Juha-Matti Rahkola

Low-code alustojen vertailu ja arviointi

Tietotekniikan pro gradu -tutkielma

19. marraskuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Juha-Matti Rahkola

Yhteystiedot: juhamatti.rahkola@gmail.com

Ohjaaja: Jonne Itkonen, Tuomo Rossi

Työn nimi: Low-code alustojen vertailu ja arviointi

Title in English: Comparison and evaluation of low-code platforms

Työ: Pro gradu -tutkielma

Opintosuunta: Ohjelmisto- ja tietoliikennetekniikka

Sivumäärä: 45+0

Tiivistelmä: Tässä tutkielmassa tutkittiin low-code -sovelluskehitysalustoja vertailemalla niiden ominaisuuksia, toiminnallisuuksia, sekä käyttökokemusta keskenään. Vertaillut alustat olivat Visual LANSA, OutSystems, sekä Mendix. Tutkimus toteutettiin kvalitatiivisena empiirisenä toimintatutkimuksena, jossa tutkielman laatija kehitti kolmella eri low-code -sovelluskehitysalustalla ruokareseptejä ylläpitävän web-sovelluksen. Kehitysalustoilla toteutettiin tietty määrä kehityskertoja, joiden jälkeen kirjoitettiin dokumenttiin kehityskerran havainnot, tuntemukset ja työtunnit. Low-code -sovelluskehitys on nopeasti kasvava tietotekniikan ala, joka kaipaisi lisätutkimusta.

Avainsanat: pro gradu -tutkielmat, no-code, low-code, sovelluskehitys

Abstract: In this thesis three different low-code platforms and their features, functionalities and usability were studied by comparison. The platforms chosen were Visual LANSA, OutSystems and Mendix. A certain number of development instances were carried out, after which the session observations, feelings and workhours were documented. The study was executed as a qualitative empiric activity analysis, where the author developed a food recipe web application with three different low-code platforms. Low-code application development is a radiply growing line of information technology, which lacks further studying.

Keywords: master's theses, no-code, low-code, software development

Termiluettelo

Low-code

Vähäkoodinen. Sovelluskehitystapa, johon sisältyy hieman ohjelmointia.

No-code

Kooditon. Sovelluskehitystapa, johon ei sisälly lainkaan ohjelmointia.

Sovelluskehitys

Ohjelman tai ohjelmiston valmistaminen.

Sovelluskehitysalusta

Ohjelmointiympäristö, jossa sovellusta kehitetään ja joka tarjoaa työkalut kehitykseen.

Modulaarinen sovelluskehitys

Moduulien, eli itsenäisten ja yhteensopivien osien yhdistämiseen perustuva sovelluskehitys.

PAAS

Platform-as-a-service, ratkaisu, jossa alusta on ulkoistettu palvelu. Yleensä pilvipalveluna tarjottu kokonaisuus.

Citizen developer

Kansalaiskehittäjä. Henkilö, jolla ei ole ohjelmointitaitoa, mutta kehittää sovelluksia low-coden avulla.

Yritysekosysteemi

Kokonaisuus yrityksen erilaisia tuotteistettuja palveluita, joiden välinen integraatio kannustaa pysymään samassa tuoteperheessä.

IDE

Integrated development environment, eli ohjelmointiympäristö. Ohjelmisto, joka sisältää muun muassa tekstieditorin, kääntäjän ja useita muita merkittäviä sovelluskehityksen työkaluja ohjelmoinnin helpottamiseksi.

Kuviot

Sisällys

1 Johdanto

No- ja low-code ovat perin itsestäänselviä termejä. No-codella tarkoitetaan ohjelmistokehitystä, joka ei sisällä lainkaan ohjelmointia. Low-code puolestaan tarkoittaa ohjelmistokehitystä, joka sisältää vain hieman ohjelmointia. Low- ja no-coden perusidea on sovelluskehitys, missä laaditaan kokonaisuuksia yhdistelemällä modulaarisia elementtejä, jotka sisältävät tietynlaista toiminnallisuutta. Tämä toiminnallisuus voi olla vaikkapa lomakekomponentti, joka liitetään suoraan tietolähteeseen, jolloin lomake osaa alustaa kentät oikeisiin formaatteihin ja oikeilla otsikoilla vähäisellä vaivalla. Low-codessa ohjemointiosuus on tyypillisesti näiden komponenttien ja tietolähteen välisen toiminnallisuuden toteuttamista. Low-code -sovelluskehitysalustat ovat pääosin PAAS (Platform-as-a-service) ympäristöjä, joissa kehityksessä tarvittavat työkalut tarjotaan kehittäjälle johdonmukaisessa käyttöliittymässä. Tässä pro gradu -tutkielmassa keskitytään pääosin low-codeen, joten no-codea ei mainita aina erikseen low-coden yhteydessä tekstin selkeyden parantamiseksi.

Richardson ym. (2014) mukaan sovelluskehityksen osaajien pieni lukumäärä ja sovelluskehityksen hidas elinkaari eivät vastaa alan nykyisiä tarpeita. Low-coden eräs tavoite on mahdollistaa, että kuka tahansa, jolla on tarpeeksi tietoteknistä osaamista käyttää taulukkolaskentaa, voi toimia myös sovelluskehittäjänä. Tällaisia sovelluskehittäjiä kutsutaan low-coden yhteydessä termillä *citizen developer*, eli kansalaiskehittäjä. Lebens ym. (2022) mukaan kansalaiskehittämisellä tavoitellaan tilannetta, jossa muut kuin tietotekniikan osaajat voivat vastata välittömiin ja kriittisiin liiketoiminnallisiin tarpeisiin. Frank, Maier ja Bock (2021) kertovat, kuinka low-coden tarkoitus ei ole korvata ohjelmoijia liiketoiminnan osaajilla, vaan vähentää merkittävää sovelluskehittäjien tarvetta vähentämällä heidän työtaakkaansa. Sovelluskehittäjien työtaakkaa keventämällä saatetaan parhaimmassa tapauksessa välttää kehittäjien puutteesta syntyvä pullonkaula kehityksen elinkaaressa. Low-code vastaa myös sovelluskehityksen elinkaaren ongelmiin. Matka ideasta prototyyppisovellukseen on huomattavasti lyhyempi, nopeampi, sekä kehityskaaren kannalta ketterämpi, kuin perinteisissä sovelluskehitystavoissa. Di Ruscio ym. (2022) mainitsevat myös, kuin-

ka elinkaaren käytännön puoli, kuten verkkoisännöinti, resurssien jako, provisiointi ja analytiikka hoidetaan sovelluskehitysalustan toimesta.

Tämän pro gradu -tutkielman tavoitteena oli aluksi vertailla erilaisia tarjolla olevia ilmaisia low-code -alustoja, mutta täysin ilmaisten alustojen puute sai jättämään pois kyseisen rajauksen lähes välittömästi ja keskittymään yleisesti markkinoilla tarjolla oleviin alustoihin. Tutkielma jakautuu johdantoon, kolmeen päälukuun, sekä yhteenvetoon, jossa käydään läpi tulokset, johtopäätökset ja pohdinta. Johdannon jälkeisessä toisessa luvussa käsitellään yleisesti low-code -kehitystapaa, sekä tutustutaan low-coden historiaan, hyötyihin, haittoihin ja tulevaisuudennäkymiin. Lisäksi toisessa luvussa tutustutaan useisiin eri teknologioihin ja sovelluskehitystapoihin, jotka johtivat low-code -sovelluskehityksen syntyyn. Kolmannessa luvussa tarkastellaan tutkimuksen asetelmaa, tutustutaan tutkimuskysymyksiin, sekä käsitellään tavoitteita, strategiaa ja käytettyjä tutkimusmenetelmiä. Neljännessä luvussa esitellään käytetyt low-code -alustat, sekä tutustutaan niiden historiaan ja analysoidaan tuloksia. Neljännessä luvussa käydään läpi myös tutkimuksen aikana tehdyt havainnot, tutkimuksen tulokset, sekä käydään läpi syntynyt dokumentaatio. Viimeinen viides luku on yhteenveto, joka sisältää tutkimuksen tulokset ja johtopäätökset, sekä pohdinnan. Yhteenvedon jälkeen löytyy vielä osio tutkimuksessa käytetyille lähteille ja materiaaleille.

2 Low-code

Useiden eri ohjelmointikielten, sovelluskehitysmallien ja kehitysalustojen kehittyminen on mahdollistanut low-coden kehittymisen nykyiseen muotoonsa. Di Ruscio ym. (2022) jakavat low-codeen johtaneet merkittävät kehitysaskeleet seuraavasti:

- 1980-luku: 4GL ja CASE (Computer Aided Software Engineering) - Neljännen sukupolven ohjelmointikielet.
- 1990-luku: RAD, Rapid Application Development - Nopean kehityksen malli.
- 2000-luvun alku: End-User Development - Loppukäyttäjän kehitys / Kansalaiskehitys.
- 2000 - 2020: MDE/MDA/MDD, Model Driven Engineering/Architecture/Development - Malliperustainen ohjelmistokehitys.
- 2010 - Nykyhetki: LCDP, Low-code Development Platform - Low-code -kehitysalustat.

Neljännen sukupolven ohjelmointikielet ovat laajassa käytössä olevien kolmannen sukupolven ohjelmointikielten, kuten Java, C ja Pascal, korkeamman abstraktiotason kieliä, jotka pyrkivät olemaan edeltäjiään lähempänä luonnollisia kieliä. Shipley ja Jodis (2003) kertovat, kuinka neljännen sukupolven kielet mahdollistivat ohjelmoinnin helpottumisen ja tehostumisen vähemmän ammattitaitoisille ohjelmoijille. Alzahrani (2020) mukaan neljännen sukupolven ohjelmointikielten eräs ongelma on, että ne ovat liian kontekstisidonnaisia ja kehitetty ratkaisemaan jokin tietty tarve tai ongelma. CASE työkalut ovat neljännen sukupolven ohjelmointikielten avustustyökaluja, joiden avulla voi generoida sovelluksen osia järjestelmämalleihin perustuen.

Nopean kehityksen malli, eli *rapid application development* tai RAD, oli vastalause perinteisille kankeille sovelluskehitysmalleille, kuten vesiputousmalli. Gottesdiener (1995) kuvailee, kuinka RAD koostuu nk. aikalaatikoista, joilla tarkoitetaan kehityspyrähdyksiä, jotka sisältävät yhden tai useamman nk. palan, jotka ovat omia kehityskokonaisuuksiaan. RAD projektissa tuotetaan liiketoiminnallisesti tärkein osa

ensimmäisenä ja toimitetaan osa kerrallaan. Nopean kehityksen mallissa pyritään vaatimusmäärittelystä siirtymään suoraan iterointiin perustuvaan kehitysvaiheeseen, jossa aloitetaan käyttäjän näkökulmasta toteuttamaan halutut ominaisuudet ja syötteet. Seuraava työvaihe koostuu kehitystyöstä, ohjelmoinnista ja testaamisesta. Kaksi edellämainittua vaihetta ovat iteratiivisia ja niiden välillä voidaan siirtyä uusien tarpeiden mukaan. Viimeistelyvaihe sisältää siirtymän uuden sovelluksen käyttöön, testaamiseen ja koulutukseen.

Kleppe, Warmer ja Bast (2003) kertovat, kuinka MDA, eli malliperustainen arkkitehtuuri on ohjelmistokehityksen kehys, jossa kehitysprosessia ajaa järjestelmän kuvaaminen mallien avulla. He kuvaavat malleja todellisuuden abstraktioiksi, eli ideoiksi ja luonnoksiksi, jotka eivät ota kantaa toteutustapaan tai yksityiskohtiin. Mallit ovat täten uudelleenkäytettäviä ja useilla eri tavoilla toteutettavissa. Malliperustaisesta ohjelmistokehityksestä löytyy aineistoa nimillä MDE, eli *model driven engineering*, MDA, eli *model driven architecture*, sekä MDD, eli *model driven development*. Di Ruscio ym. (2022) vertaavat malliperustaista ohjelmistokehitystä ja low-code -sovelluskehitystä kolikon kahteen puoleen. Heidän mukaansa kehitystavat jakavat paljon samankaltaisuuksia, mutta myöskin sen verran eroja, että ne eivät ole täysin yhteensopivia. Molemmat pyrkivät kasvattamaan sovelluskehityksen abstraktiotasoa ja piilottaa käytännön toteutuksen yksityiskohtia. Monet malliperustaisen ohjelmistokehityksen mallit eivät puolestaan pyri millään tavalla vähentämään ohjelmoinnin määrää, eivätkä kaikki low-code -lähestymistavat ole malliperustaisia.

Low-code termi vakioitui Forrester Researchin julkaisemassa ja Richardson ym. (2014) laatimassa raportissa, jossa käsiteltiin low-code -henkisten alustojen kasvavaa markkina-alaa. Julkaisussa low-codea kuvailtiin seuraavasti: ”Alustoja, jotka mahdollistavat nopean sovellusten toimituksen minimaalisella ohjelmoinnilla ja nopealla käyttöönotolla reaaliaikaista dataa hyödyntäville liiketoimintajärjestelmille”. Forrester Researchin Rymer (2017) tarkensi low-coden määritelmää kolme vuotta myöhemmin seuraavasti: ”Tuotteita tai pilvipalveluita sovelluskehitykseen, jotka hyödyntävät malliperustaisia tekniikoita ohjelmoinnin sijaan, sekä ovat saatavilla asiakkaille matalalla tai olemattomalla aloituskustannuksella tai ilmaisella koulutusjak-

solla, jonka hinnoittelu määritellään alustan arvoon suhteutettuna". Toisen markkinointitutkimusyrityksen Gartnerin tutkijat Vincent ym. (2019) kuvailivat low-code -kehitysalustoja kaksi vuotta myöhemmin seuraavasti: "Low-code kehitysalusta on sovellusalusta, joka tukee nopean kehityksen mallia, yhden askelen käyttöönottoa, toimeenpanoa ja hallintaa käyttäen deklaratiivisia korkean tason ohjelmointiabstraktioita, kuten malli- ja metadataperustaisia ohjelmointikieliä. Ne tukevat käyttöliittymän, liiketoimintalogiikan ja tietopalvelujen kehitystä, sekä parantavat tuottavuutta palvelun toimittajan vaihtamisen kustannuksella verrattuna perinteisiin sovellusalustoihin". Cabot (2020) mukaan low-code ei ole mitenkään uusi teknologia, vaan innovatiivinen yhdistelmä useita olemassaolevia teknologioita ja kehityskonsepteja. Vaikka Low-code terminä vakioitui vuonna 2014, oli ennen tätä jo huomattava määrä low-code -sovelluskehitysalustoja markkinoilla. Low-coden eräs ideaali on se, että kuka tahansa voi toimia ohjelmistokehittäjänä ilman ohjelmointitaitoa tai tietoteknistä osaamista. Huolimatta low-coden huomattavista hyödyistä, vaatii se silti perinteiset sovelluskehityksen vaiheet, kuten huolellisen suunnittelun ja testaamisen. Khorram, Mottu ja Sunyé (2020) kertovat, kuinka jotkin alustat tarjoavat työkaluja suunnitteluun, mutta useat low-code -alustat eivät tarjoa kovinkaan kattavia testausmenetelmiä.

Hurlburt (2021) mainitsee, kuinka tekoälytoteutukset, koneoppiminen ja robottinen prosessiautomaatio ovat selkeästi kasvattaneet low-coden näkyvyyttä markkinoilla. Useat low-code alustat ovat tuoneet lähiaikoina muun muassa liittimiä, jotka mahdollistavat esimerkiksi DALL-E ja ChatGPT -tekoälytoteutusten käytön kehitysalustoilla. Gartner (2022) ennakoi vuoden 2022 joulukuussa low-code teknologioiden kasvattavan markkinaosuuttaan 25% vuodessa välillä 2023 – 2024. Gartner myös ennakoi, että vuoteen 2026 mennessä 80% käyttäjistä hyödyntävät low-code -työkaluja IT-alan yritysten ulkopuolella.

2.1 Low-coden vahvuudet

Low-code -lähestymistavalla on selkeitä vahvuuksia, jotka erottavat sen muista sovelluskehitystavoista. Luo ym. (2021) ovat selvittäneet ihmisten käsityksiä ja mie-

lipiteitä low-code -sovelluskehityksestä. Low-code -kehitysalustoista löytyy muun muassa seuraavia vahvuuksia:

- sovelluskehityksen nopeus ja ketteryys
- sovelluskehityksen tuottavuus
- sovelluskehityksen helppous
- sovelluksen liitettävyys muihin palveluihin
- hinta.

Ehdottomasti suurin vahvuus low-codessa on kehittämisen nopeus. Richardson ja Rymer (2016) selvittivät, että low-code -kehitysalustojen käyttö nopeutti useiden yritysten sovelluskehitysprosessia viisi–kymmenkertaisesti. Low-code nopeuttaa selkeästi myöskin käyttöönottoa ja ylläpitoa. Sahay ym. (2020) kuvailevat modulaarisen käyttöliittymäkehityksen olevan low-coden suurin nopeuttava tekijä. Kehitettyjä kokonaisuuksia voidaan myöskin käyttää useissa samankaltaisissa sovelluksissa modulaarisuuden ansiosta. Sanchis ym. (2019) kertovat, kuinka low-code mahdollistaa hyvän kilpailuaseman verrattuna perinteiseen sovelluskehitykseen, sillä vauhdikas sovelluskehitysprosessi mahdollistaa nopeamman vastauksen äkillisesti syntyneisiin käyttäjien tai markkinoiden tarpeisiin. Sanchis ym. (2019) mainitsevat low-code -kehityksen olevan myöskin ketterää, sillä sovelluskehityksessä voidaan vastata nopeasti uusiin ilmeneviin tarpeisiin ja vaatimuksiin. Sovellusta voidaan esimerkiksi havainnollistaa asiakkalle ja muuttaa jopa välittömästi demonstraation aikana.

Frank, Maier ja Bock (2021) kertovat, kuinka low-coden suurin houkutus yrityksille on tuottavuuden kasvaminen. Low-code mahdollistaa parhaillaan huomattavan tuottavuuden kasvun, sillä useat low-code -alustat on suunniteltu juurikin yritystarpeiden täyttämiseen. Frank, Maier ja Bock (2021) mainitsevat myös, että useat low-code -alustat kehuvat alustansa laajaa kirjoa erilaisia projektinhallintatyökaluja. Luo ym. (2021) mainitsevat, kuinka yksi kannustava tekijä low-coden käyttöönotolle yrityksissä on prosessien automaation mahdollistavat työnkulut, joilla on merkittävä vaikutus tuottavuuteen.

Smith ym. (2020) mukaan low-code -sovelluskehitysalustoista on kehitetty tarkoituksella helppokäyttöisiä, sillä niiden tavoite on tehdä kehityksestä helposti lähestyttävää ja yksinkertaista. Alustojen tarvitsee olla niin helppokäyttöisiä, että henkilö ilman ohjelmointikokemusta tai suurempaa tietoteknistä taitoa pystyy, ainakin osittain, niitä käyttämään. Tällaista kehitystä kutsutaan kansalaiskehittämiseksi.

Sahay ym. (2020) kertovat, kuinka low-code -alustoilla on yleensä erittäin laaja ja helppokäyttöinen liitettävyys muihin alustoihin ja palveluihin integraatioiden avulla. Alustat voi tyypillisesti liittää johonkin tietolähteeseen, tavallisesti johonkin pilvipalveluun alustojen PAAS toteutuksesta johtuen, tai joissain tapauksissa esimerkiksi palveluntarjoajan hallitsemaan tietokantaan. Useat alustat tarjoavat myös toimivia ja helppokäyttöisiä rajapintoja, joiden avulla sovelluksen saa yhdistettyä vaikkapa sähköpostipalveluihin tai sosiaalisen median kanaviin. Osa näistä rajapinnoista saattaa olla maksullisia tai vaatia erillisen kolmannen osapuolen lisenssin toimiakseen.

Hinta on sekä low-coden vahvuus, että heikkous. Hinnasta ja projektien kustannuksista puhutaan hieman low-coden heikkoudet osion jälkeen.

2.2 Low-coden heikkoudet

Kuten muukin sovelluskehitys, myöskään low-code ei ole pelkästään positiivista. Luo ym. (2021) selvittivät low-code -sovelluskehityksen vahvuuksien lisäksi myöskin heikkouksia. Low-coden merkittävimmät heikkoudet ovat:

- sovelluskehityksen jäykkyys
- kehitettävän sovelluksen alustasidonnaisuus
- kehitysalustan käytön opettelu
- hinta.

Vaikka eräs low-coden vahvuuksista on ketteryys, eräs sen suurimmista heikkouksista on jäykkyys. Ketteryys ja jäykkyys tosin kohdistuvat eri osiin sovelluskehitystä. Jos käytetty sovellusalusta ei sisällä ominaisuuksia tai toimintoja joita sovellus

vaatii, saattaa ohjemoinnin tai komponenttien kustomoinnin määrä kasvaa huomattavasti. Kaikki alustat eivät tarjoa mahdollisuutta koodin tai mukautettujen komponenttien lisäämiseen. Luo ym. (2021) mainitsevat, kuinka oikein käytettynä low-code toteutukset ovat hyödyllisiä ja tehokkaita, mutta väärin käytettynä saattavat toimia hidasteena tai rajoitteena. On olemassa myös toteutuksia jotka on hyvin vaikeita, tai jopa mahdottomia toteuttaa joillain low-code -alustoilla. Esimerkiksi erittäin monimutkainen ja suuri sovellus, kuten toiminnanohjausjärjestelmä on vaikea pilkkoa tarpeeksi pieniin osiin, että low-code -alusta ei varoittaisi suorituskykyongelmista.

Low-coden toisena heikkoutena on sen alustasidonnaisuus. Sufi (2023) mainitsee yhdeksi merkittäväksi syyksi sen, kuinka kehitysalustan omistama yritys haluaa pitää asiakkaat omassa yritysekosysteemissään. Sovellus jää kehitysvaiheessa komponenttiensa takia usein sidonnaiseksi käytettyyn sovelluskehitysalustaan, jonka lähdekoodia alustan kehittäjät eivät usein julkaise. Sovelluskehitysympäristöt ovat itsessään tuotteita, joiden innovaatioita halutaan suojata. Sovelluksilla on tapana kasvaa käyttäjäkunnan kasvaessa käyttäjien palautteen perusteella. Tällöin saatetaan saavuttaa tilanne, jossa havaitaan, että käytetty kehitysalusta ei enää vastaakkaan kehityksen tarpeita. Jatkuvat muutokset ja päivitykset alustan ulkoasuun ja ominaisuuksiin voivat aiheuttaa kehittäjille huomattavaakin päänvaivaa ja hidastaa kehittämistä. Jos kehitysalustaa haluaa vaihtaa toiseen, joudutaan usein koko sovellus kehittämään uudelleen. Frank, Maier ja Bock (2021) kuvailevatkin yhdeksi low-coden suurista kompastuskivistä kehitysalustojen suuren lukumäärän, sekä yhteensopimattomuuden eri tuotteiden välillä.

Huolimatta siitä, että low-codea kaupataan tulevaisuuden ratkaisuna, jonka avulla kuka tahansa voi olla sovelluskehittäjä, ei totuus ole ehkä aivan niin optimistinen. Low-code -sovelluskehitys on silti sovelluskehitystä ja vaatii IT-alan ymmärrystä. Bernsteiner ym. (2022) selvittivät kansalais- ja ammattikehittäjien low-code kehityskokemuksia. Tutkimuksessa havaittiin, että kansalaiskehittäjien toteuttamat sovellukset eivät aina täyttäneet tarvittuja laatuvaatimuksia ja vaativat täten apua ja ohjausta ammattikehittäjiltä. Tietokannat, metodit, suoritusajat, turvallisuus ja useat

muut IT-alan käytänteet ja keinot pätevät low-code -kehityksessä samoin kuin perinteisessä sovelluskehityksessä. Low-code -alustat ovat toteutustavoissaan, käytetyissä ohjelmointikielissään ja sovelluskokonaisuuksissaan täysin erilaisia toisistaan. Tietyn low-code -alustan osaaminen, ymmärrys ja ammattitaito ei takaa menestystä toiseen alustaan siirryttäessä.

2.3 Low-coden vaikutukset sovelluskehityksen hintaan

Low-coden hinnoittelun voi käsittää sekä vahvuutena, että heikkoutena. Sufi (2023) kertoo, kuinka low-code -alustojen vaatimat lisenssit ja muut toiminnallisuudet ovat jatkuva menoerä, joka jatkuu myös kehitetyn sovelluksen ollessa jo tuotannossa. Alustojen hinnat saattavat vaihdella hyvinkin paljon, sillä eri alustat ovat tarkoitettu eri käyttötarkoituksiin ja erikokoisille yrityksille. Täysin ilmaisia low-code -alustoja ei tutkielman laatimisen aikana tehdyn kartoituksen mukaan ole markkinoilla, vaan alustojen käyttö vaatii maksun jossakin sovelluksen elinkaaren vaiheessa. Alustoja, joilla kehittäminen on ilmaista löytyy useita, mutta näillä alustoilla ei ole mahdollista julkaista sovellusta ilmaiseksi käytettäväksi. Hinta ja rajoitetut kokeiluajat ovat tekijöitä, jotka saattavat saada yksityishenkilöt karttamaan low-code -alustoja. Tästä syntyy tilanne, jossa low-code keskittyy pitkälti yritysten käyttämäksi kehitystavaksi, eivätkä yksityishenkilöt päädy sovelluskehitysalustoja käyttämään. Sahay ym. (2020) mainitsevat, että lisenssit ovat usein yksittäisille kehittäjille. Kehittäjämäärän lisäksi yritysten täytyy osata myöskin ennakoida, kuinka useita sovelluksia ja ympäristöjä he tarvitsevat.

Isot, pitkäikäiset ja monia eri teknologioita yhdistävät sovellusprojektit ovat kalliita useista eri syistä. Kompleksisuus, suunnittelu, projektinhallinnan keinot, sekä pitkä kehitysaika ja suuri kehittäjien määrä johtavat hinnan eksponentiaaliseen nousuun. Low-code -sovelluskehitys saattaa lyhentää suunnitteluvaihetta, sillä kehittäjät voivat siirtyä heti prototyyppien kehittämiseen. Low-code -sovelluskehitysalustat ovat kaupallisia tuotteita, joten ne on tavalla tai toisella monetisoitu. Usein alustat vaativat joko maksullisen kertamaksuisen lisenssin, tai käyttäjää laskutetaan käytön mukaan.

3 Tutkimusmenetelmä

Tutkimuksen tarkoituksena on vertailla kolmea eri low-code -sovelluskehitysalustaa, sekä kehittää niillä kaikilla luvussa 3.2 esitetyn kaltainen web-sovellus. Tutkimus toteutetaan kvalitatiivisena empiirisenä toimintatutkimuksena. Mitään valituista kehitysalustoista ei ole testattu ennen kehityksen aloittamista, joten alustan perehdytysmateriaali ja alustan käytön opettelu ovat osana vertailua. Kirjoittaja toimii itse alustojen testaajana, eli testaaja on tietotekniikan maisteriopiskelija, jolla on noin vuoden työkokemus low-code -sovelluskehityksestä Microsoftin Power Platform -kehitysalustalla.

3.1 Tutkimusaihe ja -strategia

Markkinoilla on tarjolla useita no- ja low-code -alustoja eri tarkoituksiin web-sovellusten kehittämisestä aina suurempiin kokonaisuuksiin. Alustoista useiden markkinat kohdistuvat yrityskäyttöön, mutta myös yksityishenkilöille löytyy kehitysratkaisuja. Tutkielman tutkimuskysymyksenä on: *millä tavoilla samoihin tarkoituksiin kehitetyt low-code -alustat eroavat ja onko niiden käytössä, perehdytyksessä, materiaaleissa, hinnoittelussa ja suorituskyvyssä merkittäviä eroja?* Tutkimus on tarpeellista, sillä markkinoille on ilmestynyt valtava määrä erilaisia low-code -alustoja ja niiden välillä on huomattavia eroja. Yksityishenkilöille ja yrityksille on tärkeää päätyä juuri heidän tarpeisiinsa sopivaan low-code -alustaan, sillä kehitysalustan vaihtaminen kesken kehityksen saattaa aiheuttaa huomattavia kustannuksia. Toisena tutkimuskysymyksenä esitetään myös: *mitä tarkoitetaan low-code -alustalla ja mikä tekee alustasta low-code -kehitysalustan?* Low-codelle on esitetty määritelmä, joka on otettu yleisesti käyttöön tieteenalalla, mutta low-coden -määritelmä on epäselvä ja kaipaa tarkennusta.

Tutkimus toteutetaan valitsemalla kolme erilaista markkinoiden johtavaa, suosittua tai merkittävää low-code -alustaa, jotka ovat kaikki tarkoitettu web-sovellusten kehittämiseen. Sovellukseksi valittiin web-sovellus, sillä web-sovellusta ei tarvitse pakata ajettavaan muotoon, vaan se on vapaasti käytettävissä selaimessa. Kaikil-

la näistä valituista sovellusalustoista kehitetään ennalta suunnitellun mukainen sovellus. Sovelluksen kehitykseen on määrätty tietty määrä työjaksoja, joiden jälkeen dokumentoidaan tulokset, ajatukset ja huomiot kuvion 1 mukaiseen dokumenttiin.

Alustan nimi

Kehitys	Syötä tunnit	Opiskelu	Syötä tunnit
Päivämäärä	Syötä päivämäärä		

Havainnot:

1. Havainto

Syötä havainnot

2. Havainto

Syötä havainnot

Kuvio 1. Dokumentti, johon kirjoitetaan kehityskertojen päivämäärät, havainnot ja käytetyt työtunnit.

Alustat, joita tutkimuksessa päädyttiin käyttämään ovat: Visual LANSA (2023b), OutSystems (2023a) ja Mendix (2023a). Alustoissa vertaillaan ainakin seuraavia ominaisuuksia:

- alustan käytön helppous
- komponenttien käytettävyys, joustavuus ja muokattavuus
- vaadittu koodaamisen määrä, jotta komponentit saa toimimaan halutulla tavalla
- hinta ja skaalautuvuus, esim. minkä kokoisille yrityksille ja sovelluksille alusta on tarkoitettu.

Kehityskerran päätösdokumenttiin kirjataan kehityskerran päivä, sekä alusta jolla kehitystyötä tehtiin. Kehityskerran päätösdokumentti täytetään välittömästi kehityskerran päätyttyä. Dokumentiin kirjattavat tärkeimmät havainnot ovat:

- kehityskerran jälkeiset tuntemukset
- mitä tehtiin
- erityishuomioita.

Dokumenttiin kirjataan välittömästi kehityskerran herättämät tuntemukset. Tuntemuksiin voi avata esimerkiksi omaa ärtymystä jonkin ominaisuuden tai toiminnon käytöstä, tai positiivista yllätystä. Havainnoissa avataan, mitä käytännössä tehtiin kehityskerran aikana. Tässä osiossa mainitaan kehityskerralla lisätyt komponentit, niiden väliset yhteydet ja toiminnallisuudet. Erityishuomioina voi puolestaan mainita esimerkiksi seuraavaan alustan kehityskertaan kohdistuvia huomioita tai havaintoja, jotka pitää ottaa huomioon seuraavalla kehityskerralla.

3.2 Toteutettava sovellus

Alustoilla toteutetaan ruokareseptisovellus. Reseptisovellus valittiin toteutettavaksi, sillä se on riittävän yksinkertainen ja yksikselitteinen, että sovellus pysyy helposti ymmärrettävänä, mutta samalla tarpeeksi monimuotoinen, että toteutettavista ominaisuuksista kyetään tekemään informatiivisia havaintoja. Sovelluksessa pystyy selaamaan olemassa olevia reseptejä, sekä lisäämään uusia reseptejä. Resepti sisältää valmistusajan, raaka-aineet, sekä monivaiheiset ohjeet reseptin toteuttamiseen. Jos kehittämiseltä jää aikaa, reseptisovelluksessa on tavoitteena lisätä myös satunnaiseen reseptiin navigoiminen, sekä käyttäjän syötteeseen ennakoiva hakukenttä raaka-aineiden lisäämisen yhteyteen.

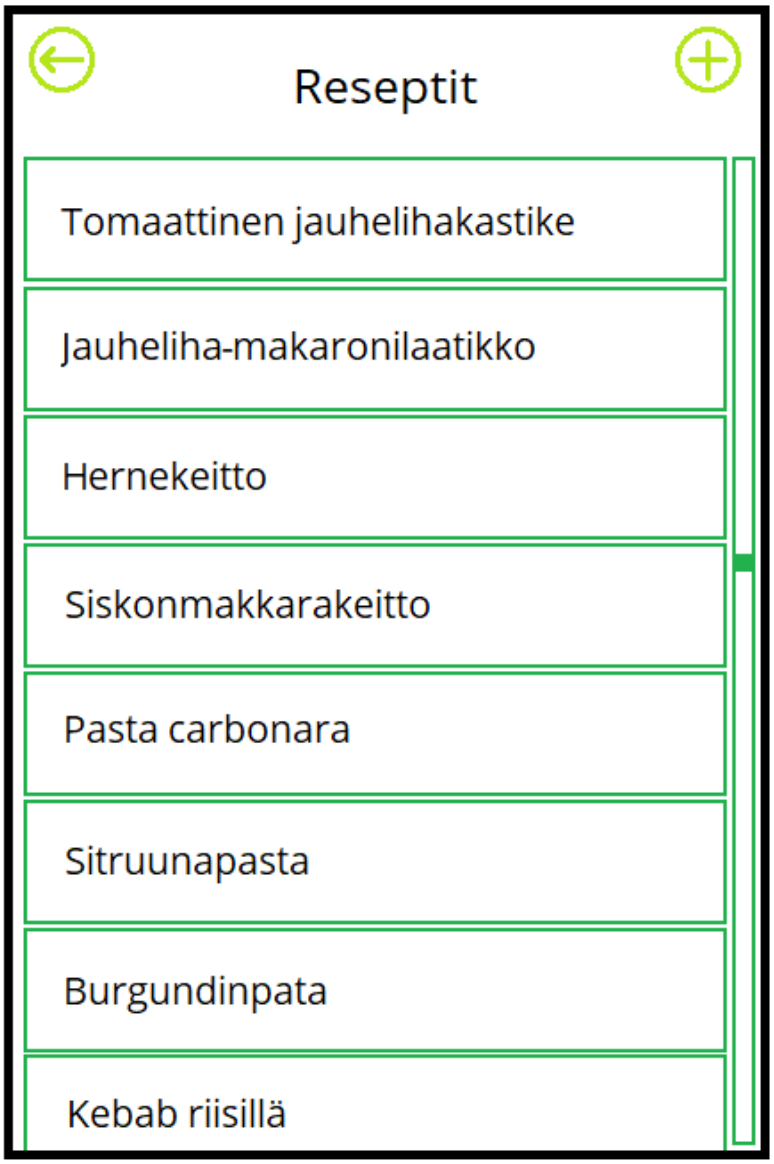
Kuviossa 2 esitellään sovelluksen etusivun näkymä. Sovelluksen etusivu on perin tyypillinen valikkonäkymä, jossa tervehditään käyttäjää ja havainnoidaan sovelluksen käyttötarkoitusta ruoka-aiheisella kuvalla. Etusivulta pääsee siirymään reseptit- tai ostoslistasivuille, tai valitsemaan satunnaisen reseptin esimerkiksi viikon ruokaideoita miettiessä. Etusivun oikeassa yläkulmassa esitetyn hammasratasikonin painaminen ohjaa ylläpitosivulle. Ikoni näkyy vain pääkäyttäjille.

Kuviossa 3 on esitelty etusivun *Reseptit* -valinnan takaa löytyvä seuraava sivu. Sivulla voi lisätä uuden reseptin painamalla yläoikealta löytyvää pluspainiketta. *Re-*

Kuvio 2. Sovelluksen etusivu.

septit -listauksessa voi selata kaikkia sovellukseen tallennettuja reseptejä. Reseptipainikkeiden taustalle asetetaan osittain läpinäkyväksi kuva reseptin mukaisesta ruoka-annoksesta, joka on syötetty reseptin lisäämisen yhteydessä.

Kuviossa 4 esitellään reseptien lisäämissivu. Reseptien lisäämiseen siirrytään reseptisivulta. Sivulla lisätään reseptiltä vaaditut tiedot, eli reseptin nimi, kokkausaika, vaihtoehtoinen kuva annoksesta, haluttu määrä ainesosia, sekä haluttu määrä kok-



Kuvio 3. Etusivun *reseptit* -valinnan takaa löytyvä reseptien listaus.

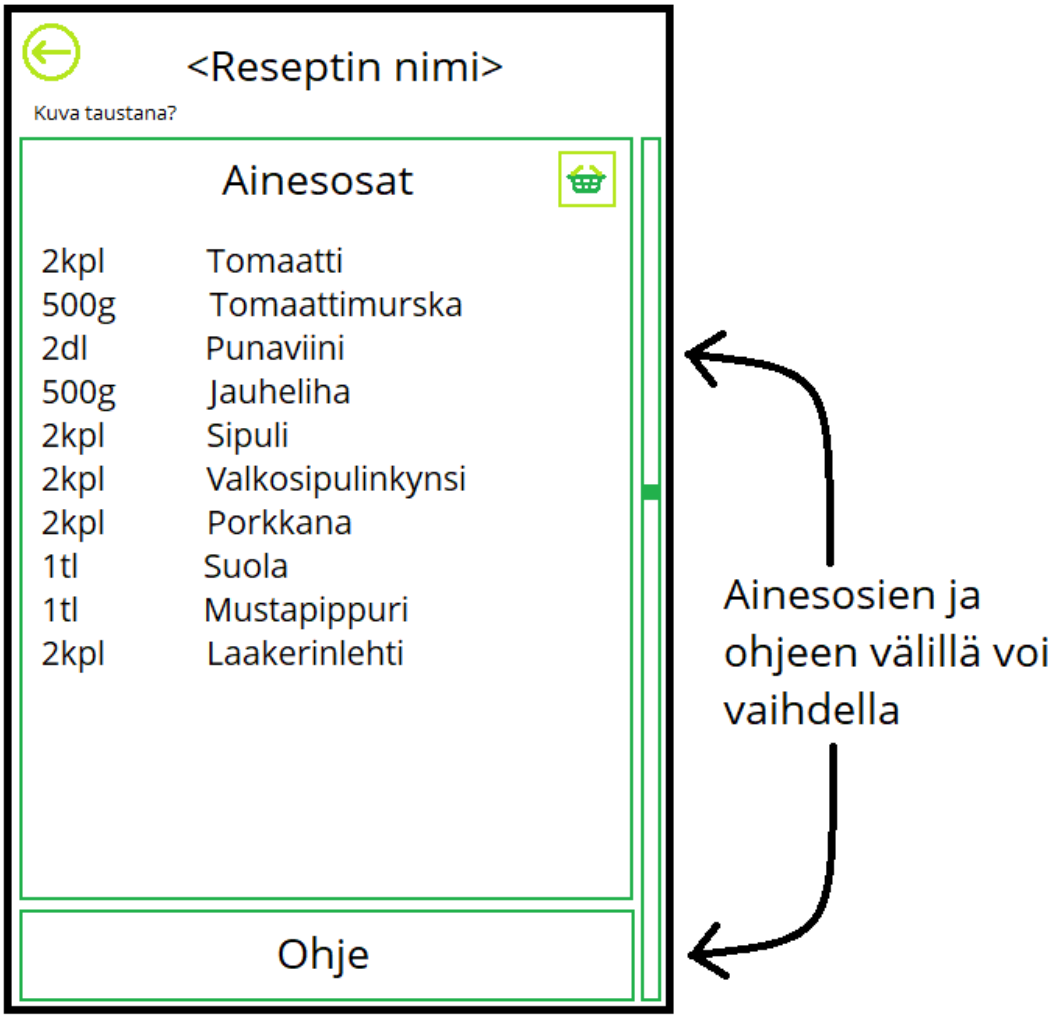
kausohjeen vaiheita. Ainesosien ja ohjeen vaiheiden minimimäärä on yksi. *Tallenna* -painike aktivoituu vasta, kun kaikki reseptin vaatimat vähimmäismääritykset täyttyvät.

Kuviossa 5 esitellään sovelluksen reseptisivu. Reseptisivun taustana on kuva reseptin mukaisesta annoksesta haalennettuna siten, että se ei häiritse reseptin lukemista. Reseptin nimi toimii sivun otsikkona. Sivulla esitellään ensimmäisenä ainesosat.

Kuvio 4. Reseptisivun yläoikeassa kulmassa sijaitsevan pluspainikkeen kautta aukeava reseptin lisäyssivu.

Ainesosaikkunasta voi myös lisätä halutessaan kaikki reseptin ainesosat ostoskoriin myöhempää käyttöä varten. Ainesosat esitetään listamuodossa. Ainesosaikkunan alapuolelta löytyy *ohje* -painike, jota painamalla ainesosaikkuna kutistuu painikkeeksi ja ohjeikkuna aukenee. Tämä näkymä on esitetty kuviossa 6. Ohjeikkunassa on kerrottu reseptin arvioitu valmistusaika, sekä reseptin kokkausvaiheet. Ainesosa- ja ohjeikkunan välillä voi loikkia vapaasti painamalla vastaavaa paini-

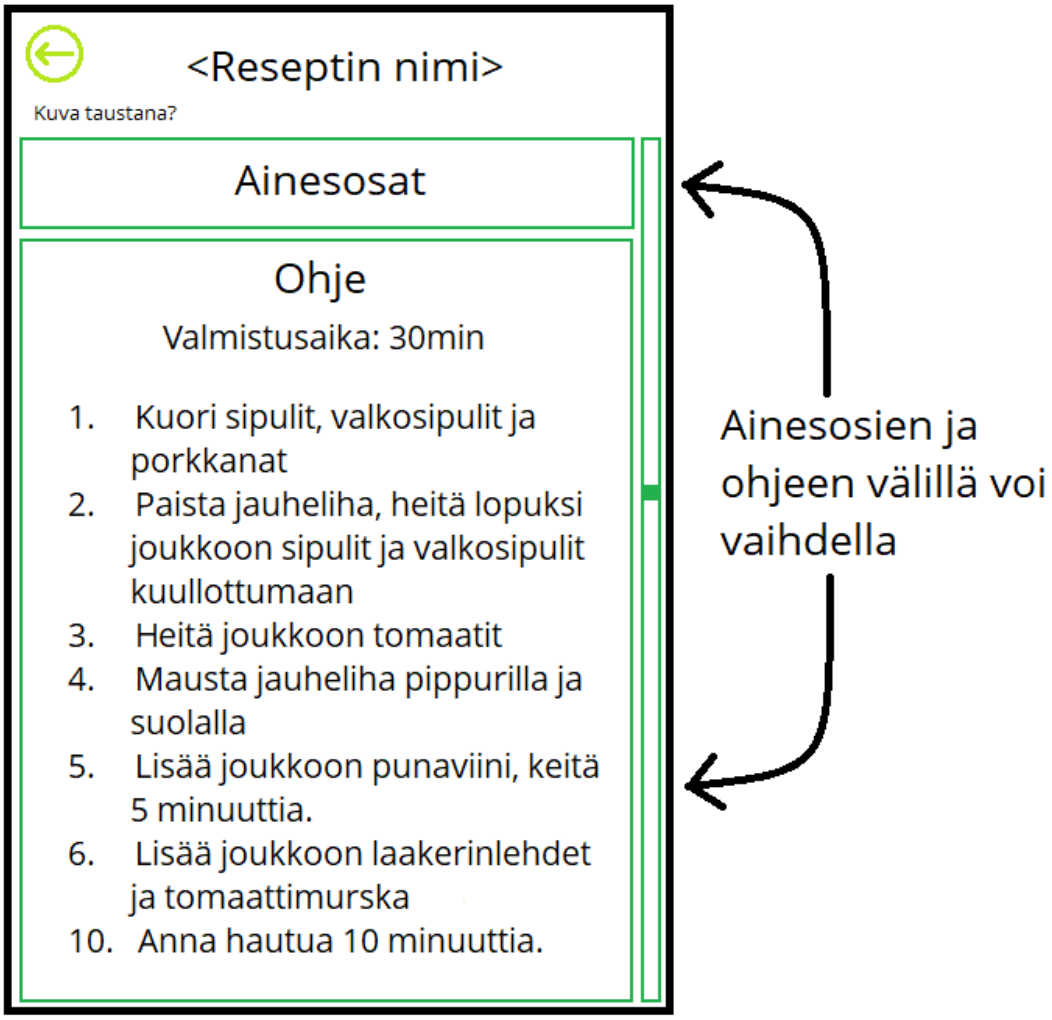
ketta.



Kuvio 5. Reseptilistauksesta valittu resepti tai etusivun *satunnainen resepti* -valinta johtavat reseptisivulle.

Kuviossa 7 esitellään ylläpitosivu, mihin pääsee siirtymään etusivun hammasratasikonista. Ikoni esitetään vain pääkäyttäjille. Ylläpitosivulla pääkäyttäjä voi muokata ja poistaa sovellukseen lisättyjä reseptejä, sekä lisäämään raaka-aineita ja ainesosia. Raaka-aineet ja ainesosat esitetään pikahakuna reseptin lisäämisen yhteydessä.

Tarkoituksena on toteuttaa myös ainesosaikkunaan ennakoiva haku, josta esimerkki löytyy kuviosta 8. Käyttäjän kirjoittaessa sanaa ainesosan tekstilaatikkoon, esitetään alle ehdotuksia, jotka vastaavat siihen asti kirjoitettua sanaa. Ennakoiva haku



Kuvio 6. Reseptisivulla *ohje* -painiketta painaessa *ainesosat* -ikkuna pienenee ja *ohje* -osuus aukenee.

helpottaa reseptin kirjoittamista ja ohjaa käyttäjän ajatusprosessia.

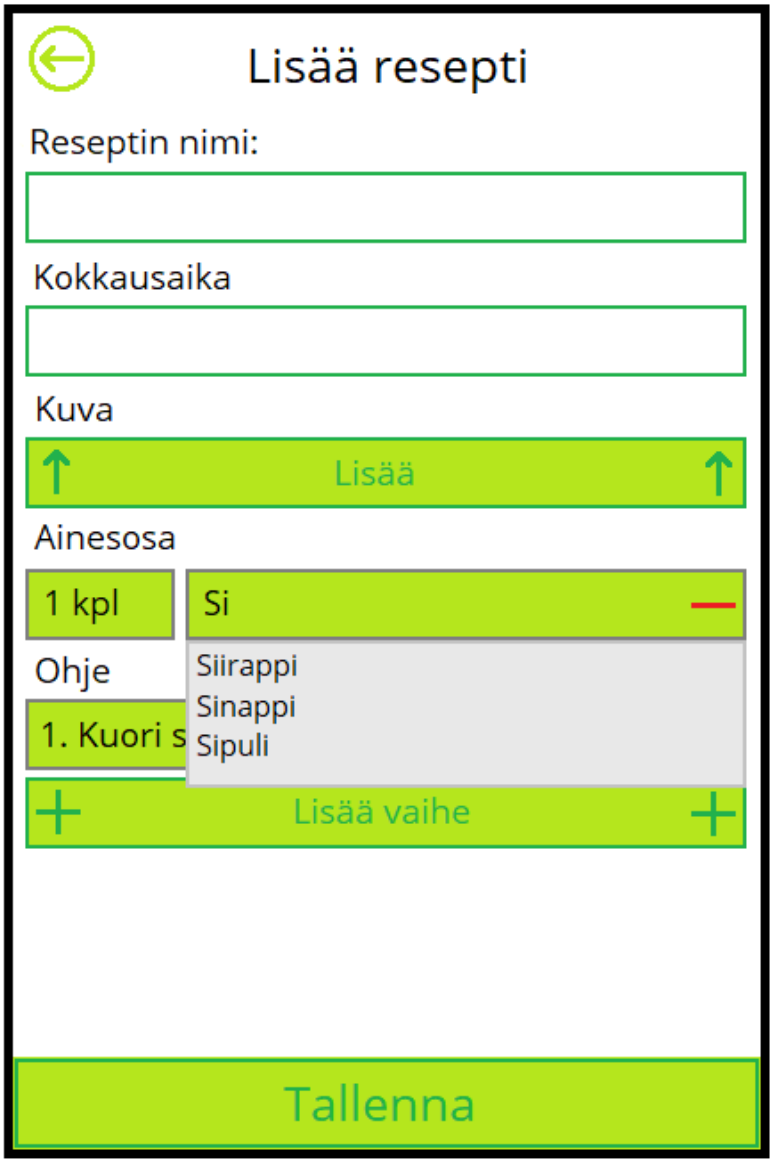
Kuviossa 9 esitellään sovelluksen sisäinen navigointikartta. Nuolilla on havainnollistettu, miltä sivuilta on siirtymä minnekkin. Punaisella on havainnollistettu siirtymät, joita vain ylläpitäjä voi tehdä. Punamustalla siirtymät, joissa ylläpitäjällä on tavallista käyttäjää enemmän oikeuksia tai toiminnallisuuksia.

Sovelluksella on siis seuraavat vaatimukset:

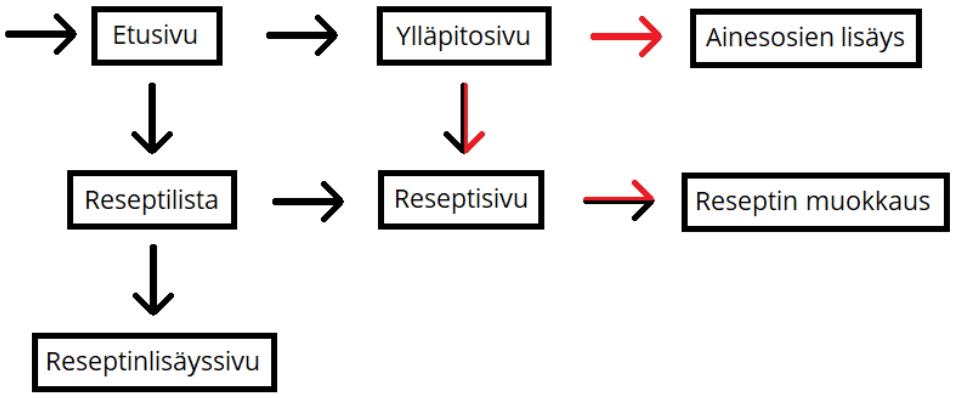
- lista, tietokanta tai muu tietolähde

Kuvio 7. Ylläpitosivulla pääkäyttäjät voivat muokata ja poistaa reseptejä, sekä lisätä raaka-aineita ja ainesosia järjestelmään.

- skaalautuvia komponenttisäilöjä, joilla toteuttaa reseptisivu
- galleria, tai vastaava keino esittää listan tai tietokannan sisältöä
- keino hallita näkyvyyttä eri käyttäjäryhmille (admin/peruskäyttäjä)
- reaaliaikainen haku sanojen ehdottamiseen ainesosien lisäämisessä.



Kuvio 8. Reseptin lisäyssivulla ainesosia lisätessä ehdotetaan sanoja, jotka vastaavat laatikkoon kirjoitettua sisältöä.



Kuvio 9. Sovelluksen navigointikartta. Nuolilla on esitetty siirtymät sovelluksen eri sivuille. Mustat nuolet ovat kaikille käyttäjille mahdollisia siirtymiä. Punaisella nuolella on merkitty siirtymät, joita vain ylläpitäjä voi tehdä. Punamustalla nuolella on havainnollistettu sivut, joilla ylläpitäjällä on lisäoikeuksia.

4 Tutkimustulokset ja analyysi

Jokaisen alustan kehityksen parissa työskenneltiin noin 30 tuntia, eli neljän täyden työpäivän verran. Kehitykseen sisältyi alustalle rekisteröityminen, lataaminen, asentaminen ja kaikki muu käyttöönottoon liittyvät vaiheet. Kehityskertoja oli keskimäärin kuusi kappaletta per alusta. Kehittäminen toteutettiin seuraavilla aikaväleillä:

- Visual LANSA: 20.4.2023 – 15.5.2023
- OutSystems: 17.5.2023 – 6.6.2023
- Mendix: 31.5.2023 – 21.6.2023

OutSystemsin ja Mendixin kehityksessä syntyi hieman päällekkäisyyttä, sillä OutSystemsillä pidettiin Mendix kehittämisen alun jälkeen vielä yksi kehityskerta, että päästiin lähemmäs samaa kehitysaikaa kuin Visual LANSA kehitykseen käytettiin. Tutkimusvaihe toteutettiin lähes tarkalleen kahden kuukauden aikana. Suunnitelman mukaisesti kehityskertojen jälkeen kirjattiin tehdyt havainnot ja tuntemukset erilliseen dokumenttiin. Joitakin kehityskertoja videoitiin kokonaan tai osittain, mutta videointi osoittautui nopeasti epäinformatiiviseksi, sillä suuri osa kehitysajasta meni tiedon hakemiseen ja dokumentoinnin selaamiseen.

4.1 Visual LANSA

Ensimmäinen vertailuun valittu alusta on Visual LANSA hybridi low-code -ympäristö. Visual LANSAn mainitaan olevan hybridialusta, sillä käyttäjä pääsee muokkaamaan low-code kontrolleja ja niiden toimintaa ohjelmoimalla. Merkittäviä Visual LANSAn kumppaneita ovat: Honda[1], Visa[2], Kellogg's[3] ja Porsche[4]. Visual LANSA on alusta, jossa käyttäjä voi kehittää työpöytä-, web- ja mobiilisovelluksia, jotka voidaan julkaista useisiin erilaisiin pilvipalveluihin. LANSA (2023a) verkkosivuilla

1. https://lansa.com/casestudies/honda/
2. https://lansa.com/casestudies/visa/
3. https://lansa.com/casestudies/kelloggs/
4. https://lansa.com/casestudies/porsche/

kerrotaan, kuinka sovelluskehitys- ja integraatioalusta julkaistiin ensimmäisen kerran vuonna 1987 varastopohjaisena RAD, eli *rapid application development* toteutuksena. Visual LANSAlla on siis yli kolmenkymmenen vuoden historia, mikä tekee siitä poikkeuksellisen low-code -alustojen piirissä. Vuonna 2018 Visual LANSA myytiin Houstonilaiselle Idera yhtiölle.

4.1.1 Käyttökokemus ja perehdytysmateriaali

Visual LANSA tarjoaa aloittamiseen useita valmiita sivustopohjia, joiden ympärille rakentaa oma web- tai mobiilisovellus. Alustan käytön perehdytys ja tutoriaali on myöskin esitetty tietyn valmiin sivustopohjavaihtoehdon perusteella. Visual LANSAn perehdytysmateriaali löytyy erilliseltä Learn LANSA -sivustolta[5]. Perehdytysmateriaali koostuu erilaisista kursseista, jotka sisältävät sekä videoita, että tekstimuotoista materiaalia. Käyttäjä tarvitsee voimassaolevan tunnuksen, jotta kurssien materiaaleihin pääsee käsiksi, mutta kurssien sisällysluetteloa voi käydä selaamassa ilman tunnusta. Perehdytyskurssit ja tiettyihin aiheisiin sitoutuvat kurssit ovat laadultaan erittäin vaihtelevia. Tutoriaalikurssi LANSAn käyttöön antaa hyvän perusymmärryksen työkalujen käytöstä ja toiminnasta, mutta ei johdata paljon perusteita pidemmälle. Joistakin aiheista, kuten esimerkiksi datan tallentaminen listoihin tai tauluihin, löytyi todella heikosti opetusmateriaalia. Learn LANSAn lisäksi alustasta löytyy myös perin kattava dokumentaatio[6], joka on tosin jaoteltu hieman kummallisen laajoihin kokonaisuuksiin tiettyjen ominaisuuksien tai funktioiden sijaan.

Visual LANSAn kehittämisessä käytetään alustaa nimeltään LANSA editor, sekä alustan omaa ohjelmointikieltä nimeltään RDML, eli *Rapid Development and Maintenance Language*. Bettin (2001) luokittelee RDML:n neljännen sukupolven ohjelmointikieleksi. Kun LANSAssa lisätään komponentti ruudulle suunnittelutilassa, lähdekoodiin generoituu komponentille aloitus- ja lopetusrivit, joiden väliin käyttäjä voi lisätä halutessa toiminnallisuutta. LANSAssa pääsee perin pitkälle käyttöliittymän kehittämisessä ilman ohjelmointia, mutta lähes kaiken toiminnallisuuden kehittä-

5. https://learn.lansa.com/
6. https://docs.lansa.com/14/en/

minen vaatii lähdekoodin muokkaamista. Reseptisovelluksen kehittämisestä vain hyvin pieni osa koostui käyttöliittymän muokkaamisesta ja valtaosa kehitysajasta meni tietolähteiden parissa toimimiseen ja lähdekoodin muokkaamiseen. Visual LANSAn käyttöliittymä eroaa merkittävästi muista käytetyistä low-code alustoista, johtuen mahdollisesti alustan iästä ja alustan ominaisuuksiin tottuneesta käyttäjäkunnasta. LANSA Editorin käynnistyksessä valitaan käytetty *partitio*, eli osio, joka voi sisältää useita sovelluksia ja on oma muista partitioista eristetty kokonaisuutensa. *Partitio* voi sisältää kustomoituja komponentteja, tietovarastoja ja muuta haluttua materiaalia. Tietovaraston valinnan yhteydessä valitaan kehitettävä sovellus. LANSA Editorin vasemmassa laidassa on ikkuna, josta löytyvät valinnat: repository, controls, details, sekä outline. Repository välilehdessä hallinnoidaan partition erilaisia resursseja, kuten tietokantaa ja tauluja, verkkosivuja, prosesseja, funktioita, lomakkeita ja useita muita sovellusten osia. Controls välilehdessä sijaitsevat kaikki mahdolliset sovelluksessa käytettävät kontrollit ja komponentit. Details välilehdellä esitetään yksittäisten kontrollien, komponenttien ja objektien yksityiskohtaiset ominaisuudet ja asetukset. Outline välilehdellä puolestaan esitetään editorissa auki olevat sovellukset, sivut, näkymät ja muut kokonaisuudet, sekä näiden rakenne sisäkkäisinä pudotusvalikkoina. LANSAssa käyttäjä pystyy koska vain vaihtamaan suunnittelutilan ja lähdekoodin välillä pääikkunan design ja source välilehtiä vaihtamalla. Suunnittelumoodissa kehittäjä pystyy muokkaamaan suoraan joitakin komponentin ominaisuuksia, kuten tekstilaatikossa näkyvää tekstiä, mutta jotkin monimutkaisemmat operaatiot vaativat RDML ymmärrystä ja muokkausta lähdekoodin puolella.

LANSA Editoria ajetaan ainakin koeaikana virtuaalikoneella selaimen välityksellä Cameyo virtualisointialustalla. Virtuaalikonetta käytettäessä on havaittavissa pientä viivettä kehittäjän syötteen ja virtuaalikoneen toiminnan välillä, mikä tekee kehittämisestä hieman epämiellyttävää. Sovelluksen julkaiseminen ja selaimessa ajaminen on nopeaa ja vaivatonta. Julkaistu sovellus toimii moitteettomasti ja hyvin skaalautuvasti useissa selaimissa.

4.1.2 Lisenssointi ja hinnoittelu

Visual LANSAsta on tarjolla 30 päivän ilmainen koeaika. Visual LANSAn verkkosivuilla ei ole suoraa tietoa hinnoitelusta, eikä hinnoittelua mainittu myöskään ilmaisen kokeiluajan aikana. Visual LANSAn verkkosivuilla kehotetaan olemaan yhteydessä asiakaspalveluun hinnoittelun selvittämiseksi.

4.2 OutSystems

Toinen vertailuun valittu alusta on nimeltään OutSystems. Merkittäviä OutSystemsin kumppaneita ovat: Burton[7], Logitech[8], Merdcedes-Benz[9] ja Mazda[10]. OutSystems (2023b) verkkosivuilla kerrotaan, kuinka OutSystems perustettiin vuonna 2001 vastaamaan perinteisen sovelluskehityksen, sekä erityisesti vesiputousmallisen kehityksen, hitaaseen ja kankeaan elinkaareen. OutSystemsin tavoitteet nykypäivänä on kehittää ratkaisuja, jotka mahdollistavat liiketoiminnan digitalisaation ja nopean kehityksen, toimituksen ja ylläpidon. OutSystems on sekä tuotteen, että tuotetta kehittävän Bostonilaislähtöisen yrityksen nimi.

4.2.1 Käyttökokemus ja perehdytysmateriaali

OutSystemsin käytön aloittaminen on erittäin helppoa, vaivatonta ja nopeaa. Rekisteröimisprosessissa pyydetään henkilön sähköposti, etunimi ja sukunimi, organisaatio johon hän kuuluu, sekä salasana. Käyttäjän luomisen jälkeen pääsee suoraan tutoriaaliin, jossa esitetään vaihe vaiheelta kehittämiseen käytettävän Service Studion asentaminen ja ensimmäisen sovelluksen aloittaminen. OutSystemsin sovelluksien analytiikka, tietoturva ja hallinnointi hoidetaan erillisessä selaimen kautta käytettävässä LifeTime sovelluksessa. Forge on puolestaan ohjelmakoodivarasto, josta käyttäjät voivat ladata valmiita uudelleenkäytettäviä moduuleita, liittimiä, sekä käyttöliittymäkomponentteja käytettäväksi sovelluksissa. Forgen kautta pystyy

7. https://www.outsystems.com/case-studies/burton-logistics-solution/
8. https://www.outsystems.com/case-studies/fast-development/
9. https://www.outsystems.com/case-studies/mercedes-benz-complaints-management-portal/
10. https://www.outsystems.com/case-studies/mazda-legacy-migration-to-save-millions/

lataamaan esimerkiksi Microsoft Excel integroinnin mahdollistavan liittimen, sekä PDF-generoinnin mahdollistavan moduulin. OutSystemsillä on erinomainen perehdytysmateriaali[11]. Koulutus välilehdestä löytyy opastettuja kokonaisuuksia, kursseja, opastus livekoulutuksiin, sertifikaatteja ja paljon muuta. Kotona opiskeltavaksi löytyy useita kursseja, jotka sisältävät ohjevideoita, joissa näytetään käytännössä miten jokin ominaisuus tai toiminto toteutetaan. OutSystemsillä on myös hyvin kattava dokumentaatio[12], joka koostuu tutoriaalinomaisista kokonaisuuksista tiettyjen operaatioiden sijaan. Sivustolta löytyy myös keskustelupalsta, missä voi keskustella kaikesta alustaan liittyvästä.

Kehittämisen OutSystemsissä voi aloittaa joko tyhjältä pohjalta, tai valitsemalla yhden monista sapluunvaihtoehdoista. Sovellukseen pystyy luomaan tietokannan rakenteen helposti Excel-tiedoston avulla, jossa on määritetty sarakkeiden nimet ja vapaavalintaisesti myös haluttua tietokantaan tallennettavaa sisältöä. Alusta osaa generoida tiedostosta halutun kaltaiset tietokantataulun. Sovelluksen käyttöliittymän kehittäminen on erittäin helppoa. Uusia näkymiä, eli navigoinnissa esitettäviä näyttöjä, voi lisätä sovellukseen halutun määrän. Käyttäjä voi hiirellä raahata ruudun vasemmalla puolella sijaitsevasta työkalupakista komponentteja suoraan näihin näkymiin. Käyttöliittymän komponenttien sijoittaminen on vahvasti säiliöpainoitteista. Komponentit lisätään säiliöihin, joiden sisällä komponentin sijainnin voi määrittää. Säiliöt latoutuvat ruudulle päällekäin, eikä niitä voi sijoittaa vapaalla kädellä haluttuun paikkaan. Sovelluksen oikeassa laidassa sijaitsee ikkuna, josta löytyvät välilehdet: *processes*, *interface*, *logic*, sekä *data*. *Processes* välilehdellä voi luoda ja muokata työnkulkuja, joiden avulla voidaan tehdä esimerkiksi ajonaikaisia tietokantamuutoksia. *Interface* välilehdellä on kaksi alivälilehteä: *elements* ja *widget tree*. *Elements* valinnassa esitetään pudotusvalikkoina näkymät, sekä näkymissä sijaitsevat työnkulut, muuttujat, tietokantaprosessit, sekä muuta tärkeää toiminnallista tietoa. *Element* valinnassa navigoidaan myös eri näkymien välillä. *Widget tree* valinnassa esitetään aktiivisen näkymän kontrollit ja komponentit pudotusvalikkonäkymässä. Valikossa myöskin pystyy muuttamaan valitun kontrollin ominaisuuksia ja ul-

11. https://www.outsystems.com/training/
12. https://success.outsystems.com/documentation/

koasua. *Logic* välilehdellä hallitaan sovelluksen sisäistä logiikkaa, eli hallinnoidaan käyttäjärooleja, ulkoisia rajapintaliittimiä, sekä riippuvuuksia. *Data* välilehdellä hallinnoidaan sovelluksen dataa, kuten tietokannan tauluja, rakennetta ja prosesseja. Service Studiosta löytyy valmiina joitakin integraatioita, kuten esimerkiksi SOAP, REST ja SAP. Alustalla voi myös määritellä käyttäjäryhmiä, sekä antaa käyttäjäryhmille tiettyjä oikeuksia sovelluksen sisällä. Tietokannan sisältöä saa näkyville esimerkiksi lisäämällä *taulu* tyypin komponentin ruudulle ja valitsemalla aukeavasta valikosta *lista*. Painikkeille voi luoda toiminnallisuuksia, kuten klikkauksen operaatiot, erilaisten työnkulkujen avulla. Työnkulussa eri toiminnallisuuksia vedetään työkalupakista ruudulle *alku* ja *loppu* vaiheiden välille, kuten sovelluksen käyttöliittymässä. Toteutetussa sovelluksessa luotiin esimerkiksi uuden reseptin luomiselle oma työnkulku. Painiketta napsauttaessa luotu työnkulku alustaa tyhjän reseptin ja siirtää käyttäjän seuraavalle sivulle, missä syötetään uuden reseptin tiedot lomakkeen avulla. Tietoa ruudulta toiselle voi viedä joko suoraan muuttujien välityksellä, tai käyttämällä työnkulkuja. Muuttuja alustetaan lähdesivulla ja kohdesivulla pitää olla muttujan vastaanottava *input* -tyypin muuttuja. Reseptisovelluksessa toteutettiin esimerkiksi muuttujalla reseptilistasta tietyn reseptin valinta kyseisen reseptin sivulle, missä *GetReseptit* -operaatiolla haettiin reseptin tiedot. OutSystems vaatii tietynlaisen komponentin tietylle toiminnolle. Esimerkiksi uuden reseptin tallennusta ei saatu toimimaan muulla tavalla, kuin lisäämällä lomake-elementti sivustolle. Alustan oma lomake-elementti lisäsi samalla automaattisesti tallennuspainikkeen, sekä tauluista löytyville kentille tekstinsyöttöelementit sivulle. Tämän tallennuspainikkeen toiminnallisuutta ei voinut muokata, vaan se tallensi automaattisesti lomake-elementin sisällön linkitettyyn tietokantatauluun.

4.2.2 Lisenssointi ja hinnoittelu

OutSystemsin hinnoittelu on avoin ja löytyy verkkosivuilta[13]. OutSystemsistä on tarjolla ilmainen versio, joka mahdollistaa yhden sovelluksen kehittämisen rajatuilla ominaisuuksilla. Ilmaiskäyttäjälle ei ole tarjolla OutSystemsin tarjoamaa tukea,

13. https://www.outsystems.com/pricing-and-editions/

eikä sovellusta pysty julkaisemaan tai ajamaan tuotantoon. Monen sovelluksen samanaikaiseen kehittämiseen ja julkaisemiseen tarvitaan keskitason lisenssi, jonka hinta alkaa 1250 eurosta. Lisenssin omistajalle tarjotaan myös täysi tuki arkipäivisin kello 8 ja 17 välillä. OutSystems lupaa lisenssöidyille käyttäjille 99,5% käytettävyysajan ja loputtoman käyttäjämäärän sovelluksille. Kolmas lisenssityyppi on suurille organisaatioille tarjottava lisenssi, jonka hinta täytyy tiedustella OutSystemsin asiakaspalvelusta. Lisenssin mukana tarjotaan 24/7 täysi tuki, ylimääräisiä testiympäristöjä, mahdollisuus omilla palvelimilla lokaaliin kehittämiseen ja ylläpitoon, sekä ylimääräisiä tietoturva- ja monitorointiratkaisuja.

4.3 Mendix

Kolmas valittu alusta on Mendix. Merkittäviä Menxidin kumppaneita ovat: Sanoma[14], ISS[15], Mitsubishi[16] ja Continental[17]. Mendix on vuonna 2005 perustettu Hollantilainen yritys, jonka low-code -ohjelmointialustan ensimmäinen yleisölle tarkoitettu versio julkaistiin vuonna 2014. Mendix (2023b) kerrotaan, kuinka vuonna 2005 perustettu yritys otti nimensä englanninkielen sanasta *mend*, joka tarkoittaa parantamista tai korjaamista. Mendix perustettiin, koska perustajat Roald Kruit, Derek Roos ja Derckjan Kruit halusivat korjata kehittäjien ja liiketoiminnan välisen kuilun hyödyntämällä malliperustaista ohjelmistokehitystä. Mendix haluttiin kuitenkin selkeästi erottaa neljännen sukupolven ohjelmointikielistä, sillä se on perustajien mukaan huomisen perintökoodia. Mendixiin haluttiin korkeampi malliperustainen abstraktiotaso, kuin perinteisessä ohjelmoinnissa. Tämä malli haluttiin sitten muuttaa automaation avulla toimivaksi sovelluslogiikaksi. Mendixissä haluttiin välttää kaikenlainen käyttäjälle näkyvän ohjelmakoodin generointi ja keskittyä suoraan esitettävään toiminnallisuuteen, jolloin vaatimusmäärittelyn toteuttaminen myöskin helpottuu. Siemens osti Mendixin vuonna 2018.

14. https://www.mendix.com/customer-stories/sanoma-media/
15. https://www.mendix.com/customer-stories/iss-facility-services/
16. https://www.mendix.com/blog/mendix-fsm-mitsubishi-elevator-europe/
17. https://www.mendix.com/customer-stories/continental/

4.3.1 Käyttökokemus ja perehdytysmateriaali

Mendix tunnusten luominen on erittäin helppoa ja nopeaa. Käyttäjän luomiseen tarvitsee antaa oma etu- ja sukunimi, yrityksen tai organisaation sähköposti, salasana, sekä asuinalue. Mendixillä kehittämisen aloittaminen on ilmaista, mutta julkaiseminen ja tuotantoon vieminen vaatii maksullisen lisenssin. Kehittäminen suoritetaan Mendix Studio Pro IDE -sovelluksella. Mendixin dokumentaation[18] selaaminen on hankalaa, sillä se koostuu useista kerroksista sisäkkäisiä pudotusvalikoita, joiden seuraaminen ei ole aina täysin johdonmukaista. Dokumentaatio on jaettu toiminnallisiin kokonaisuuksiin yksittäisten operaatioiden sijaan, minkä seurauksena tietyn operaation tai funktion löytäminen on vaikeaa. Mendixillä on tarjolla myös *akatemia*[19], josta löytyy ilmoittautuminen luennoitsijoiden pitämiin koulutuksiin, sekä kokonaisuus erilaisia sertifikaattikokeita. *Akatemian* kautta löytyy myös itseopiskelumateriaalina luentoja, jotka koostuvat myös demotehtäviä sisältävistä moduuleista.

Kehittäminen Mendixillä on hyvin samankaltaista, kuin OutSystemsillä. Ruudun oikeasta sivusta löytyy työkalupakki, josta voi siirtää komponentteja suoraan ruudulle haluttuun paikkaan hiirellä vetämällä. Käyttöliittymän komponenttien sijoittelu perustuu pitkälti säiliöihin, joiden koko määritettään mendixin omalla mittakaavalla. Ruutu on leveyssunnassa jaettu kahteentoista osioon ja komponenttien leveys määritellään antamalla sille arvo yhden ja kahdentoista välillä. Menetelmä on perin kankea, mutta pitää huolen siitä, että sovellus skaalautuu eri kokoisille näytöille ja laitteille. Vasemmassa laidassa Studio Pro IDE:stä löytyy sovelluksen rakenteen navigointi. Navigointi koostuu projektin hallinnoinnin asetuksista, järjestelmämoduulista, sekä käyttäjän haluamasta määrästä muita moduuleja. Ylimpänä on koko projektin hallinnoinnin asetukset, missä käyttäjä voi hallita esimerkiksi ulkoisesti ladattuja komponentteja, tietoturvan asetuksia, sekä käyttäjärooleja. Toisena valikosta löytyy järjestelmämoduuli, missä käyttäjä voi nähdä ja muokata sovelluksen automaattisesti generoimia materiaaleja ja toiminnallisuuksia. Käyttäjä voi itse lisätä

18. https://docs.mendix.com/
19. https://academy.mendix.com/link/home

sovellukseen haluamansa määrän moduuleja, joiden sisään luodaan muun muassa yhteydet tietovarastoihin, lisätään ulkoiset tiedostot kuten kuvat ja luodaan halutut näkymät ja työnkulut. Tietokanta luodaan sovelluksen sisällä helposti luomalla ER, eli *entity-relationship*, kaavio. Jokaisesta entiteetistä muodostuu oma taulunsa ja entiteettien väliset suhteet määräävät tietokannan väliset suhteet ja toiminnallisuuden sovelluksessa. Sovelluksen kautta voi lisätä tietokantaan dataa lisäämällä *fieldset* nimisen säiliökontrollin, jonka sisälle voi määrittää tietokantaan kirjoitettavat kentät. Painikkeiden toiminnallisuus toteutetaan luomalla työnkulku, joita kutsutaan Mendixissä nano- ja microfloweiksi. Työnkuluilla on alku- ja loppuvaiheet, joiden väliin asetetaan käyttöliittymän komponenttien tapaan erilaisia toiminnallisuuksia mahdollistavia osia. Reseptisovelluksessa kehitettiin työnkululla painikkeen toiminnallisuus, jonka avulla luodaan tietokantaan uusi resepti. Microflow toteutetaan raahaamalla entiteetin alustava operaatio työnkulkuun ja laittamalla uuden entiteetin tyypiksi *no commit*, eli entiteetti ei ole sitoutunut mihinkään olemassaolevaan tietoon, vaan pelkästään pohjustaa uuden reseptin. Reseptien lisäyssivulle lisättiin form tyyppinen komponentti, joka pystyi lisäämään suoraan tietokannan rakenteeseen perustuen halutut kentät lomakkeelle. Lomakkeen lisäyksen yhteydessä sovellus luo suoraan tallennuspainikkeen lomakkeen pohjalle. Mendixillä osoittautui hankalaksi luoda sivulle lomake, joka kirjoittaa yhdellä tallennuksella kolmeen eri tauluun sisältöä. Toteutetussa sovelluksessa saatiin ainesosat ja ohjeet näkymään lomakkeelle lisäämällä tyhjien ainesosien ja ohjeiden alustaminen microflowiin. Tallennus saatiin lopulta luomaan kaikkiin kolmeen tauluun sisältöä, mutta rivien väliset suhteet eivät toimineet oikein, minkä seurauksena ainesosien ja ohjeiden rajaus ei toiminut halutulla tavalla. Sovelluksessa onnistuttiin myös saamaan reseptilistasta navigointi ja tiedonsiirto yksittäisen reseptin omalle sivulle.

4.3.2 Lisenssointi ja hinnoittelu

Mendixin hinnoittelu on avoin ja löytyy verkkosivuilta[20]. Mendixiä voi käyttää kehittämiseen täysin ilmaiseksi hyvin pienillä rajoituksilla toteutukseen. Ilmaisver-

20. https://www.mendix.com/pricing/

siolla julkaistua sovellusta pystyy tosin ajamaan vain kaksi tuntia kerrallaan, joten toimivaa sovellusta ei pysty ilmaiseksi ylläpitämään. Ilmaislisenssin lisäksi on saatavilla basic, standard, sekä premium tason lisenssit. Basic tason lisenssin hinnoittelu alkaa 50 eurosta kuukaudessa ja se on tarkoitettu yksinkertaisten, pienille ryhmille tarkoitettujen sovellusten toteuttamiseen. Standard lisenssin hinnoittelu alkaa 800 eurosta kuukaudessa ja se on tarkoitettu yritystason käyttöön. Neljäs vaihtoehto on premium lisenssi, joka on tarkoitettu kriittisten järjestelmien kehittämiseen. Mendixin sivuilla ei mainita premium lisenssin hinnoittelun hintahaitaria.

5 Yhteenveto

Low-codella on pitkä teoreettinen tausta, mutta vasta viime vuosikymmenen aikana markkinoille on alkanut nousemaan erityisesti low-code -kehitykseen tarkoitettuja kehitysalustoja. Suurimmat markkinajohtajat kehitysalustoissa ovat Microsoftin Power Platform, OutSystems, SalesForce, Appian ja Mendix. Microsoftin Power Platformia ei tässä työssä vertailtu, sillä tekijä työskentelee kyseisen alustan parissa, mikä olisi vääristänyt tutkimuksen tuloksia. Power Platformin ja muiden nykyisten markkinajohtajien sijaan vertailuun valittiin Visual LANSA, sillä se oli muita alustoja huomattavasti vanhempi ja laajasti käytetty alusta ennen useiden muiden kilpailijoiden ilmaantumista.

Tässä pro gradu -tutkielmassa kehitettiin kolmella eri low-code -sovelluskehitysalustalla mahdollisimman samankaltainen, ennalta suunniteltu sovellus. Kaikilla kolmella alustalla kehittäminen jäi kesken, eikä yhdelläkään saavutettu suunnitelmien mukaista valmista sovellusta. Sovellusten kesken jääminen oli odotettavissa, sillä lisenssointi ja aikataulutus vaikuttivat käytettävissä olevaan kehitysaikaan. Yksikään alustoista ei siis ollut niin intuitiivinen ja helppokäyttöinen, että aiemmalla low-code -kehityskokemuksella olisi ollut merkittävää hyötyä alustan käytössä. Myöskin tutkimusasetelma, jossa alustan toimintaan ja käyttöön perehdytään vasta kehitystä aloittaessa, johti keskeneräisiin sovelluksiin. Kehittämisessä keskityttiin ensisijaisesti päätoiminnallisuuksien kehittämiseen, eli datan tallentamiseen, tiedon siirtämiseen sivulta toiselle, sekä tiedon esittämiseen halutulla tavalla. Käyttöliittymän kehittäminen tuli kaikilla alustoilla tutuksi toiminnallisuuksia laatiessa, sillä kaikki käytetyt low-code -alustat seurasivat malliperustaisen ohjelmistokehityksen periaatteita.

5.1 Johtopäätökset

Tutkielmassa vertailtiin kolmea eri low-code -sovelluskehitysalustaa, jotka olivat: Visual LANSA, OutSystems ja Mendix. Tavoitteena oli löytää alustojen käytöstä,

opetus- ja perehdytysmateriaaleista, sekä lisenssoinnista ja hinnoittelusta eroavaisuuksia ja samankaltaisuuksia. Alustojen käyttökokemuksissa oli selkeitä samankaltaisuuksia, mutta myöskin merkittäviä eroja. Kaikkien käytettyjen alustojen kehitys toteutettiin low-code kehitykselle tyypillisellä tavalla käyttöliittymä edellä. Mendix ja OutSystems kehittäminen toteutettiin tietokoneelle ladattavalla ohjelmointiympäristösovelluksella. Visual LANSAn LANSA editor ohjelmointiympäristöä ajettiin selaimella Cameyo virtualisointialustalla.

Kuviossa 10 esitellään tutkimuksen aikana eri alustoilla toteutetut ja toteuttamatta jääneet toiminnallisuudet. Kaikilla alustoilla aloitettiin työskentely tyhjältä ruudulta, eli millään alustalla ei valittu tarjottuja mallipohjia. Kaikilla käytetyillä alustoilla oli lomaketyyppinen elementti, joka mahdollisti tiedon tallentamisen tietolähteeseen helposti ja vaivattomasti. Visual LANSAssa kyseistä elementtiä ei voinut jostakin syystä lisätä websovellukseen, joten kuvion 10 listassa sitä ei ole merkitty toteutetuksi, vaikka itse lomake-elementti toteutettiinkin. Visual LANSA oli hankala kehitysalusta työskennellä, sillä edellämainitun kaltaisia yhteensopivuusongelmia esiintyi muissakin käytetyissä komponenteissa. Visual LANSAn perehdytysmateriaalissa oletettiin mallipohjan käyttö, joten materiaali ei ollut kovinkaan hyödyllinen useassa tilanteessa.

Kuviossa 10 tallennetun tiedon esittämisellä tarkoitetaan reseptit sivulla reseptilistan esittämistä, mikä vaihteli suuresti eri alustojen välillä. Visual LANSAssa oli useita animoituja listakontrolleja, joita kutsuttiin LANSAssa karuselleiksi. LANSAssa tiedon esittäminen ei onnistunut, sillä tietolähteen dataa ei saatu sovelluksessa linkitettyä karuselleihin. Tiedon lisääminen karuselliin olisi vaatinut todennäköisesti lähdekoodin muokkaamista RDML:n[1] avulla, mutta dokumentaatiosta ei löydetty tähän selkeitä ohjeita. OutSystemsissä ja Mendixissä oli kummassakin mahdollisuus esittää tietoa eri suuntiin rullattavilla listoilla, joiden sisältämän tiedon määrää pystyi muuttamaan tarpeen mukaan. OutSystemsissä ja Mendixissä onnistuttiin esittämään dataa sovelluksessa, sillä tietolähteen linkittäminen onnistui helposti listakontrollin ominaisuuksien kautta.

1. RDML, eli *Rapid Development and Maintenance Language*. Visual LANSAn ohjelmointikieli.

Kuviossa 10 useiden taulujen esittäminen samassa näkymässä tarkoittaa reseptisivua, jolla tuli esittää reseptin nimen ja kokkausajan lisäksi myös erillisiin tauluihin tallennetut ainesosat ja ohjeet. Visual LANSAssa ei onnistuttu esittämään useiden taulujen dataa samassa näkymässä, sillä jo yksittäisen taulun datan esittäminen epäonnistui. OutSystemsissä onnistuttiin esittämään reseptin tiedot, sekä rajaamaan ohjeet ja ainesosat halutulla tavalla. Reseptisivulla valitun reseptin *ID:n* avulla pystyi helposti noutamaan tietokannasta kahteen erilliseen listakontrolliin ainesosien ja ohjeiden kentät. Mendixissä onnistuttiin reseptisivulla esittämään reseptin nimi ja valmistusaika, mutta ainesosien ja ohjeiden rajaamista koskemaan tiettyä valittua reseptiä ei saatu toimimaan, eli sivulla näytettiin kaikkien reseptien ohjeet ja ainesosat. Rajauksen epäonnistuminen johtui varmaankin lomakkeen toteutuksesta, sillä tietokannan tauluja vertaillessa huomattiin, että taulujen väliset tunnisteet eivät toimineet oikein. Taulujen välisiä suhteita pystyi muokkaamaan ER-kaavion tapaan esitetyssä näkymässä. Näiden assosiaatioiden käyttäminen sovelluksessa vaikutti alkuun helpolta, mutta ei johtanut toivottuun lopputulokseen. Myöhemmin koitettiin vielä muuttaa tunnisteet oikeiksi suoraan tauluissa, mutta tämä rikkoi koko sovelluksen tietokannan, jolloin sovellus ei enää suostunut kääntymään tai ajamaan.

Visual LANSAssa ei onnistuttu siirtämään tietoa ruudulta toiselle. Muuttujien tallentaminen oli helppoa, mutta toiminnallisuutta muuttujien viemisestä ruudulta toiselle ei löytynyt. LANSAn näkymät vaikuttivat olevan huomattavasti eristetympiä kokonaisuuksia, kuin muissa käytetyissä alustoissa. OutSystemsissä tieto ruudulta toiselle vietiin asettamalla kohdesivulle *input* -tyyppinen muuttuja, joka vastaanotti esimerkiksi reseptin tapauksessa reseptin yksilöllisen tunnisteen, eli *ID:n*. Reseptin tunnisteella päästiin hakemaan ohjeet ja ainesosat kyseiselle reseptille. Mendixissä reseptin tietojen vieminen sivulta toiselle oli hieman monimutkaisempaa, sillä siirrettävä tieto piti liikuttaa työnkulkujen välityksellä. Työnkulkuja oli käytettävissä kahta laatua; *nanoflow*, eli erittäin lyhyt työnkulku, sekä *microflow*, joka voi sisältää hieman laajempaa toiminnallisuutta. Aluksi yritettiin toteuttaa *microflow*, joka ottaisi reseptit -sivulta valitun reseptin tunnisteen ja tätä käyttäen hakisi työnkulun ajon aikana reseptin, ohjeet ja ainesosat yhdellä kerralla. Ikävä kyllä kohdepäässä sovellus ei tunnistanut muuta kuin pelkästään reseptin tiedot, eli ohjeita ja aineso-

sia ei onnistuttu noutamaan halutulla tunnisteella. *Reseptit* -sivulta saatiin siis reseptin tiedot siirrettyä sivulta toiselle, mutta muiden taulujen sisällön noutamista ei saatu toteutettua.

	Tiedon tallentaminen käyttöliittymän kautta	Tiedon esittäminen sovelluksessa	Useiden taulujen esittäminen samassa näkymässä	Tiedon siirtäminen sivulta toiselle
Visual LANSA	Ei toteutettu	Ei toteutettu	Ei toteutettu	Ei toteutettu
OutSystems	Toteutettiin	Toteutettiin	Toteutettiin	Toteutettiin
Mendix	Toteutettiin	Toteutettiin	Toteutettiin, ilman rajausta	Toteutettiin

Kuvio 10. Alustoilla toteutetut ominaisuudet.

Johtuen rajoitetusta kehitysajasta, sekä kehityskertojen rajoitetusta lukumäärästä, useita toiminnallisuuksia jäi toteuttamatta kaikilla alustoilla. Visual LANSA:lla saatiin toteutettua pääosin vain käyttöliittymän kappaleita, sekä pieniä yksittäisiä eristettyjä osia toiminallisuudesta. Lista toiminnallisuuksista, joita ei onnistuttu tai ehditty toteuttaa millään alustalla:

- ennakoiva hakukenttä
- satunnaisen reseptin valinta
- reseptin muokkaus
- ostoskori
- ylläpitosivu ja sen toiminnallisuus.

Edellämainitut toiminnallisuudet jätettiin toteuttamatta pääosin siksi, että ne eivät tuottaneet merkittävää lisäarvoa kuviossa 10 esitettyihin perustoiminnallisuuksiin verrattuna.

5.2 Pohdinta

Aloittaessani tätä pro gradu -tutkielmaa, oli low-code -sovelluskehitys minulle lähes tuntematonta. Tunsin termin ja tiesin mistä low-code -sovelluskehityksessä on kyse, mutta en olisi osannut nimetä yhtäkään low-code -alustaa tai kertoa miten low-code -sovelluskehitystä toteutetaan. Nykyään työskentelen low-code -sovelluskehittäjänä Microsoftin Power Platform -kehitysalustan parissa keskisuuressa IT-alan yrityksen-

sä. Olen siis päässyt pro gradu -tutkielman kirjoittamisen aikana näkemään low-codea useasta eri näkökulmasta.

Tässä pro gradu -tutkielmassa verratuilla kolmella alustalla ei pystytä vielä muodostamaan selkeää linjausta sille, mikä tekee low-code -alustan, mutta pystytään rajaamaan niiden samankaltaisuuksia ja eroja. Kaikissa käytetyissä alustoissa käyttöliittymän kehittäminen oli yksinkertaista ja helposti toteutettavissa pienellä tietoteknisellä ymmärryksellä, eli kehittämisen alkuvaihe erityisesti oli nopeaa ja ketterää. Alustoista erityisesti OutSystems ja Mendix olivat nopeasti opittavissa ja helppokäyttöisiä. Visual LANSAlla kehittäminen puolestaan takkusi, mikä johtui LANSAn huomattavasti alemmasta abstraktiotasosta ja epäintuitiivisesta käyttöliittymästä. Tietolähteiden parissa työskentely, sekä monimutkaisemman toiminnallisuuden kehittäminen vaativat silti huomattavaa tietoteknistä osaamista kaikilla alustoilla, sekä ymmärrystä esimerkiksi tietokantojen rakenteesta ja taulujen välisistä relaatioista. Kaikissa käytetyissä alustoissa sovelluksen ajaminen, testaaminen ja tuotantoon vieminen oli tehty erittäin helpoksi. Mikään kehitysalustoista ei eronnut merkittävästi oman käyttöliittymänsä suhteen, vaan kaikkien rakenne oli erittäin perinteinen IDE:n, eli ohjelmointiympäristön käyttöliittymä. OutSystemsissä ohjelma julkaistiin ja ajettiin yhden painikkeen painalluksella. Mendixissä ohjelmaa pystyi ajamaan painamalla yhtä painiketta ja julkaisemalla painamalla toista. Visual LANSAssa pakattiin sovellus yhdellä toiminnolla ja ajettiin tämän jälkeen nappia painamalla. Kaikissa käytetyissä alustoissa oli sisäänrakennettu versionhallinta, joten käyttäjän ei tarvitse itse huolehtia versioinnista ja varmuuskopioista. Vaikka alustat eroavat selkeästi toisistaan toiminnallisuuksiltaan, löytyy niistä silti huomattava määrä samankaltaisuuksia, jotka saattavat päteä myös muille markkinoilla saatavilla oleville low-code -kehitysalustoille. Selkeimmät samankaltaisuudet olivat:

- käyttöliittymän kehittämisen nopeus *vedä ja pudota* käyttöliittymän avulla
- mahdollisuus yhdistää sovellus useisiin erilaisiin alustan sisäisiin ja ulkoisiin tietolähteisiin
- mahdollisuus yhdistää sovellus ulkoisiin palveluihin
- toteutetun sovelluslogiikan sidonnaisuus kehitysalustaan.

Huomattavimmat erot olivat puolestaan:

- syntaksi, jolla sovelluslogiikkaa toteutetaan
- dokumentaation taso ja alustan intuitiivisuus
- hinnoittelu, sillä jotkin alustat eivät hinnoittelunsa perusteella tarjoa palveluitaan yksityishenkilöille.

Yllättävin ero alustojen välillä oli vaihtelu dokumentaation tasossa ja alustan intuitiivisuudessa. Visual LANSA -sovelluskehitys tuntui huomattavasti hitaammalta ja hankalammalta kuin OutSystemsin ja Mendixin käyttö. Visual LANSAn kattava dokumentaatio helpotti kehitystä huomattavasti, mutta tutoriaalin ja perehdytyksen puute aiheutti hankaluuksia kehityksen alkuvaiheessa. Myöskin hinnoittelu erosi selkeästi alustojen välillä. Visual LANSA ei ilmoittanut ollenkaan hinnoitteluaan, OutSystemsin halvin lisenssi oli tuhannesta eurosta ylöspäin ja Mendix puolestaan tarjosi yksityishenkilölle sopivaa lisenssiä noin 50 euron hinnalla. Bock ja Frank (2021) listaavat low-code -sovelluskehitysalustojen peruspiirteitä seuraavasti:

- low-code alustat integroivat erilliset klassiset kehityskomponentit yhteen ympäristöön
- tuottavuus saavutetaan pääosin vähentämällä rutiininomaisia tehtäviä
- ei uutta teknologiaa, vaan yhdistelmä useita olemassaolevia teknologioita
- konseptuaalinen mallinnus ei ole markkinoinnin ytimessä, vaan alustan ytimessä
- tiettyyn kehitysalustaan jumiin jäämisen uhka.

Aikarajoitteet, sekä tutkimuksen yksin toteuttaminen rajasivat tutkimuksen keskittymään vain kolmeen alustaan. Low-code -sovelluskehitysalustat vaikuttavat olevan täällä jäädäkseen, joten samankaltainen, mutta laajempi tutkimus olisi tarpeellista. Low-code -alustojen tutkimuksen kautta termille saattaisi löytyä selkeä, yksiselitteinen määritelmä.

Lähteet

Alzahrani, Abdullah AH. 2020. "4GL Code Generation: A Systematic Review". *International Journal of Advanced Computer Science and Applications* 11 (6).

Bernsteiner, Reinhard, Stephan Schlögl, Christian Ploder, Thomas Dilger ja Florian Brecher. 2022. "Citizen vs. professional developers: differences and similarities of skills and training requirements for low code development platforms". Teoksessa *Iceri2022 Proceedings*, 4257–4264. IATED.

Bettin, Joern. 2001. "Practical Use of Generative Techniques in Software Development Projects: an Approach that Survives in Harsh Environments". Teoksessa *OOPSLA Workshop on Generative Programming*.

Bock, Alexander C, ja Ulrich Frank. 2021. "Low-code platform". *Business & Information Systems Engineering* 63:733–740.

Cabot, Jordi. 2020. "Positioning of the low-code movement within the field of model-driven engineering". Teoksessa *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 1–3.

Di Ruscio, 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?" *Software and Systems Modeling* 21 (2): 437–446.

Frank, Ulrich, Pierre Maier ja Alexander Bock. 2021. *Low code platforms: promises, concepts and prospects. A comparative study of ten systems.* Tekninen raportti. ICB-Research Report.

Gartner. 2022. "Gartner Forecasts Worldwide Low-Code Development Technologies Market to Grow 20 % in 2023". https://www.gartner.com/en/newsroom/press-releases/2022-12-13-gartner-forecasts-worldwide-low-code-development-technologies-market-to-grow-20-percent-in-2023.

Gottesdiener, Ellen. 1995. “RAD realities: Beyond the hype to how RAD really works”. *Application Development Trends* 2 (8): 28–38.

Hurlburt, George F. 2021. “Low-Code, No-Code, What’s Under the Hood?” *IT professional* 23 (6): 4–7.

Khorram, Faezeh, Jean-Marie Mottu ja Gerson Sunyé. 2020. “Challenges & opportunities in low-code testing”. Teoksessa *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 1–10.

Kleppe, Anneke G, Jos B Warmer ja Wim Bast. 2003. *MDA explained: the model driven architecture: practice and promise.* Addison-Wesley Professional.

LANSA. 2023a. “About LANSA”. https://lansa.com/about-lansa/.

———. 2023b. “Homepage”. Viitattu 12. lokakuuta 2023. https://lansa.com/.

Lebens, Mary, Roger J Finnegan, Steven C Sorsen ja Jinal Shah. 2022. “Rise of the citizen developer”. *Muma Business Review* 5:101–111.

Luo, Yajing, Peng Liang, Chong Wang, Mojtaba Shahin ja Jing Zhan. 2021. “Characteristics and challenges of low-code development: the practitioners’ perspective”. Teoksessa *Proceedings of the 15th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)*, 1–11.

Mendix. 2023a. “Homepage”. Viitattu 12. lokakuuta 2023. https://www.mendix.com/.

———. 2023b. “Why Was Mendix Founded?” https://www.mendix.com/evaluation-guide/why-founded/.

OutSystems. 2023a. “Homepage”. Viitattu 12. lokakuuta 2023. https://www.outsystems.com/.

———. 2023b. “It began with a vision”. https://www.outsystems.com/evaluation-guide/it-began-with-a-vision/.

Richardson, Clay, ja John R Rymer. 2016. "Vendor landscape: The fractured, fertile terrain of low-code application platforms". *Forrester, Janeiro*, 12.

Richardson, Clay, John R Rymer, Christopher Mines, Alex Cullen ja Dominique Whittaker. 2014. "New development platforms emerge for customer-facing applications". *Forrester: Cambridge, MA, USA* 15.

Rymer, J. 2017. "The forrester wave™: Low-code development platforms for ad&d pros, q4 2017". *Cambridge, MA: Forrester Research.*

Sahay, Apurvanand, Arsene Indamutsa, Davide Di Ruscio ja Alfonso Pierantonio. 2020. "Supporting the understanding and comparison of low-code development platforms". Teoksessa *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 171–178. IEEE.

Sanchis, Raquel, Óscar García-Perales, Francisco Fraile ja Raul Poler. 2019. "Low-code as enabler of digital transformation in manufacturing industry". *Applied Sciences* 10 (1): 12.

Shipley, Charles, ja Stephen Jodis. 2003. "Programming languages classification". *Encyclopedia of Information Systems*, 545–552.

Smith, Greg, Michael Papadopoulos, Joshua Sanz, Michael Grech ja Heather Norris. 2020. *Unleashing innovation using low code/no code–The age of the citizen developer.*

Sufi, Fahim. 2023. "Algorithms in low-code-no-code for research applications: a practical review". *Algorithms* 16 (2): 108.

Vincent, Paul, Kimihiko Iijima, Mark Driver, Jason Wong ja Yefim Natis. 2019. "Magic quadrant for enterprise low-code application platforms". *Gartner report.*