

Miska Merikukka

ENTERPRISE APPLICATION INTEGRATION WITH REST APIS

Master of Science Thesis
Faculty of Engineering and Natural Sciences
Examiner: Professor Kari Systä
Examiner: University Lecturer Timo Mäkinen
April 2021

ABSTRACT

Miska Merikukka: Enterprise application integration with REST APIs
Master of Science Thesis
Tampere University
Degree Programme in Management and Information Technology, MSc (Tech)
April 2021

Consulting and engineering business is extremely project-orientated, and its design tasks are conducted by developing engineering information from different source data and specifications and by iterating design. Project success depends significantly on the project organization's ability to manage source data, their change and the work related to them. In addition to talented personnel, robust processes, powerful software tools can enable effective management of source data. In the target company, the tasks and source data were previously managed in two different systems, there were no true common practices and there was no proper way to exchange information between the systems.

This thesis aimed to produce a proof of concept where two systems were integrated into single system where source data can be managed. The proof of concept was a web application that communicated with the two systems and provided a user interface (UI) by extending the UI of Jira Cloud. All elements of the application have done with JavaScript on top to Node.js framework while relying on libraries such as React and Express.

With the produced application and integration, the users can easily define the source data needed for specific tasks and the documents produced by the task. This provides a robust way to systematically document source data and manage it.

Keywords: REST, API, EAI, Node.js, Express.js, integration, add-on, middleware

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

TIIVISTELMÄ

Miska Merikukka: Järjestelmäintegraation toteuttaminen käyttäen REST-rajapintoja
Diplomityö
Tampereen yliopisto
Johtamisen ja tietotekniikan koulutusohjelma
Huhtikuu 2021

Konsultointi ja suunnitteluliiketoiminta on erittäin projektiorientoitunutta ja työtehtävät toteutetaan suunnittelumateriaalia kehittäen lukuisten lähtötietojen ja suunnitteluiteraatioiden pohjalta. Menestyminen projekteissa riippuu merkittävästi projektiorganisaation kyvystä hallita käytettyjä lähtötietoja, niistä johtuvia muutoksia ja niihin liittyviä töitä. Osaavan henkilökunnan ja toimivan prosessin lisäksi tehokkaat tietotekniset työkalut mahdollistavat tehokkaan lähtötietojen hallinnan. Aiemmin työtä ja lähtötietoja hallittiin kohdeyrityksessä kahdessa eri järjestelmässä, eikä yhteistä toimintatapaa ollut, eikä tietoja voitu tehokkaasti vaihtaa järjestelmien välillä.

Tämä työ pyrki luomaan integraation toteuttamiskelpoisuutta esittelevän yksinkertaistetun toteutuksen, jossa kaksi järjestelmää integroidaan yhdeksi kokonaisuudeksi. Se toteutettiin rakentamalla itsenäinen verkkopohjainen sovellus, joka kommunikoi integroitavien järjestelmien kanssa ja jonka käyttöliittymä on toteutettu laajentamalla Jira Cloud -sovelluksen käyttöliittymää. Kaikki sovelluksen osat on tehty JavaScript-kielellä Node.js viitekehykselle, ja se käyttää muun muassa React ja Express -kirjastoja.

Sovelluksen ja integraation avulla käyttäjät voivat määrittellä työtehtävien vaatimat lähtötiedot sekä toimitettavat dokumentit yksittäisiin työtehtäviin, jolloin lähtötietojen hallinta helpottuu ja sen dokumentointi systematisoituu.

Avainsanat: REST, API, EAI, Node.js, Express.js, integraatio, add-on, middleware

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

PREFACE

This thesis consisted of design and implementation of an integration and management tool for a Finnish consulting and engineering company. The thesis was started in September 2020 and was finished in April 2021. I would like to thank my employer for giving me the opportunity to find solutions for a significant issue in our daily work as a part of my thesis. I would also like to thank my thesis supervisor Kari Systä for his advises on the documentation process. Biggest thanks go to my wife Krista and our daughter for cheering me up and pushing me to accomplish this task.

In Turku, Finland on April 14th 2021

Miska Merikukka

CONTENT

1.	INTRODUCTION.....	1
2.	BACKGROUND	2
2.1	Problem identification and motivation.....	2
2.2	Engineering design process	3
2.3	Existing tools and process.....	4
2.4	Objectives.....	6
2.5	Enterprise Application Integration	7
2.5.1	Architecture patterns	8
2.5.2	RESTful services	10
2.5.3	Building user interfaces with React	11
2.5.4	Implementation methods of EAI	11
2.5.5	Commercial integration platforms.....	13
3.	METHODOLOGY.....	15
4.	RESULTS.....	17
4.1	System requirements	17
4.2	Add-on for Jira Cloud	22
4.2.1	Architecture	23
4.2.2	Application components and structure	24
4.2.3	User interface	27
4.2.4	Functionalities.....	31
4.2.5	Backend and business logic.....	34
4.2.6	Security.....	37
4.2.7	Development environment and installation.....	38
5.	EVALUATION AND DISCUSSION	39
5.1	Requirements	39
5.2	Development process.....	40
5.3	Future development	41
6.	CONCLUSION	42
	REFERENCES	43

LIST OF SYMBOLS AND ABBREVIATIONS

API	Application Programming Interface
AR	Action Research
ASM	Applied Signposting Model
CAD	Computer-aided Design
COM	Component Object Model
COTS	Components Off-the-Shelf
CRM	Customer Relationship Management
CRM	Customer Relationship Management
DMS	Document Management System
DS	Design Science
DSM	Design Structure Matrix
DX	Digital Transformation
EAI	Enterprise Application Integration
EC	Engineering Change
ECM	Engineering Change Management
EDA	Event-driven Architecture
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IM	Information Management
iPaaS	Integration Platform as a Service
JSON	Javascript object notation
JWT	JSON Web Token
ODBC	Open Database Connectivity
OSS	Open-source Software
PoC	Proof-of-Concept
REST	Representational State Transfer
ROI	Return on Investment
SaaS	Software as a Service
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TMS	Task Management System
UI	User Interface
XML	eXtensible Markup Language

1. INTRODUCTION

Arguably all contemporary businesses can be considered as IT companies as they use their enterprise applications as an integral part of their business processes. Companies today often choose “best-of-breed” software to fit their specific business needs in different functions [1]. While this can provide better tools and, it also presents the need to integrate that heterogeneous set of applications to work as a whole to enhance and increase competitive advantage [2], [3]. Today, many companies rely heavily on *software-as-a service* (SaaS) applications in addition to traditional on-premise and legacy systems [1]. That set of applications is created with different technologies and hold data in separate databases, different platforms, both in and out of bounds of the enterprise [4].

This study aims to solve a particular business and project management problem by extending the capabilities of one SaaS application by integrating it with an on-premise application. By data integration and added functionality, a particular business processes can be handled in a superior way, thus lowering costs caused by rework, poor information management and communication, while making daily work easier and more convenient.

The rest of the study is organized as follows: section 2.1 defines the problem and elaborates on the motivation for the study, sections 2.2 and 2.3 introduce the context in which the study will focus on and section 2.4 formulates the objective for the study. Section 2.5 discusses enterprise application integration purposes, patterns, architectures and solutions. The used design science methodology is discussed in chapter 3. Chapter 4 discusses the implementation of the design and execution of the add-on. The system requirements defined during the study are presented in section 4.1, discussion of chosen technologies and architecture and the description of the created solution is found in section 4.2. Chapter 5 presents the analysis for the study and the conclusion is presented in chapter 6.

2. BACKGROUND

This chapter introduces the business problem and why it would need to be solved. Chapter introduces the objective of the thesis and elaborates on the environment in which the problem lies in. It also discusses Enterprise application integration in general and its technical aspects.

2.1 Problem identification and motivation

This thesis is commissioned to a multi-discipline engineering and consulting company working in wide variety of fields. Engineering activities rely heavily on specialized software which is used to conduct data-driven design where the iterative nature of the design process, detailed specifications and thousands of different individual tasks require lot of effort in planning and orchestration of the process. Consulting and engineering activities are heavily impacted by digital transformation (DX), which is argued to be mandatory for all businesses to survive [5]. DX is a complex, and extensive concept, and is a lot more than automation of manual tasks [5]. However, automation and process digitalization are fields that can have huge impacts to many activities. Digitally encoding business processes can improve significantly companies' capabilities and productivity by leveraging automation, system integration and more efficient communication. Information management (IM) is a field where digitalization and intelligent solutions offer lots of potential for improvement. Within IM, task management and in particular *engineering change management* (ECM) can benefit significantly from process encoding and automation [6]. Thus, studying potential improvements in ECM was chosen as one of the DX projects at this time in the target company. Engineering changes (EC) can be understood as "*modifications to the structure, behavior and function of a technical artefact that has already been released during the design process*" and ECM as "*organisation, control and execution of ECs*" to minimize the negative impacts of ECs [6].

ECM is a significant issue that cannot be neglected or under-estimated and it should be an integral and important part of the design process. Significant portion of actual engineering work is spent on engineering changes. In an environment where the work is done by multiple teams with varying skill levels, co-operation and management will be challenging. [7] The target company had identified the ECM to be a major factor in project cost and schedule overruns. It was also understood that most of the needed infor-

mation was available or at least acquirable, but it was not generally known how different information was linked to each other. There also was no robust way to manage and utilize that information. This meant that due to lack of common practices and tools, project outcomes varied significantly and data-driven continuous improvement in company's project execution capabilities was not possible in this aspect. The hypothesis is that the previously described challenges could be addressed by data integration and additional functionality in the used software. It could also help in lowering costs caused by rework, poor information management and communication.

2.2 Engineering design process

Engineering and design activities are series of iterative actions based on initially set specifications, information defined in up- and downstream activities, and information created within the activity itself. To streamline and shorten the time span of engineering projects, many of the activities will be planned to be executed in parallel. This concurrent engineering approach can reduce the lead time significantly but can often lead to unwanted rework and unplanned design iteration. Also, many engineering tasks inherently need iteration, independent of other tasks. This means that planned design loops are formed in addition to unwanted rework.

In an environment where subsequent tasks are interconnected, and output of a particular task is dependent on other tasks, the efficient management of the whole process is essential. In a static plan, tasks can, and should be, determined in a way that unintentional design loops are avoided or at least broken into separate tasks, thus allowing the information to flow and specify in an appropriate way. Methods such as *Design Structure Matrix* (DSM) [8] or *Applied Signposting Model* (ASM) [7] can be used to model such complex engineering systems. Methods such as DSM and ASM can help visualize and understand the system as a whole, provide the needed structure to analyze the system with powerful tools based on graph theory. Such analysis could reveal indirect links and change propagation, distribution and impacts, as well as process iterations and other patterns and effects in the network [8], [9]. In addition to setting the static baseline for a project, the robust modelling of tasks can also provide better understanding of design process system in situations where changes are unavoidable. Deeper understanding of the effects caused by a change can help reduce the cost and lead time.

Relying solely on heuristic understanding of a design process is not a sufficient way to successfully manage complex projects. Instead, a robust and deterministic system of interconnected tasks is to be defined, where relationships and dependencies are managed so that at least the baseline can be set optimally. Such system, if kept up to date,

can also provide better situational awareness to support decision-making in dynamic environments where changes are unavoidable and where actions must be taken to resolve the possible issues.

2.3 Existing tools and process

Effective information management is a crucial requisite in a complex project in many ways. In this study, the focus is on source data management while other aspects, even important, are excluded. Source data is all the information needed to perform a particular engineering task. Such information can be, for example, performance requirements, specifications, instructions, or any kind of information that defines the work. The origin of source data can be external such as the end user, main contractor, subcontractor, or internal, such as another department in an organization. Source data can originate also from stakeholders outside the project organization, for example, from non-governmental organizations. Effective management of this information is crucial for any project.

The target company uses on-premise M-Files document management software to manage and distribute documentation in a project. Most often the design and engineering work cannot be done directly to a document and thus the document is only a representation of the actual design. It is also often the output and the end result of a task. The documents are also inputs for other design activities, which in turn produce documents for other purposes. In the target company, specific design and engineering tasks are managed in *Jira Cloud* software, a *SaaS* collaboration system. Wide variety of information is collected to specific tasks (in *Jira Cloud*) in order to plan, follow-up, guide and report the execution of activities. However, the information in these two systems is not linked and there is currently no way to effectively link inputs, outputs and the tasks to each other. Such functionality with a vigorous process could create the needed capability to effectively and efficiently manage the design process.

The basic structure to describe the process was formulated in accordance with the needs of the design process. A specific design task can have multiple input documents from which information is used and refined to produce an outcome (Figure 1). Thus, specific documents are linked (related to) to specific tasks. As an example, and more specifically, a *task* (in Figure 1) can use engineering information such as *energy consumption of an equipment*, *preliminary layout of a space* and *industry standard for working environment temperature* from documents *A1*, *A2* and *A3* respectively. The task is to produce a *heat balance calculation for space*, the output *B1*. All projects consist of large groups of such interconnected and dependent tasks and documentation as inputs and outputs of those tasks. Figure 2 illustrates such relationships in a very small

scale. Even small projects often consist of hundreds of tasks whereas in larger projects the number of tasks can easily grow to tens of thousands. Following the previous example, the *heat balance calculation for space (B1)* could then be used as an input for further downstream engineering activities (tasks) to produce, for example, the *air balance calculation* which in turn could be used to produce *ducting layout and routing plan*. Engineering changes in such networks can have major impacts on downstream activities and thus impacts to the project schedule and costs. Figure 2 shows also a change in a document (marked with a red exclamation mark) and how that change can propagate (red tasks, documents and relationships) through the network of engineering tasks thus requiring rework for all affected tasks.

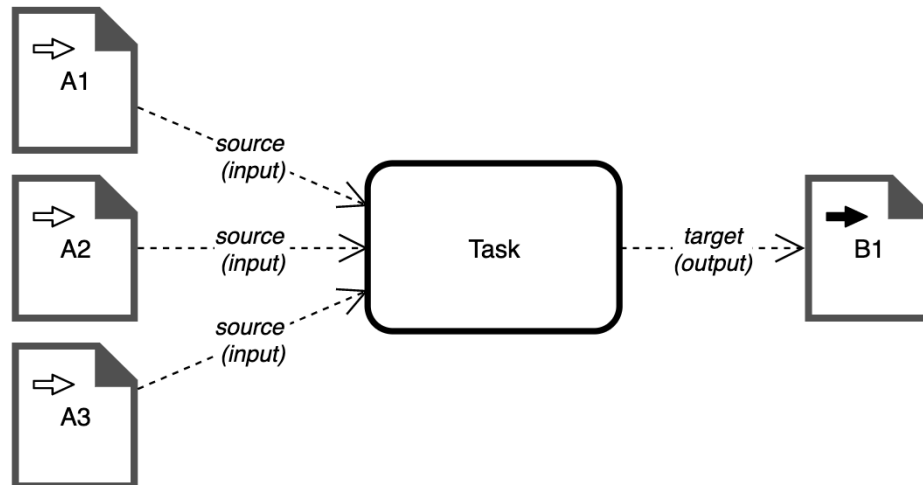


Figure 1. Relationship types between tasks and documents

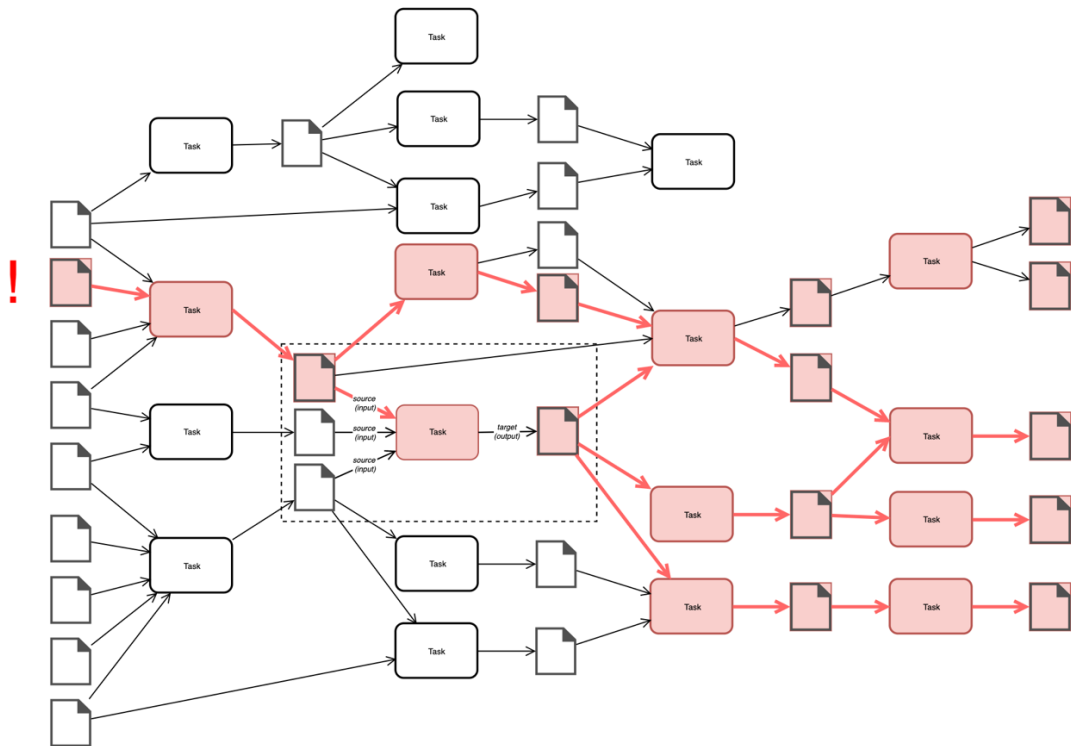


Figure 2. Example of a change propagating in a small network of tasks and documents (Figure 1 in within the dashed line)

2.4 Objectives

The objective of this study is to develop a proof-of-concept (PoC) that integrates the document management system (DMS) *M-Files* and the task management system (TMS) *Jira Cloud* in a way that documentation (in DMS) can be linked to engineering tasks (in TMS) in order to make management of engineering work easier and more robust. The PoC aims to test the hypothesis that such a system can help in planning the work, controlling the used data, and understanding the propagation of changes. By having such tools for better understanding and control, unwanted costs related to planned work, rework, and extra work can be lowered.

The study does not include the testing, deployment, and operations phase of the produced software. Also, the planning and collecting of user feedback after the initial release is out of the scope of this study. As only engineering activities are tracked in the task management software, only activities and documents related to them are considered.

2.5 Enterprise Application Integration

According to Manouvrier and Menard [9] Enterprise Application Integration (EAI) “is a collection of methods, tools, and services that work together to bring heterogenous applications into communication, as part of the traditional, distributed or extended enterprise”. The ultimate target is to get the entire system to work as a single unit [1]. Even though EAI often deals with “automation islands” [10] that cannot easily communicate with each other, that is certainly not the case here, as both systems provide highly capable application programming interfaces (API) to interact with their data [11], [12].

The topic of EAI is vast and has developed considerably from its conception in the 1980s in terms of its purpose, architecture, and technology [13]. Its purposes can broadly be divided into three types: 1.) propagation and consistency of data, 2.) management of multi-step processes, and 3.) creation of composite applications (shown in Figure 3) [9]. Data propagation can, in its simplest form, be seen as copying data to other applications behind the scenes, but can also include data integrity checks, reformatting or some rule-based actions. Use cases for multistep processes can found in situations where different stages of a process are handled in different applications, for example, in purchasing, production and shipping applications. In such cases integration is unidirectional and asynchronous but forms a coherent interdependent system. Composite applications act as common façade for multiapplication setup using their functionalities, or they can be seen as middleware intermediating between multiple applications. Interactions can be single or multistep processes and are often bidirectional. [9]

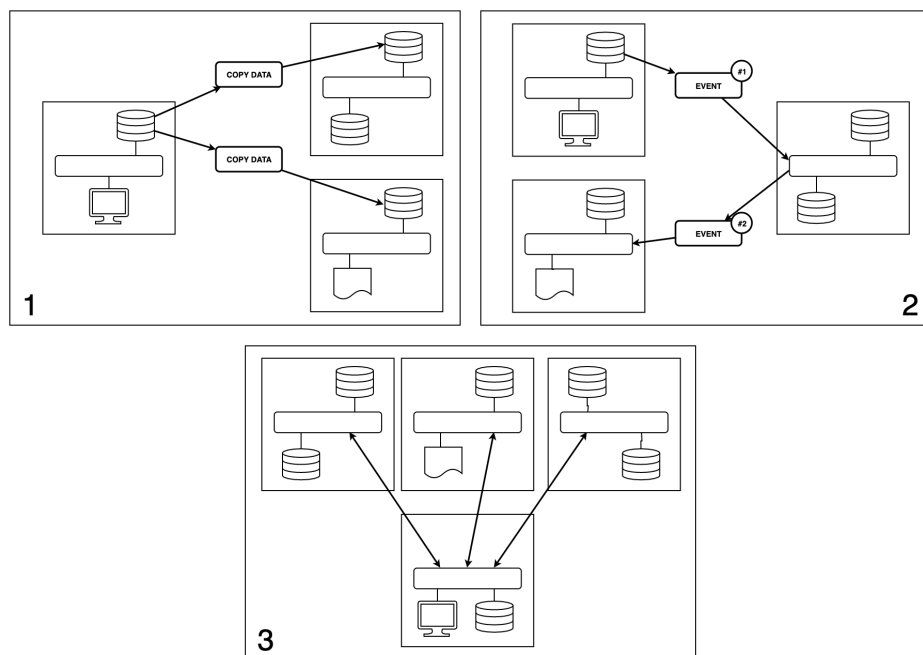


Figure 3. Purposes for EAI [9]

2.5.1 Architecture patterns

EAI deals with communication between heterogeneous applications, meaning that it does not consider individual applications architecture. This means that single application's architectural patterns are not in the domain of EAI nor the focus of this study. [9] Instead, how multiple applications, such as Jira Cloud and M-Files communicate with each other and how the system of systems is structured constitutes as the problem at hand. EAI architectures have followed their times and trends. For example, as a data format in information exchange, XML (eXtensible Markup Language) with SOAP (Simple Object Access Protocol) have in many cases been replaced by JSON (JavaScript Object Notation) and REST (Representational State Transfer) due to latter's better performance, simplicity, scalability and extensibility [4], [14]. In terms of EAI and the benefits of using JSON extend being the trend [4] as it reduces bandwidth and computing time considerably and enforces security [4], [14].

Manouvrier and Menard [9] discusses exchange architectures and presents three forms: "hub and spoke" (or star), "bus", and "snowflake" (Figure 4). In the star architecture, all information flows through a single hub. Bus acts as a technology-agnostic communication platform where applications can act as servers or clients, thus eliminating the problem of single points of failure in the star architecture. [9] Snowflake architecture is suitable for large-scale implementations with complex systems, where for example business-to-business integrations are needed [9].

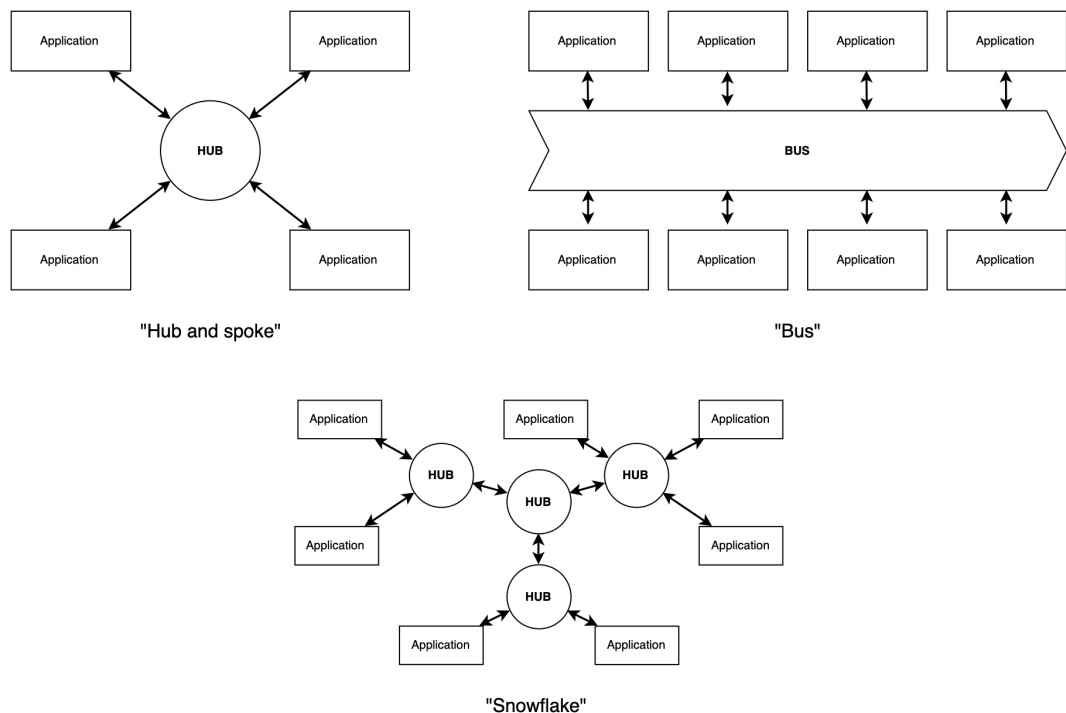


Figure 4. Basic forms of data exchange architectures [9]

In terms of software architecture paradigms, concepts such as event-driven architecture (EDA) and service-oriented architecture (SOA) are in the forefront of system design in EAI implementation as design philosophies. [15] SOA is an architectural style where individual applications offer their services with well-defined and published interfaces, and where applications can decide whether to consume those services or not. Service-oriented systems consist of reusable components that offer services to other services instead of offering them directly to users. Some SOA-based systems are built using exclusively web technologies while others are built as on-premise software [16]. Many systems are a mix of both paradigms but where SOA can be seen as client-driven request-response system, EDA is event-driven with a *fire and forget* principle, where processes are split between *sources* and *sinks* (or event sources, and event handlers) [15]. Both SOA [16] and EDA are loosely coupled architectures, which means that components are connected by a network and thus can exchange information but have their own operational logic [17]. Both systems in this study provide good APIs for data exchange and Jira even provides webhooks [11] to allow EDA functionality.

Jira Cloud can be seen as an example of *distributed component architecture* with its emphasis to extendibility. The basic features are provided by the Jira Core SaaS application and its functionalities are extended by external service providers which use Jira as a platform to offer and use their services. Figure 5 illustrates the Jira Core functionalities as a component of the whole application which can be extended with add-on applications which all can communicate with each other. Jira add-ons (or plugins or apps) are software that can be installed to plug into Jira software and aim to add functionality to it. Apps are remotely contained web applications that integrate into cloud sites. [18] Atlassian has a marketplace for these apps, similar to Apple Appstore [19]. Such architecture can provide a straightforward way to integrate other services with each other. This can be seen in Atlassian Marketplace where multiple add-ons provide integration to different systems, for example, to Salesforce and Google Drive [20]. [16]

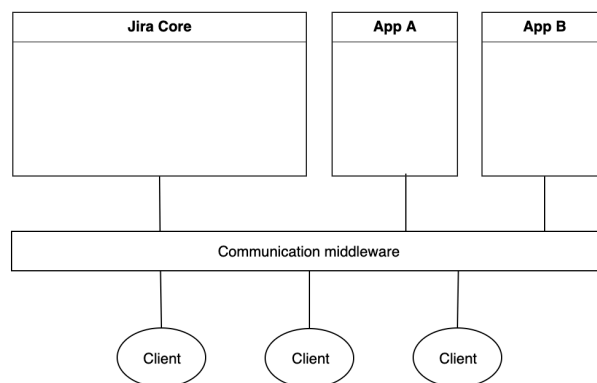


Figure 5. Jira Cloud as a distributed component architecture [16]

2.5.2 RESTful services

REST (Representational State Transfer) is a software architecture style utilizing simple client-server interaction via HTTP(S) and more specifically *post*, *get*, *put* and *delete* methods to transfer representational resources [16]. With such methods a resource can be created, read, updated and modified [4]. In addition to HTTP, REST uses other W3C/IETF standards such as URI (Uniform Resource Identifier) [12], which is used to identify the location of a particular resource. As an example, a resource such as a M-Files document object can be accessed in a URL `https://{your-domain}.com/objects/{type}/{objectid}/{version}/properties/{id}` with *get* request, given that request is authenticated by the server. Response from the server in this case would be a JSON object describing the properties of a particular document. Similarly, a *delete* method to the same URI will delete the specific resource. [16] Figure 6 illustrates the basic structure of the applications utilizing REST. Application A provides services to users while the main purpose of Application B can be to provide services, such as weather or banking data, to other services, in this case to Application A. A particular fundamental aspect of REST is its statelessness, meaning that all requests made for the server must include all necessary data to convey the information about the possible state of the interaction. Such information can be authentication tokens for login information or other data about the session stored in the request body or parameters.

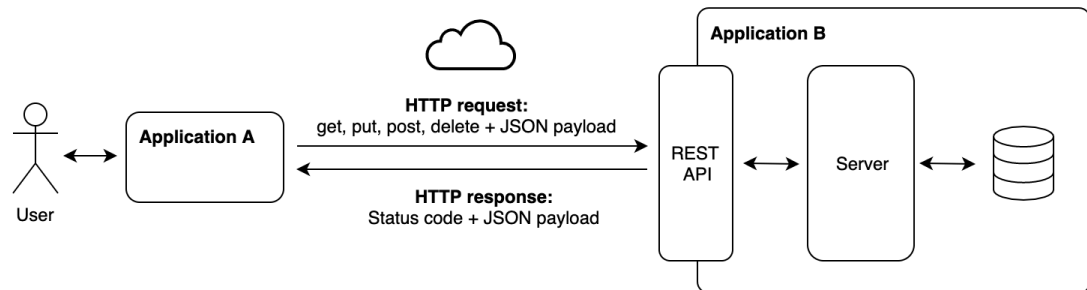


Figure 6. Application using REST API

REST has low overhead [4], and it has gained wide adoption due to its simplicity, scalability and performance [21]. According to Serrano et al. 2017 [21] “nearly all the top 100 websites from Alexa ¹ provide their own APIs”. This can be seen as an indicator of the degree to acceptance for the technology [21]. A significant amount of data online is exchanged through exposed web APIs which present the data in formats such

¹ <https://www.alexa.com/topsites>

as JSON or XML [4]. It has also become a trend to use RESTful services in particular for distributed system development.[22]

2.5.3 Building user interfaces with React

As mentioned in section 2.5, composite applications act as common façade for multi-application setup using their functionalities, and they can be seen as middleware intermediating between multiple applications. A common user interface could be created as an individual web service but in a case where an application can be extended in a way that another software's functionalities can be incorporated into it, an extension of existing UI is an option too. Creating UIs for web services can be accomplished with many technologies. One particularly popular framework is React [23]. It is an open-source JavaScript library for building user interfaces. The framework makes it easy to create component-based single page applications. It also provides a simple way to bind data and manage the state of the application using *props* and *state*. The content is often created using JSX, a syntax extension for JavaScript, which is then built into common JavaScript and HTML. [23] Source code 1 shows the basic structure of a React component where the text "Hello Taylor" is rendered to the user by using the render method, JSX and *props*.

```

1   class HelloMessage extends React.Component {
2     render() {
3       return (
4         <div>
5           Hello {this.props.name}
6         </div>
7       );
8     }
9   }
10
11  ReactDOM.render(
12    <HelloMessage name="Taylor" />,
13    document.getElementById('hello-example')
14  );

```

Source code 1. Example of a simple React component [24]

2.5.4 Implementation methods of EAI

Badampudiet et al. [24] discussed about influencing factors for decision-making for component and solution origins. EAI solutions fall into the same four categories and are subject to similar factors. EAI can be created with *components off-the-shelf* (COTS), *open-source software* (OSS), *in-house development* or *outsourced development*, which can be evaluated with factors presented in Table 1 [24]. Badampudi et al. [25] argued that the benefits for COTS & OSS over in-house development would be reduced time-

to-market, reduced development effort, and ability to add complex functionality, whereas in-house development over COTS & OSS was seen to have the ability to add unique functionality, to have reduced maintenance costs, easy integration and reduced testing time. In addition to implementation factors, it is important to have a detailed understanding of the integration requirements for the existing systems and environment, as well as a view of opportunities their integration. [24]

Table 1. Collection of themes and factors comparing system integration options [9]

High-level themes	Themes / factors	Description
Project metrics factors	Time	Time to test and integrate Time to market Cost of components Total cost of ownership Cost of replacing components Maintenance cost
	Effort	Selection and integration effort Development effort
	Quality	Quality in general External factors
External factors	Market trend	Component evolution
	Source code availability	Access and use of source code Source code documentation
	Technical support	Response time Support availability Code customization Changes in requirements
	License	License fee License obligations
Software development activity factors	Integration	Ease of integration
	Requirements	Task complexity Task uniqueness Requirement uncertainty Requirements negotiations Requirements suitability
	Maintenance	Ease of maintenance

2.5.5 Commercial integration platforms

In the past enterprise application integrations have been done as point-to-point integrations where different applications have been connected directly with each other [9]. Separate *connectors* (or *adapters*) are made for different applications to link them with each other via specific information exchange infrastructure [1]. These types of connections are most often custom-made for each case, and while they can be straightforward to create, their maintainability may cause issues due to high complexity and rapidly rising number or individual connections. Figure 7 illustrates the rising number of connections as the number of connected applications increases. If all applications must be connected to each other, as application increase from 1 to 5, the connections increase from 1 to 10. While companies often can easily have, for example, 10 applications, the number of connections will rise to 45 as number of connections c follows function $c = (n-1)!$, where n is the number of applications.

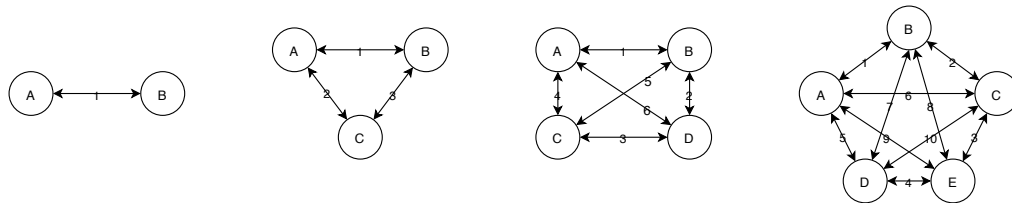


Figure 7. Rising number of point-to-point connections

The above-mentioned fact is highlighted in many iPaaS (Integration Platform as a Service) system providers' marketing material and in literature [26]. It can be argued that all systems do not necessarily have to be connected to all other systems. This alone lowers the number of needed connections significantly. This may be the case, for example, where the master data stored in certain applications must be accessible by a particular application such as an ERP system, but there may not be a need for the rest of the applications to communicate with each other. Another key selling point of an iPaaS is the ready-made connectors for legacy systems [1], [26]–[28]. In these cases, iPaaS solutions can also be a convenient way to modernize on-premise legacy systems by lift-and-shift approach [27], [28]. In addition to highly customizable logic, iPaaS solutions provide *low code* and *no code* rule-based configurations in connecting systems [29]. iPaaS can be set to listen and react to requests and act upon those [30]. Though iPaaS solutions simplify integration and definitely streamline connections, especially in cases where shadow copying and simple joining or rule-based actions are needed, the high-level views, such as shown in Figure 8, hide the complexity. Implementing iPaaS solutions does not remove the need to manage the connections in a robust way. The

needed business logic must be defined and created, and possible changes must be handled in an orderly manner.

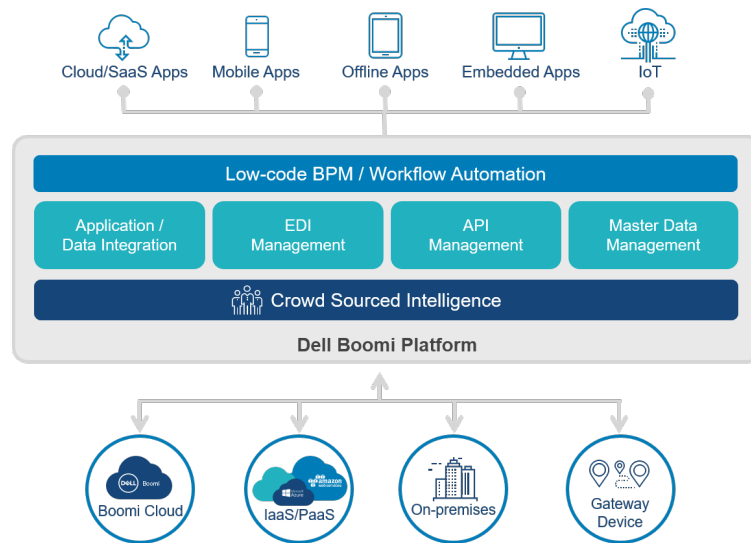


Figure 8. Dell Boomi system overview Think AI [31]

3. METHODOLOGY

The thesis relies on the three-step cycle view (Figure 9) proposed by Hevner [32] and thus follows the design science (DS) methodology. The aim of design science research is to improve the target environment by introducing new and innovative artefacts and processes [31]. Design science research method was chosen due to its wide use in the field of information technology and its process' good fit to iterative problem solving in cases where the outcome is unclear and needs to be define during the execution. This study, as well as many others, resemble closely action research (AR) studies by nature but where AR concept relies on the researcher being an active participant in the organization solving practical issues therein, DS aims at designing and creating a useful artefact [33]. In this particular case, much of the AR has been done and thus the documentation for it has been excluded and the research question formalized in terms and scope of DS research method.

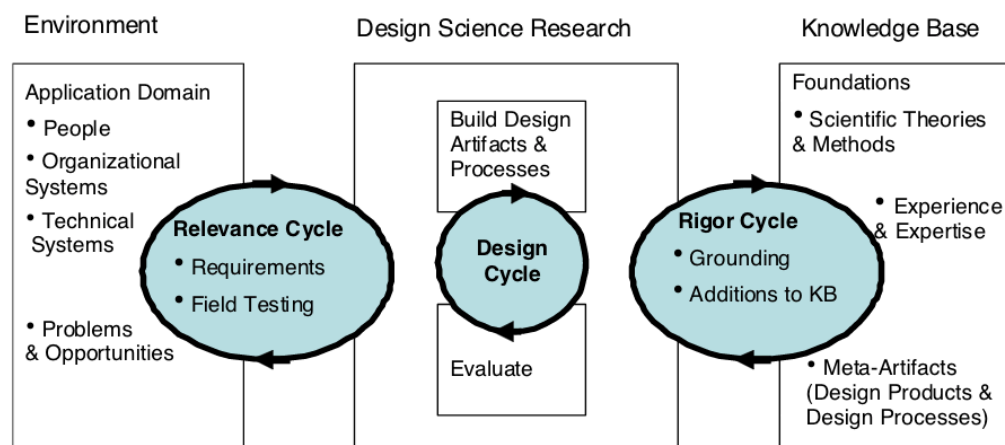


Figure 9. Design Science Research Cycles [31]

As Hevner [31] presents, in this design science research the environment with the current systems (sections 2.2 and 2.3), problems and opportunities (2.1), and Study objectives (2.4) are presented in the chapter 2, *Background*. Aside from study objectives, the information was known in detail prior to the study by the author but multiple discussions with company personnel were conducted in order verify mutual understanding. Based on this information the relevance cycle generated the functional and non-functional requirements presented in the section 4.1, *System requirements*. As proposed by Hevner [31], the design cycle iterates between the rigor and relevance cycle as additional information from the environment, existing knowledge base and from the application devel-

opment is uncovered. Application logic, structure and UI design needed multiple iterations. This was mostly due to know-how and understanding developing as the study progressed. At the same time the effectiveness of the produced system was increased as multiple versions were developed and reviewed. Chapter 4 elaborates on the decision made, and technologies and approaches chosen during the study, and as such describe the design cycle. Hevner's [34] rigor cycle should link the past knowledge to the research project, thus ensuring its novelty and innovation. This was done by leveraging the existing studies, frameworks and technical solutions in the design and creation of the produced application. Findings of the rigor cycle as discussed in chapter 5 – *“Evaluation and Discussion”*.

4. RESULTS

The objective of the thesis was to produce a proof-of-concept that could help the management of engineering source data by integrating two applications and extending their functionality. This chapter presents the solution for the problem and discusses the formulated system requirements and elaborates on the produced proof-of-concept's technology, functionality and design decisions.

4.1 System requirements

Functional (Table 2) and non-functional (Table 3) requirements were formed based on current understanding of the problem and by gathering feedback from people using both systems (Jira and M-Files) and those who were familiar with the desired process during the study. The feedback was documented in personal notes during multiple informal discussions with these people and compiled to the system requirements tables.

Table 2. Functional requirements

No.	Functional requirement	In PoC	Non-functional requirement
1	Can be accessed by all employees	Yes	Usability
2	Documents stored in M-Files can be searched in Jira	Yes	Usability, Extensibility
3	Documents stored in M-Files can be accessed in Jira	Yes	Usability, Extensibility
4	Notifies when related entities change	No	Usability
5	Can execute actions for both systems within the application	Yes	Usability, Extensibility
6	Can create reports	No	Usability
7	Can create visualizations of formed entities and their relationships	No	Usability
8	Documents can be linked to tasks	Yes	Usability
9	Documents can be source data	Yes	Usability
10	Documents can be output of the task	Yes	Usability
11	Changing linked documents must be controllable by tasks status	No	Usability
12	Change history of linked documents must be stored	No	Usability
13	System must require credentials to access documents	Yes	Security

Based on the objectives of the study, functional requirements that were not essential in creating the proof-of-concept (PoC) application and whose creation would've caused significant increase in development time or their implementation was found to be beneficial to create outside application, were left out of the PoC add-on. For example, as seen in Table 2 "In PoC" column, the *notification system* (No. 4) was left out for its estimated high development time, and the *reporting function* (No. 6) because it would be better to be handled by reporting systems in Jira Cloud, M-Files, or by external reporting solutions used by the company. Functional requirement 11 was seen to be handled by Jira Cloud system administration and functional requirement 12 can be fulfilled by how M-Files stores the related data. Other functional requirements were fulfilled, and their documentation is discussed in section 5.1. Basic non-functional requirements were formed in the beginning of the study in order to capture specific aspects important to the implementation, such as information security, the ease of use and the availability of the system. The practical fulfilment of these requirements is discussed in section 5.1.

Table 3. *Non-functional requirements*

Non-functional requirement	Description
Usability	Easy to use
Usability	Shall not significantly slow down the use of the application
Security	IT Security shall not decline by the introduction of the add-on
Availability	Availability of M-Files and Jira cannot be lowered by the system
Availability	System availability 99%
Scalability	The system shall scale in par with the integrated systems (i.e., User experience shall not deteriorate due to the add-on)
Maintainability	System shall be easy to maintain. System shall be developed according to the needs of the users.
Extendibility	Functionalities can be added to the system
Portability	System can run in various environments

In addition to the functional requirements, the target users' use cases were defined and are shown in Figure 10. A particularly valuable functionality – the analysis of the linked documents was defined to be excluded from the add-on produced in this study as it would be more suitable to be included in another system (see functional requirements in Table 2). The implementation of such functionality was mandatory for the system but not for the proof-of-concept application produced in the study.

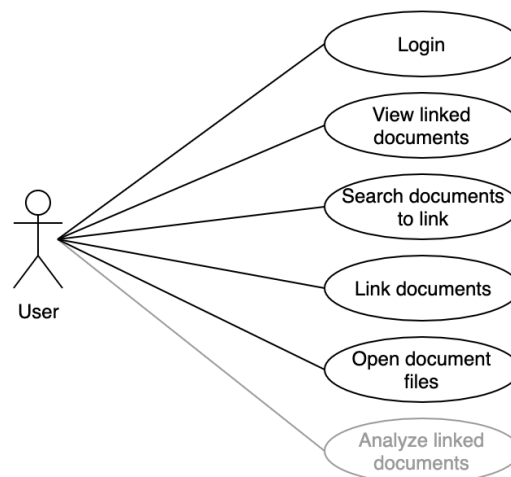


Figure 10. Use cases

The basic workflow for the use of the add-on is shown as an activity diagram in Figure 11. It shows the functionalities for browsing, opening and linking documents within the task management software Jira. Single sign-in was not implemented in the add-on so users must login also to M-Files within Jira. Basic information of related documents of a specific task can be viewed without logging in with M-Files credentials but viewing any information other than document's name, id and version, the users must log in with his or her M-Files credentials. After signing in, the users can search for documents, open them, and create links between documents and tasks. The sequence of messages between the user and different parts of the system is shown in Figure 12 where, for example, the initial opening of a task in Jira connects only to Jira servers whereas opening the view to use the add-on connects to the add-on server.

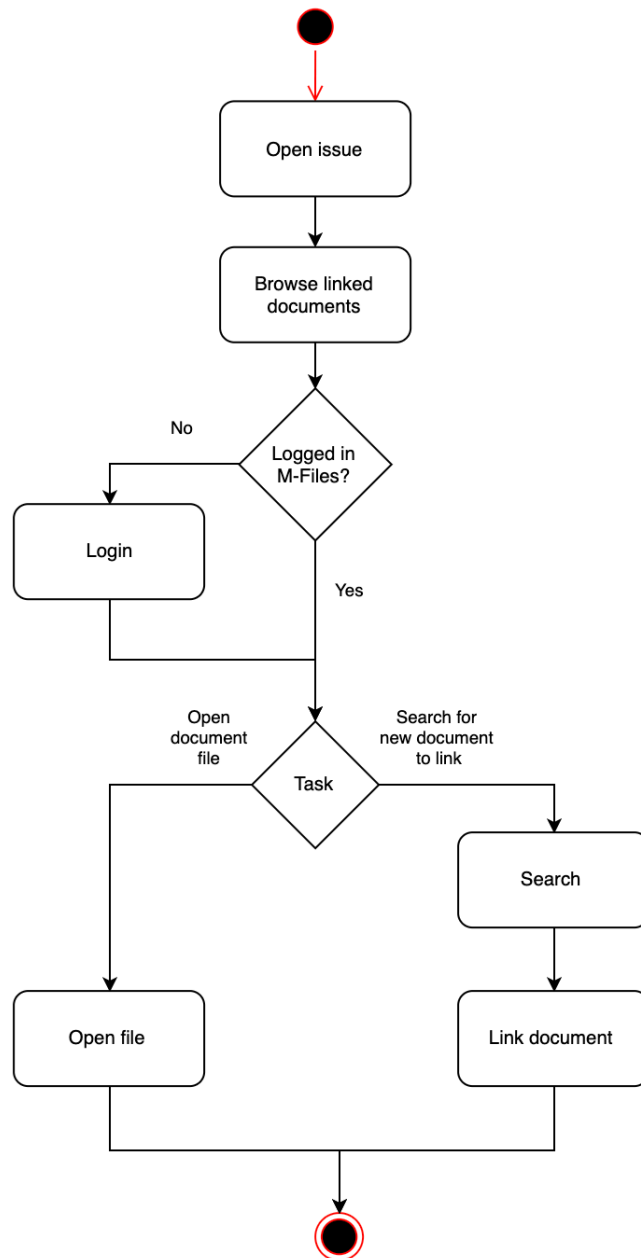


Figure 11. Activity diagram of the basic functionalities

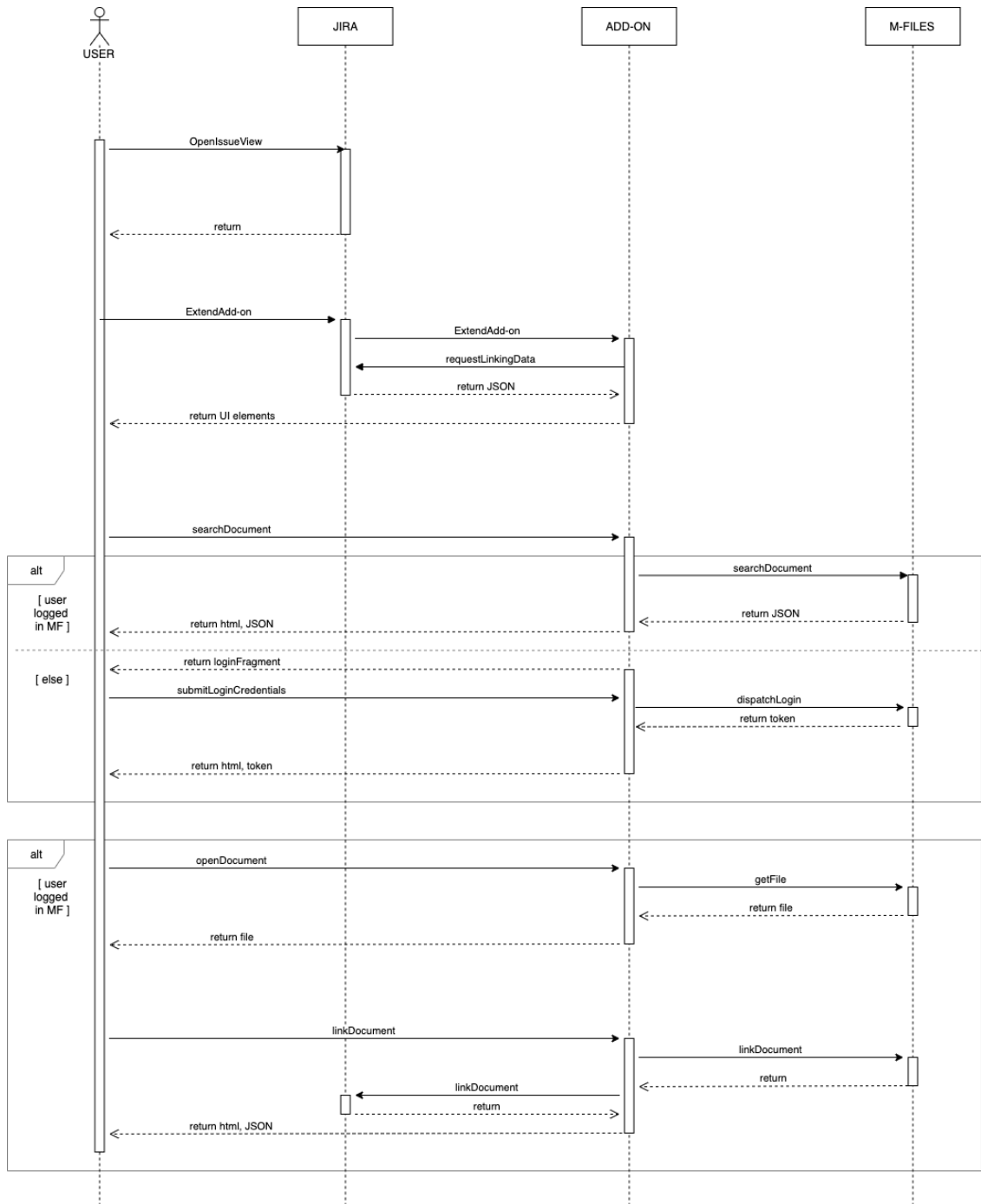


Figure 12. Sequence diagram for basic functions

4.2 Add-on for Jira Cloud

Integration of M-Files and Jira Cloud could have been done in many ways and with many technologies. Due to Jira Cloud being a SaaS application and because it provides a powerful REST API and its documentation and other resources are very user friendly, the use of REST became the starting point for the development. An ODBC (Open Database Connectivity) driver for Jira would have been available [12] but its use would've unnecessarily increased the complexity of the system. Especially writing directly to Jira's database could have caused issues in data integrity. M-Files provides a COM (Component Object Model) and a REST API [35]. The support for REST is not as extensive as it is for COM but after learning that the needed functionalities were possible to create with REST, a decision was made to use REST in both ends of integration.

The needed functionalities determined that a simple rule-based integration method was not enough to fulfill needs of the solution since the solution cannot be specified in terms of "if this then that" [36]. Data propagation with data consistency checks, even together with multistep processes is a deterministic flow which is not enough to enable the functionalities needed. The needed solution would need a specific business logic to enable user input from both systems and in a way that would be as convenient as possible to use. This meant that using commercial of the shelf (COTS) products such as *IFTTT*, *Microsoft Flow*, or *Dell Boomi Flow*, or other open-source software (OSS) such as *Apache Airflow*, was not an option. COTS products and OSS could have been extended to fulfill the needs of the target company but in this case would have increased the complexity of the system and needed further development effort. There also was no need to use iPaaS solution to connect legacy systems nor need to connect multiple applications. In addition to above-mentioned technical aspects the cost of commercial iPaaS systems such as Dell Boomi would have increased the cost of the PoC unnecessarily. For future use, it could be beneficial to use iPaaS for the basis for the integration if more systems need to be integrated. Future activities in relation to the add-on are discussed in section 5.3.

Since user input was needed to define the links between tasks and documents, the most convenient way was to extend either of the existing systems. Both Jira Cloud and M-Files can be extended but as Jira Cloud acts as the task management system in which designers work in, it was considered to be the more suitable option. Also, the developer-friendliness and the technology agnosticism of extension of Atlassian products played a big role in the decision. The use of REST APIs and server-client architecture allowed to choose quite freely the used technology. Based on the developer's previous experience, simplicity of having one programming language for front and backend, as

well as being able to develop and run the application in various environments, a JavaScript-only solution was chosen.

A *composite application* with a *star* topology was created in service-oriented architecture (SOA) using *Node.js* and specifically *Express* server to act as an independent middleware between Jira Cloud and M-Files. It provided data integration with a multistep process for linking information in those systems together. By extending Jira Cloud's user interface (UI) with *React.js*, the add-on application was able to seamlessly extend the capabilities of Jira Cloud as well as the Jira Cloud – M-Files system of systems. The produced solution plays also a key role in endorsing vendor independence as it allows the TSM to use other REST-enabled DMSs with little modification. However, as the UI component is specifically made for Jira Cloud, migrating to another TMS would mean major recreation of the integration application.

Functional requirements that were defined to be implemented in the PoC were successfully fulfilled and their details are discussed in sections 4.1, 4.2.4 and 5.1. Non-functional requirements for usability were fulfilled with Atlassian-compliant modern UI, maintainability and extendibility was kept in mind with proper commenting of code. Extendibility could have been better if the codebase would have been structured with a large-scale application in mind. However, as a PoC, the application structure (see 4.2.2), originating from Atlassian scaffoldings, will suit the purpose. As a Node.js application, the portability is good as it can be run in basically any Windows, Mac or Linux environments. Based on tests in the development environment, the application performed reasonably fast, and no availability issues were found in the system itself or in the systems it connected to. Scalability, as was determined in the non-functional requirements, was not considered in the application itself. Stress tests with 100+ simultaneous users should be conducted during the test phase. Planned way to ensure scalability and availability is to use products such as *NGINX* [33] for horizontal scalability and load balancing. Security aspects are discussed in detail in section 4.2.6.

4.2.1 Architecture

The architecture of the add-on follows the guidelines provided by the application platform provider Atlassian. The produced add-on cloud service sits between the two software services used in the company. Figure 13 depicts the high-level entities and the way they communicate. In terms of system architecture as presented in section 4.2.1, the created application is a of form *star* in its simplest way. Atlassian provides its own

framework for extending their offerings in cloud with add-ons. The add-ons are independent web-applications which communicate with Atlassian application via REST APIs. All functionalities of Jira Cloud are not available via REST interfaces [33]. Often add-ons render content to the main application inside an iframe but that doesn't have to be the case. Because Atlassian provides a scaffolding for add-on development for their cloud services, and due to their particular technology stack, the following technologies and libraries were used to produce the add-on in this study: *Node.js*, *Express.js*, *Atlassian*, *Material-ui*, *React* and *Axios*.

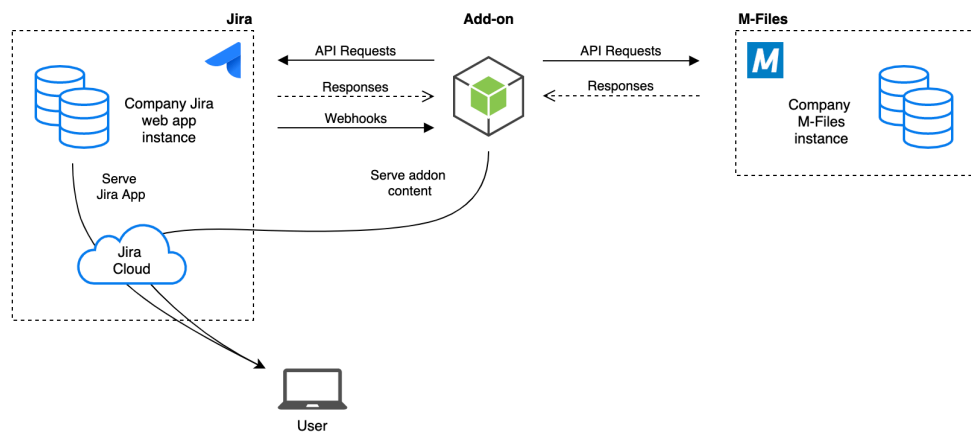


Figure 13 High-level representation of overall architecture of the produced add-on – “Middleware add-on” (Derived from article “Introduction to Atlassian Connect” [37])

4.2.2 Application components and structure

Atlassian provides a scaffolding for a Jira Cloud add-on. It is a blueprint and an example application to show the basic structure and functionality of an add-on application for Jira Cloud. Such scaffolding will ease and speed up the development process. Hence, the PoC deliberately utilizes that structure, even though in the longer run the code will most likely need refactoring. File `.env` holds environment variables such as API endpoints, ports, database mapping configurations separately from the other source code. This is done in order to allow easier configuration and better security, as if for example, API endpoints change, different information need to be mapped or mappings change, the parameters can be set and changed in this file (Source code 2).

```

    REST_***_JIRA_DOMAIN=https://*****.atlassian.net/
2  REST_***_MFILES_DOMAIN='https://mfiles.*****/REST'
    REST_***_MFILES_VAULT='C8BD0C48-2A52-4063-8B56-*****'
4  [...]

6  LISTEN_PORT=5000
    [...]

8

    JIRA_DOCUMENT_LINK_FIELD=customfield_10042
10 [...]

12 JIRA_ISSUE_OBJECT_TYPE_IN_M_FILES=112
    JIRA_ISSUE_LINK_PROPERTY_DEFINITION_IN_M_FILES=1189
14 JIRA_ISSUE_CLASS_ID=2

```

Source code 2. Parts of the environment variables in .env file

The entry point to the application is *App.js* and it imports dependencies, configures application and starts the server. Source code 3 shows parts of the code defined in *App.js*. *Atlassian-connect.json* is a descriptor file which is used to specify the add-on specifications such as endpoint URL for the add-on and needed permissions in JSON to Jira Cloud. In this particular use case, both read and write permissions are needed. The information in the descriptor is used during the installation of the add-on inside Jira. The dependencies and configurations used in both development and production environments are stored in *config.json*. Source files for dependencies are located in *node_modules* folder. A separate directory was reserved for utility functions such as *logging.js*. A public directory provides static files such as images and client-side JavaScript (*glance.js*, Source code 4). Source code 4 shows the importing of the React library and parts of the React component and its states. The Express routings for requests from client are separated into a single file (*index.js*), which is imported in the entry point *App.js*. Source code 5 shows the importing of necessary libraries such as Axios, and an example for two routes. All the different React components are defined in the *views* directory as JSX files such as *Login.jsx* and *Search.jsx*. They correspond to all the building blocks of the UI and they are discussed more in the section 4.2.3.

```

import express from 'express';
2 import bodyParser from 'body-parser';
  [...]
4
const app = express();
6 const addon = ace(app);
  [...]
8
app.set('port', port);
10 app.set('trust proxy', 1)
  [...]
12
const devEnv = app.get('env') === 'development';
14 app.use(bodyParser.json());
app.use(addon.middleware());
16 [...]

18 if (devEnv) app.use(errorHandler());
  [...]
20
routes(app, addon);
22 [...]

24 http.createServer(app).listen(port, () => {
  console.log('App server running at http://' + os.hostname() + ':' +
26 port);
  [...]

```

Source code 3. *Parts of the application entry point App.js*

```

import React, { flex } from 'react';
2 import { useState, useEffect } from 'react';
  [...]
4
const Glance = () => {
6
  const [searchWord, setSearchWord] = useState('');
8  const [results, setResults] = useState([]);
  [...]
10

12  return (
    <div>
14    {!loggedInMF ? <Login/> : <></>}
    <Links/>
16    <Search/>
    <Results/>
18  </div>
  )
20 };

22 export default Glance;

```

Source code 4. *Parts of the main React component*


```

import axios from 'axios';
2 import fs from 'fs';
  [...]
4
export default function routes(app, addon) {
6
  app.post('/createissuemfiles', addon.authenticate(),
8  addon.checkValidToken(), (req, res) => {
10  [...]
12  });
  [...]
14
  app.get('/', (req, res) => {
16    res.redirect('/atlassian-connect.json')
18    });
  [...]
  })

```

Source code 5. Parts of the Express routes in *Index.js*

4.2.3 User interface

The user interface (UI) follows mostly the design guidelines provided by Atlassian – company behind Jira [37]. It uses also UI elements from Atlassian’s Atlassian Kit to ensure the look of seamless integration [38]. In addition to elements in Atlassian Kit, also custom elements and elements from *Material-UI* [37], [38] were used. Atlassian Kit and Material-UI are UI libraries which offer ready-to-use UI components [23].

The placement of the UI for the integration application is implemented as an *issue glance* – an interface element provided by the Jira Cloud developer framework. It has the ability to have a minimal footprint with minimum information by default and has an extended view with all the functionalities. Figure 14 shows a sketch for the user interface where the functionalities for the add-on are placed in a *glance view*. Green elements represent the add-on and its varied footprints and functionalities in *minimized* (on left) and *extended* (on right) form. White and blue elements represent default UI elements in Jira Cloud.

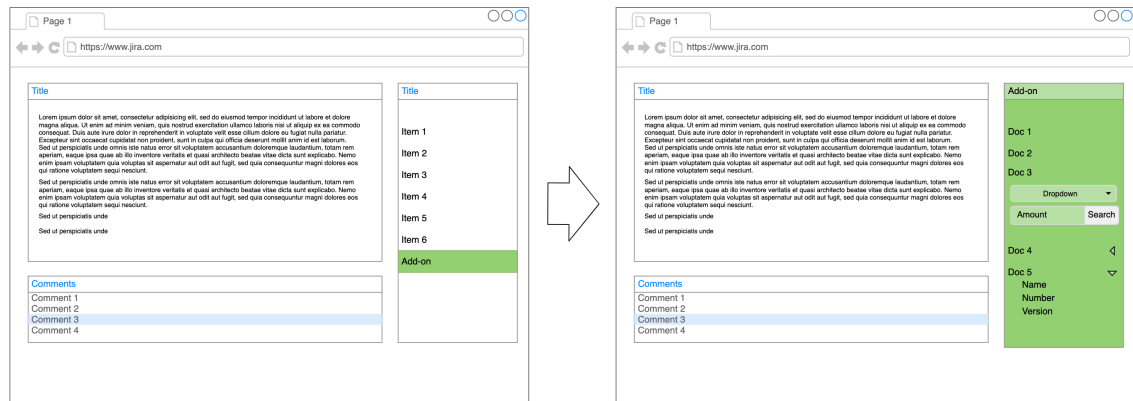


Figure 14. Sketch for the add-on's user interface Atlassian "Glance view"

The extended view holds functionalities for all actions executed with the add-on. Figure 15 illustrates the basic layout of the extended view and its functionalities. Top part of the UI is reserved for the currently linked documents and the lower part for the searching functionality. Each element in both previously mentioned parts can be extended to show additional information and corresponding files. The user interface of the add-on is done using *React* [39] components which are rendered inside an *iframe*. Figure 16 depicts the nested hierarchy of the used components. In addition to the component mentioned below, other components such as buttons, forms, and icons from *Material-ui* and *Atlaskit* were used [40]. The state of the application was handled with React state functionality. Table 4 lists the used variables and their purpose. Picture 1 shows the actual *issue view* where the add-on is placed in the section on the right column similarly to Figure 14. Picture 2 shows the "Manage related documents" section where the number of linked documents is shown.

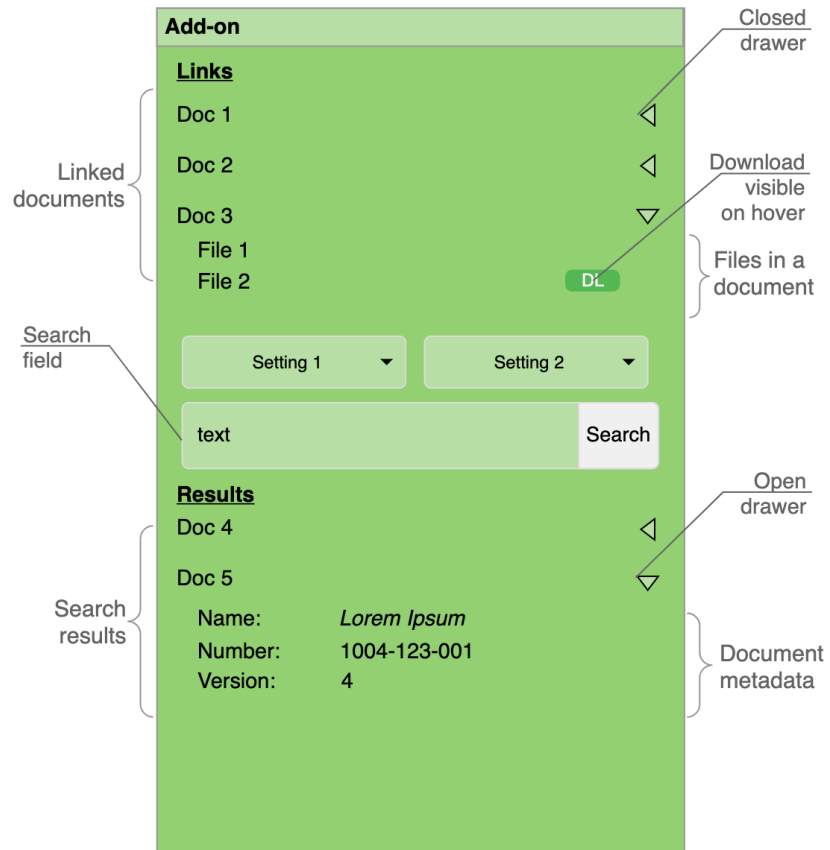


Figure 15. Sketch for extended glance view

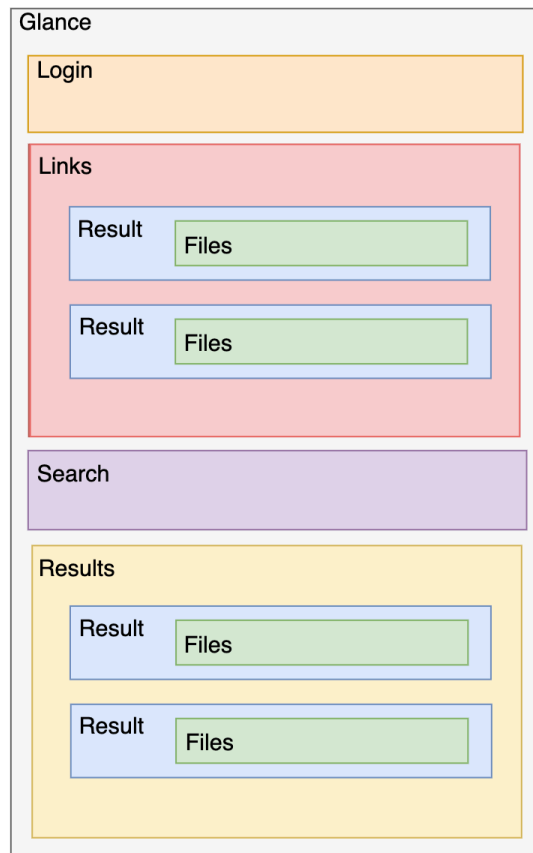
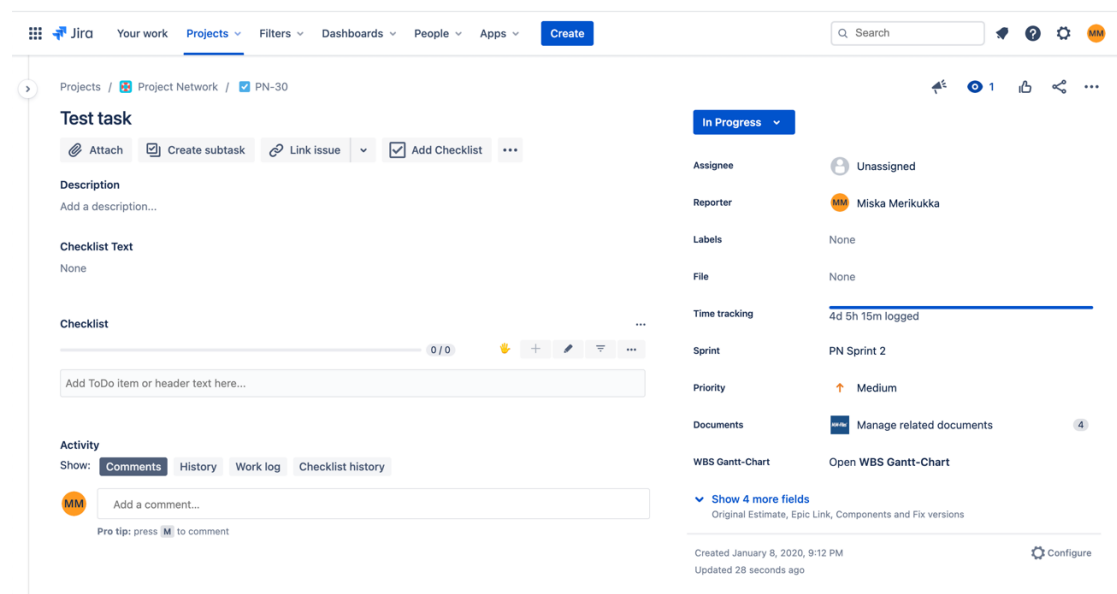
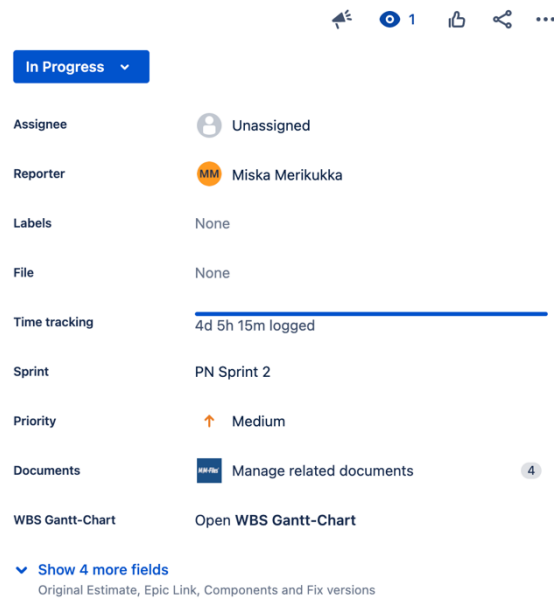


Figure 16. Hierarchy of the used React components

Table 4. UI's state variables

Variable	Type	Purpose
searchWord	String	Search word in the search box
results	Array	List of search results from M-Files
links	Array	Existing linked source documents
targets	Array	Existing linked target documents
token	String	Authentication token to authenticate traffic
isSaved	Boolean	Status of saving (controls the visibility of the save button)
loggedInMF	Boolean	Login status of M-Files (controls the visibility of the login view)
linkSource	Boolean	Linking type (controls if document is linked as source or target)
showHint	Boolean	Controls the visibility of UI hints

**Picture 1.** Jira issue view - Basic layout



Picture 2. Jira issue view, Right panel with issue glance element

4.2.4 Functionalities

Opening the basic issue view in Jira shows the basic content of the issue. During page loading only visible data is sent to the browser. This means that in the case of the produced integration add-on only the number of linked documents is fetched. This information is stored in Jira database and is thus provided by the Jira server. By clicking the "Manage related documents" section (Picture 2) the user interface reveals the extended view. In this stage, the view with its elements and data is fetched from the add-on server (see also Figure 12). Within this view, the documents inside M-Files can be searched, downloaded, inked and unlinked. Picture 3 illustrates downloading a file related to a document. The extended view shows possible linked documents in two sections: target and source data documents (see Figure 1) as well as the search box (Picture 4). All linked and searched documents show the name of the document, its unique id number and the version of the document. By clicking the "i" icon the view extends to show all metadata related to that document (Picture 5). Next to the "i" icon there will be a down arrow if the document has files related to it. By clicking the down arrow, the element extends to show the files related to it. Those files in turn can be downloaded and viewed by clicking the name of the file in the extended view. Clicking the actual document name adds or removes the linking between the issue and document. The linking type (target or source) is shown as a paper clip icon (Picture 4) and can be switched by clicking it.

Documents

Target data documents

Commented file 20.10. - Rev. A (ID TEST01-151)
ID: 151 Version: 3

Commented file 20.10. - Rev. A (ID TEST01-151).docx

Työnkierron kuvakkeet - Rev. B (ID TEST01-143)
ID: 143 Version: 23

excel_virheilmoitus.pdf

Source data documents

Test Contract - Rev. A (ID TEST01-3)
ID: 50 Version: 18

mfiles4B.txt

Työnkierron kuvakkeet - Rev. B (ID TEST01-143)
ID: 143 Version: 23

excel_virheilmoitus.pdf

Picture 3. Jira issue view, extended glance with available file downloads

Documents

Target data documents

Commented file 20.10. - Rev. A (ID TEST01-151)
ID: 151 Version: 3

Työnkierron kuvakkeet - Rev. B (ID TEST01-143)
ID: 143 Version: 23

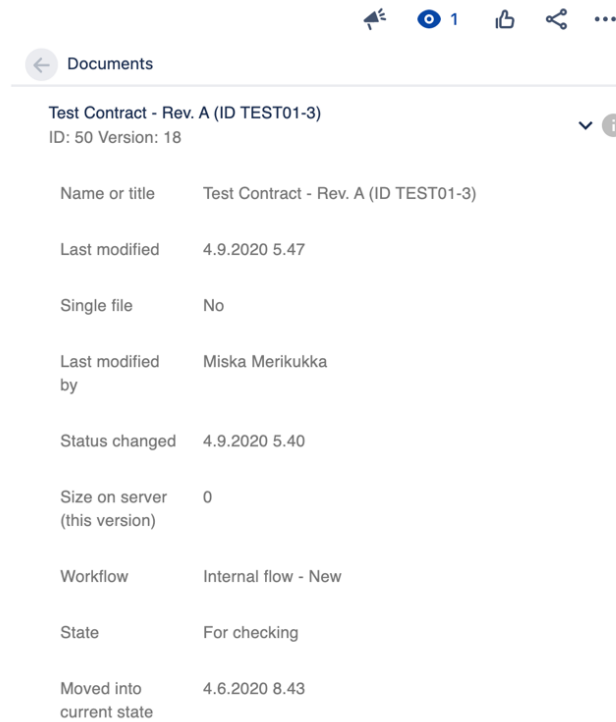
Source data documents

Test Contract - Rev. A (ID TEST01-3)
ID: 50 Version: 18

Työnkierron kuvakkeet - Rev. B (ID TEST01-143)
ID: 143 Version: 23

Search for documents in M-Files

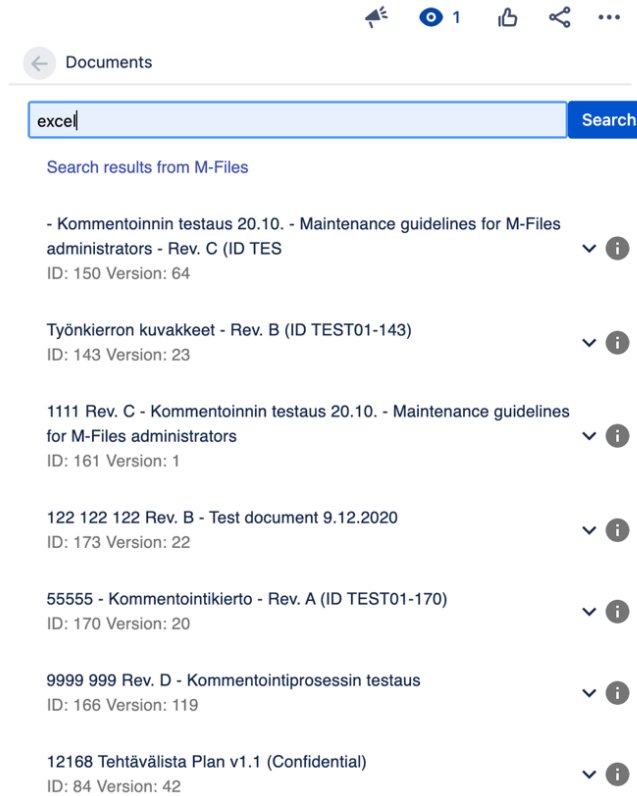
Picture 4. Jira issue view, extended glance with linked documents



Documents	
Test Contract - Rev. A (ID TEST01-3) ID: 50 Version: 18	
Name or title	Test Contract - Rev. A (ID TEST01-3)
Last modified	4.9.2020 5.47
Single file	No
Last modified by	Miska Merikukka
Status changed	4.9.2020 5.40
Size on server (this version)	0
Workflow	Internal flow - New
State	For checking
Moved into current state	4.6.2020 8.43

Picture 5. Jira issue view, extended glance with document metadata

The search functionality is available only for users who are authenticated with their M-Files credentials. If the user has not provided the credentials and thus may not have the rights to access files, only the linked documents' names are visible to them. Documents cannot be viewed or searched. After authorization, the logged in user can search documents within the configured M-Files Vault in the basic extended glance view (Picture 6). All previously mentioned functionalities apply also for the search.



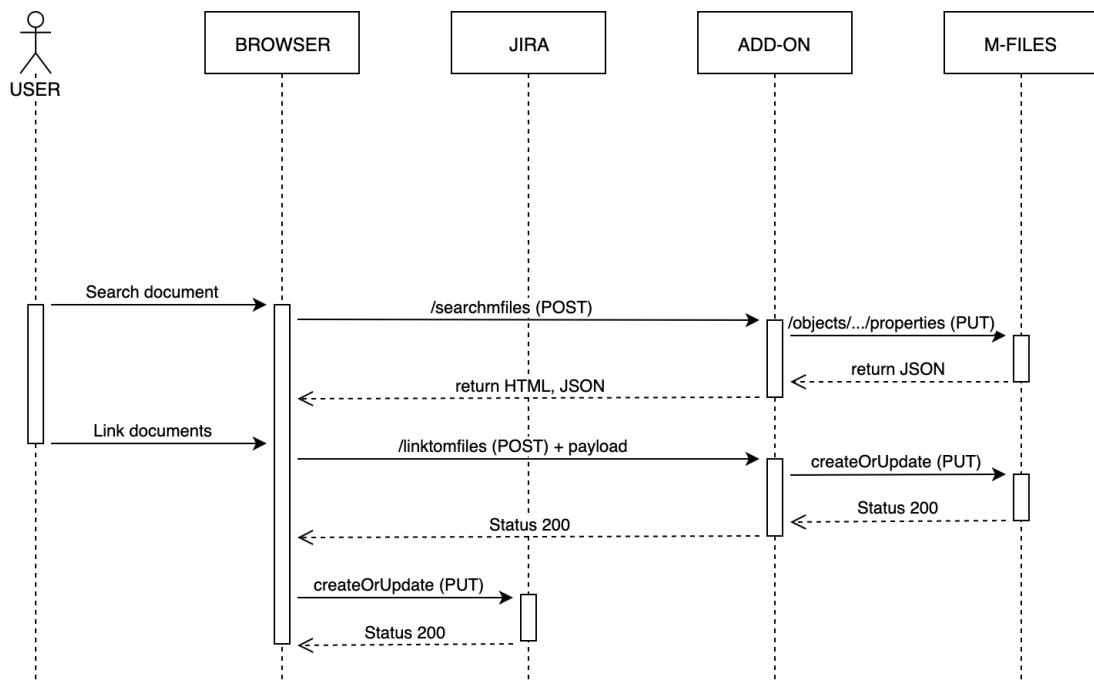
Picture 6. Jira issue view, extended glance with search results

4.2.5 Backend and business logic

Express framework is an open-source backend web application framework for Node.js [41]. It handles the registration of the add-on, user login for M-Files and creation, updating and erring the links in both systems. Any actions done for M-Files were first requested from the add-on. From there the add-on server parsed and redirects the requests with additional required attributes to the M-Files REST API with *Axios*, a promise-based HTTP client and library for Node.js [39]. The additional attributes are used to map data and between systems according to environment in use. In addition to data mapping the server also handles errors, checks for duplicates and keeps a simple high-level log. A custom express middleware in the add-on scaffolding also authenticates and checks the validity of the token in every request [42], [43]. Table 5 lists all endpoints for the produced add-on. As an example, when user saves changes for linked documents, the front-end JavaScript makes a *post* request to the add-on, which in turn makes a *put* request to M-Files endpoint. Upon success, the add-on forwards confirmation to the front-end which calls (*put*) Jira Cloud API directly to create a similar record to Jira database. Figure 17 illustrates the simplified sequence of actions in the previous example while leaving out exceptions, necessary preceding events and internal calls such as checks and authorizations.

Table 5. Add-on API endpoints and their required parameter

Method	Path	Body parameters
POST	/createissuemfiles	description(String), jiraKey(String)
POST	/linktomfiles	linkedObjectIDs(Array)
POST	/signtomfiles	username(String), password(String)
POST	/searchmfiles	searchWord(String)
POST	/getDocProperties	id(int), version(int)
GET	/	-
GET	/login-dialog	-
GET	/glance	-

**Figure 17.** Sequence diagram for linking documents in M-Files to issues in Jira Cloud

The data related to the linking of documents (in M-Files) and tasks (in Jira Cloud) is stored primarily within Jira Cloud database while the same information is mirrored to M-Files for browsing and reporting purposes. Thus, the add-on does not store any information. The linking information is processed as JSON with the REST APIs, but the information is stored in some SQL databases [43]. Database schemas are not freely available for neither products, and the data structures are encapsulated and not visible to the add-on nor the developer. However, due to the chosen technologies for the integrations, there is no need to gain access for this information. The link between a document and a task is stored in Jira Cloud as shown in Source code 6 and is accessed with *post* and *get* requests to a task-specific API endpoint. A specific task in Jira Cloud

is linked to a specific version of a specific document in M-Files. For example, a task with a key “PN-30” and an id of “10144” in Jira Cloud has the record shown in Source code 6. It shows that it is linked to a document of type “0”, id of “50” and version of “18”. The record also shows additional information such as the name of the document and the files related to that document.

```

2   {
   "key": "relatedDocuments",
   "value": {
4    "documents": [
       {
6     "Title": "Test Contract - Rev. A (ID TEST01-3)",
       "ID": "50",
8     "ObjVer": {
       "Version": 18,
10    "VersionType": 4,
       "ID": 50,
12    "Type": 0
       },
14    "Files": [
       {
16     "Name": "mfiles4B",
       "EscapedName": "mfiles4B.txt",
18     "Extension": ".txt",
       "FileGUID": "{0CDA39CF-8B22-4800-99AE-CF610B268488}",
20     "ID": 4,
       "Version": 2,
22     "FileVersionType": 3
       }
24    ]
     }
26   ]
   }
28  }

```

Source code 6. Abbreviated example of a document link in Jira Cloud as JSON

The information in M-Files can be assessed and modified in a similar manner but the information is most likely stored in separate tables and linked with foreign keys. In M-Files (Figure 18), the links between engineering tasks and documents can be seen as a hierarchical structure even though no actual hierarchies are formed. This can be seen as the hierarchy wraps around itself showing the task “PN-30” as its own child in Figure 18. To continue the previous example, in M-Files, browsing the task “PN-30” will show related documents “Työnkierron kuvakkeet – Rev. B (ID TEST01-143)” and “Test Contract - Rev. A (ID TEST01-3)”, which alongside other information such as *Author* and *Project*, shows the linked task “PN-30”. Linking functionality with the add-on in Jira Cloud is shown in Picture 4 on page 32, and data topology is shown Figure 1 and Figure 2 on page 6.

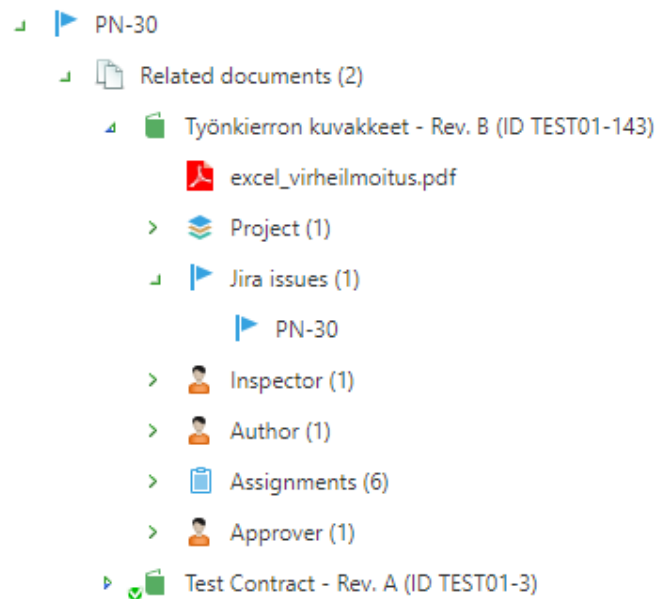


Figure 18. *Linked documents and Jira issues in M-Files*

4.2.6 Security

Many aspects must be considered when building applications that are open to the web as they can expose company data to the web. Petzold & Hoogvliet [44] blogged about Atlassian add-on security where they raised endpoint communication, authorization concept, add-on requirements and the protection of private add-on as the main aspects of Atlassian *Connect* application security. This, in addition to *OWASP Top 10 standard awareness documentation* [11], was set as a reference for security.

The fact that both integrated applications interact via public network with REST APIs was considered and evaluated during the study. In the case of this particular add-on, the goal was to ensure that the security of the used applications does not deteriorate upon introduction of the add-on. In particular, this means that the communication between applications is secure and that all actions are authorized. This was achieved by communicating only with *Hypertext Transfer Protocol Secure* (HTTPS) and authenticating all requests with *JSON Web Tokens* (JWT).

As users log in to Jira, no user authentication for the actual use of the add-on is necessary but the communication between the add-on and the actual Jira Core application is done by authenticating each request with JWT [12]. This means that all requests for the browser to the server are authenticated and validated and thus only the communication between authorized clients is possible. The JWT authentication method is provided by Atlassian but the use is under responsibility of the developer.

There is currently no way to integrate the Jira user permission management with the user management of M-Files. Thus, users' permissions to query documentation must be verified in the add-on. All allowed Jira users can see the name and ID of the linked documents, but they cannot search, add, open or request other metadata without being authenticated. The authentication is done by forwarding the login details to the M-Files REST API, which in turn, returns a token which is then attached to the user session, thus allowing the use based on M-Files privileges. M-Files developer portal documentation recommends *authentication tokens* which upon receiving can be attached to the request headers. [45]

4.2.7 Development environment and installation

Based on ease of use and past experience the application was developed with *Visual Studio Code* [46] and *ngrok* [47] was used to expose local development server to the Jira platform during development. The application itself runs on Node.js JavaScript runtime environment [31].

Installation of the add-on is done by Jira Cloud instance administrator via administration panel within the system. In this particular case the produced add-on is intended for internal use within the organization so the application is installed by referring to the add-on server address. This particular method is not optimal for use in production but is suitable for the purpose of the study. If a decision is made to extend the use of the add-on the Atlassian Marketplace supports (and endorses) private listings as which the produced add-on should be registered. Updating a cloud service add-on is very straightforward as any new functionality or modification is instantly available to all users after modifications are pushed to the add-on server.

5. EVALUATION AND DISCUSSION

The evaluation of the proof-of-concept done against the requirements specified during the study. This chapter discusses the fulfilment of those requirements, describes the development process, and presents possible future development initiatives.

5.1 Requirements

As described in chapter 4.1 System requirements, all functional requirements were fulfilled with the exception of functional requirements 6, 7, 11 and 12 (see Table 2) which were considered to be more suitable to handle outside the produced add-on and number 4 which was excluded due to estimated relatively high development time. The add-on can be accessed by anyone in the organization with proper authorization, documents and their links can be accessed in both systems and links can be created in Jira Cloud.

Based on initial demonstrations the use of the add-on was seen by future users as intuitive and easy to use. The “uncached” load time of a page in the development environment ranged between 7-9 seconds and the add-on timing as on average 2,5 seconds with actual download time of 1,5 seconds. Load times will be significantly lowered by minifying code and deploying the add-on to a proper production environment. The presence of the add-on doesn't seem to be affecting the usability in terms of loading time since identical performance was documented without the add-on. The add-on does not deteriorate the availability of Jira Cloud or M-Files even if the availability of the add-on would be affected in some way. Only the capability to link documents (in M-Files) to tasks (in Jira Cloud) and browsing files in Jira Cloud would be affected. Availability was not stress tested during this thesis. In its proof-of-concept form, the application's scalability is not very scalable. This is due to decisions made during development. However, as stated in section 4.2 the scalability can be easily increased with software such as NGINX.

System security relies to the authorization from M-Files and Jira Cloud. All traffic to the add-on must originate from either Jira Cloud or from M-Files and the requests are authenticated with tokens. All data generated and handled by the add-on is routed with HTTPS and the token does not store any information outside Jira Cloud or M-Files, and also in that respect cannot deteriorate the security.

As the add-on is created as a Node.js application, the portability is great. Depending on the IT departments preferences the application can be deployed to companies own servers running Windows, or to cloud running most probably some Linux-based system. Extendibility suffered in terms of application structure since no systems were created in order to allow additional components to be added to the add-on. However, new functionalities can be added to the system. Maintenance of the source code must be done in-house and can require some work. Security updates for the dependencies used must done and possible updates on Jira Cloud or M-Files REST APIs must be reacted upon.

The objective of this study was to create a proof-of-concept application that would help the target company to lower costs caused by rework, poor information management and communication, while making daily work easier and more convenient. The study showed that such system can be created by extending the capabilities of existing systems by creating a cloud-based add-on that integrates two separate systems into a connected system of systems that functions as one. This system ties existing digital tools together with the engineering process and allows an easy way to manage the use of source data.

5.2 Development process

The development of the add-on followed the chosen design science methodology fairly naturally. The insights from the environment came to a large degree from the developer itself but discussions with different interest groups verified and developed the understanding. Even though the basic idea behind the developed system was formed many design cycles were done in order fully understand the issues related to both systems to be integrated. Also, aspects of the user experience required multiple iterations. In addition to iterations in application design, a significant portion of design loops were due to personal learning in system design and application development. It can be said that even though the knowledge base formed for the target company by this add-on is fairly confined, the personal knowledge space was significantly increased. As [48] describes the three-cycle-process, the creation of the proof-of-concept consisted of dozens of *relevance*, *design* and *rigor* cycles thus making the design fulfill the objectives of the study.

5.3 Future development

Based on the results of this study, the created application will be transferred to production and will be used in a chosen future project. As this study focused on creating a proof-of-concept application, future development could focus on extending functionality of the application and redesigning the architecture to ease further development work. Future functionalities could be 1.) settings panel that would enable field and metadata mapping settings, 2.) Notification functionality, 3.) integration with CAD software Cadmatic with its newly released REST APIs, 4.) splitting software into microservices or at least split the front and backend and 5.) study of benefits its integration with commercial iPaaS solutions.

Further work and efforts could be done to extend the integration further in other highly process-wise connected domains such as CAD, CRM and ERP application. Also, broad analysis of needed tools, their integration with dedicated integrations platforms (iPaaS) and custom integration solutions and their security, performance, functionality and maintainability are topics that must be addressed in the near future.

6. CONCLUSION

The objective of this thesis was to design and develop an application that integrates the target company's document management system M-Files with its task management system Jira Cloud. This proof-of-concept was done in order to assess whether integrating two essential systems could form a coherent process and digital system that could help in planning of the work, controlling the used data and understanding the propagation of changes.

According to Nunamaker et al. [49] "*the last mile of research*" consists of three stages: proof-of-concept, proof-of-value and proof-of-use. The proof-of-concept was validated by demonstrating the feasibility of the cloud service and its potential to provide a robust way of facilitating source data management. Proof-of-value was produced by defining and creating the solution. With that a deeper understanding of the engineering processes and tasks and their dependencies, extent, effects and propagation can be obtained, thus allowing better decision-making and cost-saving. With the study, understanding of system integration and its value was improved and further activities in more comprehensive integration can be done with greater confidence. However, as the produced add-on (Node application) can be considered as fairly trivial to create and creation of such can be seen as *business as usual*, the scientific knowledge base was not mentionable extended during this study. In terms of design science research, the study managed to improve the target environment by introducing a new and innovative application and process to the target company. The proof-of-use was validated, and the weak market test was passed as the system was found operationally feasible and it will be first tested in a chosen project, and if found suitable, the system is planned to be used in all other complex projects that use Jira and M-Files as well. At this point, it is yet impossible to estimate the *return on investment* (ROI) to the produced system and its possible successors. Such estimations can be done considering, for example, previously unbillable hours, rework, degree of manual work, or ease of use in relation to invested resources put to the add-on. However, it is probably very challenging to measure such aspects accurately because of their dependencies to many other aspects. The author suggests the target company to invest in determining longer-term goals and requirements for their whole technology stack to ensure that they have a coherent and effective overall toolset to support their business.

REFERENCES

- [1] N. Ebert, K. Weber, and S. Koruna, "Integration Platform as a Service," *Business and Information Systems Engineering*, vol. 59, no. 5, pp. 375–379, Oct. 2017, doi: 10.1007/s12599-017-0486-0.
- [2] W. He and L. da Xu, "Integration of distributed enterprise applications: A survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1. IEEE Computer Society, pp. 35–42, 2014, doi: 10.1109/TII.2012.2189221.
- [3] T. Mitra, *Business-driven development*. 2005.
- [4] J. Meng, S. Mei, and Z. Yan, "RESTful web services: A solution for distributed data integration," 2009, doi: 10.1109/CISE.2009.5365234.
- [5] G. Doukidis, D. Spinellis, and C. Ebert, "Digital Transformation? A Primer for Practitioners," *IEEE Software*, vol. 37, no. 5, pp. 13–21, Sep. 2020, doi: 10.1109/MS.2020.2999969.
- [6] N. Iakymenko, A. Romsdal, E. Alfnes, M. Semini, and J. O. Strandhagen, "Status of engineering change management in the engineer-to-order production environment: insights from a multiple case study," *International Journal of Production Research*, vol. 58, no. 15, pp. 4506–4528, 2020, doi: 10.1080/00207543.2020.1759836.
- [7] S. Eppinger and T. Browning, *Design Structure Matrix Methods and Applications*. 2012.
- [8] D. Wynn, C. Eckert, and P. Clarkson, "Applied Signposting: A Modeling Framework to Support Design Process Improvement," *Proceedings of the ASME Design Engineering Technical Conference*, vol. 2006, Jan. 2006, doi: 10.1115/DETC2006-99402.
- [9] B. Manouvrier and L. Menard, *Application Integration : EAI B2B BPM and SOA*. Hoboken, UNITED STATES: John Wiley & Sons, Incorporated, 2008.
- [10] F. Losavio, D. Ortega, and M. Perez, "Modeling EAI [Enterprise Application Integration]," in *Proceedings - International Conference of the Chilean Computer Science Society, SCCC*, 2002, vol. 2002-January, pp. 195–203, doi: 10.1109/SCCC.2002.1173194.
- [11] "Jira REST API," <https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/>, 2020.
- [12] "M-Files Developer Portal," 2020. <https://developer.m-files.com/> (accessed Feb. 04, 2021).
- [13] D. Chen, G. Doumeingts, and F. Vernadat, "Architectures for enterprise integration and interoperability: Past, present and future," *Computers in Industry*, no. 59, pp. 647–659, 2008, doi: 10.1016/j.compind.2007.12.016.
- [14] J.-C. Jeon and J. Chung, "Developing a Prototype of REST-Based Database Application for Shipbuilding Industry: A Case Study," 2017, doi: 10.1109/PlatCon.2017.7883701.
- [15] J. Mathew, "SOA vs. EDA: Is Not Life Simply a Series of Events?," 2019. <https://www.confluent.io/blog/soa-vs-eda-is-not-life-simply-a-series-of-events/> (accessed Jan. 12, 2021).
- [16] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2015.
- [17] D. Chen, G. Doumeingts, and F. Vernadat, "Architectures for enterprise integration and interoperability: Past, present and future," vol. 59, 2008, doi: 10.1016/j.compind.2007.12.016.
- [18] "Marketplace," 2020. <https://marketplace.atlassian.com/> (accessed Apr. 02, 2021).
- [19] "NGINX," 2021. <https://www.nginx.com/> (accessed Feb. 22, 2021).

- [20] "Atlassian Marketplace," 2021. <https://marketplace.atlassian.com/> (accessed Feb. 05, 2021).
- [21] D. Serrano, E. Stroulia, D. Lau, and T. Ng, "Linked REST APIs: A Middleware for Semantic REST API Integration," in *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017*, 2017, pp. 138–145, doi: 10.1109/ICWS.2017.26.
- [22] "Why is ReactJS Gaining So Much Popularity?," 2020. <https://medium.com/devtechtoday/why-is-reactjs-gaining-so-much-popularity-6af4c43a3236> (accessed Apr. 08, 2021).
- [23] "React," 2021. <https://reactjs.org/> (accessed Feb. 05, 2021).
- [24] D. Badampudi, C. Wohlin, and K. Petersen, "Software component decision-making: In-house, OSS, COTS or outsourcing-A systematic literature review," *The Journal of Systems and Software*, vol. 121, pp. 105–124, 2016, doi: 10.1016/j.jss.2016.07.027.
- [25] N. Bolloju and S. Murugesan, "Cloud-based B2B systems integration for small-and-medium-sized enterprises," in *ACM International Conference Proceeding Series*, 2012, pp. 477–480, doi: 10.1145/2345396.2345475.
- [26] S. Orban, "6 Strategies for Migrating Applications to the Cloud," Nov. 01, 2016. <https://medium.com/aws-enterprise-collection/6-strategies-for-migrating-applications-to-the-cloud-eb4e85c412b4> (accessed Feb. 05, 2021).
- [27] "Boomi - Getting started," 2021. <https://community.boomi.com/> (accessed Feb. 05, 2021).
- [28] "MuleSoft Developer," 2021. <https://developer.mulesoft.com/> (accessed Feb. 05, 2021).
- [29] Y. Y. Lin, Y. Nagai, T. H. Chiang, and H. K. Chiang, "Design and Develop Artifact for Integrating with ERP and ECS Based on Design Science," in *ACM International Conference Proceeding Series*, Mar. 2020, pp. 218–223, doi: 10.1145/3388176.3388193.
- [30] Think AI, "New Enterprise Integration Services," 2021. <https://thinkaicorp.com/1new-enterprise-integration-services/> (accessed Feb. 24, 2021).
- [31] A. Hevner, "A Three Cycle View of Design Science Research," *Scandinavian Journal of Information Systems*, 2007, [Online]. Available: <https://www.researchgate.net/publication/254804390>.
- [32] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, Dec. 2007, doi: 10.2753/MIS0742-1222240302.
- [33] E. Petzold, "Introduction to Atlassian Connect," 2016. <https://blog.codecentric.de/en/2016/06/introduction-atlassian-connect/> (accessed Jan. 12, 2021).
- [34] "CData Jira ODBC Driver," 2021. <https://www.cdata.com/drivers/jira/> (accessed Feb. 23, 2021).
- [35] IFTTT, "IFTTT home page," 2021. <https://ifttt.com/home> (accessed Feb. 24, 2021).
- [36] "Marketplace App Licensing," 2020. <https://www.atlassian.com/licensing/marketplace> (accessed Apr. 02, 2021).
- [37] "Atlaskit," 2021. <https://atlaskit.atlassian.com/> (accessed Feb. 05, 2021).
- [38] "Material-UI," 2020. <https://material-ui.com/> (accessed Feb. 01, 2021).
- [39] "Customize Atlassian products with apps," <https://developer.atlassian.com/>, 2020. <https://developer.atlassian.com/> (accessed Jan. 12, 2021).
- [40] "Express," 2020. <https://expressjs.com/> (accessed Apr. 02, 2021).
- [41] "Axios," 2020. <https://github.com/axios/axios> (accessed Apr. 02, 2021).

- [42] M-Files User Guide, "Database engine and data storage," 2021. https://www.m-files.com/user-guide/latest/eng/technical_details.html (accessed Feb. 05, 2021).
- [43] E. Petzold and O. Hoogvliet, "Security of Atlassian Connect add-ons," Sep. 13, 2016. <https://blog.codecentric.de/en/2016/09/security-atlassian-connect-add-ons/> (accessed Jan. 12, 2021).
- [44] OWASP® Foundation, "Top 10 Web Application Security Risks," 2017. <https://owasp.org/www-project-top-ten/> (accessed Jan. 12, 2021).
- [45] "Visual Studio Code," 2020. <https://code.visualstudio.com/> (accessed Feb. 04, 2021).
- [46] "ngrok," 2020. <https://ngrok.com/> (accessed Feb. 04, 2021).
- [47] "Node.js," 2020. <https://nodejs.org/en/about/> (accessed Feb. 04, 2021).
- [48] J. F. Nunamaker, R. O. Briggs, D. C. Derrick, and G. Schwabe, "The Last Research Mile: Achieving Both Rigor and Relevance in Information Systems Research," *Journal of Management Information Systems*, vol. 32, no. 3, pp. 10–47, Jul. 2015, doi: 10.1080/07421222.2015.1094961.
- [49] M. Marian, "iPaaS: Different Ways of Thinking," *Procedia Economics and Finance*, vol. 3, pp. 1093–1098, Jan. 2012, doi: 10.1016/s2212-5671(12)00279-1.