HAMK
Hämeen ammatti-
korkeakoulu

Reskontran ulkopuolisten kirjanpitovientien syöttö Evitec Power Lending -tuotteeseen

Ammattikorkeakoulututkinnon opinnäytetyö

Tieto- ja viestintätekniikka, insinööri (AMK)

Syksy 2024

Nina Tulilahti

HAMK
Hämeen ammatti-
korkeakoulu

Tiivistelmä

Koulutus	Tieto- ja viestintätekniikka	
Tekijä	Nina Tulilahti	Vuosi 2024
Työn nimi	Reskontran ulkopuolisten kirjanpitovientien syöttö Evitec Power Lending -tuotteeseen	
Ohjaaja	Petri Kuittinen (HAMK), Timi Voutilainen (Evitec Solutions)	

Opinnäytetyön tarkoituksena oli kehittää Evitec Power Lending -sovelluksen kirjanpitomoduuliin lisäosa, joka mahdollistaa kirjanpidon tositteiden tekemisen manuaalisesti. Työtä tehtiin osana ketterää järjestelmäkehitystiimiä, jolloin opinnäytetyöprosessiin kuuluivat sovelluksen kehittämisen lisäksi päivittäispalaverit, määrittelyt, toteutuksen esittäminen sidosryhmille sprintin päätteeksi sekä tiimin sisäiset palaverit työskentelytapojen kehittämiseksi. Työn toimeksiantaja oli Evitec Solutions.

Opinnäytetyössä perehdyttiin full stack -sovelluskehitykseen. Työn teknisessä keskiössä olivat .NET Framework, React, TypeScript, Azure ja SQL Server, jotka ovat keskeisiä tekniikoita web-sovelluksen kehittämisessä. Versiohallinnassa käytettiin sekä komentoriviä että graafisen käyttöliittymän tarjoavaa Fork-sovellusta. Palvelinpuolen kehityksessä editorina käytettiin Visual Studiota ja asiakaspuolen kehityksessä Visual Studio Codea.

Opinnäytetyön tuloksena valmistui manuaalisten kirjanpidon tositteiden lisäämiseen tarkoitettu modaali, jossa käyttäjä pystyy lisäämään yhden tai useita kirjanpitovientejä yhdellä kirjanpidon tositteella. Modaalissa voidaan lisätä uusia kirjanpitovientejä tarpeen mukaan ja poistaa tarpeettomia lomakkeita. Kirjanpitoviennit noudattavat kirjaussääntöjä, jotka jokaiselle tapahtumalle on määritelty aikaisemmin.

Lisäksi toteutettiin mahdollisuus lisätä muu-tyyppisiä kirjanpitovientejä, jotka eivät noudata mitään valmiita kirjaussääntöjä. Muu-tyyppisellä kirjauksella käyttäjä määrittelee kaikki tarvittavat tiedot täysin manuaalisesti. Tietokantaan lisättiin uusi kirjaustyyppi muu, ja sitä vastaava tapahtumalajikoodi.

Toteutuksen kehittäminen jatkuu vielä opinnäytetyön valmistumisen jälkeen. Toteutukseen lisätään neljän silmän tarkistusperiaate, joka vaatii toiselta käyttäjältä hyväksynnän manuaalisille kirjanpitotapahtumille. Tällä ehkäistään inhimillisiä virheitä sekä väärinkäytöksiä. Lisäksi toteutuksen jatkokehittämisessä otetaan huomioon demon pohjalta nousseet kehitysideat.

Avainsanat	Full stack -sovelluskehitys, ketterä järjestelmäkehitys, kirjanpito, web-sovellus
Sivut	64 sivua ja 3 liitettä

HAMK
Hämeen ammatti-
korkeakoulu

Abstract

DP	Information and Communication Technology	
Author	Nina Tulilahti	Year 2024
Subject	Adding External Ledger Accounting Entries into Evitec Power Lending Product	
Supervisors	Petri Kuittinen (HAMK), Timi Voutilainen (Evitec Solutions)	

The goal of this thesis was to create an add-on module for the Evitec Power Lending Product, which enables creating manual postings into an accounting system. The thesis was conducted as a part of an agile system development team, which means that the project included sprint planning, daily meetings, task refinements, demos with a customer and a retrospective. The commissioner of the thesis was Evitec Solutions.

The focus of this thesis was full-stack application development. The technical core of the thesis included .NET Framework, React, TypeScript, Azure, and SQL Server, which are widely used technologies in web development. Command line and the Fork application were used for the version control. The Visual Studio editor was used for the backend development and the Visual Studio Code editor for the frontend development.

As a result of this thesis, a new modal for adding manual postings was created. Users can send one or several posting entries at the same time. Posting entries can be removed or added in the posting modal form. All posting entries follow posting set rules that have been predefined for each transaction.

A feature that allows users to add posting entries with other event types was also created. The other event type has no posting set rules so users must define all needed information manually. A new event type and event type code was added to the database.

The development of the manual posting modal will continue. The next step would be to implement a verification system that would require another user to verify that all information is correct. This helps mitigate potential human errors and mismanagements. In addition, the further development of the implementation will take into account the wishes and development ideas that emerged from the demo.

Keywords	Accounting system, agile, full stack application, web development
Pages	64 pages and appendices 3 pages

Sisällys

Kuvat

Liitteet

Kirjanpidon keskeiset käsitteet

Antolainaus: Lainan tai luoton myöntäminen yksityishenkilöille, yrityksille tai julkisyhteisöille. Antolainaus on osa pankkien ja rahoituslaitosten perustoimintaa, jossa lainanottajalta veloitetaan korkoa.

Debet ja kredit: Jokaisella tilillä on kaksi puolta, joista toiselta puolelta lähtee rahaa ja toinen vastaanottaa rahaa. Jokainen kirjanpitoon merkittävä summa merkitään aina kahdelle tilille: toiseen debet puolelle ja toiseen kredit puolelle. Debet puolelle merkitään kaikki saapuvat tulot ja kredit puolelle merkitään lähtevät maksut. Kehitysympäristössä termeistä käytetään nimiä per (debet) ja an (kredit) ja nämä termit näkyvät kehitysympäristössä otetuissa kuvakaappauksissa. Käännöskirjaston avulla eri asiakkaille voidaan näyttää termit asiakkaan haluamalla tavalla.

Eräajo: Prosessi, jossa suoritetaan suuri joukko tehtäviä yhdellä kertaa. Eräajot voivat olla esimerkiksi raporttien tuottamista, tiedostojen siirtämistä tai tietojen päivittämistä. Eräajoilla voidaan hallita myös monimutkaisempia prosesseja kuten laskutusta tai asiakastietojen hallintaa. Eräajo on automatisoitu prosessi, mikä vähentää manuaalisen työn määrää ja täten myös virheiden määrää. Toisaalta se on myös hidas ja raskas prosessi, minkä takia eräajot laitetaan pyörimään sellaisena ajankohtana, jolloin järjestelmällä on vähemmän käyttäjiä, esimerkiksi yöaikaan.

Kirjanpidon tosite: Kokonaisuus, joka voi sisältää useita kirjanpitovientejä. Evitec Power Lending -järjestelmässä kirjanpidon tosite liittyy tavallisesti jollekin sopimukselle. Kirjanpidon tositteen tietoja ovat mm. tapahtumapäivä, kirjauspäivä sekä lista yksittäisistä kirjanpitovienneistä.

Kirjanpitovienti: Yksittäinen taloudellinen tapahtuma, joka kirjataan yrityksen kirjanpitoon. Taloudellisia tapahtumia ovat esimerkiksi ostot, myynnit, palkkojen maksamiset ja lainanotto. Evitec Power Lending -järjestelmässä kirjanpitoviennin tietoja ovat mm. tapahtumatyyppi, tapahtumalajikoodi, rahamäärä sekä kirjauksen suunta.

Kirjaussäännöt: Kokoelma sääntöjä ja ohjeita, jotka määrittelevät, miten kirjanpitoviennit merkitään kirjanpitoon. Jokaiselle kirjaustyypille liittyy omat kirjaussääntönsä. Säännössä määritellään kirjauksen suunta (debet / kredit), liikekirjanpidontili, talousarviotili ja pankkitili.

Kirjaustyyppi: Muutamia esimerkkejä tapahtumatyypeistä Evitec Power Lending järjestelmässä ovat mm. hoitopalkkio, korko, lyhennys, myynnin ALV, palkkio ja pidätys. Kirjaustyypit liittyvät tuotteeseen ja määrittävät tapahtumalajikoodin.

Leasing: Rahoitusjärjestely, joka mahdollistaa käyttöomaisuuden hankkimisen vuokraamalla se pitkäaikaisesti, tavallisesti usean vuoden ajaksi. Vuokralle ottaja ei tarvitse suurta alkuinvestointia leasingtuotteen hankkimiseen. Rahoitusleasingissa leasingtuotteesta tulee sopimuskauden lopussa automaattisesti vuokralle ottajan omaisuutta. Käyttöleasingissa omistus ei siirry automaattisesti vuokralle ottajan haltuun, mutta usein vuokraajalla on mahdollisuus ostaa tuote edulliseen hintaan sopimuskauden lopussa.

Liikekirjanpidontili: Tili, jonne kirjataan liiketoiminnan tapahtumia. Erilaisia liikekirjanpidontilejä ovat esimerkiksi tulostilit, tasetilit, kulutilit ja tuottotilit.

Reskontra: Kirjanpidon apujärjestelmä, jossa yrityksen maksutapahtumia seurataan yksityiskohtaisesti. Reskontra jaetaan tavallisesti kahteen päätyyppiin. Myyntireskontra seuraa yrityksen saamisten tilaa, eli maksamattomia laskuja. Ostoreskontra seuraa yrityksen ostovelkoja, eli maksettavia laskuja toimittajille. Myös lainareskontra on usein käytössä isommilla yrityksillä. Kirjanpidon tapahtumat kirjataan sekä reskontriin että kirjanpidon tileille, mutta valvonta tapahtuu reskontrien kautta.

Regulaatio: Sääntely tai määritykset, joilla pyritään varmistamaan toiminnan laillisuus, tehokkuus ja turvallisuus. Taloudellisella regulaatiolla valvotaan markkinoiden toimintaa ja estetään väärinkäytöksiä.

Talousarviotili: Tili, jonne budjetoidaan ennakoituja tuloja tai menoja. Talousarviotilien avulla seurataan kuinka toteutuneet tulot ja menot vastaavat budjetoituja summia.

Pankkitili: Evitec Power Lending järjestelmässä pankkitilillä tarkoitetaan IBAN-muotoista tilinumeroa. Suomalainen tilinumero on FI-alkuinen, esimerkiksi FI12 3456 7890 1234 56.

Vakuushallinta: Pankkien tai rahoituslaitosten toimintaa, jossa ne hoitavat ja hallinnoivat vakuuksia, joita asiakkaat antavat lainojen vakuudeksi. Vakuuksia voivat olla esimerkiksi kiinteistöt, osakkeet tai muut omaisuuserät, jotka toimivat lainan myöntämisen ehtona ja turvana lainanantajalle.

1 Johdanto

Opinnäytetyön tavoitteena on toteuttaa laajennus Evitec Power Lending järjestelmän kirjanpitomoduuliin, joka mahdollistaa reskontran ulkopuolisten kirjanpitovientien lisäämisen manuaalisesti. Tämä kokonaisuus toteutetaan työskentelemällä osana ketterää järjestelmäkehitystiimiä.

Tavoitteena on luoda tyylillisesti yhtenäinen lisäominaisuus, joka voidaan liittää suoraan jo olemassa olevaan toteutukseen. Kirjanpitomoduuliin lisätään pluspainike, jota klikkaamalla käyttäjä pystyy tekemään manuaalisesti uuden kirjanpidon tositteen. Yksi kirjanpidon tosite voi pitää sisällään useamman kirjanpitoviennin. Kirjanpitovientejä täytyy voida lisätä, poistaa ja muokata. Lisätyt kirjanpitoviennit näkyvät kirjanpidon tositteella haitareina, joita käyttäjä voi halutessaan avata ja sulkea lisätietojen katselua varten. Kirjanpidon tositteen loppuun lisätään taulukko, jossa eri tileihin kohdistuneet maksut ja tuotot on laskettu yhteen.

Toteutuksessa tulee noudattaa olemassa olevia kirjaussääntöjä. Kirjanpidon tosite sekä kaikki tositteelle liittyvät kentät tulee validoida niin, että ne vastaanottavat vain oikeantyyppistä dataa. Kirjanpidon tositteelle syötettyjen debet- ja kreditvientien tulee täsmätä ja ne tulee ohjata oikeille tileille. Tilit esitetään alasvetovalikossa, josta käyttäjä voi valita oikean tilin.

Työelämässä on selkeä tarve manuaalisten kirjanpitovientien tekemisen mahdollisuudelle. Vaikka kirjanpitäjän arkipäiväisessä työssä kirjanpitotapahtumien luominen on automatisoitu, on silti tilanteita, joissa kirjaukset täytyy pystyä tekemään manuaalisesti. Esimerkkejä tällaisista tilanteista on virheellisten tapahtumien korjaaminen, EU:n ulkopuolelta tulevien maksujen erityiskäsittelyt, kuten tullimaksujen kirjaaminen, sekä tilinpäätösoikaisut maksukauden lopussa. Olemassa olevaa kirjanpidon tositetta ei saa koskaan suoraan muokata tai poistaa, vaan virheelliset kirjaukset tulee käsitellä korjaavalla kirjanpidon tositteella. Manuaalisten kirjausten mahdollisuus onkin olennainen osa toimivaa kirjanpitojärjestelmää, jotta kaikki taloudelliset tapahtumat voidaan käsitellä oikein ja ajantasaisesti.

Toteutuksessa käytettäviä tekniikoita ovat Azure, ASP.NET Core, React, TypeScript ja SQL Server, jotka soveltuvat isojen yritysten tarpeisiin moniulotteisten sovellusten

kehittämisessä. Kaikkien edellä mainittujen tekniikoiden etuja ovat skaalautuvuus, suorituskyky, tuki modulaariseen arkkitehtuuriin, joustavuus ja ylläpidettävyys.

Opinnäytetyössä käsitellään teknisen toteutuksen lisäksi keskeisimmät kirjanpitoon liittyvät termit, jotta myös taloushallintoalaan vähemmän perehtyneet lukijat voivat ymmärtää toteutusta. Tietoteknistä termistöä ei käydä läpi. Aiheen ulkopuolelle on rajattu tietoturva ja suorituskykymittaukset. Manuaalisten kirjanpitovientien toteuttaminen Evitec Power Lending -järjestelmään tapahtuu osana ketterää järjestelmäkehitystiimiä, joten tarkemman tarkastelun ulkopuolelle jää sellaiset osat toteutusta, jotka ovat toisen tiimin jäsenen toteuttamia.

Opinnäytetyön tuloksella tulee olemaan merkittävä vaikutus Evitec Power Lending -järjestelmässä. Se lisää tuotteen joustavuutta, tehokkuutta, ja sen hyöty on erityisen merkityksellinen silloin kun kirjanpitäjä kohtaa erityiskäsittelyä vaativan kirjauksen. Järjestelmän on tarjottava ratkaisu myös niihin tilanteisiin, joita automatiikka ei pysty ratkaisemaan. Manuaalisten kirjanpitovientien mahdollistamisen myötä erityiskäsittelyä vaativiin tilanteisiin pystytään reagoimaan nopeasti. Aikaisemmin tämä ei ole ollut mahdollista.

Manuaalisissa kirjanpitovienneissä varjopuolena on mahdollisuus ihmisen tekemälle inhimilliselle virheelle, ja siitä syystä toistuvien kirjausten tekemiseen on parempi käyttää automaatiota. Virheiden minimoimiseksi kaikki manuaalisten kirjanpitovientien modaaliin lisättävät kentät validoidaan niin, että ne vastaanottavat vain oikean tyyppistä dataa. Lisäksi kirjanpidon tositteen tarkastuksessa otetaan käyttöön niin sanottu neljän silmän periaate, jonka mukaan toisen käyttäjän on manuaalisesti hyväksyttävä uusi kirjanpitovienti, ennen kuin se pääsee järjestelmästä läpi. Tähän tarkoitukseen luodaan erilliset tilat tarkastusta odottaville ja hyväksytyille kirjanpitovienneille.

Tämän opinnäytetyön avulla pyritään ratkaisemaan käytännön haaste, jossa automatisoitu kirjanpito ei riitä käsittelemään kaikkia erityistilanteita, joita kirjanpitojärjestelmässä ilmenee. Manuaalisten kirjanpitovientien toteutus tuo järjestelmään joustavuutta ja se varmistaa, että kirjanpitäjillä on käytössään kattavat työkalut myös poikkeuksellisten tapahtumien käsittelyyn. Toteutuksen onnistuminen parantaa Evitec Power Lending -järjestelmän käyttäjäkokemusta ja luo pohjan entistä joustavammalle taloushallinnon työkalulle.

2 Yleiskatsaus Evitec Power Lending -järjestelmään

Evitec Power Lending on kokonaisvaltainen antolainauksen ja vakuushallinnan elinkaarijärjestelmä (Evitec Solutions, n.d.), joka on ollut tuotannossa vuodesta 2017. Järjestelmä on suunnattu rahoituslaitoksille ja muille lainaamista tekeville yrityksille, kuten pankeille. Arkkitehtina Evitec Power Lending -järjestelmän parissa työskentelevä Jussi Kauhanen kertoo haastattelussa (henkilökohtainen tiedonanto, 4.10.2024), että Power Lendingin ajatuksena on olla järjestelmä, jossa asiakas saa hoidettua lainaamisen kaikki vaiheet yhden järjestelmän sisällä.

Prosessi alkaa siitä, että järjestelmä vastaanottaa lainan ulkoisen integraation kautta. Sen jälkeen laina pyörii järjestelmässä. Evitec Power Lending tekee muun muassa maksuohjelman laskentaa, laskutusta ja maksujen kohdistusta lainalle. Myös sopimusmuutokset ja muutoshallinta ovat osa Evitec Power Lendingin työkalupakkia. Joskus joudutaan toteamaan, että asiakas onkin maksukyvytön, jolloin laina siirtyy perintään. Tässä vaiheessa Evitec Power Lendingin rooli tulee päätökseen, mutta perintää varten Evitecillä on myös ratkaisu: perintäjärjestelmä Gorilla, Kauhanen kertoo. Usein nämä kaksi järjestelmää kulkevatkin käsi kädessä. Oletusarvoinen prosessin kulku on sellainen, jossa laina pyörii järjestelmässä niin kauan, että jossain vaiheessa se on maksettu pois ja sopimus päättyy, Kauhanen jatkaa.

Kirjanpito on tärkeässä osassa antolainausjärjestelmän elinkaarta. Kirjanpidossa varmistetaan, että lainojen elinkaari on asianmukaisesti dokumentoitu ja että yritys noudattaa taloudellisia velvoitteitaan. Lainoja myönnettäessä syntyy kirjanpidollinen merkintä lainasummasta sekä mahdollisista hallintokuluista. Jokainen lainan lyhennys kirjataan kirjanpitoon, samoin kuin kertyneet korot ja muut mahdolliset maksut. Kirjanpito antaa myös tärkeää tietoa lainojen elinkaaren eri vaiheista, joita tarvitaan raportointia ja sääntelyä varten. Finanssialan sääntelyviranomaiset voivat vaatia raportteja lainakannasta, maksusuorituksista ja mahdollisista riskienhallintatoimista (Finanssivalvonta, 2018).

Evitec Power Lending on myös erittäin skaalautuva ja asiakkaan tarpeisiin mukautuva järjestelmä. Lainaamiseen liittyvä prosessi saattaa eri tahoilla olla hyvinkin erilainen. Sama prosessi saatetaan tehdä hyvin eri tavoin tai asioista puhua eri nimillä. Evitec Power Lending pystyy tarjoamaan asiakaskohtaisesti räätälöidyn version järjestelmästä ja eri moduuleiden näkyvyyttä voidaan säädellä asiakkaiden välillä. Käyttöliittymän räätälöiminen onnistuu hyvin yksityiskohtaisellakin tasolla esimerkiksi tarpeettomia kenttiä piilottamalla tai

lisäämällä sellaista mitä järjestelmästä ei vielä löydy, kertoo tuoteomistajana työskentelevä Mari Bazia. (Henkilökohtainen tiedonanto, 5.9.2024)

Evitec Power Lending koostuu moduuleista, jotka kattavat rahoituksen koko elinkaaren. Näillä moduuleilla voidaan hallita lainoja, vakuuksia, luottopäätöksiä ja luottoriskilaskelmia. Vakuushallinta ja reskontra ovat tällä hetkellä järjestelmän valmiimpia osioita. Tavallisesti lainassa on mukana jonkinlainen vakuus tai vakuuksia, joita hallitaan järjestelmässä. Niitä voivat olla panttaukset, takaukset, erilaiset kirjeet, dokumentit ja muut todistukset. Myös vakuusarvolaskenta ja kohdistusten laskenta ovat osa Evitec Power Lendingin ominaisuuksia. Niiden avulla arvioidaan kattaako vakuusarvo lainan arvon, ja mikäli vajetta jää, niin kuinka paljon. Tällaiset ovat raportoinnin kannalta hyvin merkittäviä asioita luoton riskiä arvioidessa, Kauhanen kertoo.

Myös Kauhanen nostaa haastattelussaan esiin Evitec Power Lendingin mahdollisuudet asiakaskohtaiseen räätälöintiin: eri tahoilla saattaa olla esimerkiksi erilaisia lyhennystapoja lainoissa tai kohdistukset saattavat mennä eri tavalla, Kauhanen kertoo esimerkkeinä. Myös vakuusarviot saattavat poiketa, tai samaa asiaa halutaan tehdä hieman eri parametreillä, hän jatkaa.

Muihin vastaaviin järjestelmiin verrattaessa sekä Kauhanen että Bazia nostavat esille integraatiot, jotka ovat isossa roolissa Evitec Power Lending järjestelmässä. Niiden avulla keskustellaan eri järjestelmien välillä, mikä vähentää tiedon manuaalista syöttämistä. Prosesseja automatisoidaan niin pitkälle kuin mahdollista. Yhtenä esimerkkinä voitaisiin mainita, vaikka laskutus, Kauhanen kertoo.

Vastaaviin tuotteisiin verrattuna Evitec Power Lending on modernimpi vaihtoehto, mikä näkyy sekä käytettävissä teknologioissa että käyttöliittymällä. Evitec Power Lending pyrkii olemaan intuitiivinen käytettävyydeltään ja vastaamaan nykyaikaisen ohjelmiston odotuksiin nopealla reagoinnilla. Integraatiot ja automaatio ovat tärkeässä roolissa, sillä niiden avulla Evitec Power Lending pystyy reagoimaan muutoksiin välittömästi vanhempien ohjelmien nojatessa pitkälti ainoastaan eräajoihin. Integraation kautta tieto saadaan välittömästi ilman että tarvitsee odotella eräajon valmistumista, Kauhanen kertoo. Eräajot ovat toisaalta osa finanssialan liiketoimintalogiikkaa, ja myös Evitec Power Lendingissä on toimintoja, jotka pohjaavat eräajoihin, Kauhanen täydentää.

Evitec Power Lending on SaaS-ratkaisuna tarjottava palvelu ja sitä kehitetään jatkuvasti. Sitä kautta markkinoiden muutoksiin ja asiakkaiden tarpeisiin pystytään reagoimaan

nopeasti ja järjestelmä pysyy kilpailukykyisenä. Kevyt ja ketterä toimintamalli helpottaa moniin haasteisiin vastaamista ja uudet kehityskohteet voidaan toteuttaa ja saattaa asiakkaalle nopealla aikataululla. Esimerkiksi leasing-toiminnallisuutta on kehitetty Evitec Power Lending -järjestelmään asiakkaan tarpeen pohjalta, Bazia kertoo. Toisinaan tulee myös tilanteita, jolloin regulaatioon on reagoitava. Lisäksi integraatiot ovat ulkoisissa järjestelmissä ja ne saattavat välillä muuttua, joten niihinkin on reagoitava nopeasti, Kauhanen täydentää.

Yleisesti voidaan todeta, että Evitec Power Lendingin liiketoiminta-ala on todella laaja. Integraatioiden kautta suuri määrä järjestelmiä on kytketty toisiinsa. Rajan vetäminen sille, mitä otetaan osaksi järjestelmää ja mikä rajataan sen ulkopuolelle, on toisinaan haastavaa, sillä järjestelmästä ei toisaalta haluta liian massiivista ja kompleksista kokonaisuutta. Toisaalta Evitec Power Lending on vielä elämänsä alkuvaiheessa ja sitä kehitetään jatkuvasti, ja tuoteaspekti jossain määrin vielä hakee muotoaan, Kauhanen pohtii. Power Lendingillä on kuitenkin potentiaalia kansainvälisillekin markkinoille, ja tuotekehitysyksikkö pitää tuotteen ajan hermoilla, Bazia täydentää.

3 Työskentelytavat

Evitec Power Lending -tiimissä sovelletaan ketterää järjestelmäkehitysmallia. Siihen kuuluu Azure DevOps -ympäristössä työskentely, scrum periaatteiden noudattaminen ja versionhallinta. Ketterän järjestelmäkehitysmallin pääperiaatteisiin kuuluu läpinäkyvyys projektin ja yksittäisten tehtävien tilanteesta, mihin Azure DevOps -ympäristö tarjoaa monia hyödyllisiä työkaluja. Ohjelmistokehitystä tehdään tiimityönä, ja tiimin sisäisessä työskentelyssä hyödynnetään mm. Teams-videopuheluita ja koodin jakamista Live Share -toiminnolla. Lisäksi toteutustavoista sekä tekniikoista keskustellaan sisäisesti.

3.1 Azure DevOps -ympäristössä työskentely

Projektin suunnittelu tapahtuu Azure DevOps -ympäristön Boards-näkymässä. Tuotteen kehitysjonolle (product backlog) kirjataan käyttäjätarinat ja niihin liittyvät tarkemmin määritellyt tehtävät. Tuotteen kehitysjonoa ylläpidetään ja työstetään jatkuvasti, ja sieltä käy ilmi esimerkiksi tehtävien prioriteetti. Sprintin suunnittelu tapahtuu Sprints-välilehdellä. Sprinttiä suunnitellessa tuotteen kehitysjonolta valitaan sopiva määrä tehtäviä toteutettavaksi niille määritetyn tärkeysjärjestyksen tai sidosryhmien toiveiden mukaisesti. Valitut tehtävät asetetaan sprintin kehitysjonolle (sprint backlog).

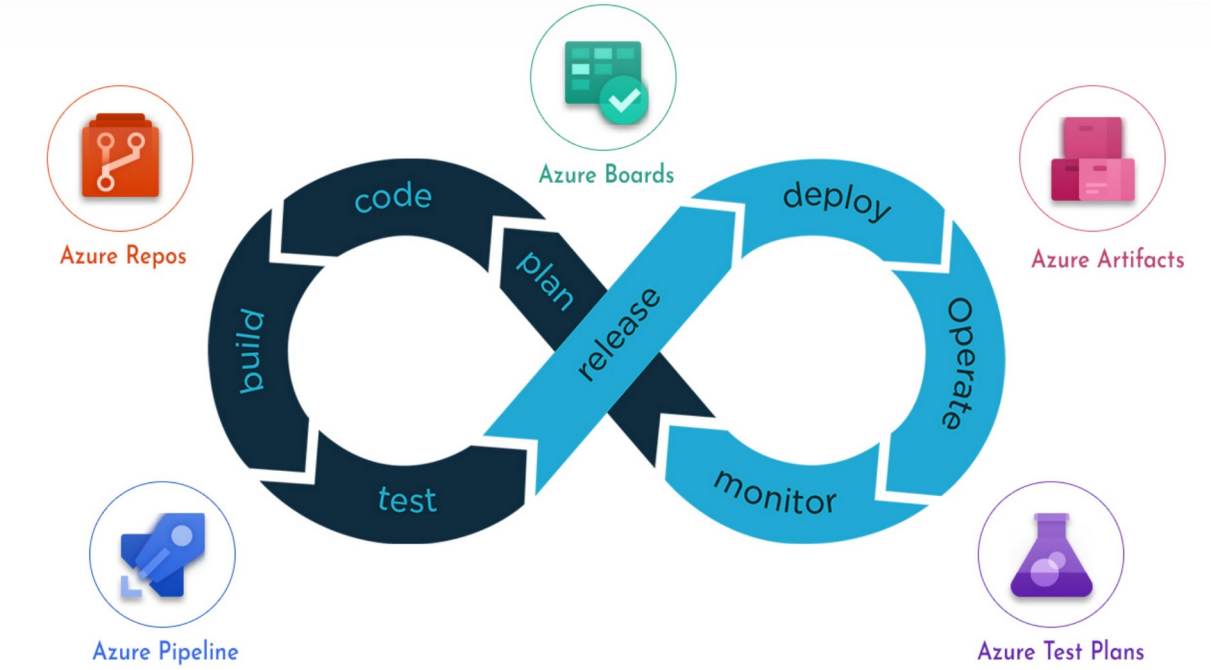
Versionhallinta tapahtuu myös Azure DevOps -ympäristössä, ja Repos-näkymä on suunniteltu sitä varten. Versiohallinnassa sovelletaan usean repositorion mallia eli niin sanottua multi-repo -mallia, mikä mahdollistaa mikropalveluarkkitehtuurin mukaisen kehityksen. Mikropalveluarkkitehtuurilla tarkoitetaan sellaista ohjelmistokehitysmallia, jossa sovellus jaetaan useampiin, pienenpiin kokonaisuuksiin, jotka toimivat itsenäisesti (Syrjä, 2016, s. 11). Multi-repo -malli tarjoaa joustavuutta, sillä se mahdollistaa pienempien kokonaisuuksien itsenäisen ja toisistaan riippumattoman kehityksen.

Repos-näkymän alta löytyy muun muassa välilehdet Files, Commits ja Branches. Nämä työkalut auttavat seuraamaan, mitä muut tiimin jäsenet ovat tehneet omilla kehityshaaroillaan ja avun tarjoaminen on silloin helpompaa. Files -näkymässä voidaan tarkastella projektin kaikkia tiedostoja eri kehityshaaroilla. Yläreunan valikosta voidaan valita develop-haara tai jokin olemassa olevista kehityshaaroista. Files-välilehti on hyödyllinen myös silloin, jos kehittäjä on vahingossa rikkonut omalla haaralla jotain. Tällöin voidaan Files-näkymästä tarkistaa, miltä tietty tiedosto näyttää develop-versiossa ja sitä kautta palauttaa rikkoutunut koodi entiselleen. Commits-välilehdellä voidaan seurata muutoksia commit-tasolla. Myös tällä välilehdellä voidaan valita commitien tarkastelu develop-haaralla tai millä tahansa olemassa olevalla kehityshaaralla. Branches-välilehti tarjoaa näkymän kaikista projektiin liittyvistä haaroista.

Pipelines-näkymässä voidaan hallita erilaisia automaatioita. Sieltä löytyy muun muassa automatisoidut testit, jotka ajetaan aina kun uusia commiteja pusketaan repositorioon ja päivittäin ajettavat testiautomaatiot. Pipelines mahdollistaa myös npm-pakettien rakentamisen ja julkaisun täysin automatisoidusti. Luodut npm-paketit voidaan hakea Artifacts-näkymästä.

CI/CD Pipelinella tarkoitetaan jatkuvan integraation (Continuous Integration) ja jatkuvan toimituksen (Continuous Delivery) prosessia, joka kulkee päättymättömässä syklissä. Tämä sykli kattaa ohjelmiston kehitys- ja toimitusprosessin. Jatkuvalla integraatiolla tarkoitetaan prosessia, jossa kehittäjät yhdistävät säännöllisesti koodimuutoksia yhteiseen versionhallintaan. Jatkuva toimitus kuvaa prosessia, jossa koodi on tarkistettu ja aina valmiina julkaistavaksi, kun se on ajettu automaatioiden läpi. Kuvassa 1 havainnollistetaan visuaalisesti CI/CD Pipelinea päättymättömänä silmukkana. Azure DevOps työkalut on sijoitettu kuvassa vastaamaan niitä kehitysprosessin vaiheita, joissa niitä tavallisesti käytetään.

Kuva 1. Azure DevOps-työkalut CI/CD Pipelinen ympärillä (SingularTech, n.d.).



3.2 Scrum-käytäntöjen soveltaminen

Evitec Solutionsin Power Lending -tiimissä työskennellään kahden viikon mittaisissa sprinteissä. Uusi sprintti alkaa suunnittelupalaverilla (sprint planning), jossa valitaan uudelle sprintille otettavat tehtävät, asetetaan ne sprintin kehitysjonolle ja jaetaan kehittäjien kesken. Ennen sprintille tulevien tehtävien valintaa arvioidaan, kuinka suuri työmäärä todennäköisesti pystytään toteuttamaan. Arvio perustuu tiimin yhteenlaskettuihin työtunteihin seuraavan kahden viikon aikana sekä aikaisemmalla sprintillä valmistuneisiin tehtäviin ja niiden story points -pisteisiin.

Sprintin suunnittelun jälkeen jokainen uusi päivä alkaa päivittäispalaverilla, joka on lyhyt, maksimissaan 15 minuutin mittainen palaveri. Tiimin jäsenet kertovat vuorotellen lyhyesti mitä ovat edellisenä päivänä tehneet, mitä aikovat tehdä seuraavaksi ja onko työn tekemisessä ilmennyt jonkinlaisia esteitä. Tämä tuo läpinäkyvyyttä tiimin sisäisessä työskentelyssä ja mahdolliset ongelmatilanteet pystytään ratkomaan ajoissa.

Jokaiselle sprintille kuuluu tavallisesti 1–2 lisämäärittelypalaveria (refinement) joissa tuotteen kehitysjonolle lisättyjä tehtäviä käydään tarkemmin läpi kehittäjien kesken. Tuoteomistaja käy asiakkaan kanssa varsinaisen määrittelypalaverin, jossa tehtävät luodaan, ja kehittäjät keskustelevat tehtävistä tiimin sisäisesti lisämäärittelypalaverissa.

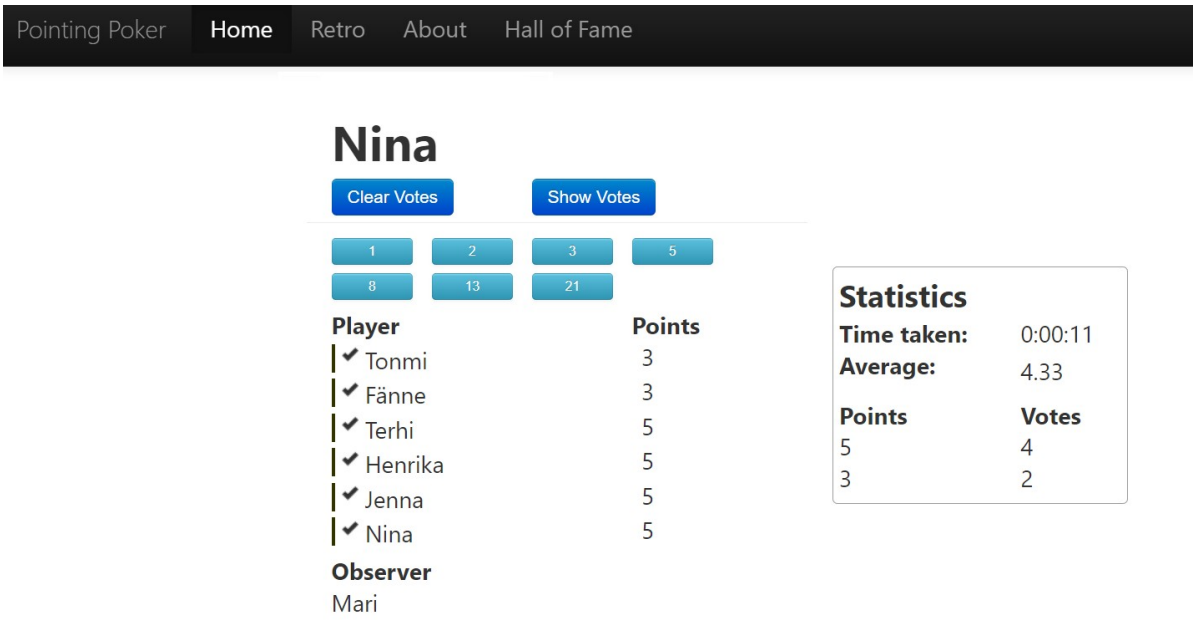
Lisämäärittelypalaverin tarkoituksena on jakaa tietoa tiimin sisällä ja varmistaa, että kaikilla on samanlainen käsitys tehtävän vaatimuksista ja sen toteutuksesta. Lisämäärittelypalaveri on paikka, jossa on mahdollisuus kysellä tarkentavia kysymyksiä ja keskustella mahdollisista toteutustavoista. Aktiivinen osallistuminen lisämäärittelypalavereihin kasvattaa kehittäjän liiketoimintatuntemusta tuotteen sisällä, vaikka keskustelun alla oleva tehtävä menisi jonkun toisen toteutettavaksi.

Tehtävälle asetetaan hyväksymiskriteerit ja sen vaatimaa työmäärää arvioidaan Pointing poker -työkalulla. Pointing poker -työkalun käyttöä on havainnollistettu kuvassa 2, jossa on esitetty kuvakaappaus opinnäytetyön tekemisen aikana käydystä lisämäärittelypalaverista. Työmääräarviossa käytetään fibbonaccin lukujonoa, joka tarjoaa kasvavia mutta epälineaarisia lukuja. Tämä lukujono soveltuu hyvin työmäärän arviointiin, sillä monimutkaisuus kasvaa usein epälineaarisesti. Suuremmat tehtävät sisältävät usein huomattavasti enemmän epävarmuutta ja kompleksisuutta ja työmääräkin on tällöin huomattavasti suurempi.

Fibbonaccin lukujonoon perustuvassa pistearvioinnissa pisteet yksi ja kaksi voivat kuvastaa esimerkiksi melko yksinkertaisen bugin korjaamista ja kolme pistettä voisi olla pienehkön uuden toiminnallisuuden toteuttaminen. Viiden pisteen tehtävä on yleensä jo hieman monimutkaisemman toiminnallisuuden toteuttaminen, kun taas kahdeksan tai sitä enemmän pisteitä kuvastaa jo huomattavasti suurempaa kokonaisuutta. Pieni hajonta äänestyksessä on tavallista ja tehtävälle annetaan story points -pisteet äänestystuloksen keskiarvon mukaan. Mikäli äänestyksessä ilmenee suurta hajontaa annettujen pisteiden välillä, tarkoittaa se sitä, että työn määrittely on epäselvä tai joku ei ymmärrä riittävän hyvin mitä tehtävä pitää sisällään. Kehittäjä voi luulla työn olevan monimutkaisempi toteutus mitä se todella on. Toisaalta, joskus yksi kehittäjistä voi tietää kyseessä olevan kompleksisempi toteutus, miltä se tehtävän kuvauksessa vaikuttaa. Näissä tilanteissa tehtävän toteutuksesta keskustellaan uudestaan ja varmistetaan, että kaikilla on samanlainen käsitys tehtävän vaatimuksista.

Pointing poker on verkossa toimiva, erityisesti ketterään järjestelmäkehitykseen suunniteltu työkalu, jossa tiimille voidaan luoda oma istunto. Tiimin jäsenet liittyvät istuntoon omilta tietokoneiltaan, jonka jälkeen äänestystyökalut ovat käytössä. Lisämäärittelypalaverissa käydyn keskustelun perusteella tiimin jäsenet antavat oman arvionsa tehtävän työmäärästä tietämättä mitä muut aikovat äänestää. Kun jokainen on antanut oman äänensä, näytetään annetut pisteet kaikille. Järjestelmä laskee äänestyksen perusteella keskiarvon, jonka perusteella tiimi päättää tehtävälle asetettavat story points -pisteet.

Kuva 2. Pointing poker -työkalun käyttö lisämäärittelypalaverissa.



Jokaisen sprintin lopussa pidetään demo, jossa kehitystiimi esittelee sidosryhmille sprintin aikana valmistuneet uudet toiminnallisuudet ja bugikorjaukset sekä muut mahdolliset muutokset. Demon aikana sidosryhmiltä saadaan välitön palaute, ja jos jokin osa-alue ei toimi odotetulla tavalla, tarvittavat korjaukset voidaan suunnitella jo seuraavaan sprinttiin. Säännölliset, kahden viikon välein pidettävät demot lisäävät projektin läpinäkyvyyttä ja varmistavat, että sidosryhmien odotukset ja tiimin työ pysyvät linjassa. Tämä jatkuva vuorovaikutus vähentää merkittävästi riskiä siitä, että projektissa syntyisi suurempia väärinymmärryksiä.

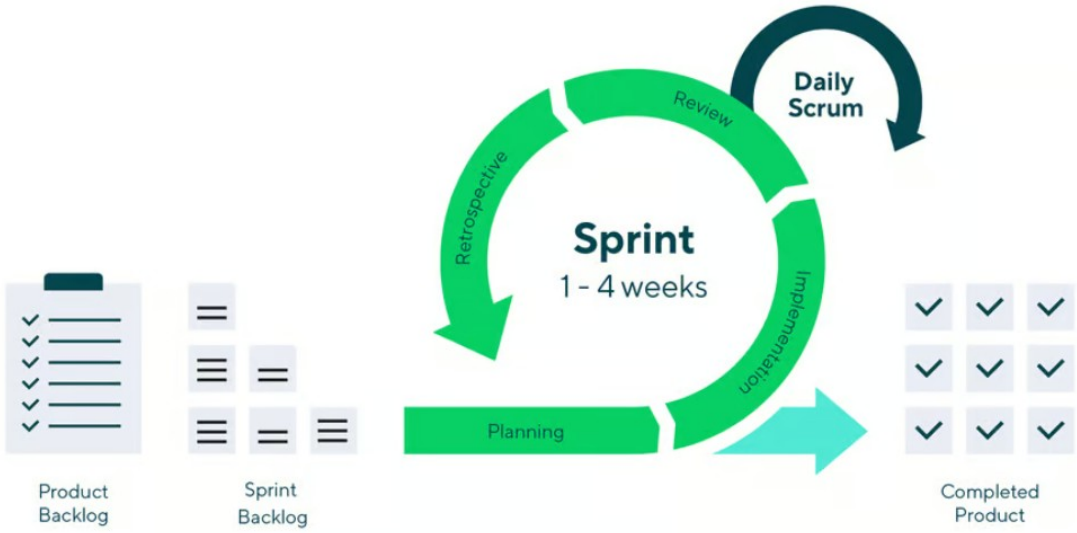
Sprintin loppuun kuuluvat myös sprintin katselmointi (sprint review) ja retrospektiivi (sprint retrospective), joiden aikana arvioidaan sprintin suoritusta ja kehitetään tiimin työskentelytapoja. Sprintin katselmoinnin tarkoituksena on käydä läpi sprintin aikana saavutetut tulokset ja kerätä palautetta sidosryhmiltä. Tavoitteena on tarkistaa, miten hyvin toimitettu työ vastaa sidosryhmien odotuksia ja projektin tavoitteita. Scrum-tiimi ja sidosryhmien edustajat arvioivat, mitä sprintissä saavutettiin, miten tuotteen toimintaympäristössä on tapahtunut muutoksia ja mitä seuraavaksi tulisi tehdä. Tarvittaessa tuotteen kehitysjonoa päivitetään uusien mahdollisuuksien ja prioriteettien mukaiseksi.

Retrospektiivissä keskitytään tiimin sisäisiin prosesseihin ja yhteistyöhön. Retrospektiivin tarkoituksena on auttaa tiimiä jatkuvasti parantamaan itseään, työprosessejaan ja

yhteistyötään. Tavoitteena on kehittää tiimin työskentelyä, parantaa prosessien tehokkuutta ja löytää keinoja työntekoa estävien asioiden poistamiseksi. Retrospektiivi mahdollistaa avoimen keskustelun teknisistä ongelmista, yhteistyöhaasteista, prosessien puutteista tai muista ulkoisista tekijöistä, jotka ovat vaikuttaneet sprintin sujuvuuteen. Retrospektiivissä tarkastellaan myös aikaisemmissa retrospektiiveissä esiin nousseita asioita ja arvioidaan, onko asioiden eteen tehty konkreettisia parannuksia, jotka johtaisivat kehitykseen.

Kuvassa 3 on esitetty kaavion avulla sprinttien toistamaa kiertokulkua. Kuvassa havainnollistetaan, kuinka tuotteen kehitysjonolle listattuja tehtäviä siirretään sprintin kehitysjonolle, ja ne kulkevat suunnittelun kautta toteutukseen ja osaksi tuotetta. Työskentelytavoista, tavoitteista, onnistumisista ja epäonnistumisista käydään jatkuvasti keskustelua sekä tiimin sisäisesti että sidosryhmien kanssa ja tekemistä pyritään parantamaan ja mahdollisia ongelmatilanteita ennakoimaan. Päivittäispalaverit pyörivät omana syklinään prosessin sisällä.

Kuva 3. Sprintin sykli kaavion avulla kuvattuna (Rani, 2023).



3.3 GIT-käytännöt

Jokainen uusi toiminnallisuus tai bugikorjaus toteutetaan uudelle, develop-haarasta johdetulle kehityshaaralle. Develop-haara on kehitystyön päähaara, johon kaikki sprintillä valmistuneet ominaisuudet ja bugikorjaukset yhdistetään sprintin lopussa. Main-haara on versiohallinnan ydin ja siellä sijaitsee viimeisin vakaa tuotantoversio tuotteesta.

Kehitystyön jakamisessa useille eri haaroille on paljon etuja. Kehityshaarat (feature branch) mahdollistavat uusien ominaisuuksien ja toiminnallisuuksien kehittämisen ja testaamisen eristyksissä tuotantoversiosta, sillä muutokset tehdään tässä vaiheessa lokaalisti kehittäjän omalla tietokoneella. Kehittäjän tehdessä pieniä commiteja työn edetessä pystytään kehityksen kulkua seuraamaan ja virhetilanteissa voidaan palata edelliseen toimivaan commitiin. Kehityshaarat mahdollistavat useiden kehittäjien työskentelyn samanaikaisesti.

Erilaisten haarojen käyttöä havainnollistetaan kuvassa 4. Vihreät osat kuvastavat kehityshaaroja, joissa kehittäjä tekee päivittäistä työtään. Isossa kehitystiimissä kehityshaaroja voi olla samanaikaisesti kymmeniä, mutta kuvassa 4 on esitetty kaksi erillistä kehityshaaraa. Kehitystyön tavallinen kulku git-komentoja käyttäen on seuraavanlainen:

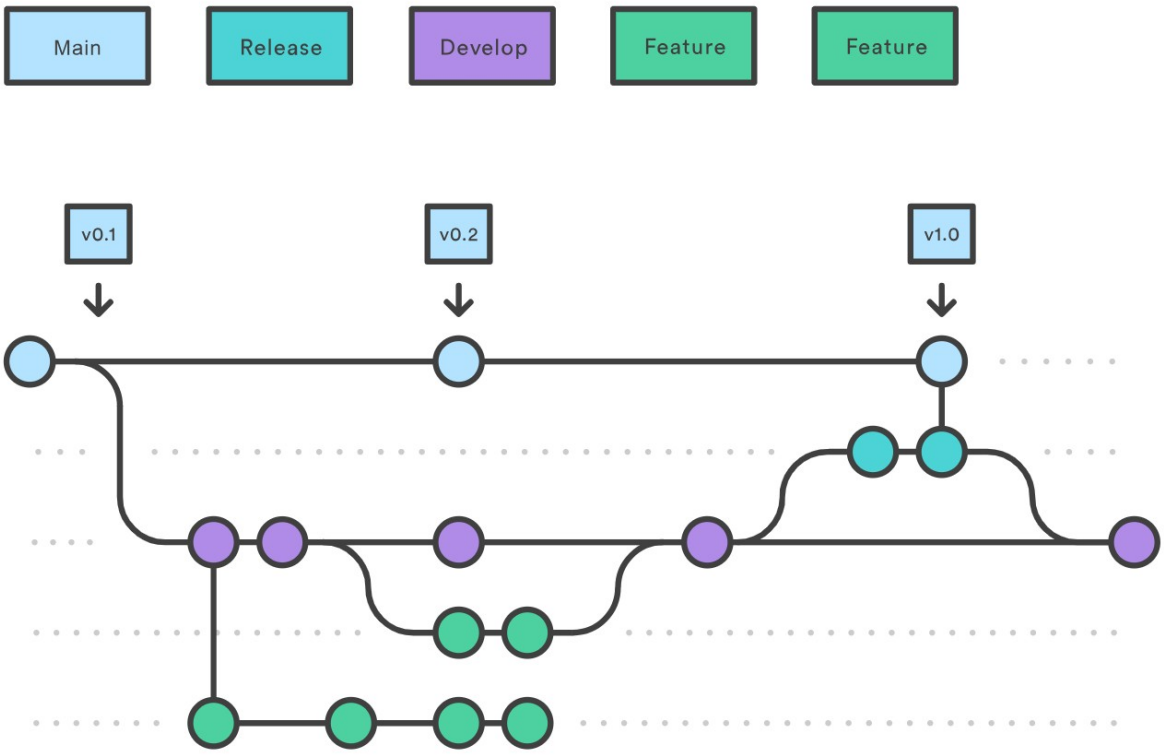
```
C:\Users\nina.tulilahti\Documents\PowerLending\(feature/
Manuaalisen-kirjanpitonviennin-lisaaminen -> origin)

$ git status
$ git add <tiedosto>
$ git commit -m "kuvaus"
$ git pull
$ git push
```

Git status -komennolla voidaan tarkistaa työstettävän haaran tila. Komento näyttää ensimmäisenä haaran nimen, ja kehittäjä voi näin varmistua olevansa oikealla haaralla. Koodiesimerkissä haara näytetään ennen varsinaisia komentoja vihreällä. Git status -komento antaa yleiskuvan siitä, mitkä tiedostot ovat muuttuneet, mitä on mahdollisesti jo lisätty stagelle ja mitkä muutokset stagelta puuttuvat.

Komennolla git add <tiedoston nimi> lisätään kulmasulkeissa määritelty tiedosto stage-tilaan. Tiedostoja voidaan lisätä kerralla useampia erottelemalla lisättävät tiedostot toisistaan välilyönnillä, esimerkiksi komennolla: git add tiedosto1 tiedosto2 tiedosto3. Piste ei kuulu komentoon. Vaihtoehtoisesti kaikki muokatut tiedostot voidaan lisätä yksinkertaisesti komennolla git add . (piste), jossa pistemerkki kuuluu komentoon. Komennolla git commit lisätään tehdyille muutoksille kuvaus. Kuvausteksti tulee lainausmerkkien sisälle, ja se näkyy Azure ympäristössä commitin yhteydessä.

Ennen omien muutosten puskemista on hyvä käytäntö tarkistaa, onko kehityshaaralle tullut muutoksia toisen kehittäjän toimesta. Tämä tehdään komennolla git pull. Kun kaikki on kunnossa, voidaan uusimmat muutokset lopuksi lähettää versiohallintaan komennolla git push.

Kuva 4. Kehityshaarojen käyttö Evitec Power Lending -järjestelmäkehitystiimissä (Atlassian, n.d.).



Kehitystyön lomassa on hyvä tarkistaa säännöllisesti develop-haaran muutokset ja yhdistää ne osaksi omaa kehityshaaraa. Tällä tavoin haara pysyy ajan tasalla ja synkronoituna muiden kehittäjien tekemien muutosten kanssa. Muutosten yhdistäminen tapahtuu komentorivillä seuraavalla tavalla:

```
C:\Users\nina.tulilahti\Documents\PowerLending\(feature/
Manuaalisen-kirjanpitonviennin-lisaaminen -> origin)

$ git checkout develop

C:\Users\nina.tulilahti\Documents\PowerLending\(develop -> origin)
```

```
$ git pull
$ git checkout feature/Manuaalisen-kirjanpitoviennin-lisääminen



C:\Users\nina.tulilahti\Documents\PowerLending\(feature/
Manuaalisen-kirjanpitonviennin-lisaaminen -> origin)

$ git merge develop
$ git add .
$ git commit -m ”merge feature branch into develop”
$ git push
```

Haarojen välillä liikkuminen tapahtuu git checkout -komennolla, jota seuraa sen haaran nimi, jolle halutaan siirtyä. Omalta kehityshaaralta siirrytään ensin develop-haaralle ja vedetään sieltä uusimmat muutokset git pull -komennolla. Tämän jälkeen siirrytään taas omalle kehityshaaralle. Komennolla git merge develop yhdistetään develop-haaran muutokset osaksi omaa kehityshaaraa. Lopuksi uusi versio haarasta voidaan tallentaa versiohallintaan. Kehityshaara sisältää nyt kaikki develop-haaran muutokset.

3.4 Katselmointi

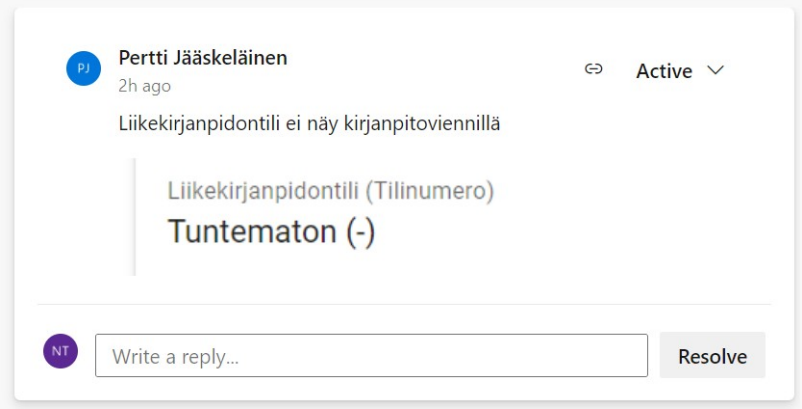
Azure DevOps -ympäristön Repos-näkymästä löytyy myös välilehti Pull request (PR). Kun kehittäjä saa tehtävänsä valmiiksi, tehdään siitä pull request eli ehdotus koodimuutoksesta. Evitec Solutionsin sisäisissä ohjeissa määritellään, että mitään muutoksia ei koskaan yhdistetä suoraan develop-haaralle, vaan kahden muun tiimin jäsenen on ensin hyväksyttävä muutokset. Kehittäjä kirjoittaa PR:lle kuvauksen siitä, mitä toteutuksessa on tarkalleen tehty, ja sen lisäksi alkuperäinen tehtävä linkataan PR:lle. PR:lle kirjoitetaan myös selkeät testausohjeet katselmoijia varten. Näin katselmoija saa hyvän käsityksen testattavasta ominaisuudesta. (Evitec Solutions, henkilökohtainen tiedonanto, n.d.)

Katselmoijan tehtävä on testata kehitetty ominaisuus tai bugikorjaus läpikotaisin. Katselmoija siirtyy sille haaralle, jonne toteutus on tehty, ja testaa toimivuutta käyttöliittymällä testausohjeiden mukaisesti. Ohjeiden mukaisen testauksen lisäksi hyvässä katselmoinnissa pyritään myös rikkomaan toteutus, eli testataan, hajoaako sovellus esimerkiksi syöttämällä lomakkeelle vääriä arvoja tai miten sovellus reagoi virheelliseen dataan. On myös hyvä testata, että virhetilanteissa käyttäjälle annetaan selkeät virheilmoitukset eikä sovelluksen käyttöliittymä hajoa.

Käyttöliittymätestauksen lisäksi katselmoijan tulee tehdä koodikatselmointi, jossa tarkistetaan, että koodi on luettavaa ja muuttujien nimet ovat kuvaavia. Koodin tulee olla Evitec Solutionsin sisäisten koodikäytäntöjen mukaista ja muut mahdolliset tiedostot, esimerkiksi SQL-scriptit, on nimetty voimassa olevien käytänteiden mukaisesti (Evitec Solutions, henkilökohtainen tiedonanto, n.d.).

Mikäli katselmoija löytää toteutuksesta bugin tai muun virheen, hän jättää PR:lle kommentin, jossa kuvaa havaitsemaansa ongelmaa. Kuvassa 5 on esitetty esimerkki opinnäytetyön tekemisen aikana PR-vaiheessa olevalta tehtävältä löytyneestä bugista, josta katselmoija jätti kommentin. Myös mahdollisista parannusehdotuksista voidaan jättää kommentti. Kehittäjä käy kommentit läpi ja korjaa havaitut ongelmat, jonka jälkeen katselmoija testaa toteutuksen uudelleen.

Kuva 5. PR-katselmoijan jättämä kommentti löydetystä bugista.



Kun kaikki virheet on korjattu, katselmoija voi päättää katselmoinnin PR:ltä löytyvällä Approve-painikkeella. PR-katselmoinnilla varmistetaan koodin laatu ja yhteensopivuus aikaisempaan toteutukseen, ja samalla se edistää tiimityöskentelyä kehittäjien välillä. Kun PR on saanut hyväksynnän kahdelta katselmoijalta, voidaan se liittää develop-haaraan.

4 Työssä käytettävät tekniikat ja työkalut

.NET Framework, React, Git, SQL Server ja Azure muodostavat tämän opinnäytetyön teknisen osuuden ytimen. Git-työskentelyssä on käytetty sekä komentoriviä että Fork-sovellusta. Koodieditoreina on käytetty Visual Studiota sekä Visual Studio Codea. Luvussa 3 on kuvattu tarkemmin Azure-työskentelyä ja komentorivityöskentelyä, joten niitä ei tässä luvussa käsitellä syvällisemmin.

4.1 Palvelinpuolen kehitys

ASP.NET Core on Microsoftin kehittämä avoimen lähdekoodin viitekehys, ja se vastaa tässä työssä palvelinpuolen toiminnallisuuksista. ASP.NET Core yhdistää ASP.NET Web API- ja ASP.NET MVC -teknologiat yhdeksi viitekehykseksi, ja yhdessä tämä kokonaisuus on osa laajempaa .NET Frameworkia. ASP.NET Core tarjoaa monipuoliset työkalut web-sovellusten kehittämiseen, ja se sopii erityisen hyvin pilvipohjaisten sovellusten kehittämiseen. ASP tulee sanoista Active Server Pages. (Sharma, 2020)

ASP.NET julkaistiin vuonna 2002 Classic ASP:n seuraajana. ASP.NET Framework esitteli monia uusia ominaisuuksia web-kehityksessä, mukaan lukien "drag and drop" -komponentit. ASP.NETiä käytettiin yhdessä ADO.NETin kanssa luomaan datalähtöisiä verkkosovelluksia. Vuonna 2009 Microsoft julkaisi ASP.NET MVC -teknologian, joka oli merkittävä askel kohti paremman arkkitehtuurin tarjoamista verkkosovelluksille. MVC-lyhenne tulee sanoista Model, View ja Controller, ja se kuvastaa sovelluksen jakamista erilaisiin osiin. ASP.NET Core 1.0 julkaistiin vuonna 2016 yhdessä Entity Framework Coren, Identity Coren, Web API:n ja monen muun teknologian kanssa. Tämä oli osa suurempaa siirtymistä kevyempiin arkkitehtuurimalleihin. (.NET Trainer & Instructor, 2023) ASP.NET on sittemmin kehittynyt nopeaa tahtia. Marraskuussa 2024 julkaistiin versio .NET 9 (Microsoft, 2024d).

ASP.NET Core on suosittu teknologia, kun tehdään suuria ja skaalautuvia verkkosovelluksia, ja modulaarisuus on yksi sen keskeisimpiä etuja. ASP.NET Core pohjautuu uudelleen käytettäviin NuGet-paketteihin, jotka ovat ohjelmaan asennettavia koodikirjastoja (Sharma, 2020). Entity Framework Core on esimerkki yhdestä NuGet-paketista, jonka päälle myös Evitec Power Lending on rakennettu. EF Core tarjoaa mm. monipuolisen tuen erilaisille tietokannoille, se tukee LINQ-kyselyitä ja se integroituu helposti dependency injection -mallin kanssa. (Microsoft, 2021) EF Coren suurin etu on siinä, että sen tuottamasta C#-koodista voidaan tehdä suoraan SQL-kyselyitä. ASP.NET Core tukee lisäksi API-rajapintoja alustariippumattomasti, jolloin ASP.NET Core-sovellukset toimivat käyttöjärjestelmästä riippumatta (Sharma, 2020).

ASP.NET Core tukee asynkronista ohjelmointimallia, mikä on tärkeä ominaisuus silloin kun käsitellään tietokantoja tai lähetetään dataa sovelluksen sisällä. Funktio määritellään asynkroniseksi avainsanalla async, ja tämä avainsana sallii await -avainsanan käytön funktion rungossa. Await-avainsanalla määritellään, että funktion toiminnallisuus keskeytetään siksi aikaa, että await-avainsanalla kutsuttava metodi on valmistunut.

(Microsoft, 2023b) Näin voidaan varmistua esimerkiksi siitä, että tietokannasta haettava data on funktion käytössä ennen kuin sitä aletaan prosessoimaan.

Kehitystyössä editorina käytetään Visual Studiota. Visual Studio on myös Microsoftin kehittämä ja siksi siinä on erittäin hyvä tuki .NET-kehitykselle. Visual Studio on kokonaisvaltainen työkalu, jolla voidaan koodin kirjoittamisen lisäksi rakentaa projekti, etsiä virheitä ja testata koodia. Visual Studio tukee versiohallintaa ja git-prosessi on mahdollista tehdä myös suoraan Visual Studiossa. Live Share -ominaisuuden avulla kehitystyötä voidaan tehdä yhteistyössä toisten kehittäjien kanssa. (Microsoft, 2024e)

.NET-kehityksen lisäksi Visual Studio tukee useita eri ohjelmointikieliä ja projektityyppejä. Sillä voidaan kehittää web-sovellusten lisäksi esimerkiksi 3D-videopelejä, IoT-sovelluksia, työpöytäsovelluksia ja pilvipohjaisia ratkaisuja. Asennustyökalulla valitaan, minkä tyyppisille projekteille ja ohjelmointikielille tuki halutaan asentaa, ja asennuksia voidaan lisätä ja poistaa myöhemmin. (Microsoft, 2024e)

Palvelinpuolella käytettävistä tekniikoista .NET Frameworkin rinnalla Java on toinen suosittu teknologia, joka soveltuu myös hyvin isojen ja skaalautuvien sovellusten luomiseen. Sun Microsystems, joka nykyisin on Oraclen tytäryhtiö, julkaisi Javan vuonna 1995. Java pyörii Java Virtual Machinen (JVM) päällä ja on sitä kautta alustariippumaton kehitysympäristö, mutta vaatii siitä syystä myös enemmän suorituskykyä. (Sharma, n.d.)

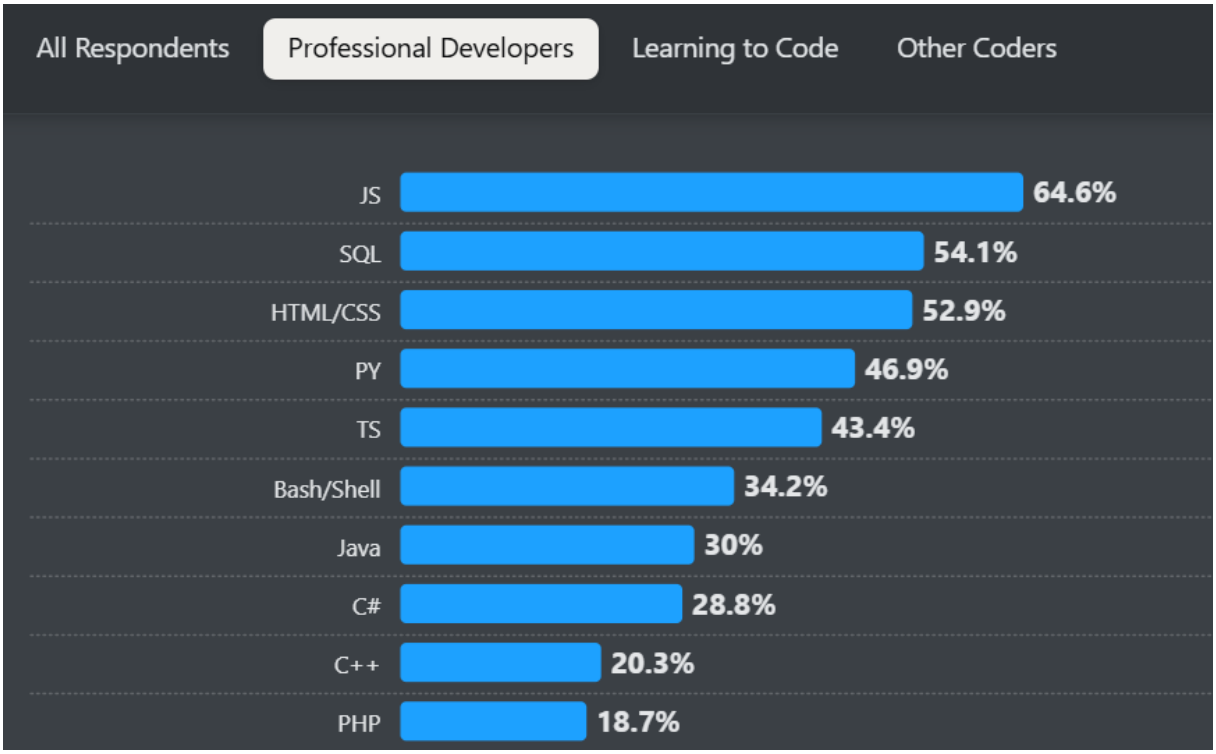
Sekä Java että .NET tukevat olio-ohjelmointimallia, ja .NET Coren myötä molemmat ovat alustariippumattomia kehitysympäristöjä. .NET integroituu parhaiten muiden Microsoftin kehittämien teknologioiden kanssa, kun taas Javalle on tarjolla paljon kolmansien osapuolien tuottamia ratkaisuja. Esimerkiksi Javalle paremmin sopivia editoreita ovat JDeveloper, IntelliJ IDEA tai Eclipse, kun .NET kehityksessä käytetään Visual Studiota. Pilvipalveluiden osalta Amazon Web Services (AWS) ja Google Cloud Platform (GCP) ovat suosittuja Java-pohjaisten ohjelmistojen ylläpitämisessä. (Sharma, n.d.)

Javassa ja .NETissä on paljon yhtäläisyyksiä ja molemmat ovat suhteellisen vakaita kehitysympäristöjä. Molemmilla on kattava dokumentaatio ja oppimiskäyrää kuvataan suhteellisen helpoksi. Toisaalta Javalle on tarjolla ehkä hieman kattavammin itseopiskelumateriaalia. Sekä Javalle että .NETille on olemassa laajat, ulkoiset kirjastot jotka tarjoavat valmiita ratkaisuja useisiin ohjelmistokehityksen tarpeisiin. Kehitysympäristön valinta voi olla haasteellista näiden kahden teknologian välillä, ja valinta

saattaakin joskus perustua puhtaasi siihen, kummasta teknologiasta sattuu löytymään osaamista.

Stack Overflow kerää vuosittain dataa IT-alan käytetyimmistä tekniikoista. Kuvassa 6 on esitetty kaavio Stack Overflown vuoden 2024 Developer Survey -kyselyn tuloksista. Kysely kattaa kaikki ohjelmointi-, scriptaus- ja merkkikielet, mutta kuva on rajattu kärjessä oleviin tekniikoihin. Lisäksi kuvan 6 kaavio esittää vain ohjelmistokehitystä ammatikseen tekevien vastaajien tulokset. Kuvasta 6 nähdään, että Java ja C# ovat lähes yhtä suosittuja ohjelmointikieliä. Javalla on 1,2 % enemmän käyttäjiä ohjelmistokehityksen parissa työskentelevien ammattilaisten keskuudessa, jotka ovat vastanneet kyselyyn. C# on .NET Framework -työskentelyssä käytetty ohjelmointikieli.

Kuva 6. Vuoden 2024 suosituimmat ohjelmointi-, scriptaus- ja merkkikielet Developer Survey -kyselyssä (Stack Overflow, 2024).



4.2 Asiakaspuolen kehitys

Asiakaspuolen kehityksessä käytetään Reactia ja TypeScriptiä. React on Facebookin kehittämä, avoimen lähdekoodin JavaScript-kirjasto, jota käytetään käyttöliittymien luomiseen. Reactin ydinajatus on luoda nopeammin toimivia käyttöliittymiä, ja tämä

toteutetaan muodostamalla sovellus käyttäen itsenäisesti toimivia komponentteja. Kun data liikkuu sovelluksessa, riittää että kyseisestä datasta vastaava komponentti renderöityy uudelleen koko verkkosivun sijaan. Tämä on mahdollista virtuaalisen DOM (Document Object Model) rakenteen ansiosta. (Camus, 2022)

TypeScript on tyypitettyä JavaScriptia: se on Microsoftin luoma laajennus JavaScript-kieleen. Tämä tarkoittaa sitä, että TypeScript pitää sisällään kaikki JavaScriptin ominaisuudet sekä TypeScriptin mukanaan tuomat laajennukset. Olemassa oleva JavaScript-koodi on siis myös toimivaa TypeScript-koodia. (Torppa, ym., 2024)

Seuraavassa koodiesimerkissä on esitetty funktio helloUser, joka ottaa kaksi argumenttia. Toinen on tyypiltään merkkijono (string), ja toinen on numero (number). Tyypitys tehdään TypeScriptissä syntaksilla muuttujan nimi, kaksoispiste, tyyppi.

```
const helloUser = (name: string, age: number): string => {
    return `Welcome to my web site ${name}! You are ${age} years old!`;
  };

  const userName = "Maija Mehiläinen";
  const age = 22;

  console.log(helloUser(userName, age));
```

TypeScriptistä on hyötyä erityisesti laajoissa sovelluksissa, joissa muuttujia ja funktioita on paljon ja samaa koodia käytetään useissa paikoissa. Tyypityksen ansiosta kehittäjä voi varmistua muun muassa siitä, minkä tyyppistä dataa funktio odottaa saavansa, ja koodieditori herjaa saman tien, jos sille yritetään syöttää vääräntyyppistä dataa. Esimerkiksi Visual Studio Code tarjoaa IntelliSense-ominaisuuden, joka pystyy melko hyvin osoittamaan missä virhe tapahtuu. JavaScriptin tuottamat virheviestit konsolissa ovat toisinaan vaikeasti tulkittavia, jolloin TypeScriptin käyttö säästää kehittäjän yhdeltä päänvaivalta.

React tukee myös TypeScriptia. React-komponentti palauttaa elementin, joka muistuttaa hieman HTML-syntaksia, mutta kyseessä on XML-tyylinen tapa kirjoittaa JavaScriptia tai TypeScriptia, ja kyseessä onkin JSX (JavaScript XML) tai TSX (TypeScript XML) -syntaksi valitun kielen mukaan.

React-komponenttien voidaan ajatella olevan sovelluksen rakennuspalikoita. React-komponentit ovat JavaScript-pohjaisia luokkakomponentteja tai funktionaalisia

komponentteja. Funktionaaliset komponentit ovat nykyaikaisempi tapa kirjoittaa React-sovelluksia. React-komponentit ottavat propseina vastaan tarvitsemiansa tietoja, ja palauttavat React-elementin, joka kuvaa, mitä ruudulla pitäisi näkyä. React-sovellus on yksi suuri kokonaisuus komponentteja komponenttien sisällä. Tämä kehitystapa eroaa aikaisemmasta tyylistä luoda verkkosivuja, ja modulaariseen ajatteluun siirtyminen saattaa vaatia hieman pureskelua. React-sovelluksessa kehittäjä luo sivujen sijaan komponentteja, ja jokaisella komponentilla on yksi tietty tarkoitus. Samaa komponenttia voidaan uudelleen käyttää lähettämällä sille propsien avulla eri tietoja, joita se renderöi näytettäväksi. (Camus, 2022)

Propsit (properties) ovat tapa lähettää tietoa komponentilta toiselle. Propsien avulla tietoa lähetetään komponenttihierarkiassa ulommaisemmalta komponentilta sisemmälle komponentille. Propseja voisi verrata funktion argumentteihin. Usein ilmenee tarvetta liikutella dataa React-sovelluksen sisällä moneen suuntaan, ja tätä varten React-sovelluksella on käytössään myös tilat, tilanhallintakirjastot sekä hookit joilla on omat käyttötarkoituksensa. (Pattern, n.d.) Seuraavassa koodiesimerkissä on havainnollistettu propsien käyttöä React-koodissa:

```
const SayHello = ({ name, age }: { name: string; age: number }) => {
    console.log(name, age)
    return (
      <div>
        <p>
          Hello {name}, you are {age} years old
        </p>
      </div>
    )
  }
```

Komponentti SayHallo ottaa vastaan tietoa propseina. Propsit tulostetaan konsoliin, jolloin kehittäjä voi nähdä mitä tietoa propsien mukana tarkalleen ottaen tulee. Komponentti palauttaa tervehdyksen, jossa käyttäjää tervehditään nimellä ja kerrotaan hänen ikänsä.

```
const App = () => {
    const nimi: string = 'Pekka'
    const ika: number = 10

    return (
      <div>
        <h1>Tervehdys</h1>
        <SayHello name="Maija" age={26} />
```

```
        <SayHello name={nimi} age={ika} />
      </div>
    )
  }
```

Komponentti App määrittää muuttujat nimi ja ikä. Komponentti palauttaa ylempänä esitellyn SayHello-komponentin kahteen kertaan eri tiedoilla. Ensimmäisellä kerralla komponentille lähetetään kovakoodatut arvot name=”Maija” ja age=26. Toisella kerralla komponentille lähetetään ylempänä määritellyt muuttujat. Kun komponentti renderöityy App-komponentin sisällä, näkyy ruudulla otsikon alapuolella ensin tervehdys ”Hei Maija, sinä olet 26 vuotta vanha.” ja sitä seuraa tervehdys ”Hei Pekka, sinä olet 10 vuotta vanha.”. Koodiesimerkit havainnollistavat hyvin myös TSX-syntaksia. <h1>, ja <p> ovat HTML-kielestä tuttuja elementtejä.

Tilalla tarkoitetaan sellaista objektin sisältämää tietoa, joka saattaa muuttua sovelluksen käytön aikana. React-sovellusta luodessa on hyvä pyrkiä suunnittelemaan sovellus mahdollisimman tilattomaksi suosien mieluummin propseja ja tapahtumia. React-tilat tekevät sovelluksesta helposti monimutkaisen ja vaikeasti hallittavan, minkä takia on suositeltavaa välttää niiden käyttöä aina kun sama asia voidaan toteuttaa jollakin muulla tavalla. Näin koodi pysyy myös helpommin testattavana ja ymmärrettävänä. Silloin kun tiloja tarvitaan, on siihen hyvä käyttää tilanhallintajärjestelmää kuten Reduxia tai MobX:ää. (Pattern, n.d.)

Asiakaspuolen kehityksessä koodieditorina käytetään Visual Studio Codea. Se on Microsoftin kehittämä koodieditori, joka tukee useita eri ohjelmointikieliä. Visual Studioon verrattuna se on huomattavasti kevyempi editori, mutta silti sillä on monia samoja ominaisuuksia. Visual Studio Codesta löytyy oma versionhallinta, virheiden etsintä -työkalu ja lisäosien avulla ohjelmaan voidaan asentaa monia hyödyllisiä lisäominaisuuksia kuten esimerkiksi Live Share tai Prettier, joka huolehtii koodin oikeaoppisesta sisentämisestä.

Reactin rinnalla toinen suosittu asiakaspuolen teknologia on Vue.js. Vuen on kehittänyt Evan You, tavoitteenaan luoda viitekehys, joka yhdistää parhaat puolet Angularista, Reactista ja Emberistä. Nämä ovat kaikki JavaScript-viitekehyksiä. Vue ratkaisee monia samoja ongelmia mitä React, mutta tekee sen hieman eri tavalla. (Buddha, 2024)

Myös Vue perustuu uudelleenkäytettäviin komponentteihin. Vue-syntaksi koostuu HTML-elementtejä laajentavasta tavasta kirjoittaa komponentteja (Vue.js, n.d.), kun React-

komponentit käyttävät JSX-syntaksia. Lisäksi Vue hyödyntää Reactin esittelemää virtuaalista DOM -rakennetta komponenttien renderöimiseen. Molemmat tekniikoista ovat myös hyvin joustavia ja skaalautuvia, ja sopivat monimutkaisten web-sovellusten kehittämiseen. (Emadamerho-Atori, 2023)

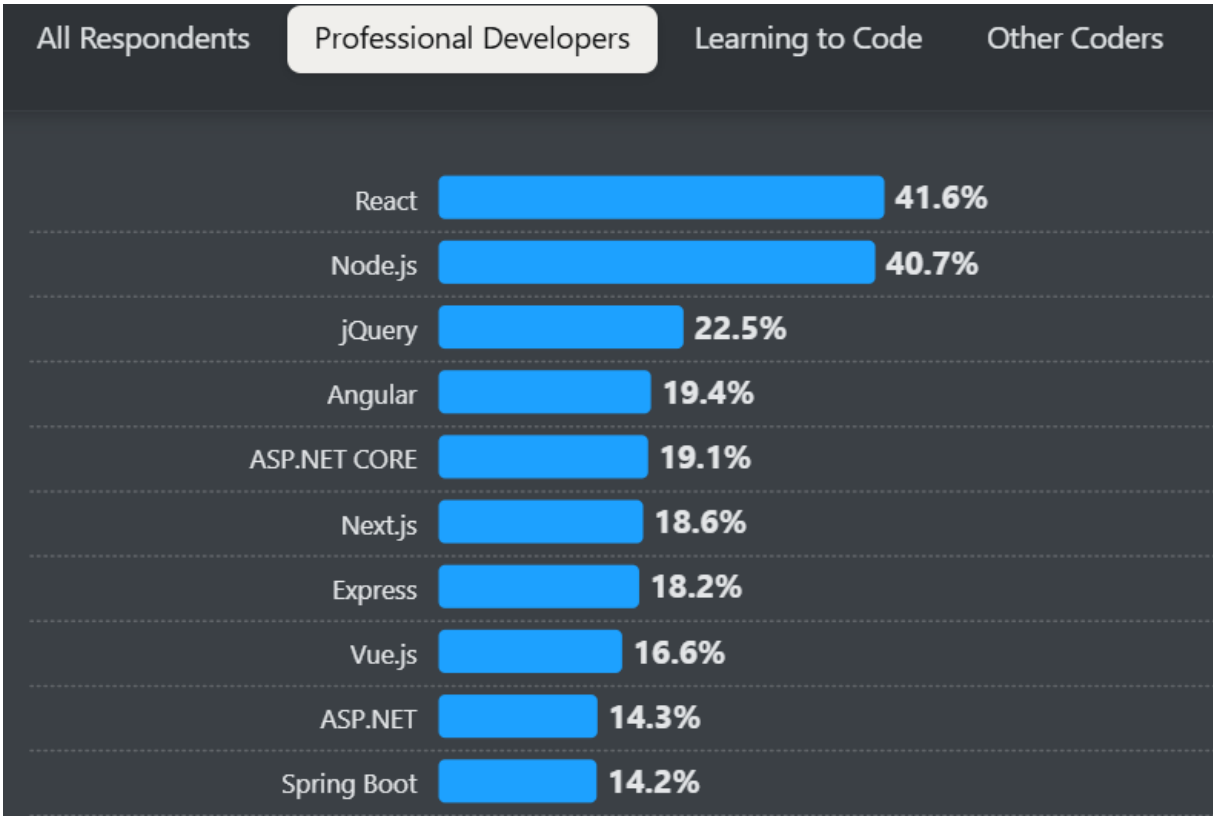
Vuen HTML-pohjainen syntaksi on saanut inspiraatiota Angularista. Vue-komponentissa rakenne on jaettu selkeästi verkkosivulla näkyvään osaan ja toimintalogiikan sisältävään osaan. (Emadamerho-Atori, 2023) Seuraavassa Vue-koodiesimerkissä luodaan painike, joka laskee klikkauksia. Koodin JavaScript-osio löytyy <script>-elementin sisältä ja HTML-osio <template>-elementin sisältä. Lisäksi tässä esimerkissä tyylit on sijoitettu <style>-elementin sisälle. Tämän tyyppinen syntaksi on hieman lähempänä perinteistä tapaa luoda web-sovelluksia, kuin Reactin JSX-syntaksi. Osa kehittäjistä kokee logiikan ja HTML-elementtien erottamisen selkeämpänä, minkä takia Reactilla ja Vuella on omat kannattajansa. (Emadamerho-Atori, 2023)

```
<script setup>
import { ref } from "vue";
const clickCounter = ref(0);
</script>

<template>
  <div>
    <h1>Klikkilaskuri</h1>
    <button @click="clickCounter++">Clicks: {{ clickCounter }}</button>
  </div>
</template>

<style scoped>
button {
  font-weight: bold;
  padding: 10px 20px;
  border: none;
  background-color: #42b983;
}
</style>
```

Stack Overflown vuoden 2024 Developer Survey -kyselystä käy ilmi, että ammattimaisiksi ohjelmistokehittäjiksi itsensä luokittelevien vastaajien keskuudessa React on suosituin web-kehitykseen käytetty tekniikka. Kyselyn tulokset on esitetty kuvassa 7. Kyselyn mukaan Reactia käyttää 41.6 % vastaajista. Myös vanhempia tekniikoita, kuten jQuerya ja Angularia käytetään edelleen. Vue on saanut 16.6 % annetuista äänistä. Äänet ovat

jakautuneet samassa suhteessa Reactin ja Vuen välillä jo muutaman vuoden ajan (Emadamerho-Atori, 2023).

Kuva 7. Vuoden 2024 suosituimmat web-tekniikat Developer Survey -kyselyssä (Stack Overflow, 2024).



4.3 SQL Server

SQL (Structured Query Language) tarkoittaa kyselykieltä, jolla tietokannasta haetaan tietoa. SQL Server on Microsoftin kehittämä tietokannan hallintajärjestelmä, joka usein lyhennetään myös muotoon SQL. (Actian, n.d.). Tietokannan hallintajärjestelmällä tarkoitetaan sellaista ohjelmistoa, jolla kehittäjät ovat vuorovaikutuksessa varsinaisen tietokannan kanssa. Tietokannan hallintajärjestelmä toimii palvelimena, jonne voidaan tehdä kyselyitä, ja se sisältää tätä varten kaikki tarvittavat palvelut ja komponentit. (Rouse, 2024) SQL Serveriin voidaan tehdä myös suoraan API-kutsuja (Actian, n.d.).

SQL Server integroituu hyvin muiden Microsoftin tuotteiden, kuten Azure-pilvipalveluiden ja .NET-kehitysympäristön kanssa. SQL Server tarjoaa tehokkaan ja skaalautuvan alustan tietojen tallennukseen, hallintaan ja hakemiseen. SQL Server tukee relaatiotietomallia,

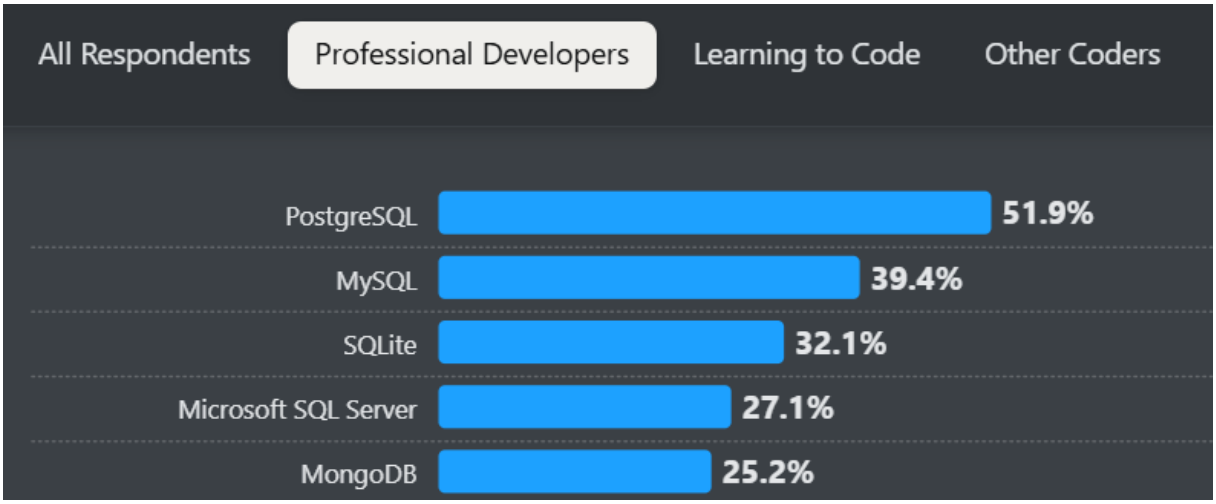
jossa tiedot tallennetaan taulukoihin, ja tietoja voidaan hakea ja muokata SQL-kielen avulla. (Microsoft, 2024a)

SQL Server Management Studio (SSMS) tarjoaa graafisen käyttöliittymän SQL Server -palvelun parissa työskentelyyn. SSMS mahdollistaa tietokantojen luomisen, muokkaamisen ja poistamisen graafisen käyttöliittymän kautta, jolloin kehittäjän ei tarvitse tehdä SQL-kyselyitä komentorivin kautta. (Microsoft, 2024b)

Toinen suosittu tietokantojen hallintajärjestelmä on MySQL. Se on Oraclen omistama, vastaava palvelu, jolla on myös oma graafinen käyttöliittymä MySQL Workbench. Suurin ero SQL Serverin ja MySQL:n kanssa on siinä, että MySQL on avoimen lähdekoodin ohjelmisto ja sitä voi käyttää täysin ilmaiseksi. SQL Server on Microsoftin tuote ja soveltuu parhaiten Windows-ympäristöön ja osaksi Microsoftin ohjelmistokokonaisuutta. (Ravikiran, 2024)

Kuvassa 8 esitetyn Stack Ovwerflown Developer Survey -kyselyn mukaan MySQL on hieman suositumpi, mitä SQL Server. MySQL on saanut 39.4 % annetuista äänistä, kun SQL Server on saanut 27.1 %. Koska SQL Server on käytännössä sidottu Microsoftin tuotteisiin, rajaa se osan käyttäjistä pois siinä missä MySQL on sopiva useammille viitekehyksille.

Kuva 8. Vuoden 2024 suosituimmat tietokantatekniikat Developer Survey -kyselyssä (Stack Overflow, 2024).



4.4 Fork

Fork on graafinen käyttöliittymä git-komentojen suorittamiseen. Forkin avulla voidaan suorittaa kaikki gitin peruskomennot, jonka lisäksi ohjelma tarjoaa myös useita kehittyneempiä toimintoja. Fork on kokonaisvaltainen git-ohjelmisto, joka on luotu helpottamaan päivittäistä git-työskentelyä. (Fork, n.d.) Kuvassa 9 on esitetty yleiskatsaus Forkin käyttöliittymästä. Siinä on näkyvillä avoinna olevat repositoriot, uusimmat commitit listattuna, projektiin liittyvät kehityshaarat ja yksityiskohtaisempia tietoja valitusta commitista.

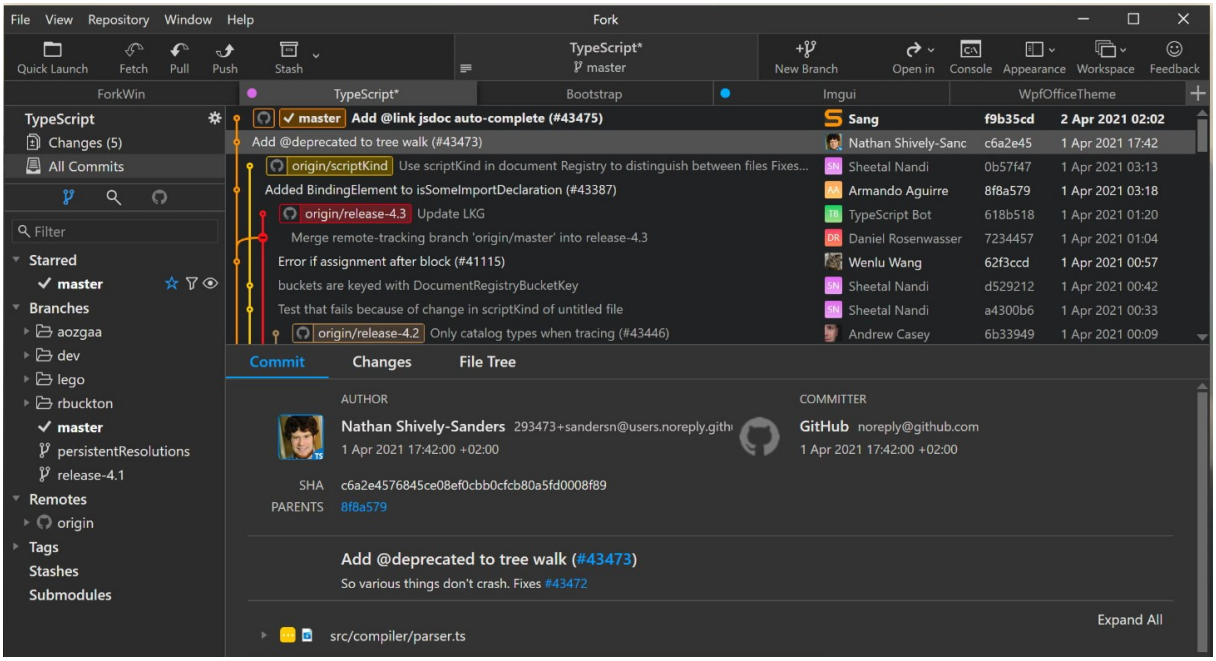
Yleisnäkymässä avoinna olevia tietoja on korostettu vaalean harmaalla pohjavärillä. Avoinna on TypeScript-repositorio, All Commits -näkymä ja Nathan Shively-Sandersin commit, jonka yksityiskohtaisempia tietoja voi tarkastella kuvan 6 näkymässä keskellä alhaalla.

Forkin parissa työskentely alkaa siitä, että avataan ensin repositorio jonka parissa kehitystyötä tehdään. Ohjelmassa voi olla auki samanaikaisesti useita repositorioita. Kuvassa 9 avoinna olevat repositoriot näkyvät välilehdillä, joita ovat ForkWin, TypeScript, Boorstrap, Imgui ja WpfOfficeTheme. Repositorion nimen perässä * (tähti) -merkki kertoo siitä, että repositoriossa on muutoksia, joita ei ole vielä lisätty versiohallintaan. Uusi repositorio voidaan avata ohjelmaan ylärivin valikosta File → Open repository.

Repositorio-välilehtien perässä olevasta pluspainikkeesta aukeaa Repository Manager, joka näyttää yleiskatsauksen avoinna olevista repositorioista. Repository Manager -näkymä sisältää erilaista statistiikkaa siitä, ketkä ovat kehittäneet repositoriota, mitkä ovat viimeisimmät aktiiviset kehityshaarat, viimesimmät commitit sekä repositorion readme-tiedoston.

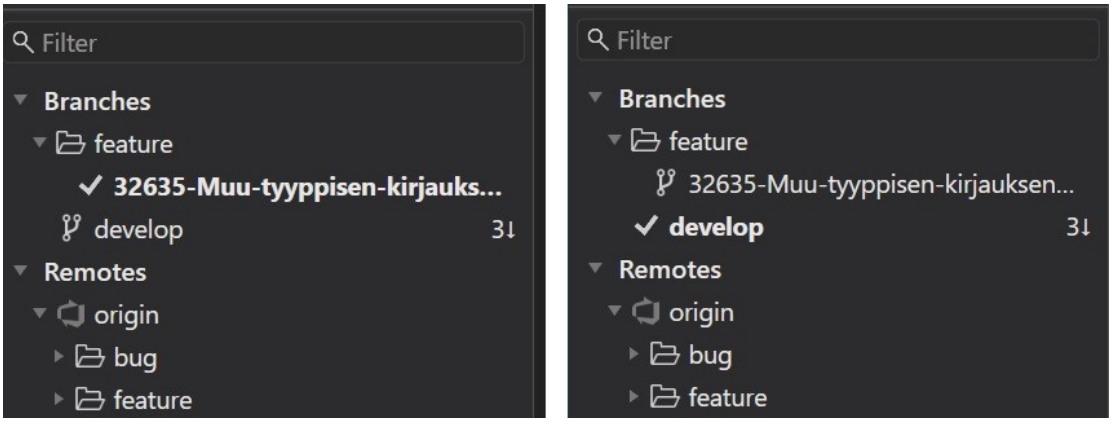
Kuvassa 9 aktiivisena oleva repositorio on TypeScript. Aktiivinen repositorio näkyy yleisnäkymässä aina Fork-nimen alla isommassa sarakkeessa. Repositorion nimen alla näkyy tieto siitä, millä haaralla kehittäjä on sillä hetkellä. Kuvassa 9 kehittäjä on TypeScript-repositorion master-haaralla. Lisäksi repositorio-välilehdillä aktiivista repositoriota on korostettu vaalean harmaalla välilehden pohjavärillä ja repositorion nimi näkyy vasemmassa reunassa muutoksista kertovassa sarakkeessa.

Kuva 9. Yleiskatsaus Forkin käyttöliittymästä (Fork, n.d.).



Vasemman reunan sarakkeesta löytyy myös lista repositorioon kuuluvista haaroista. Branches-otsikon alta löytyy kehittäjän omat haarat, sekä haarat, joita hän seuraa. Remotes-otsikon alta löytyy kaikki repositorioon liittyvät haarat. Haarojen välillä voidaan liikkua tuplaklikkaamalla haaran nimeä, ja valittu haara näkyy listassa lihavoituna. Kuvassa 10 on havainnollistettu valitun haaran näkymistä sovelluksessa. Ensimmäisessä tilanteessa kehittäjä on haaralla 32635-Muu-tyyppisen-kirjauksen-valinta, jolloin haaran nimi on lihavoituna ja sen edessä on pieni check-merkki. Seuraavassa tilanteessa kehittäjä on siirtynyt develop-haaralle, jolloin sen haaran nimi on lihavoituna ja check-merkki löytyy haaran nimen edestä.

Kuva 10. Kehityshaarojen näkyminen Forkissa.



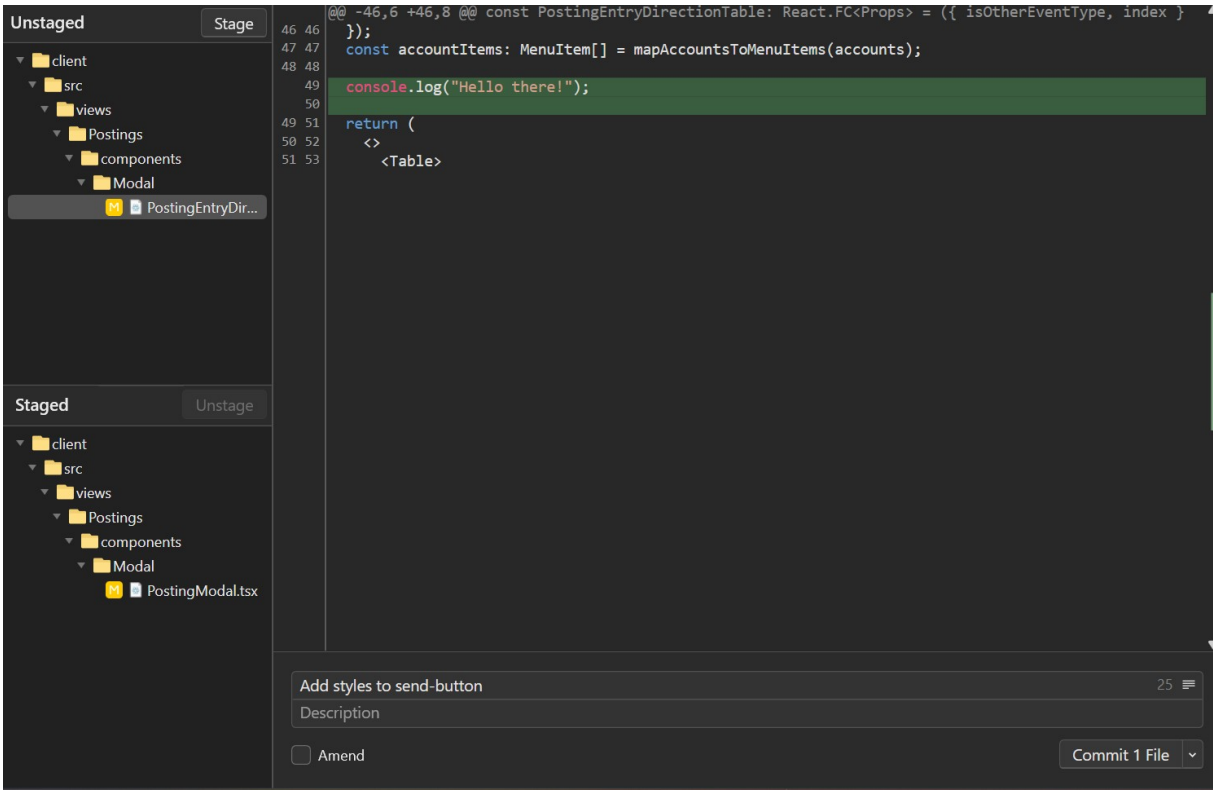
Valitulla haaralla tapahtuneet muutokset näkyvät vasemmassa yläreunassa haarojen listauksen yläpuolella. Kuvassa 9 nähtiin, kuinka TypeScript-otsikon alla oli tieto Local Changes (5), mikä tarkoittaa, että valitussa TypeScript-repositoriossa viidessä tiedostossa on tapahtunut muutoksia lokaalisti. Tätä klikkaamalla avautuu näkymä, josta lokaaleja muutoksia voidaan tarkastella. Kuvassa 11 on esitetty miten lokaalit muutokset näkyvät Forkissa.

Kaikki tallennetut muutokset näkyvät ensin vasemmanpuoleisessa palkissa Unstaged-otsikon alla, kuten kuvassa 11. Tiedostoja klikkaamalla oikealle puolelle avautuu näkymä, josta voidaan tarkastaa mitä muutoksia kuhunkin tiedostoon on tehty. Vihreällä värillä korostetut osiot ovat tiedostoon tehtyjä lisäyksiä, ja punaisella korostetaan mitä on poistettu. Kuvan 11 esimerkissä nähdään, että ainut muutos valittuun tiedostoon on console.log -tulostus. Tätä koodia on turha viedä versiohallintaan, joten muutos voidaan poistaa. Se tapahtuu Forkin kautta klikkaamalla tiedostoa hiiren oikealla näppäimellä ja valitsemalla avautuvasta valikosta ”Discard Changes”.

Sitä mukaa kun kehittäjä käy muutoksia läpi, hän voi kunkin tiedoston kohdalla klikata yläreunan Stage-painiketta, mikäli muutokset ovat sellaisia mitkä halutaan viedä versiohallintaan. Tiedostot siirtyvät alempaan sarakkeeseen, mistä nähdään mitkä kaikki tiedostot on laitettu stagelle. Muutoksia voidaan tarkastella myös stagella olevista tiedostoista ja tarvittaessa poistaa stagelta unstage-painikkeella.

Commitin tekeminen tapahtuu valitsemalla yhden tai useamman tiedoston stagelta, ja kirjoittamalla commit-tekstin sille varattuun ruutuun. Kuvassa 11 commit-tekstissä lukee ”Add styles to send-button”. Description on vapaaehtoinen kenttä, johon voi halutessaan kirjoittaa tarkemman kuvauksen muutoksista. Tämä teksti ei siirry versiohallintaan, mutta sitä voi tarkastella Forkin kautta myöhemmin. Commit-painikkeessa tehdään commit-komento, ja commitoidut tiedostot häviävät stage-näkymästä.

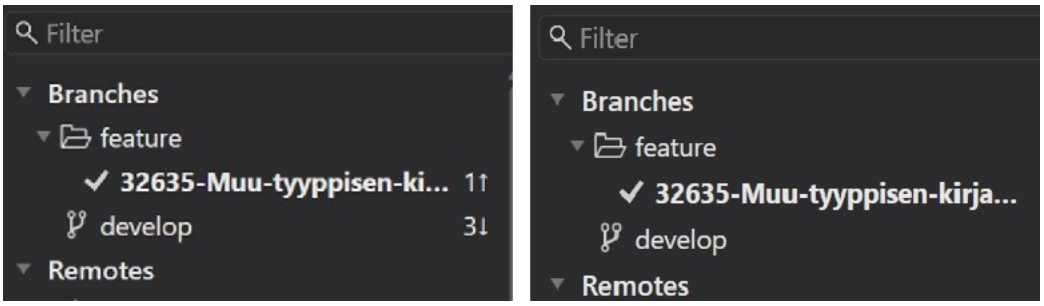
Kuva 11. Kehittäjän lokaaleja muutoksia.



Tarkastellaan vielä vasemmanpuoleista näkymää kehityshaaroista ennen push komennon tekemistä. Huomataan, että haaran nimen perään on ilmestynyt numero 1 ja nuolisymboli ylöspäin. Tämä implikoi sitä, että kehityshaaralla on yksi muutos, jota ei ole vielä puskettu versiohallintaan. Develop-haaran perässä on numero 3 ja nuolisymboli alaspäin, joka impikoi sitä, että develop-haaralle on tullut uusia muutoksia, joita ei vielä löydy lokaalisti.

Kuvasta 9 nähtiin, että Forkin yläreunasta löytyy komennot Fetch, Pull ja Push. Commitin tekemisen jälkeen voidaan painaa push-painiketta, jolloin numero ja nuolisymboli häviävät haaran nimen perästä. Siirtymällä develop-haaralle ja klikkaamalla Pull painiketta, saadaan uusimmat developille tehdyt muutokset päivitettyä ja symbolit myös develop-haaran perästä häviävät. Muutoksia on havainnollistettu kuvassa 12.

Kuva 12. Haarojen muutosten seuraaminen Forkissa.



Kun halutaan yhdistää develop-haaralle tulleet muutokset osaksi omaa työskentelyhaaraa, klikataan develop-haaraa hiiren oikealla näppäimellä. Oman kehityshaaran tulee olla aktiivisena, eli Forkissa se näkyy lihavoituna. Develop-haaran päällä oikealla hiiren klikkauksella avautuvasta valikosta valitaan komento Merge into <feature_haaran_nimi>. Kun kehittäjä on oikealla haaralla, näkyy haaran nimi valikossa oikein eikä sitä tarvitse erikseen asettaa. Tämän toiminnon jälkeen huomataan, että omalle kehityshaaralle on tullut muutoksia. Nämä muutokset voidaan lähettää versiohallintaan commit-viestillä Merge branch into develop, minkä Fork automaattisesti generoi, ja klikkaamalla sitten push-painiketta.

Yllä kuvattu työnkulku kattaa suurimman osan tavanomaisesta git-työskentelystä. Forkista löytyy tämän lisäksi monipuolisesti myös muita hyödyllisiä ominaisuuksia ja ohjelmistoa kehitetään jatkuvasti (Fork, n.d.). Verrattuna komentorivityöskentelyyn Fork tarjoaa erilaisen lähestymistavan git-työskentelyyn. Forkin avulla on helppo ymmärtää, mitä git-prosessissa todella tapahtuu, ja versiohallintaan lähetettävät tiedostot on helppo tarkistaa ennen push-komennon tekemistä. Toisaalta komentorivin kautta työskentely on Forkiin verrattuna tehokkaampaa, ja kehittäjä voi määritellä tarkasti mitä haluaa tehdä. Tämä voi olla hyödyllistä monimutkaisissa tilanteissa, kuten monen haaran hallitsemisessa tai tietyn commitin palauttamisessa. Nämä työkalut eivät kuitenkaan poissulje toisiaan, ja esimerkiksi opinnäytetyön tapauksessa Forkia ja komentoriviä on käytetty rinnakkain.

5 Manuaalisten kirjanpitovientien lisääminen kirjanpitomoduuliin

5.1 Suunnittelu

Manuaaliset kirjanpitoviennit -käyttäjätarina on jaettu kolmeen tehtävään. Jokaisen tehtävän tarkemmat määritykset käydään läpi lisämäärittelypalaverissa ja työmäärä

arvioidaan Pointing poker -menetelmällä. Käyttäjätarinan kriteerit tulevat pitkälti asiakkaiden tarpeista. Toteutuksen visuaalisesta ilmeestä vastaa Evitec Power Lending -tiimin käyttöliittymäsuunnittelija (UX Designer) Joel Hakulinen ja työ tehdään hänen suunnittelemiin Figma-kuviin perustuen.

Ensimmäisellä tehtävällä toteutetaan manuaalisten kirjanpitovientien kokonaisuuden runko, modaali, josta manuaalinen kirjanpidon tosite tehdään. Tavoitteena on rakentaa prototyyppi, jossa manuaalinen kirjanpidon tosite voidaan lisätä järjestelmään välttämättömillä tiedoilla. Tähän toteutukseen kuuluu uuden rajapinnan luominen manuaalisen kirjanpidon tositteen lisäämistä varten sekä lomakkeen lisääminen käyttöliittymälle.

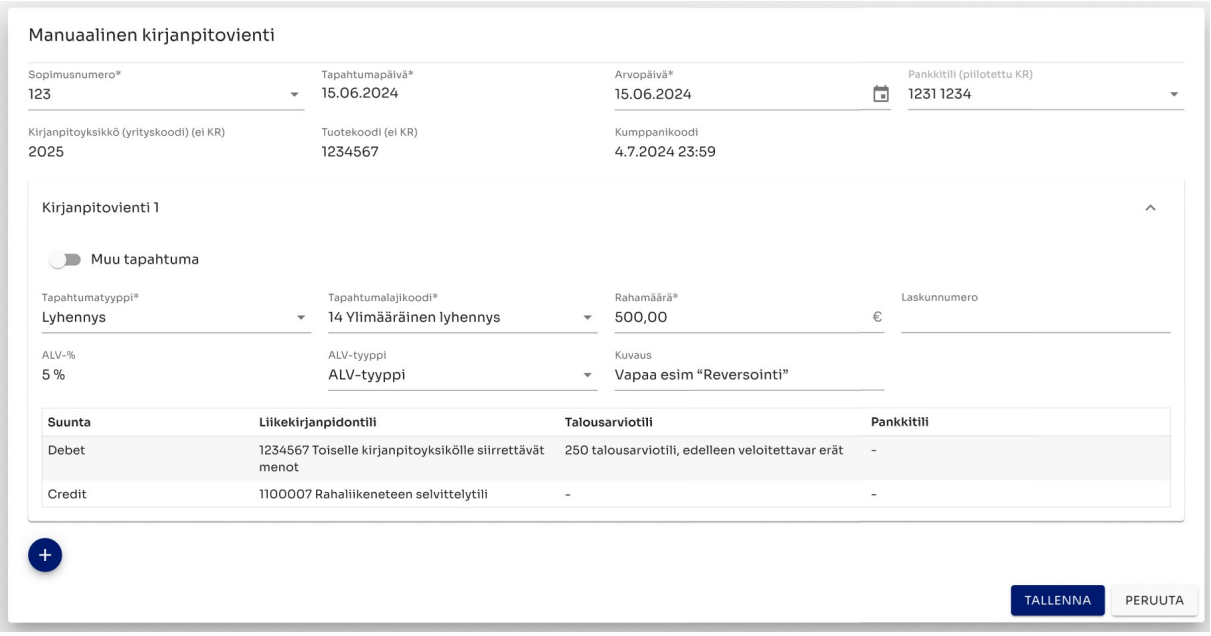
Lomakkeella on pystyttävä valitsemaan sopimus, johon manuaalinen kirjanpidon tosite liittyy. Kirjanpidon tositteelle on asetettava tapahtumapäivä ja kirjauspäivä. Sopimusnumeron valinnan jälkeen käyttäjä voi siirtyä luomaan yksittäistä kirjanpitovientiä. Kirjanpitoviennin lomakkeelta löytyy kentät tapahtumatyyppi ja tapahtumalaji, rahamäärä, laskunumero, ALV-%, ALV-tyyppi ja kuvaus. Kaksi ensimmäistä kenttää validoidaan niin, ettei niitä voi muokata ennen sopimuksen valitsemista. Rahamäärä on myös pakollinen tieto, ja kirjanpidon tosite on validoitava niin ettei Lähetä-painike aktivoidu ennen kuin kaikki pakolliset tiedot on täytetty. Käyttäjän on pystyttävä lisäämään useita yksittäisiä kirjanpitovientejä, jotka liittyvät samalle sopimukselle. Lomakkeen lähettämisen yhteydessä tietokantaan tulee tallentua kirjanpitoviennin yksilöivä Id, tieto manuaalisesta kirjauksesta sekä tieto käyttäjästä, joka on tehnyt manuaalisen kirjanpitoviennin.

Kenttien validoimisessa tulee huomioida, että esimerkiksi rahamääräkenttään voi syöttää ainoastaan lukuja ja päivämääräkenttiin ainoastaan päivämäärämuotoista dataa. Koko lomake tulee validoida niin, ettei sitä pysty lähettämään, jos kaikkia pakollisia tietoja ei ole annettu oikein. Lisäksi kentät tulee asettaa ympäristömuuttujaan siten, että niiden näkyvyyttä voidaan säädellä eri sidosryhmien välillä.

Valitun tapahtumalajin perusteella lomakkeelle aukeaa taulukko, josta käyttäjä näkee, onko kyseessä debet- vai kredittyyppinen vienti. Lisäksi taulukosta on pystyttävä näkemään kirjanpidontilit kyseisille vienneille. Nämä tiedot saadaan kirjaussäännöistä. Kuvassa 13 on esitetty käyttöliittymäsuunnittelijan visio kirjanpitomoduulista, ja toteutusta lähdetään tekemään tähän kuvaan pohjautuen. Tehtävälle asetettiin Pointing pokerin tuloksena kahdeksan pistettä, eli kyseessä on työmäärältään suurehko toteutus. Kuvassa 13 näkyvä arvopäivä vastaa kehitysympäristössä käytettävää termiä kirjauspäivää. Termien nimet

saattavat vaihdella sidosryhmien ja kehitysympäristöjen välillä, ja se on osa Power Lendingin toimintalogiikkaa. Termistöä voidaan kääntää sidosryhmäkohtaisesti vastaamaan tarpeita.

Kuva 13. Käyttöliittymäsuunnittelijan Figma-kuva manuaalisen kirjanpitoviennin modaalista (Hakulinen, 2024).

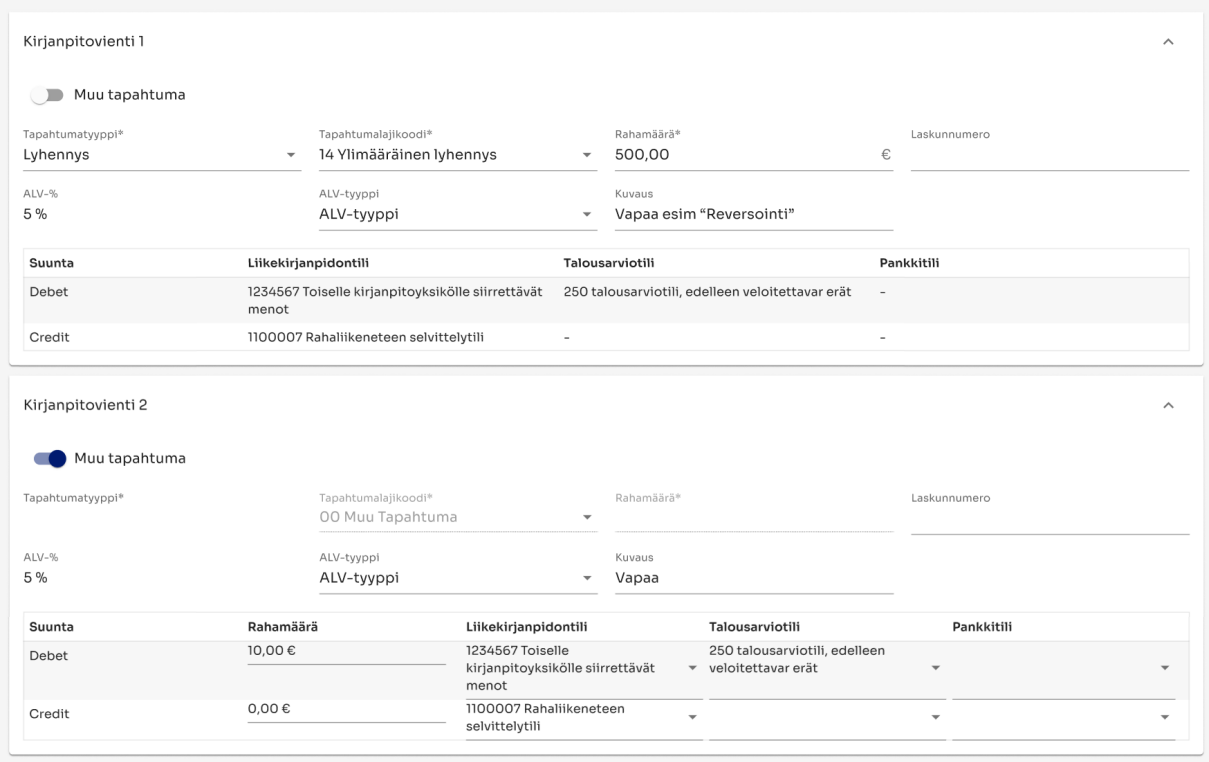


Käyttäjätarinan toisella tehtävällä luodaan muu-tyyppinen kirjanpitovienti, eli sellainen, joka ei vastaan mitään olemassa olevaa kirjaustyyppiä eikä sille ole kirjaussääntöjä. Tällä tehtävällä lomakkeelle lisätään switch-painike, jolla määritetään viennin olevan muu-tyyppinen. Painikkeen asettaminen aktiiviseksi poistaa käytöstä kentät tapahtumatyyppi, tapahtumalajikoodi ja rahamäärä, ja näihin kenttiin mahdollisesti asetetut arvot nollautuvat. Muu-tyyppisellä kirjauksella rahamäärä lisätään suoraan taulukkoon, jossa käyttäjä voi myös määritellä onko kyseessä debet- vai kredittyyppinen vienti. Viennit tulee validoida niin, että debet ja kredit summien tulee olla yhtä suuret, jos toinen vienneistä ei ole nolla. Muu-tyyppisellä kirjauksella kirjanpidontilit valitaan alasvetovalikosta.

Kirjanpitovienti, jolle on asetettu kirjaustyypiksi muu, täytyy voida tehdä ilman liittyvää sopimusta. Sitä varten määritellään, että mikäli kaikilla kirjanpidon tositteelle kuuluvilla kirjanpitovienneillä kirjaustyyppi on muu, tällöin sopimuksen valitseminen ei ole pakollista. Tällaista toiminnallisuutta voidaan tarvita esimerkiksi silloin, kun halutaan nollata ALV-tilit. Kuvassa 14 on esitetty käyttöliittymäsuunnittelijan visio toteutuksesta. Kuvan 14 yläosassa

on esitetty manuaalisen kirjanpitoviennin tavallinen käyttötapaus, ja alapuolella on esitetty kuinka näkymä muuttuisi, kun tapahtumatyypiksi valitaan muu.

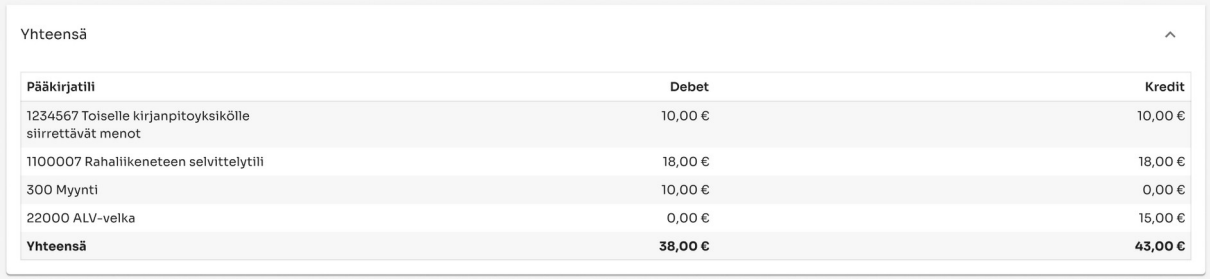
Kuva 14. Käyttöliittymäsuunnittelijan Figma-kuva muu-tyyppisen kirjanpitoviennin lisäämiselle (Hakulinen, 2024).



Tämän tehtävän toteutus vaatii myös muutoksia manuaalisten kirjanpitovientien rajapintaan, sillä Power Lending ei ole käsittänyt muu-tyyppistä kirjausta aikaisemmin. Tästä syystä myös kirjaustyypin ja tapahtumalajikoodin määrityksiä on muutettava tukemaan muu-tyyppistä kirjausta, jotta muu-tyyppisen kirjauksen hakeminen järjestelmästä on mahdollista. Tämä tehtävä määriteltiin lisämäärittelypalaverissa viiden pisteen arvoiseksi.

Manuaaliset kirjanpitoviennit käyttäjätarinan kolmannella tehtävällä toteutetaan yhteenvetotaulukko, josta käyttäjä näkee yhteenvedon kaikista samalla lomakkeella tekemistään kirjanpitovienneistä. Taulukossa tulee näkyä debet- ja kreditviennit tileittäin sekä niiden yhteenlasketut summat. Taulukon tietojen tulee täydentyä sitä mukaa, kun käyttäjä lisää lomakkeella uusia kirjanpitovientejä. Kuvassa 15 on esitetty käyttöliittymäsuunnittelijan visio taulukosta. Taulukko sijoitetaan lomakkeella alimmaiseksi kaikkien kirjanpitovientien alle, mutta ennen lähetä- ja peruuta -painikkeita.

Kuva 15. Käyttöliittymäsuunnittelijan Figma-kuva yhteenvetotaulukosta (Hakulinen, 2024).



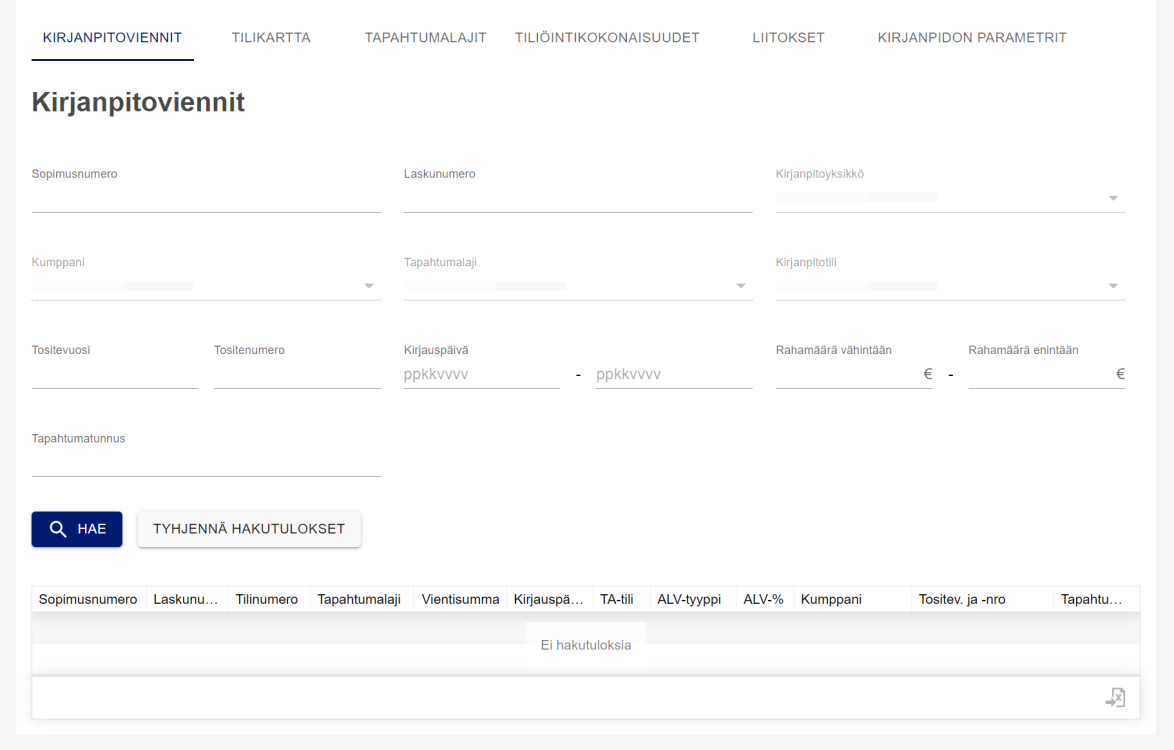
5.2 Toteutus

Manuaalisten kirjanpitovientien käyttäjätarina koostuu kolmesta tehtävästä. Sprintin päätteeksi valmistuneet tehtävät demonstroidaan sidosryhmille. Demossa varmistetaan, että toteutus vastaa sidosryhmien odotuksia, ja valmistuneesta työstä on mahdollista saada palautetta. Demossa nousseiden huomioiden perusteella lopullinen toteutus saattaa poiketa alkuperäisestä suunnitelmasta.

5.2.1 Painike manuaalisen kirjanpidon tositteen lisäämiselle

Projektien alkaessa käyttäjällä ei ole mahdollisuutta manuaalisesti luoda uusia kirjanpitovientejä Evitec Power Lending -järjestelmässä. Kirjanpidon tositteet syntyvät järjestelmään automaattisesti esimerkiksi laskun maksamisen tai rahasuorituksen saapumisen jälkeen. Kuvassa 16 havainnollistetaan lähtötilannetta Power Lending -sovelluksen kirjanpitomodaalissa. Alkutilanteessa käyttäjä voi hakea olemassa olevia kirjanpitovientejä erilaisia hakuehtoja hyödyntäen tai listata kaikki olemassa olevat kirjanpitoviennit tekemällä haun ilman hakuehtojen määrittelyä. Lista kirjanpitovienneistä ilmestyy hakuehtojen alla näkyvään taulukkoon, josta yksittäistä kirjanpitovientiä voidaan tarkastella yksityiskohtaisemmin.

Kuva 16. Alkutilanne Evitec Power Lending -kirjanpitomoduulissa.



Manuaalisten kirjanpitovientien toteutus aloitetaan lisäämällä Kirjanpitoviennit-sivulle pluspainike, jota klikkaamalla aukeaa manuaalisen kirjanpidon tositteen lisäämiseen tarkoitettu moduuli. Plus-painikkeen lisäämiseen käytetään Material-UI-kirjaston AddIcon-ikonia. Material-UI (MUI) on React-kirjasto, joka tarjoaa käyttövalmiita ja mukautettavia komponentteja käyttöliittymän nopeaan ja valmiiksi tyyliteltyyn toteutukseen. Komponentit ovat myös täysin kustomoitavissa. Kirjaston käyttöönotto vaatii asennuksen npm-pakettina, ja jokainen haluttu komponentti täytyy tuoda sovelluksen käyttöön import-lausekkeella. (Material UI, n.d.) Ikonit voidaan asentaa projektiin komentorivin kautta komennolla:

npm install @mui/icons-material @mui/material @emotion/styled @emotion/react

MUI-kirjastosta löytyy erilaisten ikoneiden lisäksi esimerkiksi tyyliteltyjä painikkeita, lomakkeita ja taulukoita. MUI-kirjaston avulla voidaan luoda mukautettuja teemoja määrittelemällä väripaletit, fontit ja layout-tyylit, mikä helpottaa brändin mukaisen käyttöliittymän luomista. (Material UI, n.d.)

Kirjaston asentamisen jälkeen kaikki halutut komponentit on tuotava import-lausekkeella osaksi sovellusta. Plus-painike (AddIcon) lisätään sovellukseen seuraavalla tavalla:

```
import AddIcon from "@mui/icons-material/Add";
```

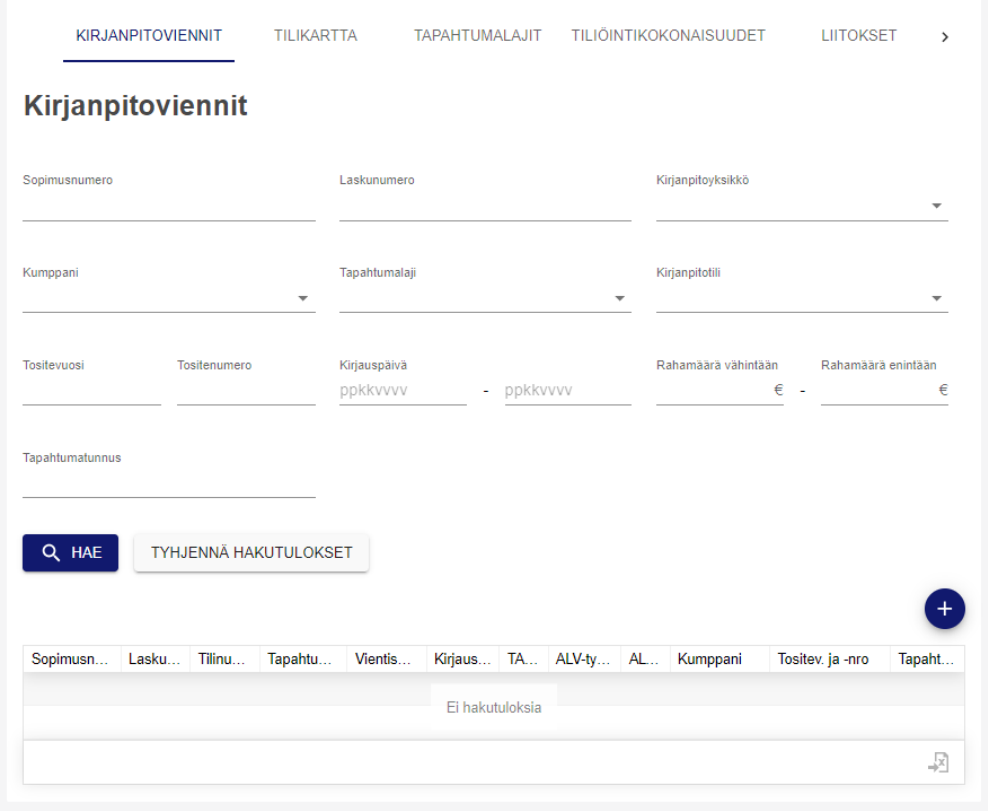
Mikäli sovelluksessa käytettäisiin useampaa erilaista ikonia, tulisi kaikki tuoda erikseen sovelluksen käyttöön siinä tiedostossa, jossa niitä käytetään. Importin jälkeen MUI-kirjaston komponentit ovat sovelluksen käytettävissä.

Ikonista halutaan tehdä linkkipainike, ja sitä varten se kääritään toisen MUI-kirjaston komponentin, Floating Action Button (FAB), sisälle. FAB-elementti täytyy yhtä lailla tuoda sovelluksen käyttöön importin avulla, jonka jälkeen sille voidaan määritellä väri, koko ja tapahtumankäsittelijäfunktio. Seuraava koodi havainnollistaa MUI-kirjaston komponenttien käyttöä sovelluksessa:

```
import { Fab } from "@mui/material";
import AddIcon from "@mui/icons-material/Add";

<Fab
    disabled={disabled}
    id={id}
    color="primary"
    size="small"
    onClick={handleClick}>
        <AddIcon />
</Fab>
```

Määritysten jälkeen painike saadaan näkyville käyttöliittymässä. Kuvassa 17 AddIcon-linkkipainike on lisätty sivun oikeaan reunaan, hakuehtojen ja hakutulostaulukon väliin. Painike tulee avaamaan modaalin, jonka luomisessa hyödynnetään lisää MUI-kirjaston komponentteja sekä React Final Form -kirjastoa.

Kuva 17. Kirjanpitoviennit-sivulle on lisätty pluspainike manuaalista kirjapitovientiä varten.



5.2.2 Modaalin luominen ja komponentteihin jako

Uuteen ikkunaan aukeava modaali voidaan luoda MUI-kirjaston Dialog-komponentin avulla. Tähän kokonaisuuteen kuuluu DialogTitle, joka määrittelee modaalin otsikon, DialogContent, joka määrittelee modaalin sisällön sekä DialogActions, joka määrittää modaalin toiminnallisuudet, toisin sanoen Lähetä- ja Peruuta -painikkeiden määrittelyt.

Reactin perusajatus on kokonaisuuden jakaminen pienempiin, uudelleen käytettäviin komponentteihin. Ennen lomakkeen toteutusta tehtiin Figma-kuviin pohjautuva suunnitelma siitä, mitä komponentteja tarvitaan ja kuinka ne tulisi asetella koodiin. Liitteessä 3 on esitetty komponenttirakenteen kehittymistä projektin edetessä. Tarvittavien komponenttien määrä arvioitiin alussa hieman alakanttiin, ja tiedostojen paisuessa toteutusta alettiin jakamaan pienempiin osiin liitteen 3 mukaisesti.

Komponenttirakenteen uloin taso pitää sisällään Dialog-komponenttien määritykset sekä React Final Formin määritykset. Uloimmalta tasolta löytyy lomakkeen validoimisesta vastaavat funktiot sekä määritykset sille, mitä tapahtuu, kun käyttäjä klikkaa Lähetä- tai

Peruuta -painikkeita. Komponenttirakenne muodostuu puurakenteen omaisesti niin, että kaikki uudelleen käytettävät elementit muodostavat omat komponenttinsa ja määrittelyt löytyvät yhdestä paikkaa. Toteutuksen edetessä huomattiin, että yksittäiset tiedostot alkavat paisua todella suuriksi, joten itsenäisesti toimivia kokonaisuuksia eroteltiin omiksi komponenteiksi.

5.2.3 Lomakkeen luominen React Final Form -kirjaston avulla

React Final Form on kirjasto, joka helpottaa lomakkeiden hallintaa React-sovelluksessa. Kirjasto asennetaan React-sovellukseen npm-pakettina:

npm install --save final-form react-final-form

React Final Form on hyödyllinen kirjasto erityisesti monimutkaisissa lomakerakenteissa, sillä se pystyy seuraamaan lomakkeen tilaa ja päivityksiä reaaliajassa. Näin ollen lomakkeen tietoja käsitellessä ei tarvita Reactin useState -hookeja tai Reduxia tilanhallintaan. Tämä vähentää virheiden riskiä ja koodista saadaan siistimpää. (React Final Form, n.d.-a) React Final Form otetaan asennuksen jälkeen käyttöön sovelluksessa import-lauseella:

```
import { Form } from "react-final-form";
```

React Final Form API käsittää joukon työkaluja ja komponentteja, joiden avulla lomakkeita voidaan rakentaa ja hallita React-sovelluksessa (React Final Form, n.d.-b). Manuaalisia kirjanpitovientejä luodessa käytettiin komponentteja <Form/> ja <Field/> sekä hookeja useField() ja useForm(). Kaikki halutut komponentit ja hookit tulee lisätä myös import-lauseella, jotta ne saadaan sovelluksen käyttöön.

<Form/>-komponentti on lomakerakenteen ulommainen komponentti, joka ympäröi koko lomakekokonaisuutta ja hallitsee lomakkeen tilaa. Toimintalogiikka <Form/>-komponentin taustalla on samankaltainen kuin Reactissakin: tila pävitetään vain tarvittaessa, ja yhden osion päivittyessä koko <Form/>-elementin ei tarvitse päivittyä. <Field/>-komponentti sen sijaan renderöityy uudelleen joka kerta, kun sen tila muuttuu, mutta renderöitymislogiikkaa voidaan hallita <Field/>-komponentin subscription-propsin avulla (React Final Form, n.d.-a). <Form/> ja <Field/> -komponenttien tilanhallintaa on hyödynnetty manuaalisen kirjanpitodon tositteen toteutuksessa muun muassa tapahtumatyyppi- ja

tapahtumalajikoodikenttien validoimisessa, missä määriteltiin, että nämä kentät eivät ole käytettävissä ennen kuin sopimusnumero on valittu.

<Form/>-komponentti vaatii toimiakseen vähintäänkin propsin onSubmit sekä yhden näistä kolmesta: component, render tai children (React Final Form, n.d.-b). Propsi onSubmit määrittelee, mitä tapahtuu, kun lomaketta lähetetään. Manuaalisten kirjanpitovientien tapauksessa onSubmit propsiin on määritelty funktiokutsu, jossa kaikki lomakkeen tieto kerätään yhteen ja lähetetään palvelinpuolelle. Lomakkeella käytetään myös render propsia, jonka tehtävänä on määritellä se, kuinka lomakkeen kentät renderöityvät käyttöliittymässä (React Final Form, n.d.-c). Manuaalisten kirjanpitovientien lomakkeella render-propsissa määritellään se, ettei lomaketta voida lähettää ennen kuin pakolliset kentät on täytetty, tarkistetaan lomake virheiden varalta sekä estetään saman lomakkeen uudelleenlähetys.

<Field/>-komponentilla luodaan lomakkeelle kenttiä. <Field/>-komponentteja voidaan luoda niin monta kuin tarvitaan, ja yksittäinen <Field/>-komponentti vastaa kyseisen kentän tilasta. Kenttä voi olla tyypiltään tekstikenttä, pudotusvalikko, checkbox-valintaruutu, radiobutton-painike tai jokin muu mukautettu HTML-elementti. <Field/>-komponentilla täytyy olla aina yksilöivä nimi sekä <Form/> komponentin tavoin jokin kolmesta propsista: component, render tai children. (React Final Form, n.d.-d)

Manuaalisten kirjanpitovientien lomakkeella suuri osa <Field/> komponenteista käyttää component-propsia, jossa määritellään minkä tyyppisestä kentästä on kyse. Yksinkertaisen <Field/>-komponentin määrittely voi näyttää esimerkiksi tältä:

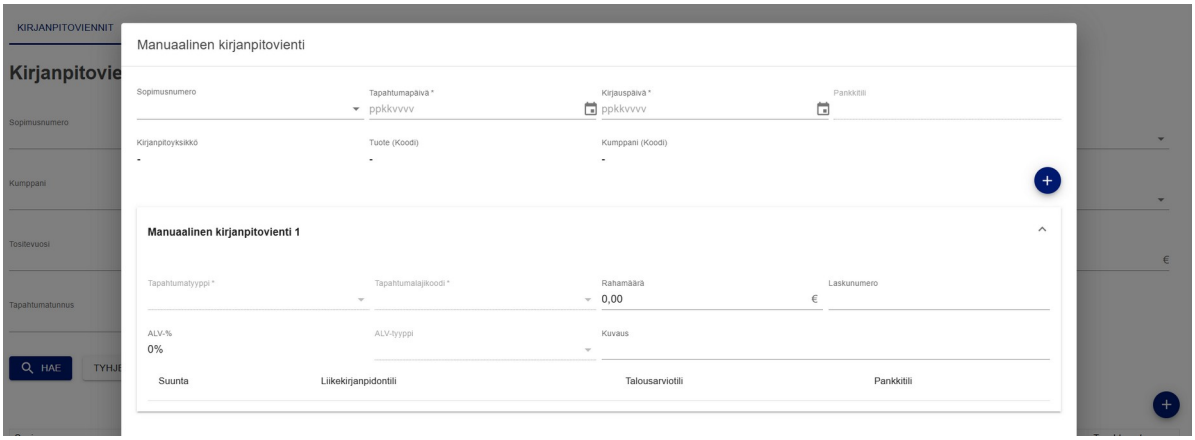
```
<Field
   name={"KokoNimi"}
   label={t("kokoNimi")}
   component={finalForm.RenderTextField}
/>
```

Kentälle on määritelty nimeksi KokoNimi ja component-propsissa määritellään, että kyseessä on yksirivinen tekstikenttä. Lisäksi <Field/>-komponentille on annettu propsi label, jossa määritellään käyttöliittymällä näkyvä teksti. T-kirjan nimen edessä viittaa i18next-käännöskirjastoon, jolla voidaan hallita millä kielellä käyttöliittymän tekstit näkyvät.

Yksittäisten kenttien tilaa voidaan käyttää missä tahansa lomakkeen sisällä olevassa komponentissa useField()-hookin avulla. Koko lomakkeen hallinta puolestaan onnistuu

useForm()-hookilla. Kuvassa 18 esitetään manuaalisten kirjanpitovientien modaali, jonne on lisätty lomake kirjanpidon tositteen yleisille kentille sekä omassa komponentissaan oleva kirjanpitovienti-lomake. Kuvassa 18 manuaalisten kirjanpitovientien modaaliin on lisätty kaikki suunnitelman mukaiset kentät käyttöliittymäsuunnittelijan Figma-kuviin perustuen.

Kuva 18. Kirjanpidon tosite, jossa on yksi kirjanpitovienti.



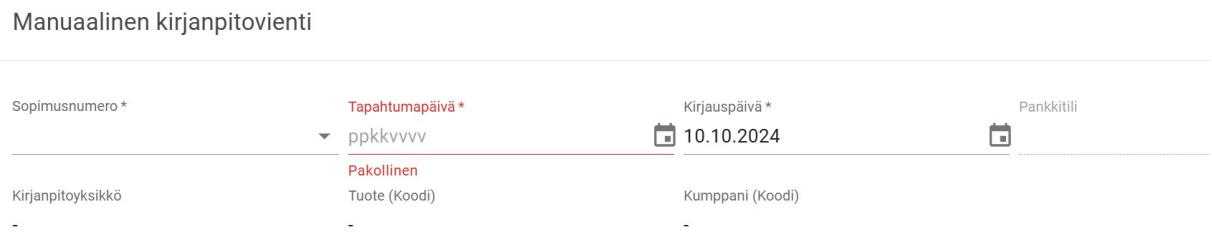
Tärkeä osa toteutusta on lomakkeen validoiminen. Validaatio tehdään koko lomakkeelle React Final Formin validate-propsilla sekä yksitäisille kentille required-propsilla. Kentät tulee validoida siten, että sopimusnumero, tapahtumapäivä, kirjauspäivä ja rahamäärä ovat pakollisia tietoja. Ennen sopimuksen valitsemista ei voi valita tapahtumatyyppiä tai tapahtumalajikoodia. Päivämääräkentät on validoitava niin, että niihin voidaan syöttää ainoastaan päivämäärämuotoista dataa ja rahamääräkenttään tulee voida syöttää ainoastaan numeroita. Seuraavassa koodiesimerkissä päivämääräkenttä on merkitty pakolliseksi käyttäen required-propsia ilman ehtoa, joka tarkoittaa, että required on aina tosi. Validate-propsille on annettu virheilmoitus, joka kertoo kentän pakollisuudesta silloin kun kenttä on tyhjä. Päivämääräkenttä on luotu finalForm.RenderDatePicker-komponenttityypillä ja sille on asetettu maksimiarvoksi kuluva päivä

```
<Grid item xs={3}>
  <Field
    name="Tapahtumapaiva"
    label={t("tapahtumapaiva")}
    component={finalForm.RenderDatePicker}
    maxDateToday
    validate={required(t("required"))}
    required
  />
```

```
</Grid>;
```

Kuvassa 19 on esitetty, kuinka kentän validaatio näkyy käyttöliittymällä virhetilanteessa. Mikäli kenttä jätetään tyhjäksi, punainen varoitusväri impikoi siitä, että virhe on tapahtunut, ja kentän alapuolelle ilmestyy kentän pakollisuudesta ilmoittava viesti. Myös tähtimerkki kentän nimen perässä impikoi kentän pakollisuudesta.

Kuva 19. Tapahtumapäivä on pakollinen tieto ja lomake ilmoittaa virheestä.



5.2.4 Ympäristömuuttujien lisääminen

Ympäristömuuttujilla mahdollistetaan sovelluksen eri versioiden ylläpito ja hallinta. Tavallisesti ohjelmistokehitystyössä sovellus on jaettu vähintään kehitys- ja tuotantoversioihin. Näiden lisäksi voi olla erilaisia testiversioita ja asiakaskohtaisia versioita. Ympäristömuuttujat määritellään .env-tiedostoissa. .env.development ja .env.production määrittelevät tuotanto- ja kehitysympäristöt, ja kaikki kustomoidut ympäristömuuttujat nimetään REACT_APP -alkuliitteellä, kuten seuraavassa esimerkissä:

```
REACT_APP_API_VERSION="1"
REACT_APP_TRANSLATION_LANGUAGES="fi,en,sv"
REACT_APP_USER_GROUPS="admin,visitor,user,moderator,anonymous"
```

Power Lending -tuotteen ominaisuuksiin kuuluu se, ominaisuuksia voidaan kustomoida sidosryhmien toiveiden mukaan, ja siksi niitä tulee voida hallita .env-tiedostossa. .env-muuttujien käyttäminen madollistaa sen, ettei koodia tarvitse jatkuvasti muokata, vaan erilaisialla määrityksillä saadaan luotua räätälöytyjä näkymiä.

Tehtävän määrityksissä kerrottiin, millä tavalla kentät halutaan näyttää eri ympäristöissä. Sen lisäksi ALV-kentät, joissa ei toistaiseksi ole vielä toiminnallisuutta, laitettiin käyttöliittymältä piiloon. Lomakkeen toteutuksessa tulee myös huomioida, että ulkoasu on

tyylitelty ja ammattimainen riippumatta siitä, mitkä ympäristömuuttujat ovat esillä tai piilossa. Piilotettujen kenttien paikalle ei haluta tyhjää aukkoa, vaan lomakkeen tulee olla mukautuva. Toteutus onnistuu MUI-kirjaston Grid-komponentilla. Mikäli env-muuttujan tietoihin ei jostain syystä päästäisikään käsiksi, on hyvä määritellä oletusarvot, jotka näytetään tällaisessa häiriötilanteessa. Oletusarvot määritellään merkkijonona, joka erotellaan split-funktion avulla pilkun kohdalta listaksi.Ympäristömuuttujat on määritelty .env-tiedoston lisäksi sovelluksen juuritiedostossa sekä configurationSlice-tiedostossa, mikä mahdollistaa sen, että muuttujat ovat koko sovelluksen käytössä kaikissa sen osissa.

Ympäristömuuttujan sisältämä tieto päästään lukemaan process.env.REACT_APP_USER_GROUPS -määritelmällä, ja käyttöliittymällä näytettävää tietoa voidaan säädellä esimerkiksi if else -lauseella.

```
const userGroup = process.env.REACT_APP_USER_GROUPS;

function App() {
  if (userGroup === "admin") {
    return <AdminView />;
  } else if (userGroup === "user") {
    return <UserView />;
  }
  return <DefaultView />;
}
```

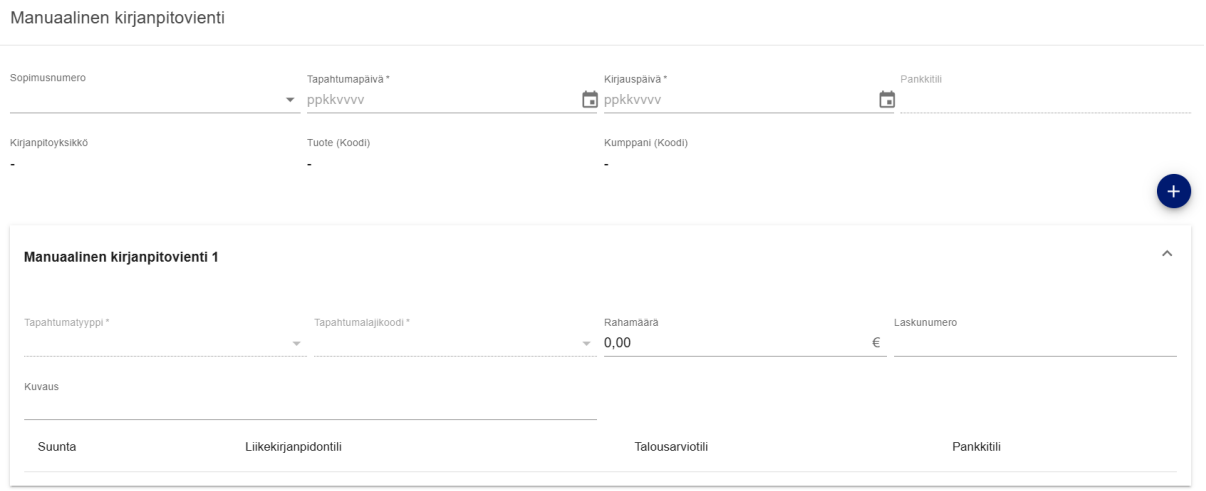
Lomaketta määrittäessä jokainen kenttä kääritään Grid-komponentin sisään, jolloin kentät mukautuvat käyttöliittymälle sen mukaan mitä kenttiä näytetään. Tämä kokonaisuus määritellään vielä ehtolauseen sisälle, jossa tarkistetaan, onko kyseinen kenttä määritelty näytettävien kenttien listassa. Kokonaisuudessaan kenttä määritellään siis seuraavalla tavalla:

```
{visibleFields.includes("Tapahtumapaiva") && (
    <Grid item xs={3}>
      <Field
        name="tapahtumapaiva"
        label={t("kirjauspaiva")}
        component={finalForm.RenderDatePicker}
        validate={required(t("required"))}
        required
      />
```

```
    </Grid>
  )}
```

Kuvassa 20 on esitetty käyttöliittymällä näkyvät muutokset .env.production-versiossa: lomakkeelta on piilotettu ALV-kentät joissa ei toistaiseksi ole toiminnallisuutta.

Kuva 20. ALV-tietoja sisältävät kentät on piilotettu env-muuttujalla.



5.2.5 Uuden rajapinnan luominen

Jotta käyttäjän syöttämät tiedot saadaan tallennettua tietokantaan, täytyy palvelinpuolelle luoda uusi rajapinta manuaalisten kirjanpitovientien lähettämistä varten. ASP.NET Core ympäristössä rajapinnat eli API-kutsut tehdään controller-luokan sisällä. Luodaan siis Controller-luokkaan uusi funktio manuaaliselle kirjanpidon tapahtumalle, joka vastaanottaa asiakaspuolella syötetyn datan DTO-tyyppisenä objektina.

DTO (data transfer object) toimii tiedonvälittäjänä. Tiedostoon on listattu ainoastaan toiminnallisuuksien kannalta välttämättömät kentät, joiden tietoja halutaan kuljettaa palvelin- ja asiakaspuolen välillä. DTO-objektista voidaan jättää pois liiketoimintaan liittyvät kentät, sisäiset tunnisteet ja metadata sekä tietokannan audikointikentät. Tiedon siirtäminen DTO-tyyppisenä objektina lisää tiedonsiirron tehokkuutta ja turvallisuutta. (Microsoft, 2022a)

Tietokantaluokat eli entiteetit määrittelevät objektin sisältämät tietokentät. Esimerkiksi käyttäjä voisi olla yksi tietokantaluokka, jonka pohjalta luodaan yksittäisiä käyttäjäobjekteja yksilöivillä tiedoilla. Jokaisen objektin tiedot tallentuvat tietokantaan omalle rivilleen. Käyttäjää kuvastava tietokantaluokka voisi näyttää tältä:

```
public class User
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
    public DateTime DateOfBirth { get; set; }
    public string SocialSecurityNumber { get; set; }
    public DateTime AccountCreated { get; set; }
}
```

Kun tietoa kuljetetaan sovelluksen sisällä, halutaan arkaluontoinen ja epäoleellinen tieto jättää pois. Luodaan DTO luokka, joka pitää sisällään vain välttämättömän informaation. Käyttäjään pohjautuva DTO-luokka voisi näyttää tältä:

```
public class UserDto
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public DateTime DateOfBirth { get; set; }
}
```

Controllerin tehtävä on käsitellä saapuvat HTTP-pyynnöt, ja yksi controller vastaa tiettyyn osoitteeseen tehtävästä tietyntyyppisestä HTTP-pyynnöstä. Opinnäytetyön tapauksessa manuaalisille kirjanpidon tapahtumille haluttiin tehdä POST-pyynnön vastaanottava Controller-metodi, ja API-kutsuun määriteltiin uusi osoite tätä varten. Tiedon hakemiselle, päivittämiselle ja poistamiselle tulee tehdä omat controller-metodit, jotka määritellään [HttpGet], [HttpPut] ja [HttpDelete] parametreilla. Controller-metodi palauttaa tiedon HTTP-pyynnön onnistumisesta tai epäonnistumisesta. Varsinainen toimintalogiikka määritellään Service-luokan metodissa, jota Controllerissa kutsutaan.

Tämäntyyppisellä eriyttämisellä sovelluksen eri osat ovat vähemmän riippuvaisia toisistaan, mikä tekee niistä helpommin ylläpidettäviä ja testattavia. Sovelluksessa hyödynnetään dependency injection -mallia, jonka ajatuksena on riippuvuuksien määrittäminen luokan construktoriin varsinaisen toimintalogiikan sijaan. Tällä tavoin luokkien rakenne saadaan yksinkertaisemmaksi ja koodista tulee helpommin ymmärrettävää. Malli tukee myös koodin uudelleenkäytettävyyttä (Microsoft, 2023a).

Luodaan Service-luokkaan metodi manuaalisen kirjanpidon tapahtuman käsittelyyn. Tässä metodissa määritellään, että jokaisella manuaalisella kirjanpitoviennillä tulee olla kuvauskentässä teksti ”Manuaalinen kirjanpitovienti”. Tätä varten tietokantaobjektia ja DTO-objektia tulee muokata, sillä kuvauskenttää ei aikaisemmin ollut olemassa. Uusi kolumni tulee luoda myös käytössä olevaan SQL Server tietokantaan. Uuden kolumnin luominen tapahtuu SQL-kielen ALTER TABLE -komennolla.

Manuaalisen kirjanpidon tositteen Service-metodissa kirjanpitoviennille määritellään yksilöllinen Id Guid.NewGuid() -metodilla, joka on osa .NETin perustoimintoja sisältävää kirjastoa. Service-metodi tallentaa myös kirjauksen luontiajankohdan päivämäärämuotoisena tietona, muut business-logiikan kannalta tarvittavat tiedot sekä DTO-objektin mukana saadut käyttäjän syöttämät tiedot. Tietojen käsittelyn jälkeen varsinainen tietokantaan tallentaminen tehdään funktiokutsulla hyödyntäen Power Lending-ohjelmaan jo aikaisemmin toteutettua metodia.

Dependency injection -malliin kuuluu luokan riippuvuuksien eriyttäminen omaan interfaceen. Interface määrittelee kaikki ne metodit tai ominaisuudet, joita luokka tarjoaa. Interface ei pidä sisällään mitään toimintalogiikkaa, vaan sen voidaan ajatella olevan sopimus, jossa luetellaan luokan tarjoamat metodit (Microsoft, 2024c). Service-luokkaan lisätyt metodit tulee lisätä myös luokkaa vastaavan interfacen sisään, jotta ne saadaan käyttöön.

Lopuksi metodi tulee vielä testata, jotta voidaan varmistua siitä, että se toimii odotetulla tavalla. Testaamiseen käytetään xUnit -yksikkötestauskirjastoa ja testit kirjoitetaan omaan testiluokkaan. Dependency injection -malli helpottaa testien tekemistä, sillä sen myötä voidaan ottaa käyttöön mokkaus (mocking) eli luokan riippuvuuksien simuloiminen. Tämä mahdollistaa sen, että testissä voidaan keskittyä yksittäisen metodin testaamiseen, ei sen riippuvuuksien. Mokkaus etenee luomalla ensin Mock-objekti, jossa määritellään mitä riippuvuutta se simuloi. Seuraavaksi määritellään, miten Mock-objekti käyttäytyy tietyissä metodikutsuissa. Sillä voidaan esimerkiksi määrittää oletusarvo, joka palautetaan aina

metodikutsussa. Tämän jälkeen Mock-objektia voidaan käyttää osana testiä. Testituloksessa nähdään, miten testi reagoi, kun siihen liitetty riippuvuus palauttaa oletusarvon. (Riordan, 2024)

Varsinaisen testimetodin runko rakentuu kolmesta osasta: 1) arrange, 2) act ja 3) assert. Ensimmäisessä osassa testiobjekti alustetaan, seuraavaksi testattava metodi suoritetaan alustetulla datalla, ja lopuksi määritetään mitä testin pitäisi palauttaa (Riordan, 2024). Seuraavassa koodiesimerkissä havainnollistetaan xUnit-testifunktion toimintaa yleisellä tasolla:

```
[Fact]
public async Task Uusi_Kayttaja_Onnistunut_Luonti()
{

    // Arrange
    var request = new UserRequestDto
    {
        KayttajaTunnus = "testUser",
        Etunimi = "Maija",
        Sukunimi = "Mehiläinen",
        Liittymispaiva = new DateTime(2024, 09, 05),
    };

    // Act
    var response = await _userService.Uusi_Kauttaja(request);

    // Assert
    response.Should().NotBeNull();
    response.AccountCreated.Should().BeTrue();
    response.KayttajaTunnus.Should().Be("testUser");
    response.Etunimi.Should().Be("Maija");
}
```

Arrange-osassa luodaan objekti, joka simuloi DTO-objektin mukana saatavaa tietoa. Act-osiossa kutsutaan Service-luokan metodia ja asetetaan sille yllä määritetty DTO-tyyppinen objekti parametriksi. Aikaisemmin luotua funktiota siis kutsutaan testidatalla. Assert-osassa määritellään, mitä testin tulisi palauttaa. Määrittelystä käy ilmi, että testitulos ei saa olla tyhjä (null) ja AccountCreated-arvon tulee olla tosi. Lisäksi tarkistetaan, että tiedot käyttäjätunnus ja nimi ovat samat kuin Arrange-osiossa määritellyt arvot.

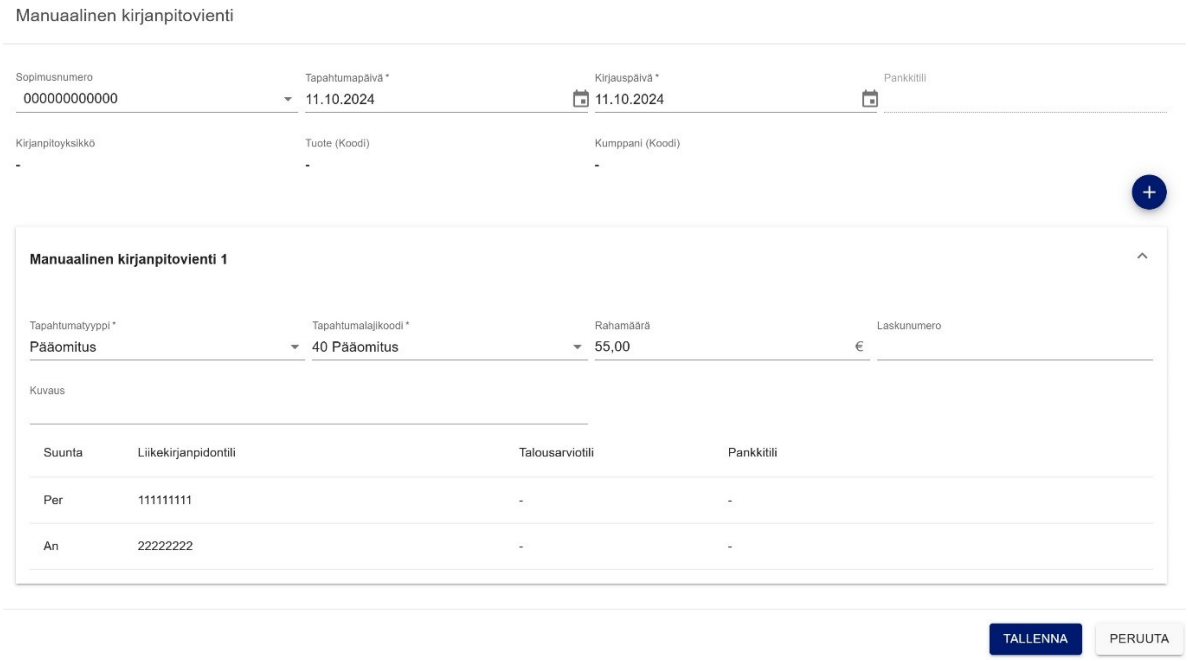
Sen lisäksi, että uutta toimintoa varten luotu testimetodi palauttaa onnistuneen tuloksen, tulee varmistaa, että kaikki aikaisemmin luodut testit menevät myös läpi, eikä koodimuutokset ole hajottaneet mitään aikaisemmin luotua toteutusta.

5.2.6 Tietojen näyttäminen taulukossa

Kun käyttäjä täyttää lomakkeella tietoja tapahtumatyyppi- ja tapahtumalajikoodikenttiin, aktivoituu taustalla kirjaussäännöt, joiden perusteella määräytyy mille tileille asetettu rahasumma siirtyy. Toteutusta varten sovelluksen palvelinpuolelle on luotava kaksi uutta GET-pyyntöä Controller-luokkiin. Toinen metodeista tehdään tapahtumatyypin Controller-metodiin ja toinen kirjaussäännöistä vastaavaan Controlleriin. Myös tapahtumatyypin Service-luokkaan luodaan uusi metodi, joka hakee tapahtumalajikoodia tuotteen perusteella. Lisäksi joihinkin olemassa oleviin funktioihin täytyy tehdä pieniä muutoksia, jotta niitä voidaan uudelleen käyttää manuaalisten kirjanpitovientien tapauksessa. Tällainen pienempi muutos voi tarkoittaa esimerkiksi sitä, että jokin aikaisemmin vakiona määritelty parametri saattaakin manuaalisen kirjanpitoviennin tapauksessa olla tyhjä. Myös interface-luokkia täytyy muistaa muokata aina kun uusia metodeita on määritelty sekä tehtävä testit uusille metodeille.

Asiakaspuolen toteutus on melko yksinkertainen sen jälkeen, kun palvelinpuolelle tehtävät hakutoiminnot on saatu kuntoon. Taulukko määritellään MUI-kirjaston Table-komponenttia hyödyntäen, ja sille määritellään samanlaiset tyylit, mitä myös muualla Power Lending sovelluksessa on käytetty. Taulukossa näytetään kirjanpitoviennin suuntaparametri debet tai kredit sekä liikekirjanpitotili. Lisäksi taulukossa on määritelty sarakkeet talousarviotilille sekä pankkitilille, mutta näille sarakkeille ei haeta tietoja tässä vaiheessa toteutusta. Kuvassa 21 on esitetty toteutus käyttöliittymällä taulukon lisäämisen jälkeen.

Kuva 21. Kirjanpoviennille lisätty kirjaussäännöt esittävä taulukko.



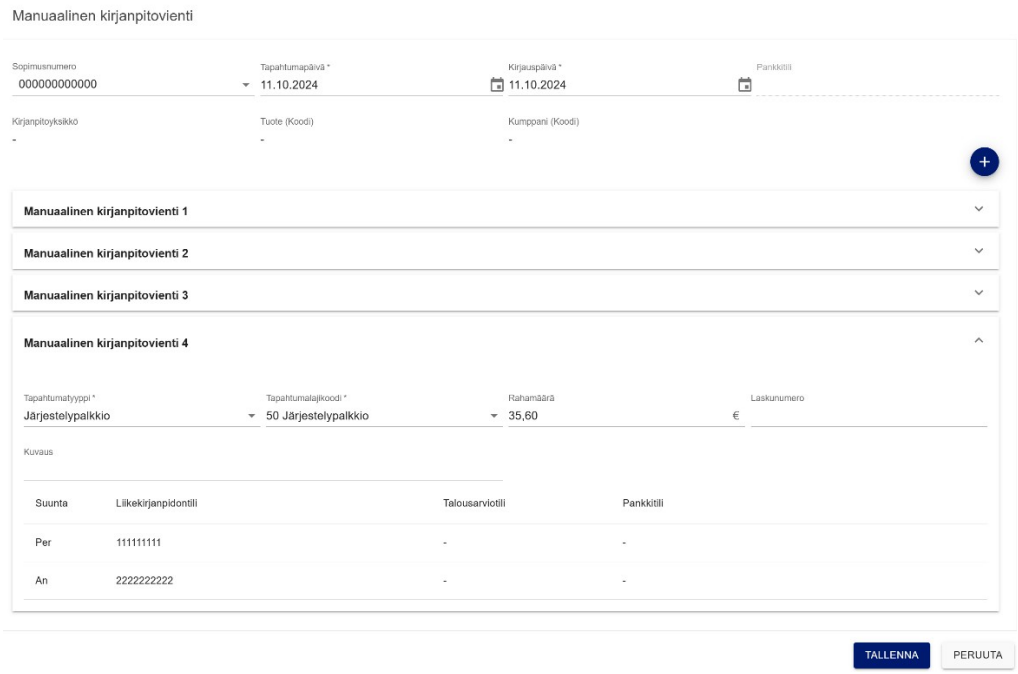
5.2.7 Useiden vientien tekeminen samalla lomakkeella

Manuaalisen kirjanpoviennin lisääminen järjestelmään onnistuu tässä vaiheessa toteutusta kirjanpidollisesti välttämättömillä tiedoilla. Kun lomakkeella määritellyt pakolliset kentät on täytetty, tallenna painike aktivoituu ja palvelinpuolen toteutus käsittelee pyynnön onnistuneesti. Käyttäjä näkee kirjanpoviennin suunnan ja liikekirjanpitotilin numeron taulukosta. Seuraavaksi lisätään mahdollisuus useiden vientien tekemiselle samalla lomakkeella. Sitä varten modaaliin lisätään samanlainen pluspainike, joka tehtiin heti toteutuksen alussa. Tässä voidaan hyödyntää samaa komponenttia.

Plus-painikkeelle määritellään yksilöivä id, joka erottaa sen muista samanlaisista komponenteista, sekä onClick-tapahtuma, jolla luodaan uusi tyhjä lomake kirjanpotivienneille. OnClick-tapahtumalle määritellyssä funktiossa alustetaan uusi kirjanpotivientiä kuvastava objekti tyhjillä arvoilla. Objektit asetetaan listaan, ja määritellään, että viimeisimpänä lisätty näkyy käyttöliittymällä avoimena ja uuden objektin lisäämisen yhteydessä aikaisemmin luodut haitarit sulkeutuvat, jolloin ne vievät käyttöliittymältä vähemmän tilaa. Jokaisessa kirjanpotivienti-haitarissa on pieni nuolisymboli oikeassa reunassa, josta ikkuna voidaan avata ja sulkea. Uusien kirjanpotivientien lisäämistä lomakkeelle on havainnollistettu kuvassa 22. Luotuja

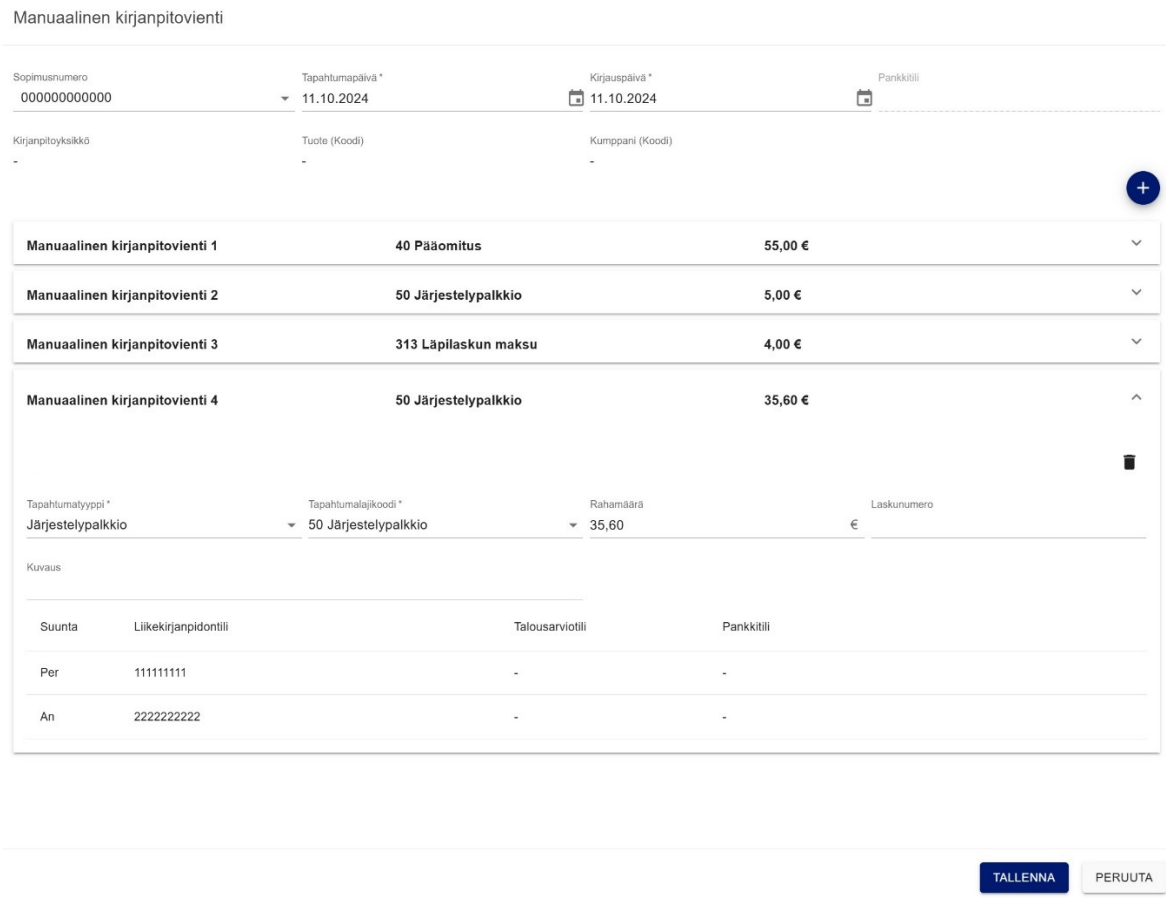
lomakkeita tulee voida myös poistaa, ja tätä varten lisätään kirjanpitoviennin lomakkeelle pieni roskakori-ikoni.

Kuva 22. Useiden kirjanpitovientien lisääminen kirjanpidon tositteelle.



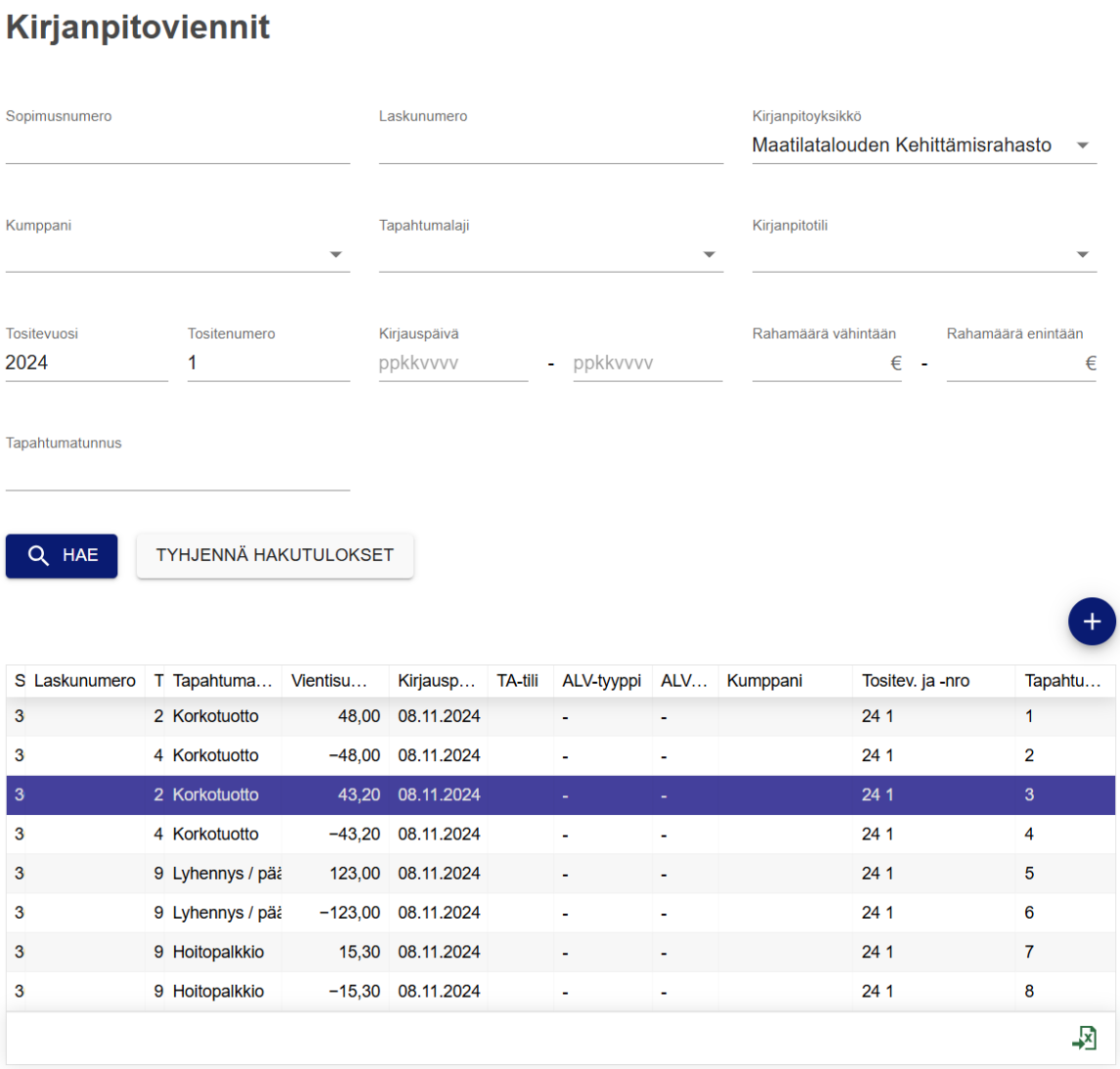
Viimeisenä lisäyksenä yksittäisen kirjanpitoviennin otsikkotasolle lisätään tiivistelmä kirjapitoviennin sisällöstä. Tämä helpottaa käyttäjää hahmottamaan mitä tietoja lomake pitää sisällään silloin, kun vientejä on paljon. Figma-kuvien mukaisesti otsikkotasolle lisätään tieto tapahtumatyypistä ja rahamäärästä. Kuvassa 23 on esitetty manuaalisten kirjanpitovientien lomakkeen viimeistelty lopputulos ensimmäisen tehtävän valmistuttua.

Kuva 23. Ensimmäinen tehtävä manuaalisten kirjanpitovientien osalta on valmis.



Kun käyttäjä painaa tallenna painiketta, lomake sulkeutuu ja käyttäjä palaa kirjanpitoviennit-sivulle onnistuneen tallennuksen jälkeen. Mikäli tallennuksessa menisi jotain pieleen, siitä näytetään virheviesti. Kuvassa 24 on havainnollistettu tilannetta, jossa manuaalinen kirjanpidon tapahtuma on luotu onnistuneesti ja käyttäjä palaa kirjanpitoviennit-sivulle. Sivulla olevaan hakutaulukkoon avautuu luodut kirjanpitoviennit. Kirjanpitovienti voidaan avata hakutulostaulukosta vielä yksityiskohtaista tarkastelua varten. Tämä toiminto tulee suoraan Power Lendingin aikaisemmasta toteutuksesta. Sarakkeet sopimusnumero ja tilinumero on piilotettu kuvaa varten säätämällä hakutulostaulukon sarakkeiden leveyksiä käyttöliittymällä.

Kuva 24. Kirjanpidon tosite näytetään hakutulostaulukossa.



5.2.8 Muu-tyyppisen tapahtuman lisääminen

Kirjaustyyppi ”muu” on tarkoitettu sellaisille kirjanpitovienneille, joille Evitec Power Lending -järjestelmässä ei ole valmiita kirjaussääntöjä. Muu-tyyppiselle kirjaukselle syötetään manuaalisesti ne tiedot, jotka muissa tapauksissa tulevat kirjaussääntöjen kautta. Käyttäjä määrittelee, tekeekö hän debet- vai kreditviennin, mikä on viennin rahamäärä ja mitkä ovat liikekirjanpitotili, talousarviotili ja pankkitili.

Muu-tyyppisen kirjauksen luominen toteutetaan lisäämällä kirjanpitovienti-lomakkeelle switch-painike, jota klikkaamalla tapahtumatyyppi määritellään muu-tyypiksi. Kuvassa 27 näkyy kirjanpitovientilomakkeelle lisätty painike, jossa kirjaustyyppi ”muu” on asetettu aktiiviseksi. Painiketta klikkaamalla kentät tapahtumatyyppi, tapahtumalajikoodi ja

rahamäärä asettuvat tilaan, jossa niitä ei voida muokata. Käyttäjän mahdollisesti näihin kenttiin asettamat tiedot nollautuvat, kun muu-tapahtumatyyppi asetetaan aktiiviseksi. Muu-tyyppisen kirjauksen tiedot täytetään suoraan lomakkeen alapuolella esitettävään kirjaussääntö-taulukkoon, kuten kuvassa 25.

Switch-painike on olemassa valmiina komponenttina MUI-kirjastossa. Komponentti kääritään lisäksi FormControlLabel-komponentin ja Field-komponentin sisälle. FormControlLabel kuuluu myös MUI-kirjastoon ja sen avulla saadaan asetettua painikkeelle käyttöliittymällä näkyvä tekstikenttä. Field-komponentti on osa React Final Form -kirjastoa ja sen avulla saadaan määriteltyä komponentin tila Formille, ja sitä pystytään hyödyntämään myös muualla sovelluksessa. Switch-komponentti näyttää koodissa seuraavalta:

```
<Switch
  {...input}
  color="primary"
  checked={input.value}
  onClick={resetFields}
  onChange={(_event: ChangeEvent<HTMLInputElement>, checked: boolean) => {
    input.onChange(checked);
  }}
/>
```

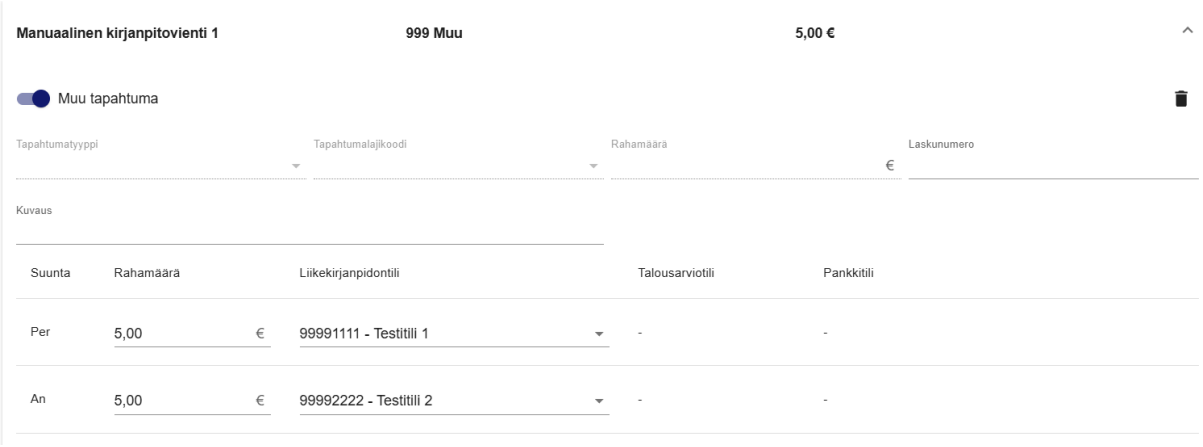
Switch-painike asetetaan käyttöliittymällä jokaiselle kirjanpitovienti-lomakkeelle. Jos käyttäjä luo kirjanpidon tositteelle useita kirjanpitovientejä pluspainikkeesta, jokaisella lomakkeella on oma switch-painike, jolla kirjanpitoviennistä voidaan tehdä muu-tyyppinen kirjaus. Kuvassa 25 kirjaustyyppi muu on asetettu aktiiviseksi ja taulukko tietojen täyttämistä varten aukeaa. Normaaleilla tapahtumatyypeillä taulukko avautuu ja tiedot täyttyvät sitä mukaa, kun käyttäjä täyttää lomaketta. Taulukko aktivoituu, kun tapahtumalajikoodi on valittu tai rahamäärä syötetään. Muu-tyyppisellä viennillä taulukko tulee näkyville heti switch-painikkeen painamisen jälkeen. Muu-tyyppistä tapahtumaa varten taulukkoon lisättiin uusi sarake, rahamäärä. Rahamäärä-kenttä on tässä tapauksessa eri, mitä kirjanpitovientilomakkeen ylemmällä tasolla, sillä summia täytyy voida muokata erikseen. Samaa kenttää ei voida tässä tapauksessa uudelleen käyttää. Palvelinpuolella kenttien arvoja käsitellään samalla tavalla.

Alkuperäisestä käyttöliittymäsuunnitelmasta poikettiin sen verran, että rahamääräsarakkeesta tehtiin kiinteä osa taulukkoa ja se näkyy myös silloin, kun

tapahtumatyyppi ei ole muu. Toteutettu ratkaisu tuntui käyttäjäystävällisemmältä, kuin se, että sarakkeet vaihtuisivat kirjaustyypin mukaan. Kun kirjaustyyppi ei ole muu, manuaalisen kirjanpitoviennin rahamäärä täytetään edelleen kirjanpitovienti-lomakkeelle, ja taulukossa näytetään ikään kuin yhteenvetona lomakkeelle lisätty rahamäärä.

Lisäksi liikekirjanpitotili-saraketta muutettiin käyttäjäystävällisemmäksi niin, että sarakkeessa näytetään tilinumeron lisäksi myös tilin nimi. Liitteessä 1 on esitetty tavallisen manuaalisen kirjanpitoviennin lisäämisessä tapahtuneet muutokset, sekä samassa kuvassa alapuolella muu-tyyppinen manuaalinen kirjanpitovienti. Lomake on kokonaisuutena yhtenäisemmän näköinen, kun molemmissa tapauksissa taulukko näyttää samalta. Myös muu-tyyppisen tapahtuman tiedot lisättiin näkymään manuaalisen kirjanpitoviennin haitarin otsikkoon.

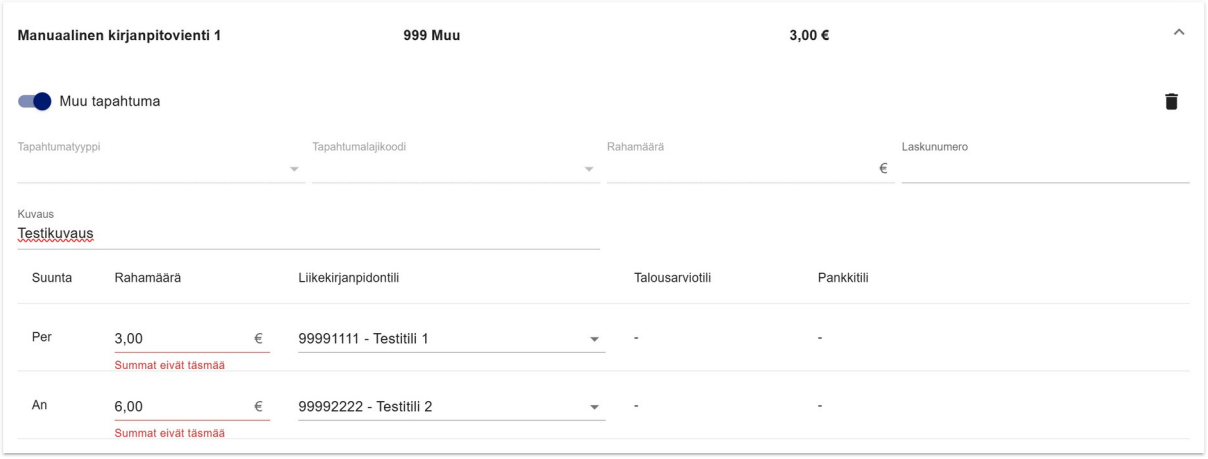
Kuva 25. Muu-tapahtumatyyppi asetettu aktiiviseksi ja tiedot asetettu suoraan taulukkoon.



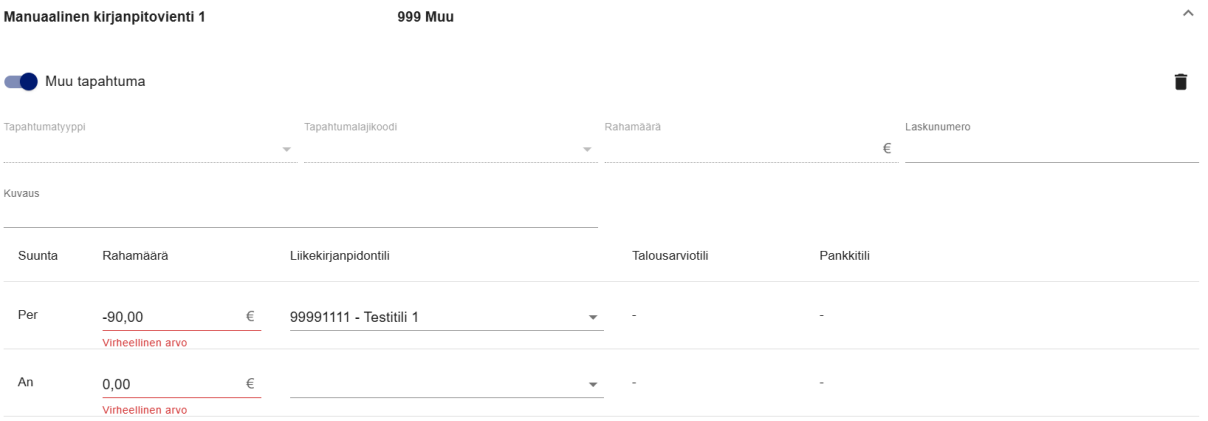
5.2.9 Kirjanpidon tositteen validoiminen

Yksittäisellä kirjanpitoviennillä joko debet- tai kredit -kenttä voidaan jättää tyhjäksi, jos tehdään vienti vain toiseen suuntaan, mutta mikäli tehdään sekä debet- että kredit -vienti samalla kirjanpitoviennillä, on syötettyjen rahamäärien täsmättävä. Kuvassa 26 on esitetty virheellinen vienti, jossa käyttäjä on syöttänyt lomakkeelle eri suuruiset rahamäärät. Tässä tilanteessa käyttäjälle näytetään varoitusteksti, joka ohjaa käyttäjää lomakkeen oikeantyyppiseen täyttämiseen. Myös negatiiviset rahamäärät ja nolla johtavat virhetilanteeseen, kuten kuvassa 27 nähdään. Mikäli halutaan tehdä vienti vain toiseen suuntaan, on rivi jätettävä tyhjäksi.

Kuva 26. Virheelliset rahamäärät muu-tyyppisellä kirjanpitoviennillä.



Kuva 27. Negatiiviset luvut ja nolla johtavat virhetilanteeseen.



Kirjanpitovientejä voidaan siis tehdä esimerkiksi kreditsuuntaan jättämällä debet-rivi kokonaan tyhjäksi, mutta kirjanpidon tositteella kokonaisuutena debet- ja kredit -vientien on aina täsmättävä. Kreditvientejä voi siis olla vaikka kymmenen, kunhan niiden yhteenlaskettu summa vastaa kaikkien debetvientien yhteenlaskettua summaa. Liitteessä 2 on esitetty oikeanlainen tapa tehdä kirjauksia.

Tällainen tilanne, jossa debet- ja kredit -vientejä täytyy tehdä eri määrä, voi tulla esimerkiksi silloin, kun yrityksen tilille tulee tuhannen euron suuruinen maksusuoritus, joka kirjataan debetpuolelle. Rahamäärästä kohdistetaan tietty summa korkoihin, toinen lainanhoitokuluihin ja loput muihin rahoituskuluihin. Lopputuloksessa kohdistettujen summien (kredit) on vastattava debettilille tullutta tuhannen euron suoritusta. Liitteessä 2 on havainnollistettu vastaavaa tilannetta. Liitteestä 2 nähdään, että debet (per) vientejä on

vain yksi kappale ja kredit (an) vientejä kolme. Kreditvientien yhteenlaskettu summa 500 + 33 + 467 = 1000 vastaa debetviennin rahamäärää, jolloin lomake on validi.

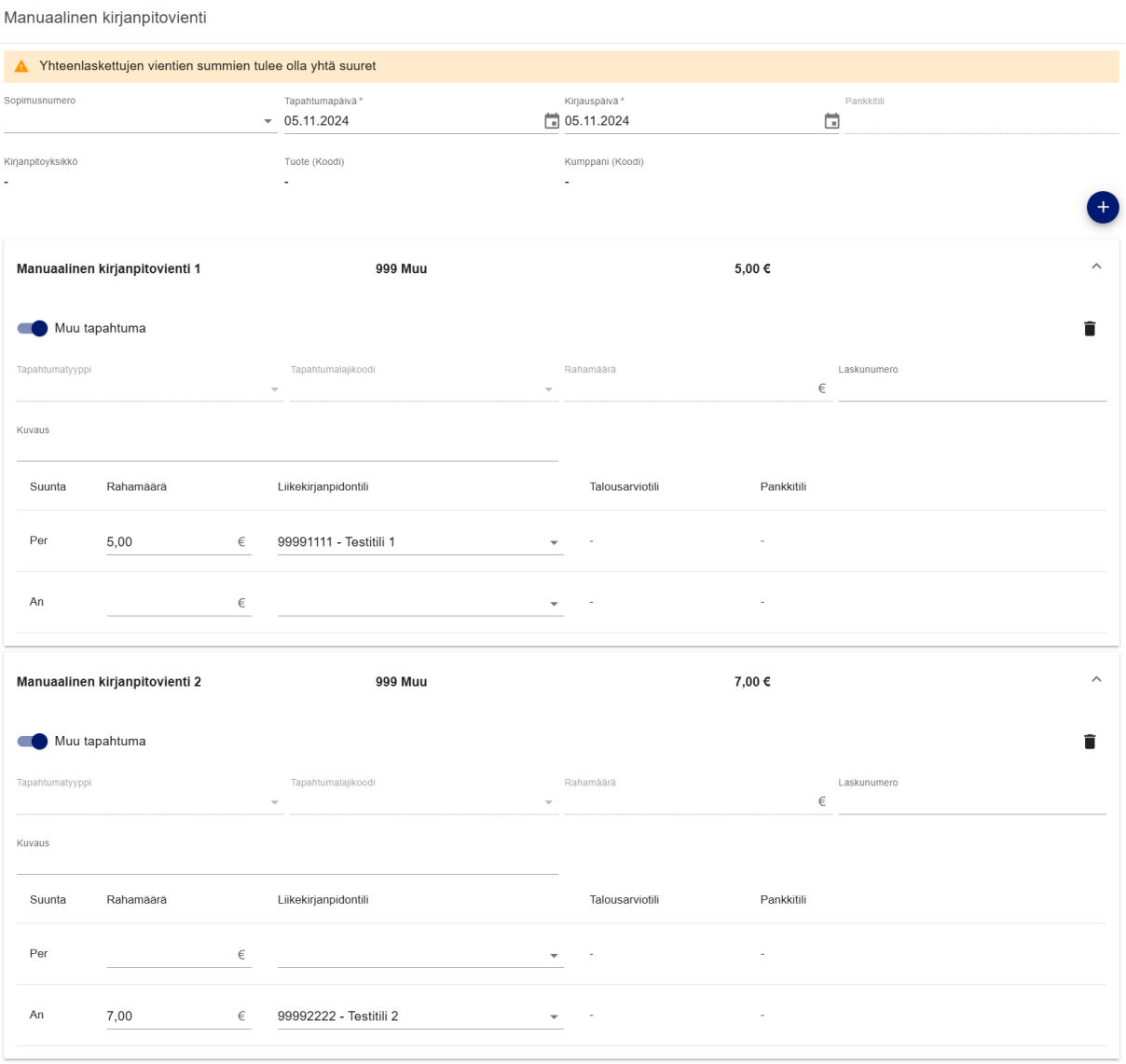
Kirjanpitoviennin rivitasolla annetaan virheviesti, jos debet- ja kredit -vientien rahamäärät ovat suuremmat kuin nolla ja summat eivät ole yhtä suuret, kuten aikaisemmin esitetyssä kuvassa 26 nähtiin. Mikäli toinen riveistä jätetään tyhjäksi, virheilmoitusta ei tule. Järjestelmän on kuitenkin validoitava, että koko kirjanpidon tositteen kaikkien debet- ja kredit -vientien yhteenlasketut rahamäärät vastaavat toisiaan. Järjestelmä pitää kirjaa debet- ja kreditvientien yhteenlasketuista rahamääristä, ja tieto summasta päivittyy joka kerta, kun rahamääräkenttää muutetaan.

Mikäli lomakkeella ei ole muita virheitä, näytetään modaalin yläreunassa palkki, joka kertoo virheestä. Tätä on havainnollistettu kuvassa 28. Ilmoituspalkki varaa pienen tilan modaalista ja sen näyttäminen ja piilottaminen muuttaa modaalin kokoa hieman. Siitä syystä se näytetään vasta, kun kirjanpidon tositteella ei ole muita virheitä. Modaalin koon vaihtuminen useasti lomakkeen täyttämisen aikana ei ole kovin käyttäjäystävällistä ja siitä syystä tämä virhe näytetään vasta kun lomakkeella ei ole muita virheitä.

Muu-tyyppinen manuaalinen kirjanpitovienti ei välttämättä liity millekään sopimukselle. Tällainen käyttötapaus voi olla esimerkiksi ALV-tilien nollaus. Tätä varten sopimuskenttään tehtyä validaatiota tuli muuttaa niin, että jos kaikki kirjanpidon tositteen kirjanpitoviennit ovat muu-tyyppisiä, sopimuskenttä ei ole pakollinen. Mikäli kirjanpitovienneistä löytyy yksikin tapahtuma, joka ei ole tyyppiä muu, tulee sopimus olla lisätty tositteelle.

Lomakkeen oltava täytetty täysin oikein ennen kuin lähetä-painike aktivoituu. Kun Kirjanpidon tositteelle lisätään uusia yksittäisiä kirjanpitovientejä, lähetä-painike menee jälleen pois käytöstä, kunnes myös uuden viennin kentät on täytetty tai jos vienti poistetaan.

Kuva 28. Ilmoitus yhteenlaskettujen summien virheestä näkyy kirjanpidon tositteen yläreunassa.



Käyttäjäystävällisyyttä lisättiin tällä tehtävällä myös tavallisen manuaalisen kirjanpitoviennin osalta ottamalla tapahtumalajikoodi-kenttä pois käytöstä siihen asti, että kirjaustyyppi on valittu. Näin ollen käyttäjän on ensin valittava sopimus, jolle kirjanpitovienti liittyy. Sopimuksen perusteella haetaan kirjaustyyppi-vaihtoehdot, ja tämä kenttä aktivoituu. Kirjaustyypin perusteella haetaan tapahtumalajikoodit, ja tämä kenttä aktivoituu tapahtumatyypin valinnan jälkeen. Aikaisemmin molemmat kentät olivat riippuvaisia sopimusnumerovalinnasta, ja mikäli käyttäjä olisi klikannut tapahtumalajikoodi-kentän auki ennen kirjaustyypin valitsemista, olisi hänelle näkynyt tyhjä alasvetovalikko, mikä saattaisi aiheuttaa hämmennystä. Myös yksittäisen manuaalisen kirjanpitoviennin poistaminen

estettiin siinä tapauksessa, jos lomakkeella on vain yksi kirjanpitovienti. Kirjanpidon tositetta ei voida tehdä ilman ainuttakaan kirjanpitovientiä.

5.2.10 Yhteenvetotaulukko ja palvelinpuolen muutokset

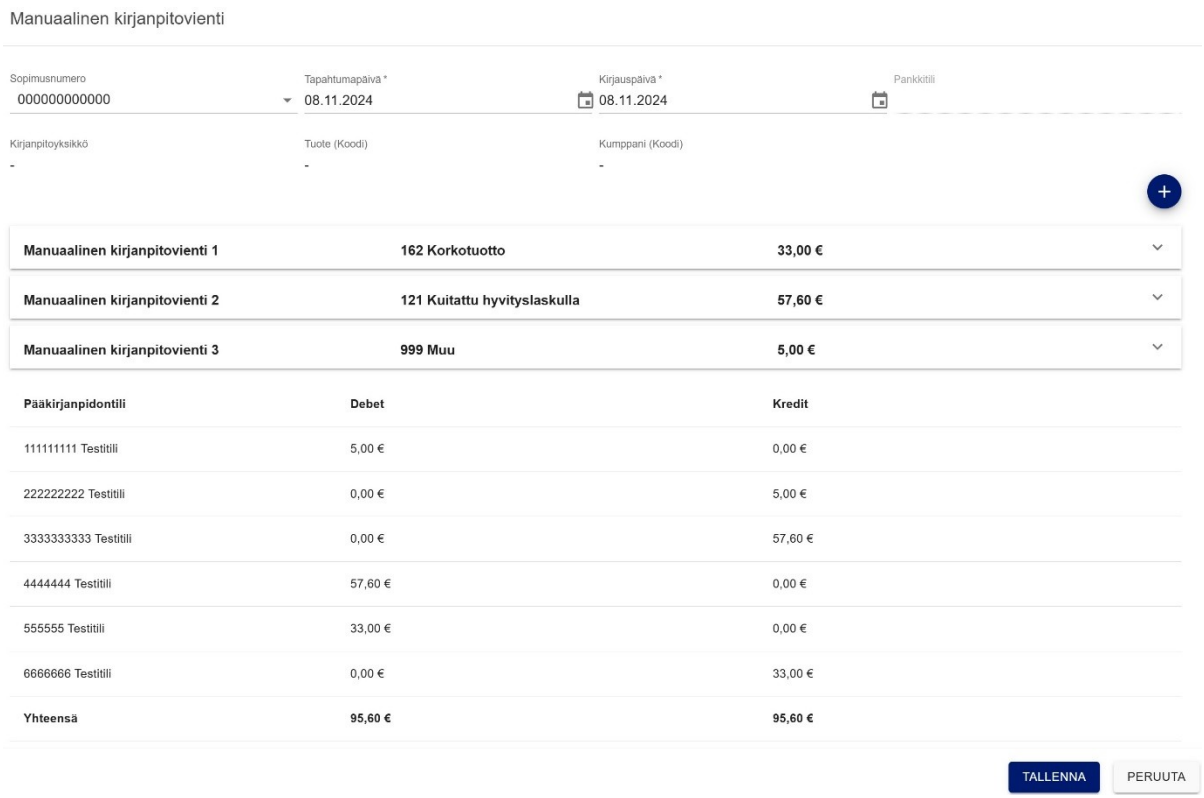
Samaan aikaan muu-tyyppisen kirjanpitoviennin toteutuksen kanssa kehitystiimissä työstettiin yhteenvetonäkymää manuaalisille kirjanpitovienneille. Yhteenvetotaulukossa lasketaan debet ja kredit -vientien summat sekä listataan pääkirjatilit taulukkoon. Toteutus ehti valmistua ennen muu-tyyppisen kirjanpitoviennin valmistumista ja se liitettiin develop-haarasta osaksi muu-tyyppisen kirjanpitoviennin kehityshaaraa.

Ohjelmistokehitys-työskentelyssä tehtävät valmistuvat aina hieman eri aikoihin, ja kehittäjät liittävät valmiita ja testattuja osioita osaksi develop-haaraa aina kun jotain saadaan valmiiksi. Develop-haaran seuraaminen on siksi tärkeää kehitystyötä tehdessä, ja develop-haaraan tulleet uudet muutokset on hyvä liittää osaksi omaa kehityshaaraa git merge -komennolla, jotta työskennellään aina uusimman version parissa ja vältytään konflikteilta.

Taulukko ei liittämisvaiheessa ottanut huomioon muu-tyyppisiä kirjanpitovientejä, joissa rahamäärä asetetaan viennille hieman eri tavalla. Tehtävällä muokattiin siis yhteenvetotaulukko tukemaan muu-tyyppisiä kirjanpitovientejä niin että summat näkyvät oikein kaikilla käyttötapauksilla. Kuvassa 29 on esitetty toteutukseen liitetty yhteenvetotaulukko, jossa on laskettu kaikentyyppisiä kirjanpitovientejä yhteen. Yhteenvetotaulukon lisääminen manuaalisen kirjanpitovientitapahtuman modaaliin helpottaa käyttäjää seuraamaan debet- ja kredit -vientien summia, ja tarkistamaan että summat täsmäävät.

Palvelinpuolelle ei tarvinnut enää tässä vaiheessa toteutusta tehdä suuria muutoksia. Evitec Power Lendingin kirjanpitojärjestelmä ei aikaisemmin tukenut muu-tyyppistä tapahtumaa, jolla ei ole mitään valmiita kirjaussääntöjä. Tapahtumatyyppi muu ja sitä vastaava tapahtumalajikoodi tuli lisätä tietokantaan ja joitain olemassa olevia palvelinpuolen funktioita ja validointeja tuli sen myötä hieman muuttaa. Muu-tyyppisen kirjauksen tiedot tallennettiin käyttäjän syötteestä, ja viennille tallennettiin myös boolean-tyyppinen tieto muu-tyyppisestä viennistä, jotta tapahtumia voidaan hakea ja rajata helpommin. Myös yksikkötestit tuli korjata kattamaan tehdyt muutokset.

Kuva 29. Yhteenvetotaulukko liitetty osaksi manuaalisia kirjanpitovientejä.

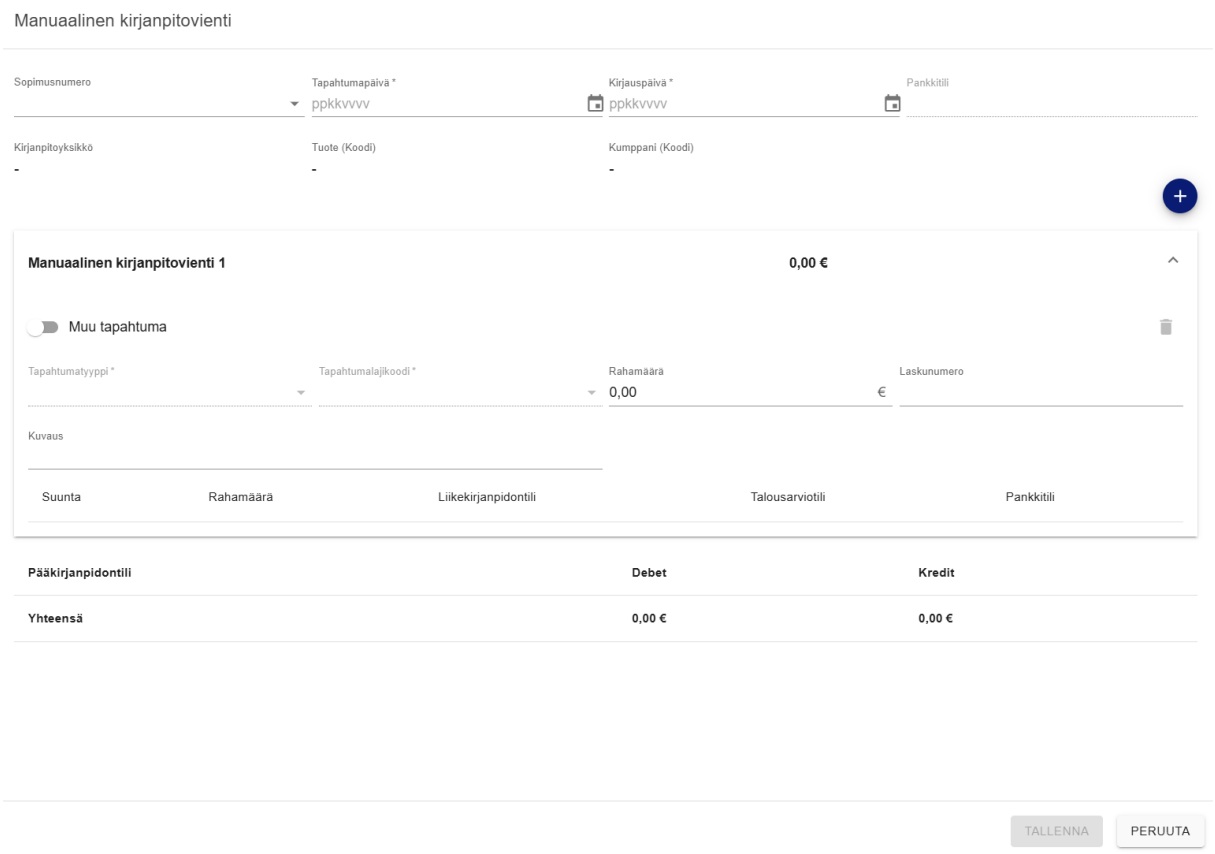


5.2.11 Bugi-korjauksia ja katselmoinnissa esiin nousseita huomioita

Muu-tyyppisen kirjanpitoviennin tehtävän viimeistelyvaiheessa löytyi vielä joitain bugeja, jotka olivat korjattava ennen toteutuksen esittämistä sidosryhmille. Havaittiin esimerkiksi, että palvelinpuolen toteutus hajoaa, jos kirjanpidon tositteella lähettää ensin normaalin kirjanpitoviennin ja sitten muu-tyyppisen kirjanpitoviennin. Toisinpäin asetellessa tapahtuma meni onnistuneesti läpi. Bugi saatiin korjattua muuttamalla tapaa, jolla palvelinpuolelle luotu funktio käsitteli muu-tyyppistä kirjanpitovientiä.

Lisäksi havaittiin, että mikäli käyttäjä poistaa kirjanpitomoduulista kaikki kirjanpitovientilomakkeet ja yrittää sen jälkeen lisätä uuden lomakkeen pluspainikkeesta, hajosi käyttöliittymä. Tämä bugi johtui vääränlaisesta tavasta käsitellä filter-funktiota ja React Final Formin form.mutator-funktiota. Bugi saatiin korjattua, ja samalla päätettiin estää lomakkeen poistaminen, mikäli se on kirjanpidon tositteen ainut kirjanpitovienti. Kirjanpidon tositetta ei voida luoda ilman että mukana olisi ainuttakaan kirjanpitovientiä, joten ei ole mitään syytä miksi käyttäjän täytyisi pystyä poistamaan ainut kirjanpitoviennin lomake. Kuvassa 30 nähdään, että kirjanpitovientien lomakkeita on modaalissa vain yksi, ja roskakori-ikoni näkyy harmaana, jolloin käyttäjä ei pysty sitä poistamaan.

Kuva 30. Kirjanpitovientiä ei voi poistaa sen ollessa modaalin ainoa vientitapahtuma.



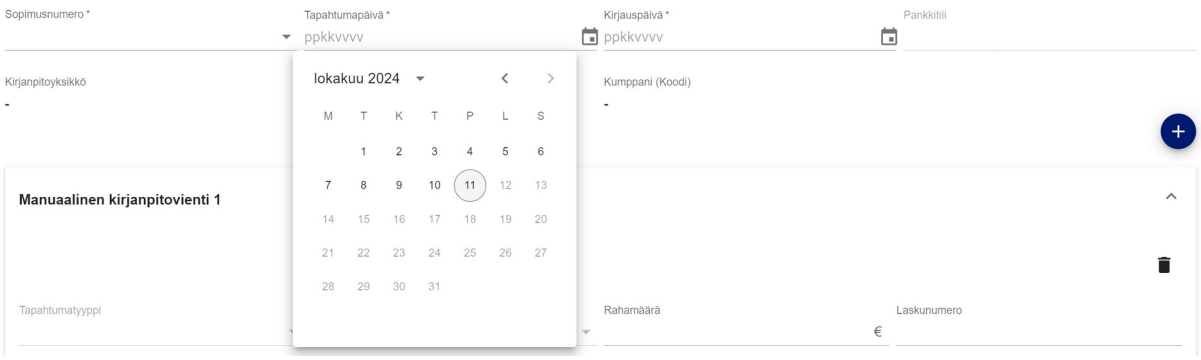
Lomakkeen oikeanlainen validoiminen osoittautui myös odotettua työläämmäksi. Logiikka tallenna-painikkeen aktivoitumisen takana yksinkertaistettuna oli se, että painike ei aktivoidu ennen kuin kaikki tiedot manuaalisen kirjanpitoviennin modaalissa on täytetty oikein. Välillä kuitenkin tallenna-painike aktivoitui, vaikka tiedot olivat puutteellisia. Esimerkiksi muu tapahtuma -painikkeen klikkailu sai tallenna-painikkeen aktivoitumaan silloin kun ei pitäisi, ja toisinaan painike ei ottanut huomioon tiettyjä pakollisia kenttiä. Muun muassa validoitavien kenttien järjestys vaikutti validaation toimintalogiikkaan, ja vertailuoperaattoreiden käytössä tuli olla tarkkana. Lopulta validaatio saatiin kuitenkin toimimaan halutulla tavalla.

5.2.12 Toteutuksen esittäminen sidosryhmille

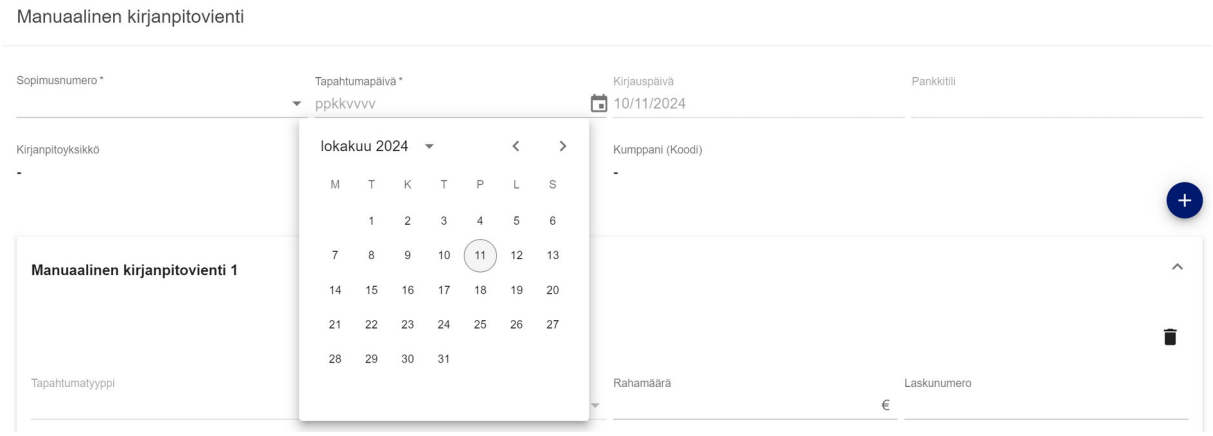
Manuaalisten kirjanpitovientien toteutusta demonstroitiin sidosryhmille kahteen kertaan Evitec Power Lending -tiimin yhteisessä, sprintin päättävässä demossa. Ensimmäisessä demossa näytettiin toteutusta ennen muu-tyyppisen tapahtuman lisäämistä, ja toinen demo piti sisällään muu-tyyppisen tapahtuman lisäämisen sekä yhteenvetotaulukon esittelyn.

Kumpikin demotilaisuus eteni hyvin ja toteutus vastasi pitkälti sidosryhmien odotuksia. Ensimmäisen demon pohjalta huomio kiinnittyi kirjanpidon tositteen kenttiin tapahtumapäivä ja kirjauspäivä, joiden valinta oli rajattu kuluvaan päivään kuten kuvassa 31 on esitetty. Demossa nostettiin esiin, että päivämäärä pitäisi mahdollisesti olla asetettavissa myös tulevaisuuteen, ja huomio korjattiin seuraavaan demoon mennessä. Korjaus on esitetty kuvassa 32.

Kuva 31. Päivämääräkenttä rajoittaa päivän valitsemisen nykyiseen päivämäärään toteutuksen ensimmäisessä versiossa.



Kuva 32. Päivämääräkentän validointia muokattiin niin, että päivämäärä voidaan asettaa myös tulevaisuuteen.



Toteutuksen kehittämistä jatketaan seuraavaksi lisäämällä neljän silmän tarkastusperiaate, jossa toinen käyttäjä tarkistaa tehdyn kirjauksen. Lisäksi uuden ominaisuuden käyttöönottamisen myötä voidaan huomata uusia kehitystarpeita tai käyttöliittymäparannuksia. Kehitettyä ominaisuutta ylläpidetään ja parannellaan sitä mukaa, kun uusia huomioita syntyy. Tämä kuuluu perinteiseen ohjelmistokehityksen elinkaareen.

6 Pohdinta

Opinnäytetyön tuloksena syntyi manuaaliset kirjanpitoviennit mahdollistava lisämoduuli Evitec Power Lending -järjestelmään. Moduulissa on mahdollista luoda uusia kirjanpitotapahtumia olemassa olevien kirjaussääntöjen mukaan tai luoda täysin manuaalisesti muu-tyyppisiä kirjanpitovientejä. Manuaalisten kirjanpitovientien modaalissa yhdellä kirjanpidon tositteella voidaan lisätä rajoittamaton määrä kirjanpitovientejä, jotka liittyvät samalle sopimukselle tai tehdä muu-tyyppisiä manuaalisia kirjanpitovientejä ilman liittyvää sopimusta. Modaaliin voidaan lisätä uusia kirjanpitovienti-lomakkeita ja aikaisempia voidaan poistaa niin kauan, että jäljellä on enää yksi lomake. Lisäksi toteutettiin ominaisuus, jonka avulla voidaan säädellä modaalissa näytettäviä kenttiä eri sidosryhmien välillä.

Manuaalisten kirjanpitovientien modaaliin lisätyt kentät on validoitu niin, että ne ottavat vastaan vain oikeantyyppistä tietoa. Kenttien täyttäminen on estetty, mikäli aikaisemmin vaadittuja kenttiä ei ole täytetty oikein. Tällä pyritään ohjaamaan käyttäjää täyttämään kentät oikeassa järjestyksessä. Pakolliset kentät on merkitty tähtimerkillä ja lisäksi punainen huomioteksti ilmoittaa, mikäli kentät jäävät tyhjiksi. Koko lomake on validoitu niin, että kaikkien kirjanpitovientien debet- ja kreditsummien on täsmättävä. Lomaketta ei voida lähettää, mikäli kaikkia tietoja ei ole täytetty oikein. Yhteenveto-taulukosta nähdään, mille tileille kirjauksia on tehty, sekä debet- ja kreditvientien yhteenlasketut summat.

Palvelinpuolelle toteutettiin uusi rajapinta, joka ottaa vastaan manuaalisia kirjanpitovientejä ja tallentaa syötetyt tiedot SQL Server -tietokantaan. Lisäksi olemassa olevaa toteutusta tuli muuttaa, jotta myös muu-tyyppisiä kirjanpitovientejä on mahdollista lähettää järjestelmään. Palvelinpuolelle luoduille uusille funktioille tehtiin testit xUnit kirjastoa hyödyntäen ja olemassa olevia testejä muokattiin niiltä osin, missä valmiiseen toteutukseen tehtiin muutoksia.

Manuaalisten kirjanpitovientien moduulia työstettiin osana ketterää järjestelmäkehitystiimiä, jolloin tiimin jäseniltä saatiin palautetta toteutuksesta ennen sen esittämistä sidosryhmille. Ennen koodin liittämistä develop-haaraan vähintään kahden muun tiimin jäsenen tuli katselmoida toteutusta, jolloin koodista havaittiin joitain bugeja, jotka korjattiin ennen koodin liittämistä tuotantoversioon. Myös käytettävyyden parantamiseksi saatiin ehdotuksia

tiimin jäseniltä. Pull requestin yhteydessä automaatio teki tarkastuksen koodin laadusta, ja sitä kautta nousseet huomiot korjattiin.

Manuaalisten kirjanpitovientien kehittäminen jatkuu vielä opinnäytetyön jälkeen. Seuraavia kehitysaskelia manuaalisten kirjanpitovientien toteutuksessa ovat tarkistusominaisuuden luominen, jota kutsutaan neljän silmän periaatteeksi. Tämä tarkoittaa sitä, että toisen henkilön on tarkistettava manuaalinen kirjanpitovienti ennen kuin se on todella lisätty järjestelmään. Tällä tavalla pyritään ehkäisemään inhimillisiä virheitä ja toisaalta myös väärinkäytöksiä. Tarkistuksen luomisesta puhuttiin jo opinnäytetyön suunnitteluvaiheessa, mutta toteutusta ei ehditty aloittamaan opinnäytetyön aikaikkunassa. Lisäksi uuden manuaalisten kirjanpitovientien käyttöönottamisen myötä voidaan huomata uusia kehitystarpeita tai käyttöliittymäparannuksia.

Tuotekehityksen parissa työskennellessä on tyypillistä, että jotkut asiat vaativat uudelleen määrittelyä ja tarkistamista, ja tällaiset toimenpiteet saattavat viedä paljonkin aikaa. Myös prioriteettilista elää projektien aikana, ja tiimin on reagoitava prioriteettilistan kärjessä oleviin asioihin. Myös uusia toiveita ja kehityskohteita herää usein kehitystyön aikana, kun kokonaisuus lähtee pikkuhiljaa hahmottumaan. Myös opinnäytetyön kohdalla sidosryhmiltä nousi esiin toiveita, joista ei vielä ensimmäisissä suunnittelupalavereissa oltu nostettu esiin. Kehitettyä ominaisuutta ylläpidetään ja parannellaan sitä mukaa, kun uusia huomioita syntyy. Tämä kuuluu perinteiseen ohjelmistokehityksen elinkaareen.

Opinnäytetyön parissa työskentely kehitti valtavasti ongelmanratkaisukykyjä, ohjelmointitaitoja ja se lisäsi asiantuntemusta kehitettävästä tuotteesta kokonaisuutena. Ennen opinnäytetyön aloittamista Reactin tilojen hallinta Reduxin avulla oli melko vieras konsepti, mutta opinnäytetyön myötä asiaan tuli perehdyttyä paljon. Myös Reactin Final Form -kirjasto tuli opinnäytetyön aikana hyvin tutuksi. Molempien kirjastojen käyttö Evitec Power Lending -tuotteen kehityksessä tulee jatkossa olemaan huomattavasti helpompaa. Sovelluksen monipuolinen testaus bugien varalta on tullut tutuksi, ja opinnäytetyö opetti, kuinka tärkeää on testata myös sellaisia toimintoja, joita ei suoraan odota käyttäjän tekevän.

Erityismainintana haluan nostaa tiimityöskentelytaitojen kehittymisen. Opinnäytetyön alussa koin haastavaksi pyytää tiimiltä apua monimutkaisissa asioissa. Vaikka manuaalisten kirjanpitovientien moduuli on itsenäinen osa, on siinä paljon riippuvuuksia aikaisemmin toteutettuihin toiminnallisuuksiin, jolloin on tärkeää ymmärtää toimintalogiikka niiden takana. Kokonaisuuden tuli jakaa dataa mutkattomasti eri repositorioiden välillä.

Esimerkiksi sopimusnumerohaku tehtiin API-kutsuna toisesta repositoriosta, mikä vaati muutoksia CORS-politiikkaan ja käyttäjäoikeuksiin. Projektin edetessä opin kuitenkin ymmärtämään, että ohjelmistokehitys on aina yhteistyötä tiimin kanssa. Näin valtavan ohjelmiston kehityksessä kukaan ei tunne koko sovellusta läpikotaisin, ja kokeneemmankin kehittäjän on välillä käännyttävä tiimin puoleen. Ohjelmistokehitys ei ole yksilösuoritusta.

Kokonaisuutena arvioisin opinnäytetyön onnistuneen hyvin. Valmistunut toteutus mahdollistaa manuaalisen kirjanpidon tositteen lisäämisen järjestelmään noudattaen kirjaussääntöjä. Kirjanpitovientejä voidaan lisätä ja poistaa, ja toteutus kokoaa yhteen kirjanpidon tositteen tärkeimmät tiedot. Toteutuksessa on käytetty aikaisemmin luotuja ominaisuuksia siellä missä se on ollut mahdollista, mikä tekee siitä yhtenäisen osan Evitec Power Lending -järjestelmää kokonaisuutena. Manuaalisten kirjanpitovientien toteutus jatkuu vielä opinnäytetyön jälkeenkin, mutta opinnäytetyön aikana saatiin toteutukselle tehtyä hyvä ja toimiva pohja.

Lähteet

.NET Trainer & Instructor. (3.3.2023). *History of ASP.NET and ASP.NET Core for Beginners.* https://www.codewithabhishekluv.com/blog/2023/03/03/history-of-asp-net-and-asp-net-core-for-beginners

Atlassian. (n.d.). Kehityshaarojen käyttö järjestelmäkehitystiimissä [kuva]. Haettu 13.10.2024. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Buddha, R. (29.8.2024). *React vs Vue: Comparison of the Best JavaScript Frameworks at a Glance!* https://radixweb.com/blog/react-vs-vue

Camus, A. (18.5.2022). *Introduction to ReactJS: A Guide for Beginners.* https://www.microverse.org/blog/introduction-to-reactjs-a-guide-for-beginners

Emadamerho-Atori, N. (23.11.2023). *Vue vs. React: How to Choose for 2024*. https://prismic.io/blog/vue-vs-react

Evitec Solutions. (n.d.). *Evitec Solutions Power Lending.* Haettu 5.10.2024. https://evitec.com/fi/ratkaisut/evitec-solutions-power-lending/

Finanssivalvonta. (12.9.2018). *Valvonta- ja toimenpidemaksut*. https://www.finanssivalvonta.fi/finanssisektorin-toimijalle/pankki/valvonta--ja-toimenpidemaksut/

Fork. (n.d.). *Fork.* Haettu 20.10.2024. https://fork.dev/

Fork. (n.d.). Yleiskatsaus Forkin käyttöliittymästä [kuva]. Haettu 20.10.2024. https://fork.dev/

Hakulinen, J. (2024). Käyttöliittymäsuunnittelijan Figma-kuva manuaalisen kirjanpidon tositteen modaalista [kuva].

Hakulinen, J. (2024). Käyttöliittymäsuunnittelijan Figma-kuva muu-tyyppisenkirjanpitoviennin lisäämiselle [kuva].

Hakulinen, J. (2024). Käyttöliittymäsuunnittelijan Figma-kuva yhteenvetotaulukosta [kuva].

Material UI. (n. d.). *Material UI – Overview*. Haettu 3.10.2024. https://mui.com/material-ui/getting-started/

Microsoft. (25.5.2021). *Entity Framework Core.* https://learn.microsoft.com/en-us/ef/core/

Microsoft. (10.5.2022a). *Create Data Transfer Objects (DTOs)*. https://learn.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5

Microsoft. (11.7.2022). *Understanding Models, Views and Controllers (C#).* https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/understanding-models-views-and-controllers-cs

Microsoft. (31.10.2023a). *Dependency injection into controllers in ASP.NET Core.* https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/dependency-injection?view=aspnetcore-7.0

Microsoft. (12.10.2023b). *Asynchronous programming scenarios.* https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/async-scenarios

Microsoft. (14.2.2024a). *What is SQL Server?* https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16

Microsoft. (30.7.2024c). *Interface (C# Reference).* https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface

Microsoft. (23.9.2024b). *SQL tools overview*. https://learn.microsoft.com/en-us/sql/tools/overview-sql-tools?view=sql-server-ver16

Microsoft. (8.10.2024d). .NET and .NET Core Support Policy. https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core

Microsoft. (19.6.2024e). *What is Visual Studio?* https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022

Patterns. (n.d.). *Overview of React.js*. https://www.patterns.dev/react/

Rani, P. (7.9.2023). Sprintin sykli kaavion avulla kuvattuna [kuva]. https://www.linkedin.com/pulse/day-2-agile-manifesto-pooja-rani/

Ravikiran, A. S. (24.10.2024). Differentiating SQL and MySQL: A Comprehensive Guide. https://www.simplilearn.com/tutorials/sql-tutorial/difference-between-sql-and-mysql

React Final Form. (n.d.-d). *API / <Field/>*. Haettu 3.10.2024 osoitteesta https://final-form.org/docs/react-final-form/api/Field

React Final Form. (n.d.-b). *API / <Form/>*. Haettu 3.10.2024 osoitteesta https://final-form.org/docs/react-final-form/api/Form

React Final Form. (n.d.-a). *Getting Started*. Haettu 3.10.2024 osoitteesta https://final-form.org/docs/react-final-form/getting-started

React Final Form. (n.d.-c). *Types / FormProps*. Haettu 3.10.2024 osoitteesta https://final-form.org/docs/react-final-form/types/FormProps

Riordan, Grant. (12.4.2024). *What is Mocking? Mocking in .NET Explained With Examples.* https://www.freecodecamp.org/news/explore-mocking-in-net/

Rouse, M. (13.2.2024). *Tietokannan hallintajärjestelmä (DBMS).* https://www.techopedia.com/fi/sanasto/tietokannan-hallintajarjestelma-dbms

Sharma, Itesh. (n.d.). *.NET Vs Java: Key Differences to Consider.* https://www.tatvasoft.com/outsourcing/2022/03/net-vs-java.html

Sharma, Sunil. (16.6.2020). *ASP.NET Core: A Server-Side Web Application Framework.* https://www.linkedin.com/pulse/aspnet-core-server-side-web-application-framework-sunil-sharma/

SingularTech. (n.d.). Azure DevOps työkalut sijoitettuna CI/CD Pipelinen ympärille [kuva]. Haettu 15.10.2024. https://singulartech.ai/en/category/software-development-en/

Stack Overflow. (2024). Vuoden 2024 suosituimmat ohjelmointi-, scriptaus- ja merkkikielet Developer Survey -kyselyssä [kuva]. https://survey.stackoverflow.co/2024/technology/

Stack Overflow. (2024). Vuoden 2024 suosituimmat tietokantatekniikat Developer Survey -kyselyssä [kuva]. https://survey.stackoverflow.co/2024/technology/

Stack Overflow. (2024). Vuoden 2024 suosituimmat web-tekniikat Developer Survey -kyselyssä [kuva]. https://survey.stackoverflow.co/2024/technology/

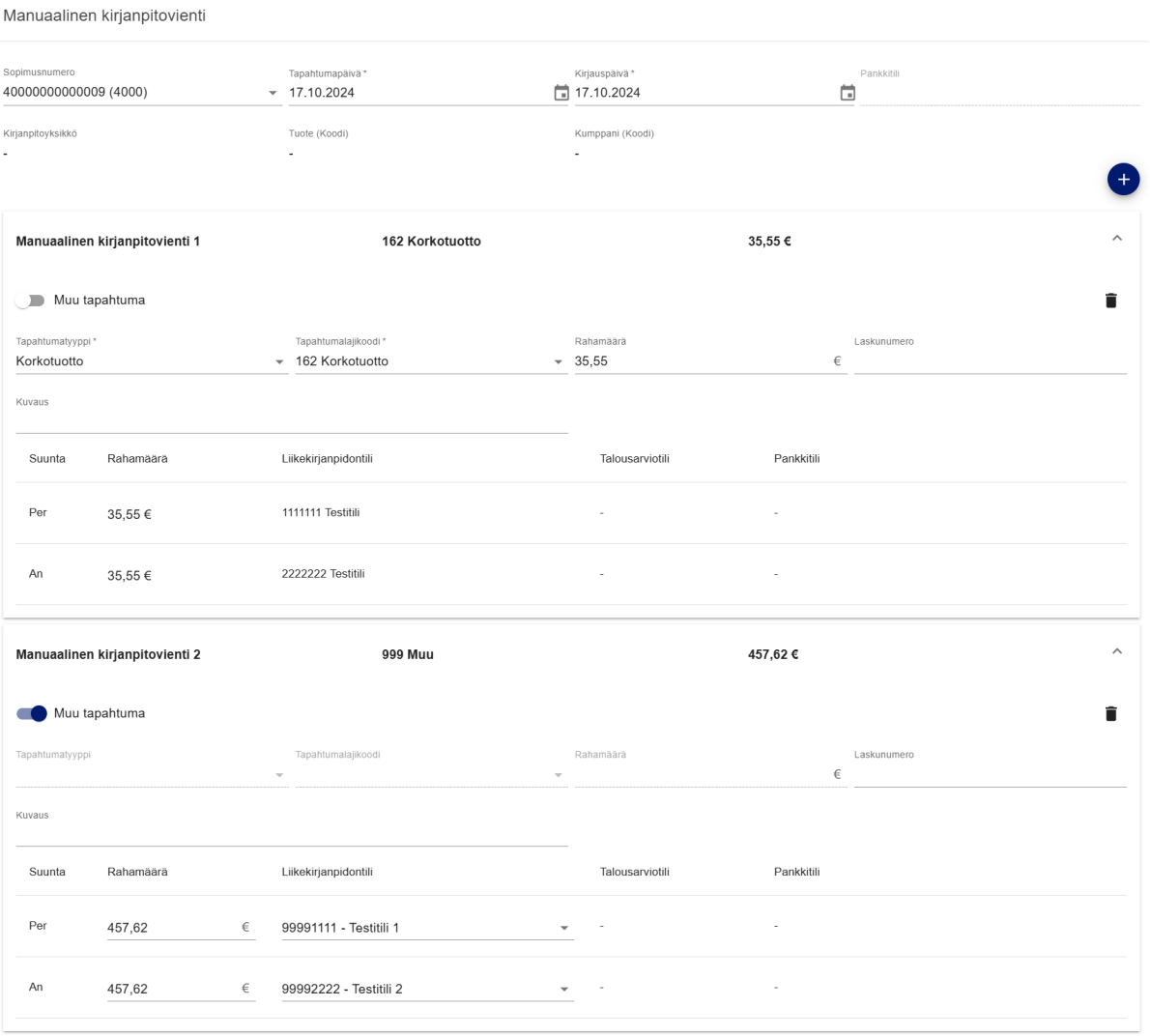
Syrjä, J. (2016). Mikropalveluarkkitehtuurit. [opinnäytetyö, Vaasan Ammattikorkeakoulu]. https://urn.fi/URN:NBN:fi:amk-201605147828

Torppa, T., Peuraniemi, T., Rapo, J. (28.8.2024). *Backround and introduction.* https://fullstackopen.com/en/part9/background_and_introduction

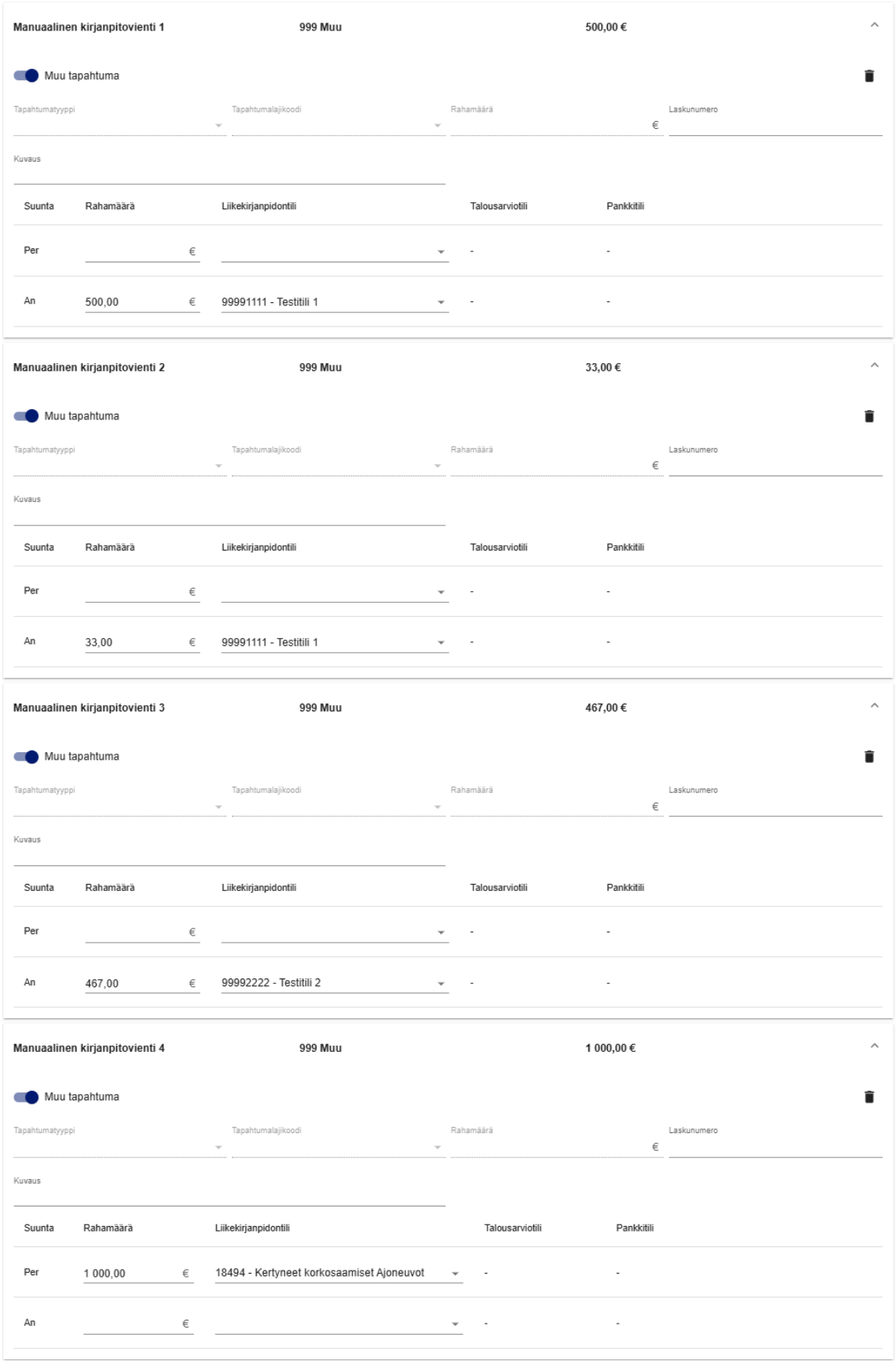
Vue.js. (n.d.). *Introduction.* Haettu 12.11.2024 osoitteesta https://vuejs.org/guide/introduction

Wanuoike, M. (9.7.2019). *How to Use Git Branches & Buddy to Organize Project Code.* https://www.sitepoint.com/use-git-branches-buddy/

Liite 1. Tavallinen manuaalinen kirjanpitovienti muutosten jälkeen ja muu-tyyppinen kirjanpitovienti.



Liite 2. Yksi debet- ja kolme kreditvientiä samalla kirjanpidon tositteella



Manuaalinen kirjanpitovienti 1 | **999 Muu** | **500,00 €**

Muu tapahtuma

Tapahtumatyyppi	Tapahtumalajikoodi	Rahamäärä	Laskunumero
		€	

Kuvaus

Suunta	Rahamäärä	Liikekirjanpidontili	Talousarviotili	Pankkitili
Per	€		-	-
An	500,00 €	99991111 - Testitili 1	-	-

Manuaalinen kirjanpitovienti 2 | **999 Muu** | **33,00 €**

Muu tapahtuma

Tapahtumatyyppi	Tapahtumalajikoodi	Rahamäärä	Laskunumero
		€	

Kuvaus

Suunta	Rahamäärä	Liikekirjanpidontili	Talousarviotili	Pankkitili
Per	€		-	-
An	33,00 €	99991111 - Testitili 1	-	-

Manuaalinen kirjanpitovienti 3 | **999 Muu** | **467,00 €**

Muu tapahtuma

Tapahtumatyyppi	Tapahtumalajikoodi	Rahamäärä	Laskunumero
		€	

Kuvaus

Suunta	Rahamäärä	Liikekirjanpidontili	Talousarviotili	Pankkitili
Per	€		-	-
An	467,00 €	99992222 - Testitili 2	-	-

Manuaalinen kirjanpitovienti 4 | **999 Muu** | **1 000,00 €**

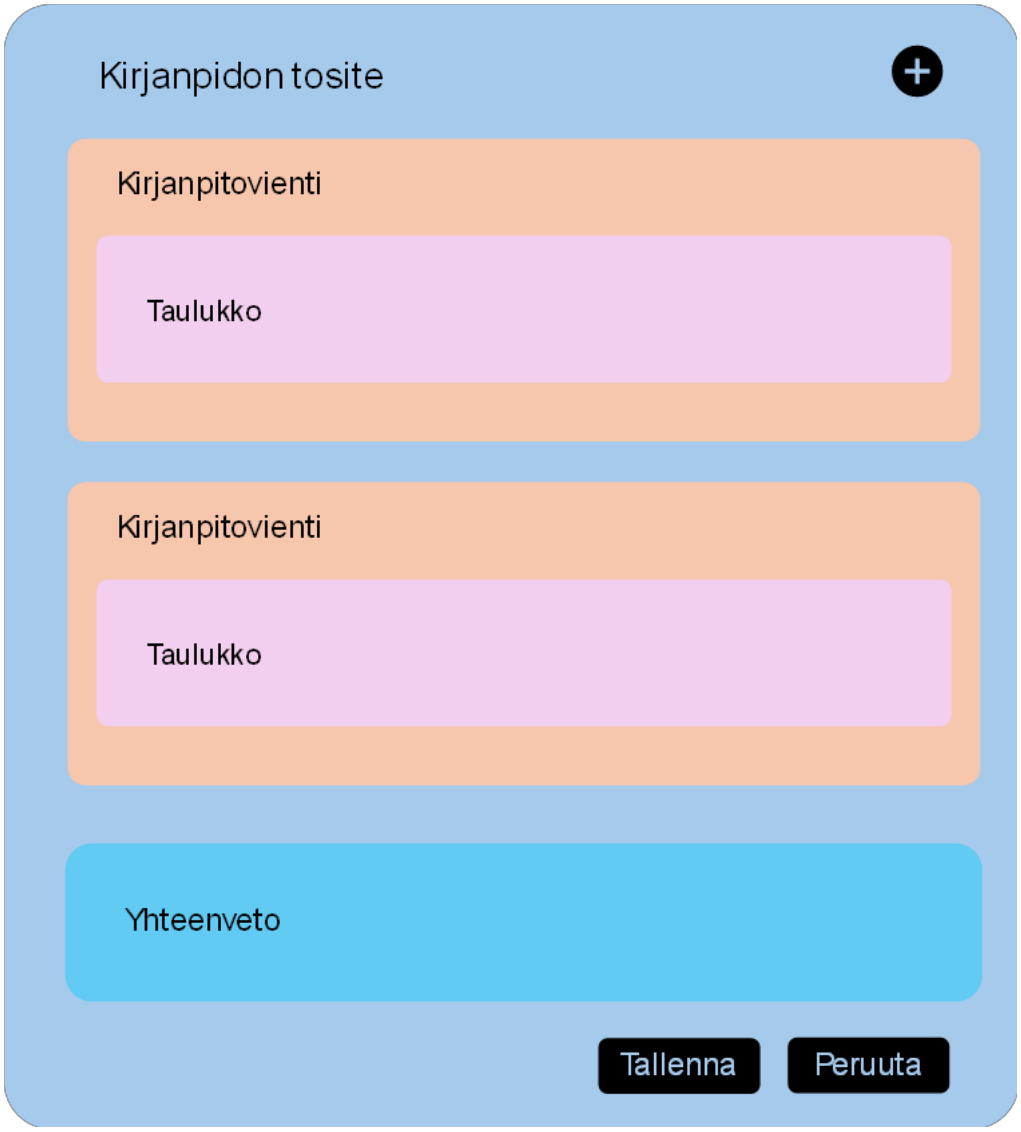
Muu tapahtuma

Tapahtumatyyppi	Tapahtumalajikoodi	Rahamäärä	Laskunumero
		€	

Kuvaus

Suunta	Rahamäärä	Liikekirjanpidontili	Talousarviotili	Pankkitili
Per	1 000,00 €	18494 - Kertyneet korkosaamiset Ajoneuvot	-	-
An	€		-	-

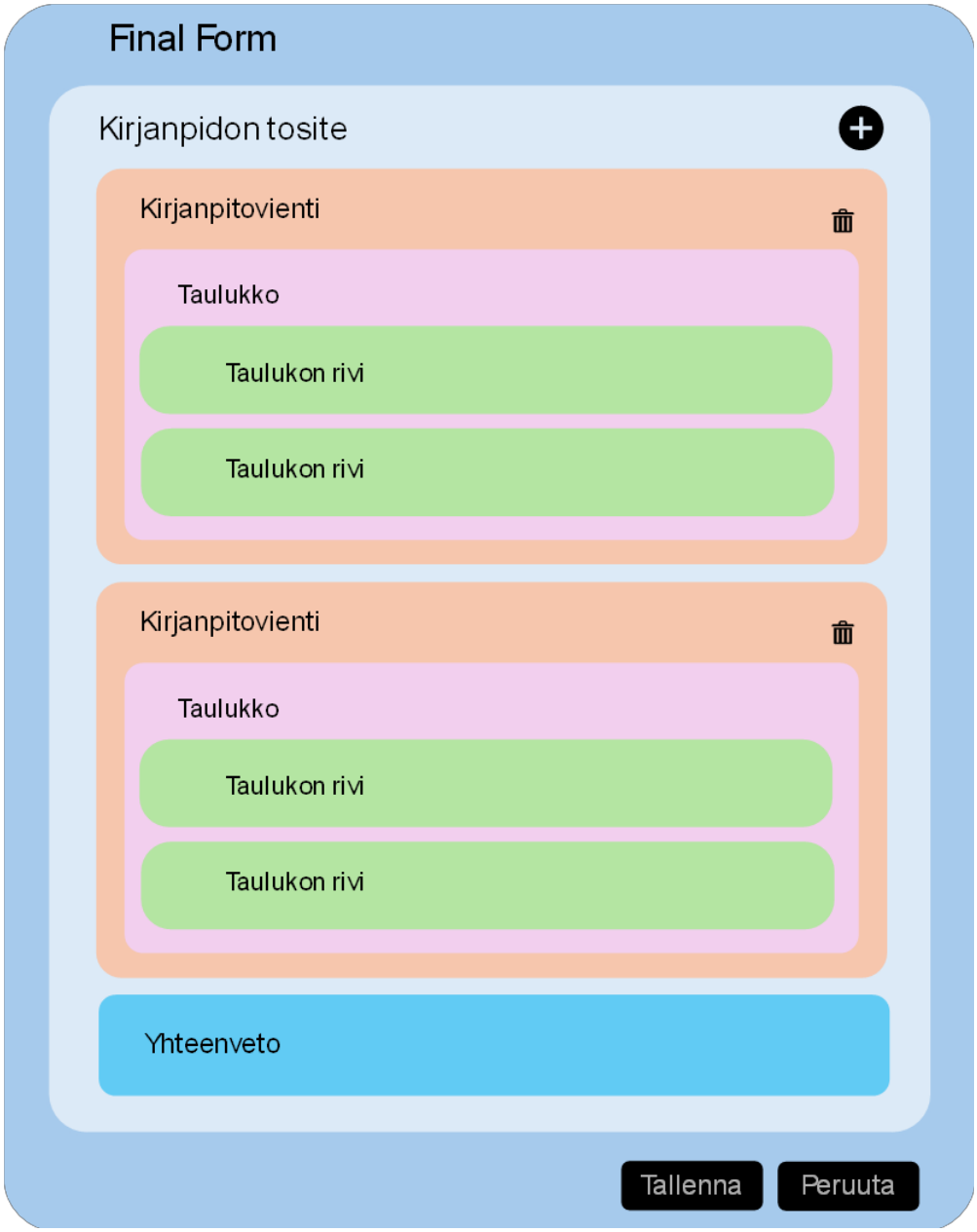
Liite 3. Komponenttirakenteen kehittyminen työn edetessä



Ensimmäisessä suunnitelmassa oli neljä pääkomponenttia:

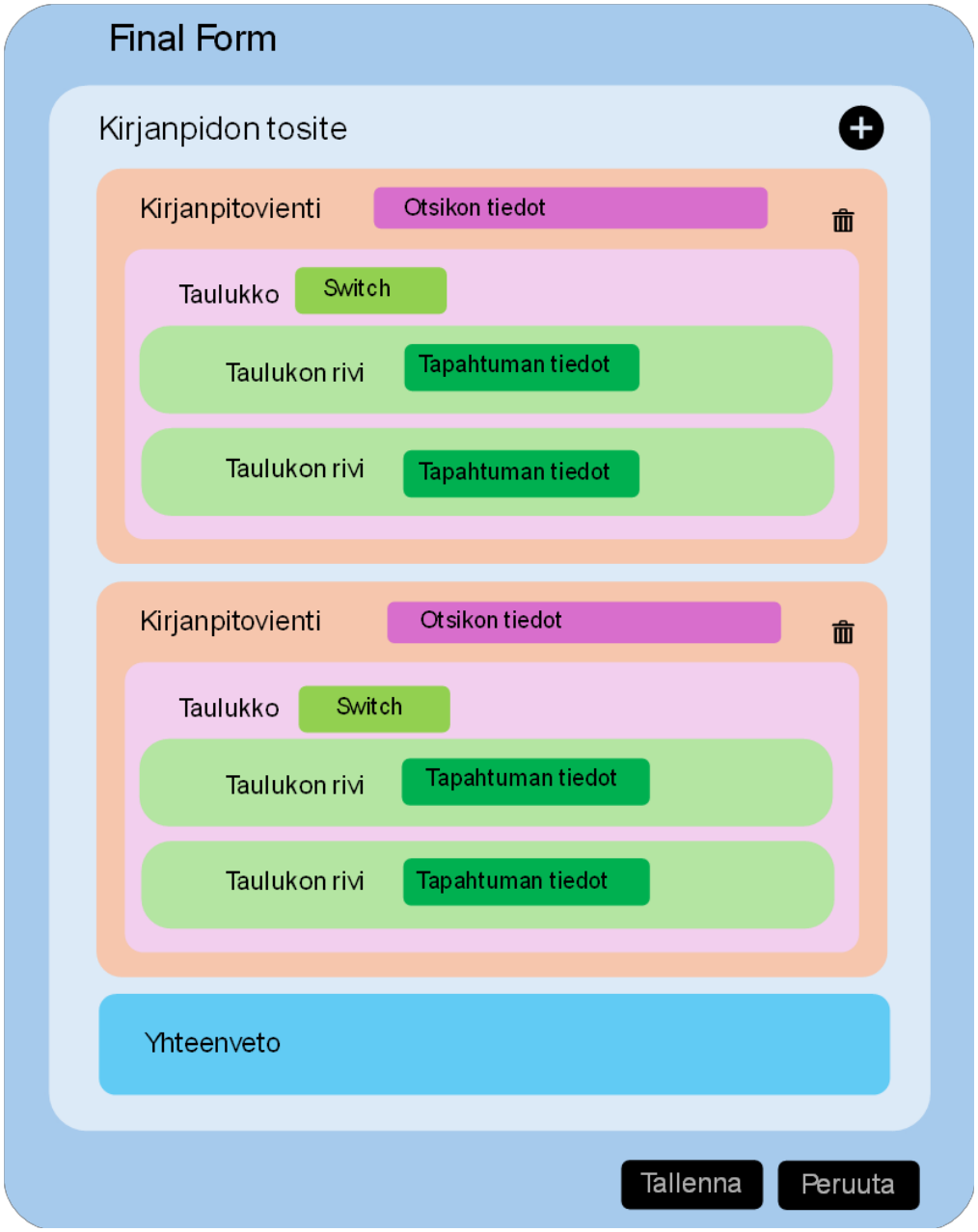
- Kirjanpidon tosite, joka pitäisi sisällään tositteelle yhteiset kentät
- Kirjanpitovienti, joka sisältää kirjanpitoviennin kentät
    - Kirjanpitoviennin sisällä taulukko kirjaussäännöille
- Yhteenveto-taulukko koko tositteen tiedoista

Lisäksi pienempiä itsenäisiä komponentteja suunnitelmassa oli pluspainike, tallenna- ja peruuta -painikkeet



Työn edetessä huomattiin, että yksittäiset komponentit alkavat paisua liian suuriksi joten totetutusta jaettiin edelleen komponentteihin.

- Koko toteutus käärittiin uuden komponentin sisään, jonne kirjattiin muun muassa React Final Form -määritykset, moduulin yleiset määritykset ja validaatio
- Kirjaussäännöistä vastaava taulukko jaettiin taulukon määrittelevään komponenttiin sekä yksittäisen rivin renderöitymisestä vastaavaan komponenttiin
- Roskakori-ikoni lisättiin omana komponenttina



Valmiissa työssä itsenäisiä komponentteja oli jo runsaasti. Uusia komponentteja muodostui seuraavista:

- Kirjanpitovienti-lomakkeen otsikko, jonne renderöidään näkyviin kirjaustyyppi, tapahtumalajikoodi ja rahamäärä
- Switch-painike, jolla asetetaan kirjaustyyppi muu
- Taulukon rivin sisällä muu-kirjaustyypille liittyvää logiikkaa