Tampere University

Niilo Haapala

EVALUATING LOW-CODE PLATFORMS FOR INTEGRATION DEVELOPMENT

Benefits, Challenges, and Practical Applications

Master's Thesis
Faculty of Information Technology and Communication Sciences
Kari Systä
David Hästbacka
11 2024

ABSTRACT

Niilo Haapala: Evaluating low-code platforms for integration development: benefits, challenges, and practical applications
Master's thesis
Tampere University
Study program of computer science
11 2024

In the rapidly evolving landscape of software development, low-code platforms are gaining traction as solutions that promise to enhance efficiency and give broader access to application development. This thesis investigates the practical implications of low-code platforms, specifically focusing on Microsoft Power Platform, OutSystems, and Mendix, in the context of integration development. The study employs a mixed-methods approach, combining a comprehensive literature review with empirical case studies to bridge the gap between theoretical benefits and real-world applications of low-code platforms.

The research identifies key advantages such as accelerated development times and increased empowerment for citizen developers, alongside potential challenges, including integration complexities and reliance on user-created connectors. The findings reveal a strong alignment between the documented benefits of low-code platforms and the experiences observed in case studies, particularly highlighting the user-friendliness and integration capabilities of Microsoft Power Platform. However, challenges persist in OutSystems and Mendix, where poorly documented connectors and the necessity of custom coding for certain integrations complicate the user experience.

This thesis contributes to the existing body of knowledge by providing actionable insights for organizations considering low-code platforms for integration projects, emphasizing the need for a balanced understanding of both their capabilities and limitations. The study concludes with recommendations for future research, including longitudinal studies to assess the long-term impacts of low-code platforms adoption on organizational agility and digital transformation.

Keywords: low-code platforms, low-code, integration, digital transformation

TIIVISTELMÄ

Niilo Haapala: Low-code alustojen arviointi integraatiokehityksessä: höydyt, haasteet ja käytännön sovellukset
Pro gradu -tutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
11 2024

Nopeasti kehittyvässä ohjelmistokehityksen kentässä low-code alustat ovat yleistymässä ratkaisuina, jotka lupaavat parantaa tehokkuutta ja mahdollistaa laajemman käyttäjäkunnan osallistumisen sovelluskehitykseen. Tämä pro gradu -tutkielma tarkastelee low-code alustojen käytännön vaikutuksia keskittyen erityisesti Microsoft Power Platformiin, OutSystemsiin ja Mendixiin integraatiokehityksessä. Tutkimuksessa yhdistetään kattava kirjallisuuskatsaus ja empiiriset tapaustutkimukset low-code alustojen teoreettisten hyötyjen ja käytännön sovellusten välisten erojen kartoittamiseksi.

Tutkimus tunnistaa keskeisiä hyötyjä, kuten nopeamman sovelluskehityksen ja kansalaiskehittäjien hyödyntämisen, sekä mahdollisia haasteita, kuten integraatioiden monimutkaisuuden ja riippuvuuden käyttäjien luomista liittimistä. Tulokset vahvistavat yhteyden low-code-alustojen dokumentoitujen hyötyjen ja tapaustutkimuksissa havaittujen kokemusten välillä, korostaen erityisesti Microsoft Power Platformin käyttäjäystävällisyyttä ja integroitavuutta. OutSystemsissa ja Mendixissa esiintyy kuitenkin haasteita, sillä huonosti dokumentoidut liittimet ja tietyille integraatioille tarvittavat kustomoidut ohjelmat vaikeuttavat käyttäjäkokemusta.

Tämä pro gradu -tutkielma täydentää olemassa olevaa tietoa tarjoamalla käytännön näkemyksiä organisaatioille, jotka harkitsevat low-code-alustojen käyttöä integraatioprojekteissa. Tutkielma korostaa tarvetta ymmärtää sekä alustojen mahdollisuuksia että niihin liittyviä haasteita. Tutkimuksen lopuksi annetaan suosituksia tuleville tutkimuksille, kuten pitkäaikaistutkimuksista, joissa selvitetään, miten low-code-alustojen käyttöönotto vaikuttaa organisaatioiden ketteryyteen ja digitaaliseen muutosprosessiin.

Avainsanat: low-code alustat, low-code, integraatio, digitaalinen muutos

USE OF AI IN THESIS

I have utilised AI tools in my thesis:

☐ No

☑ Yes

The AI tools utilised in my thesis and their purposes are described below:

Names and versions of AI tools: ChatpGPT 4o mini

Purpose of using AI tools: Polishing the language of the text and proofreading the text.

I acknowledge that I am fully responsible for the entire content of my thesis, including the parts generated by AI, and accept accountability for any violations of ethical standards in publications.

CONTENTS

1 Introduction

1.1 Background

In the evolving landscape of software development, the demand for rapid and cost-effective solutions has led to the emergence and increasing popularity of low-code platforms (LCPs). These platforms are designed to simplify the application development process by providing a visual development environment, pre-built components, and templates that allow users to design applications with minimal hand-coding. This method solves the challenges posed by traditional development cycle, which often requires extensive coding knowledge, time, and resources.

LCPs are becoming increasingly popular with businesses looking to accelerate digital transformation. According to Forrester Research, the low-code market is expected to reach $50 billion by 2028, reflecting a compound annual growth rate of 21% from 2019 (Forrester Research, 2024). This growth is driven by the platforms' ability to empower non-technical users, known as ‘citizen developers’, to create and deploy applications swiftly, thereby reducing the dependency on IT departments and professional developers.

One of the primary benefits of LCPs is their potential to enhance productivity. A study by Mendix highlights that organizations using LCPs have experienced a 50-70% reduction in development time (Mendix, 2021). This efficiency gain is attributed to the drag-and-drop interfaces, reusable components, and automated code generation that LCPs offer. Consequently, businesses can respond more quickly to market changes and customer demands, providing a competitive edge in fast-paced industries.

In the context of integration development, LCPs offer both promising opportunities and potential pitfalls. Integration projects, which involve connecting disparate systems and data sources, are inherently complex and require careful planning and execution. LCPs

can streamline these projects by providing pre-built connectors and APIs, reducing the need for custom coding. However, the efficacy of these tools in managing complex integrations remains a subject of debate among practitioners and researchers.

1.2 Research problem and research questions

While LCPs offer numerous potential benefits, their effectiveness and limitations, particularly in the context of integration development, remain under-explored. Integration development involves connecting disparate systems, data sources, and applications to work together seamlessly, which is a critical but complex task for many businesses. The problem addressed by this thesis is the lack of comprehensive understanding of the benefits and drawbacks of using low-code platforms for integration development.

Specifically, the thesis aims to identify and analyze the practical advantages and potential challenges associated with LCPs in real-world integration projects, and to compare these findings with the benefits and drawbacks documented in existing research literature. This dual approach seeks to provide a balanced view of the utility of LCPs, addressing the gap between theoretical claims and practical realities in the integration development landscape.

1. Research Question (RQ) 1: How do the theoretical frameworks and documented benefits of LCPs compare to the outcomes observed in practical integration projects?
2. RQ 2: What specific advantages and challenges do organizations face when implementing LCPs for integration development, and how do these differ across various use cases?

Objectives

- To identify the benefits and drawbacks of LCPs as documented in existing literature.
- To evaluate the practical benefits and drawbacks of LCPs through a case study.
- Comparing the theoretical insights derived from literature with the practical findings from case studies, highlighting parallels and divergences.

1.3 Delimitations

This thesis uses the Microsoft Power Platform, OutSystems, and Mendix for case studies. While there are multiple other LCPs available, this study focuses on the market leaders, which also offer free versions. It is important to note that integration-focused platforms like MuleSoft Anypoint and Frends will not be included in the case studies. This delimitation ensures a concentrated examination of platforms primarily aimed at application development rather than integration solutions.
Additionally, the research is primarily concerned with low-code applications within the domain of company information systems (e.g., internal systems, third party integrations). Low-code development (LCD) also extends to other areas, such as robotics and the Internet of Things (IoT), which are outside the scope of this study.

1.4 Structure

The first part of the research is a literature review that explores existing research on low-code and LCPs, including their benefits and drawbacks. This section provides a comprehensive overview of the theoretical foundations and key findings from prior studies, emphasizing the current state of knowledge regarding LCPs and their implications for development.

The literature review includes a detailed examination of the definitions of integration development, highlighting the distinctions between "Big I" and "Little i" integration as presented by Gulledge (2006). It combines key findings, trends, advantages, and challenges associated with LCP. The review identifies gaps in literature, particularly the lack of empirical evidence regarding the effectiveness of LCPs in real-world scenarios, underscoring the need for further research in this area.

Following the literature review, the case study chapter presents a series of small integration projects using different LCPs to evaluate their effectiveness. It includes detailed descriptions of each project, the implementation processes, and the outcomes. By highlighting specific examples and experiences from these projects, this chapter illustrates the practical benefits and challenges of using LCPs in integration development. The case studies aim to bridge the gap between theoretical insights from the literature review and practical applications, providing a deeper understanding of how various LCPs perform in real-world integration scenarios.

The results chapter will present and analyze data collected from these integration projects, employing analyses to answer the research questions and compare practical findings with theoretical insights.

Finally, the conclusion chapter will summarize the key findings of the research, provide answers to the research questions, and discuss broader implications. It will include recommendations for organizations considering the adoption of LCPs and suggest directions for future research.

2 Low-code Development with low-code platforms

The purpose of the literature review is to provide a comprehensive overview of existing research and theoretical foundations related to LCPs and their application in development. The literature search was performed using a variety of academic databases, including IEEE Xplore, Google Scholar, the ACM Digital Library, and Andor, among others. By examining a broad range of scholarly articles, industry reports, and case studies, this literature review aims to identify and synthesize key findings, trends, benefits, and challenges associated with the use of LCPs.

Keywords used in the search included “low-code development platforms,” “integration development,” “agile software development,” “application development,” and “low-code benefits and challenges.” Boolean operators were employed to combine these keywords effectively and narrow down the search results to the most pertinent studies.

The literature selection process involved several stages:

1. **Initial Screening**: Titles and abstracts of the search results were reviewed to assess their relevance to the research topic. Articles that did not directly relate to LCPs were excluded. There were few exceptions because they had other valuable info.
2. **Full-Text Review**: Selected articles were then reviewed in full to ensure they provided empirical data, theoretical insights, or case studies relevant to the study’s objectives.
3. **Quality Assessment**: The quality of the selected literature was evaluated based on criteria such as the research methodology, the credibility of the sources, and the relevance of the findings to the research questions.
4. **Data Synthesis**: The findings from the selected literature were synthesized to identify key trends, benefits, challenges, and gaps in existing research.

When searching using only keyword “low-code” there were over ten thousand hits. After using boolean operators to combine keywords like “low-code” and “benefits” or “challenges” the results were more manageable just over three hundred hits. Upon completion of the search, over five hundred articles were initially identified, out of which about fifty were deemed relevant after initial screening and included in the final synthesis. About a hundred articles were rejected due to insufficient relevance to the research focus. This process ensured a selection of literature that directly contributes to the understanding of LCPs.

This review serves several important functions within the context of this thesis. First, it establishes the current state of knowledge regarding LCPs, highlighting what is already known about their capabilities and limitations. According to Rokis and Kirikova (2023), LCPs can significantly reduce development time and costs by enabling users with minimal coding skills to create complex applications. Forrester Research (2024) notes that the market for low-code development platforms (LCDPs) is rapidly growing, driven by the need for faster and more flexible software development solutions.

Secondly, this revies identifies gaps in the existing literature that this research seeks to address, particularly the practical implications of using LCPs in real-world integration projects. While there is extensive documentation on the theoretical benefits of LCPs, such as increased productivity and enhanced collaboration between business and IT teams (Rokis & Kirikova, 2023), there is limited empirical evidence on their effectiveness in complex integration scenarios. This gap underscores the need for research that evaluates the real-world performance of LCPs in integration development projects.

Third, the literature review provides a theoretical framework that shows the research design and methodology, ensuring that the study is grounded in established academic discourse. Saunders et al. (2019) emphasize the importance of an abductive approach, which combines theoretical insights with empirical data to develop a comprehensive understanding of the research problem. This framework supports the mixed-methods approach employed in this study, which includes both qualitative and quantitative data collection and analysis.

The upcoming sections will explore the essential elements of LCDs in more detail:

3.1 Definition of Low-Code Development: This section will define LCD, providing historical context and exploring its key characteristics. It will also compare LCD to traditional software development approaches.

- **Introduction to low-code development**: This subchapter will trace the history of LCD, discussing its origins in Rapid Application Development (RAD) tools and Fourth-Generation Languages (4GLs).
- **Key Characteristics of Low-Code Development**: Here, the essential features that define LCD are examined, including visual development tools, pre-built components, and integration capabilities.
- **Basic definition**: In this subchapter, basic definition of low-code is explained and what LCD means in this thesis.
- **Comparison with Traditional Development**: This subchapter will compare the speed, required skill sets, and cost implications of low-code versus traditional software development approaches.
- **Advantages and limitations of low-code development**: This subchapter goes through advantages and limitations of LCD found in literature

3.2 Low-Code Development Lifecycle: This section will outline the typical stages involved in LCD, from requirements gathering to deployment and maintenance.

- **Requirements and Feasibility Analysis**: Discusses the importance of aligning development projects with business objectives and assessing their feasibility
- **Data Modeling in Low-Code**: Explores how LCPs facilitate data modeling through visual interfaces and APIs.
- **Definition of the User Interface**: Details the process of designing and configuring user interfaces within LCPs.
- **Implementation of Business Logic and Workflows**: Discusses how LCPs support the creation and automation of business logic and workflows.
- **Integration of External Services**: Explores how LCPs enable the integration of external services and systems.
- **Testing and Deployment**: Covers the testing and deployment processes in LCD, emphasizing efficiency and rapid delivery.

3.3 Integration Development: Defines integration development as the process of connecting various software systems for seamless data sharing, categorizing it into "Big I"

integration for comprehensive data management and "Little i" integration for diverse connectivity methods, while highlighting the ongoing relevance of these concepts in modern business practices.

2.1 Definition of low-code development

LCPs have changed the software development landscape by simplifying the creation process and making it accessible to a broader range of users. This section aims to define what constitutes LCD, providing historical context, key characteristics, and a basic definition. It will also explore the evolution of low-code, its fundamental concepts, and how it differs from traditional development approaches.

Introduction to low-code development

The term “low-code” was first used by international research and consulting company Forrester in 2014 (Bock & Ulrich 2021), but the history of LCD can be traced back to the early efforts to simplify programming and make it more accessible to non-programmers. The roots of low-code can be found in the 1980s and early 1990s with the emergence of Rapid Application Development (RAD) tools which were in response to the inefficiencies of traditional waterfall project methodologies (Di Ruscio et al., 2022). RAD tools emphasized quick iterations, user involvement in the design and development process, and the use of visual programming environments (Martin, 1991). These tools laid the foundational concepts that would eventually evolve into today’s LCPs.

Parallel to RAD, Fourth-Generation Languages (4GLs) were developed during the same period. 4GLs were designed to be more accessible than third-generation programming languages (3GLs), such as C and Java, by offering higher-level abstractions that could automate much of the coding process. These languages focus on simplifying database interactions and user interface creation, significantly reducing the amount of code developers needed to write. (Martin, 1982) These early developments laid the groundwork for modern LCPs.

Basic definition

The term “low-code” was born in 2014 and since then many definitions have been created for it (Rokis & Kirikova, 2023). Waszkowski (2019) states that low-code programming enables developers and professionals to create applications by emphasizing design and functionality, while minimizing the coding effort needed. Waszkowski does not use “low-code” term instead the author uses “low-code programming”.

Alamin et al. (2021) describe LCD as a paradigm where multi-level developers can develop applications using drag-and-drop blocks, visual programming, and a graphical user interface. According to them, LCDPs aim to move away from complex tests, deployments, and maintenance, which are used in traditional software development. The paper also mentions that LCD and Agile go hand in hand, because they are similar in their fundamental principles and goals.

Luo et al (2021) are on the same page as others about what LCD is. It is visual programming, drag-and-drop blocks, and a graphical user interface. However, the paper mentions that there is no clear definition of “low-code” academically or within the industry, and low-code users use many terms to describe practices related to low-code.

As we can see, there is no unified term for low-code and its development. In this work, LCD means the same as Rokis & Kirikova defined it:

Low-code software development is a development approach that enhances rapid, flexible, and iterative software development by enabling quick business requirements translation through visual programming with a graphical interface, visual abstraction, and minimal hand-coding; and involving practitioners with various backgrounds and software development experience. (Rokis & Kirikova, 2023)

Key characteristics of low-code development

LCD is distinguished by several fundamental characteristics that set it apart from traditional development. A critical feature is the use of visual development tools that enable users to create applications through drag-and-drop functionalities, streamlining the application lifecycle (Oteyo et al., 2021; Luo et al., 2021).

Additionally, LCPs provide extensive libraries of **pre-built components**—such as forms, charts, and buttons—that enhance efficiency and ensure consistency in applications (Rokis & Kirikova, 2023). These components, although sometimes limited in customization, allow developers to save significant time.

LCPs provide a wide range of application **templates** designed to serve as starting points for common use cases. These templates offer a basic structure and come with pre-configured settings, significantly speeding up the development process. (Rokis & Kirikova, 2023)

Another key characteristic is **accessibility for non-developers** or “citizen developers”. This democratizes the development process, making it accessible to a broader range of users within an organization (Bock & Ulrich, 2021). By enabling non-developers to build and modify applications, LCPs reduce the dependency on IT departments for every development need (Oluwaseyi, 2024).

Integration capabilities are a part of LCPs. LPCs offer robust integration capabilities with external systems, enabling seamless data exchange and process automation. These platforms typically include built-in connectors and APIs, allowing them to communicate with various external systems such as databases, ERP systems, CRM systems, and cloud services (Al Alamin et al., 2021).

Finally, **customization and extensibility** are key themes to LCPs, even though they are frequently viewed as offering limited customization options due to their simplified and streamlined nature. These platforms are equipped to support both generic and domain-specific functionalities, which enables organizations to tailor applications to their specific needs (Sahay et al., 2020). This flexibility is critical as it allows businesses to adapt and extend applications without extensive coding, thus maintaining agility and responsiveness to changing business requirements.

Comparison with traditional development

LCPs differ from traditional development in various aspects:

Development speed

LCPs significantly accelerate application development by enabling rapid prototyping through visual interfaces and pre-built components, contrasting with traditional methods that require extensive coding, debugging, and testing, which can take weeks or months (Luo et al., 2021).

Required skill set

The skill set for LCD is distinct from that of traditional software development. Traditional methods demand a deep understanding of programming languages and software engineering principles, while LCPs are designed to be accessible to non-technical users, or "citizen developers" (Martinez & Pfister, 2023).

Cost implications

The cost structure of LCPs is generally more favorable, reducing the need for skilled developers and lengthy development cycles. Traditional development incurs higher costs due to its complexity and resource demands. LCPs typically offer various pricing models based on user numbers, application deployment, and data storage, although estimating total costs can be challenging when adopted at an organizational level (Käss et al., 2022).

Advantages and limitations of low-code development

Advantages

- **Increased Development Speed:** LCPs significantly accelerate app development, enabling faster prototyping and reducing routine tasks. Research shows low-code methods can yield productivity three times higher than traditional coding (Luo et al., 2022; Bock & Ulrich, 2021; Rokis & Kirikova, 2023). Prototyping facilitates early feedback and requirements verification, leading to easier maintenance (Sahay et al., 2020; Di Sipio et al., 2020).

- **Cost savings:** LCPs lower development and maintenance costs by reducing the need for skilled developers and lengthy cycles. Free versions allow for initial development without financial investment, fostering innovation (Di Sipio et al., 2020; Martinez & Pfister, 2023). Additionally, cloud-based deployments minimize infrastructure costs (Iosup et al., 2011; Raghavendran, 2023).

- **Complexity reduction:** Visual development tools simplify app creation through drag-and-drop interfaces, making it accessible to users with minimal coding experience (Luo et al., 2021). Pre-built components streamline development, reduce errors, and enhance automation of routine tasks (Di Ruscio et al., 2022; Sahay et al., 2020).

- **Easier maintenance**: Visual environments facilitate maintenance by allowing quick adjustments without deep coding knowledge. Pre-built components enhance maintainability, as they are tested and reliable, minimizing bugs during updates (Rokis & Kirikova, 2023; Martinez & Pfister, 2023). Automation of tasks further simplifies maintenance (Waszkowski, 2019).

- **Higher Business Involvement**: LCPs enable 'citizen developers' to engage in app creation, reducing dependency on IT (Funk, 2023). This accessibility fosters rapid prototyping and continuous improvement based on real-time feedback (Elshan et al., 2023a; Elshan et al., 2023b). The ability to see immediate results and make quick changes encourages more active involvement from business profiles, as they can directly influence the development process and see the impact of their contributions in real-time. (Elshan et al., 2023b)

- **Minimized Requirements Instability**: Real-time prototyping allows for quick visualization of requirements, addressing doubts often seen in traditional methods (Beaton, 2022; Elshan et al., 2023b). This immediate feedback loop helps in refining the requirements early in the development process, significantly reducing the risk of instability or inconsistency.

- **Alignment with Agile Methodologies**: Agile methodologies have become a cornerstone of modern software development, emphasizing flexibility, rapid iteration, and continuous delivery (Freedman, 2018). LCPs are inherently aligned with the principles of Agile development, which is why they are frequently mentioned in literature related to LCPs, either directly or indirectly through features that enhance agility (Totterdale, 2018; Luo et al., 2021; Martinez & Pfister, 2023; Bock & Ulrich, 2021).

Limitations

- **Lack of customization**: LCPs often limit customization due to restricted control over the underlying code. While they simplify development through visual elements, this abstraction can hinder developers from fine-tuning applications to meet specific needs. As a result, unique functionalities may be challenging to implement, becoming a bottleneck for those requiring more flexibility (Luo et al., 2021; Martinez & Pfister, 2023).

- **Limitations in terms of scalability**: Scalability presents a significant challenge for LCPs as applications grow. Although they excel in rapid development, performance

bottlenecks may arise when handling larger datasets or increased user traffic, potentially leading to slower response times and instability (Tisi et al., 2019; Yan, 2021). However, some of the platforms claim that they have solved the scalability problem (Matvitskyy et al., 2023).

- **Fragmentation**: Fragmentation occurs when different LCPs use varied programming models, complicating data integration across platforms. This vendor lock-in can prevent applications from communicating effectively, leading to data silos that hinder collaboration and decision-making (Sanchis et al., 2020; Moitinho, 2023). Additionally, integrating fragmented applications often necessitates custom workarounds, which can be time-consuming and costly.

- **Vendor lock-in**: Vendor lock-in occurs when an organization becomes overly dependent on a single LCP, making it difficult to switch to another platform or integrate with different technologies. Most LCPs use proprietary technologies, which means that applications built on one platform are often incompatible with other platforms (Käss et al., 2022).

- **Difficult to estimate total cost**: Each LCP has unique pricing models based on user numbers, application counts, and data requirements (Käss et al., 2022). Additional costs may also arise from using specific connectors, such as databases (Martinez & Pfister, 2023)

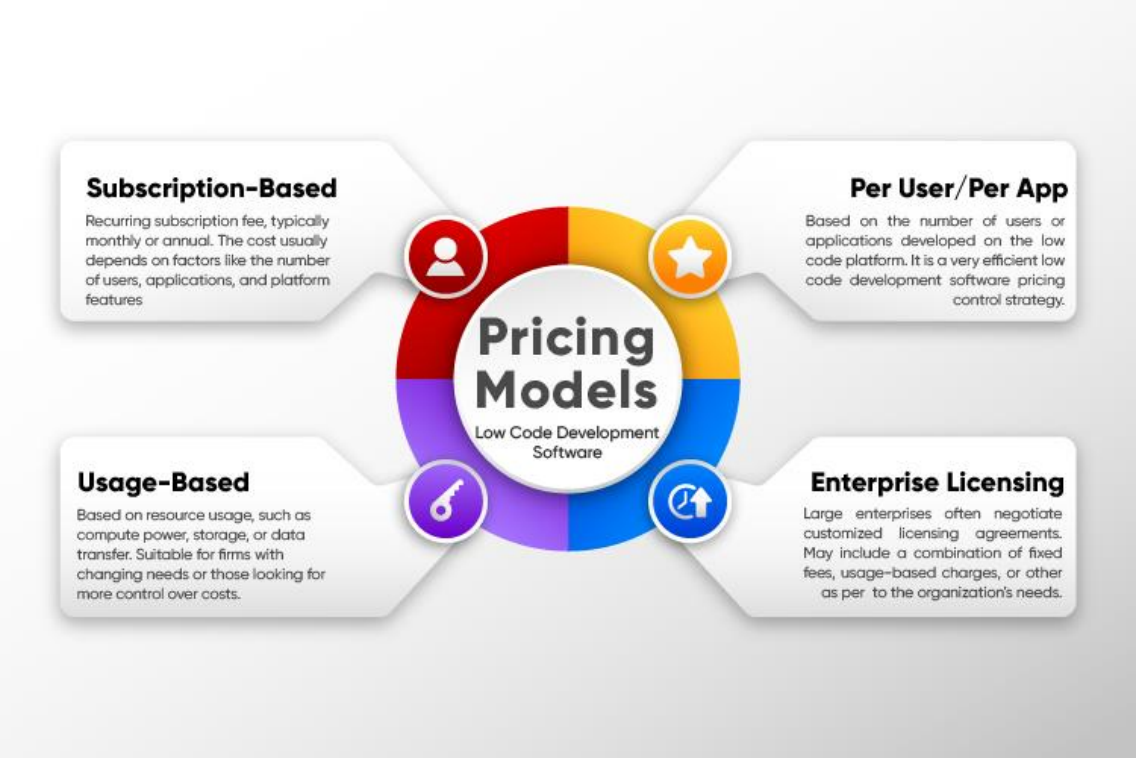


Figure 1. *LCP pricing models (Joshi, 2023)*

- **Security**: While some studies debate the security of LCPs, challenges often stem from the management of software development rather than technical factors. Citizen developers may lack the expertise of trained IT professionals in implementing corporate security practices, increasing the risk of vulnerabilities and compliance issues (Hintsch et al., 2021). Furthermore, the rise of Shadow IT—applications created without IT oversight—can worsen these security risks and hinder governance efforts (Begonha et al., 2022; Elshan et al., 2023b).

2.2 Low-code development lifecycle

This chapter outlines the Low-Code Development Lifecycle (LCDLC) (Pańkowska, 2024), emphasizing streamlined processes that enable rapid application development with minimal coding. Typically, there are two approaches to creating an LCD application: either by developing the user interface (UI) first and then connecting it to data sources, or by establishing the data model first and subsequently designing the UI (Al Alamin et al., 2021). Rokis and Kirikova (2023) identified key principles for LCD, including selecting appropriate platforms, utilizing visual tools, empowering citizen developers, and employing iterative processes. They highlighted essential steps such as feasibility analysis, data modeling, defining user interfaces, and testing and deployment.

Requirements and feasibility analysis

This phase is critical for aligning development projects with business objectives and assessing the proposed solution's practicality. Requirements must be clear, consistent, and testable, categorized into non-functional, system-characteristic, and constraint requirements (Silhavy et al., 2011). Gathering requirements can be challenging; methods such as interviews, workshops, and questionnaires make gathering requirements easier, but they should be chosen carefully based on the project's context (Lane et al., 2016).

Data modeling in low-code

Data modeling is essential for developing LCD applications, creating representations of data structures using visual tools like diagrams or graphs. LCPs facilitate this process, allowing users to connect to external data sources via APIs. (Bock & Ulrich, 2021)

Definition of the user interface

This phase involves designing the application's forms and views, determining user interactions, and configuring security mechanisms. Low-code platforms typically offer drag-and-drop features for easy customization. (Maia, 2022)

Implementation of the business logic and workflows

This step defines control and data flows using graphical workflows and business rules, facilitating integration with external services through APIs. Graphical workflows and textual business rules are examples of business logic specifications commonly used in this process. These workflows show how data and processes flow within the application. LCDPs typically provide support for integration with external services and data sources, enabling users to connect or integrate various services to build forms or compile data reports. (Bock & Ulrich, 2021)

Integration of external services

The integration of external services refers to the capability provided by LCD tools to connect with external services, systems, or data sources through Application Programming Interfaces (APIs). This functionality enables users to interact with and utilize external

services, such as web services, RESTful services, or other individual interfaces of large technology platforms, within their low-code applications. (Maia, 2022)

Testing and deployment

Testing verifies functionality and performance, while deployment involves making applications available to users. LCPs emphasize efficient deployment processes (Maia, 2022).

Proposed low-code development lifecycle

Małgorzata Pańkowska (2024) presents a model highlighting citizen developers' crucial role in software creation. The LCDLC model involves preparation, generation, improvement, evaluation, deployment, and continuous improvement, emphasizing collaboration between stakeholders.

This model's focus on end-user involvement contrasts with traditional methods, ensuring applications align with user needs. Additionally, LCDP facilitators support end-users throughout development, enhancing project success (Pańkowska, 2024). Challenges such as scalability and the need for ongoing support must be addressed to fully leverage LCPs in software development.

2.3 Integration development

In today's digital landscape, businesses rely on several of software applications and systems to operate efficiently. Integration development has emerged as a crucial practice that enables these disparate systems to communicate and work together effectively. By easing data exchange and workflow automation, integration development streamlines operational efficiency, improve decision-making, and ultimately drives business growth.

What is Integration Development?

Integration development refers to the process of connecting various software applications, systems, or components to enable them to function together. This process can involve different methods and technologies, such as application programming interfaces (APIs), middleware solutions, and data integration platforms (Gulledge, 2006). The pri-

mary goal of integration development is to create seamless communication between systems, allowing for data sharing and collaboration across different departments and functions within an organization.

Gulledge (2006) splits integration into two primary categories: "Big I" integration and "Little i" integration, each serving distinct purposes within the realm of software applications and business processes.

Big I integration involves managing all relevant data within a solid software application that oversees specific business processes. This method guarantees that any changes made in one part of the application are automatically updated across all associated business processes, thereby removing the necessity for complicated external interfaces. By ensuring data is stored only once, Big I integration allows for immediate data sharing among all processes linked to the application. This definition highlights a careful and focused approach to integration, aiming to establish a single, reliable source of truth for business operations. (Gulledge, 2006)

Little i integration refers to a range of methods for connecting different systems, including point-to-point integration, database-to-database integration, data warehouse integration, enterprise application integration (EAI), application server integration, and business-to-business (B2B) integration. Each of these approaches offers distinct ways to facilitate data sharing and system connectivity, often varying in terms of complexity and cost. (Gulledge, 2006)

Although these definitions may originate from earlier frameworks, they remain highly relevant in today's landscape of integration development. The foundational concepts of Big I and Little i integration continue to inform how organizations approach data management and system connectivity, particularly as businesses increasingly rely on integrated solutions to streamline operations and enhance data accuracy. As technology evolves, these definitions provide valuable perspectives to evaluate and implement effective integration strategies in modern environments.

3 Development of integration on low-code platforms

This chapter explains the practical aspects of integrating systems using LCPs. It begins by introducing three LCPs—Microsoft Power Platform, OutSystems, and Mendix—highlighting their features and relevance in the integration landscape. The chapter then presents a series of case studies that demonstrate how these platforms can be effectively utilized to address common integration challenges. Each case study details the approach taken, the steps involved, and the outcomes achieved. Finally, the chapter includes a comparative analysis of the platforms based on key areas such as ease of use, integration capabilities, development time, and suitability for various use cases. This exploration aims to provide insights into the effectiveness of LCPs in real-world integration scenarios.

3.1 Integration platforms

Three LCPs were selected to be used in the case studies. Those platforms are Microsoft Power Platform, OutSystems, and Mendix. Platforms were chosen because they are market leading, which also offer free versions. The next subsections will give more information on each platform.

3.1.1 Microsoft Power Platform

The Microsoft Power Platform is a comprehensive suite of tools designed to facilitate the development of applications, automate workflows, and analyze data through a low-code approach. It consists of four primary components: Power Apps, Power Automate, Power BI, and Power Pages. Each component of the platform can be used independently or together.

Power Apps allows users to create custom applications tailored to specific business needs without extensive coding knowledge. Users can design applications using a drag-and-drop interface and can connect to various data sources, including Microsoft services, databases, and third-party applications, enabling them to build apps that work across devices.

Power Automate (formerly known as Microsoft Flow) streamlines business processes by automating repetitive tasks and workflows. Users can set up triggers and actions that connect different applications and services, which helps to reduce manual intervention and improve efficiency.

Power BI is a business analytics tool that enables users to visualize and share insights from their data. It allows for the integration of multiple data sources, transforming raw data into interactive dashboards and reports that support informed decision-making.

Power Pages is a secure, enterprise-level low-code software-as-a-service (SaaS) solution designed for creating, hosting, and managing modern business websites aimed at external users. Users with low coding experience can design, set up, and launch websites that function across various web browsers and devices. For more complex business needs, professional developers can enhance and customize these features further.

The Microsoft Power Platform is integrated with Microsoft 365 and Dynamics 365, which facilitates seamless data sharing and collaboration among users. Additionally, it offers a range of connectors to various external applications, which can enhance functionality and connectivity across systems. (Microsoft, 2024a)

Power Fx is the low-code programming language utilized across the Microsoft Power Platform. It is designed as a general-purpose, strongly typed, declarative, and functional programming language. This language is characterized by its user-friendly text format, which allows for easy understanding and application by users with varying levels of technical expertise. (Microsoft, 2024b)

3.1.2 OutSystems

OutSystems is an LCP designed to streamline and accelerate the process of creating and managing enterprise-grade applications. The platform offers a visual development environment that allows users to create web and mobile applications through drag-and-drop functionality, minimizing the need for extensive coding knowledge. This capability enables both professional developers and citizen developers to collaborate effectively in the development process. Additionally, OutSystems supports the integration of existing systems, facilitating seamless connectivity between new applications and legacy systems. (OutSystems, 2024)

OutSystems uses its own visual programming language called Reactive Web. It also supports JavaScript, C#, and CSS for advanced customizations.

3.1.3 Mendix

Mendix is an LCP designed to streamline the application development process by allowing users to create, test, and deploy applications with minimal coding expertise. It offers a range of features that support rapid application development, including visual development tools, drag-and-drop functionalities, and pre-built templates.

One of the key features of Mendix is its ability to integrate with various data sources and external systems, allowing users to build applications that can interact with existing infrastructures. The platform also supports both cloud and on-premises deployments, making it flexible for different organizational needs. Additionally, Mendix provides support for building mobile applications and offers tools for testing and monitoring the performance of applications post-deployment. (Mendix, 2024a)

Mendix Runtime is implemented using Scala and Java and it runs on Java Virtual Machine. So, developers can build, customize, and extend apps using CSS, Java and JavaScript. (Mendix, 2024b)

3.2 Cases

This section presents a series of integration cases developed using LCPs—Microsoft Power Platform, OutSystems, and Mendix. Each case demonstrates the capabilities of these platforms in real-world scenarios.

The cases were selected based on their relevance, in my opinion, to common integration challenges faced by organizations today. There are some studies that support this opinion (Setiawan & Sibarani, 2024; Patel & Palaskar, 2024). Each case provides insights into different aspects of integration, showcasing how LCPs can be leveraged to improve efficiency, streamline workflows, and enhance collaboration.

3.2.1 Case 1: REST API functionality

This case focuses on automating the tracking and management of inventory across multiple locations in real-time. The integration will get data from databases that LCPs use, which will be used as an ‘inventory management system’. After that data is sent to Excel or Google Sheets which works as other locations. Integration has three steps:

1. **Retrieve Data**: Extract inventory data from database into the LCP.
2. **Update Data**: Implement features to modify inventory levels based on new entries or changes in stock.
3. **Sync Data Back**: Send the updated inventory data to Google Sheets or Excel, ensuring that all changes are reflected in real time.

While this integration may seem straightforward, it demonstrates a common and essential use case for many organizations. It demonstrates how companies can leverage LCPs to automate data management processes.

3.2.2 Case 2: Multiple Data Sources

This case aims to generate a consolidated report by pulling data from various sources, including databases, APIs, and Excel files. The resulting reporting application will aggregate data and save it to database. There are two steps:

1. **Create connection**: Connect to identified data sources
2. **Aggregate data**: Design a data aggregation process that combines the pulled data into a single dataset

This integration plays a crucial role in enhancing decision-making processes within organizations. By consolidating data from various sources, businesses can gain a holistic view of their operations, allowing for more informed and timely decisions (Mendix, 2021).

Power platform uses its own database called Dataverse.

3.2.3 Case 3: SFTP transfer

This case focuses on automating the transfer of files from an SFTP server to a designated location. There are two steps:

1. **Retrieve Data**: Get all files from SFTP folder.
2. **Send files**: Move the files one by one to the destination folder over SFTP.

This type of integration is the most used ones. Through this integration, organizations can manage their file transfers, minimizing the risk of manual errors and ensuring that most current data is readily available.

The necessary servers will be hosted on the same Windows machine, and all cases will utilize an SFTP server created with OpenSSH.

3.3 Comparison areas

In evaluating the effectiveness of LCPs for integration projects, it is essential to identify key comparison areas that reflect their capabilities and suitability for various organizational needs. This section outlines four critical areas: **Ease of use, Integration capabilities, Development time,** and **Use case suitability**. Each area will provide insights

into how well these platforms perform in real-world scenarios, particularly in terms of user experience, technical compatibility, efficiency in development processes, and alignment with specific business requirements.

While there are additional areas that could be evaluated, such as maintainability, testability, and reusability—often regarded as strengths of LCPs—focusing on these four key areas keeps the scope of this thesis manageable.

Ease of use

This area evaluates how user-friendly each LCP is for both non-technical users and professional developers. Key factors include the intuitiveness of the user interface, the learning curve associated with the platform, and the availability of tutorials or support resources. These key factors are also how the ease of use will be evaluated.

Integration capabilities

Integration capabilities refer to a platform’s ability to connect with other systems, databases, and APIs. This area assesses the variety and quality of pre-built connectors, the ease of integrating external services, and the support for real-time data synchronization. Evaluation will involve reviewing case studies integration capabilities.

Development time

Development time focuses on the speed at which applications can be built on each platform. Factors influencing this include the efficiency of visual development tools, the availability of reusable components, and the simplicity of the development process. This area will be assessed based on the time taken to complete specific integration tasks within the case studies, comparing the experiences across different platforms.

Use case suitability

This area evaluates how well each LCP manages with the case studies conducted within this thesis. It examines the platforms' capabilities in developing various types of applications.

Through these evaluations, insights will be gained regarding how effectively each platform meets the varying requirements of different use cases, providing a clearer understanding of their practical applicability in real-world scenarios.

4 Implementation

This chapter offers an overview of the execution of the integration cases described in section 4.2, highlighting the specific approaches taken with each LCP. Each platform is examined in its dedicated section, which includes subchapters focusing on the individual cases. Furthermore, a subchapter will discuss the insights gained from these cases and how these findings correlate with the established comparison areas for LCPs.

4.1 Microsoft Power Platform

4.1.1 Case 1

Case 1 has a few steps:

1. Get the modified row from Dataverse to LCP.
2. Check if row was deleted if so, delete row from Excel.
3. If row was created or modified continue.
4. Either create or modify the row to Excel.

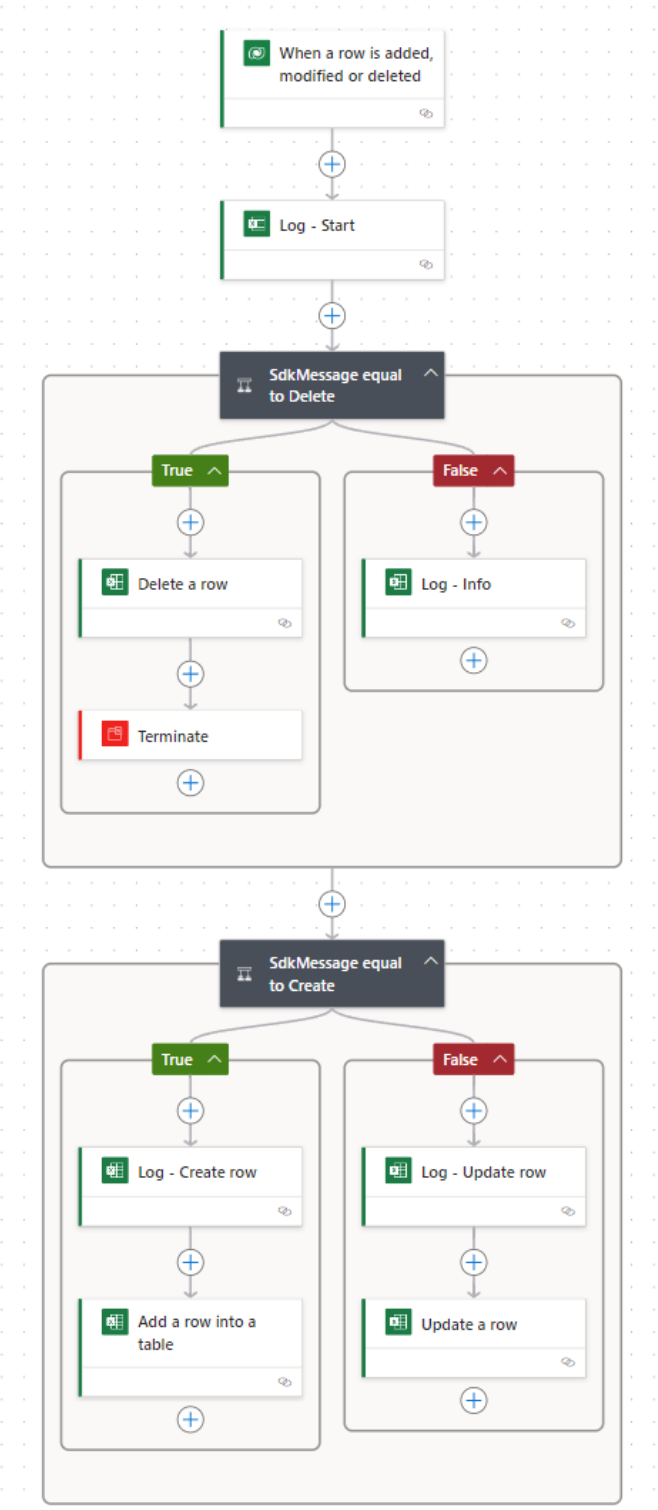
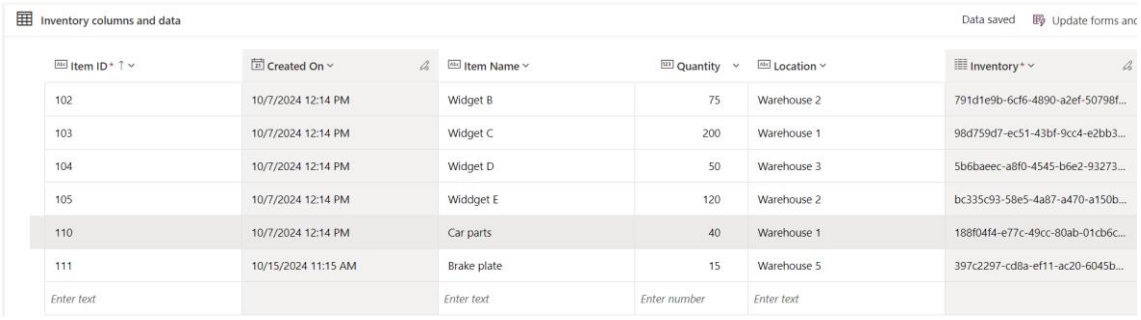


Figure 2. *Whole flow of Case 1*

Case 1 is straightforward, as it involves completing all necessary steps using only pre-built connectors. The source of the flow is Dataverse, while the destination is an Excel file. The flow is initiated by the 'When a row is added, modified, or deleted' connector, which triggers the flow whenever a row is added, modified, or deleted in the specified Dataverse table, as illustrated in Figure 3.



Item ID *	Created On	Item Name	Quantity	Location	Inventory *
102	10/7/2024 12:14 PM	Widget B	75	Warehouse 2	791d1e9b-6cf6-4890-a2ef-50798f...
103	10/7/2024 12:14 PM	Widget C	200	Warehouse 1	98d759d7-ec51-43bf-9cc4-e2bb3...
104	10/7/2024 12:14 PM	Widget D	50	Warehouse 3	5b6baeec-a8f0-4545-b6e2-93273...
105	10/7/2024 12:14 PM	Widdget E	120	Warehouse 2	bc335c93-58e5-4a87-a470-a150b...
110	10/7/2024 12:14 PM	Car parts	40	Warehouse 1	188f04f4-e77c-49cc-80ab-01cb6c...
111	10/15/2024 11:15 AM	Brake plate	15	Warehouse 5	397c2297-cd8a-ef11-ac20-6045b...
Enter text		Enter text	Enter number	Enter text	

Figure 3. *Dataverse table used in Case 1*

The connector utilizes a Webhook, implementing a push-based communication model that is triggered automatically by specific events, such as the addition of a new row in Dataverse. The configuration process is straightforward, requiring key components such as ‘Change Type’, ‘Table Name’, and ‘Scope’, as shown in Figure 4. In this case, the change type is set to ‘Added, Modified, or Deleted’, the table name is specified as ‘Inventories’, and the scope is designated as ‘Organization’. Once configured, the connector monitors for changes in ‘Inventories’ table, while the subsequent connectors in the flow send requests to Excel based on the modifications made to the row.

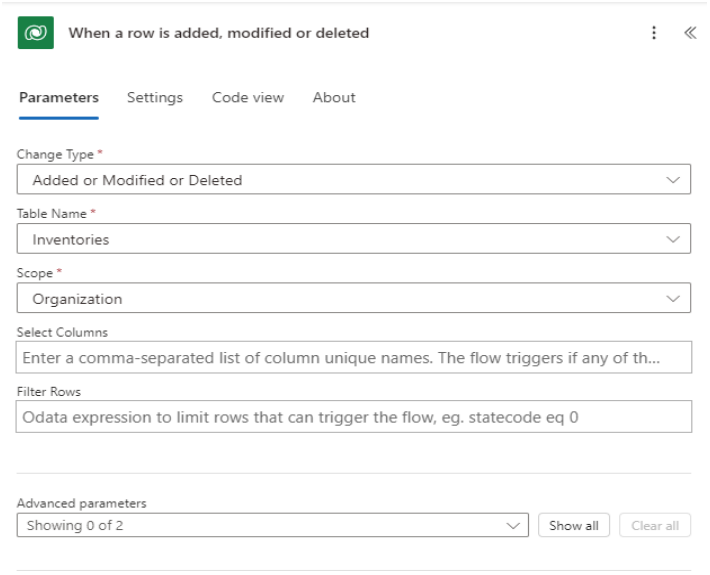


Figure 4. *Trigger connector used in Case 1*

The Excel connectors handle the rest of the flow after the row is read. There are also some Excel connectors that are used for logging. Those are only used for debugging. The Excel connectors support all the possibilities either create, update, or delete a row. The Excel connector is also easy to configure. Main parts are ‘Location,’ ‘Document Library,’ ‘File,’ ‘Table,’ ‘Key Column,’ and ‘Key Value’ are used in update and delete row but not in creating a new row. Also advanced parameters can be used when creating a new or modifying a row. These parameters tell what columns are created or modified. In

this case the key value and some of the advanced parameters are used, which are accessed from dynamic content for example key value is “@triggerOutputs()?['body/crecc_inventoryid’]” and location is “@triggerOutputs()?['body/crecc_location']”. Power Automate provides these shortcuts to these variables, which can be seen in Figure 6.

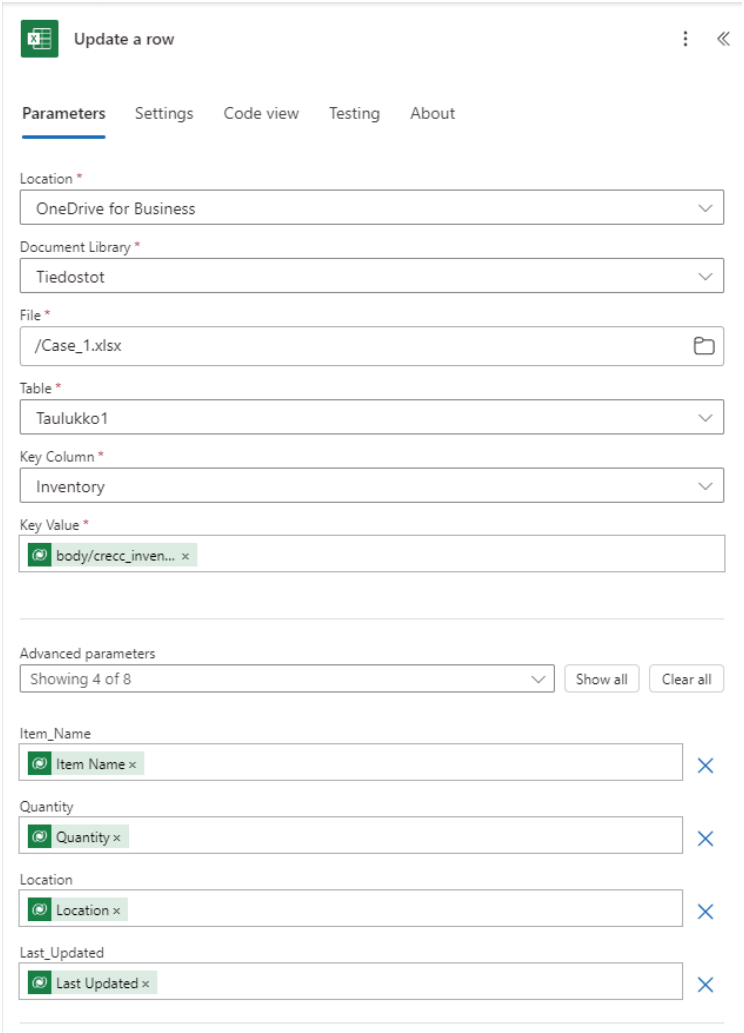


Figure 5. *Excel ‘update a row’ connector in Case 1*

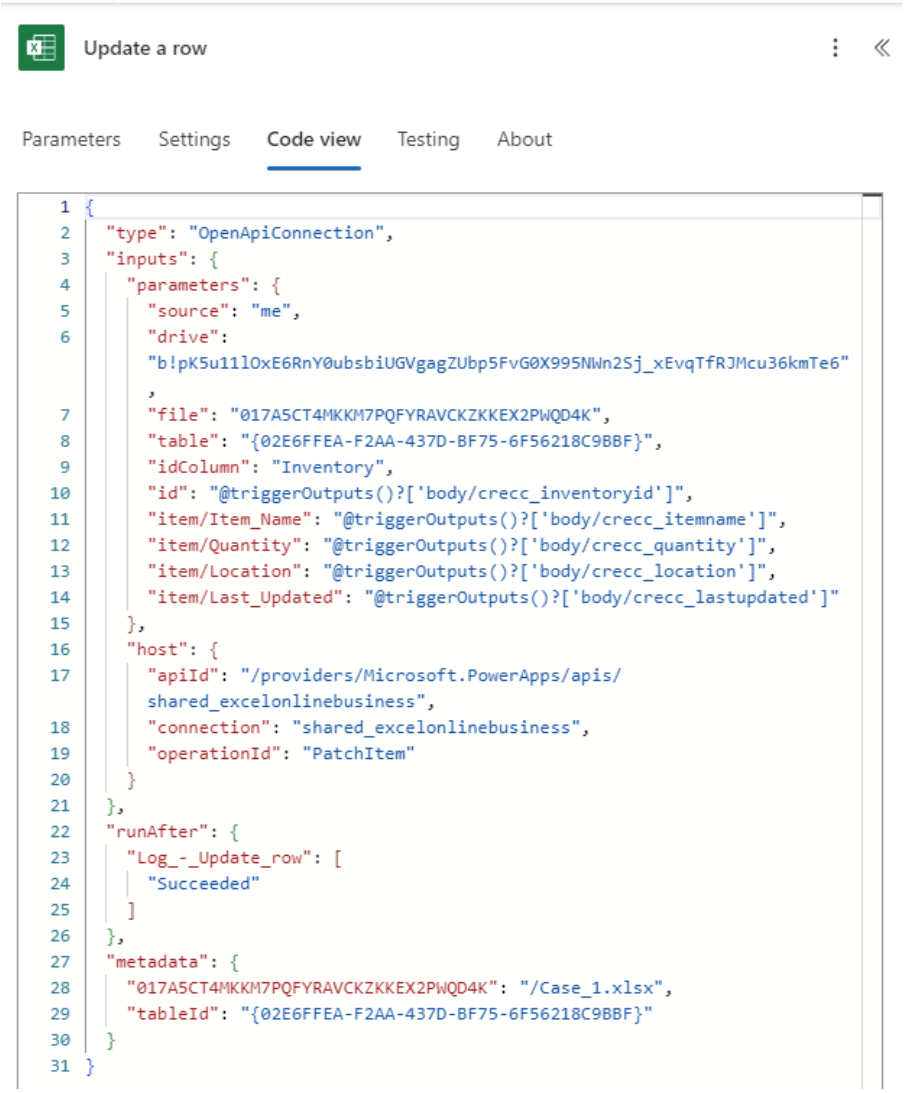


Figure 6. Code view of Figure 5.

4.1.2 Case 2

Case 2 has three concrete steps:

1. Get rows from inventory in this case from Excel
2. Get exchange rate from EUR to USD
3. In for each loop get exact row from Dataverse and update the price there.

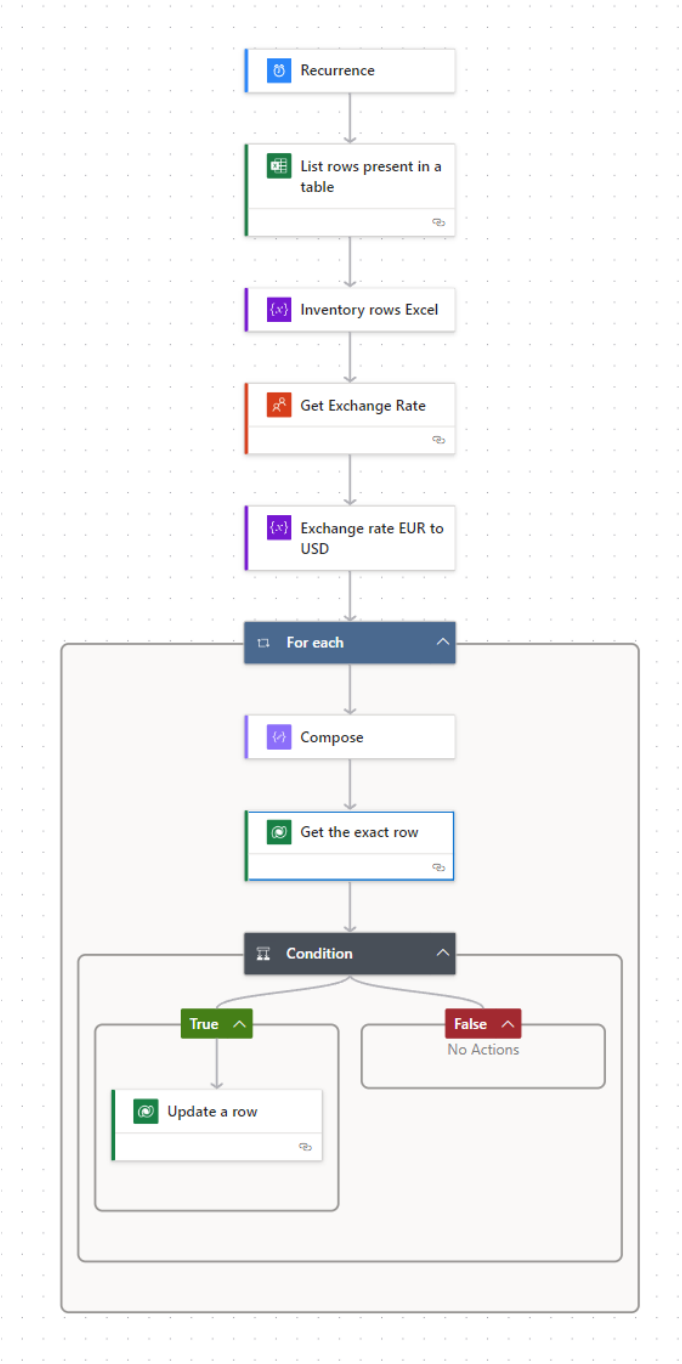


Figure 7. *Whole flow in Case 2*

The implementation of this case involves four connectors: ‘List rows present in a table,’ ‘Get Exchange Rate,’ ‘Get the exact row,’ and ‘Update a row.’ The flow initiates with the ‘Recurrence’ trigger, which is set to execute every five minutes. The ‘List rows present in a table’ connector retrieves all rows from the Inventory table in Excel, as shown in Figure 8.

	A	B	C	D	E	F	G
1	Item_ID	Item_Name	Quantity	Location	Last_Updated	Prices_EUR	Inventory
2	102	Widget B	75	Warehouse 2	2024-10-15T00:00:00Z	300,00 €	791d1e9b-6cf6-4890-a2ef-50798f6186ec
3	103	Widget C	200	Warehouse 1	2024-10-15T00:00:00Z	24,99 €	98d759d7-ec51-43bf-9cc4-e2bb3246aa00
4	104	Widget D	50	Warehouse 3	2024-10-15T00:00:00Z	12,45 €	5b6baeec-a8f0-4545-b6e2-932731ac7b03
5	105	Widget E	120	Warehouse 2	2024-10-15T00:00:00Z	0,89 €	bc335c93-58e5-4a87-a470-a150b3b1f297
6	111	Brake plate	15	Warehouse 5	2024-10-15T00:00:00Z	130,24 €	397c2297-cd8a-ef11-ac20-6045bd9f816d

Figure 8. *Inventory used in Case 2*

The ‘List rows present in a table’ connector functions similarly to its use in Case 1. It retrieves rows from a file named ‘Case_2.xlsx’ and specifically from the table named ‘Inventory.’ Although there is an option to utilize advanced parameters to filter and obtain only specific rows, this feature is not used in this case.

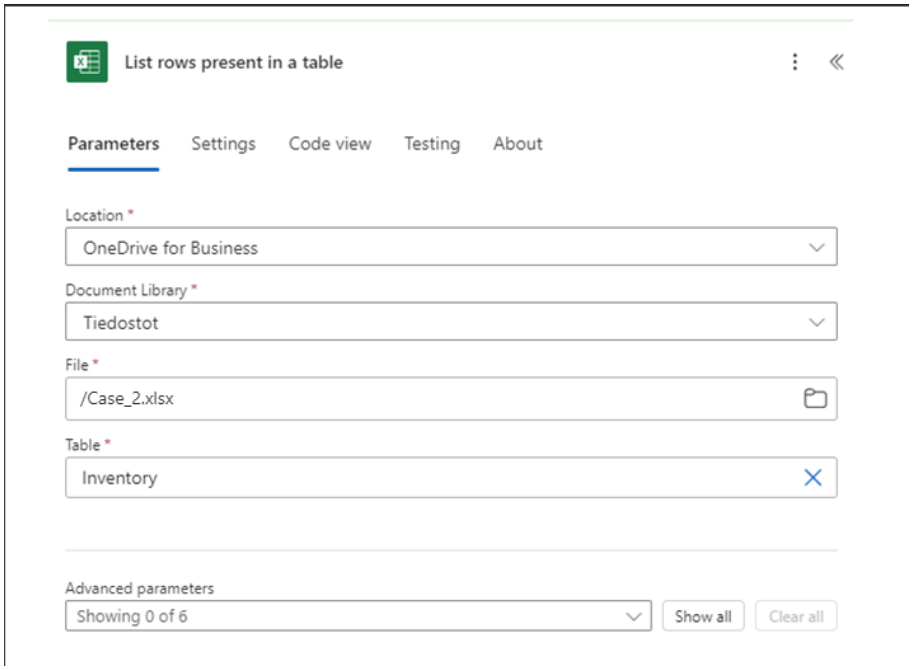


Figure 9. *Connector to get rows from excel in Case 2*

After the rows are fetched, the ‘Get Exchange Rate’ connector retrieves the current exchange rate from the ExchangeRate-API. To establish a connection to the API, this connector utilizes an API key, requiring only the ‘Base Currency’ and ‘Target Currency,’ which are set to EUR and USD in this case.

Subsequently, a ‘for each’ loop is employed to iterate through all the rows obtained from the ‘List rows present in a table’ connector. Each row is fetched from Dataverse using the ‘Get the exact row’ connector, as illustrated in Figure 10. To retrieve the specific row, a filter is applied, with the filter code formatted as follows: “crecc_item_id eq ‘@{item()?[‘Item_ID’]}’”. In this context, ‘crecc_item_id’ refers to the corresponding field in the Dataverse table, while ‘item()?['Item_ID']’ represents the same ID from the Excel Inventory. The 'eq' operator signifies equality. This method allows for the precise retrieval

of each row from Dataverse, after which the ‘Update a row’ connector is utilized to update the relevant row in Dataverse.

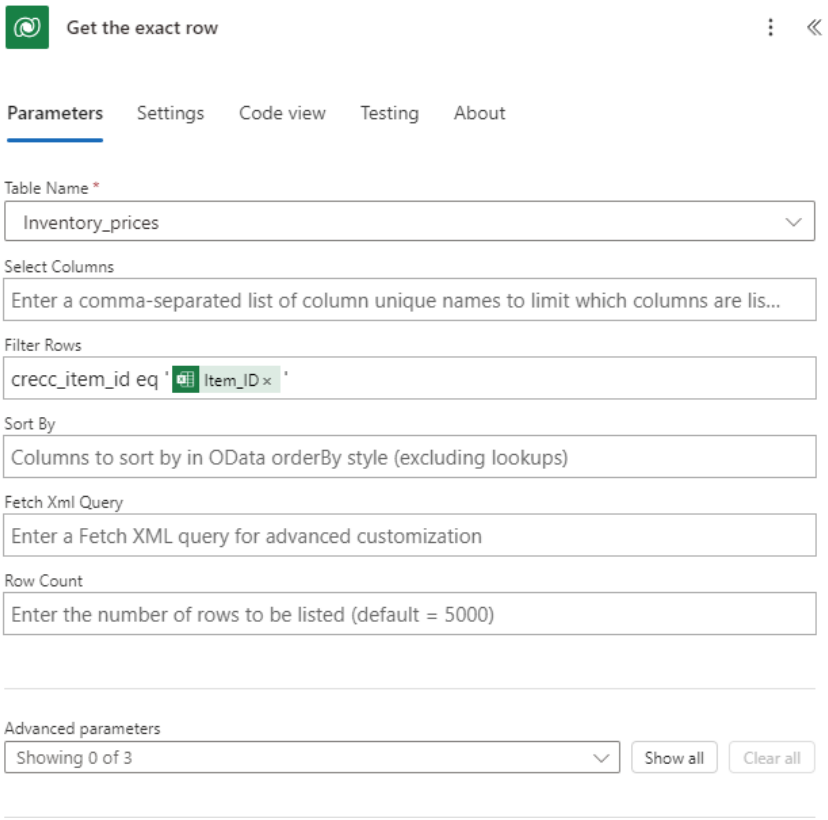


Figure 10. Get a row from Dataverse used in Case 2

In the ‘Condition’ part it is checked that ‘Get exact row’ connector returns a row. If it does not return a row ‘for each’ loop start again, if a row is found the row is updated.

4.1.3 Case 3

The last case has three steps:

1. First list all files in SFTP source folder
2. Loop each file and get file content
3. Create new file to SFTP destination folder

Like Case 2, this case utilizes the 'Recurrence' trigger to execute the flow every five minutes. As shown in Figure 11, there are three SFTP connectors in use. All these connectors share the same connection settings, which can be configured in the ‘Connec-

tions' tab of Power Automate. Figure 12 provides an illustration of this connection, highlighting several mandatory fields that must be filled out, including the 'Host Server Address,' 'User Name,' and 'Password'.

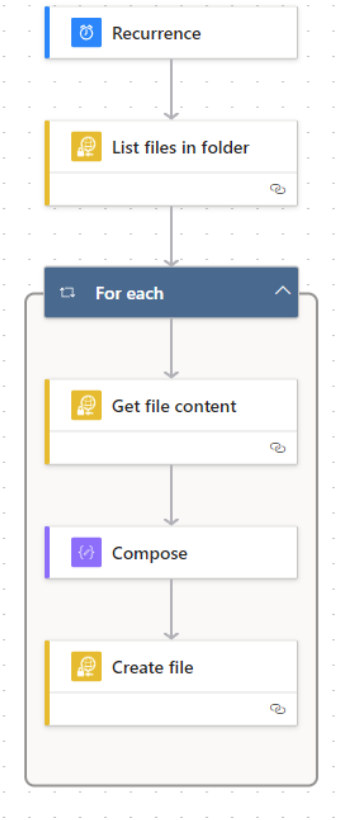


Figure 11. *Whole flow Case 3*

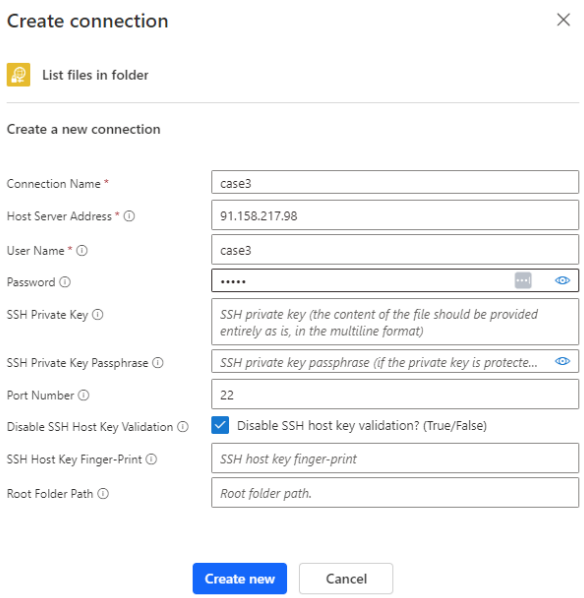


Figure 12. *SFTP connection used in Case 3*

The initial step involves the ‘List files in folder’ connector, which retrieves all files from the source folder. For each file in the list, the ‘Get file content’ connector is employed to access the content of the current file. Following this, the ‘Create file’ connector is used to generate a file in the SFTP destination folder. This connector requires specific configurations, as illustrated in Figure 13. The necessary fields for the ‘Create file’ connector include ‘Folder Path,’ ‘File Name,’ and ‘File Content’. As in Case 1, shortcuts are utilized to extract the variables for both ‘File Name’ and ‘File Content’.

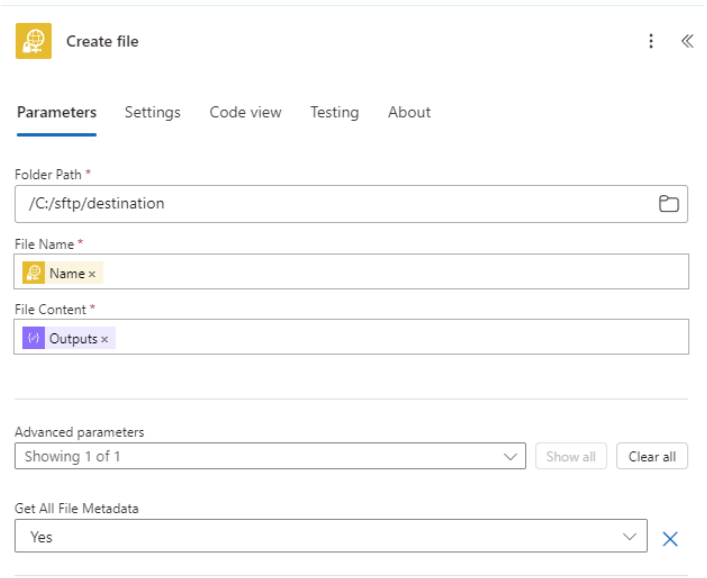


Figure 13. *'Create file' connector used in Case 3*

4.1.4 Findings

Ease of use

Throughout these cases, I found that the learnability of Power Platform is quite reasonable. Configuring pre-defined connectors is straightforward for both non-technical users and professional developers. However, users who need to utilize Power Fx, the platform's programming language, may encounter challenges. The user interface is intuitive, with a clear method for adding new steps simply by clicking the ‘+’ icon. Additionally, each connector features an ‘About’ tab that provides access to documentation, offering an easy way for users to learn how to use each connector.

While the platform is easy to learn, the learning curve can be steep. It is simple to use when working exclusively with pre-defined connectors and there are minimal data changes involved. However, when it comes to creating custom connectors, the involvement of professional developers becomes necessary. On the positive side, there are

comprehensive tutorials and support forums available for users' facing difficulties or seeking assistance.

Integration capabilities

For these cases, Power Platform demonstrated excellent integration capabilities. The components were straightforward to configure, and the system functioned nearly seamlessly. Since I only utilized pre-defined components, I cannot speak to the complexity of integrating custom connectors.

Power Platform makes it easy to integrate external services, allowing for connections with APIs, SFTPs, and various databases. I specifically used Dataverse, Power Platform's native database, but there are also pre-defined connectors available for multiple other databases, including Azure Database, MySQL, and PostgreSQL.

Additionally, real-time data synchronization is supported, as evidenced by Case 1, which successfully implemented this feature with minimal effort.

Use case suitability

For these cases, the suitability of Power Platforms was good. The processes were straightforward and relatively easy to implement. However, one significant drawback was the absence of user-friendly logging tools, which could have enhanced the overall experience. Aside from this limitation, Power Platform successfully eased the creation of these integrations without encountering major issues.

Development time

The development took more time than I anticipated. The start took the most of time, when I was learning how to get everything working, creating environments and connectors. Development by case:

- Case 1: 2 hours
- Case 2: 1 hour 30 minutes
- Case 3: 30 minutes

As I mentioned it took some time to learn how to use Power Platform, that is the reason why Case 1 took so much time compared to others.

4.2 OutSystems

4.2.1 Case 1

The case consists of three steps:

1. Get data from database
2. Get data from Excel file in Google Drive
3. Compare rows and update or create rows
4. Create update Excel file in Goole Drive

This case has slight differences compared to the implementation done with Power Platforms. One notable limitation was the lack of effective methods for removing rows from Excel, which meant this functionality was not made into the case.

Case 1 with OutSystems was not as simple as thought. It required the use of connector made by users and the connector was poorly documented.

Implementing Case 1 with OutSystems proved to be more complex than initially anticipated, as it relied on a user-created connector that was poorly documented. The source of this case was the ‘Inventory’ database located within the OutSystems environment, while the destination was an Excel file stored in Google Drive. Additionally, the flow is not automated; users must manually initiate the process by pressing the ‘Update Excel File’ button, as shown in Figure 14.

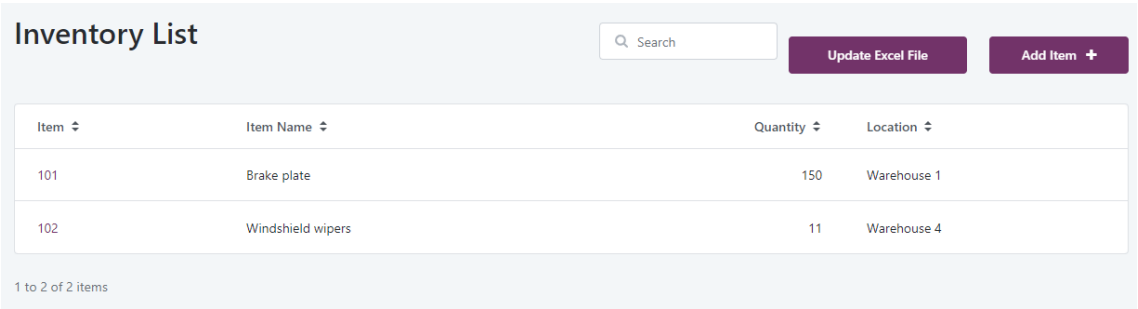


Figure 14. *User interface for Case 1*

After the button is pressed, the flow starts, as shown in Figure 15, triggering the ‘SaveToExcel’ action. This action functions like a method, allowing for the segmentation of functionalities into smaller, manageable components. Most of the logic is within ‘SaveToExcel’, as shown in Figure 16.

The ‘GetInventories’ call retrieves rows from the ‘Inventory’ database located within the OutSystems environment. Following this, the Excel file is obtained from Google Drive,

and the appropriate worksheet is selected. The flow then iterates through each row found in the ‘Inventory’ database, fetching the corresponding row from Excel. Each row is updated or created based on its existence. Once all rows have been processed, the updated Excel file is returned to the main flow. Finally, the updated file is sent back to Google Drive using the ‘Patch_Files’ action.

Figure 15. *Main flow for Case 1*

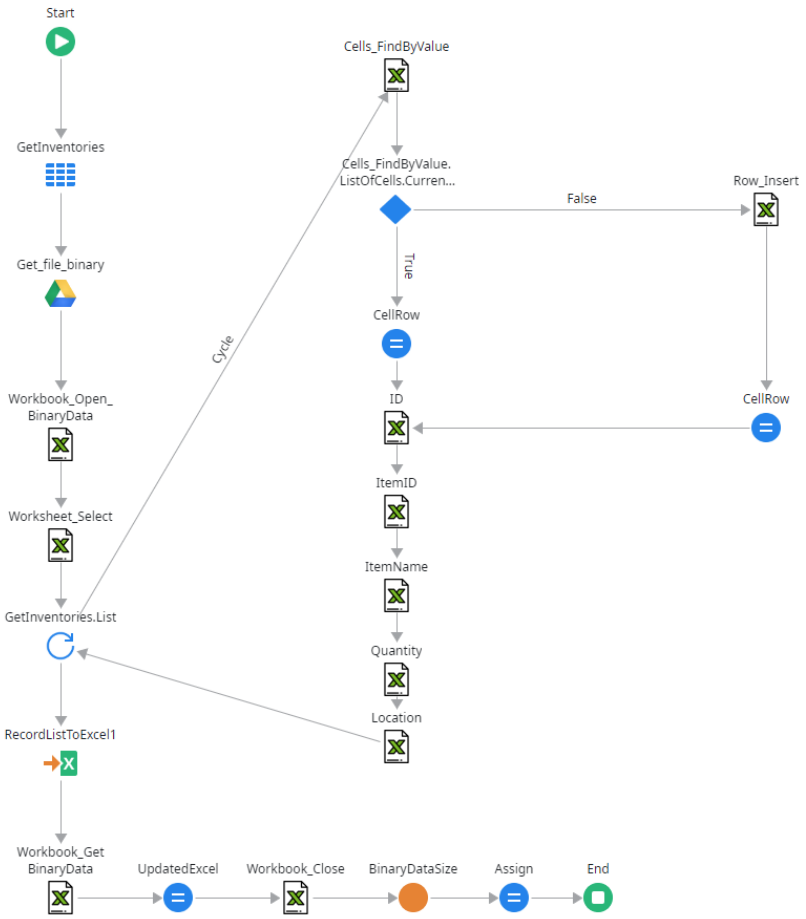


Figure 16. *'SaveToExcel' action in Case 1*

The connection to Google Drive is made using Google APIs, which involves two key connections: ‘Get_file_binary’ and ‘Patch_Files’. Both connections require the ID of the

Excel file and the authentication method. The 'Patch_Files' connection utilizes OAuth 2.0 for authentication, while the 'Get_file_binary' connection employs an API key.

4.2.2 Case 2

This case has four steps:

1. Get data from the database.
2. Get an exchange rate from API.
3. Create a new Excel file.
4. Send created Excel to Google Drive.

This case was more straightforward, as it involved creating a new file instead of updating an existing one. Since the updates had already been completed in Case 1, making this little different felt like a good option. Like Case 1, the 'Inventory' database resides within the OutSystems environment. The flow is not automated; it requires the user to press the 'Update Prices to Excel' button, as shown in Figure 17.

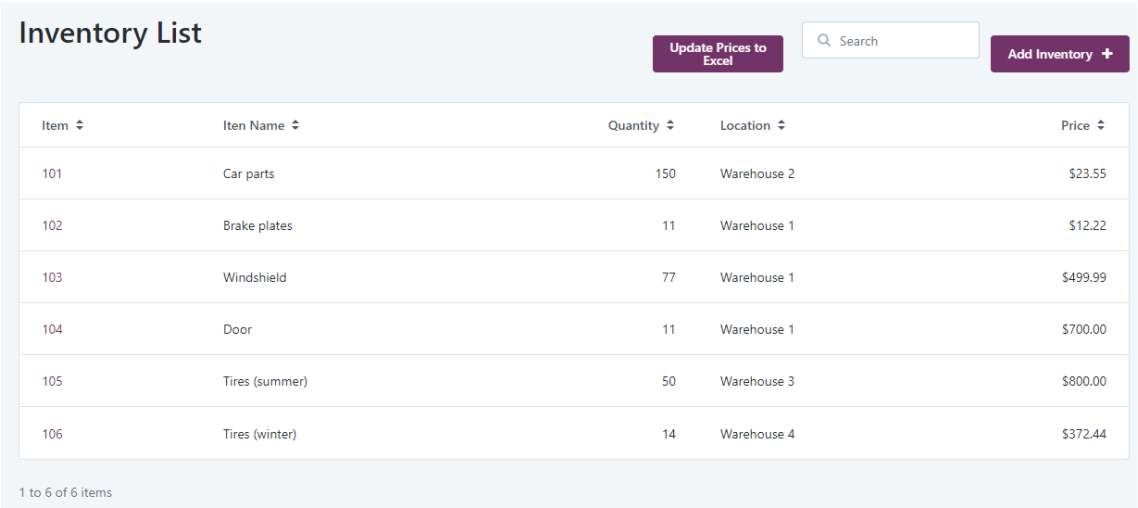


Figure 17. *User interface used in Case 2*

The flow operates similarly to Case 1. Upon pressing the button, the main flow initiates as shown in Figure 18. This main flow subsequently calls the ‘UploadUpdatedExcel’ action, which contains most of the logic for the process.

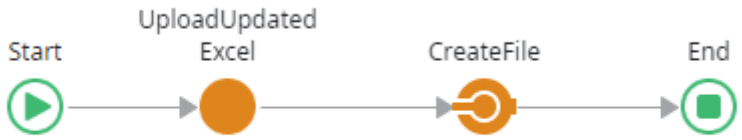


Figure 18. *Main flow used in Case 2*

The first step in this action involves retrieving all rows from the 'Inventory' database. Following this, the USD to EUR exchange rate is obtained from the ExchangeRate-API and stored in a variable. Next, the action iterates through each row in the 'Inventory' table, updating the price of each item to reflect the new value in euros. After all rows have been updated accordingly, control returns to the main flow, where a new Excel file is generated for Google Drive.

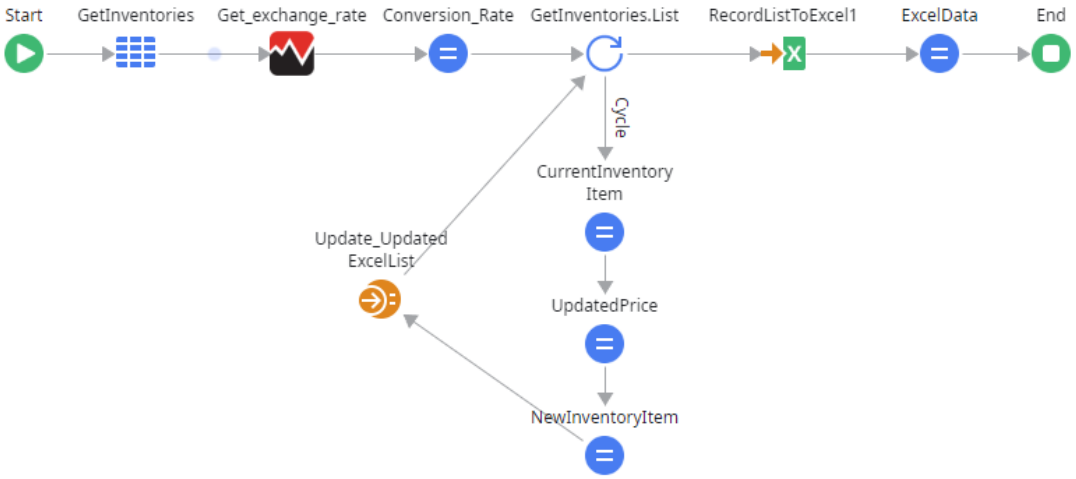


Figure 19. *Action used in Case 2*

4.2.3 Case 3

Case 3 is the simplest case, and it has four steps:

1. List files in SFTP server
2. Get one SFTP file from server
3. Save SFTP file to database
4. Move SFTP file to its destination folder

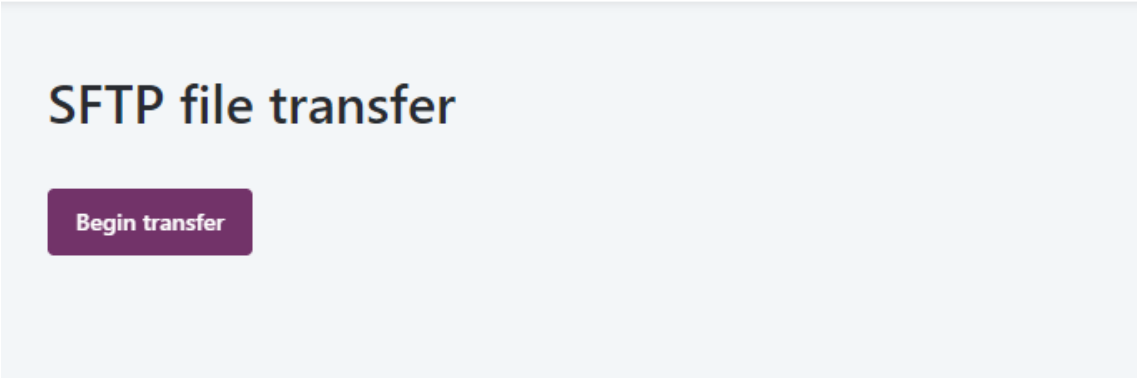


Figure 20. *User interface used in Case 3*

This case differs from the one implemented with Power Platform by including an additional feature for backing up files in the OutSystems scenario. Like the previous two cases, the main flow begins with a button click. The flow initiates by listing the files in the SFTP server's ‘source’ folder. Subsequently, it iterates through the list, fetching each SFTP file from the server. The file name and its content are then stored in the ‘Files’ database within the OutSystems environment. Finally, each SFTP file is uploaded to the ‘destination’ folder on the SFTP server. It's worth noting that the SFTP connection details remain consistent with those used in Case 3 of the Power Platform implementation.

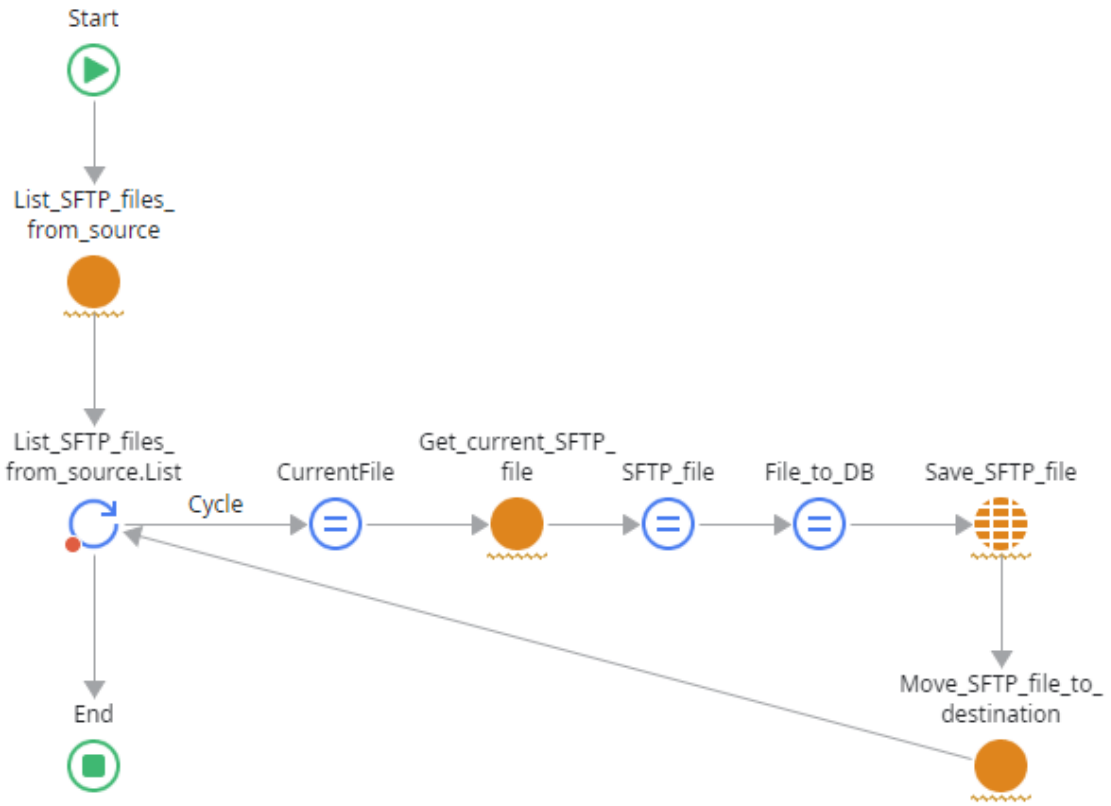


Figure 21. *Main flow used in Case 3*

4.2.4 Findings

Ease of use

Throughout the development of these cases, I found that the learnability of the OutSystems platform was good. Initially, it was challenging to grasp how various features functioned, but after completing Case 1, the next cases felt significantly easier to make. Importantly, all cases could be executed without extensive coding knowledge; however, having a foundational understanding of APIs, as well as how loops and functions operate, proved to be beneficial.

The learning curve for the platform appears reasonable. After navigating the initial complexities, the remainder of the tasks became much smoother and quicker to complete. OutSystems offers excellent documentation and a lot of training materials, making it accessible for users. Additionally, the community forums are invaluable, with numerous users actively asking and answering questions.

In my experience, the most challenging aspect to master within OutSystems was the distinction between the various types of actions. Some actions are restricted to client-

side use, while others can only be implemented within server actions. Gaining clarity on this logic may require some time and effort.

Integration capabilities

For these cases, OutSystems demonstrated strong integration capabilities. Setting up new REST API integrations was straightforward, and configuring the SFTP connector from Forge was also user-friendly. The platform offers a wide array of connectors for various integrations, including popular databases like MySQL and services such as Salesforce. Moreover, OutSystems supports real-time data synchronization through the implementation of a message queue setup, enhancing its functionality for dynamic data interactions.

Development time

The development took much longer than I was prepared. Case 1 took most of the time because of the difficulties with Excel. Development by case:

- Case 1: 3 hours
- Case 2: 1 hour
- Case 3: 30 minutes

The only difficulties were with Case 1 other two cases were easier to make.

Use case suitability

These cases can also be implemented using OutSystems. While the integrations were relatively straightforward, most of the platforms' capabilities on the market support such tasks. The SFTP integration case showcased the best compatibility with the OutSystems platform, whereas the other two cases did not perform as well. Additionally, there is a need for more efficient methods to handle Excel data, as it remains widely utilized in many organizations.

4.3 Mendix

4.3.1 Case 1

This case, which is much simpler than cases made with other platforms, has a few steps:

1. Get all rows from Mendix database
2. Update all rows in Excel file in Google Drive

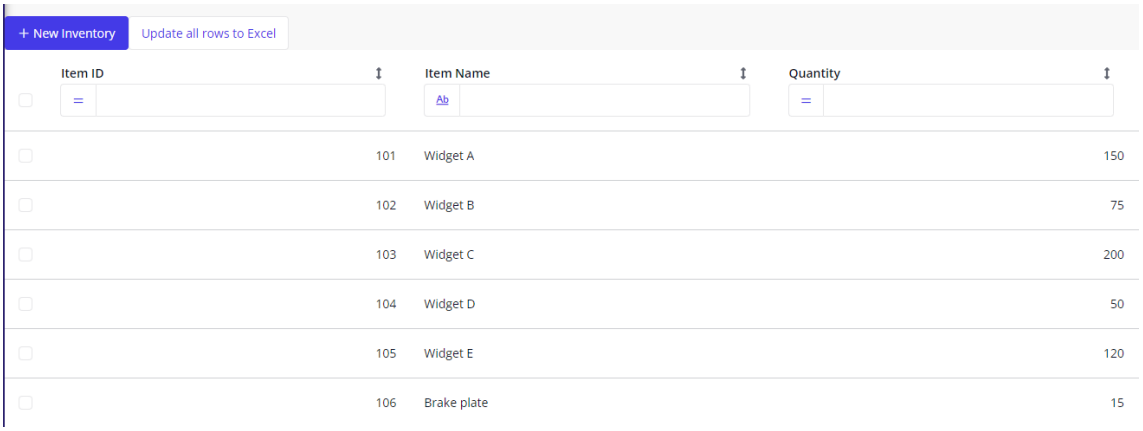


***Figure 22**. User interface used in Case 1*

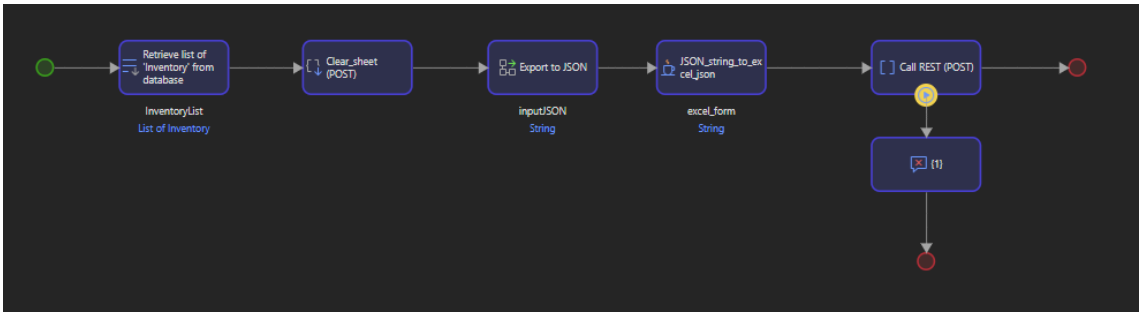


Figure 23. *Flow used in Case 1*

The flow is straightforward; however, it does not execute automatically. The user must manually press the ‘Update all rows to Excel’ button in the user interface to initiate the process. Once the button is pressed, all rows from the ‘Inventory’ database are retrieved, and the Excel sheet is cleared via a POST request. Following this, the ‘Inventory’ database is exported to a JSON string. Since Mendix lacks built-in functionality for handling JSON, a tool called ‘Export’ is utilized, which converts the database rows into a JSON string in this case.

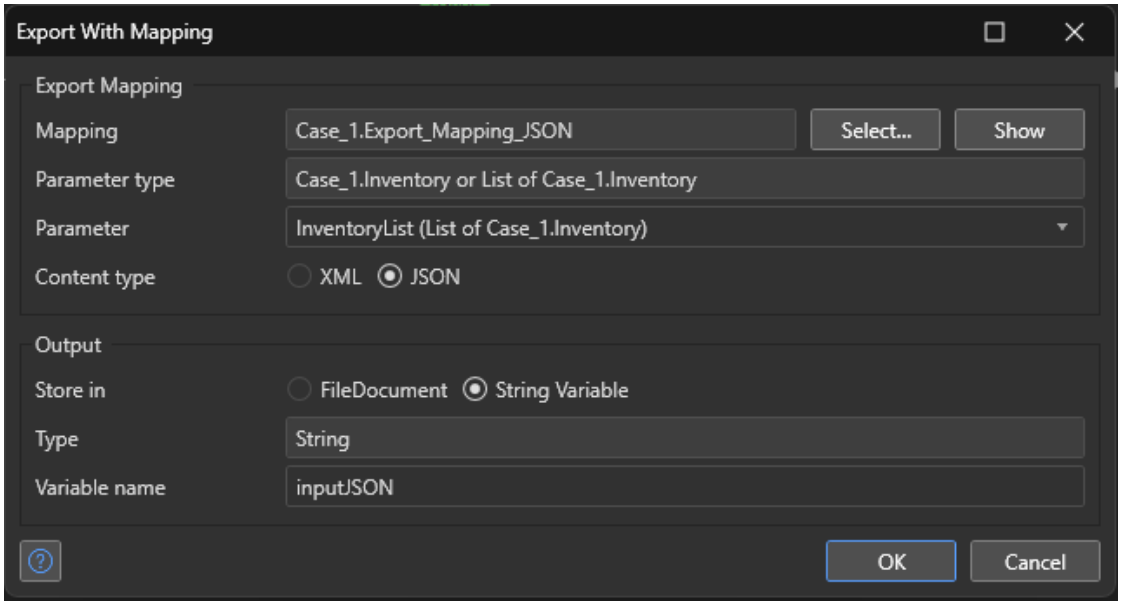


Figure 24. *Export mapping used in Case 1*

Next, a custom Java method, developed for this case and shown in Program 1, is called to construct the body for the POST request, which will update the rows in the Excel file. Once the request body is generated, a POST call is executed to send the updates to the Excel file.

```
public java.lang.String executeAction() throws Exception
{
    // BEGIN USER CODE
    try {
        // Create an ObjectMapper instance
        ObjectMapper objectMapper = new ObjectMapper();

        // Deserialize the input JSON string to a JsonNode
        JsonNode rootNode = objectMapper.readTree(inputJSON);

        // Create the output list that will hold the final JSON structure
        List<List<String>> outputList = new ArrayList<>();

        // Add header row
        List<String> header = List.of("Item ID", "Item Name", "Quantity", "Location");
        outputList.add(header);

        // Loop through the "values" array in the input JSON
        for (JsonNode itemNode : rootNode) {
            List<String> row = new ArrayList<>();
            row.add(itemNode.path("Item_ID").asText()); // Item_ID
            row.add(itemNode.path("Item_Name").asText()); // Item_Name
            row.add(itemNode.path("Quantity").asText()); // Quantity
            row.add(itemNode.path("Location").asText()); // Location

            // Add the row to the output list
            outputList.add(row);
     }

        // Serialize the output list back to JSON string
        String outputJSON = objectMapper.writeValueAsString(outputList);

        return outputJSON;
    } catch (Exception e) {
        throw new MendixRuntimeException("Error processing JSON: " + e.getMessage());
    }
    // END USER CODE
}

```

***Program 1**. Java method used in Case 1*

4.3.2 Case 2

This case is more complex and requires these steps:

1. Get Excel from Google Drive
2. Get all rows from database
3. Get an exchange rate from API
4. Update price in database

The complexity of this case comes from the challenges associated with handling JSON in Mendix, as the platform does not offer built-in methods for this purpose. The process begins in the user interface when the 'Update Prices from Excel' button is pressed.

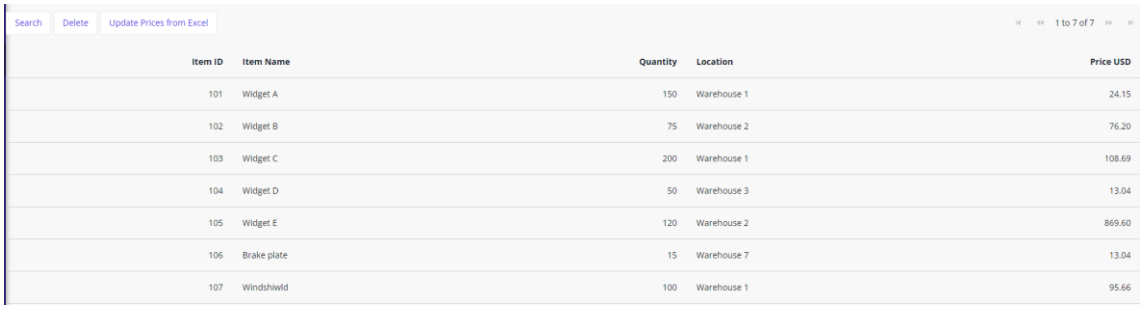


Figure 25. *User interface used in Case 2*

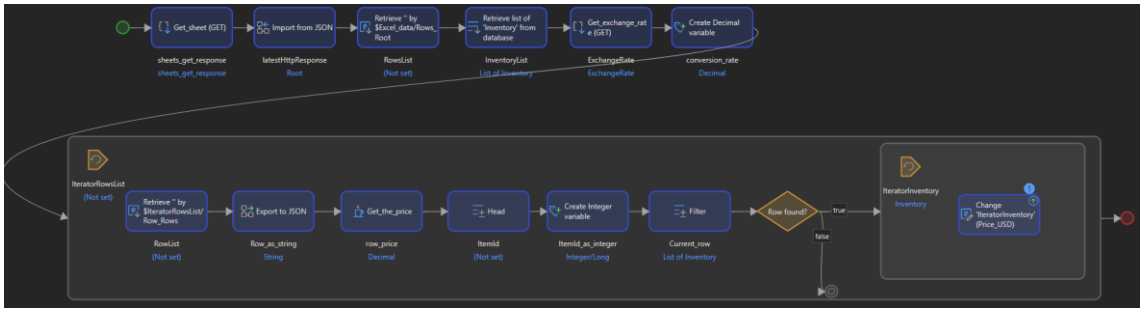


Figure 26. *Flow used in Case 2*

Once the button is pressed, the JSON data from the Excel sheet is fetched from Google Drive and imported into the domain model entities. The domain model visually represents the data utilized by the application, allowing for effective access to the JSON data in this case.

Next, all rows from the ‘Inventory’ database are retrieved, and the exchange rate is fetched and stored in the ‘conversion_rate’ variable. The process then iterates through the rows from Excel, where each row contains ‘ItemId’, ‘ItemName’, ‘Location’, ‘Quantity’, and ‘Price’. The ‘ItemId’ and ‘Price’ fields are crucial, as the ‘ItemId’ is used to identify the corresponding record in the database, while the ‘Price’ is needed to update the new price.

Each row is exported as a JSON string, which is then sent to a custom Java class to extract the price from the string and save it in the ‘row_price’ variable. This method is necessary because the Google Sheets API returns each row as a JSON array without key-value pairs.

```
    public java.math.BigDecimal executeAction() throws Exception
    {
         // BEGIN USER CODE

        try {
             // Remove brackets and split the string into parts
             String[] parts = row_as_string.replace("[", "").replace("]", "").re-
place("\"", "").split(",");

            // The price is at the last index
             String priceString = parts[parts.length - 2].trim() + "," +
parts[parts.length - 1].trim(); // Get the last item and trim spaces


             // Remove the euro sign and convert to BigDecimal
             priceString = priceString.replace("€", "").trim(); // Remove euro sign
             priceString = priceString.replace(",", ".").trim(); // Convert comma to
dot
             BigDecimal price = new BigDecimal(priceString); // Convert to BigDecimal

             return price; // Return the price
        } catch (Exception e) {
            throw new com.mendix.systemwideinterfaces.MendixRuntimeException("Error
parsing price: " + e.getMessage());
        }
        // END USER CODE
     }

```

***Program 2**. Java method used in Case 2*

```
@java.lang.Override
public java.math.BigDecimal executeAction() throws Exception
{
    // BEGIN USER CODE

    try {
        // Remove brackets and split the string into parts
        String[] parts = row_as_string.replace("[", "").replace("]", "").replace("\"", "").split(",");

        // The price is at the last index
        String priceString = parts[parts.length - 2].trim() + "," + parts[parts.length - 1].trim(); // Get the last item and trim spaces


        // Remove the euro sign and convert to BigDecimal
        priceString = priceString.replace("€", "").trim(); // Remove euro sign
        priceString = priceString.replace(",", ".").trim(); // Convert comma to dot
        BigDecimal price = new BigDecimal(priceString); // Convert to BigDecimal

        return price; // Return the price
    } catch (Exception e) {
        throw new com.mendix.systemwideinterfaces.MendixRuntimeException("Error parsing price: " + e.getMessage());
    }
    // END USER CODE
}
```

Figure 27. *Java class used in Case 2*

After fetching the row, the row ID is retrieved using the ‘Head’ operation, which extracts the first object from the row. Following this, a ‘Filter’ operation is employed to locate the corresponding record from the ‘Inventory’. If the row is found, it is updated using the ‘Change Object’ operation. This operation specifically updates only the price, with the necessary logic encapsulated within the operation itself.

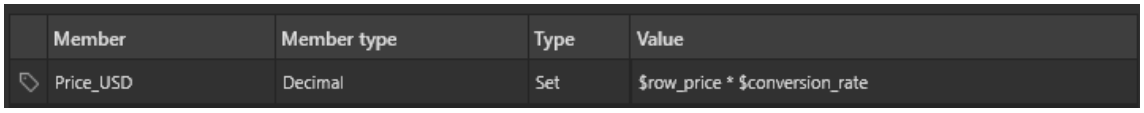


Figure 28. *'Change Object' operation price update*

4.3.3 Case 3

This case differs from the planned case. This case has these steps:

1. Create connection to SFTP server.
2. List files from SFTP server.
3. Download file/files.

Mendix does not natively support SFTP transactions. In this case, the SFTP module was obtained from the Mendix Marketplace, which serves as an app store for users to download modules or libraries created by other users. The functionality provided by the SFTP module relies heavily on several custom Java classes that have been developed to enable SFTP capabilities within the Mendix environment.

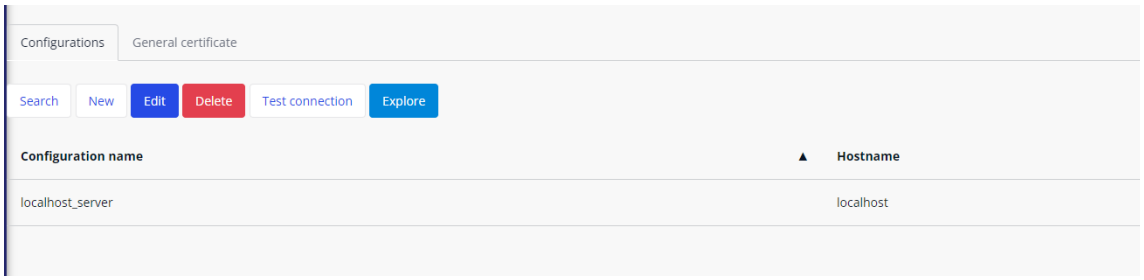


Figure 29. *User interface used in Case 3*

To configure SFTP, begin by clicking the 'New' button. The configuration settings are like those used in previous cases. This time, the host is set to 'localhost' since Mendix is running locally.

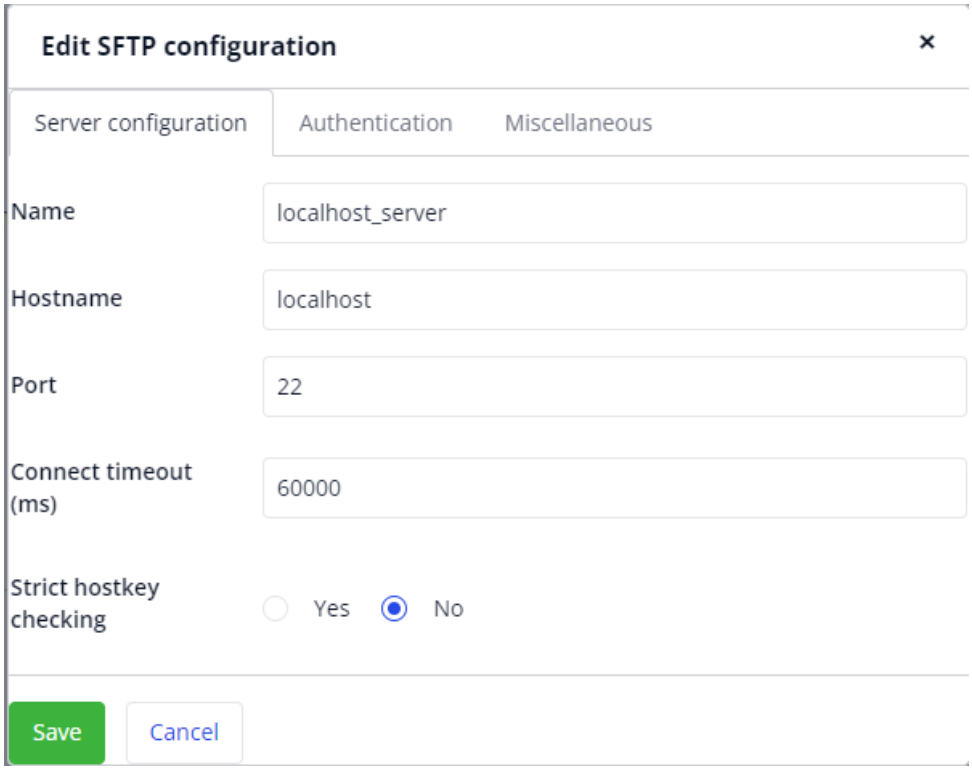


Figure 30. *Creating SFTP configuration for Case 3*

Once the configuration is complete, clicking the 'Explore' button opens the user interface. Here, users can view, download, delete, move, rename, and inspect the contents of files.

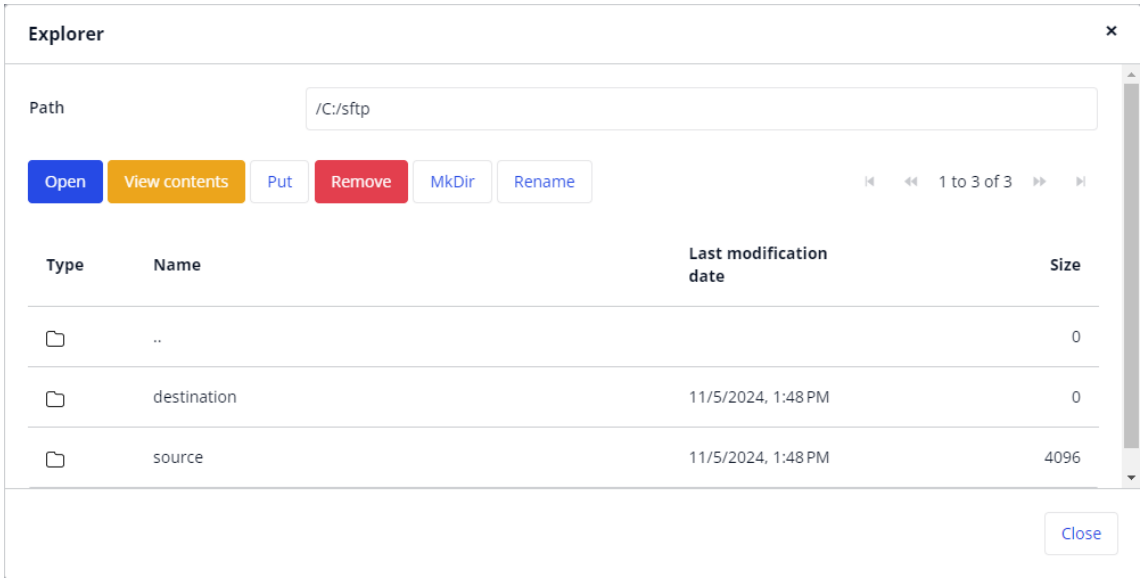


Figure 31. *Explore user interface used in Case 3*

While it would be possible to develop a module that could facilitate file transfers between SFTP servers, this falls outside the scope of this thesis. All the Java classes utilized in this case are part of the module downloaded from the Mendix Marketplace. Explaining

how these Java classes and flows interact, and working together would require extensive detail, which would take up too much space in this thesis.

4.3.4 Findings

Ease of use

Mendix proved to be the most challenging of the LCPs to learn that were utilized in these cases. It took some time to grasp the processes involved in exporting and importing JSON, including understanding the reasoning behind these methods. The need for custom Java classes was a must in several instances.

The user interface, however, is intuitive, allowing for straightforward configuration of views. For instance, linking the 'Inventory' database and displaying it in the user interface can be accomplished with just a few clicks. Additionally, components can be easily dragged and dropped into the user interface and workflows.

Despite its user-friendly interface, the learning curve for Mendix is quite steep. While utilizing predefined connectors is relatively easy, the integration of multiple custom Java classes can pose difficulties for non-technical users. Understanding the import process, the types of data generated, and how to access that data can be time-consuming.

Integration capabilities

The integration capabilities of Mendix for these cases were sufficient. It is straightforward to create REST requests to APIs, and it also supports connections to several databases, such as MySQL and Snowflake. However, establishing SFTP connections is challenging and requires either custom Java classes or modules available in the Marketplace. On a positive note, the integration capabilities overall are enhanced by the variety of integration solutions developed by other users in the Marketplace.

Use case suitability

Apart from Case 3, the use case suitability was satisfactory. While the cases required some time to complete, the results were positive. The main drawback of using Mendix was the necessity for Java custom classes. However, implementing these Java classes into the flow was straightforward.

Development time

The development times for Mendix were:

1. Case 1: 1 hour 30 minutes
2. Case 2: 3 hours
3. Case 3: 1 hour

The learning curve was quite steep, and building the flows took significantly more time than expected. I encountered multiple errors, but I was able to resolve them using documentation and assistance from forums. Additionally, the last case would have taken even longer if there had been an option to send the SFTP file to another server.

5 Results

This section presents the results of the integration projects undertaken using Microsoft Power Platform, OutSystems, and Mendix. Each case study's outcomes will be analyzed to identify the practical benefits and challenges of using low-code platforms for integration development. The results will be organized according to the key comparison areas established earlier: ease of use, integration capabilities, development time, and use case suitability. Table 1 shows a comparison of the key areas. The first column shows the key area, in the second column Power Platform, in the third column OutSystems and in the fourth Mendix column findings per platform.

Key area	Power Platform	OutSystems	Mendix
Ease of use	User-friendly interface. Straightforward for both technical and non-technical users, although learning Power Fx presents challenges.	Good learnability. Intuitive UI, but challenges with poorly documented user made connectors.	Steeper learning curve. Requires understanding of JSON and custom Java classes.
Integration capabilities	Excellent integration with pre-built connectors for APIs and databases; seamless real-time data synchronization.	Strong integration capabilities; user-friendly REST API setups, but challenges with Excel handling.	Sufficient capabilities; good API integrations, but reliance on custom Java classes for SFTP.
Development time	Initial development time longer due to learning curve; Case 1 took ~2 hours, Case 2 ~1.5 hours, Case 3 ~30 minutes.	Longer than expected; Case 1 took ~3 hours due to Excel issues, Cases 2 and 3 took ~1 hour and 30 minutes.	Notably longer; Case 1 took ~1.5 hours, Case 2 ~3 hours, Case 3 ~1 hour, largely due to JSON complexities.
Use case suitability	Highly suitable for rapid integration projects. Effective for cases done in this thesis	Suitable for various integrations but inconsistent. Worked well with SFTP	Mixed results. Worked well with Excel cases, but reliance on Java in SFTP limits in file transfer.

Table 1. *Comparison of the LCPs*

In terms of ease of use, Microsoft Power Platform stands out with its user-friendly interface, which accommodates both technical and non-technical users, despite the learning curve associated with its programming language, Power Fx. In contrast, while OutSystems provides an intuitive user interface, the user-made Excel connector used in these cases was poorly documented. Connectors provided by OutSystems are well documented. Mendix offers an intuitive UI for certain tasks but presents a steeper learning curve due to the necessity of understanding JSON and custom Java classes. Not all the integrations need custom Java classes, but for these cases custom Java classes were necessity.

When it comes to integration capabilities, Power Platform is great by providing seamless connectivity through pre-built connectors and facilitating real-time data synchronization. OutSystems also demonstrates strong integration capabilities, particularly with REST APIs; however, it encounters difficulties in handling Excel data. Mendix offers sufficient integration options but relies heavily on custom solutions for specific functionalities, such as SFTP connections.

Development times varied significantly across the platforms. Although Power Platform's initial learning curve led to longer setup times for the first case, subsequent cases were completed more quickly. OutSystems required more time than anticipated due to Excel-related challenges, while Mendix had the most extended development times owing to complexities in handling data formats and the need for custom coding.

The suitability of each platform for specific use cases also showed variation. Power Platform proved highly effective for straightforward integration tasks, while OutSystems showed mixed results depending on the structure of the integration project. Mendix demonstrated potential for handling complex scenarios, but its reliance on custom coding limited its effectiveness in SFTP integrations.

Overall, these findings reflect the practical benefits and challenges encountered when utilizing LCPs for integration development, highlighting their respective strengths and weaknesses in real-world applications.

6 Summary

The thesis investigates the increasing adoption of LCPs in software development, particularly their potential to streamline application and integration processes. Although LCPs promise significant benefits, including reduced development time and increased empowerment for citizen developers, their effectiveness in real-world integration projects has not been thoroughly explored. This research aims to bridge the gap between theoretical claims and practical applications of LCPs in integration development.

A mixed-methods approach was used, combining a literature review with empirical case studies focused on three leading low-code platforms: Microsoft Power Platform, OutSystems, and Mendix. The literature review established the theoretical foundations of LCPs, identified gaps in existing research, and outlined the conceptual framework that guided the study. Case studies were conducted to evaluate real-world integration scenarios, assessing the capabilities and challenges of each platform in practical applications.

The findings reveal a strong alignment between the theoretical benefits of LCPs—such as increased speed and productivity—and practical experiences from case studies. Notably, Microsoft Power Platform demonstrated strengths in ease of use and integration capabilities. However, challenges persist. OutSystems faced issues with poorly documented user-created connectors, particularly in integrating Excel data, while Mendix's reliance on custom Java classes complicated certain integration tasks.

The analysis revealed significant variability in development times across platforms, influenced by factors such as learning curves and the complexity of integration tasks. Power Platform required a longer initial setup but allowed for quicker subsequent tasks, whereas Mendix demonstrated the longest development times, primarily due to its data handling requirements. Each platform showed varying levels of suitability for specific use cases. Power Platform was great in straightforward integrations, while OutSystems and Mendix produced mixed results depending on the specific integration requirements.

The thesis discusses the implications of these findings for organizations considering the adoption of LCPs for integration projects. It highlights the importance of a balanced understanding of both the benefits and limitations of LCPs, highlighting that while they can enhance agility and reduce development times, careful consideration is essential for effective implementation.

7 Conclusion

This thesis explored the practical implications of using LCPs—specifically Microsoft Power Platform, OutSystems, and Mendix—for integration development. Through a combination of literature review and empirical case studies, this thesis aimed to bridge the gap between theoretical claims regarding LCPs and their practical performance in real-world scenarios.

The findings highlight several key benefits and challenges associated with the use of LCPs in integration development. The first research question addressed the comparison between the theoretical frameworks and documented benefits of LCPs and the outcomes observed in practical integration projects. Overall, the results indicate a strong alignment between the theoretical benefits outlined in existing literature—such as increased development speed and enhanced productivity—and the practical experience observed in case studies. Microsoft Power Platform demonstrated significant strengths in ease of use and integration capabilities, supporting the idea that LCPs can empower both technical and non-technical users to achieve efficient integration solutions.

However, the findings also revealed that practical challenges persist. While Power Platform was great in user-friendliness and real-time data synchronization, both OutSystems and Mendix faced problems that impacted their overall effectiveness. OutSystems struggled with poorly documented user-created connectors, particularly in handling Excel data, while Mendix's reliance on custom Java classes created additional complexities, especially for users with limited programming backgrounds. These challenges illustrate a difference between the expectations set by theoretical literature and the realities faced during implementation.

The second research question explored the specific advantages and challenges organizations encounter when implementing LCPs for integration development, particularly across various use cases. The results from the case studies revealed that while LCPs generally enable rapid development and support diverse integration scenarios, their suitability can vary significantly based on the complexity of the integration tasks. Power Platform was highly effective for straightforward integrations, while OutSystems and Mendix showed mixed results depending on the specific requirements and existing organizational infrastructure.

Moreover, the varying development times across the platforms highlighted the impact of learning curves and prior experience with low-code environments. The initial setup and

configuration stages were often time-consuming, particularly for users unfamiliar with the platforms. This finding underscores the importance of providing sufficient training and resources to facilitate easier transitions to LCD.

In summary, this thesis contributes to the growing body of knowledge on LCPs by offering empirical insights into their application for integration development. While the findings support the potential of LCPs to enhance organizational agility and reduce development time, they also emphasize the need for careful consideration of their limitations. Future research should focus on longitudinal studies to assess the long-term impacts of LCP adoption and further investigate the specific contexts in which these platforms can provide the greatest value. As organizations continue to navigate the complexities of digital transformation, understanding both the capabilities and constraints of LCPs will be crucial for making good decisions in their application development strategies.

References

Al Alamin, M. A., Malakar, S., Uddin, G., Afroz, S., Haider, T. B., Iqbal, A. (2021). “An Empirical Study of Developer Discussions on Low-Code Software Development Challenges”. IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), 46-57. *https://doi.org/10.1109/MSR52588.2021.00018*

ALazzawi, A., Yas, Q.M., Rahmatullah, B. (2023). A Comprehensive Review of Software Development Life Cycle methodologies: Pros, Cons, and Future Directions. *https://doi.org/10.52866/%20ijcsm.2023.04.04.014*

Beaton, C. (2022, December 9). Requirements Gathering Challenges & Overruns. Requiment. *https://www.requiment.com/why-requirements-gathering-challenges-can-lead-to-overruns/*

Bock, A.C., Ulrich, F. (2021). “Low-Code Platform”. Bus Inf Syst Eng, 63, 733–740. *https://doi.org/10.1007/s12599-021-00726-8*

Campbell-Kelly, M. (2004). Computer: a history of the information machine. *https://doi.org/10.4324/9781003263272*

Di Ruscio, D., Kolovos, D., de Lara, J., Pierantonio, A., Tisi, M., & Wimmer, M. (2022). Low-code development and model-driven engineering: Two sides of the same coin? Software and Systems Modeling, 21(2), 437–446. *https://doi.org/10.1007/s10270-021-00970-2*

Di Sipio, C., Di Ruscio, D., & Nguyen, P. (2020, September 18). Democratizing the Development of Recommender Systems by means of Low-code Platforms. *https://doi.org/10.1145/3417990.3420202*

Dorin, M. & Montenegro, S. (2021). A life cycle for creating an uncomplicated software. CIIS Ulima Congreso Internacional De Ingeniería De Sistemas, 129-140. *https://doi.org/10.26439/ciis2018.5485*

Dooley, J. F. (2017). Software Development, Design and Coding: With Patterns, Debugging, Unit Testing, and Refactoring. Apress. *https://doi.org/10.1007/978-1-4842-3153-1_1*

Elshan, E., Ebel, P., Söllner, M., & Leimeister, J. M. (2023a). Leveraging Low Code Development of Smart Personal Assistants: An Integrated Design Approach with the SPADE Method. Journal of Management Information Systems, 40(1), 96–129. *https://doi.org/10.1080/07421222.2023.2172776*

Elshan, E., Germann, D., Dickhaut, E., & Li, M. (2023b). FASTER, CHEAPER, BETTER? ANALYZING HOW LOW CODE DEVELOPMENT PLATFORMS DRIVE BOTTOM-UP INNOVATION. ECIS 2023 Research-in-Progress Papers, 82. *https://aisel.aisnet.org/ecis2023_rip/82*

Forrester Research (2024, January 29). The Low-Code Market Could Approach $50 Billion By 2028. Forrester. *https://www.forrester.com/blogs/the-low-code-market-could-approach-50-billion-by-2028/*

Freedman, R. (2018). What Is Agile? O'Reilly Media, Inc.

Futcher, L. & Solms, R. v. (2007). Secsdm: a model for integrating security into the software development life cycle. Fifth World Conference on Information Security Education, 41-48. *https://doi.org/10.1007/978-0-387-73269-5_6*

Funk, D. (2023). Creating a Low-Code Business Process Execution Platform With Python, BPMN, and DMN. IEEE Software, 40(1), 9–17. IEEE Software. *https://doi.org/10.1109/MS.2022.3212033*

Gartner. (2019). Gartner Magic Quadrant for Enterprise Low-Code Application Platforms.

Gulledge, T. (2006). What is integration? Industrial Management & Data Systems, 106(1), 5–20. *https://doi.org/10.1108/02635570610640979*

Heller, M. (2021, August 11). What is low-code development? Faster app creation. InfoWorld.com, NA. *https://link-gale-com.libproxy.tuni.fi/apps/doc/A671500437/GBIB?u=tampere&sid=bookmark-GBIB&xid=a3df84a6*

Iosup, A., Ostermann, S., Yigitbasi, M. N., Prodan, R., Fahringer, T., & Epema, D. (2011). Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. IEEE Transactions on Parallel and Distributed Systems, 22(6), 931–945. *https://doi.org/10.1109/TPDS.2011.66*

Kumar, N., Zadgaonkar, A. S., & Shukla, A. (2013). Evolving a new software development life cycle model SDLC-2013 with client satisfaction. International Journal of Soft Computing and Engineering (IJSCE), 3(1), 2231-2307. *https://www.ijsce.org/wp-content/uploads/papers/v3i1/A1355033113.pdf*

Laanti, M., Similä, J., & Abrahamsson, P. (2013). Definitions of Agile Software Development and Agility. In F. McCaffery, R. V. O'Connor, & R. Messnarz (Eds.), Systems, Software and Services Process Improvement (pp. 247–258). Springer. *https://doi.org/10.1007/978-3-642-39179-8_22*

Lane, S., O'Raghallaigh, P., & Sammon, D. (2016). Requirements gathering: The journey. Journal of Decision Systems. *https://www.tandfonline.com/doi/abs/10.1080/12460125.2016.1187390*

LCPs. (2024). Low-Code Platform Features. Mendix. *https://www.mendix.com/platform/*

Luo, Y., Liang, P., Wang, C., Shahin, M., Zhan, J. (2021). "Characteristics and Challenges of Low-Code Development: The Practitioners' Perspective". Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, 12, 1-11. *https://doi.org/10.1145/3475716.3475782*

Maia, A. I. F. (2022). Low-Code Data Model Designer for Manufacturing Execution Systems. *https://repositorio-aberto.up.pt/handle/10216/143447*

Mendix. (2021). The State of Low-Code 2021: A Look Back, The Trends That Matter & The Future Ahead. *https://www.mendix.com/resources/the-state-of-low-code-report/*

Microsoft. (2024a, September 23). Introduction to Microsoft Power Platform for developers—Power Platform. *https://learn.microsoft.com/en-us/power-platform/developer/get-started*

Microsoft. (2024b, March 22). Microsoft Power Fx overview—Power Platform. *https://learn.microsoft.com/en-us/power-platform/power-fx/overview*

Mohino, J., Higuera, J., Higuera, J., & Sicilia, J. (2019). The application of a new secure software development life cycle (s-sdlc) with agile methodologies. Electronics, 8(11), 1218. *https://doi.org/10.3390/electronics8111218*

Naqvi, S. A. A., Zimmer, M., Rehan, S., & Drews, P. (2023). Understanding the Socio-Technical Aspects of Low-Code Adoption for Software Development. *https://www.researchgate.net/publication/370156464_Understanding_the_Socio-Technical_Aspects_of_Low-Code_Adoption_for_Software_Development*

Oteyo, I. N., Pupo, A. L. S., Zaman, J., Kimani, S., De Meuter, W., & Boix, E. G. (2021). Building Smart Agriculture Applications Using Low-Code Tools: The Case for DisCoPar. 2021 IEEE AFRICON, 1–6. *https://doi.org/10.1109/AFRICON51333.2021.9570936*

OutSystems. (2024, October 3). Discover the fundamentals of low-code applications. *https://www.outsystems.com/tech-hub/low-code/apps/*

Palomares, C., Franch, X., Quer, C., Chatzipetrou, P., López, L., & Gorschek, T. (2021). The State-of-Practice in Requirements Elicitation: An Extended Interview Study at 12 Companies. Requirements Engineering, 26(2), 273–299. *https://doi.org/10.1007/s00766-020-00345-x*

Prinz, N., Rentrop, C., & Huber, M. (2021). Low-Code Development Platforms – A Literature Review.

Raghavendran, K. R., & Elragal, A. (2023). Low-Code Machine Learning Platforms: A Fastlane to Digitalization. Informatics, 10(2), Article 2. *https://doi.org/10.3390/informatics10020050*

Rokis, K., Kirikova, M. (2023). "Exploring Low-Code Development: A Comprehensive Literature Review". Complex Systems Informatics and Modeling Quarterly, 68-86. *http://dx.doi.org/10.7250/csimq.2023-36.04*

Saunders, M. N. K. (2019). Research Methods for Business Students. Pearson Education.

Setiawan, M., & Sibarani, A. (2024). The Design of Web Service-based Minimum Competency Assessment Application with the REST Method in Senior High Schools. International Journal of Advanced Science Computing and Engineering, 6, 1–6. *https://doi.org/10.62527/ijasce.6.1.150*

Silhavy, R., Silhavy, P., & Prokopová, Z. (2011). Requirements gathering methods in system engineering.

Sommerville, I. (2011). Software processes. Software Engineering, 2011, 30-31.

Tisi, M., Mottu, J.-M., Kolovos, D. S., de Lara, J., Guerra, E. M., Di Ruscio, D., Pierantonio, A., & Wimmer, M. (2019, July). Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms. STAF 2019 Co-Located Events Joint Proceedings: 1st Junior Researcher Community Event, 2nd International Workshop on Model-Driven Engineering for Design-Runtime Interaction in Complex Systems, and 1st Research Project Showcase Workshop Co-Located with Software Technologies: Applications and Foundations (STAF 2019). *https://hal.science/hal-02363416*

Totterdale, R.L. (2018). Case study: The Utilization of Low-code Development Technology to Support Research Data Collection. Issues in Information Systems. *https://doi.org/10.48009/2_iis_2018_132-139*.

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

What are the Top 8 Features of Low-Code Platform? | Quixy. (2023, November 21). *https://quixy.com/blog/key-features-of-low-code-platforms/*

Waszkowski, R. (2019). “Low-code platform for automating business processes in manufacturing”. IFAC-PapersOnLine, 52(10), 376-381. *https://doi.org/10.1016/j.ifacol.2019.10.060*

Yam, T., Liu, S. (2023). The Role of Citizen Developers in Low-Code Application Development: A Systematic Literature Review. Information Systems Frontiers. https://doi.org/10.1007/s10796-023-10480-5