Mikko Ahola

Low-code-alustan käytön hyödyt ja haitat sovelluskehityksessä

Tietotekniikan pro gradu -tutkielma

31. maaliskuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Mikko Ahola

Yhteystiedot: mikkoahola02@gmail.com

Ohjaaja: Raino Mäkinen

Työn nimi: Low-code-alustan käytön hyödyt ja haitat sovelluskehityksessä

Title in English: The pros and cons of using a low-code platfrom in application development

Työ: Pro gradu -tutkielma

Opintosuunta: Informaatioteknologian tiedekunta

Sivumäärä: 72+0

Tiivistelmä: Tämän pro gradu -tutkielman tavoitteena oli selvittää low-code-alustan käytön tuomia hyötyjä ja mahdollisia haittoja sovelluskehitykselle. Tutkielman tavoite pyrittiin saavuttamaan vastaamalla tutkielman tutkimuskysymykseen: Mitä yhtenevyyksiä ja eroavaisuuksia tutkimuskirjallisuudessa low-code-alustoista esiintyvillä hyödyillä ja haitoilla on käytännön sovellusprojektissa havaittuihin Mendix low-code-alustan käytön hyötyihin ja haittoihin? Tutkimuskysymykseen vastattiin toteuttamalla kaksiosainen tutkielma, jonka osien tuloksia verrattiin toisiinsa. Tutkielman ensimmäinen osa on kirjallisuuskatsaus, jolla pyrittiin selvittämään tutkimuskirjallisuudessa esiintyviä low-coden ja low-code-alustojen käyttämisen hyötyjä ja haittoja. Tutkielman toinen osa on vanhan varastonhallintasovelluksen yhden päätoiminnallisuuden uudelleentoteutus Mendix low-code-alustaa käyttäen. Uudesta sovelluksesta toteutettiin MVP-versio natiivina mobiilisovelluksena. Tutkielmassa selvisi, että low-code-alustojen käyttäminen sovelluskehityksessä voi tuoda hyötyjä, kuten kehitysnopeuden kasvua, jos low-code-alustaa käytetään kohdealuella, jota varten low-code-alusta on suunniteltu ja sen vahvuuksia mukaillaan. Low-code-alustan käyttäminen väärällä kohdealueella voi kuitenkin johtaa vastakkaisiin tuloksiin.

Avainsanat: Low-code, Mobiilikehitys, Mendix, Agile, Scrum, MDD/MDE, RAD, CASE, Microsoft Power Platform, SalesForce, OutSystems,

Abstract: The purpose of this theses was to examine what pros and cons come from using

a low-code platforms in application development. The purpose of the thesis was meant to be achieved by answering the research quesetion: What similarities and differences about the pros and cons of development could be found between research literature and the application project using Mendix low-code platform? The research question was answered by conducting a thesis which consist of two parts. The first part was a literature review where the aim was to find the pros and cons of using low-code platforms from the research literature. The purpose of the second part was to collect physical experiences of the pros and cons of using Mendix low-code platform in a software development project. In the second part a part of a storage hall management application was rebuilt using Mendix low-code platform. The newly built application was meant to be an MVP version of the old application and to only contain a single major use case of the apllication. In this thesis it was found that the usage of a low-code platform could produce positive results when used in right domain and is used making use of its strengths. Although the results of using a low-code platform could be quite opposite if a low-code platform wouldn't be used in a domain where its strengths lie.

Keywords: Low-code, Mobile development, Mendix, Agile, Scrum, MDD/MDE, RAD, CASE, Microsoft Power Platform, SalesForce, OutSystems

Termiluettelo

DSL	Domain Specific Language on visuaalinen tai tekstuaalinen kieli, joka erikoistuu ratkaisemaan tietyn kohdealueen ongelmia
MDD/MDE	Model Driven Development/Engineering eli malliohjattu kehitys ottaa kantaa sovelluskehityksen paradigmoihin käyttämällä erilaisia mallinnettuja prosesseja eli malleja sovellukehityksen elinkaaren eri vaiheiden toteuttamiseksi.
RAD	Rapid Application Development on 1990-luvulla esiin noussut vahvaan iteratiiviseen prototyypitykseen erikoistunut metodologia.
CASE	Computer-Aided Software Engineering on 1980-luvulla esiin noussut kokoelma automaatiotyökaluja ja -metodeita, jotka avustavat kehittäjiä sovelluksen eri kehitysvaiheessa.
MVP	Minimal Viable Product on sovellusprojekteissa usein käytetty termi vähimmäisen toiminnallisuuden sisältävästä tuotteesta, jota sen käyttäjä voi kuitenkin käyttää.

Kuviot

Taulukot

Sisällys

1 Johdanto

Tässä tutkielmassa selvitetään low-coden ja low-code-alustojen potentiaalia sovelluskehityksessä. Low-code on mainittu terminä kirjallisuudessa ensimmäisen kerran 2014 (Sanchis ym. 2019), mutta se alkoi näkyä vertaisarvioiduissa teksteissä vasta vuonna 2018 (Bucaioni, Cicchetti ja Ciccozzi 2022). Monilla eri tahoilla on omat määritelmänsä siitä, mitä low-code on, mutta itse termin määritelmästä ei vaikuttaisi olevan vielä tieteellistä konsensusta. Low-codea on kritisoitu terminä, sillä sitä käytetään epäjohdonmukaisesti eri lähteissä (Bock ja Frank 2021). Vaikka low-coden määritelmä ei ole vielä vakiintunut tiedeyhteisöön, low-code-alustoille on muodostunut määritelmä. Low-code-alustojen tarkoituksena on vähentää järjestelmän kehittämiseen vaaditun koodin määrää abstraktion avulla (Vincent ym. 2022).

Tutkielman tavoitteena on selvittää low-code-alustan käytön tuomia hyötyjä ja mahdollisia haittoja sovelluskehitykselle. Tutkielman tavoite pyritään saavuttamaan vastaamalla tutkielman tutkimuskysymykseen: Mitä yhtenevyyksiä ja eroavaisuuksia tutkimuskirjallisuudessa low-code-alustoista esiintyvillä hyödyillä ja haitoilla on käytännön sovellusprojektissa havaittuihin Mendix low-code-alustan käytön hyötyihin ja haittoihin? Tutkimuskysymykseen pyritään vastaamaan toteuttamalla kaksiosainen tutkielma, jonka osien tuloksia verrataan toisiinsa.

Tutkielma koostuu teoreettisesta kirjallisuuskatsauksesta ja konstruktiivisesta osasta. Tutkielman teoreettinen osa on kirjallisuuskatsaus low-code:sta, siihen liittyvistä käsitteistä ja tutkielman konstruktivistisessa osassa käytettävästä Mendix low-code-sovellusalustasta. Kirjallisuuskatsauksella pyritään selvittämään, mitä asioita tutkimuskirjallisuudessa pidetään low-code-alustojen hyötyjä ja haittoina. Tutkielman konstruktiivisessa osassa dokumentoidaan Mendix low-code-sovellusalustalla toteutetun mobiilisovelluksen MVP-version kehitysprosessi. Dokumentaatiota analysoimalla pyritään löytämään Mendix low-code-alustan käytön hyötyjä ja haittoja. Mobiilisovelluksen MVP-versio on tuotettu yrityksen käyttöön.

Tutkielman tuloksissa verrataan kirjallisuuskatsauksen tuloksia sovellusprojektin kehitysprosessin aikana tehtyihin havaintoihin. Sovellusprojektin kehitysprosessia analysoidaan tarkastelemalla kehitystiimin päivittäisten tapaamisten dokumentaatiota, viikoittaisten asiakas-

tapaamisten dokumentaatiota sekä kehitystiimin kokemuksia kehitysprosessista. Kirjallisuuskatsauksen tuloksia ja sovellusprojektin kehitysprosessia vertaamalla saadaan tuotettua informaatiota siitä, mitä hyötyjä ja mahdollisia haittoja low-coden käytöllä sovellusprojekteissa voi olla.

2 Kirjallisuuskatsaus

Tässä kirjallisuuskatsauksessa tullaan esittämään koonti low-codea koskevan tärkeimmän kirjallisuuden sisällöstä. Kirjallisuuskatsauksen tavoitteena on vastata tutkimuskysymyksen teoreettiseen osaan low-code-alustojen hyödyistä ja mahdollisista haitoista selvittämällä tutkimuskirjallisuudesta low-code-alustojen hyötyjä ja haittoja. Kirjallisuuden osalta syvennytään low-coden historiaan, sen määritelmään, vahvuuksiin, heikkouksiin, parhaisiin tunnistettuihin käyttökohteisiin sekä eri low-code-teknologioiden potentiaalisuuteen. Kirjallisuuskatsauksessa esitellään myös tämän tutkielman kontribuutioluvussa käytettävä Mendix low-code-sovellusalusta. Mendixiä käsittelevässä luvussa kerrotaan Mendix low-code-alustan perusteista sekä mistä osista Mendixillä kehitetty sovellus arkkitehtuurisesti koostuu.

2.1 Low-code ja Low-code-sovellusalustat

Low-code on terminä tuore, mutta ajatuksena vanha. Terminä low-code on mainittu kirjallisuudessa ensimmäisen kerran vuonna 2014 (Sanchis ym. 2019). Kyseiset julkaisut eivät olleet vielä vertaisarvioituja. Low-code alkoi esiintyä vasta vuonna 2018 julkaistuissa vertaisarvioiduissa julkaisuissa. (Bucaioni, Cicchetti ja Ciccozzi 2022) Myös Sanchis ym. (2019) maintsevat artikkelissaan low-coden termin nuoruuden. Low-coden on kehitysmalli, jonka avulla pyritään tuottamaan sovelluksia nopeasti vähäiselläkin sovelluskehityksen taustalla. Termin uutuus ei tarkoita, että se olisi ilmiönä uusi. Itseasiassa low-code ja myöhemmin käsiteltävä no-code ovat hyvinkin vanhoja ideoita. Ne ovat osa kauan liikekeellä ollutta jatkumoa, jossa sovelluskehityksestä on pyritty tekemään intuitiivisempaa. (Woo 2020) Low-coden tausta onkin juurtunut pitkälle sovelluskehityksen historiaan, joten sillä itselläänkin on syvä historia. Low-coden historia on suurelta osin myös eri termeihin sidottuna.

2.1.1 Low-coden historia

Tässä luvussa tullaan kertomaan low-coden historiasta terminä, sekä sen historiasta sovelluskehityksen ajatusmallina. Kuten aiemmin tutkielmassa mainittiin, low-coden historia terminä ei ole pitkä. Termi on alkanut esiintyä vertaisarvioidussa kirjallisuudessa vasta vuodesta

2018 eteenpäin (Bucaioni, Cicchetti ja Ciccozzi 2022). Kyseessä on siis melko uusi termi. Low-coden ajatusmailma on esiintynyt vahviten malliohjatun kehittämisen yhteydessä ja MDD onkin relevantein metodologia low-coden kehittymisen kannalta (Bucaioni, Cicchetti ja Ciccozzi 2022). Bock ja Frank (2021) mukaan low-code-alustojen ja malliohjatun kehittämisen työkalujen raja on häilyväinen ja kyseinen raja vaatii vielä tarkennusta. Bock ja Frank (2021) kertovat myös, että low-coden hyödyt yrityksille syntyvät kykenevyydestä vastata kasvaviin tehokkuustarpeisiin sekä järjestelmien kehittämisen ja ylläpitämisen hinnan laskemiseen. Heidän mukaansa low-code vastaa myös tarpeeseen vastata alati muuttuviin järjestelmien kehittämiseen liittyviin vaatimuksiin (Bock ja Frank 2021). Myös Cruz ym. (2021) kertovat low-coden yhden suurimmista eduista olevan kehittämisen yksinkertaisuus, jonka ansiosta low-codea käyttävät organisaatiot voivat mukautua nopeasti jatkuvasti muuttuvilla sovelluskehityksen markkinoilla. Idea käsinkirjoitetun koodin määrän vähentämisestä on vuosikymmeniä vanha. Ruscio ym. (2022) kertovat artikkelissaan low-coden idean kehittyneen ajan saatossa 1980-luvulta tähän päivään asti. Aikaisimpia työkaluja käsinkirjoitetun koodin vähentämiseen ovat olleet 4GL ja CASE. Näiden jälkeen ovat 1990-luvulla Rapid application development (RAD), 2000-luvulla käyttäjälähtöinen kehittäminen ja viimeisten kahden vuosikymmenen aikana MDE (myös tunnettu MDD ja MDSD) eli malliohjattu kehittäminen. (Ruscio ym. 2022) Yksi Low-coden varhaisimmista esi-isistä on CASE. Seuraavassa kappaleessa kerrotaan tarkemmin CASE:sta ja siitä mitä sillä on pyritty saavuttamaan.

1980-luvulla CASE (Computer-Aided Software Engineering) luotiin vastaamaan datan määrän ja prosessien kompleksisuuden tuottamaan nousevaan tarpeeseen kehittää sovelluksia tehokkaammin (Sanchis ym. 2019). Tehokkuutta lisätään CASE:ssa esimerkiksi työkaluihin liitetyn automaation lisäämisellä. Sanchis ym. (2019) mukaan CASE onkin siis kokoelma automaatiotyökaluja ja -metodeita, jotka avustavat kehittäjiä sovelluksen eri kehitysvaiheissa. CASE-työkalut pohjautuvat usein oliosuuntautuneen suunnittelun lähestymistavan visuaaliseen työkaluun UML:ään, sisällönhallintajärjestelmiin ja prosessinhallintajärjestelmiin (Sanchis ym. 2019). Lundell ja Lings (2004) kertovat CASE työkalujen käyttöönoton olevan haastavaa, sillä työkalujen tarjoajat tarkastelevat markkinoita kontekstiriippumattomasti. Tämä voi vaikeuttaa työkalun implementointia hyvin kontekstiriippuvaiseen ympäristöön. Lundell ja Lings (2004) mukaan CASE:n käyttöönotto teknologiana on ollut heikkoa. Yksi syy tälle heikolle käyttöönotolle voi olla epärealistisen optimistiset odotukset CASE:a

kohtaan. Syynä voi olla myös aitojen käyttötapausten ja vaatimusten yhteensopimattomuus CASE-työkalujen kanssa. (Sanchis ym. 2019) CASE-metodologian ja -työkalujen idea saattoi olla jopa edellä aikaansa, mutta niiden jalkauttaminen käyttäjille oli vaikeaa ja heikosti suoritettua. Näiden syiden vuoksi onkin todennäköisesti syntyntyt tarve uusille abstraktion työkaluille, kuten RAD:lle.

1990-luvulla RAD eli Rapid Application Development nousi suosioon sovelluskehityksen kentällä. RAD-työkalut esiintyivät ensimmäistä kertaa perinteisen tekstieditorin haastajina sovelluskehityksen työkaluna (Beranic, Rek ja Heričko 2020). Mackay ym. (2000) kertovat RAD:n olevan enemmänkin kuin vain paketti työkaluja. Heidän mukaansa se on enemmänkin metodologia, jota hyödynnetään informaatiojärjestelmien kehitystyössä. Heidän mukaansa se on perinteistä vesiputousmallia nopeampi kehitystapa. Metodologian ideana on nostaa käyttäjää lähemmäs kehitysprosessia. Mackay ym. (2000) mukaan RAD:ssa käyttäjät eivät ole kehitysprosessissa vain informaation antajina ja järjestelmän vastaanottajina, vaan he ovat aktiivisesti mukana kehitysprosessissa. Heidän mukaansa RAD:ssa keskitytään vahvasti iteratiiviseen prototyypitykseen (Mackay ym. 2000). Sovellusta siis rakennetaan useamman kerran erittäin varhaisessa vaiheessa. RAD sopii erityisen hyvin käytettäväksi sellaisten sovellusten kehitysprosessissa, joiden pääpainona ovat käyttäjän suorittamat toiminnot, eikä niinkään laskenta tai sovelluksen kompleksisuus (Mackay ym. 2000). Sekä CASE että RAD olivat kirjallisuuden mukaan enemmän kuin paketti työkaluja käyttäjilleen. Kun RAD:n oli vakiintunut 1990-luvulla suosituksi metodologiaksi, seuraava ehdokas saman ajatusmailman edustajalle tuli näkyviin. Malliohjattu kehittäminen on tuonut vuosituhannen vaihteesta tähän päivään uutta näkökulmaa sovelluskehitykseen.

2000-luvun alussa malliohjattu suunnittelu (MDD/MDE) toi sovelluskehityksen kenttää automatisoidumpaan suuntaan tuomalla uuden keinon automatisoida sovelluskehityksen eri vaiheita. Ruscio ym. (2022) kertovat MDD:n ottavan kantaa sovelluskehityksen paradigmoihin käyttämällä erilaisia mallinnettuja prosesseja eli malleja sovellukehityksen elinkaaren eri vaiheiden toteuttamiseksi. Mallien tarkoituksena on abstrahoida perinteinen koodi erilaisiksi prosesseiksi, joita kehittäjän on intuitiivisempaa käyttää. MDD-malleja käytetään monissa eri sovelluskehityksen vaiheissa, kuten esimerkiksi testaamisessa, ylläpidossa ja koodigeneroinnissa. MDD-malleista rakennettavien työkalujen luomisessa hyödynnetään usein

käyttöaluekohtaisia kieliä. (Ruscio ym. 2022) Bock ja Frank (2021) mainitsevat low-coden ja MDD:n olevan hyvin samankaltaisia. Ruscio ym. (2022) kyseenalaistavat low-coden uutuutta nimittämällä low-codea on vain MDD:n uudeksi sanamuodoksi. Myös Bock ja Frank (2021) kyseenalaistavat low-coden innovatiivisuutta. Ruscio ym. (2022) tosin esittävät mielenkiintoisen ajatuksen low-coden ja MDD:n yhdistämisestä. He pohtivat mitä osia MDD:stä ja low-codesta voitaisiin hyödyntää, jotta saavutettaisiin vielä eheämpi kokonaisuus (Ruscio ym. 2022).

Low-codella on selkeästi pitkä historia jo ennen sen varsinaista syntyä vuonna 2014. Low-code ajatusmallina pohjautuu pitkälle kehittäjien työn helpottamiseksi ja tehokkuuden lisäämiseksi CASE:n, RAD:n ja MDD:n kautta. Näillä ja low-codella on Ruscio ym. (2022) mukaan yhteistä niiden tarkoitus tarjota työkaluja järjestelmien rakenteen ja käytöksen määrittämiseen. Low-coden uutuutta on haastettu ajatusmallina, mutta mikä on moderni käsitys low-codesta? Mitkä ovat sen vahvuudet ja mitä kehityskohteita siinä on? Tutkielman seuraavassa luvussa käsitellään low-coden määritelmää, tarkemmin ajatusmallia, joka siihen yhdistyy, sekä sen vahvuuksia ja heikkouksia.

2.1.2 Low-coden määritelmä

Low-code on terminä vielä uusi ja se hakee vielä määritelmänsä. Bock ja Frank (2021) kertovat tutkimuksessaan low-coden olevan terminä ongelmallinen. Heidän mukaansa sitä käytetään epäjohdonmukaisesti monissa eri lähteissä. Bucaioni, Cicchetti ja Ciccozzi (2022) määrittävät low-coden olevan kokoelma työkaluja ja metodeja laajemman metodologian kontekstissa, joka yleensä identifioidaan MDE:ksi. Ruscio ym. (2022) kuvaavat low-codea low-code-alustojen näkökulmasta. Heidän mukaansa low-code-alustat ovat sovelluskehitykseen tarkoitettuja järjestelmiä tai pilvipalveluja, joihin on sisällytetty visuaalisia deklaratiivisia tekniikoita perinteisen kirjoittamalla ohjelmoinnin sijaan. He myös määrittävät low-code-sovelluskehityksen pyrkivän vähentämään järjestelmän impelementoimiseksi vaaditun kirjoitettavan koodin määrää, kuitenkin ilman sovelluksen elinkaarenhallintaan liittyviä työkaluja (Ruscio ym. 2022). Vincent ym. (2022) mukaan low-code-alustojen tarkoituksena on vähentää järjestelmän kehittämiseen vaaditun koodin määrää abstraktion avulla. Low-code-alustan tarjoaja Mendix kertoo low-coden olevan sovelluskehitysmetodi, jossa teknisen teks-

timuotoisen kehitysympäristön sijaan käytössä on malliohjattu raahaamiseen ja pudottamiseen pohjautuva käyttöliittymä (“The Definitive Guide to Low-Code Development” 2022). Mendixin määritelmä low-codelle ei ole kovinkaan tarkka, sillä siinä on määriteltynä vain visuaalinen editori. Microsoft on yksi suurimmista low-code-sovellusalustan tarjoajista. Microsoft määrittää low-coden olevan sovelluskehityksen metodi, jolla sen käyttäjät voivat luoda sovelluksia raahaus ja pudotus -toiminnallisuuden ja visuaalisen avustuksen mahdollistamana (“Why low-code development matters right now” 2022). Molempien suurien low-code-sovellusalustojen tarjoajien määritelmä low-codesta on epämääräinen. Low-code termiä siis todellakin käytetään hyvin kevyillä perusteilla ja tähän liittyen Bock ja Frank (2021) maininta sen epäjohdonmukaisesta käytöstä vaikuttaa oikeutetulta. Low-coden tarkan määritelmän puute voi jättää monille erilaisille visuaalisille kehitystyökaluille mahdollisuuden väittää kuuluvansa low-code termin alle. Selkeästikin low-coden määritelmä kaipaa vielä selvennystä, josta Bock ja Frank (2021) ovat samaa mieltä.

Tällä hetkellä vaikuttaisi siis, että low-coden määritelmä on sovelluskehitystä, johon sisällytetään työkaluja, joilla minimoidaan kirjoitettavan koodin määrää. Kirjoitetun koodin määrän vähentämiseksi käytettävät työkalut ovat visuaalisia editoreja, joissa raahataan ja pudotetaan erilaisia komponentteja käyttöliittymään tai toimintalogiikkaan. On myös havaittu, että low-code identifioidaan joskus MDE:ksi (Bucaioni, Cicchetti ja Ciccozzi 2022) Seuraavassa luvussa avataan low-codea ajatusmallina.

2.1.3 Low-code ajatusmallina

Low-code voi olla ajatusmallina ohjelmistokehityskokemusta omaavalle henkilölle aluksi hieman haastava. Sanchis ym. (2019) kertovat artikkelissaan ihmisten olevan luonnostaan muutosta vastustavia olentoja ja tästä johtuen kehittäjän ajatusmallin omaksuminen voi olla haastavaa. Kehittäjän tulisi ajatella sovelluskehitystä uudella tai vähintäänkin huomattavasti aikaisemmasta eroavalla tavalla. Lähes yhtään komponenttia ei tarvitse rakentaa itse. Melkein kaikki on luotavissa konfiguroitavilla valmiskomponenteilla. Sovelluksen suunnittelun kannalta valmiskomponenttien rajoitteiden ymmärtäminen on yksi tärkeimmistä low-code ajatusmallin perusteista. Kehittäjän tulee ymmärtää parhaat tavat hyödyntää käytössä olevien komponenttien vahvuuksia. Hurlburt (2021) kertoo low-coden olevan kuin lego-

palikoiden yhdistelyä. Jos legoilla rakentaessa ei ymmärrä legojen liitoskohtien yhteensopivuuksia, rakennelmaa ei joko saa kasaan, tai siitä tulee helposti hajoava.

Low-code-ammattilaisten tulisi ymmärtää, että projekteissa voidaan hyödyntää myös yrityksen eri osastoja (Beranic, Rek ja Heričko 2020). Myös negatiiviset kulttuurilliset ajatusmallit ja tietoisuuden puute voivat hidastaa low-coden ajatusmallin hyväksymistä ilmentymällä käyttöönoton hitautena (Bhattacharyya ja Kumar 2021). Beranic, Rek ja Heričko (2020) huomasivat tutkimuksessaan, että yrityskontekstissa mielikuva low-codesta painottuu vielä skeptiseen ja negatiiviseen suuntaan. Yrityskontekstissa low-coden positiivisen ajatusmallin eteenpäin ajaminen voisi mahdollistaa low-coden ajatusmallin nopeamman hyväksymisen. Low-coden ajatusmalli eroaa siis perinteisestä kehittäjän ajatusmallista huomattavasti ja jotta se saataisiin hyväksyttyä nopeammin, tulisi yrityksen sisäisen kulttuurillisen ajatusmallin myös muuttua positiivisemmaksi. Seuraavassa luvussa tarkastellaan low-code-metodologian ja low-code-alustojen vahvuuksia.

2.1.4 Low-coden vahvuudet

Tässä luvussa käsitellään eri lähteiden näkökulmia low-coden vahvuuksiin. Jokaisella metodologialla on omat vahvuutensa. Low-codesta on kirjallisuudessa annettu paljon vahvoja mielipiteitä, joista osa on erittäin positiivisia. Ruscio ym. (2022) arvioivat low-coden menestymismahdollisuuksia. He ovat löytäneet monia syitä, miksi low-code voi pärjätä paremmin kuin hiljaiseloon vajonneet CASE ja 4GL. Pilvipalveluiden yleistyminen ja niiden käyttöönotto low-code-sovellusten julkaisualustana nopeuttavat sovelluksen julkaisua. Toisena huomiona low-code-alustojen pilvipohjaisuus minimoi tarvittavan asentamisprosessin ja näin käyttäjät pääsevät helpommin käsiksi kehitystyökaluihin. (Ruscio ym. 2022) Low-coden on todettu vähentävän kuluja yhden kehityssyklin vaatiman ajan lyhentyessä. Kun sovelluksesta on pohja valmiiksi rakennettuna, voidaan keskittyä visuaaliseen liiketoimintalogiikan kannalta tärkeään prosessien kehittämiseen järjestelmän alusta asti koodaamisen sijaan. (Sanchis ym. 2019) Sovelluksen valmis pohja mahdollistaa sovelluskehityksen nopeutumista samalla tavalla, kuin Ruscio ym. (2022) mainitsema pilvipalveluiden tarjoama valmis pohja. Low-coden yksi vahvuuksista on selkeästi sen mahdollistama sovelluksen valmis pohja sekä kehitystyölle että julkaisemiselle.

Low-coden vahvuus voi olla myös nykypäivän tarjoamat resurssit. Nykypäivän digitaalisessa mailmassa koko elämänssä asuneille henkilöille opetetaan ohjelmointitaitoja jo peruskoulussa ja he ymmärtävät todennäköisesti teknologiaa huomattavasti paremmin kuin aiempi sukupolvi. Ruscio ym. (2022) kertovat kehittäjäpulasta ja low-coden mahdollisuuksista vastata kyseiseen haasteeseen antamalla matalamman ohjelmointikokemuksen omaaville henkilöille ja yrityksille keinon kehittää omia sovelluksiaan. Myös Sanchis ym. (2019) mainitsevat low-coden mahdollistavan yritysten sisäisten henkilöiden sisällyttämisen sovelluskehitykseen ja jopa mahdollisuuden yritysten sisäisten henkilöiden itse kehittämiin sovelluksiin. Bhattacharyya ja Kumar (2021) ovat huomanneet low-coden ja no-coden antavan kansalaiskehittäjille vallan rakentaa yritysten sisäisiä sovelluksia. Yrityksen sisäisten liiketoimintaan erikoistuneiden henkilöiden sisällyttäminen vahvasti sovelluskehitykseen voisi mahdollisesti vähentää kuluja huomattavasti. Myös Beranic, Rek ja Heričko (2020) ovat huomanneet low-coden mahdollistavan yrityksen eri osastojen sisällyttämisen sovelluskehitykseen. Kun sovelluskehityksen yksinkertaisiin vaiheisiin saadaan sisällytettyä vähemmän sovelluskehitystaustaa omaavia henkilöitä, sovelluskehityksen ammattilaisille jää enemmän aikaa haastavampiin ja aktivoivampiin tehtäviin (Woo 2020). Ruscio ym. (2022) mainitsevat median vaihtuvan analogisesta digitaaliseen muotoon. Tarvittavat koulutusmateriaalit ja eri järjestelmien dokumentaatiot ovat nykyään kaikkien saatavilla internetin välityksellä.

Yksi suurimmista low-coden vahvuuksista on sen suoraviivaisuus (Cruz ym. 2021). Low-code-alustojen valmispohjien ja -komponenttien mahdollistamana yrityksen työntekijät voivat rakentaa sovelluksia ilman suuria aikaresursseja. Sanchis ym. (2019) kertovat low-coden mahdollistavan jopa viidestä kymmeneen kertaisen kehitysnopeuden. Myös Bhattacharyya ja Kumar (2021) mukaan sovelluskehityksen vaatima aika pienentyy ja sovelluskehityksestä tulee joustavampaa low-code-ja no-code-ympäristöjen mahdollistaman nopean muutoksen ansiosta. Toki jokainen käyttökohde ei sovi low-codelle niin hyvin, että voitaisiin saavuttaa joka kerta sama kehitysnopeus. Low-code-alustat käyttävät usein DSL (Domain Spesific Language)-kieliä, jotka suoriutuvat tietyn kohdealueen tehtävistä erittäin tehokkaasti, mutta eivät kuitenkaan sovi jokaiselle kohdealueelle käytettäväksi (Galhardo ja Silva 2022). Low-code-kehitystä tukee myös hyvin RAD-metodologia (Cruz ym. 2021). Myös Sanchis ym. (2019) mukaan low-code tulee mahdollisesti olemaan monien yritysten kohdalla ratkaisevassa osassa sovelluskehityksen työskentelyn tehokkuudessa ja kilpailukyvyn säilyttämi-

sessä. Heidän mukaansa low-code-alustat antavat hyöddyllisiä ratkaisuja sovelluskehityksen automatisointiin ja nopeuttamiseen (Sanchis ym. 2019). Woo (2020) kertoo joidenkin low-code-alustojen sisältävän tekoälyä hyödyntäviä apureita. Esimerkiksi Mendixin low-code-alustassa on sisäänrakennettuna tekoälyä hyödyntävä avustaja, joka oppii käyttäjän kehittämistottumusten mukaisia malleja ja pystyy ehdottamaan käyttäjälle hänen suosimiaan kehittämismalleja (Woo 2020). Low-code ei kuitenkaan ole täydellinen metodologia, eikä sitä varten tuotetut alustat ole täydellisiä. Sillä on myös monia heikkouksia. Seuraavassa luvussa käsitellään low-coden heikkouksia ja negatiivisia asenteita low-codea kohtaan.

2.1.5 Low-coden heikkoudet

Sovelluskehityksen haasteet alkavat näkyä viimeistään, kun sovelluskehityksen valta luovutetaan käyttäjille. Tässä luvussa tullaan käsittelemään kirjallisuudesta löydettyjä low-coden heikkouksia ja siihen liittyviä negatiivisia asenteita. Woo (2020) esittää monia huolia low-codella tuotetuista sovelluksista. Hän kertoo vähän kehityskokemusta omaavien kansalaiskehittäjien kehittämien sovellusten olevan huonompilaatuisia. Yksi syy kansalaiskehittäjien kehittämien sovellusten huonompaan laatuun on laadun varmistuksen heikkous ja taidottomuus sovelluksen sisältämien virheiden alkuperän selvittämiseen. (Woo 2020)

Woo (2020) esittää myös huolensa low-codella tuotettujen sovellusten ylläpidettävyydestä ja päivitettävyydestä. Beranic, Rek ja Heričko (2020) tuottaman kyselytutkimuksen tuloksista selviää, että maisteriopiskelijat eivät usko low-codella tuotettujen ratkaisujen olevan helposti ylläpidettäviä. Heidän tutkimukseensa on osallistunut 78 ammattilaista ja 35 maisteriopiskelijaa (Beranic, Rek ja Heričko 2020). Tutkimukseen osallistuneiden maisteriopiskelijoiden määrä on melko vähäinen ja maisteriopiskelijoiden tietotaidot IT-alalla eivät välttämättä ole vielä riittävät. Tutkimuksen tulosta low-coden ylläpidettävyydestä voidaan ehkä tulkita enemmänkin asenteena, kuin faktana. Ylläpidettävyys on yksi suurimmista huolen aihesta, sillä myös Cruz ym. (2021) esittävät huolensa low-codella tuotettujen sovellusten ylläpidettävyydestä. Heidän mukaansa low-codella tuotettujen sovellusten ylläpidettävyys voi olla vaikeaa. He kertovat,että vaikeutta luo itse low-code-alustan ylläpito ja muokattavuus. Alustaa käyttävien henkilöiden on erittäin haastavaa ellei jopa mahdotonta korjata ongelmaa, jos alustan sisäisessä optimisaatiossa on jotain vikaa. Alustaa hyödyntävät kehittäjät joutuvat

odottamaan seuraavaa päivitystä ja toivomaan sen sisältävän korjauksen kyseiseen ongelmaan. Sovellusten ylläpito voi olla jopa mahdotonta, jos low-code-alustan tarjoaja lopettaa toimintansa. (Cruz ym. 2021)

Jos alustan tarjoaja lopettaa toimintansa, sovelluksen kehittäjä voi huomata olevansa kuin lukolla kiinni tarjoajassa. Yksi suurimmista lähdekirjallisuudessa esiintyneistä huolista on niin sanottu "vendor lock"eli tiettyyn tarjoajaan lukkiutuminen toteutetun ratkaisun muokkaamisen mahdottomuus ilman kyseisen tarjoajan työkaluja, myös tunnetttu toimittajaloukkuna. Tämä ilmiö on yksi suurimmista syistä, jonka vuoksi yritykset eivät ota low-codea käyttöön omissa projekteissaan (Sanchis ym. 2019). Low-code-ympäristöjen käyttöönottoa haittaa erityisesti työkalujen ja low-code-kehitystyylien pirstaloituneisuus eri tarjoajien välillä (Sanchis ym. 2019). Pirstaloituneisuus voi ehkä näkyä myös yhtenä toimittajaloukun aiheuttajana, sillä yhden tarjoajan työkaluilla toteutettuja ympäristöjä voi harvoin muokata toisen tarjoajan vastaavilla työkaluilla. Myös Bucaioni, Cicchetti ja Ciccozzi (2022) pohtivat toimittajaloukun huolettavuutta monilla low-codea käyttävillä tahoilla. Beranic, Rek ja Heričko (2020) tutkimuksen tuloksissa maisteriopiskelijat eivät uskoneet low-coden sopivan kriittisessä asemassa olevien järjestelmien kehittämiseen. Toimittajaloukun ja pirstaloitumisen mahdollisuus saattaa ohjata asennetta low-coden sopivuutta kohtaan negatiiviseen suuntaan. Jos kriittisen järjestelmän ylläpito muuttuu mahdottomaksi, sekä järjestelmän toimittaja että tilaaja voivat kärsiä tappioita. Low-coden käyttöönotto vaatii siis vahvaa luottamusta ja todennäköisesti monissa tapauksissa myös sopimuksellisia varmistuksia low-code-alustan tarjoajan ja järjestelmän toimittajan välillä. Mendix kertoo alustansa dokumentaatiossa estävänsä toimittajaloukun syntymistä avaamalla alustaansa mahdollisimman paljon (“Platform Evaluation Guide” 2023). Mendix väittää, että Mendix-projektien lähdekoodin voi siirtää mihin tahansa teknologiapinoon (“Platform Evaluation Guide” 2023). Jotkin low-code-alustojen tarjoajat siis ainakin väittävät pyrkivänsä välttämään toimittajaloukun syntymistä, mutta lähdekirjallisuudessa esiintyy edelleen huolta kyseisestä aiheesta.

Sovellukehitys low-code sovellusalustoilla voi muuttua painajaiseksi, jos sovelluskehityksen parhaista käytänteistä ei pidetä kiinni. Aiemmin mainitut Beranic, Rek ja Heričko (2020) löydökset maisteriopiskelijoiden uskosta low-codella tuotettujen ratkaisujen ylläpidettävyyteen on huolettavaa. Myös muussa lähdekirjallisuudessa huomioitiin low-code-alustoilla tuotet-

tujen sovellusten ylläpidettävyyden ja päivitettävyyden mahdolliset haasteet. Suurin kirjallisuudessa esiintynyt huoli oli kuitenkin toimittajaloukku. Seuraavassa luvussa tarkastellaan low-codelle parhaaksi tunnistettuja käyttökohteita.

2.1.6 Parhaat tunnistetut käyttökohteet

Low-code-alustat hyödyntävät usein DSL:iä (Galhardo ja Silva 2022), joten ne eivät sovi jokaiselle kohdealueelle. Low-codella on omat käyttökohteensa ja tässä tutkielmassa haluttiin myös selvittää low-code-alustoille sopivia käyttökohteita. Low-code-alustojen relevanttiuteen tietyllä kohdealueella vaikuttaa huomaattavasti niiden käyttämän DSL:n sopivuus kyseiselle kohdealueelle (Galhardo ja Silva 2022). Alamin ym. (2023) kertovat artikkelissaan low-coden olevan suosittua erityisesti dynaamisten lomakepohjaisten sovellusten tuottamisessa. Totterdale (2018) kehitti Mendix low-code-teknologialla tutkimuskyselysovelluksen, joka koostui pääosin lomakkeista. Hänen tutkimuksensa tuloksista selvisi, että Mendixillä toteutettuna lomakepohjaisen sovelluksen toteutus oli nopeaa. Rakennettu sovellus oli kuitenkin melko pieni, sillä se sisälsi vain 25 lomaketta ja vain vähän toimintalogiikkaa (Totterdale 2018). Huomattavasti laajempaa sovellusta kehittäessä sovelluksen kehittämisen nopeudesta ei Totterdale (2018) tutkimuksessa ollut mitään tietoa.

Low-coden perii monia periaatteita RAD:lta. Mackay ym. (2000) kertovat RAD:n olevan sopiva erityisesti sellaisten sovellusten kehittämiseen, joissa pääpaino on käyttäjän suorittamissa toiminnoissa, eikä niinkään laskennassa tai sovelluksen kompleksisuudessa. Mackay ym. (2000) väittämä RAD:n sopivuudesta peilaa mielenkiintoisesti Totterdale (2018) löydöksiin hänen lomakepohjaisen hyvin vähän toimintalogiikkaa sisältävän sovelluksensa Mendixillä toteuttamisen nopeudesta. Laajempien kokonaisuuksien toteutuksista on erittäin vähän tutkimustietoa kirjallisuudessa, joten low-coden sopivuudesta laajoihin kehitysprojekteihin on vaikeaa määritellä. Kuitenkin jotkin low-code-alustat, kuten Mendix panostavat erityisesti palveluidensa skaalautuvuuteen Vincent ym. (2019), joten ne voivat sopia alustansa arkkitehtuurin puolesta laajoihin projekteihin. Monet low-code-alustojen tarjoajat tarjoavat myös listan parhaista käytänteistä, joita noudattamalla projektien kehitystiimien on helpompi ylläpitää suuren kokoluokan projekteja. Koska eri low-code-alustat sopivat eri kokoluokan projekteille (Vincent ym. 2019), on hyvä tunnistaa oikea teknologia omaan projektiin.

Projektin kohdealue tulee erityisesti ottaa huomioon projektin teknologian valinnassa. Seuraavassa luvussa esitellään neljä johtavaa low-code-alustaa ja vertaillaan niiden potentiaalisuutta sovelluskehityksessä.

2.1.7 Eri low-code teknologioiden potentiaalisuus

Tässä luvussa käsitellään neljän johtavan low-code-alustan vahvuuksia ja heikkouksia. Suurin osa tämän luvun tiedosta on peräisin Vincent ym. (2019) sekä Vincent ym. (2022) tutkimuksista. Näitä kahta eri tutkimusta käytetään, koska halutaan myös nähdä, ovatko johtavat low-code-alustat vaihtuneet kolmen vuoden aikajänteellä. Nämä kaksi tutkimusta arvioivat monia eri low-code-alustoja, mutta niistä on avattu tässä tutkielmassa laajemmin neljää. Tarkemmin avattavat teknologiat Vincent ym. (2019) ja Vincent ym. (2022) tutkimuksista ovat Mendix, Microsoft Power Platform, Outsystems ja Salesforce. Kuivoista 1 ja 2 selviää näiden neljän olevan huomattavasti edellä muita kilpailijoita sekä vuonna 2019 että vuonna 2022 ja vertailuun haluttiin ottaa mukaan vain selvät johtajat. Kuviosta 2 nähdään ServiceNow:n nousseen johtajien kategoriaan vuonna 2022, mutta sitä ei oteta mukaan arviointiin.

Kuviossa 1 nähdään x-akselilla yrityksen vision valmius low-code-alustan tulevaisuuden ja kyvykkyyksien kannalta. Y-akseli kuvastaa yrityksen low-code-alustan kykyä tuottaa toimivia ratkaisuja. Kuviosta 1 huomataan, että Salesforce, Microsoft, OutSystems ja Mendix ovat selkeitä johtajia Vincent ym. (2019) tutkimuksen mukaan. Ensin käsitellään kuvion 1 mukaan visiossa pisimmällä olevaa Mendixiä sekä sen vahvuuksia ja heikkouksia.

Kuvio 1. Gartner Magic Quadrant 2019 enterprise-tason low-code-alustoista, lähde: Vincent ym. (2019)

Mendix on Siemensin 2018 ostama yritys ja low-code-alusta. Yrityksenä Mendix pyrkii parantamaan kehittäjien ja liiketoiminnan yhteistyötä sekä sovelluksien elinkaaren hallintaa (Vincent ym. 2019). Vincent ym. (2019) mukaan Mendix on kaikista arviointiin valituista low-code-alustoista visionäärein. Syinä tähän on heidän panostuksensa low-code-alustan tutkimukseen ja kehitykseen. Mendixin vahvuus perustuu sovelluskehityksen siiloja rikkovaan kahden kehitysympäristön rakenteeseen. Mendix tarjoaa sekä kansalaiskehittäjille, että ammattikehittäjille omat työkalunsa. Mendixillä on myös on tekoälyä hyödyntävä apuri, jonka tarkoituksena on auttaa sovelluksen kehityksessä. Mendix sopii kehittyneiden, integraatioita ja haastaviakin arkkitehtuureita sisältävien enterprise-tasoisten sovellusten kehittämiseen. On ammattikehittäjien mielestä helppokäyttöinen. (Vincent ym. 2019) Suureksi eduksi katsottiin toimintalogiikan muodostamisessa käytettävien Microflow:ien ja Nanoflow:ien vuo-

kaaviomainen visuaalinen rakenne. Mendixin vahvuudet ovat pysyneet suurilta osin samoina vuosien kuluessa (Vincent ym. 2022). Tosin Vincent ym. (2022) eivät painota enää Mendixin strategiaa hajottaa sovelluskehitys kahdeksi eri kehitystyökaluksi. Heikkoutena Mendixissä koettiin kansalaiskehittäjien tuen heikkous, sekä hinnoittelun jäykkyys (Vincent ym. 2019).

Kuvio 2. Gartner Magic Quadrant 2022 enterprise-tason low-code-alustoista, lähde: Vincent ym. (2022)

Microsoft Power Platform on vahva monissa käyttötarkoituksissa. Sen suurimmat vahvuudet sijaitsevat M365-palveluita käyttävien asiakkaiden ympäristöjen laajentamisessa ja integraatiossa. Power Platform koostuu monista eri työkaluista, joilla voidaan tuottaa eri kohdealueille erilaisia sovelluksia. Power Platformissa on yli 240 erilaista liitintä, joilla sovelluksia voi yhdistää vaivattomasti eri verkkopalveluihin. Sovelluksia voidaan luoda muun muassa Canvas App -toiminnallisuudella, jossa on yksinkertainen "raahaa ja pudota"-tyyppinen käyttöliittymä. Canvas App:ssa toimintalogiikka toteutetaan Excel-funktioiden tyylisel-

lä PowerFx-ohjelmontikielellä. Power Platform on saanut eniten palautetta sen nopeudesta. (Vincent ym. 2019) Vincent ym. (2022) mukaan Microsoft Power Platform tulee saavuttamaan suurimman määrän käyttäjiä Microsoft 365 ja Dynamics ympäristöjen nykyisten käyttäjämäärien suuruuden vuoksi. Vincent ym. (2022) kertovat uusina Microsoftin vahvuutena Microsoft Power Appsin olevan oletuksena useiden M365-lisenssien mukana käytettävissä. Uutena heikkoutena he kertovat Power Platformilla tuotettujen mobiilisovellusten vaativan kehittyneen käyttäjienhallinnan, joka sulkee ulos useat B2C-asiakasprojektit (Vincent ym. 2022).

OutSystems on alunperin .NET:iin pohjautuva RAD-kehitysympäristö, joka nykyään tarjoaa enterprise-tasoisen low-code-alustan (Vincent ym. 2019). Vincent ym. (2019) mukaan OutSystems keskittyy enterprise-tasoisten sovellusten tuottamiseen, ketterään kehitykseen ja jatkuvaan julkaisuun. He kertovat OutSystemsin olevan vaikeakäyttöinen kansalaiskehittäjien näkökulmasta. Se tukee AWS-ja Azure-pilvi-integraatioita, mutta keskittyy erityisesti Microsoftin palveluihin .NET-taustansa vuoksi. OutSystems on saanut negatiivista palautetta laajojen sovellusten osalta. (Vincent ym. 2019) Vuonna 2022 OutSystems tarjoaa suuren määrän intensiivikoulutusta ja sertifikaatteja, jotta käyttäjät onnistuisivat tavoitteissaan (Vincent ym. 2022). Uutena heikkoutena Vincent ym. (2022) näkevät OutSystemsin olevan niin yleiskäyttöinen, että joissain kohdealuekohtaisissa ratkaisuissa asiakkaat voivat löytää nopeampia ratkaisuja muilta toimittajilta.

Vuonna 2019 suurimpana taloudellisena voimana Vincent ym. (2019) mukaan Salesforce on saavuttanut vakaan jalansijan low-code-markkinoilla. Salesforcen Lightning Platform keskittyy asiakkaiden käyttämiin sovelluksiin ja olemassa olevien SaaS:ien laajennuksiin. Suurimpana heikkoutena Lighting Platformissa on sen käytön monimutkaisuus erityisesti kompleksisia sovelluksia rakentaessa. Salesforce on saanut positiivista palautetta innovatiivisesta otteestaan. (Vincent ym. 2019) Vincent ym. (2022) kertovat Salesforcen vastaavan markkinoiden tarpeisiin nopeasti. Nopeus tulee Salesforcen kolme kertaa vuodessa julkaistavista uusista alustan ominaisuuksista sekä aktiivisesta alustan virheiden korjaamisesta. Salesforcea kritisoidaan sen innovaation puutteesta vuonna 2022 (Vincent ym. 2022). Näitä kahta julkaisua vertailtaessa huomataan, että Salesforce vaikuttaa löytäneensä ainakin hetkeksi oman paikkansa. Tämä ei kuitenkaan välttämättä ole hyvä asia, jos innovaatio katoaa.

Aiemmin mainittujen neljän teknologian joukosta valittiin projektissa käytettävä teknologia. Eri teknologioiden vahvuuksia ja heikkouksia tarkasteltaessa kaksi saavuttivat eniten huomiota. Microsoft Power Platform loisti nopealla projektien toteutusnopeudellaan. Microsoft Power Platformiin kuuluva Canvas App vaikutti nopealta tavalta rakentaa MVP-versio vanhasta mobiilisovelluksesta. Myös toimintalogiikan rakentaminen Excel-funktioiden kaltaisella Power Fx:llä vaikutti mielenkiintoiselta mahdollisuudelta. Kyseiseen teknologiaan tutustuttaessa päädyttiin kuitenkin jättämään Microsoft Power Platform sivuun kirjallisuudessakin esiintyneistä lisenssisyistä. Projektissa käytetyksi teknologiaksi valikoitui Mendix erityisesti sen joustavien intergraatiomahdollisuuksien vuoksi. Toinen suuri etu on MVP-sovelluksen kehityksen hinta. Mendixillä voitaisiin kehittää sovelluksen MVP-versio täysin ilmaiseksi (“Platform Evaluation Guide” 2023). Vanhan mobiilisovelluksen korvaavan MVP-sovelluksen rakentamiseen ei tarvitsisi hankkia lisenssejä. Tämä nopeuttaisi toimintaa. Microsoft Power Platformin nopeutta tai tehokkuutta ei voida tämän tutkielman perusteella arvioida, koska sitä ei käytetty sovellusprojektissa. Mendixin vahvuuksien lisäksi sen heikkoudet kansalaiskehittäjien tuen ja hinnoittelun osalta eivät luoneet esteitä projektille, sillä tarkoituksena oli rakentaa MVP-versio, jota käytetään testiympäristöstä käsin. MVP-sovellus oli tarkoitus rakentaa ammattilaiskehittäjien toimesta, joten kansalaiskehittäjien tuen puutteesta ei tarvinnut huolehtia. Teknologian valintaan vaikuttivat myös kehittäjien kokemus ja taustatiedot Mendixiä koskien. Mendixin osalta taustatyötä tehdessä heräsi huoli Mendixin tarjoaman projektinhallinnan työkalun jäykkyydestä. Seuraavassa luvussa tarkastellaan Mendixiä ja sen osia viitaten Mendixin dokumentaatioon.

2.2 Mendix low-code-alustasta teknologiana ja kehitystyökaluna

Tässä luvussa tullaan tarkastelemaan Mendixiä teknisestä näkökulmasta. Luvussa tarkastellaan Mendixiä yleisesti, sillä toteutettujen sovellusten arkkitehtuuria, Mendixin low-code-alustan lisäksi tarjottavia projektinhallintatyökaluja sekä kehitysympäristöjen eroavaisuuksia. Lähteenä käytetään pääosin Mendixin verkkosivuilta löytyvää teknistä dokumentaatiota (“Mendix Documentation” 2023) sekä Mendixin alustan arviointiohjetta (“Platform Evaluation Guide” 2023). Ensimmäisenä tarkastellaan Mendix low-code-alustaa yleisesti.

2.2.1 Low-code-sovellusalusta

Mendix on enterprise-tasoisten sovellusten kehitykseen tarkoitettu low-code-alusta, jolla pystytään luomaan myös pienemmän skaalan sovelluksia. Sitä markkinoidaan PaaS(Platform-as-a-Service)-tyylisenä palvelumuotona. Low-code-alusta on vastuussa muun muassa lähdekoodin generoinnista, kehitysympäristön hallinnoinnista ja kehittäjän avustamisesta kehityksen aikana. Kehittäjän tarvitsee vain ladata haluamansa versio low-code-alustasta ja asentaa se helppokäyttöisen asennusohjelman avulla omalle tietokoneelleen. Kun alusta on asennettu, kehittäjä voi päivittää sen asentamalla uuden version samalla tavalla kuin ensimmäinen on asennettu. Näin kehittäjälle jää vanha versiotuki käyttöönsä, jos käytettävää alustan versiota ei haluttaisi päivittää projektin edetessä. Jos alusta ja sovellus halutaan päivittää projektin edetessä, uuden alustan asennettuaan kehittäjän tarvitsee vain avata sovellus, jonka jälkeen kehittäjän luvan saatuaan alusta tekee tarvittavat päivitykset sovellukseen.

Sovelluksia voidaan luoda mallien pohjalta, jolloin tarvittavat sovelluksen arkkitehtuuriset rakenteet tulevat valmiina sovellusmallin mukana. Mendixillä pystytään luomaan esimerkiksi yksinkertainen CRUD-toiminnallisuuden sisältävä sovellus pelkän Excel-tiedoston pohjalta tekoälyä hyödyntäen. Mendixillä pystytään luomaan myös täysin tyhjiä sovelluksia, joissa on vain sovelluksen perusarkkitehtuuri, kuten turvallisuusasetusten määrittämismahdollisuudet ja navigaatio valmiina konfiguroitavaksi. Sovelluksen perusarkkitehtuuri riippuu sovelluksen tyypistä riippuen siitä, tuleeko sovellus käyttöön mobiililaitteille, vai onko sovellus verkkoselainpohjainen. Sovellukseen voidaan vielä myöhemmin lisätä myös esimerkiksi verkkoselainpohjainen käyttöliittymä eli sovellukset eivät ole lukittuja tiettyyn kumpaankaan ennaltamääritettyyn malliin. Kun kehittäjä on luonut sovelluksen tarvittavien konfiguraatioiden mukaisesti, hän pääsee välittömästi kehittämään sovellusta joko kansalaiskehittäjille tarkoitetussa Mendix Studiossa tai ammattilaiskehittäjille tarkoitetussa Mendix Studio Prossa. Sovellukset luodaan joko Mendix Studio Pro:sta tai Mendixin tarjoamasta verkkoselainpohjaisesta pilvikäyttöliittymästä eli Mendix Portalista.

Mendix Developer Portalissa (myöhemmissä vaiheissa vain Portal) kehittäjät pääsevät konfiguroimaan sekä sovellusten että oman yrityksensä tietoja. Portalista löytyy myös yhteistyötyökaluja, jotka avustavat kehittäjiä työskentelemään tiiminä ketterällä tavalla. Mendix on rakentanut Portaliin oman projektihallintatyökalun, jota voidaan tarkastella joko perinteise-

nä listauksena tai modernimpana kanban-tauluna. Projektinhallintatyökalu integroituu automaattisesti sekä Mendix Studioon että Mendix Studio Pro:hon, joista voidaan muun muassa vaihtaa tikettien tilaa. Mendix Portalissa kehittäjät pääsevät tarkastelemaan omien oikeuksiensa mukaisesti yrityksen kehittämiä sovelluksia ja niiden konfiguraatioita, sekä projekteja joissa on itse mukana. Turvallisuusasetuksia, projektin osallisten oikeuksia sekä rooleja voidaan hallinnoida Portalista käsin. Portalissa on myös mahdollisuus rakentaa oma yrityksen sisäinen koulutusohjelma, jolla saadaan uudet kehittäjät helposti ohjattua oikeisiin kursseihin ja opintokokonaisuuksiin. Portalissa voidaan myös hallinnoidan julkaisuja kehitysympäristöstä testiympäristöön ja tuotantoympäristöön. Ympäristöjen pystytyksestä vastaa Mendix, joten siitä ei yrityksen tarvitse huolehtia. Tämän tutkielman sovelluksen MVP-versiossa ei tarvittu useita eri ympäristöjä, sillä sovelluksen uusien ominaisuuksien testaus ja hyväksyntä pystyttiin suorittamaan ketterästi kehitystiimin ja asiakkaan kesken kehitysympäristössä. Jos sovelluksen kehitystä jatketaan MVP-versiosta eteenpäin, tarvitaan omat ympäristöt testaukselle, sekä tuotannolle. Tässä tutkielmassa Portalista hyödynnettiin sovellusten tarkastelua, projektihallintatyökalua ja turvallisuusmäärityksiä. Portalin osien käytön lisäksi projektiin tehtiin myös linjaus kehitystyökaluista. Tässä projektissa ei käytetty Mendix Studiota, vaan kaikki projektin varsinainen kehittäminen tehtiin Mendix Studio Pro:ssa. Mendix-sovellusten kehitystyökaluja on kaksi, mutta sovelluksia voidaan myös laajentaa tarvittaessa luomalla komentorivityökaluilla uusia komponentteja. Näistä komponenteista kerrotaan lisää myöhemmin. Seuraavaksi tutkielmasssa tullaan esittelemään Mendix Studio Pro:ta. Mendix Studiota ei esitellä yksintyiskohtaisesti, sillä se ei ole merkityksellinen tutkielman kannalta sen käytön puutteen vuoksi.

2.2.2 Mendix Studio ja Mendix Studio Pro

Osa sovelluskehityksen tehtävistä, kuten ominaisuuksien raakamallien suunnittelu on mahdollista tehdä ilman laajamittaista ohjelmointikokemusta. Jotkin ominaisuudet vaativat huomattavasti enemmän substanssiosaamista ohjelmoinnin ja sovellussuunnittelun alalta, jolloin tarvitaan ammattilaiskehittäjän apua. Ammattilaiskehittäjää vaativien ominaisuuksien sisällyttäminen kansalaiskehittäjän hyödyntämään ympäristöön voi mahdollisesti tehdä kansalaiskehittäjän työstä haastavampaa. Tämän vuoksi Mendix Studio ja Mendix Studio Pro

sisältävät eri kirjon ominaisuuksia. Tässä luvussa tarkastellaan Mendix Studion ja Mendix Studio Pro:n eroavaisuuksia.

Mendix Studio on kansalaiskehittäjille suunnattu low-code-sovelluskehitystyökalu. Se on verkkoselainpohjainen pilvityökalu, joka on helposti käytettävissä kehittäjän fyysisestä laitteesta riippumatta. Kehittäjä voi löytää Mendix Studion Portalista kehitettävän projektin alta. Se sisältää kyvykkyyden rakentaa uusia näkymiä sekä uutta yksinkertaista toimintalogiikkaa. Mendix Studion vahvuutena on sen helposti opittava intuitiivinen käyttöliittymä ja kehittäjää avustava tekoälyä hyödyntävä apuri. Mendix Studion ero Studio Pro:hon tulee esille monissa paikoissa, kuten monimutkaisemman toimintalogiikan rakentamisen yhteydessä. Studiossa ei pysty esimerkiksi toteuttamaan rajapinta-integraatioita, kun taas Studio Pro:ssa kehittäjällä on mahdollisuudet siihen. Studio on siis tarkoitettu kansalaiskehittäjille, joiden tarkoituksena on enemmänkin tuottaa uusia ideoita ja yksinkertaisia näkymiä. Syvempi toimintalogiikan määrittäminen jää siis ammattikehittäjien harteille. Yksinkertaistetulla kehitystyökalulla työskennellessä kansalaiskehittäjää ei häiritse Studio Pro:ssa esiintyvä valtaisa mahdollisuuksien ja työkalujen kirjo, vaan kansalaiskehittäjä voi keskittyä hänen työnsä kannalta olennaisiin asioihin. Seuraavaksi käsitellään ammattilaiskehittäjän työkalua Mendix Studio Pro:ta.

Mendix Studio Pro on ammattilaiskehittäjille tarkoitettu low-code-sovelluskehitystyökalu. Se on työpöytäsovellus, jonka asennettua kehittäjä kirjautuu omalle Mendix-käyttäjälleen. Mendix Studio Pro:sta kehittäjä voi avata visuaalisia työkaluja käyttäen haluamansa projektin tai luoda uuden projektin. Studio Pro sisältää työkaluja käytännössä kaikkiin sovelluksen ulkoasun ja toimintalogiikan toteuttamiseen vaadittaviin ominaisuuksiin ja vaiheisiin. Studio Pro:ssa voidaan myös toteuttaa testausta yksikkötestien muodossa. Muita Studio Pro:n mahdollistamia testauksen keinoja ovat muun muassa automaatiotestaus sekä käyttäjätestaus. Kun sovellusta kehitetään Mendix Studio Pro:ssa, sovelluksen tarkastelussa voidaan hyödyntään joko verkkoselainta tai mobiililaitetta. Mobiililaitteella kehitettävää sovellusta tarkastellaan Mendix Make It Native -mobiilisovellusta käyttäen. Kehitysvaiheessa sovellusta tarkasteltaessa joudutaan usein konfiguroimaan uusi sovelluskohtaisesti räätälöity Mendix Make It Native -sovellus. Räätälöity sovellus joudutaan konfiguroimaan, koska Mendixillä rakennettujen sovellusten komponentit tulee sisällyttää myös Make It Native -sovellukseen.

Konfiguraatioprosessi on kehittäjän onneksi suureksi osaksi automatisoitu. Kehittäjän tarvitsee vain välittää Mendix Studio Pro:lle tarpeelliset resurssien osoitteet ja avaimet. Studio Pro generoi automaattisesti resurssien avulla uuden Make It Native -sovelluksen, joka voidaan asentaa joko lähiverkon kautta tai suoraan siirtämällä sovelluksen asennustiedosto mobiililaitteelle.

Mendix Studio on suunniteltu selkeästi visuaalisempaan työhön, jossa kansalaiskehittäjä työstää sovellusta riittävän monimutkaisella, mutta silti yksinkertaisella työkalulla. Kansalaiskehittäjä pystyy toteuttamaan omia ideoitaan ominaisuuksista teknologialle ominaisesti käytössä olevilla käyttöliittymäelementeillä. Sovelluksen käyttöliittymäkehitys tapahtuu Mendix Studiossa WYSIWYG(What You See Is What You Get)-tyylisellä työkalulla. Kansalaiskehittäjä näkee heti miltä sovellus tulee näyttämään, eikä hänen tarvitse huolehtia sovelluksen rakenteesta. Mendix Studio Pro ei ole työkaluna yhtä selkeä visuaalisesti, sillä sen on tarkoitus olla enemmänkin sovelluksen rakenteellisen ja funktionaalisen toiminnan kannalta kykenevä. Mendix Studio Pro:ssa on kaksi eri kehitystilaa, joista toinen on rakenteellinen ja toinen enemmän Mendix Studiota muistuttava. Mendix Studio Pro:ssa on huomattavasti enemmän mahdollisuuksia toteuttaa sekä toimintalogiikkaa että käyttöliittymän räätälöintiä. Tutkielman sovellusprojektissa käytettyjä Studio Pro:n työkaluja esitellään myöhemmin. Seuraavaksi käsitellään Mendix low-code-alustan täyden teknologiapinon toteuttamisen ja konfiguroitavuuden kyvykkyyttä.

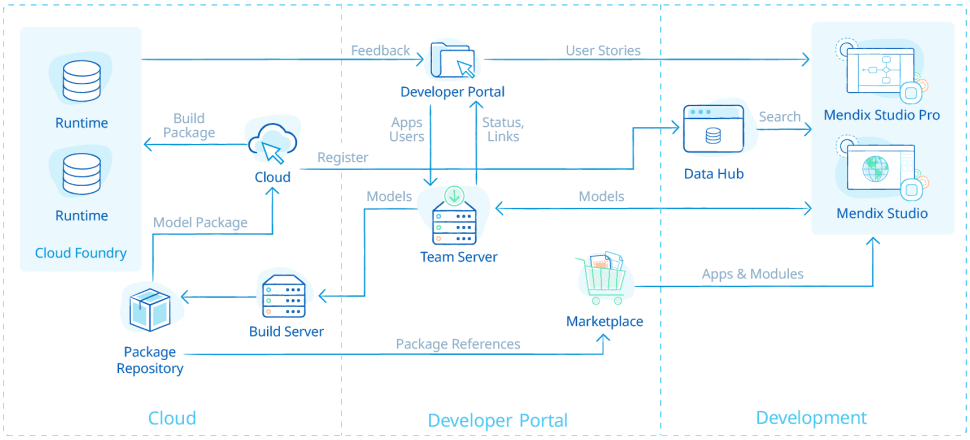
2.2.3 Kyvykkyys toteuttaa täysin toimiva sovellus

Mendix-sovellukset ovat periaatteessa samanlaisia kuin perinteisemmilläkin kehitystyökaluilla tuotetut sovellukset. Ne koostuvat käyttöliittymästä, toimintalogiikasta ja tietokannasta. Sovellusta voidaan suorittaa joko mobiililaitteella paikallisesti tai palvelimella. Palvelinvaihtoehdot Mendix-sovelluksille ovat erittäin kattavat. Yritykset ovat usein kiinnostuneita hallinnoimaan omia palvelimiaan, mikä antaa Mendixille etulyöntiaseman monia muita low-code-alustoja vastaan. Mendix-sovellus voidaan julkaista kaikkiin laajalti käytössä oleviin pilvipalveluihin, kuten Amazon Web Servicesiin tai Google Cloud Platformiin. Pilvipalveluiden lisäksi Mendix-sovellus voidaan julkaista myös "on-premises"eli asiakkaan paikalliselle palvelimelle. Tämä mahdollistaa Mendixin hyödyntämisen arkaluontoista tietoa kä-

sittelevien yritysten ja valtioiden hallintoelinten toimesta. Eri julkaisumuodot tuovat mukanaan luonnollisesti tiettyjä rajoitteita eri ominaisuuksien käyttöön. Esimerkiksi pilvipalveluihin nojautuvat Mendixin osat eivät luonnollisesti ole täysin käytettävissä "on-premies"-ratkaisuissa. Mendix vastaa tähän haasteeseen tarjoamalla asiakkailleen korvaavia työkaluja samankaltaisten toiminnallisuuksien hyödyntämiseen. Yksinkertaisin ja parhaiten tuettu arkkitehtuuriratkaisu Mendixiä käytettäessä on Mendixin hallinnoima pilviratkaisu, joka voi olla joko julkinen tai yksityinen. Mendixin ratkaisumahdollisuudet ovat siis melko joustavat. Tässä tutkielmassa hyödynnettiin vain Mendixin ilmaispakettia, johon kuuluu yksi pilviresurssi, johon kehityksessä oleva sovellus tallentuu. Seuraavassa kappaleessa tutustutaan sovellusten arkkitehtuuriseen rakenteeseen pintapuolin.

2.2.4 Sovellusten arkkitehtuurinen rakenne

Mendixillä rakennetut sovellukset muistuttavat paljon mikropalveluita. Mendix-sovellukset koostuvat käyttöliittymästä, toimintalogiikasta ja tietokannasta. Kaikki osat voidaan mallintaa käyttäen Mendix Studiota tai Mendix Studio Pro:ta. Sovelluksissa ei tarvitse hyödyntää kaikkia ominaisuuksia, vaan sovelluksista voidaan myös tehdä vain taustalogiikkaa sisältäviä mikropalveluita. Mendixin laajat integraatiomahdollisuudet mahdollistavat helposti rajapintojen muodostamisen ja tietyn toimintalogiikan suorittamisen päätepistettä kutsuttaessa. Sovelluksen eri osat erotellaan moduulien avulla. Sovelluksiin luotavia näkymiä kutsutaan sivuiksi. Toimintalogiikka haarautuu palvelimella suoritettaviin microfloweihin, päätelaitteella suoritettaviin nanofloweihin sekä workfloweihin. Jokaisen sivun, microflowin, nanoflowin, workflowin sekä tietokannan palasen näkyvyysasetukset ja käyttöoikeudet voidaan määrittää kunkin ominaisuuksia määritettäessä. Turvallisuus ja näkyvyysasetuksia voidaan myös tarkastella ja määrittää moduulitason näkymän kautta. Mendix low-code-alusta sisältää kuvion 3 mukaisesti pilviarkkitehtuuria, verkkoselainpohjaisen Developer Portalin sekä kehitysympäristön.



Kuvio 3. Mendix low-code-alustan syvällinen arkkitehtuuri, lähde: “Platform Evaluation Guide” (2023)

2.2.5 Tietomalli ja tietokannat

Tietomalli eli Data Model on Mendix-sovellusten ydin. Kaikkialla sovelluksessa käytetään tietomallia. Jokaiselle Mendix-sovellukselle generoituu automaattisesti sovelluksen sisäinen relaatiotietokanta. Jotta tietokantaan voi tallentaa dataa, perinteisesti sinne tarvitaan tauluja. Myös Mendix-sovellukset vaativat taulun kaltaisen rakenteen, mutta Mendixillä tehdessä niitä kutsutaan entiteeteiksi (entity). Entiteetille voi määrittää attribuutteja, joille voidaan määrittää laaja kirjo eri asetuksia. Attribuutit vastaavat perinteisessä tietokannassa taulun sarakkeita. Ne voivat olla joko staattisia tai laskettuja. Jos attribuutti on laksettu, sen arvo määritetään jokaisessa arvoa käyttävässä widgetissä erikseen, joten laskettujen attribuuttien käyttö on harvinaista. Vähäinen käyttö johtuu laskennan tarpeesta jokaisen widgetin kohdalla, joka aiheuttaa helposti suorituskykyongelmia. Tietomalli mallinnetaan UML:ää(Unified Modeling Language) muistuttavalla notaatiolla. Tietokantojen taulut ovat usein listattuna tekstimuodossa perinteisessä sovelluskehityksessä, kun taas Mendixillä ne ovat välittömästi visuaalisessa helposti ymmärrettävässä muodossa. Tämä mahdollistaa liiketoiminnan ja kehittäjien paremman keskustelun aiheesta.

2.2.6 Käyttöliittymäkehitys

Käyttöliittymät rakentuvat viiden kerroksen varaan. Jokainen kerros vastaa sovelluksen sivujen yleistettävyyden tasosta. Mitä ylempänä kerros on, sitä useampaan sivuun sillä tulee olemaan vaikutusta. Jos ylempien kerroksien rakenne on heikko, kehitystehokkuus kärsii huomattavasti näkymiä tehdessä. Käyttöliittymän eri kerroksien tärkeys korostuu usein vasta, kun sovellus kasvaa suureksi. Usein sovelluksen ollessa jo suuri on myöhäistä tai vähintään huomattavasti resursseja kuluttavaa korjata eri kerroksien heikko toteutus.

Ylimpänä kerroksena toimii yrityksen sisäinen tyylittelymääritelmä, eli design system. Sen tarkoituksena on mahdollistaa tietyn yrityksen kaikkien sovellusten ulkoasun yhtenevyys. Jokaisessa Mendix-sovelluksessa vakiona design systeminä tulee Atlas 3, joka on modernin ulkoasun mahdollistava pohja sovellusten muotoilulle. Yritys voi jakaa oman design systeminsä sovellusten välillä Mendixin kauppapaikan kautta ilmaiseksi. Seuraavana kerroksena toimii jo näkymiin eli sivuihin liittyvä Layout.

Toisena kerroksena toimii sivujen asettelumääritykset (Layout). Asettelumäärityksillä kerrotaan Studio Pro:lle, millaisia eri alueita sivuilla tulee olemaan. Asettelumäärityksillä voidaan esimerkiksi kehittää mobiililaitteille ja selainpohjaisille sivuille oma yleistettävä ulkonäkö. Jokaisella sivulle valitaan sitä luodessa sen asettelumääritys ja sen voi vaihtaa tarvittaessa vielä myöhemmin. Tällä kerroksella siis määritetään sivujen eri alueiden sijoittaminen. Seuraavalla tasolla lähestytään hieman yksittäisiä komponentteja.

Kolmennella kerroksella sijatsevat sivusapluunat (Page template). Sivusapluunoiden tarkoituksena on tarjota mahdollisuus tallentaa hyväksitodettuja sapluunoita sivuista, joissa tyylit ja käyttöliittymäkomponentit on jo aseteltu ja muotoiltu halutulla tavalla. Tämä säästää huomattavasti aikaa esimerkiksi useiden samankaltaisten lomakkeiden toteuttamisessa. Mendix-alustaan kuuluvia valmiita sivusapluunoita on saatavilla useita jokaiseen projektiin liitettynä. Sivusapluunalla luodut sivut eivät ole pakotettuja käyttämään valmiita tyylejä vaan ne ovat täysin vapaasti muokattavissa sivun luomisen jälkeen. Seuraavana käsitellään toiseksi alinta Mendix-sovelluksen käyttöliittymän kerrosta.

Neljännellä kerroksella rakennuspalikat (Building block) ovat käyttöiittymäkomponenteista muodostettuja ennalta määritelyjä ryhmiä. Sovelluksia rakennettaessa voidaan usein tarvi-

ta jotain tiettyä kombinaatiota käyttöliittymäkomponentteja, jolloin on sovelluskehityksen parhaiden käytänteiden mukaista muodostaa niistä uudelleenkäytettävä joukko. Rakennuspalikat eivät ole toisiinsa sidonnaisia, vaan ne luovat oman instanssinsa ja ovat tarpeen tullen erikseen muokattavissa. Myös rakennuspalikoita on huomattava määrä valmiina jokaisessa projektissa. Seuraavana tarkastellaan viimeistä käyttöliittymän kerrosta.

Alimmalla kerroksella käyttöliittymäkomponentit eli Widgetit ovat Mendixin käyttöliittymäkehityksen perusosia. Jokaisessa Mendix-sovelluksessa on käytössä laaja kattaus widgettejä. Valmiit widgetit riittävät usein sovelluksen tarpeisiin. Jos Mendixissä valmiina olevat widgetit eivät riitä projektiin, niitä voi joko luoda itse tai hyödyntää Mendixin kauppapaikkaa. Jos kehittäjä haluaa luoda omia widgettejä, täytyy ne koodata joko JavaScriptillä tai TypeScriptillä. Widgettien rakentamisessa hyödynnetään React-kirjastoa. Mendix-sovelluksien kehittämisen yhteydessä on kuitenkin harvinaista, että joutuu kehittämään uusia widgettejä, sillä Mendixin valmiskomponentit ja kauppapaikasta löytyvät Mendix-yhteisön kehittämät komponentit riittävät usein kehittämistarpeisiin.

Tässä luvussa käsiteltiin Mendixin käyttöiittymäkehityksen viittä eri tasoa. Ylhäältä alaspäin mentäessä tasot ovat Design system, asettelumääritykset (Layout), Sivusapluunat (Page template), rakennuspalikat (Building block) ja Widgetit. Kun sovellusta kehitetään nämä kaikki huomioonottaen, suurienkin sovellusten kehittämisestä ja ylläpitämisestä tulee huomattavasti helpompaa. Jos tarvitaan lisää widgettejä, voidaan joko kehittää ne itse tai ladata niitä Mendixin kauppapaikasta. Kauppapaikassa ei ole kuitenkaan vain widgettejä vaan myös valtava määrä valmiiksi luotua toimintalogiikkaa. Seuraavassa luvussa käsitellään Mendix-kehitystä toimintalogiikan näkökulmasta.

2.2.7 Toimintalogiikan kehitys

Nykypäivän sovellusten tulee olla enemmän, kuin vain käyttöliittymä, jossa on staattista tietoa. Jotta käyttöliittymään saadaan dynaamista tietoa ja käyttöliittymästä saadaan tietoa muille sovelluksille, tarvitaan toimintalogiikkaa. Mendixissä toimintalogiikkaa on pääosaisesti kolmenlaista. On microfloweja, nanofloweja ja workfloweja. Näiden lisäksi low-code-alusta käsittelee jotain toimintoja automaattisesti, kuten vakiona generoitujen "tallenna"-pai-

nikkeiden tallennustoimintalogiikan. Microflowien ja nanoflowien toimintalogiikka kehitetään Mendix-sovelluksissa graafisen käyttöliittymän avulla BPMN:ää (Business Process Modeling Notation) muistuttavalla notaatiolla. Tämä notaatio on intuittiivinen ja helppo lukea, joten se on juuri sopiva liiketoiminnan ja kehittäjien yhteistyön helpottamiseen. Toimintalogiikkaa voidaan sijoittaa moniin paikkoihin, kuten painikkeisiin. Ensimmäisenä kasitellään microfloweja.

Microflowit ovat selainpohjaisten Mendix-sovellusten yleisin toimintalogiikan muoto. Ne suoritetaan kutsuttuna hetkenä ja niiden elinaika on erittäin lyhyt. Ne koostuvat yksittäisistä aktiviteeteista, jotka voivat olla esimerkiksi rajapintakutsua mallintava toimintalogiikan kokonaisuus. Microfloweista pystytään myös kutsumaan toisia microfloweja, nanofloweja tai workfloweja. Kun microflowia kutsutaan jostain sovelluksen osasta, sille voidaan välittää parametrina esimerkiksi käsiteltävänä oleva tietue. Microflowit eroavat muusta toimintalogiikasta siten, että kun niitä kutsutaan, ne suoritetaan palvelimella eikä päätelaitteella. Kun toimintalogiikkaa suoritetaan palvelimella, sillä ei ole tietoa sen hetkisen käyttäjän sessiosta. Sessiokohtaiseen toimintalogiikan suorittamiseen tarvitaan nanofloweja.

Nanoflowit ovat natiivisovellusten yleisin toimintalogiikan muoto. Nanoflowien toiminta on hyvin samanlaista, kuin microflowienkin, mutta niissä on joitain eroja. Kun microflowia kutsuttaessa toimintalogiikka suoritettiin palvelimella, nanoflowia kutsuttaessa toimintalogiikka suoritetaan käyttäjän päätelaitteella, joko mobiililaitteella tai verkkoselaimessa. Nanoflowit sisältävät myös erilaisia mahdollisia aktiviteetteja microfloweihin verrattuna. Nanofloweissa on mahdollista hyödyntää käyttäjän laitteesta saatavaa laajaa datan kirjoa. Käyttäjän laitteen dataa hakevia aktiviteetteja on muun muassa kameran käyttö ja gps-lokaation haku. Selainpohjaisten sovellusten ja mobiilisovellusten välillä on eroja käytössä olevissa aktiviteeteissä. Nanoflowien suorittamisessa kestää myös hyvin vähän aikaa. Seuraavaksi käsitellään Mendixin uusinta toimintalogiikan muotoa, eli workfloweja.

Omia kehitysmuutoksia puskettaessa versionhallintaan perinteisessä ohjelmoissa on hyvä käytänne luoda ”pull request”. Kun se on lähetetty, se jää odottamaan toisen henkilön tarkastelua ja sitten hyväksyntää tai hylkäystä. Workflowien ideana on mahdollistaa ”pull requestin” tyylisen toimintalogiikan ja käyttäjän yhteistyö. Workflowit ovat siis pitkäikäisiä toimintalogiikan palasia. Ne eroavat suurelta osin microfloweista ja nanofloweista. Workflowien

tarkoituksena on toimia käyttäjän syötettä vastaanottavana toimintalogiikkana, jolla voidaan automatisoida joitain osia prosessien varrella. Workfloweja rakennetaan yksinkertaistetummalla mallilla, jossa edetään ylhäältä alaspäin vaihe vaiheelta. Sen kehityskäyttöliittymä on intuitiivinen. Mendixissä on siis monia eri toimintalogiikan kehitykseen soveltuvia palasia. Jokaiselle on oma paikkansa ja oikein käytettynä niiden avulla saadaan tehtyä sovelluksesta hyödyllinen käyttäjälle. Pitkäikäistä ihmisen syötettä kuuntelevaa toimintalogiikkaa on workflowit. Lyhytikäistä suoritushetkellä saatavilla olevaa dataa hyödyntävää toimintalogiikkaa ovat microflowit ja nanoflowit. Microflowit suoritetaan palvelimella, kun taas nanoflowit käyttäjän laitteella. Toimintalogiikan, tietomallin ja käyttöliittymien lisäksi tarvitaan vielä muita resursseja.

2.2.8 Resurssien luonti ja konfigurointi sekä rajapintakutsut

Tässä luvussa tullaan käsittelemään Mendix-sovellusten luomisessa olennaisimpia resursseja ja niiden konfigurointia. Näitä resursseja voivat olla esimerkiksi Mendix-sovellusten sisäisesti määritellyt rajapintakuvaukset, joiden perusteella Mendix pystyy hakemaan dataa rajapinnoista ja tajoilemaan dataa sitä pyytäville palveluille. Muita resursseja ovat esimerkiksi sovelluksissa käytettävät kuvat sekä erilaiset tiedostotyypit kuten JSON-tiedostot. Suurin osa resursseista luodaan Mendix Studio Pro:ssa. Resurssien luonti kuvia lukuunottamatta on usein ammattilaiskehittäjien vastuulla. Käsitellään tarkemmin moniin moderneihin sovelluksiin mukaan lukeutuvien rajapintayhteyksien luontia eli rajapintaintegraatioita.

Rajapintakutsut ovat olennainen osa nykypäivänä toteutettavia moderneja sovelluksia. Rajapintakutsuilla voidaan hakea ja lähettää dataa. Rajapinta vastaanottaa kutsun ja käsittelee sen rajapinnan oman toimintalogiikan mukaisesti. Tämän jälkeen rajapinta lähettää kutsun lähettäjälle vastauksena tietoa kutsun onnistumisesta ja tarvittaessa vastaukseen liitettyä dataa, jota kutsuja voi hyödyntää. Mendix-sovelluksissa rajapintakutsut on yksinkertaistettu helppokäyttöisiksi konfiguroitaviksi resursseiksi. Ne vaativat toimiakseen kuvauksen lähetettävästä ja vastaanotettavasta esimerkkidatasta, sekä Mendixin tietomalliin esimerkkidatan pohjalta automaattisesti generoidun tietorakenteen. Tietorakenteen avulla Mendix-sovelluksissa voidaan visuaalistaa data käyttöliittymään ja halutessa myös tallentaa se Mendix-sovelluksen tietokantaan. Mendix-sovelluksissa voidaan hyödyntää joko JSON- tai XML-malli-

sia tietorakenteita rajapintakutsuissa, sekä vastaanotettavan datan käsittelyssä. Rajapintakutsuja Mendix-sovelluksissa voidaan muodostaa joko REST-rajapintaintegraatioita, julkaista SOAP-pohjaisia rajapintoja tai hyödyntää OData-standardisoitua mallia REST-rajapinnoista. ODataa hyödyntävät REST-rajapinnat on helppo integroida esimerkiksi Microsoftin palveluihin, kuten Exceliin ja PowerBI:hin. Rajapintaintegraatioita voidaan siis luoda hyvin joustavasti. Rajapintaintegraatioiden hyötyjä pystytään lisäämään dokumentaatiolla.

Rajapintaintegraatiot ovat yksi Mendixin suurimmista vahvuuksista. Niiden luonti on yksinkertaista ja joustavaa. Mendix-sovelluksiin päin kohdistuvien rajapintaintegraatioiden luontia nopeuttaa huomattavasti Mendixin automaattisesti generoima ja tarjoama Swagger-rajapintakuvaus jokaiselle luodulle rajapinnalle. Laajalti käytössä olevasta Swagger-rajapintakuvauksesta saadaan tieto rajapinnan toiminnasta, esimerkkidatasta, sekä mahdollisista rajapintaa koskevista rajapinnan päätepisteistä. Rajapinnan selkeä dokumentaatio helpottaa huomattavasti rajapinnan käyttöönottoa muista sovelluksista käsin. Mendix-sovelluksen rajapintojen sisältämää toimintalogiikkaa voidaan muokata huomattavasti.

Mendix-sovelluksilla toteutettujen rajapintaintegraatioiden yhteyteen voidaan liittää myös täysin räätälöityä toimintalogiikkaa microflowien avulla. Toimintalogiikkaa voidaan hyödyntää esimerkiksi datan parsimisessa ja mahdollisesti jopa uusien rajapintakutsujen lähettämisessä tarpeellisen datan kutsujalle takaisinlähetettäväksi. Toimintalogiikan laajentaminen on myös mahdollista perinteistä koodia hyödyntämällä. Microfloweihin voidaan liittää Javalla ohjelmoitua toimintalogiikkaa ja nanofloweihin voidaan liittää JavaScriptillä tai TypeScriptillä ohjelmoitua toimintalogiikkaa. Toimintalogiikkaa voidaan putkittaa esimerkiksi aloittamalla toimintalogiikka microflowilla, josta kutsutaan nanoflowia, josta taas kutsutaan JavaScript-funktiota.

Mendix-sovelluksessa resursseilla siis laajennetaan sovelluksen sisäistä toimintaa, sekä tuotetaan uusia sovelluksen ulkopuolelle ulottuvia yhteyksiä. Sovellukseen tuleva sekä sovelluksesta poistuva tieto tulee muotoilla oikean muotoiseksi datamalleja kuvaavilla JSON- tai XML-resursseilla. Sovelluksen muun toiminnallisuuden ja ulkoasun laajentaminen tapahtuu myös resurssien, kuten kuvien avulla. Kaikkia resursseja ei ole mahdollista luoda Mendix Studiossa, sillä joidenkin resurssien konfigurointi on kansalaiskehittäjille liian haastavaa. Seuraavassa luvussa tullaan tarkastelemaan Mendix-sovellusten käyttöliittymien ja toimin-

talogiikan eroavaisuuksia sovelluksen käyttöalustasta riippuen.

2.2.9 Verkkoselainpohjaisten -ja natiivien sovellusten erot

Perinteisillä ohjelmoinnin työkaluilla voidaan yleisesti toteuttaa vain joko verkkosovelluksia tai natiiveja sovelluksia. Moderneimmat ohjelmointityökalut ja kehitysympäristöt kykenevät tuottamaan yhdestä koodipohjasta sekä verkkosovelluksen että natiivin mobiilisovelluksen. Tässä luvussa tarkastellaan Mendixin kyvykkyyksiä tuottamaan verkkosovelluksia sekä natiiveja mobiilisovelluksia. Mendixillä myös mobiilisovellusten ja verkkosovellusten tuottaminen eroaa hieman perinteisistä ohjelmistokehitysmenetelmistä.

Yhdessä Mendix-sovelluksessa voi olla sekä natiivi mobiilikäyttöliittymä että verkkoselainpohjainen käyttöliittymä. Ne täytyy kehittää erikseen, mutta itse sovelluksen ei tarvitse sijaita eri projektissa. Mobiili- ja verkkopohjaisille näkymille on omat erilliset sivut, jotka sisältävät eri vaatimuksia esimerkiksi sovelluksen toimintalogiikasta. Mobiililaitteiden käyttäjien vaatimukset sovelluksilta ovat usein laajemmat, kuin verkkosovellusten käyttäjien. Yksi suurimmista eroavaisuuksista on verkkosovelluksen oletettu jatkuva verkkoyhteys. Mobiilisovellusta käytettäessä käyttäjä voi olla paikassa, jossa verkkoyhteys on heikko. Mobiilisovellusten kehittämisessä tulee siis olla olennaisesti mukana ajatus sovellusten toiminnasta myös verkkoyhteyden katkeamisen yhteydessä. Tämän vuoksi käyttöliittymän rakentaminen mobiilisovellukselle asettaa kehittäjälle enemmän vaatimuksia sovelluksen toiminnan varmistamiseksi. Mobiillikäyttöliittymiin ei voi esimerkiksi hakea tietoa suoraan tietokannasta, vaan se täytyy ensin joko tuoda laitteen paikalliseen SQLite-tietokantaan tai tallentaa laitteen muistiin kyseisen näkymän olemassaolon ajaksi. Verkkosovelluksen kohdalla tilanne on lähes päinvastainen. Verkkosovellukset nojaavat huomattavasti suoraan tietokantaan, josta haetaan ja sinne voidaan päivittää tietoa suoraan sovelluksen käyttöliitymien sisältävästä toimintalogiikasta. Tietokantamahdollisuudet ovat molemmissa vaihtoehdoissa samat. Verkkopohjaisten ja natiivisti mobiilien sovellusten käyttöliittymien ja toimintalogiikan kehittäminen eroaa myös räätälöidyn kehitystyön osalta.

Räätälöityjen käyttöliittymäelementtien, eli widgettien kehittäminen verkkosovelluksiin toteutetaan JavaScriptin React-kirjastoa käyttäen. Mobiilisovelluksiin widgettejä kehitettäes-

sä ohjelmointikielenä käytetään edelleen JavaScriptia, mutta kirjastona toimii React Native. Toimintalogiikan kehittäminen on mahdollista sekä mobiili- että verkkosovelluksien osalta Javalla ja JavaScriptillä. Räätälöidyn toimintalogiikan lisäksi mobiilikäyttöliittymäkehityksessä on olemassa tiettyjä rajoituksia dataa palauttavan toimintalogiikan osalta. Mobiilikäyttöliittymissä itse kehitetyn toimintalogiikan hyödyntämiseen pystyy käyttämään vain nanofloweja. Widgettien osalta käyttöliittymäkehityksessä on käytössä lisänä laitteen natiiviominaisuuksia hyödyntäviä widgettejä, kuten laitteen kameralla kuvan ottaminen ja laitteen paikallistaminen. Mobiili- ja verkkoselainpohjaisia käyttöliittymiä kehittäessä kaikkia samoja widgettejä ei ole mahdollista käyttää, sillä osa verkkoselainpohjaisista widgeteistä on tarkoitettu käytettäväksi jatkuvan verkkoyhteyden kanssa.

Mobiili- ja verkkosovellusten erot painottuvat siis suurimmaksi osaksi käyttöliittymän ja toimintalogiikan kehittämiseen. Mobiilikäyttöliittymän sivut ovat eri mallisia, kuin verkkosovelluksen sekä käytettävät JavaScript-kirjastot eroavat mobiileissa ja verkkoselainpohjaisissa sovelluksissa. Keskittymispiste sovelluksen suunnittelun tasolla mobiilikehityksessä tulee olla verkkoyhteyden epävakauden siedettävyydessä. Sovellusten sunnittelua avustaa tuntemus sovelluksen arkkitehtuurista. Seuraavassa luvussa käsitellään Mendix-sovellusten arkkitehtuuria.

2.2.10 Sovelluksen arkkitehtuuri

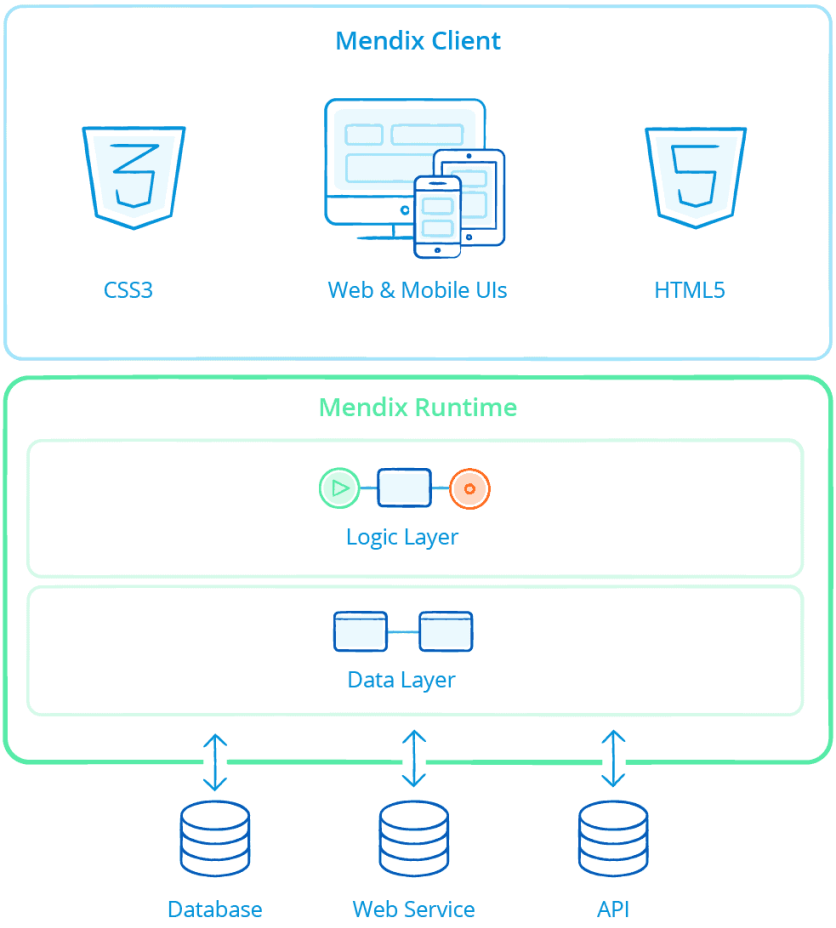
Mendix-sovellusten arkkitehtuuri mahdollistaa mikropalveluiden kaltaiset sovellukset sekä monoliittiset sovellukset. Mendix-sovellukset koostuvat kahdesta pääosasta. Nämä osat ovat Mendix Client ja Mendix Runtime. Nämä kaksi Mendix-sovellusten pääosaa erottuvat fyysisesti käyttäjän laitteen ja palvelimen suorittamiin toimintoihin. Ensin tarkastellaan käyttäjän laitteella tapahtuvia toimintoja.

Käyttäjän päätelaitteen vastuulla on kaikki asiat, jotka ovat on kytköksissä tiettyyn sovelluksen käyttöinstanssiin eli sessioon. Session aikana käyttäjän laite kommunikoi palvelimen kanssa HTTP-protokollan turvin. Käyttäjän päätelaitteella suoritettavat toiminnot voidaan jakaa neljään eri kerrokseen. Nämä kerrokset ovat kommunikaatio-, data-, logiikka-ja käyttöliittymäkerros. Sovelluksen kehittäjälle tärkeimmät osat näistä kerroksista ovat kom-

munikaatiokerroksen sessiotieto, datakerroksen offline-tiedon tallennus ja logiikkakerroksen nanoflowien suorittaminen. Arkkitehtuurisesti päätelaite voi olla hetkellisesti ideaalissa tilanteessa täysin riippumaton Mendix Runtimestä. Mendix-sovellus tarvitsee kuitenkin usein huomattavasti enemmän toiminnallisuuksia, kuin vain laitteen paikalliset kyvykkyydet mahdollistavat.

Mendix Runtime käsittelee huomattavasti suuremman osan sovelluksen toiminnallisuudesta, kuin Mendix Client. Runtime käsittelee esimerkiksi Mendix Studio Pro:ssa hyvin samankaltaisia nanofloweja muistuttavia, mutta silti sessioriippumattomia microfloweja. Runtimen vastuulla ovat myös tietokantayhteydet, rajapintaintegraatiot sekä monet muut päätelaitteesta ja Mendix Clientistä riippumattomat toiminnallisuudet. Myös sovelluksen ulkopuolisen kommunikaation turvallisuus varmistetaan Mendix Runtimessä, jotta kehittäjän ei tarvitse huolehtia siitä itse. Se käsittelee siis käytännössä kaiken sovelluksen toiminnallisuuden, jota ei päätelaitteella ole kannattavaa käsitellä.

Kuviossa 4 Mendix Client ja Mendix Runtime ovat muodostavat yhdessä verkkoyhteydellisen turvallisen kokonaisuuden. Mendix-sovellusten arkkitehtuuri ei näy kehittäjälle kuin joissain kehittyneissä sovelluskehityksen tapauksissa, joissa esimerkiksi tieto sessiosta on oleellista. Arkkitehtuuria ei tässä tutkielmassa tämän vuoksi tarkastella lähemmin. Seuraavaksi tarkastellaan Mendix Studioon ja Mendix Studio Pro:hon integroitua sisäistä versionhallintaa.



Kuvio 4. Mendix-sovelluksen Mendix Runtime ja Mendix Client arkkitehtuuri, lähde: “Platform Evaluation Guide” (2023)

2.2.11 Versionhallintatyökalut

Versionhallintatyökalut kuten Git ovat yleisesti käytössä modernissa sovelluskehityksessä. Mendix Studio Pro:ssa on mahdollista hyödyntää siihen integroitua SVN-pohjaista versionhallintaa. Kuten monet muut toiminnallisuudet Mendix Studio Pro:ssa, myös versionhallinta toteutuu visuaalisen käyttöliittymän avulla. Tässä luvussa tarkastellaan Mendix Studio Pro:hon integroidun versionhallinnan eroavaisuuksia perinteisten sovelluskehitysprojektien käyttämiin versionhallintatyökaluihin. Ensimmäisenä tarkastellaan Mendix Studio Pro:n versionhallintatyökalua.

SVN-pohjainen versionhallinta mahdollistaa projektin versionhallinnan sekä Mendix Studio Pro:hon integroidulla työkalulla että kolmannen osapuolen työkaluilla, kuten Tortoise SVN:llä. Studio Pro:hon integroitua versionhallintatyökalua käyttäen voidaan tehdä versionhallinan perustoiminnot, mutta toiminnallisuus ei ole aivan yhtä laaja kuin esimerkiksi Git:ssä. Projekteille voidaan tehdä omia haaroja, niitä voidaan poistaa ja niitä voidaan yhdistää toisiinsa. Kehittäjät voivat kommentoida muutoksiaan niitä haaraan lisätessään. Mendix Studio Pro:hon yhdistetty versionhallinta eroaa perinteisesti käytössä olevasta Git:stä esimerkiksi muutostenkäsittelyn ja niiden tarkastamisen mahdollistavan ”Pull Request” -toiminnon puutteena. Tämä ero vaatii kehitystiimin yhteisten käytänteiden muodostamista projektin ylläpidettävyyden varmistamiseksi. Mendixin dokumentaatiossa ei ole ohjeita muutosten oikeellisuuden varmistamiseksi.

Kolmannen osapuolen versionhallintatyökalujen, kuten Tortoise SVN:n käyttäminen on joskus jopa pakollista. Esimerkiksi haarojen yhdistämisessä vastaan tulevissa konfliktitilanteissa, joissa kehittäjän muutokset ylikirjoittavat toisen kehittäjän tekemiä räätälöityjä tyylimuutoksia sovellukseen. Muitakin kolmannen osapuolen työkaluja voidaan integroida Mendix-sovelluksien versionhallintatyökaluiksi. Projekteissa voidaan hyödyntää esimerkiksi Jenkins-automaatio ohjelmistoa, jolla voidaan viedä projektin uusia versioita esimerkiksi kehitysympäristöstä testausympäristöön.

Mendix Studio Pro:hon integroidut versionhallintatyökalut mahdollistavat yksinkertaisen tavan hallita sovelluksen eri versioita. Integroidut työkalut eivät kuitenkaan aina riitä, vaan joskus tarvitaan käyttöön myös kolmannen osapuolen työkaluja. Kolmannen osapuolen versionhallinnan työkaluja on kuitenkin mahdollista integroida Mendix-projekteihin. Seuraavassa luvussa tarkastellaan Mendix-projekteissa hyödynnettäviä projektinhallintatyökaluja.

2.2.12 Projektinhallintatyökalut

Projektinhallintatyökalujen avulla saadaan helpotettua projektin dokumentaatiota ja sen laadullista varmistamista. Ohjelmistoprojekteihin on mahdollista sisällyttää nykypäivänä valtava määrä erilaisia projektinhallintatyökaluja. Sovelluskehityksen tukena voidaan hyödyntää esimerkiksi käyttötapauslistoja tai käyttötapausten visuaalisempaan tarkasteluun tarkoitet-

tuja kanban-tauluja. Mendix-projekteissa on mahdollisuus käyttää molempia vaihtoehtoja. Tässä luvussa tullaan kertomaan Mendix-projekteissa avustavasta Mendixin omasta projektinhallintatyökalusta.

Mendix on rakentanut ja integroinut oman projektinhallintatyökalunsa minimoidakseen Mendix-ekosysteemiin projektien aloittamisen vaatiman työn määrän sekä myös dokumentoimisen helpottamiseksi. Projektinhallintatyökalu on perinteinen Scrum-metodologiaa noudattava käyttötapauksienhallintatyökalu. Sillä voidaan luoda uusia käyttötapauksia. Sillä voidaan luoda käyttötapauksia kategorisoimaan luokittelulappuja eli labeleita. Käyttötapauksiin voidaan lisätä otsikko, kuvaus ja mahdollisia käyttötapauksen valmistumisen vaativia tehtäviä. Käyttötapaukselle voi asettaa myös tilan indikoimaan käyttötapauksen työvaihetta. Käyttötapauksia ei voida linkittää toisiinsa, joten työkalu on yksitasoinen. Projektinhallintatyökalu on kuitenkin suoraan integroitu Mendixin työkaluihin.

Mendix Studio:sta ja Mendix Studio Pro:sta käsin projektin käyttötapauksia työstäessä kehittäjä näkee samassa käyttöliittymässä myös käyttötapaukset. Kehittäjä voi muuttaa käyttötapausten tilaa ja kommentoida kyseistä käyttötapausta molemmissa kehitysympäristöissä. Jos käyttötapauksia halutaan hallinnoida jollain toisella projektinhallintatyökalulla, sen integroiminen Mendixiin on usein mahdollista. Mendixin projektinhallintatyökaluihin kuuluu myös projektin jäsenten hallintaa varten oleva erillinen näkymä. Sieltä voidaan hallita muun muassa projektin jäsenten oikeuksia, muokata sovelluksen koodia, käyttötapauksia tai julkaisuja. Projektin eri ympäristöjen oikeuksien hallinta tapahtuu myös tästä työkalusta.

Projektinhallintatyökalujen ja versionhallintatyökalujen yhdistelmänä Mendix-sovelluksien eri ympäristöjä voidaan hallita sekä manuaalisesti että automatisoidusti. Manulaalisesti voidaan julkaista uusi versio Mendix-sovelluksesta haluttuun ympäristöön yhdellä napin painalluksella. Automatisaatiossa voidaan hyödyntää esimerkiksi Jenkins-työkalua, jolla voidaan siirtää projekti esimerkiksi kehitysympäristöstä testiympäristöön, ja jos vaadittavat ohjelmistotestit menevät läpi, julkaista sovellus jopa tuotantoon asti. Mendix-projektien hallintaa varten on siis monia työkaluja, jotka helpottavat asiakkaan ja kehitystiimin kommunikaatiota sekä selventää vaatimusten määrittelyn tuloksia.

2.3 Kirjallisuuskatsauksen yhteenveto

Low-code on ajatusmallina vanha, mutta terminä uusi. Terminä low-code on mainittu ensimmäistä kertaa vuonna 2014 (Sanchis ym. 2019), mutta se esiintyy vertaisarvioiduissa julkaisuissa vasta vuodesta 2018 eteenpäin (Bucaioni, Cicchetti ja Ciccozzi 2022). Sen määritelmä on vielä erittäin hatara, eikä se ole varsinaisesti vielä vakiintunut. Low-coden juuret ulottuvat hyvin pitkälle sovelluskehityksen historiaan. 1980-luvulta 2000-luvulle kulkiessa low-coden ajatusmallin ja työkalujen on viitattu kirjallisuudesssa olevan hyvin samankaltaisia CASE-, RAD- ja MDD/MDE-työkalujen ja ajatusmallien kanssa (Ruscio ym. 2022; Lundell ja Lings 2004; Beranic, Rek ja Heričko 2020). RAD on ollut low-codelle lähde monille sen periaatteille. Low-code voi alkaa vaikuttaa monille vanhalta asialta, kun vanhasta ajatusmallista on otettu paljon vaikutteita. Low-coden uutuusarvoa onkin kritisoitu. Jotkin lähteet väittävät jopa, että low-code ei tuo mitään uutta sovelluskehityksen kentälle (Bock ja Frank 2021). Low-codella on kuitenkin käyttökohteensa.

Low-code-teknologian sopivuus tietylle kohdealueelle riippuu hyvin pitkälti itse kohdealueesta ja low-code-teknologian käyttämästä DSL:stä (Galhardo ja Silva 2022). Yleisimmin low-codea on kehuttu dynaamisten, lomakepohjaisten sovellusten toteuttamisen tehokkuudesta. Kirjallisuuden mukaan low-code soveltuu hyvin pienten ja vähän toimintalogiikkaa sisältävien sovellusten toteuttamiseen (Totterdale 2018). Laajojen sovellusten toteuttamisesta ei löytynyt lähdekirjallisuudesta tutkimustietoa. Jotkin low-code-alustojen tarjoajat kuitenkin vahvasti sovellusten skaalautuvuuteen (“Platform Evaluation Guide” 2023), joten on ainakin niillä on ainakin mahdollista rakentaa suuremman kokoluokan projekteja.

Eri low-code-alustat sopivat eri käyttötarkoituksiin ja niillä on omat vahvuutensa sekä heikkoutensa. Tutkielmassa low-code-teknologioista tarkasteltiin Microsoft Power Platformia, Mendixiä, OutSystemsiä ja Salesforcea. Nämä neljä teknologiaa ovat olleet jo useita vuosia johtajia low-code-alustoina ja niitä tarjoavat yritykset low-code-alustojen tarjoajina (Vincent ym. 2019; Vincent ym. 2022). Näistä neljästä teknologiasta lähemmin tarkasteltiin Mendixiä.

Mendixin teknistä sisältöä avattiin tutkielman konstruktivistisen osan kannalta tärkeistä osista. Sovellusalustana Mendix on yleiskäyttöinen low-code-alusta, jota markkinoidaan PaaS-tyylisenä palvelumuotona (“Platform Evaluation Guide” 2023). Mendix-projektia muoka-

takseen kehittäjän ei tarvitse suorittaa paljon aikaa vieviä konfiguraatioita vaan kehittäjä vain lataa ja asentaa työpöytäsovelluksen oikean version tietokoneelleen. Mendix sisältää hyvin pitkälti kaikki tarvittavat ominaisuudet kehittää asiakaskäyttöön ja yrityskäyttöön soveltuvia luotettavia ja ylläpidettäviä ohjelmistoja. (“Platform Evaluation Guide” 2023)

3 Mobiilisovelluksen MVP-version rakentaminen Mendix low-code-sovellusalustalla

Tässä osassa analysoidaan kehitystyöhön käytetyn low-code-sovellusalustan hyötyjä ja haittoja. Tutkielman konstruktivistisen osan tavoitteena on vastata tutkimuskysymyksen toiseen osaan Mendix low-code-alustan käytön hyödyistä ja haitoista. Sovelluksen kehittäminen aloitettiin keväällä 2022 ja sovellus valmistui kesällä 2022. Projektin kehittämistyöhön osallistui kolme kehittäjää, joista kokeneimmalla oli Mendix low-code-alustasta kokemusta sovelluksen kehittämisen alkaessa puolitoista vuotta ja vähemmän kokeneilla noin kaksi kuukautta. Sovellus kehitettiin tutkielman tilanneen yrityksen käyttöön. Low-code-teknologiaksi valittiin Mendix kehittäjien kokemuksen ja sovelluksen vaatimusten perusteella. Mendix tukee tehokkaasti ulkoisten rajapintojen integraatioita (Vincent ym. 2019), (Vincent ym. 2022). Sovellus tulisi perustumaan suurelta osin ulkoisten rajapintojen hyöydntämiseen, joten sekä kehittäjien kokemus että Mendixin tuki tekivät teknologian valinnasta yksinkertaisen. Myös yrityksen halu lisätä tietoa Mendixistä vaikutti teknologian valintaan.

3.1 Sovelluksen rakentamisen tarve

Sovelluksen rakentaminen rakentamisen vuoksi ei ole kenellekään hyödyllistä. Tämän tutkielman tapauksessa sovellusta oltiin rakentamassa myös ilman tutkielmaa. Yrityksessä kuitenkin koettiin hyödylliseksi tutkia sovelluksen toteutusta tarkemmin. Yritys halusi myös tietää, mitkä asiat tuntuisivat haastavilta sovelluksen kehittäjien näkökulmasta. Näillä tiedoilla voitaiisin varautua seuraavien projektien kohdalla samankaltaisiin haasteisiin. Voitaisiin myös toteuttaa kestäviä ja yleistettäviä ratkaisuja low-code-alustalla kehittämiseen. Sovelluksen rakentamisen suurimmaksi tarpeeksi nousi sovelluksen edellisen toisella teknologialla toteutetun version ikä ja perinteisillä menetelmillä uuden version rakentamisen vaatiman työn määrä.

Toteutuksen ikä tuo mukanaan monia riskejä ja näitä riskejä haluttiin välttää. Sovelluksen ikä ei varsinaisesti tuo suuria ongelmia, vaan siihen suoraan korreloiva kehitystyökalujen sekä teknologian ikä. Vanhan sovelluksen kehitystyökalut olivat vanhoja, eikä työmarkki-

noilta tulisi tulevaisuudessa helposti löytymään kehittäjiä kyseisille työkaluille. Perinteisesti ohjelmoimalla työmääräarvio koko sovelluksen toteuttamiselle oli kolme henkilötyövuotta, joka on paljon aikaa. Myös vanhan sovelluksen käyttöliittymä oli vanhanaikainen, eikä se palvellut käyttäjää parhaalla mahdollisella tavalla. Low-code-alustan uskottiin helpottavan käyttöliittymän modernisointia ja käyttäjäkokemuksen parantamista.

Vanhan sovelluksen korvaaminen ei ollut varsinaisesti pakollista tutkielman aloitusvaiheessa, mutta sovelluksen korvaamisen vaihtoehtoja haluttiin tutkia ja yhdeksi mahdollisuuksista osoittautui Mendix low-code-alusta. Mendix oli yritykselle uusi teknologia, joten sen tutkiminen projektikäytössä oli tärkeää yrityksen teknologiavalintojen kannalta. Yhdistelmä vanhan sovelluksen työkalujen korvaamisesta, käyttäjäkokemuksen kehittämisestä ja uuden teknologian projektikäytännön tutkimisesta toi low-coden ja erityisesti Mendixin suotuisaan asemaan. Samaan aikaan pystyttiin siis arvioimaan yritykselle uuden low-code-teknologian mahdollisuuksia sekä vanhan mobiilisovelluksen korvaamisen vaatimasa aikaa. Tuloksia odotettiin myös toteutusprosessin eri vaiheiden haastavuudesta. Jotta pystytään luomaan uusi paranneltu versio vanhasta sovelluksesta, tarvitaan tietoa vanhasta. Seuraavassa luvussa kuvataan vanhaa sovellusta ja niitä ominaisuuksia, jotka valittiin tutkielman käyttötapauksiin.

3.2 Kuvaus vanhasta sovelluksesta

Vanhan sovelluksen käyttöympäristö sijaitsi varastorakennuksissa varastonhallinnan kontekstissa. Sen tarkoituksena oli erilaisten varastonhallinnallisten toimintojen helpottaminen käyttäjälle. Varastonhallinnan tehtäviin kuuluu toimintoja, kuten nimikkeiden inventointi sekä työjonojen hallinta. Vanhan sovelluksen osalta tutkielman MVP-sovellukseen haluttiin valita sopivan yksinkertaisia käyttötapauksia, jotta saataisiin tuloksia low-coden kyvykkyyksistä yksinkertaisissa käyttötapauksissa. Käyttötapaukset sisälsivät kuitenkin useiden projektien vaatimia hieman monimutkaisempia ominaiusuuksia, kuten rajapintaintegraatioita. Vanhasta sovellusksesta pääomaiseksi käyttötapaukseksi valittiin varaston pikainventointi.

Vanhan sovelluksen työkulku eteni seuraavasti. Ensin käyttäjä avaa sovelluksen ja kirjautuu käyttäjätunnuksellaan sovellukseen. Sovellus avaa käyttäjälle kotinäkymänä varaston työjo-

non -hallintanäkymän, josta varastotyöntekijä pääsee työstämään hänen varastossaan olevia työta vaativia työjonoja. Käyttäjä avaa hampurilaisvalikon ja valitsee hampurilaisvalikosta pikainventoinnin. Pikainventoinnin kotinäkymä avautuu. Käyttäjä valitsee seuraavaksi varaston, johon hän haluaa tehdä muutoksen. Sen jälkeen hän valitsee varastopaikan, jossa hänen etsimäänsä nimikettä on. Varastopaikan valinnan jälkeen varastotyötekijä valitsee listasta nimikkeen. Varaston, varastopaikan ja nimikkeen valitseminen on mahdollista myös viivakoodinlukijalla. Valittuaan nimikkeen varastotyöntekijä pääsee muokkaamaan tuotteen tietoja, sekä syöttämään kommentin muokkaukselle selitteen muodossa. Tämän jälkeen käyttäjä tallentaa muutoksen ja hänet palautetaan pikainventoinnin etusivulle. Käyttäjä toistaa pikainventoinnin askelia niin kauan, kuin on tarvetta. Kun käyttäjä on suorittanut kaikki tarvittavat pikainventoinnit, hän voi joko kirjautua ulos sovelluksesta tai vain sulkea sovelluksen. Kirjautumistietoja hallitseva rajapinta aikakatkaisee kirjautumisen tietyn ajan jälkeen.

Varastonhallinnan kontekstissa pikainventoinnin tarkoituksena on mahdollistaa nimikemäärien muuosten tekemisen nopeasti ja vaivattomasti. Mahdollisia tilanteita kyseisen käyttötapauksen tarpeelle on esimerkiksi tuotteiden rikkoutuminen. Pikainventoinnin sisälsi käyttötapauksia, kuten tuotteen sijainnin, eli varastorakennuksen sekä varastopaikan haku ja valinta. Myös nimikkeen haku ja valinta toteutettiin. Pikainventoinnin tärkeimpänä ominaisuutena toteutettiin myös nimikkeen tietojen muokkaaminen ja tallentaminen. Pikainventoinnin lisäksi vanhasta sovelluksesta valittiin käyttötapauksiksi myös sisäänkirjautuminen ja uloskirjautuminen rajapintaintegraatioita hyödyntäen sekä käyttäjän varaston valinta ja automaattinen määrittäminen. Rajapintaintegraatiot toteutettiin lähettämään kutsuja samaan palveluun, johon myös vanha sovellus lähetti kutsuja. Yhtenä keskeisimmistä käyttötapuksista toimi vanhassa sovelluksessa hyvin pieneen arvoon jäänyt viivakoodinlukuominaisuus.

Vanhasta sovelluksesta toteutettiin pikainventoinnin toiminnallisuuden vaatimat käyttötapaukset. Vanhan sovelluksen käyttötapauksien lisäksi uuteen sovellukseen valittiin käyttötapaukseksi myös vanhan käyttöliittymän kehittäminen käyttäystävällisempään ja intuitiiviempaan muotoon. Sovelluksen käytön vaatimista vaiheista haluuttin vähentää niin monta kuin vain mahdollista, silti säilyttäen täyden toiminnallisuuden. Viivakoodinlukuominaisuus nousi merkittävään rooliin sovelluksen käyttäjäkokemuksen kehittämisessä. Sovelluskehitys ei ole vain ohjelmointia, vaan se sisältää myös huomattavia määriä kommunikaatiota

ja käytänteiden sopimista. Seuraavassa luvussa käsitellään uuden sovelluksen rakentamista prosessina.

3.3 Uuden sovelluksen rakentaminen prosessina

Uusi version sovelluksesta haluttiin kehittää noudattaen ketterän kehityksen periaatteita. Vanhan sovelluksen tila tulisi ensin selvittää ja sen jälkeen voitaisiin keskustella uuden sovelluksen käyttötapauksista. Näin ei synny tilannetta, jossa joudutaan keksimään pyörä monta kertaa uudestaan, vaan voidaan hyödyntää vanhasta hyväksitodettuja ominaisuuksia ja työnkulkuja. Ennen uuden sovelluksen kehittämistä tulisi kuitenkin päättää mitä käytäntöjä tultaisiin hyödyntämään projektissa.

Projektiin valittiin ketteräksi kehitysmenetelmäksi Scrum low-coden visuaalisuuden, kirjallisuudesta löydettyjen esimerkkien sekä Mendixin perehdytyskurssien suositusten perusteella (Kadenic, Koumaditis ja Junker-Jensen 2023). Low-coden visuaalisuuden uskottiin tukevan ketterää kehitystä, sillä low-codella tuotettua sovellusta ja sen toimintalogiikkaa voitaisiin tarkastella myös asiakkaan kanssa. Kirjallisuudesta huomattiin uskoa low-coden mahdollisuuksiin tehostaa sovelluskehitystä (Sanchis ym. 2019) (Wang ym. 2022). Low-code-alustojen käytön on koettu myös nopeuttavan sovelluskehitystä (Văduva ja Vălean 2021). Low-coden tehokkuutta ei olla kuitenkaan juurikaan tutkittu (Trigo, Varajão ja Almeida 2022). Koska low-coden kirjallisuudessa on paljon viitteitä low-coden nopeuteen, myös tiimityöskentelyn käytänteiden tulisi olla nopeita ja ketteriä. Näiden todisteiden valossa Scrum valittiin projektiin ketteräksi kehitysmetodologiaksi. Seuraavassa luvussa kerrotaan tarkemmin ketteristä kehitysmenetelmistä.

3.3.1 Ketterät kehitysmenetelmät

Low-codessa korostuu sovelluskehityksen ketteryys. Totterdale (2018) käyttää ketteriä kehitymenetelmiä sovelluksensa muutoksien toteuttamisen nopeuttamiseksi. Poe ja Mew (2022) mukaan ketterien kehitysmenetelmien tavoitteena on lyhentää sovellusten kehittämiseen kulunutta aikaa. Ketterien kehitysmenetelmien sisällyttäminen low-code-työskentelyyn voisi mahdollistaa näiden kahden metodologian yhteisvaikutuksena sovellusten kehittämisen te-

hokkuuden parantumista. Tästä ei tosin ole paljoa tutkimustietoa, joten ei voida varmuudella väittää sovelluskehityksen tehokkuuden nousevan. Ketterien kehitysmenetelmien on myös tarkoitus pienentää sovelluskehittäjien ja liiketoiminnan henkilöstön välistä kuilua (Poe ja Mew 2022).

Scrum-metodologiassa projektitiimi koostuu useista eri rooleista. Kadenic, Koumaditis ja Junker-Jensen (2023) kertovat Scrum-projektitiimin koostuvan tuotteen omistajasta (PO), Scrum masterista sekä kehittäjistä. Kullakin roolilla on omat tehtävänsä. Tuotteen omistaja on vastuussa muun muassa projektinhallinnassa sijaitsevien käyttötapausten tärkeysjärjestyksestä. Scrum master on vastuussa kehitystiimin sisäisen kommunikaation ja yhteistyön sujuvuudesta sekä myös tuotteen omistajalle kommunikoinnista (Kadenic, Koumaditis ja Junker-Jensen 2023). Seuraavassa luvussa käsitellään tämän tutkielman projektin ensimmäistä vaihetta, eli Scrumin mukaista ensimmäistä "Kick off"-tapaamista.

3.3.2 ”Kick off”-tapaaminen

Ensimmäisenä ”Kick off”-tapaamisen asiana käsiteltiin projektitiimin roolitusta. Scrum-metodologian mukaisesti tämänkin tutkielman projektissa määriteltiin projektiin osallistuville henkilöille heidän roolinsa. Yrityksen sisältä löydettiin projektille tuotten omistaja. Scrum masterin rooli määrättiin Mendix low-code-alustalla eniten kokemusta omaavalle kehittäjälle ja Scrum masterin roolia vaihdeltiin kehittäjien välillä kokemusten jaon vuoksi projektin edetessä. Scrum masterin rooli ei vaihtunut usein, joten roolin vaihdon ei koettu vaikuttavan tuloksiin. Kehitettävä sovellus on kuitenkin vain harvoin yrityksen sisäinen.

Projektilla pyrittiin simuloimaan mahdollisimman hyvin asiakasprojektia, joka sisältää aina myös tuotteen tilaaman asiakkaan. Tuotteen omistajan ja Scrum masterin lisäksi projektille määritettiin yrityksen sisältä asiakas, minkä tarkoituksena oli simuloida asiakasrajapintaa projektin aikana. Näin projektista saatiin mahdollisimman todenmukainen, vaikka projekti toteutettiin yrityksen sisäisenä työnä. Projektin roolien määrittämisen jälkeen projektitiimi sopi sekä sisäisten että simuloitujen ulkoisten tapaamisten ajankohdista ja toistuvuuksista.

Sovittuja tapaamisia olivat projektitiimin sisäiset päivittäiset tapaamiset ja ulkoiset simuloidun asiakkaan kanssa viikoittaiset tapaamiset. Tapaamisten ajankohdat sovittiin yhdessä kai-

kille parhaiten sopiviksi. Projektitiimin päivittäiset tapaamiset sovittiin alkamaan aamulla ja kestämään 15 minuuttia. Päivittäisten tapaamisten tarkoituksena oli toimia tiedonjakotilaisuuksina, joissa tuotiin esille kiireellisimmät asiat. Päivittäisessä tapaamisessa ei ratkaistu kenenkään haasteita, vaan sovittiin sopivat ajankohdat ratkaista haasteet. Seuraavaksi sovittiin viikoittaiset tapaamiset.

Simuloitujen ulkoisten viikoittaisten tapaamisten kestoksi sovittiin yksi tunti, jotta saataisiin esiteltyä tarvittavat asiat. Sovellusten uudistusten esittelyyn varattiin puoli tuntia ja palautteeseen sekä uusien ideoiden luomiseen yhteensä puoli tuntia. Viikoittaisiin tapaamisiin sisällytettäisiin myös jokaiselta viikolta kirjallinen esitys siitä, miten kyseisellä viikolla käsittelyssä olevat käyttötapaukset olivat edenneet. Ennen ensimmäistä projektitiimin ja asiakkaan välistä tapaamista projektitiimi tapasi vanhan sovelluksen asiantuntijan kanssa, jolta saatiin vanhan sovelluksen testiympäristöön tunnukset ja ohjeet vanhan sovelluksen suorittamiseen kehittäjien tietokoneilla. Ensimmäisten käyttötapausten muodostaminen tapahtuisi seuraavan tapaamisen aikana.

Ensimmäissä määrittelypalaverissa käsiteltiin vanhan sovelluksen hyviä puolia ja kehityskohteita, joiden pohjalta suunniteltiin ensimmäiset käyttötapaukset uudelle sovellukselle. Tapaaminen pidettiin siten, että kaikki osapuolet olivat samassa tilassa. Tapaaminen olisi ollut mahdollista toteuttaa myös hybriditapaamisena. Määrittelypalaverissa suunniteltiin taulukon 1 mukaiset käyttötapaukset. Taulukossa 1 näkyvät käyttötapaukset eivät jääneet projektin lopullisen version käyttötapauksiksi, vaan käyttötapauksia lisättiin ja laajennettiin projektin edetessä. Käyttötapausten suunnittelun jälkeen kehitystiimi tuotti työmääräarvion kyseisille käyttötapauksille.

Käyttötapauksista pienitöisimmiksi koettiin perinteisiä ja Mendixillä nopeasti toteutettvia listauksia ja niiden välillä navigointia. Viivakoodilla nimikkeiden hakua haluttiin painottaa uudessa sovelluksessa, eikä kehitystiimi ollut aiemmin implementoinut viivakoodinlukuominaisuutta, joten sille varattiin kaksinkertaisesti enemmän työaikaa, kuin nimikkeen listasta valitsemiselle. Sisäänkirjautuminen rajapintaintegraatioita hyödyntäen oli myös uusi käyttötapaus kehitystiimille, joten myös siihen varattiin huomattavasti aikaa. Nimikkeen määrän muokkaamiselle varattiin myös huomattavasti aikaa Mendixillä hieman kompleksisesti to-

Käyttötapauksen otsikko	**Käyttötapauksen työmäärä (HTP)**
Sisäänkirjautuminen	4
Uloskirjautuminen	1
Inventointiin etusivulta	1
Listaus kaikista lokaatioista toiminnallisuus	1
Varaston valitseminen listauksesta	1
Viivakoodilla varaston hyllypaikka	2
Viivakoodilla nimikkeen lukeminen	2
Yksittäisen nimikkeen tarkastelu	1
Yksittäisen nimikkeen määrän muokkaaminen	3

Taulukko 1. Käyttötapaukset ja niiden työmäärät

teutettavan tuotteen muokkaamisen käyttöliittymän rakentamisen vuoksi.

3.3.3 UI/UX suunnitelma

Käyttötapausten määrittämisen jälkeen kehitystiimi tapasi sovellusmuotoilijan kanssa. Tavoitteena tapaamiselle oli ideoida ja luoda uusi käyttöliittymä vanhaa hyödyntäen sekä tuoda vanhasta sovelluksesta kaikki käyttäjäkokemuksen kannalta hyväksi todettu uuteen käyttöliittymään. Kehitystiimi esitteli vanhan sovelluksen pikainventoinnin toiminnallisuutta ja esitti myös uuteen sovellukseen asiakkaalta tulleet toiveet sovelluksen käyttäjäkokemuksen parantamisesta. Asiakkaan toiveet painottuivat viivakooodinlukijan korostamiseen ja toiminnallisten käyttöliittymäelementtien käytettävyyden ja tarkoituksenmukaisuuden selkeyttämiseen.

Tapaamisen aikana ei muodostettu käyttöliittymärunkoa, vaan annettiin sovellusmuotoilijalle tarvittavat määritykset toivotusta käyttäjäkokemuksesta ja käyttöliittymän tyylistä. Sovellusmuotoilijalle annettiin vapaat kädet muovata sovelluksen käyttöliittymästä uusi versio. Rajoitteina toimivat vain teemavärit, jotka haluttiin säilyttää vanhasta sovelluksesta ja Mendixin hyödyntämä Atlas 3 -käyttöliittymäkomponenttikirjasto. Tapaamisen päätteeksi

sovellusmuotoilijalla oli tarpeelliset tiedot sovelluksen uuden käyttöliittymän muodostamiseksi. Sovellusmuotoilijan arvio käyttöliittymäsuunnitelman valmistumisesta oli kaksi viikkoa eteenpäin tapaamisen ajankohdasta.

3.3.4 Sprint 0

Työmääräarvion tuottamisen jälkeen pidettiin Scrum-metodologian mukaista ensimmäistä sprint planningia, jonka aikana muokattiin käyttötapauksia työmääräarvioon nojaten. Käyttötapaukset järjestettiin tuotteen omistaja johdolla niiden kriittisyyden ja kiireellisyyden mukaan. Tapaamisessa esitettiin myös uusia ideoita käyttötapauksiksi, joista osa jätettiin jatkokehityskohteiksi. Tämän tapaamisen jälkeen kehitystiimi aloitti kehitystyön.

Ensimmäisen työviikon aikana kaikki kehitystiimin jäsenet pystyttivät vanhan sovelluksen testiympäristön tietokoneilleen, jotta saatiin tarkemmin tarkasteltua vanhan sovelluksen hyviä puolia ja vanhaa käyttöliittymää. Kehitystiimi perusti myös uuden sovelluksen mobiiliympäristön, jonka päälle uusi sovellus tultaisiin rakentamaan. Jotta uuteen sovellukseen saataisiin tuotua samaa dataa, kuin vanhaan sovellukseen, tarvittiin tietoa rajapinnoista ja niihin lähetettävistä kutsuista. Käytettävän rajapinnan toimintaa tutkittiin käyttämällä Git Bash -komentorivityökalua, jolla suoritettiin cURL-komennon avulla rajapintakutsuja. Rajapintaintegraation toteuttamista tukemaan saatiin myös Swagger-dokumentaatio rajapinnan päätepisteistä ja vakiomuotoisista vastauksista. Käytettävään rajapintaan tutustuminen ei eronnut perinteisestä sovelluskehityksestä, sillä siinä ei käytetty low-code-työkaluja. Ensimmäisen viikon aikana mahdollistettiin myös mobiilisovelluksen testaaminen natiivisti mobiililaitteella.

Mendixillä on suurissa sovelluskaupoissa natiivien mobiilisovellusten käyttöliittymätestaamista varten oma Mendix Make It Native -sovellus. Sovelluksen asennettua kehittäjän tarvitsee vain ajaa sovellus Mendix Studio Pro:ssa lokaalisti, yhdistää mobiililaite samaan lähiverkkoon ja skannata Make It Native -sovelluksella Mendix Studio Pro:n tarjoama QR-koodi. Kehitettävä sovellus avautuu sen hetkisessä tilassaan mobiililaitteelle. Kun kehittäjä tekee muutoksia kehitettävään sovellukseen, sovellus päivittyy uudelleenajamisen yhteydessä automaattisesti myös mobiililaitteelle. Tämä toiminnallisuus toimii, kun mobiilikäyttö-

liittymissä käytetään vain kaikista yksinkertaisimpia widgettejä. Jos käytetään esimerkiksi laitteen kameraa tai paikkatietoja, kehittäjän täytyy tehdä oma versio Mendix Make It Native -sovelluksesta. Räätälöidyn Make It Native -sovelluksen tuottaminen oli aiemmissa projekteissa kehittäjille haastavaa ja aikaavievää, mutta ensimmäisen projektin jälkeen prosessi oli kehittäjillä muistissa ja tässä projektissa räätälöidyn Make It Native -sovelluksen tuottaminen oli nopeaa ja yksinkertaista. Ensimmäisen viikon työmäärät näkyvät taulukossa 2.

Taulukosta 2 huomataan, että viikon työssä ei ilmennyt suurta määrää aikaa kuluttavia haasteita low-coden näkökulmasta. Työ oli vielä suurimmaksi osaksi perinteisen ketterän sovelluskehityksen kaltaista kommunikaatioon painottuvaa työtä, joten varsinaisia low-codeen ja Mendixiin kohdistuvia haasteita ei syntynyt. Ensimmäisen viikon viikoittaisessa tapaamisessa esiteltiin työmääräarvio kaikkien sovelluksen sen hetkisten käyttötapausten osalta. Asiakas oli tyytyväinen työmääräarvioon, mutta esimerkiksi sisäänkirjautumisen suuri työmäärä herätti kysymyksiä. Kehitystiimi avasi sisäänkirjautumisen toteutuksen vaativan suunnittelua ja sen tuntemattomuuden vuoksi sen työmäärään haluttiin lisätä hieman puskuria. Esitelmässä esiteltiin myös jokaisen viikolla työstetyn käyttötapauksen kohdalta yksityiskohtaisesti, mihin aika oli kulunut ja mitä vaiheita käyttötapauksista oli valmiina. Tapaamisessa ilmoitettiin myös käyttöliittymäsuunnitelman ennakoitu valmistumisajankohta. Käyttötapauksien avaaminen ei herättänyt huomioita. Viikon kehitystyön yksityiskohtaisen avaamisen jälkeen esiteltiin kehittäjien ja tuotteen omistajan ehdotus seuraavalla viikolla työstettävistä käyttötapauksista ja vaiheista. Asiakas hyväksyi ehdotuksen, jonka jälkeen tapaaminen päätettiin ja oli aika siirtyä ensimmäisen virallisen sprintin töiden pariin.

Käyttötapauksen otsikko	**Käytetty aika (HTP)**
Vanhan mobiiliympäristön pystyttäminen	1,5
Vanhaan mobiilisovellukseen tutustuminen	2
Mobiilikehitysympäristön pystyttäminen	0,5
Yhteyden mahdollistaminen rajapintaan	0,5

Taulukko 2. Sprint 0 käyttötapaukset ja niiden vaatima työaika

3.3.5 Sprint 1

Ensimmäisen sprintin työt painottuivat sisään- ja uloskirjautumisen toimintalogiikkaan sekä sovelluksen etusivun ja pikainventoinnin navigaatioon. Toisella viikolla toteutettiin myös käyttöliittymä sisäänkirjautumiselle. Kehitystiimin päivittäisisten tapaamisten dokumentaatiosta selvisi, että sisäänkirjautumisen rajapintaintegraatio oltiin tuotettu nopeasti, mutta rajapinnasta saatavan tiedon ja Mendixin käyttäjänhallinnan yhdistäminen tuotti haasteita kehitystiimille. Taulukosta 3 nähdään, että kirjautumistoiminnallisuuden kehittämiseen on kulunut huomattavasti aikaa. Käyttäjänhallinta koettiin muillakin tavoilla haastavaksi. Käyttäjädatan synkronisaatio tietokannan ja mobiililaitteen välillä ei päivittynyt automaattisesti, kun data poistettiin tietokannasta. Kehitystiimi joutui rakentamaan erillistä logiikkaa kirjautumisdatan hallintaa varten sovelluksen kehitysvaiheessa.

Sisäänkirjautuminen vaati rajapintaintegraation, joten Mendix Studio Pro:ssa luotiin tarvittavat resurssit kyseiselle rajapintakutsulle. Resurssien luonnin ja konfiguroinnin jälkeen generoitiin automaattisesti tietomalli kirjautumisdatasta. Tietomallin pohjalta Mendix-sovellus osaisi käsitellä vastauksena tulevaa dataa. Automaattisesti JSON-datasta generoitu tietomalli ei ollut täydellinen ja se aiheutti virheitä rajapintaa kutsuttaessa. Tämän ongelman selvittämiseen kului muutamia tunteja aikaa. Kyseessä oli automaattisen generaation väärin tulkitsema olion attribuutin tietomuoto. Sisäänkirjautuminen aiheutti muitakin ongelmia.

Kehitystiimi koki teknologian hyödyntämisen kannalta parhaaksi kehittää sisäänkirjautumisen, joka rekisteröisi käyttäjän automaattisesti rajapintaan kirjautumisen yhteydessä myös Mendix-sovelluksen käyttäjäksi, mutta tämän todettiin vievän liikaa aikaa. Mendix-käyttäjäksi rekisteröiminen olisi helpottanut esimerkiksi sovelluksen turvallisuusmääritysten ohjaamista roolipohjaisesti. Asiasta keskusteltiin poikkeuksellisesti asiakkaan kanssa jo viikolla. Asiakas halusi sovelluksen käsittelevän käyttäjiä Mendix-sovelluksen sisällä anonyymeinä käyttäjinä. Tämä päätöksen ansiosta kirjautumistiedon ylläpitäminen helpottui ja kirjautumiseen käytetty rajapinta jäi vastuuseen kirjautumistiedon ylläpitämisestä. Jatkossa sovelluksessa tarkastettiin käyttäjän kirjautumisen voimassaolo rajapintakutsulla, joka varmisti käyttäjä olevan kirjautunut. Kirjautumisen jälkeen käyttäjä ohjattiin etusivulle.

Asiakas halusi uuden mobiilisovelluksen etusivun olevan ominaisuusvalikko, josta käyttä-

jä voisi valita haluamansa sovelluksen. Etusivulle lisättiin navigaatiomahdollisuus pikainventointiin. Mendix Studio Pro:ssa luotiin tarvittavat sivut eri näkymille ja näkymistä navigointi tapahtui painikkeeseen asetetulla hiirenpainallusta odottavalla toiminnallisuudella. Näkymien luonnissa ja näkymistä toiseen navgaatiossa ei ilmennyt haasteita. Kehitystiimi oli yllättynyt kuinka helposti ja nopeasti eri näkymiä pystyi luomaan ja kuinka nopeasti esimerkiksi painikkeeseen pystyi konfiguroimaan navigaatio-ominaisuuden.

Ensimmäisen sprintin viikoittaisessa tapaamisessa asiakkaan kanssa käsiteltävät asiat käsiteliin samassa järjestyksessä, kuin ensimmäiselläkin viikolla. Tässä tapaamisessa raportoitiin edistyminen sisäänkirjautumisen, uloskirjautumisen sekä ensimmäisten näkymien luomisen osalta. Tapaamisessa eniten huomiota herätti sisäänkirjautumisen tekninen näkökulma. Teknisesti Mendixillä ei ole mahdollista luoda käyttäjiä, ellei käyttäjä ole kirjautunut sovellukseen tarpeeksi korkeita oikeuksia omaavalla käyttäjällä. Tämä esti turvallisen käyttäjien automaattisen rekisteröinnin Mendix-käyttäjiksi sovellukseen. käyttötapausten käsittelyn jälkeen kehitystiimi esitteli projektin kokonaiskuvan, joka kertoi paljonko toteutettavan sovelluksen MVP-versiosta oli valmiina ja paljonko tekemättä. Viikoittaisen tapaamisen lopuksi keskusteltiin seuraavan sprintin aikana toteutettavista käyttötapauksista.

Käyttötapauksen otsikko	**Käytetty aika (HTP)**
Sisäänkirjautuminen	4
Uloskirjautuminen	1
Etusivulta pikainventointiin	1

Taulukko 3. Sprint 1 käyttötapaukset ja niiden työmäärät

3.3.6 Sprint 2

Toisen sprintin käyttötapauksiin kuului viivakoodinlukijan implementaatio sovellukseen, viusaalinen muotoilu kirjautumissivulle ja etusivulle (kts. taulukko 4). Toiselle sprintille työstettiin myös varastojen hakua rajapinnasta, niiden listausta käyttöliittymään ja yksittäisen varaston valitsemista. Myös kehitystiimin ehdotus kirjautumistiedon vanhenemisen yhteydessä käyttäjän automaattiesta ohjaamisesta kirjautumiseen hyväksyttiin ja otettiin työlistalle. Toisen

sprintin aikana huomattiin myös tarve sovelluksen datanhallinnan muutokselle optimaation vuoksi. Harvoin muuttuva tieto, kuten varastot ja varastopaikat tultaisiin hakemaan rajapinnasta ja päivittämään lokaaliin tietokantaan kirjautumisen yhteydessä. Kehitystiimin päivittäisissä palavereissa kysymyksiä herätti kehitystiimin yhteistyön mahdollistaminen versionhallinnan avulla. Käytänteitä yhteistyölle jouduttiin muodostamaan itse, kun niihin ei oltu saatu Mendixin verkkokursseista valmiita malleja.

Tiedon tallentaminen mobiililaitteelle ja sieltä Mendix-sovelluksen tietokantaan koettiin monimutkaiseksi, sillä toimintalogiikan rakentamisessa tuli ottaa huomioon laitteen ja Mendix-sovelluksen tietokannan datan synkronisaatio. Tietokantakyselyt tuli suorittaa asynkronisesti, jotta edellisen toimintalogiikka ja synkronisaatio valmistuisi ennen uutta synkronisaatiokutsua. Datan synkronisaation vaatima logiikka oli valmiiksi toteutettu nanoflowiin liitettävä toiminto. Sovelluksen datan synkronisaation konfigurointi vei ensimmäisellä toteutuskerralla huomattavasti aikaa. Kehittäjät tosin arvioivat, että seuraavilla toteutuskerroilla synkronisaation konfigurointi olisi huomattavasti nopeammin toteutettavissa yleisimpien haasteiden ilmennyttyä tutkielman projektin aikana.

Joidenkin toiminnallisuuksien implementaatio sovellukseen oli huomattavasti helpompaa kuin toisten. Viivakoodinlukijan implementaatio vaati vain widgetin lisäämisen erilliseen näkymään ja hieman widgetin konfigurointia. Sovelluksen testaaminen ei toiminut enää Make It Native -mobiilitestaussovelluksella, vaan kehitystiimin täytyi tehdä räätälöity testaussovellus. Tämä prosessi oli nopeaa, sillä kehitystimillä oli jo aiempaa kokemusta räätälöidyn tetstaussovelluksen konfiguroinnista. Kehitystiimi koki kuitenkin räätälöidyn testaussovelluksen toteuttamisen olevan varsin umpinainen musta laatikko. Räätälöity testaussovellus tuotettiin Mendix Studio Pro:hon sisäänrakennetulla Mendix Native Mobile Builder -toiminnallisuudella. Siihen konfiguroitiin tarvittavat versionhallintaosoitteet ja avaimet, jotta toiminnallisuus pääsi hyödyntämään versionhallintaresursseja ylläpitääkseen ja luodakseen uuden testisovelluksen. Kehitystiimi kritisoi Native Mobile Builderin toimintavarmuutta, sillä kehitystiimi joutui rakentamaan testisovelluksen usein uudelleen rakennusprosessin epäonnistumisen vuoksi. Tämä vei turhaan aikaa sovelluskehitykseltä, sillä sovellusta ei voinut kehittää testaussovelluksen rakentamisen aikana. Räätälöidyn mobiilisovelluksen rakentamisen ajamiseen kehitystiimi hyödynsi Android Studion mukana asennettavaa mobiililaitet-

ta emuloivaa ominaisuutta, fyysisiä Android-käyttöjärjestelmään pohjautuvia mobiililaitteita sekä iOS-käyttöjärjestelmään pohjautuvia mobiililaitteita. Kehitystiimin mukaan sovelluksen asentaminen oli yksinkertaista ja nopeaa.

Viivakoodinlukijan toimintaan vaadittiin myös rajapintakutsuja. Rajapintakutsujen toteuttaminen oli nopeaa. Viivakoodinlukijan implementaation ja sen vaatimien rajapintakutsujen toteuttamiseen vaadittiin odotettua vähemmän aikaa. Rajapintaintegraatioita toteutettiin myös varastopaikkojen viivakoodilla hakemisen sekä nimikkeen tietojen viivakoodilla hakemisen toimintalogiikan yhteyteen. Rajapintapintgraatiot olivat nopeita, mutta jo aiemmin huomattu tietomallin automaattinen generaatio aiheutti virheitä toteutuksessa. Virheet oli kuitenkin nopeaa korjata tietomalliin muuttamalla attribuutin tyyppiä ja konfiguroimalla tarvittavat resurssit uudelleen.

Toisen sprintin viikoittaisessa tapaamisessa käsiteliin jälleen tilannekatsaus sovelluksen tilasta. Edistyminen oli odotetulla tasolla toiminnallisuuden osalta, mutta käyttöliittymäsuunnitelman puuttuminen oli aiheuttanut työn hidastumista visuaalisen mouotoilun osalta. Käyttöliittymän toteuttamisen odotettiin kuitenkin olevan nopeaa ja tehokasta, kun käyttöliittymäsuunnitelma saataisiin. Seuraavan sprintin pituudeksi määritettiin kuukausi, sillä suurin osa kehitystiimistä ja myös asiakas oli lomalla suuren osan seuraavasta kuukaudesta. Seuraavan sprintin tavoitteiden määrittämiseksi järjestettiin erillinen tapaaminen.

Käyttötapauksen otsikko	**Käytetty aika (HTP)**
Kirjautumisen yhteydessä varasto- ja varastopaikkadatan päivittäminen	2
Listaus kaikista varastoista	1
Varaston valitseminen listauksesta	2
Viivakoodilla varastopaikan lukeminen	1
Viivakoodilla nimikkeen lukeminen	1

Taulukko 4. Sprint 2 käyttötapaukset ja niiden työmäärät

3.3.7 Sprint 3

Kolmannen sprintin tavoitteiksi asetettiin tähän mennessä kehitettyjen näkymien visuaalisen muotoilun viimeisteleminen ja nimikkeen tietojen muokkaamisen näkymän muodostaminen (kts. taulukko 5). Tällä sprintillä töissä oli kehittäjätiimistä vain yksi työntekijä. Tässä sprintissä huomattiin joitain low-coden ja Mendixin rajoitteita kehittämisessä. Ensimmäisenä kehitettiin navigaatioon sijoitettavaa logiikkaa.

Kehittäjä kohtasi haasteen navigaatioelementtien sijoittamisen yhteydessä. Mendix on linjannut mobiilikäytettävyyteen liittyen, että heidän työkaluillaan pystyy tekemään navigaation vain mobiilikäyttöliittymän alaosaan. Mendixillä ei ole siis mahdollista tehdä mobiilikäyttöliittymiin perinteisempiä hampurilaisvalikoita. Navigaatiossa huomattiin myös muita haasteita. Navigaatioelementtiin ei voinut sijoittaa toimintalogiikkaa normaalin painikkeen tavoin, joten logiikkaa täytyi hyödyntää navigaatiosta avautumalla välisivulla. Toimintalogiikka ajettiin välittömästi sivulle tultaessa jolta navigoitiin uudelle sivulle toimintalogiikan loputtua. Näin saatiin navigaatioon käyttöön toimintalogiikkaa. Kehitystiimi koki tämän olevan työtä hidastava turha kehityksen välivaihe. Toiseen haasteen kehitystiimi kohtasi nimikkeen tietojen muokkaamisessa.

Kun kehitettiin tuotteen tietojen muokkaamisnäkymää, huomattiin, että usein sovelluksissa näkyvän alasvetovalikon valintajoukon dynaaminen määrittäminen oli erittäin vaikeaa. Kehittäjä koki sen jopa niin aikaa vieväksi, että koettiin parhaaksi käyttää aika muihin toimintoihin ja hyödyntää Mendixin vahvuuksia näkymien luomisessa. Nimikkeen yksikön valinta kehitettiin ”pop-up”-ikkunalla, josta valittiin haluttu yksikkö. Kehitystiimi tutki alasvetovalikon valintajoukun dynaamista määrittämistä ja tuli siihen tulokseen, että haluttu tulos olisi vaatinut uuden räätälöidyn widgetin kehittämistä perinteisellä ohjelmoinnilla. Tämä olisi vienyt huomattavasti enemmän aikaa. Huomaattiin myös, että räätälöityjä widgettejä tulisi itse päivittää Mendix-versioiden päivittyessä, joka olisi lisännyt sovelluksen ylläpitokustannuksia. Tästä pääteltiin, että Mendixillä ja todennäköisesti myös muilla low-code-alustoilla tulisi hyödyntää käyttöliittymän suunnittelussa valmiita komponentteja ja kyseisen low-code-alustan vahvuuksi sovelluskehityksessä. Jos vahvuuksia ei hyödynnetä mahdollisimman hyvin, siitä voisi koitua huomattava määrää ylimääräistä työtä.

Kolmannen sprintin asiakastapaamisessa esiteltiin kuukauden työn tulokset. Käytetyn työn määrä oli hieman ennustettua korkeampi. Odotettua enemmän työaikaa oli vienyt nimikkeen tietojen muokkausnäkymän kehittäminen, kuten taulukosta 5 huomataan. Tuotteen määrän muutoksen toimintalogiikka aiheutti hieman haasteita, mutta ne eivät varsinaisesti liittyneet low-coden vahvuuksiin tai heikkouksiin.

Käyttötapauksen otsikko	**Käytetty aika (HTP)**
Sisäänkirjautuminen	0,5
Etusivulta pikainventointiin	0,5
Listaus kaikista varastoissa	0,5
Yksittäisen nimikkeen tarkastelu	1
Yksittäisen nimikkeen määrän muokkaaminen	5
Viivakoodilla varastopaikan ja nimikkeen lukeminen	2

Taulukko 5. Sprint 3 käyttötapaukset ja niiden työmäärät

3.3.8 Sprint 4

Kolmannen sprintin jälkeen sovelluksen MVP-version alkuperäisten käyttötapausten toiminnallisuus oli täysin toteutettu ja käyttöliittymä oli ensimmäisen käyttöliittymäsuunnitelman mukainen. Neljännen spritin aikana kehitettiin taulukon 6 mukaisia käyttötapauksia. Asiakas halusi kehittää sovelluksen MVP-versiosta vielä käyttäjäystävällisemmän. Sovelluksen ulkoasu ei vielä täysin vastannut asiakkaan toiveita ja käyttäjäkokemus oli vielä kömpelö. Käyttäjäkokemusta lähdettiin kehittämään muuttamalla pikainventoinnin työkulkua.

Työnkulun muutokset osoittautuivat yhdeksi Mendixin suurimmista vahvuuksista. Kehittäjätiimin mukaan kaikki toiminnallisuus oli helposti siirrettävissä eri paikkoihin sen modulaarisuuden ansiosta. Käyttäjäkokemusta pyrittiin parantamaan vähentämällä käyttäjältä vaadittujen painalluksien määrää sekä käyttöliittymää selkeyttämällä. Käyttöliittymään lisättiin ohjeistavia tekstejä ja käyttäjän oletusvarasto asetettiin kirjautumisdatasta tulevasta tiedosta, jotta käyttäjän ei tarvitsisi sitä aina määrittää itse. Oletusvaraston asettamiseen tarvittavat muutokset olivat erittäin nopeita tehdä, sillä kaikki tarvittava data oli jo käytettävissä raja-

pintaintegraation ansiosta. Käyttöliittymään widgettien lisääminen ja datan niihin liittäminen oli myös erittäin nopeaa. Kehitystiimi koki sovelluksen työnkulkuun muutosten tekemisen nopeaksi ja vaivattomaksi. Sovellukseen lisättiin myös nimikkeiden listasta hakua varten hakukenttä, jonka avulla käyttäjä pystyi helpommin löytämään haluamansa tuotteen tarpeen tullessa.

Neljännen sprintin asiakastapaamisessa tultiin siihen tulokseen, että sovelluksen MVP-version valmistumiselle olisi aikaa vielä yksi sprint. Sovellukselle oli juuri valmistunut uusi versio käyttöliittymäsuunnitelmasta ja viimeisiä muutoksia päästiin tekemään. Viimeisen sprintin tavoitteena oli viimeistellä sovelluksen ulkoasu vastaamaan uutta käyttöliittymäsuunnitelmaa. Lisänä työnkulun parantamiseksi asiakas halusi käyttäjän pystyvän muokkaamaan nimikkeen tietoja vain sen viivakoodin lukemalla.

Käyttötapauksen otsikko	**Käytetty aika (HTP)**
Automaattinen varaston valinta	0,5
Varaston ja varastopaikan näyttäminen työnkulun eri vaiheissa	1
Oletusvaraston vaihtaminen sessiokohtaisesti	1
Hakutoiminto nimikkeen etsimiseen	1
Listaus varastopaikoista	1
Viivakoodilla nimikkeen hakeminen	1,5

Taulukko 6. Sprint 4 käyttötapaukset ja niiden työmäärät

3.3.9 Sprint 5

Viimeinen sprint alkoi käyttöliittymäsuunnitelman mukaisen teemävärityksen lisäämisellä sovellukseen. Teemavärien lisääminen oli kehitystiimin kokemuksen mukaan yksinkertaista, nopeaa ja helposti ylläpidettävää. Uuden käyttöliittymäsuunnitelman mukaan painikkeista tehtiin suurempia ja selkeämpiä. Nimikkeen haku vain sen viivakoodin perusteella toteutettiin erittäin nopeasti, sillä jo toteutettua toimintalogiikkaa pystyttiin hyödyntämään uudessa työnkulun tarpeessa. Taulukosta 7 huomataan, että sovelluksen ulkoasun viimeistely on

vienyt kohtalaisen paljon aikaa. Tämä johtui ulkoasumuutosten määrän suuruudesta, eikä haasteista kehityksessä.

Yksittäisen nimikkeen löytäminen viivakoodilla toteutettiin ensin avaamalla käyttäjälle viivakoodinlukija. Kun käyttäjä luki viivakoodin ja nimikettä ei ollut varastossa kuin yhdessä varastopaikassa, hänelle näytettiin kyseisen nimikkeen tiedot, joita hän pääsi muokkaamaan. Jos varastossa oli nimikettä useassa varastopaikassa, hänellä näytettiin selkeä näkymä, josta hän pystyi valitsemaan tarkoittamansa tuotteen varastopaikan perusteella. Työnkulun muutos vähensi käyttäjältä vaadittuja klikkauksia huomattavasti.

Kun kaikki käyttötapaukset oli tehty loppuun, oli aika esitellä sovellus asiakkaalle. Sovellusta esiteltiin myös yrityksen sisällä useille eri osastoille ja vastaanotto oli erittäin positiivinen. Asiakas oli tyytyväinen sovelluksen ulkonäköön ja toimintaan. Yritys sai sovellusprojektista tärkeää informaatiota Mendixin kyvykkyyksistä mahdollisissa tulevissa asiakasprojekteissa natiivien mobiilisovellusten kehittämisestä. Tärkeimmiksi huomioiksi koettiin Mendix Studio Pro:lla tehtävien sovelluksen työkulun muutosten helppous ja sovelluksen arkkitehtuurin jopa Mendixin pakottama modulaarisuus, joka tulisi todennäköisesti helpottamaan sovelluksen ylläpitoa.

Käyttötapauksen otsikko	**Käytetty aika (HTP)**
UI/UX-suunnitelman mukaan sovelluksen ulkoasun viimeistely	3
Nimikkeen tietojen hakeminen pelkällä nimikkeen viivakoodilla	1

Taulukko 7. Sprint 5 käyttötapaukset ja niiden työmäärät

3.3.10 Projektin päättäminen

Kun kaikki vanhan sovelluksen korvaavan mobiilisovelluksen MVP-version kaikki käyttötapaukset oli suoritettu, oli aika päättää projekti. Projektin uskottiin tuovan arvokasta tietoa yrityksen low-code-työskentelylle, mutta sovellusta ei lähdetty jatkokehittämään. Yrityksen

tarpeena oli saada ulkoisia asiakkaita, jonka tueksi tätä tutkielmaa varten kehittyä sovellusta voitaisiin käyttää esimerkkinä onnistuneesta low-code-toteutuksesta. Yrityksen tarkoituksena uudelle mobiilisovellukselle oli nähdä voisiko Mendixillä toteuttaa käytössä olevalla työvoimalla sovelluksia ja se onnistuttiin osoittamaan todeksi.

Projektin viimeisessä viikoittaisessa asiakastapaamisessa käsiteltiin mahdollisia sovelluksen jatkokehitystoimenpiteitä. Tapaamisessa tulitiin siihen tulokseen, että seuraavat askeleet olisivat sovelluksen automaatiotestaus ja sovelluksen julkaiseminen useaan eri ympäristöön sekä tarvittavien automaatioiden rakentaminen jatkuvan kehittämisen ja integraation saavuttamiseksi. Näitä käyttötapauksia ei kuitenkaan toteutettu käytännössä, vaan niistä selvitettiin teoria, jonka avulla voitaisiin tarvittaessa tulevaisuudessa tuottaa kyseiset käyttötapaukset. Yritys oli tyytyväinen projektin tilaan ja oli aika lopettaa sovelluksen aktiivinen kehittäminen.

3.4 Sovellusprojektin analyysimenetelmä

Tässä luvussa kerrotaan siitä, miten sovellusprojektin kehitysprosessi dokumentointiin ja miten kehitysprosessin dokumentaatiosta tehtiin päätelmät sovellusprojektin tuloksista. Oman kontribuution osassa toteutetun sovellusprojektin kehitysprosessia analysoitiin kehitystiimin päivittäisten tapaamisten muistiinpanojen, kehitystiimin ja simuloidun asiakkaan välisten viikoittaisten tapaamisten muistiinpanojen, sekä kehitystiimin kokemusten avulla.

Kehitystiimin päivittäiset tapaamiset dokumentointiin Scrum-metodologian mukaisesti. Päivittäisissä tapaamisissa käsiteltiin kehittäjien kysymyksiä, onnistumisia ja edistymistä. Tarkoituksena oli saada dokumentoitua mahdollisia haasteita sekä onnistumisia Mendix low-code-alustalla työskentelystä. Viikoittaisista asiakastapaamisista tuotettiin samankaltainen dokumentaatio kuin päivittäisistä kehitystiimin sisäisistä tapaamisista. Tämän lisäksi jokaisesta viikoittaisesta tapaamisesta tuotettiin Microsoft PowerPoint -esitys, jossa esiteltiin kyseisen viikon edistys ja projektin tila. Kehittäjien kokemuksia low-codella työskentelystä kerättiin suullisesti, sekä kokenein kehittäjä kertoo laajasti omasta kokemuksestaan low-code-kehityksestä tutkijan äänellä.

Sovellusprojektin prosessia analysoitiin yhdistämällä huomioita päivittäisten tapaamisten

muistiinpanoista, kehittäjien kokemuksista sekä viikoittaisten tapaamisten dokumenteista. Viikoittaisten tapaamisten dokumentaatiosta saatiin selville jokaisessa sprintissä kehitetyistä käyttötapauksista ja niiden vaatimasta ajasta. Käyttötapaukseen vaadittu aika viittaa usein joko käyttötapauksen monimutkaisuuteen tai haasteisiin käyttötapauksen toteuttamisessa. Kehitystiimi halusi tuottaa jokaisesta käyttötapauksesta laadukkaan ratkaisun, joten käyttötapauksen vaatima aika ei riipu sen laatutavoitteista. Käyttötapausten vaatimaa aikaa verrattiin sekä päivittäisten tapaamisten dokumentaatioon, että kehittäjien kokemuksiin käyttötapauksen kehittämisestä. Näin saatiin tuotettua näkemys siitä, mitkä sovelluskehityksen vaiheet Mendix low-code-alustaa käyttäen olivat tutkielman sovellusprojektissa helppoja ja mitkä vaikeita.

3.5 Sovellusprojektin tulokset

Vanhasta varastonhallintasovelluksesta uuden mobiilisovelluksen MVP-version kehittäminen low-code-alustalla osoittautui nopeaksi, mutta ei aivan mutkattomaksi. Projektiin kului aikaa yhteensä 37 henkilötyöpäivää, joka oli huomattavasti enemmän, kuin alkuperäisessä käyttötapausten suunnittelupalaverissa määritelty 16 henkilötyöpäivää. Tätä valtavaa prosentuaalista eroa selittää uusien käyttötapausten muodostuminen projektin edetessä. Tämä kuvaa melko hyvin sovellusprojektien yleistä ongelmaa, joka on heikko määrittelyprosessi. Jos projektin tavoitteiden määrittely olisi ollut selkeämpää, oltaisiin todennäköisesti päästy tarkempaan arvioon uuden sovelluksen MVP-version toteuttamiseen kuluvasta ajasta.

Tutkielmassa käytetty low-code-alusta Mendix vaikutti projektin etenemiseen sekä nopeuttavana että myös joissain kohdissa hidastavana tekijänä. Yhdeksi Mendixin parhaista puolista osoittautui olemaan lomakepohjaisten yksinkertaisten ratkaisujen toteuttaminen. Uusien näkymien generointi tietomallin pohjalta oli erittäin nopeaa. Jos kehitystyökalun vahvuuksia noudatettaisiin tiukasti käyttöliittymäsuunnitelmassa, olisi todennäköisesti mahdollista saavuttaa sovellusten kehittämisnopeuden huomattavaa nousua tämän tutkielman projektin kehitysnopeuteen verrattuna. Jos low-code-alustan vahvuuksia ei oteta huomioon käyttöliittymäsuunitelmassa, niin low-code-alustan käyttö voi myös hidastaa kehittämisnopeutta. Tämä huomattiin taulukossa 5 näkyvän nimikkeen määrän muokkaamisen yhteydessä. Kyseisen näkymän toteuttaminen vei huomattavasti enemmän aikaa kuin muut näkymät, vaikka käyt-

töliittymä ei ollut kovinkaan monimutkainen. Kyseisen näkymän osalta käyttöliittymäsuunnitelma ei täysin huomioinut Mendixin käyttöliittymäkehityksen vahvuuksia, jonka johdosta huomattiin välittömästi kehitysnopeuden laskua. Kehittäjät kokivat myös Mendixin asettamat rajoitteet alustalle hieman raskaiksi.

Kehittäjien täytyi tehdä Mendixillä työskennellessä ratkaisuja, jotka eivät vaikuttaneet optimaalisilta alustan asettamien rajoitteiden vuoksi. Alusta asettaa monia rajoitteita, jotka voivat hidastaa suunnitellun toiminnallisuuden toteuttamista tai jopa estää sen kokonaan. Jotta sovelluksien kehittämisessä vältyttäisiin suurilta haasteilta, low-code-alustasta vastaavan arkkitehdin tulee tietää alustan asettamat rajoitteet tarkasti. Tutkielman projektissa tämä haaste esiintyi kirjautumislogiikan tuottamisessa. Mendix on rajoittanut käyttäjänhallintaa turvallisuussyistä, joka teki alkuperäisestä käyttötapauksen toiminnallisuudesta mahdotonta toteuttaa. Tämä haaste näkyy selvästi myös sisäänkirjautumisen toteuttamiseen vaaditussa ajassa taulukossa 3. Mendixin asettamat rajoitteet on tehty turvaamaan sovelluksen toimintaa ja estämään kehittäjiä tekemästä turvallisuutta vaarantavia virheitä. Mendixillä sovellusta kehitettäessä kaikki ei ollut kuitenkaan haasteellista. Jotkin käyttötapaukset olivat kehittäjien kokemuksen mukaan erittäin nopeita toteuttaa.

Jo sprint 0 aikana huomattiin Mendixin tekevän sovelluskehityksen alkuvaiheesta erittäin tehokasta. Mendixillä sovelluksen kehittämisen aloitus on erittäin nopeaa. Varsinkin sovelluksia tyhjästä luotaessa kehittäjien ei tarvinnut käyttää aikaa sovelluksen konfiguraation muodostamiseen. Verkkoselainpohjaisten ja natiivien sovellusten luomisessa ei ole merkittäviä eroavaisuuksia ja molempia pystyy luomaan samalla työkalulla. Kehittäjät pitivät sovellusprojektin ympäristön asentamista ja konfiguraatiota erittäin pienitöisenä prosessina. Tämän ansiosta sovelluskehityksessä päästään keskittymään nopeasti asiakkaan näkökulmasta merkittäviin asioihin. Asiakkaan kanssa kommunikointia tehosti myös toimintalogiikan notaation yksinkertaisuus. Toimintalogiikan kehittäminen osoittautui myös nopeaksi kehittäjien kokemuksen mukaan.

Rajapintaintegraatioita REST-rajapintaan oli erittäin nopea toteuttaa automaattisen tietomallin generaation ja rajapintakutsujen muodostamisen yksinkertaisuuden ansionsta. Rajapintakutsujen kehittämisessä tarvitsi vain konfiguroida oikeat parametrit nanoflowiin sijoitettavaan aktiviteettiin, josta kutsu palautti ja generoi automaattisesti sovelluksessa käytettäviä

olioita. Rajapinnasta saatavan tiedon käyttäminen oli yksinkertaista käyttöliittymään asetettavilla widgeteillä. Kehittäjät kokivat, että Mendixin kehityksessä vahvasti läsnäoleva lähes pakotettu modulaarisuus hidasti toimintaa hieman, mutta he uskoivat sen tuovan suurissa projekteissa hyötyjä ylläpidettävyyteen. Samaa toimintalogiikkaa pystyttiin hyödyntämään monissa eri paikoissa, sillä niiltä vaadittiin mahdollisimman pientä sitoutuneisuutta muihin toimintalogiikkaa suorittaviin aktiviteetteihin. Myös käyttöliittymämuutoksien tekemisen koettiin olevan tehokasta. Projektin viimeisillä sprinteillä asiakas halusi muutoksia sovelluksen työnkulkuun. Vaatimuksiin pystyttiin vastaamaan erittäin tehokkaasti modulaarisuuden ja uusien näkymien datasta generoinnin ansiosta.

4 Yhteenveto

Tässä tutkielmassa toteutettiin kirjallisuuskatsaus aiheesta low-code sekä sovellusprojekti, jossa vanhasta varastonhallintasovelluksesta toteutettiin pikainventointitoiminnallisuus uuteen mobiilisovelluksen MVP-versioon. Tutkielman tavoitteena oli selvittää low-code-alustan käytön tuomia hyötyjä ja mahdollisia haittoja sovelluskehitykselle. Tutkielman tavoite pyrittiin saavuttamaan vastaamalla tutkimuskysymykseen: Mitä yhtenevyyksiä ja eroavaisuuksia tutkimuskirjallisuudessa low-code-alustoista esiintyvillä hyödyillä ja haitoilla on käytännön sovellusprojektissa havaittuihin Mendix low-code-alustan käytön hyötyihin ja haittoihin? Kirjallisuuskatsauksessa tavoitteena oli vastata tutkimuskysymyksen teoreettiseen osaan löytämällä tietoa low-codesta yleisesti, sen vahvuuksista, heikkouksista, parhaista käyttökohteista ja eri low-code-alustojen potentiaalisuudesta. Kirjallisuuskatsauksessa haluttiin myös avata sovellusprojektiin valittua low-code-alustaa Mendixiä teknologiana. Sovellusprojektissa pyrittiin löytämään käytännön kautta tutkielman sovellusprojektiin valitusta Mendix low-code-alustasta sovelluskehitystä hyödyttäviä elementtejä sekä myös kehitystä hidastavia haittoja. Sovellusprojektin tavoitteena oli selvittää mitä hyötyjä ja haittoja Mendix low-code-alustan käytöstä oli sovelluskehitysprosessille käytännössä.

Kirjallisuuskatsauksessa low-codesta ja low-code-alustoista selvisi lukuisia hyötyjä ja haittoja. Low-code on terminä uusi, mutta ajatusmallina vanha. Termi ei ole täysin vakiintunut, mutta se on jo laajalti käytössä. Bock ja Frank (2021) mukaan low-codea käytetään epäjohdonmukaisesti eri lähteissä. Low-codelle on kuitenkin esitetty eri määritelmiä, joista harvat esiintyvät vertaisarvioidussa kirjallisuudessa. Low-code-alustojen tarjoajilla on termille omat määritelmänsä (“Why low-code development matters right now” 2022; “The Definitive Guide to Low-Code Development” 2022). Low-code-teknologioiden soveltuvuus riippuu Galhardo ja Silva (2022) mukaan hyvin paljon kohdealueesta, jolla sovellus tulisi sijaitsemaan, sillä low-code-alustat käyttävät yleisesti omia DSL:iä. Kaikki low-code-alustat eivät sovellu kaikille kohdealueille, koska DSL:t menestyvät omilla kohdealueillaan (Galhardo ja Silva 2022). Täten, jos sovelluksen kehittämisessä hyödynnettävä low-code-teknologia on suunnattu kyseiselle kohdealueelle, sillä sovellusten toteuttaminen voi olla hyvin nopeaa. Low-coden uskotaan tehostavan sovelluskehitystä ja siitä on myös näyttöä (Vă-

duva ja Vălean 2021; Trigo, Varajão ja Almeida 2022). Yleiskäyttöisempiä low-code-alustoja, kuten Mendixiä ja OutSystemsiä voidaan käyttää laajemmalla aluella, mutta ne eivät todennäköisesti pärjäisi tietylle kohdealueelle sijoittuvalle low-code-teknologialle (Vincent ym. 2022). Lähdekirjallisuudessa on esitetty useita huolia kehitystyöstä low-code-alustoilla. Huolia on esitetty sovellusten ylläpidettävyydestä, sovellusten päivitettävyydestä, sovellusten laadusta, toimittajaloukusta ja alustan muokattavuudesta (Woo 2020; Cruz ym. 2021; Sanchis ym. 2019; Beranic, Rek ja Heričko 2020). Tutkielmassa käytetyn low-code-alustan dokumentaatiosta "Platform Evaluation Guide" (2023) löydettiin ratkaisuja moniin huoliin, kuten toimittajaloukkuun ja sovellusten päivitettävyyteen. Sovellusprojektissa löydettiin sekä hyötyjä että haittoja low-code-alustalla työskentelystä.

Tutkielman sovellusprojektissa kehittäjät pitivät sovelluksen kehityksen aloitusta erittäin nopeana. Lähdekirjallisuudessa ei oltu esitetty tähän liittyen väittämiä. Kehittäjät kokivat lomakepohjaisten yksinkertaisten ratkaisujen toteuttamisen erittäin nopeaksi low-code-alustalla. Lähdekirjallisuudessa esiintyi huomattavasti samankaltaisia mielipiteitä low-code-alustoilla lomakepohjaisten ratkaisujen toteuttamisesta (Alamin ym. 2023). Käytetyn low-code-alustan vahvuuksien hyödyntäminen oli myös sekä lähdekirjallisuudessa esiintynyt huomio, että myös kehittäjien painottama huomio sovellusprojektien nopeuttamisessa (Galhardo ja Silva 2022). Vastakohta tästä tuli myös lähdekirjallisuudessa ja sovellusprojektissa esille hidastavana tekijänä. Mackay ym. (2000) mukaan low-code sopii parhaiten sovelluksiin, joissa pääpaino on käyttäjän suorittamissa toiminnoissa, eikä niinkään laskennassa tai sovelluksen kompleksisuudessa. Sovellusprojektissa toteutettu sovellus ei sisältänyt paljoa toimintalogiikkaa, mutta ei voida väittää, että se olisi tehostanut toimintaa, koska tutkielmassa ei toteutettu paljon toimintalogiikkaa sisältävää sovellusta. Paljon toimintalogiikkaa sisältävän sovelluksen rajaa tai yksinkertaisen ja kompleksisen sovelluksen rajaa ei oltu myöskään määritelty lähdekirjallisuudessa, joten on erittäin vaikeaa verrata kirjallisuutta ja toteutettua sovellusta tämän väitteen kohdalla.

Lähdekirjallisuudessa oli mainittu, että low-code voi helpottaa liiketoiminnan ja kehittäjien vuorovaikutusta visuaalisuutensa avulla (Galhardo ja Silva 2022). Sovellusprojektissa todettiin, että liiketoiminnan henkilöiden oli huomattavasti helpompaa ymmärtää Mendixillä toteutettua toimintalogiikkaa, kuin perinteistä koodia. Lähdekirjallisuudessa lähes esiin-

tymätön rajapintaintegraatioiden tehokkuus esiintyi sovellusprojektissa edukseen. Rajapintaintegraatioiden kehittäminen oli kehittäjien mukaan vaivatonta ja tehokasta. Myös (Vincent ym. 2019) mukaan Mendix low-code-alusta tukee rajapintaintegraatioden kehittämistä hyvin. Tutkielman kirjallisuuskatsauksesta ja sovellusprojektista löytyi yhtäläisyyksiä, mutta sovellusprojektissa nousi myös esille asioita, joita ei oltu esitetty hyödyiksi tai haitoiksi lähdekirjallisuudessa. Sovellusprojektin perusteella ei kuitenkaan pystytä vastaamaan kaikkiin lähdekirjallisuudessa esiintyneisiin huoliin.

Tutkielmassa huomattiin, että kehittäjätiimi koki low-coden tehostavan sovelluskehitystä joiltain osin, mutta suuri osa sovelluskehityksestä on myös kiinni tehokkaasta määrittelystä ja suunnittelusta. Low-code-alustat voivat tehostaa toimintaa, kun niitä käytetään oikealla kohdealueella ja toimitaan low-code-alustojen vahvuuksia mukaillen. Tutkielman sovellusprojekti oli onnistunut. Asiakas oli tyytyväinen sovellusprojektin vaatimaan aikaan, sovelluksen MVP-version toimintaan sekä sen ulkonäköön. Tutkielman viimeisessä luvussa pohditaan tutkielman tuloksia, rajoitteita sekä mahdollisia jatkotutkimuskohteita.

5 Pohdintaa

Low-coden uskotaan saavuttavan suuren suosion keskisuurissa ja suurissa yrityksissä lähitulevaisuudessa (Vincent ym. 2019). Jos low-codeen halutaan panostaa yritystasolla, täytyy sen avusta sovelluskehityksessä olla myös jotain näyttöä. Näitä näyttöjä haluttiin tämän tutkielman tukemassa yrityksessä saada selville. Tutkielmaa varten toteutettua sovellusprojektia voitaisiin myös käyttää yrityksessä referenssinä mahdollisien asiakasprojektien voittamiseksi.

Tämän tutkielman tuloksena löydettiin low-coden mahdollistavan jonkintasoista sovelluskehityksen nopeuden parantumista. Tuloksista nähtiin myös, että low-code voi myös hidastaa sovelluskehitystä, jos sitä käytetään väärällä kohdealueella tai jos low-code-alustan vahvuuksia ei oteta huomioon. Kirjallisuus tukee monia tutkielmassa toteutetun sovellusprojektin tuloksia, joten niiden uskotaan olevan todenmukaisia, mutta tutkielmassa on monia rajoitteita, jotka vähentävät tulosten totuusarvoa.

Tutkielman tulosten totuudenmukaisuuden kannalta tutkielmassa oli monia rajoitteita. Tutkielman sisältämän sovellusprojektin kehittäjätiimi oli suurimmaksi osaksi vähän Mendixistä kokemusta omaavia kehittäjiä, joten sovellusprojektin toteutukseen kuluneet aikamäärät ovat todennäköisesti suurempia, kuin kokeneemmalla kehitystiimillä. Kehittäjien kokemukset ovat myös subjektiivisia käsityksiä sovelluskehityksen vaatimuksista, eivätkä ole tällöin välttämättä samoja, kuin kokeneemmilla kehittäjillä. Toteutettu sovellus on vain MVP-versio mahdollisesti toteutettavasta sovelluksesta, joten sovelluksen koko elinkaaren eri vaiheiden hyviin ja huonoihin puoliin low-codella ei voida ottaa kantaa tutkielmassa. Myös tutkielman kirjallisuuskatsauksen lähdekirjallisuus voi vaikuttaa tutkielman painottamiin huomioihin, jos lähdekirjallisuutta olisi suurempi määrä, tutkielman kirjallisuuskatsaukseen olisi voinut löytyä eri painotuksia. Tutkielmassa on toteutettu vain yksi sovelluksen MVP-versio, eikä dataa ole useiden sovellusten toteutuksista. Low-coden edut ja haitat voivat korostua näin pienellä otoskoolla huomattavan epätasapainoisesti. Yksi suurimmista tutkielman tulosten mahdollisista vääristäjistä oli se, että itse tutkija oli myös osana sovellusprojektin kehitystiimiä.

Jatkotutkimuskohteita tälle tutkielmalle olisi ilmeisimpänä tutkielmassa kehitetyn sovelluksen MVP-version laajentaminen ja käyttöönotto sekä sen koko elinkaaren tutkiminen. Tämä vaatisi huomattavasti enemmän resursseja. Tutkielman pohjalta voitaisiin myös tutkia samankaltaisten sovellusten MVP-versioiden rakentamista suuremmalla otoskoolla. Tällöin saataisiin huomattavasti tarkempaa tietoa juuri kyseisen aihealueen tutkimusdatan paikkansapitävyydestä. Low-coden tutkimus on ollut nousussa viime vuosina ja todennäköisesti se tulee myös nousemaan jatkossa. Tässä tutkielmassa selvisi myös, että low-coden määritelmä ei ole vielä vakiintunut ja siksi low-coden määritelmän ja termin määrittäminen voisi olla myös mielenkiintoinen jatkotutkimuskohde.

Lähteet

Alamin, M. A. A., G. Uddin, S. Malakar, S. Afroz, T. Haider ja A. Iqbal. 2023. "Developer Discussion Topics on the Adoption and Barriers of Low Code Software Development Platforms." *Empirical Software Engineering : An International Journal* 28 (1). https://doi.org/10.1007/s10664-022-10244-0.

Beranic, T., P. Rek ja M. Heričko. 2020. "Adoption and usability of low-Code/No-code development tools." *Faculty of Organization and Informatics Varazdin,* 97–103. https://www.proquest.com/conference-papers-proceedings/adoption-usability-low-code-no-development-tools/docview/2531366275/se-2.

Bhattacharyya, S. S., ja S. Kumar. 2021. "Study of deployment of "low code no code" applications toward improving digitization of supply chain management." *Journal of Science and Technology Policy Management.,* https://doi.org/10.1108/JSTPM-06-2021-0084.

Bock, A. C. I, ja U. Frank. 2021. "Low-Code Platform." *Business I& information systems engineering* 63 (6): 733–740. https://doi.org/10.1007/s12599-021-00726-8.

Bucaioni, A., A. Cicchetti ja F. Ciccozzi. 2022. "Modelling in low-code development: A multi-vocal systematic review." *Software and systems modeling.,* https://doi.org/10.1007/s10270-021-00964-0.

Cruz, M. A. A. da, H. T. L. de Paula, B. P. G. Caputo, S. B. Mafra, P. Lorenz ja J. J. P. C. Rodrigues. 2021. "OLP—A RESTful Open Low-Code Platform." *Future internet* 13 (10). https://doi.org/10.3390/fi13100249.

Galhardo, Pedro, ja Alberto Rodrigues da Silva. 2022. "Combining Rigorous Requirements Specifications with Low-Code Platforms to Rapid Development Software Business Applications". *Applied Sciences* 12 (19). ISSN: 2076-3417. https://doi.org/10.3390/app12199556. https://www.mdpi.com/2076-3417/12/19/9556.

Hurlburt, G. 2021. "Low-Code, No-Code, What's Under the Hood?" *IT professional* 23 (6): 4–7. https://doi.org/10.1109/MITP.2021.3123415.

Kadenic, Maja Due, Konstantinos Koumaditis ja Louis Junker-Jensen. 2023. “Mastering scrum with a focus on team maturity and key components of scrum”. *Information and Software Technology* 153:1–13. ISSN: 0950-5849. https://doi.org/10.1016/j.infsof.2022.107079. https://www.sciencedirect.com/science/article/pii/S0950584922001884.

Lundell, B., ja B. Lings. 2004. “Changing perceptions of CASE technology.” *The Journal of systems and software* 72 (2): 271–280. https://doi.org/10.1016/S0164-1212(03)00087-6.

Mackay, H., C. Carne, P. Beynon-Davies ja D. Tudhope. 2000. “Reconfiguring the User: Using Rapid Application Development.” *FSocial studies of science* 30 (5): 737–757. https://doi.org/10.1177/030631200030005004.

“Mendix Documentation”. 2023. Viitattu 13. helmikuuta 2023. https://docs.mendix.com.

“Platform Evaluation Guide”. 2023. Viitattu 31. tammikuuta 2023. https://www.mendix.com/evaluation-guide/.

Poe, Laura, ja Lionel Mew. 2022. “The effects of using the agile methodology as an instructional format for software development courses”. *Industry and Higher Education* 36 (5): 638–646. https://doi.org/10.1177/09504222211058658.

Ruscio, Di, Davide, Dimitris Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi ja Manuel Wimmer. 2022. *Low-code development and model-driven engineering: Two sides of the same coin?* 21:437–446. 2. https://doi.org/10.1007/s10270-021-00970-2.

Sanchis, R., Ó. García-Perales, F. Fraile ja R. Poler. 2019. “Low-Code as Enabler of Digital Transformation in Manufacturing Industry.” *Applied sciences* 10 (1). https://doi.org/10.3390/app10010012.

“The Definitive Guide to Low-Code Development”. 2022. Viitattu 28. syyskuuta 2022. https://www.mendix.com/low-code-guide/#lowcode-features-and-benefits.

Totterdale, Robert L. 2018. “CASE STUDY: THE UTILIZATION OF LOW-CODE DEVELOPMENT TECHNOLOGY TO SUPPORT RESEARCH DATA COLLECTION.” *Issues in information systems* 19 (2): 132–139. https://doi.org/10.48009/2_iis_2018_132-139.

Trigo, A., J. Varajão ja M. Almeida. 2022. “Low-Code Versus Code-Based Software Development: Which Wins the Productivity Game?” *IT Professional* 24 (5): 61–68. https://doi.org/10.1109/MITP.2022.3189880.

Văduva, Bogdan, ja Honoriu Vălean. 2021. “Designing a Low-Code CRUD framework.” *Carpathian Journal of Electronic & Computer Engineering* 14 (1). https://doi.org/10.2478/cjece-2021-0003.

Wang, L., S. Guan, W. Deng ja P. Lu. 2022. “ERP System Design for Hydrogen Equipment Manufacturing Industry Based on Low Code Technology.” *Mobile information systems* 2022:1–9. https://doi.org/10.1155/2022/5371471.

“Why low-code development matters right now”. 2022. Viitattu 28. syyskuuta 2022. https://powerapps.microsoft.com/en-us/what-is-low-code/.

Vincent, P., K. Iijima, M. Driver, J. Wong ja Y. Natis. 2019. “Magic quadrant for enterprise low-code application platforms”. *Gartner report.*

Vincent, P., K. Iijima, A. Leow, M. West ja O. Matvitskyy. 2022. “Magic quadrant for enterprise low-code application platforms”. *Gartner,* https://www.gartner.com/document/4022825?toggle=1&refval=353947599&ref=solrAll.

Woo, M. 2020. “The Rise of No/Low Code Software Development—No Experience Needed?” *Engineering (Beijing, China)* 6 (9): 960–961. https://doi.org/10.1016/j.eng.2020.07.007.