväksyntä reititykseen (Kuva 20). Hyväksyntäreititys vastaanottaa ulosoton id:n reveiwQueueItem-parametrinä, jotta oikea ulosotto hyväksytään. Pyynnön mukana

lähetetään admin_note-parametri, joka sisältää ulosottoon liittyvän viestin. Viesti on hyödyllinen käsittelijöiden välisessä viestinnässä. Viestillä pystytään kertomaan esimerkiksi syy sille, että ulosotto hyväksyttiin.

```
router.patch(
  route('admin.review-queue.approve', {
    message: approveState.message,
    reviewQueueItem: props.reviewQueueItem.id,
    admin_note: props.pendingForeclosureForm.admin_notes
  }),
```

Kuva 20. Ulosoton approve-funktion hyväksyntä reititys

Hylkäystoimintoa varten ei tarvitse tallentaa ulosoton tietoja, koska tiedoilla ei ole merkitystä, jos ulosotto hylätään. Ulosoton hylkääminen tapahtuu samalla tavalla kuin hyväksyminen. Discard-funktio ottaa yhteyttä reittiin, joka tallentaa parametrinä lähetetyt tiedot. Erona approve-funktioon on se, että discard-funktio lähettää myös message-parametrin. Message-parametri lähettää käyttäjälle viestin, jolla pystytään kertomaan käyttäjälle ulosoton hylkäämisen syy.

Hyväksymis- ja hylkäämistoiminnoille luodaan painikkeet PendingForeclosureActions-komponenttiin. Painikkeiden luomiseen käytetään Vuetifyn VBtn-komponenttia (Kuva 21). Komponentti luo painikkeen valmiilla tyyliasetuksilla. Hyväksy-painike määritetään aktivoimaan approve-funktion painamisen yhteydessä.

```
<VBtn color="primary" class="mr-4" @click.prevent="approve">
  Hyväksy
</VBtn>
```

Kuva 21. Vuetifyn VBtn-komponentti, joka aktivoi approve-funktion

Hylkäyspainike toimii hyväksymispainikkeen tavalla, mutta siihen lisätään Vuetifyn Vdialog-komponentti, joka avaa modaalin, kun hylkäyspainiketta painetaan. Modaalissa on VtextArea-syöttökenttä, joka tallentaa käyttäjälle lähetettävän hylkäämisviestin.

3.2.10 Tehtäväkokonaisuus 10: Ulosoton arvojen näyttäminen muokkaussivun syöttökenttien vieressä.

Ulosottojen muokkaussivulla syöttökenttien vasemmassa ylälaidassa halutaan näyttää ylätunniste, jossa näkyy ulosoton tietoja. Tiedon näyttämisen tarkoitus on viestiä käsittelijälle, mikä oli ulosoton alkuperäinen arvo, kun tieto haettiin tietokannasta. Tästä on hyötyä silloin, kun käsittelijä on vaihtanut syöttökentän arvon ja hänen tarvitsee esimerkiksi vaihtaa arvo takaisin alkuperäiseen

arvoon. Silloin käsittelijä pystyy näkemään vasemmassa ylälaidassa alkuperäisen arvon ja vaihtamaan syöttökentän arvon takaisin alkuperäiseen arvoon.

PendingForeclosureDetails-komponentissa saadaan luotua ylätunniste syöttökentille käyttämällä Vuetifyn VlistSubheader-komponenttia. VListSubheader-komponenttia käytetään koska sille on valmiiksi asetettu sopiva ylätunnisteen ulkoasu. PendingForeclosureDetails-komponentin template-elementtiin tuodaan VListSubheader-komponentti, jonka sisälle syötetään näytettävä arvo, esimerkiksi fromMoney(pendingForeclosure.maximum_amount_per_salary) (Kuva 22).

```
<VListSubheader>
  {{ fromMoney(pendingForeclosure.maximum_amount_per_salary) }}
</VListSubheader>
<VTextField
  v-model="pendingForeclosureForm.maximum_amount_per_salary"
  density="comfortable"
  label="Maksimi per palkka"
  type="input"
/>
```

Kuva 22. VListSubheader-komponentti luo ylätunnisteen syöttökentän yläpuolelle

3.3 Toteutuksen tulokset

Tehtäväkokonaisuuksien toteutuksen jälkeen toimeksiantajayrityksen yhteyshenkilön kanssa pidettiin kokous, jossa katselmoitiin toteutuksen tuloksia. Tuloksia tarkasteltiin ylläpitopalvelun puolelta, jossa varmistettiin ominaisuuksien toimivuus.

Tehtäväkokonaisuus 1: Ulosottojen listaussivun luonti, linkin lisääminen listaussivulle ylläpitopalvelun navigaatiopalkkiin ja listattavien ulosottojen suodattaminen ulosoton statuksen mukaan.

Tehtäväkokonaisuus 1 toteutui onnistuneesti. Ylläpitopalvelussa taloushallinnoitsija pystyy ohjautumaan sivulle, jossa kaikki käsiteltävät ulosotot on listattu. Listaussivulla taloushallinnoitsija pystyy suodattamaan listattuja ulosottoja sillä perusteella onko ulosotto käsitelty, ei vielä käsitelty tai ilman käsittelijää. Toimeksiantajalla ei ollut mitään kommentoitavaa tehtäväkokonaisuuteen liittyen.

Tehtäväkokonaisuus 2: Muokkaussivun luonti, muokkaussivulle johtavan linkin lisääminen jokaiselle listatulle ulosotolle ja käsittelijän liittäminen käsiteltävälle ulosotolle.

Tehtäväkokonaisuus 2 oli onnistunut, koska listaussivulla jokaisella listatulla ulosotolla on linkki, joka johtaa ulosoton muokkaussivulle. Toimeksiantajalla ei ollut kommentoitavaa toteutuksesta.

Tehtäväkokonaisuus 3: Käsittelyjonon tietojen lisääminen ulosoton muokkaussivulle.

Tehtäväkokonaisuus 3:n toteutus täytti kaikki vaatimukset. Muokkaussivun oikeassa yläkulmassa näytetään, kuka käsittelee ulosottoa, milloin käsittely alkoi ja milloin käsittelyn lukitus tietylle taloushallinnoitsijalle loppuu. Taloushallinnoitsijan irrottaminen ulosoton käsittelystä toimii. Toimeksiantajalla ei ollut mitään lisättävää tehtäväkokonaisuuteen liittyen.

Tehtäväkokonaisuus 4: Linkkien lisääminen muokkaussivulle käyttäjän tarkastamiseen ja listaussivulle siirtymiseen.

Tehtäväkokonaisuus 4 toteutui, mutta linkit käyttäjän tietojen tarkasteluun ja käyttäjän muiden ulosottojen tarkasteluun ei toiminut. Syy tähän oli, että näiden sivujen toteutus ei liittynyt ulosottojen aihealueen rajaukseen. Toimeksiantajalla ei ollut tästä mitään kommentoitavaa, koska oltiin yhteisymmärryksessä siitä miksi linkit eivät toimineet. Linkki, joka palauttaa takaisin ulosottojen listaussivulle toimi hyvin.

Tehtäväkokonaisuus 5: Ulosoton tietojen palautus muokkaussivulle.

Tehtäväkokonaisuus 5:tä ei käyty läpi, koska sen toteutus näkyi ainoastaan ohjelmistokoodin puolella. Toteutuksen koodia ei loppujen lopuksi käytetty ollenkaan muissa tehtäväkokonaisuuksissa. Tämän perusteella pitäisi tarkistaa, että ulosoton tiedot ovat oikein. Tällä hetkellä tietoja palautetaan kahdella eri tavalla ulosoton muokkaussivulle. Tehtäväkokonaisuus 9 vaati toisen hakutavan luonnin, koska tehtäväkokonaisuuden 5 tapa ei palauttanut aloitus- ja lopetuspäivämääriä.

Tehtäväkokonaisuus 6: Huomioviestin näyttäminen muokkaussivulla, jos usealla käyttäjällä on sama henkilötunnus.

Tehtäväkokonaisuus 6 toteutui onnistuneesti. Huomioviesti näkyy, kun usealla käyttäjällä on sama henkilötunnus ja antaa linkin käyttäjätietoihin. Toimeksiantajalla ei ollut mitään kommentoitavaa.

Tehtäväkokonaisuus 7: Ulosoton tietojen hakeminen xml muodossa.

Tehtäväkokonaisuutta 7 ei käyty läpi, koska sen toiminta on täysin koodin puolella.

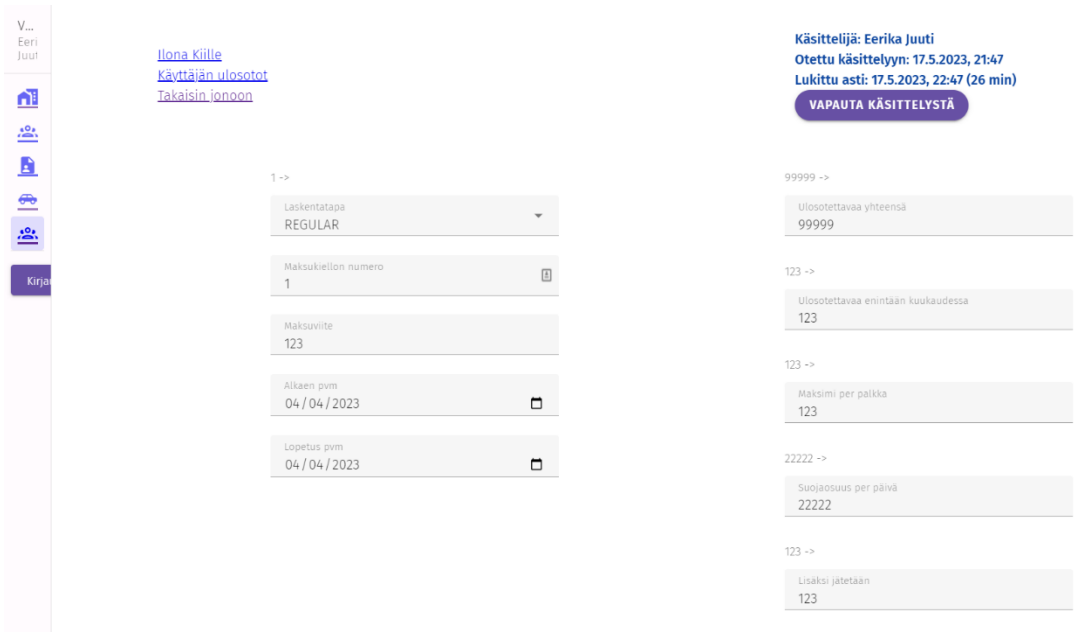
Tehtäväkokonaisuus 8: Ulosoton tietojen palauttaminen xml muodossa muokkaussivulle.

Tehtäväkokonaisuus 8 toteutui onnistuneesti. Vastaanottaa xml tiedoston ja näyttää tiedon muokkaussivulla oikeamuotoisesti välilyönneillä ja rivimuutoksilla. Tämä tarkoitta myös, että tehtäväkokonaisuus 7 toteutui onnistuneesti sillä vastaanotettu xml tiedosto vastasi odotettua tietosisältöä ja tekstimuotoa. Toimeksiantajalla ei ollut kommentoitavaa.

Tehtäväkokonaisuus 9: Lomakkeen lisääminen ulosoton muokkaussivulle.

Tehtäväkokonaisuus 9 toteutui onnistuneesti. Lisää useita toimivia syöttökenttiä muokkaussivulle ja näyttää haettua ulosottodataa syöttökentissä. Data ja hyväksytty tila tallentuu onnistuneesti tietokanttaan, kun ulosotto hyväksytään. Hylkäyksen yhteydessä hylätty tila ja viesti, joka lähetetään käyttäjälle, tallentuu tietokantaan.

Toimeksiantaja haluaa useita muutoksia muokkaussivulle. Kuvassa 23 näkyy kuinka syöttökentät on jaettu kahteen vierekkäiseen riviin. Toimeksiantajan mielestä syöttökentät pitäisi jakaa tasaisesti rivien välillä, jotta muokkaussivu olisi pienempi ja vaatisi vähemmän selaamista. Ulosoton hylkäyksen yhteydessä käyttäjälle lähetetty viesti halutaan poistaa, koska se ei ole tarpeellinen. Ulosotoilla on olemassa viranomaispuolella ulosottovastaava ja toimeksiantaja haluaa, että ulosottovastaavan sähköposti ja mahdolliset lähetetyt viestit näkyvät muokkaussivulla. Muokkaussivulla halutaan myös näyttää ulosoton tyyppi.



Kuva 23. Muokkaussivun näkymä

Tehtäväkokonaisuus 10: Ulosoton arvojen näyttäminen muokkaussivun syöttökenttien vieressä.

Syöttökenttien vasemmassa ylälaidassa näkyy syöttökenttien alkuperäinen arvo, joten tehtäväkokonaisuus toteutui onnistuneesti. Toimeksiantajalla ei ollut mitään kommentoitavaa.

Lopputulos

Lopputuloksena ylläpitopalvelun ulosottojen käsittely toimii kokonaisuudessaan ja tämä mahdollistaa vastuullisen ulosottojen käsittelyn. Toimeksiantaja ehdotti jatkokehitysideoita, joiden perusteella on luotu uusia tehtäväkokonaisuuksia ja niitä on tarkoitus toteuttaa lähitulevaisuudessa. Toimeksiantaja ehdotti esimerkiksi:

- Ohjaa taloushallinnoitsija uudelleen ulosottojen lisatussivulle hyväksymisen tai hylkäämisen jälkeen.
- Estä ponnahdusikkunat ulosoton käsittelyn jälkeen.
- Poista aktiivinen ulosotto käyttäjältä, jos ulosoton tyyppi on cancel.

Ulosottojen käsittelyn kehityshaara yhdistetään palvelutuotteen kehityshaaran kanssa tulevaisuudessa. Ulosottojen käsittely julkaistaan tuotantoon palvelutuotteen mukana kehityshaarojen yhdistämisen jälkeen.

4 Pohdinta

4.1 Monoliittisen arkkitehtuurin soveltaminen Inertialla

Keskeisiin monoliittisen arkkitehtuurin hyötyihin kuuluu helppo kehittäminen. Sanoisin, että opinnäytetyössä käsiteltävän ylläpitopalvelun ulosottojen käsittelyn toteutus hyötyi suuresti kehitettävyyden helppoudesta. Hyödyt tulivat esiin eri tavalla backend ja frontend puolella.

Backend toteutuksessa tuli käytettyä paljon palvelutuotteen olemassa olevia luokkia, jotka yksinkertaistivat tehtäväkokonaisuuksia ja käsittelivät monimutkaisia datatarpeita ilman lisäkehittämisen tarpeita.

Kehittämisen helppous oli huomattava Vue-komponenttien toteutuksessa, sillä datan käsittely ja reititys toimi eri lailla usealla tasolla. Ensinnäkin Inertiaa käyttäen frontend puolella ei tehdä reititystä ollenkaan. Sen sijaan käytettiin Ziggy kirjaston route apufunktiota aktivoimaan Laravelissa luotuja reitityksiä Vuen puolelta. Tämä oli mielestäni todella yksinkertainen ja helposti käytettävä tapa luoda navigaatioita Vue-komponenteissa. Datan käsittely oli helpompaa sillä Inertia-palautusten ja useForm-apufunktioiden ansiosta ei tarvinnut luoda välimuistin hallintaa. Ei myöskään ollut tarvetta ottaa huomioon mikropalveluiden tai ohjelmointirajapintojen aiheuttamaa viivettä, sillä data vastaanotetaan kontrolleri luokan palautuksesta. Viive on palvelutuotteen kehityksessä aiheuttanut ongelmia, jos esimerkiksi rajapinnasta haettu data pitäisi käyttää Vuen computed-attribuutissa. Computed-attribuutti on Vue-komponentissa yksi ensimmäisistä elementeistä, joka latautuu sivunlatauksen yhteydessä ja sen takia rajapinnasta haettua dataa ei voi helposti käyttää computed attribuutissa.

Viiveen puute näkyi suorituskyvyn puolella toteutuksessa. Mikropalvelu arkkitehtuurissa esimerkiksi hyväksymis- ja hylkäämistoiminnot kulkisivat ohjelmointirajapinnan kautta, joka voisi pahimmassa tapauksessa kestää useita sekunteja. Inertialla hyväksymisen ja hylkäyksen status päivittyi heti tietokantaan ja teki manuaalisesta testauksesta helpompaa.

Monoliittisen arkkitehtuurin vaikutus testaukseen jäi osittain avoimeksi. Manuaalinen testaus ohjelmoinnin aikana helpottui, mutta esimerkiksi regressiotestausta ei tehty. Ongelmatilanteiden ilmetessä virheiden löytämiseen käytettiin enimmäkseen Laravelin tarjoamia ominaisuuksia, kuten esimerkiksi dump and die-metodia. Koska kunnon testejä ei tehty ja virhetilanteita ratkottiin muilla työkaluilla, en pysty varmuudella sanomaan helpottiko vai vaikeuttiko monoliittinen arkkitehtuuri testausta.

Backend- ja frontend-toteutukset hyötyivät molemmat siitä, että olemassa olevia luokkia, metodeja, funktioita ja komponentteja pystyi käyttämään projektissa. Olemassa olevan koodin

uudelleenkäytettävyys on hyöty, jota pystyy käyttämään hyväksi ainoastaan, jos uusi monoliittinen ohjelma rakennetaan toisen palvelun päälle. Toimeksiantajan palvelutuotteessa on paljon luokkien välisiä riippuvuuksia, joiden kopioiminen uuteen projektiin olisi vaikea ja aikaa vievä prosessi. Koska ylläpitopalvelu on yritystoiminnan näkökulmasta kumppani toimeksiantajan palvelutuotteelle niin ylläpitopalvelulla on paljon vaatimuksia siitä, mitä ja miten dataa käsitellään. Riippuvuuksien ja datan käsittelyn vaatimusten takia ainoa helppo ratkaisu oli rakentaa ylläpitopalvelu monoliittisesti palvelutuotteen päälle. Sen takia ylläpitopalvelun toteutus on hyötynyt ja helpottunut huomattavasti. Palvelun kehitys monoliittisella arkkitehtuurilla ei ainoastaan hyödynnä pieniä ja aloittavia ohjelmistoja sekä palveluja. Myös suuret palvelut voivat saada siitä paljon hyötyä, kunhan käyttötarkoitus ja toteutustapa ovat selviä.

4.2 Henkilökohtainen kehittyminen

Oman kehittymisen kohdalla helpoin seurattava mittari on ollut kykyni toteuttaa tehtäväkokonaisuuksia. Toteutuksen aikana käytin enimmäkseen Inertian ja Laravelin dokumentaatiota kehittämisen tukena. Alkuvaiheissa ja tehtäväkokonaisuus 9:n kanssa pyysin usein apua kollegoilta. Opin itse parhaiten tekemällä, joten se oli pääsääntöisin tapa, jolla rakensin dokumentaatiosta lukemani tiedon käytännön ymmärtämiseksi.

Inertian kanssa oli mukava työskennellä, koska se on yksinkertainen käytännössä ja dokumentaatio on yksiselitteistä. Koin Inertian ominaisuuksien olevan hyödyllisiä ja helppoja käyttää.

Minulla ei ole ollut aiempaa kokemusta monoliittisesta arkkitehtuurista ja projektin toteuttaminen monoliittisella arkkitehtuurilla on tuonut lisää ymmärrystä sen toiminnasta, ja syistä miksi ohjelmistokehityksessä on siirrytty mikropalveluarkkitehtuuriin. Väittäisin, että monoliittisella arkkitehtuurilla on vieläkin paikkansa ohjelmistokehityksen projekteissa niin pienissä harrastusprojekteissa kuin suuremmissa palvelutuotteissa.

Lähteet

Blinowski, G. Ojdowska, A. & Przybyłek, A. 2022. ”Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation”, in IEEE Access, vol. 10, pp. 20357-20374, 2022, doi: 10.1109/ACCESS.2022.3152803. Luettavissa: https://ieeexplore.ieee.org/document/9717259 Luettu: 26.4.2023

De Lauretis, L. 2019. "From Monolithic Architecture to Microservices Architecture," 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Berlin, Germany, 2019, pp. 93-96, doi: 10.1109/ISSREW.2019.00050. Luettavissa: https://ieeexplore.ieee.org/document/8990350 Luettu: 26.4.2023

Dehghani, Z. 24.4.2018. How to break a Monolith into Microservices. martinFowler.com. Luettavissa: https://martinfowler.com/articles/break-monolith-into-microservices.html. Luettu: 10.4.2023

Doglio, F. 2.2.2022. Microservices aren’t always the answer: a case for monoliths. Luettavissa: https://blog.bitsrc.io/when-microservices-are-not-the-answer-a-case-for-monoliths-895ccefa2728 Luettu: 16.4.2023

Ford, N., Richards, M., Sadalage, P & Dehghani, Z. 2021. Software Architecture: The Hard Parts. O’Reilly Media, Inc. Sebastopol, CA 95472. E-Kirja. Luettu: 27.4.2023

IBM, Cyclomatic Complexity, Luettavissa: https://www.ibm.com/docs/en/raa/6.1?topic=metrics-cyclomatic-complexity Luettu: 14.4.2023

Inertia. JavaScript apps the monolith way. Luettavissa: https://inertiajs.com/ . Luettu: 14.4.2023

Inertia. Who is Inertia.js for? Luettavissa: https://inertiajs.com/who-is-it-for. Luettu: 14.4.2023

Inertia. How it works. Luettavissa: https://inertiajs.com/how-it-works. Luettu: 14.4.2023

Inertia. The protocol. Luettavissa: https://inertiajs.com/the-protocol. Luettu: 14.4.2023

Inertia. Pages. Luettavissa: https://inertiajs.com/pages. Luettu: 20.4.2023

Inertia. Responses. Luettavissa: https://inertiajs.com/responses. Luettu: 20.4.2023

Inertia. Links. Luettavissa: https://inertiajs.com/links. Luettu: 20.4.2023

Inertia. Routing. Luettavissa: https://inertiajs.com/routing. Luettu: 20.4.2023

Inertia. Forms. Luettavissa: https://inertiajs.com/forms. Luettu: 20.4.2023

Martin, R. 2017. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson. E-Kirja. Luettu: 26.4.2023

MDN Web Docs. XMLHttpRequest. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest Luettu: 14.5.2023

Ozkaya, M. 17.2.2023. Application Architecture. Luettavissa: https://medium.com/design-microservices-architecture-with-patterns/architecture-comparison-monolithic-vs-microservices-4109265c4806 Luettu: 15.5.2023