



Teemu Kataja

## Systems Integration Application

Implementation of an integration and automation application for strengthening digitalisation

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master's Thesis

8.4.2022

## Abstract

Author: Teemu Kataja  
Title: Systems Integration Application  
Number of Pages: 44 pages + 3 appendices  
Date: 8 April 2022

Degree: Master of Engineering  
Degree Programme: Information Technology  
Specialisation option: Networking and Services  
Instructors: Miikka Kallberg, Development Manager  
Ville Jääskeläinen, Principal Lecturer

---

Digitalisation is a hot topic in the modern world that aims to create better services that are more available and accessible. Transaction times for anything and everything needs to be as short as possible, and usable by virtually anyone, at anytime and anywhere.

This thesis investigates the digitalisation project of one Finnish government agency. The agency deals with resource usage applications in paper forms via postal mail. They have recently acquired new digital systems for the application and archival procedures to replace their physical world counterparts. Even with brand new digital systems, the clerks must still deal with the processes of handling applications manually.

The topic of this thesis was to create a systems integration application that ties three individual information systems together and automates data flows between them. The new integration application aims to streamline the resource application process for clients, and to help ease manual work tasks of the clerks.

The product of this thesis was a brand-new software application. The application was developed to be highly available through asynchronicity and easily maintainable and deployable through containerisation and configurability. The application was designed as a central relay hub for horizontal integration model using a stateless architecture to ensure problem-free operation.

Keywords: systems integration, automation, digitalisation

## Contents

### List of Abbreviations

1	Introduction	1
1.1	Employer	1
1.2	Customer	2
1.3	Thesis Topic and Structure	3
2	Method and Design	4
2.1	Project Method	4
2.2	Integration Design	5
2.2.1	Star Integration	5
2.2.2	Vertical Integration	6
2.2.3	Horizontal Integration	7
2.3	Current State Analysis	8
2.4	Goal State	9
3	Project Specifications	13
3.1	Information Systems	13
3.1.1	Resource Entitlement Management System	13
3.1.2	Digital Document Archive	14
3.1.3	Customer Resource Management System	15
3.2	Initial Plan	16
3.3	Revised Plan	18
3.4	Integration Workflows	19
3.4.1	Submission	20
3.4.2	Archival	26
3.4.3	Decision	29
4	Application Development	31
4.1	Technologies Selection	31
4.2	Application Implementation	32
4.3	Application Deployment	35
5	Application Testing	38
5.1	Testing Utilities	38

5.2	Regression Testing	39
5.3	End to End Testing	41
5.4	Acceptance Testing	42
6	Conclusion	44
	References	45
	Appendices	48
	Appendix 1, Screenshot of REMS demo environment	
	Appendix 2, Resource application submission full workflow diagram	
	Appendix 3, Resource application archival process full workflow diagram	

## List of Abbreviations

AAI	Authentication and Authorisation Infrastructure
API	Application Programming Interface
CRMS	Customer Relationship Management System
CSC	CSC* – IT Center for Science Ltd.
CSC*	Center for Scientific Computing ( <i>legacy acronym</i> )
DDA	Digital Document Archive
ESB	Enterprise Serial Bus
E2E	End to End (Testing)
HAS	Hub-and-Spoke
HTTP	Hypertext Transfer Protocol
IA	Integration Application
NDA	Non-disclosure Agreement
OAS	OpenAPI Specification
OS	Operating System
PDF	Portable Document Forma
PEP	Python Enhancement Proposal
REMS	Resource Entitlement Management System
REST	Representational State Transfer
UI	User Interface
VM	Virtual Machine

# 1 Introduction

Up until the recent times humans have always dealt with contracts in verbal or written format. These methods of storing and passing information are falling into obscurity, as the physical and analogue world start to downshift to make way for digital services. Digitalisation is not only a hype word of the 21<sup>st</sup> century, but a steadily growing trend of adopting new technologies and mindsets to ease our everyday life.

Digitalisation aims to modernise services for businesses and individuals, and to help enterprises grow and maintain hold of their ever-growing mountains of data. Digitalisation also makes services more readily available and accessible, which is a key point in restructuring services for a global world that never sleeps. Companies can spread rapidly to provide solutions to a wider audience and individuals will find that their lives become more flexible as errands can be taken care of remotely.<sup>1</sup>

This thesis looks into the digitalisation project of one Finnish government agency, where the thesis objective is to create system integrations and automations that help bring online services to people and ease the tasks of government clerks.

## 1.1 Employer

The work conducted in this thesis was performed under the employment of CSC – IT Center for Science Ltd. (CSC). CSC is a government owned non-profit organisation operating under the direction of the Ministry of Education and Culture (the Ministry).<sup>2</sup> CSC is owned 70% by the Ministry, and 30% by Finnish Higher Education Institutions, which in turn are governed by the Ministry.<sup>3(p3)</sup>

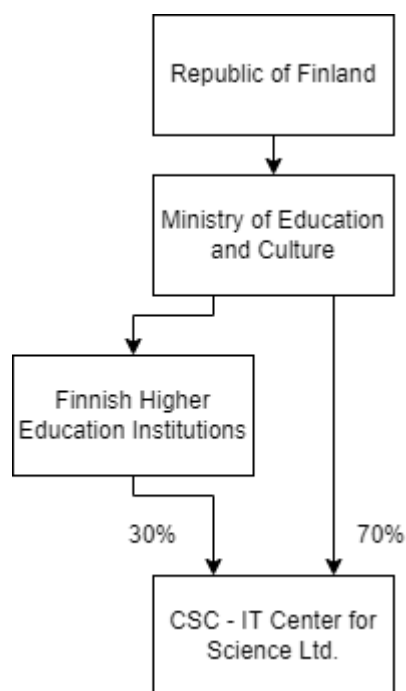


Figure 1. CSC ownership hierarchy.

CSC is a special task company that provides IT infrastructure and solutions for the needs of scientific research, education, and public administration. As an in-house company, CSC's customers (public sector agencies) can buy services from CSC directly without the need of a public procurement process, bypassing the bids from third party vendors completely.<sup>3(p3-4),4</sup> CSC's non-profit nature helps to lower the rise of expenses in the public sector, while keeping Finnish institutions competitive on an international scale.<sup>3(p11)</sup>

## 1.2 Customer

The Customer is a government agency that collects, maintains, and administers register data for research and statistical use. The Customer wishes to digitise their resource application process, which until now, has been conducted solely on paper forms via postal mail. The Digital Services Act<sup>5</sup> dictates, that publicly funded services are to be made accessible and available in digital (online) format.

In pursuit of digitising their services, the Customer has recently acquired two new information systems to bolster their application handling process: a proprietary

Digital Document Archive (DDA) and an open-source e-form service called Resource Entitlement Management System<sup>6</sup> (REMS). These two new systems are to be integrated to the Customer's existing proprietary Customer Relationship Management System (CRMS).

### 1.3 Thesis Topic and Structure

The topic of this thesis, and one of the goals for the digitalisation project, is to **implement an Integration Application (IA)** that ties the Customer's three aforementioned systems together.

Chapter 2: Method and Design focuses on the background of integration design and makes a case for the chosen integration method. The chapter also touches on the administrative side of how the project was conducted. Finally, the current situation is described as well as the goal state that the IA will establish.

Chapter 3: Project Specifications starts off with introducing the information systems mentioned in 1.2 in more detail. The initial and final plans are laid out and the chapter is concluded with detailed workflow diagrams framing the integration data flows.

Chapter 4: Application Development presents the technical design solutions of the IA on a high level. Features are abstracted and generalised for security reasons.

Chapter 5: Application Testing shows how the developed IA was tested and what were the outcomes of acceptance testing and production deployment.

Chapter 6: Conclusion closes the thesis with final remarks.



## 2 Method and Design

This chapter touches on the aspects of project culture, system integration and implementation method (integration design). System integration is a process of identifying isolated systems and their functionalities to create solutions that enable communication across system boundaries. Integrated systems are often created for the needs of automated data flows, but due to their inherent nature of adding value to existing systems, emergent properties can be found.<sup>7,8</sup>

### 2.1 Project Method

The project was carried out in an agile method because the initial plan and requirements were open-ended and not quite polished. Had the initial plan and requirements been well fleshed out, the IA could have been implemented following the waterfall model. Development began using the initial plan, which was tested, and feedback was gathered in weekly meetings from the Customer and their consultants of the proprietary systems.

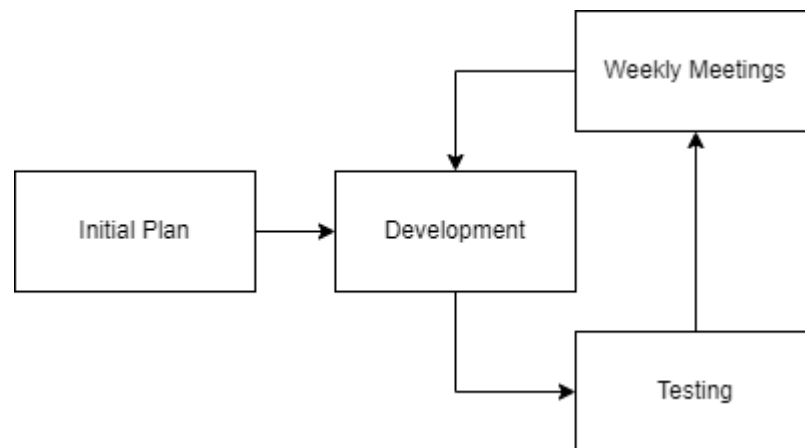


Figure 2. Agile loop for the development of the IA.

The agile software development method is a way of producing incremental changes that are then tested to get feedback that finally turns into new requirements. Agile also expects teams to be self-organised to be able to bring new ideas into the table at a rapid pace. Self-organisation in this case was

obvious as the IA has only one developer. Agile puts an emphasis on trial and error: when things are not quite clear, it's a perfectly reasonable method to test ideas out and pick the one solution that fits the case best. As such, agile is an iterative process that continually tries to improve the product.<sup>9</sup> This method is extremely powerful for creating ad hoc solutions.

## 2.2 Integration Design

Integration design is an important decision to make, as it lays out the scope of the system integration. Some methods are more invasive than others in terms of required changes to source code. Selecting the design for the job also depends on available resources, as some options are simpler, faster, and easier to implement than others. The following subchapters introduce common integration designs and their benefits and drawbacks.

### 2.2.1 Star Integration

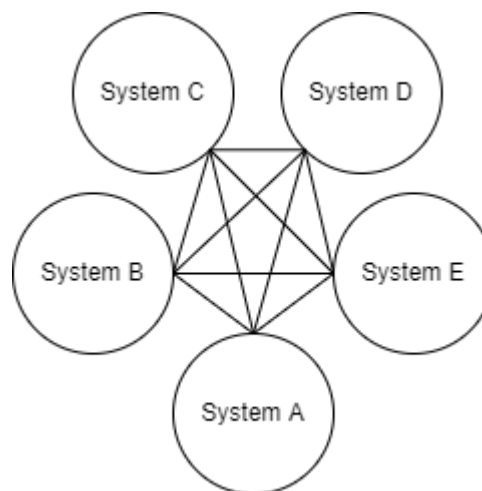


Figure 3. Star integration model where all systems are interconnected.

The star integration model, also called a point-to-point integration, is one of the earliest integration designs. Using this method all systems are interconnected to one another in a mesh of overlaying connections. Such a network is only manageable with a few systems, and promptly becomes overly complicated and unmaintainable as more systems are added. The star integration shows its

weakness when a new system needs to be integrated into an established network, as Application Programming Interfaces (API) need to be updated on every connected system. A system whose complexity increases exponentially is difficult to maintain.<sup>7,8,9(p.8)</sup>

### 2.2.2 Vertical Integration

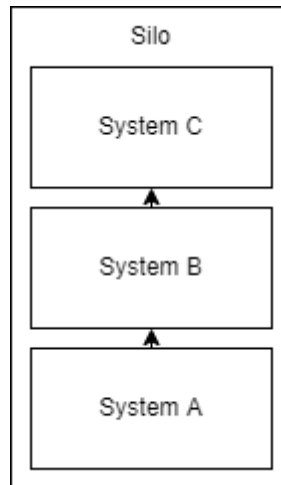


Figure 4. Vertical “silo” integration.

Vertical integration is often also called silo integration. In this integration design unrelated subsystems are chained together into an isolated block. In vertical integration, systems typically push data upwards in the silo in a way that each upper system depends on the lower system to provide data for its needs. Vertical integration may work well in cases where the systems need to work sequentially like a pipeline, but it is difficult to modify and update the silo later, e.g., in case a new system layer needs to be included in the stack. Vertical integration is therefore a one-off type of method that provides a quick, but unmaintainable solution.<sup>7,8</sup>

### 2.2.3 Horizontal Integration

Horizontal integration is divided to two general designs: Hub-and-Spoke (HAS) model and Enterprise Serial Bus (ESB) model. The common denominator of the horizontal design is the addition of a new subsystem: a message broker.<sup>7,8,9(p.10-11)</sup>

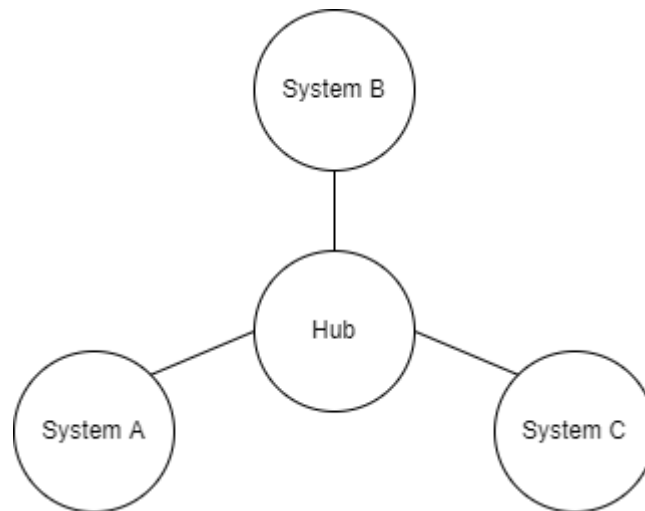


Figure 5. Horizontal integration design with HAS model. Central hub serves as a relay between systems.

The HAS model is reminiscent of the star integration model<sup>Fig.3</sup>, with a distinguishing feature being the central hub substituting the interconnected mesh-like network. The HAS model requires systems to expose APIs that can be used to push or pull data on demand. The HAS model is easily expandable, as the Hub provides a consistent API to which the systems can adhere to.<sup>7,8,9(p.10, p.20)</sup>

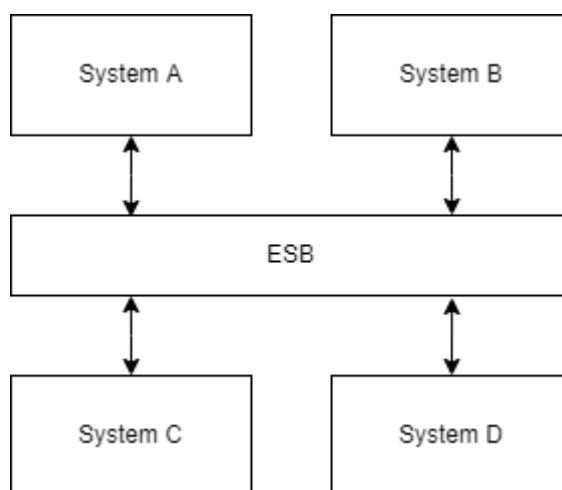


Figure 6. Horizontal integration design with ESB model. ESB provides a publisher-and-subscriber platform for services to communicate with each other.

The ESB model is a more refined horizontal integration design that builds on the base idea of the HAS model. ESB provides a publisher-and-subscriber platform to which systems can push data that the ESB makes available for other systems to consume. In this kind of design, the connected systems can potentially see data flows from each other and selectively choose which sources to use in their internal mechanisms.<sup>7,8,9(p.10-11, p.19)</sup>

The HAS model is relatively simpler to adopt for use with existing systems, as not all connected systems need to know the Hub's API. The Hub can be triggered by one system, and other systems merely provide endpoints where data can be gathered from. The ESB model is more laborious in this sense, as systems must know and adhere to the ESB's API in order to send and read data. This key difference is pivotal for the selection of the integration design for the IA and will be reiterated upon in Chapter 2.4 Goal State.

### 2.3 Current State Analysis

The current state of resource application process is extremely cumbersome for the applicant and requires manual labour from the clerks at the Customer's office. There are multiple manual steps to take on both sides of the exchange that have

been identified and analysed to be time consuming, and in need of automation to respond to the ever-growing number of applications.

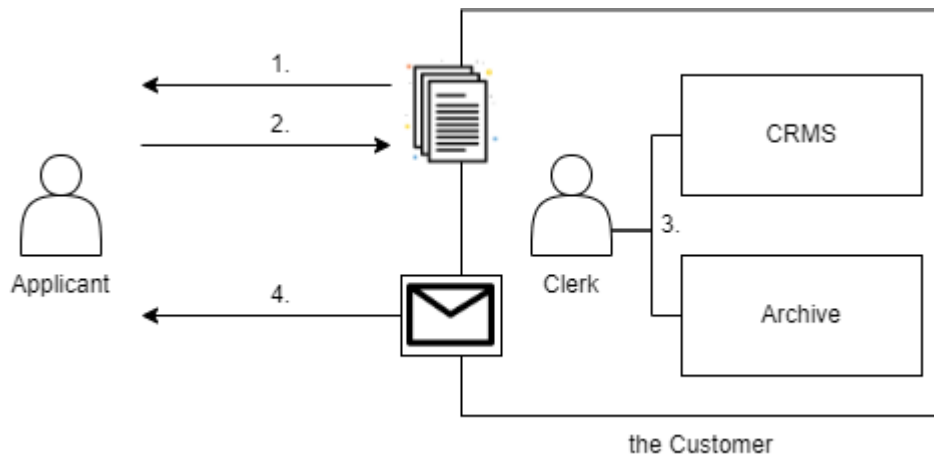


Figure 7. Current state of resource application process.

The resource application process begins with the applicant downloading a paper form from the Customer's website.<sup>Fig.7(1.)</sup> The filled paper form is then delivered to the Customer via mail<sup>Fig.7(2.)</sup> where a clerk reviews it and processes it into the relevant systems and folders for keeping track of customer relations and preservation of documents.<sup>Fig.7(3.)</sup> The decision is returned to the applicant in paper form via mail.<sup>Fig.7(4.)</sup>

## 2.4 Goal State

The thesis objective is to integrate the Customer's three information systems together, and to automate data flows between them. The chosen integration design is a **horizontal integration** using the **HAS** model. This method is advantageous, as it uses existing APIs to create channels for data flow and relieves the existing systems of needing to implement new features. This choice was possible, because not all systems need to initiate events at the IA, which means, not all systems need to know the API of the IA. A more detailed sequence diagram depicting the workflows conducted by the IA will be introduced in chapter 3.4 Integration Workflows.

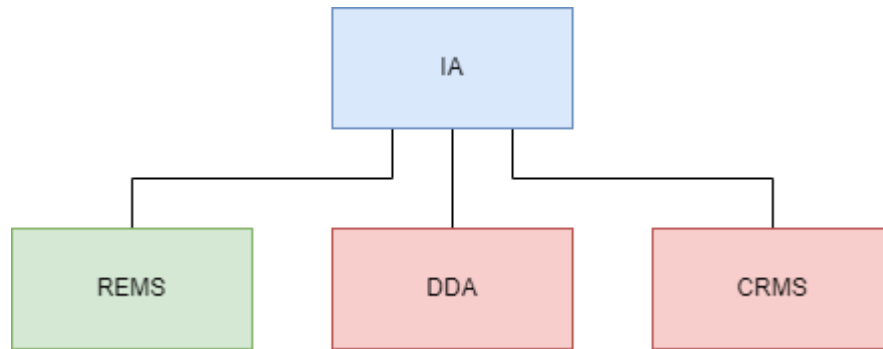


Figure 8. Overview of the information systems and the integration connections.

REMS will replace the use of paper forms for resource applications by providing extensive online tooling for creating and managing electronic forms. The filled applications and possible attachments are transferred to the DDA via the IA. The CRMS is used to keep track of applicants (clients of the Customer) and their resource permissions.

REMS (in green) is an open-source product developed at CSC, while DDA and CRMS (in red) are proprietary systems of the Customer. As such, REMS is a white box system, where it is possible to monitor the internal mechanisms and to understand the system, whereas DDA and CRMS are black box systems, of which only the APIs are exposed.

REMS will be the primary system to trigger events in the IA, while DDA and CRMS mainly serve as data repositories where data is pushed to and pulled from. Another event in the IA network is an intrinsic timed trigger that executes certain tasks within the connected systems.

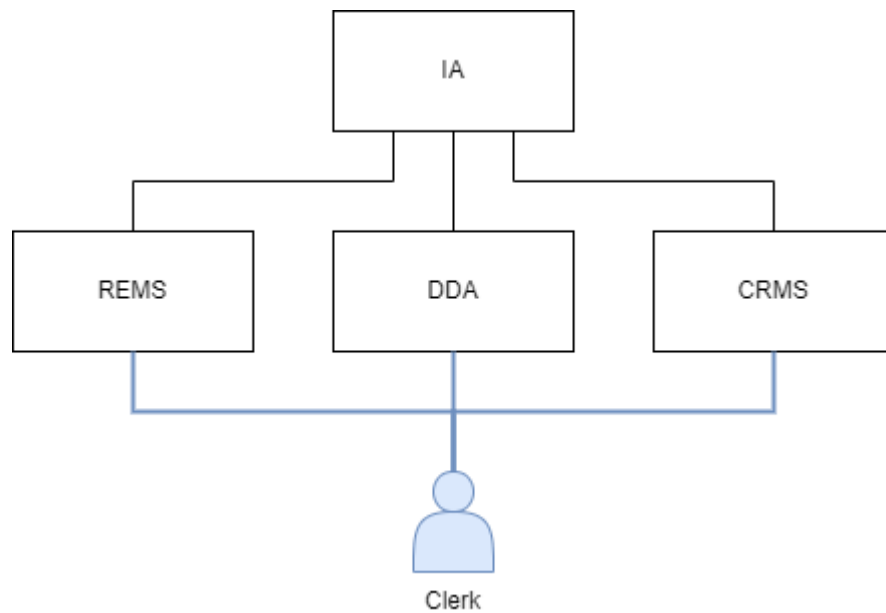


Figure 9. A clerk will oversee the systems that are automated by the IA.

The IA will integrate the information systems together and automate data flows between them. This automation removes most of the clerk's manual labour tasks, but the clerk is still required to oversee the systems, and to approve and initiate some tasks and functions.

With the new REMS and DDA systems replacing paper forms and paper archives respectively, the clerk's tasks remain mostly unchanged, only being replaced by digital and automatic alternatives. The digital nature of the systems however allows the clerks to sift through applicant details and documents using user interfaces (UI) more easily and efficiently, as they provide search functionalities which would not be available in traditional paper archive circumstances.



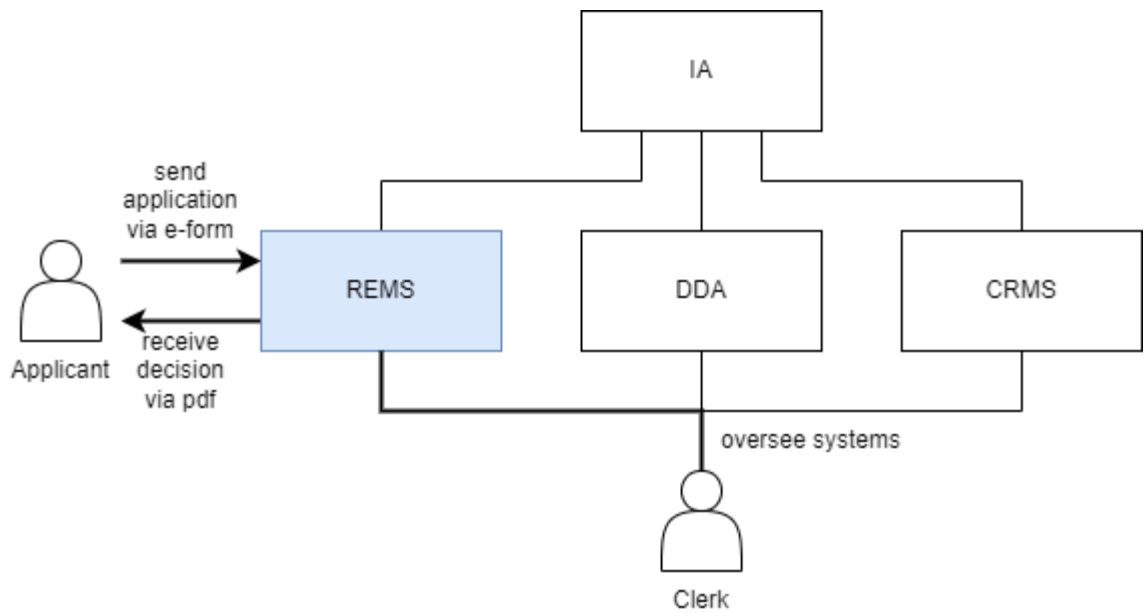


Figure 10. Single point of contact with the Customer via REMS.

When the IA and the system network is complete, REMS will become the main point of contact for the applicant to converse with the Customer. The applicant is provided with a single portal where they can send applications and receive decisions. This digital service removes many time-consuming steps from the old application process from both sides of the exchange, and additionally provides more security as well as robust transaction and history logs for audit purposes.

## 3 Project Specifications

This chapter describes the information systems in more detail and presents the initial and final (revised) plans that were received and conceived. The chapter is concluded with workflow diagrams depicting the data flows of the IA based on the revised plan.

### 3.1 Information Systems

The IA acts as a message broker between the following three information systems.<sup>Fig.5, Fig.8</sup> The systems provide extensive APIs which the IA leverages to create value and relieves the vendors of the system from needing to create point-to-point integrations.<sup>Fig.3</sup>

#### 3.1.1 Resource Entitlement Management System

REMS is an open-source product developed at CSC.<sup>6</sup> REMS was launched in 2012 and is currently living its second iteration (version 2). A screenshot of the REMS UI taken from a public demo environment can be seen in Appendix 1.

REMS is an online e-form service that was developed for permission handling of datasets. It provides organisations with a platform that contains tools for enabling secure identification of users via federated Authentication and Authorisation Infrastructures (AAI) or other trusted identity sources such as national identity providers (e.g., Suomi.fi).

Resource permissions authorised from REMS can be given end dates, which improves security by not allowing indefinite access rights to resources. Permissions can also be revoked, or users blacklisted by administrators for limiting access to data. REMS applications also contain licenses, which are digitally signed with the authenticated users' details, which provide legal validity to contracts generated online.

REMS provides an extensive OpenAPI Specification (OAS) with examples and written documentation to the internal functions, which makes REMS an extremely easy system to interface with.

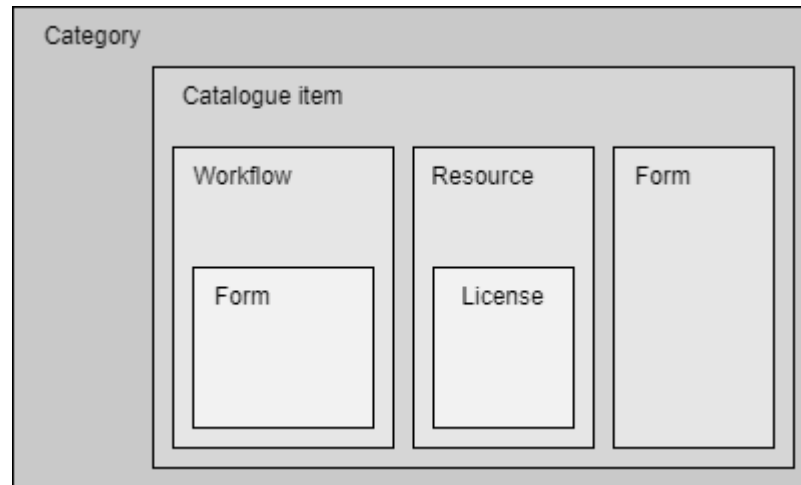


Figure 11. REMS data model.

The REMS data model consists of several vertical and horizontal hierarchies. Categories are top-level entities that hold catalogue items. Catalogue items are the e-forms that applicants select in the web UI for filling. Catalogue items consist of workflows, resources, and forms. Workflows can be paralleled to ownership, they are controlled by selected handlers, that are responsible for dealing with the applications. Workflows can hold forms of their own. Resources are links to the data that is being requested. Resources contain licenses which the applicant must adhere to in order to gain access to the resources. Catalogue items can also hold additional forms with the workflow-forms (it is possible that applicants must fill multiple standalone forms for one application, e.g., personal details, data use plan).

### 3.1.2 Digital Document Archive

DDA is provided by a third-party vendor, a supplier of the Customer. The purpose of the DDA is to replace traditional paper folder archives, and to store documents in digital format instead. The DDA is used to store all documents coming from REMS, such as the resource application document and attachments provided by

the applicants. The clerks also use the DDA to store decision documents which are responses to the applicants. The DDA is intended to be a long-term digital storage and is equipped with the same legal requirements and protections that concern traditional archives, such as viewing level privileges and disposal time limits.

The DDA is a closed source system of which only the API is exposed of. Luckily the DDA also provides an OAS similarly to REMS, but it lacks example values, which makes it more difficult to work with. Example values for the duration of the project were however provided on a separate document, but APIs are more accessible when all relevant information is readily available from the APIs themselves.

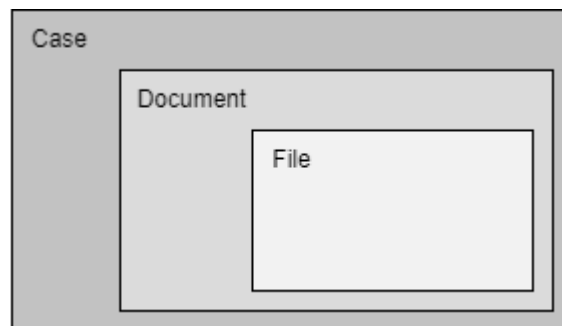


Figure 12. DDA data model.

The DDA data model is built in a hierarchical way. Top-level entities are cases that are assigned permanent journal numbers for identification. Inside cases there can be many document entities that are the real-world equivalent for folders. Inside of documents there can be many files. Files are the real-world equivalent for the actual paper forms and attachments.

### 3.1.3 Customer Resource Management System

CRMS is another system from a third-party vendor, from a supplier of the Customer. The CRMS is used to keep track of all applicants and their resource permissions. Resource applications are often valid only for the duration an individual is a member of the organisation they were in when the resource

permission was granted. There also exists an integration between the CRMS and the DDA which syncs customer information between the systems. This integration has been done prior to the IA and is vital for the intended workflows to operate.

The CRMS is another closed source system of which only the API is exposed. Unlike REMS and DDA, the CRMS does not provide an OAS, but the relevant endpoints and example values were provided on a separate document.

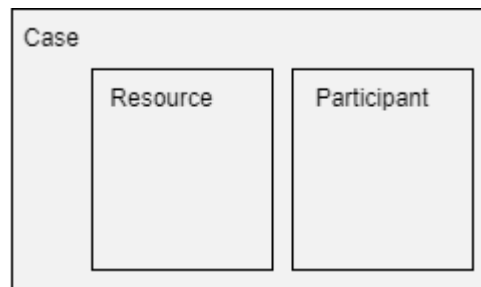


Figure 13. CRMS data model.

The CRMS data model is built in a more horizontal fashion. Like in the DDA, the case is the top-level hierarchy, but children in the case are not embedded within each other. Inside of the case there can be many resources and participants. Resources have permanent identifiers, and they represent the data the applicants are requesting access for. Participant entities are created for each member of the project group.

### 3.2 Initial Plan

The initial plan for the IA workflows at the beginning of the project were vague and not fully scoped. Implementation of the IA was however started based on the rough idea of what needs to happen and how, and the final plan was fleshed out over the course of several weeks of feedback meetings with the Customer's consultants, the vendors of CRMS and DDA.<sup>Fig.2</sup> Below is a workflow diagram depicting the logic for the data flows and directions.<sup>Fig.11</sup>

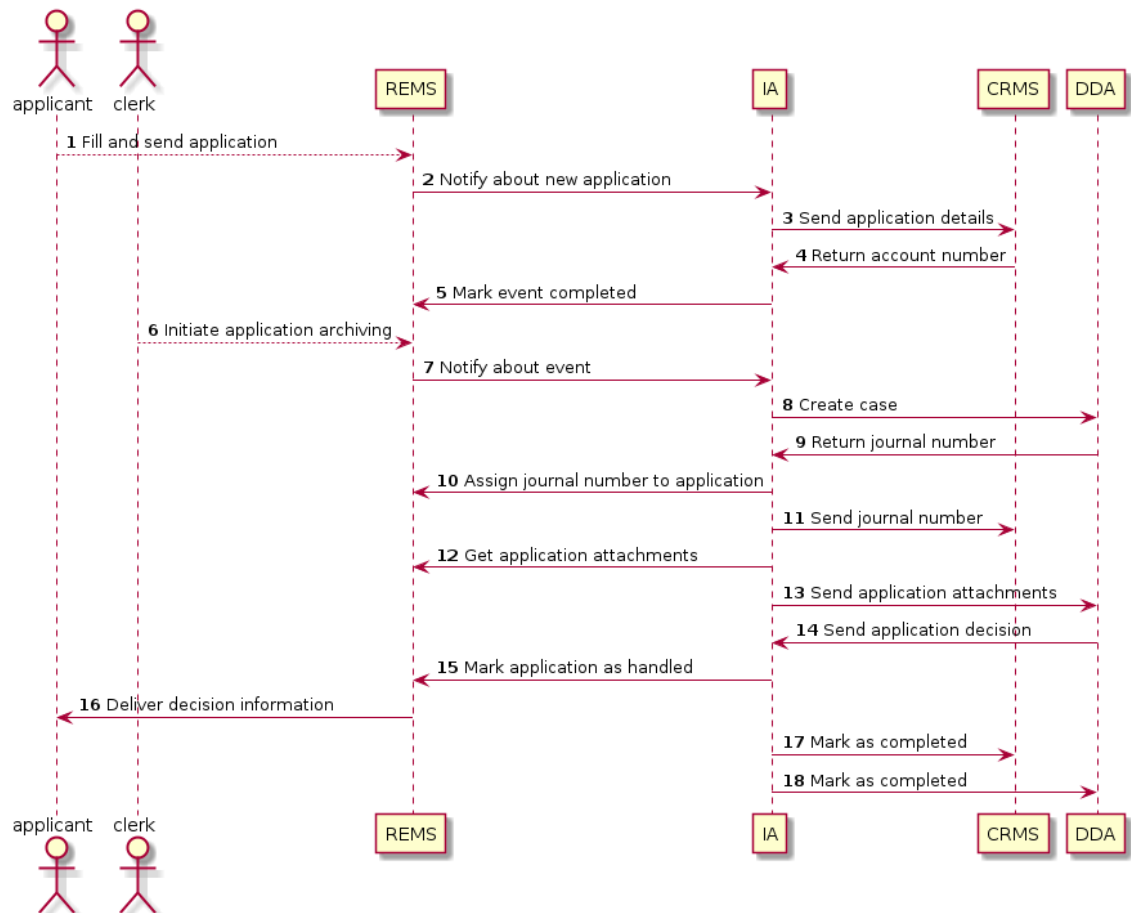


Figure 14. Initial plan for the IA as a workflow diagram.

The general idea was, that applicants would fill e-forms in REMS and send them forward to be processed. The IA would get an event notification that would start the workflow (trigger: applicant). Details from a new application were to be extracted and sent to the CRMS for registration, upon which an account number was to be returned to the IA that was stored to REMS with an event completion message. Fig.14(1-5)

A clerk would then visually verify the e-form and start an archival process. The IA would get a second event notification that would start the workflow (trigger: clerk). The IA would create a new case in DDA, which would return a journal number for bookkeeping. This journal number would be returned to REMS and CRMS where it would be assigned to the application. Next the IA would request attachments from the application and send them to DDA for archival. Fig.14(6-13)

The final workflow is the decision process. The IA would get a third event notification that would start the workflow (trigger: DDA). The IA would forward the decision to REMS and mark the application as handled. REMS in turn would inform the applicant of a new decision on their case. Finally, the IA would close the case in both CRMS and DDA. <sup>Fig.14(14-18)</sup>

Considering this workflow diagram with greater analysis, we can see, that it clearly consists of three separate events and workflows. These events would later be framed in more detail; (1) application submission, (2) application archival, (3) decision returning. The revised plan is introduced next in Chapter 3.3 Revised Plan, and more detailed workflow diagrams for each stage are shown in the following Chapter 3.4 Integration Workflows.

### 3.3 Revised Plan

Further development of the IA revealed that there are different types of applications sent from REMS that are to be processed in separate workflows. The table below lists three application types that were identified: resource applications, non-disclosure agreements (NDA) and other, unsupported, types of applications.

Table 1. Destination matrix of REMS applications based on application type.

<i>Application type</i>	<i>CRMS</i>	<i>DDA</i>
<i>Resource application</i>	yes	yes
<i>NDA application</i>	no	yes
<i>Other applications</i>	no	no

Resource applications are the main type applications. They are used for gaining permissions to use restricted access data. Resource applications are registered to both CRMS and DDA. The CRMS keeps track of client (applicant) details and DDA is used to store relevant documents in the application process. NDA

applications are always required from each participant, they are not registered to CRMS, as the participant information from the resource application already contains this data. NDAs are however stored in DDA for legal purposes. Other types of applications are not handled by the IA at all, they are processed manually in REMS by the clerks.

The third event, (3) decision returning<sup>Fig.11(14)</sup>, was found to be intended to be launched internally by the IA as a timed job, and not by the DDA. Other aspects of the initial plan remained largely the same, only that more comprehensive actions would be found out during the development. These more detailed steps are shown in the next chapter's each sub chapter representing the different workflow types.

### 3.4 Integration Workflows

Author's notes:

- I. This chapter is highly technical and covers the IA workflow diagrams in great detail. Explanations of steps are written below the figures for offering understanding on the process.
- II. Workflow diagrams have been simplified to contain short sentences. Endpoints and authentication protocols have been redacted for security reasons. Some minor (internal processing) steps have been omitted for brevity.
- III. Some workflow diagrams consist of so many steps, that it would not be feasible to show the full workflow in a single figure on one page. In these cases, the workflow diagram has been split up into sections, and the full unsliced diagram has been added into the appendices.



### 3.4.1 Submission

#### Common Characteristics

A new case always begins with an application submission from REMS. No other workflows can be initiated, if the relevant metadata created by the submission workflow doesn't exist. It's important to notice in this chapter, that all submission workflows begin with the same 5 steps:

1. An applicant fills an e-form in REMS and sends it forward,
2. REMS sends an event to the IA regarding a new submission,
3. IA requests the application data from REMS,
4. REMS returns the application data to IA,
5. IA parses the application data and decides which workflow, if any, to initiate based on what kind of application it detected.

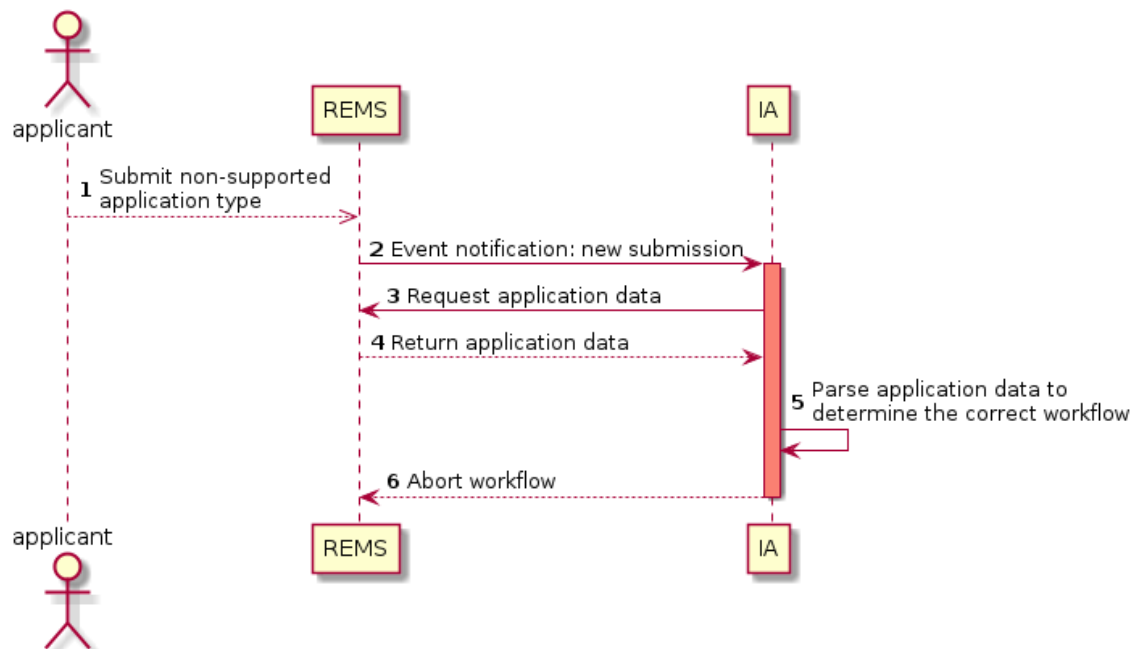


Figure 15. Workflow diagram for unsupported application type submissions.

## **Unsupported Applications**

When an applicant submits an application that is of other type from a resource or an NDA application, the application gets rejected by the IA, and the workflow is aborted.<sup>Fig.12(6)</sup> This process happens silently, and the applicant or clerk are not aware of the behind-the-scenes actions. These kinds of unsupported applications are handled directly in REMS by the clerks, and as such, are not part of the IA's agenda, and will not be discussed further.

## **NDA Applications**

NDA applications are the simpler kind of the two application types that get processed by the IA. NDAs are signed by applicants prior to resource applications, and they are always required in tandem. NDAs are valid forever, and as such, are only required to be filled once. Old NDAs can be reused in future resource applications.

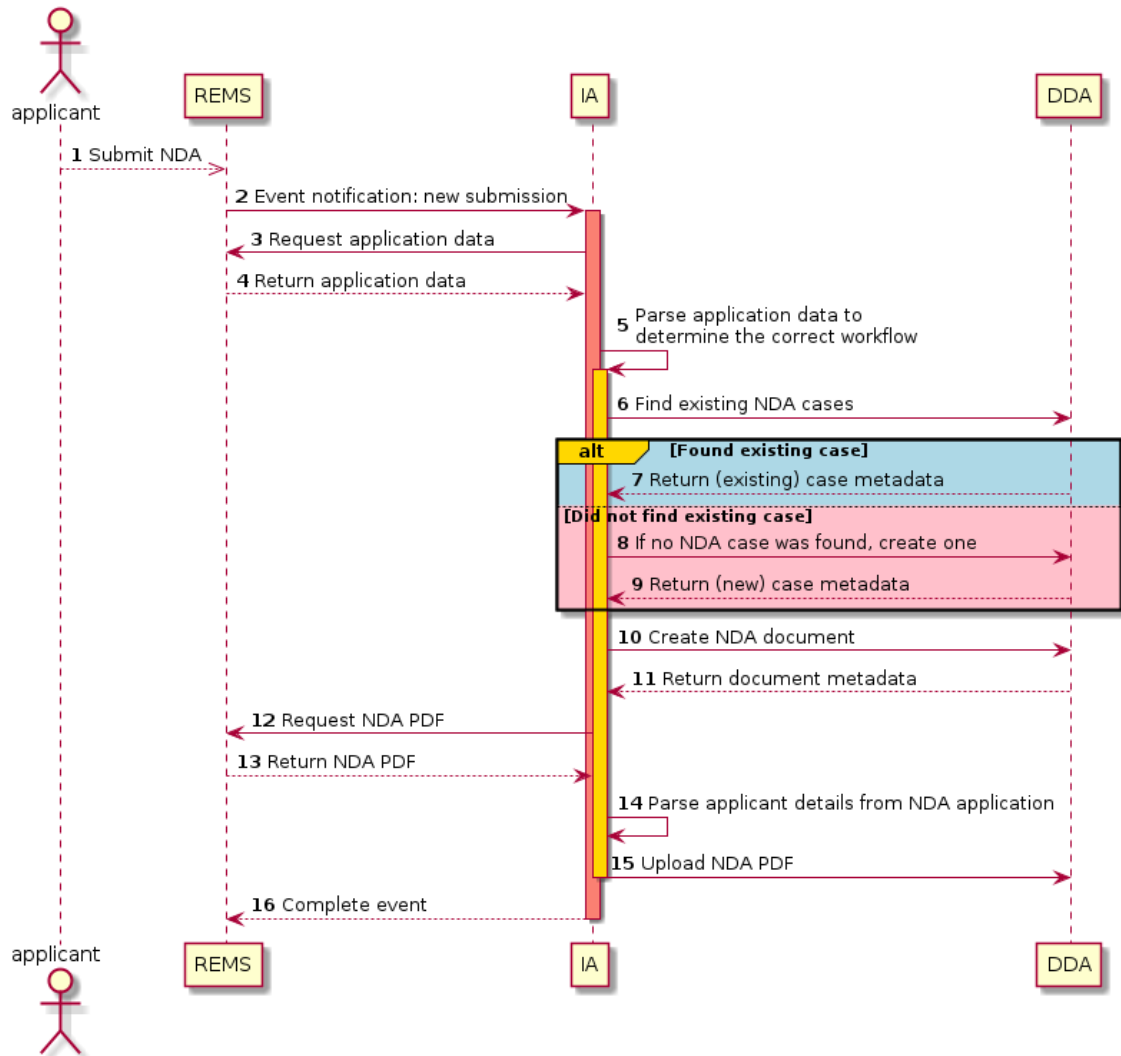


Figure 16. Workflow diagram for NDA application submissions.

When an NDA application is received at the IA, the IA checks from DDA if there is an existing NDA case where the application can be filed into. NDAs are filed in annual cases. If an existing case was found, a new document entity is created in that location. In other cases, a new case is first established for the year.<sup>Fig.16(6-11)</sup> After the initial DDA operations the NDA application and signed licenses are downloaded from REMS in Portable Document Format (PDF). These are uploaded to DDA in a file entity.<sup>Fig.16(12-15)</sup> The workflow is completed by sending a success message to REMS.<sup>Fig.16(16)</sup>

## Resource Applications

The resource application submission workflow consists of so many steps, that for reading assistance, the workflow diagram has been split to 3 parts, and each section will be covered sequentially. The complete, unsliced, workflow diagram can be seen in Appendix 2.

A resource application submission begins in a similar fashion to NDA submissions, with one distinct caveat: instead of interfacing with DDA, the IA first interfaces with CRMS.

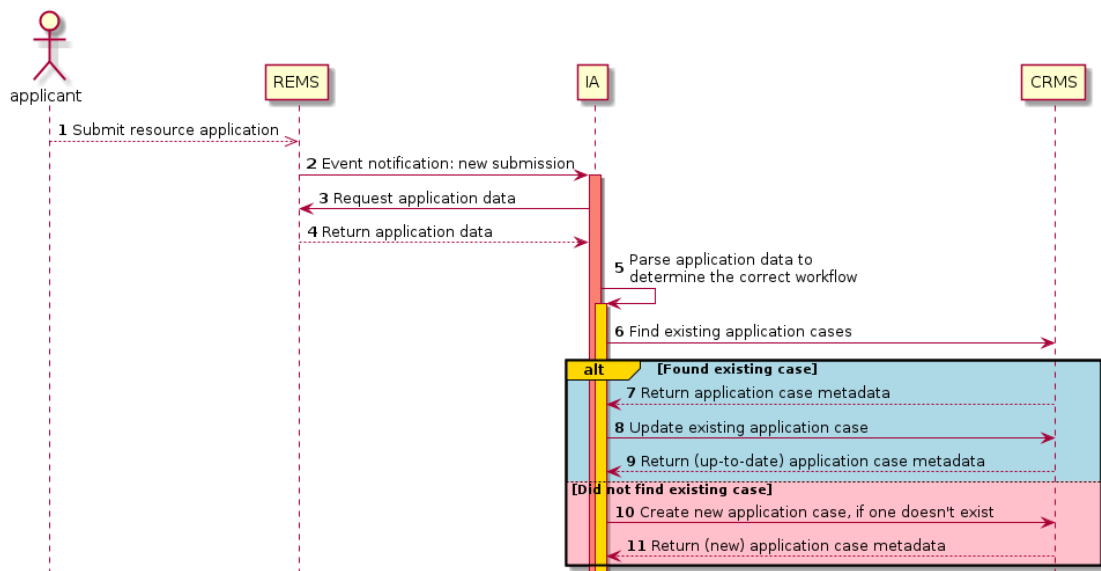


Figure 17. Workflow diagram for resource application submissions, part 1/3.

The IA checks from CRMS if a case has already been created for the application. If an existing case was found, then the case details are updated with changed information from the REMS e-form (application data). If there are no existing cases, then a new one is created. Fig.17(6-11)

The next step in resource application submissions is to handle the resources. The resources contain technical identifiers and usage licenses that describe what data the applicant is requesting access for, and what are the terms of use.

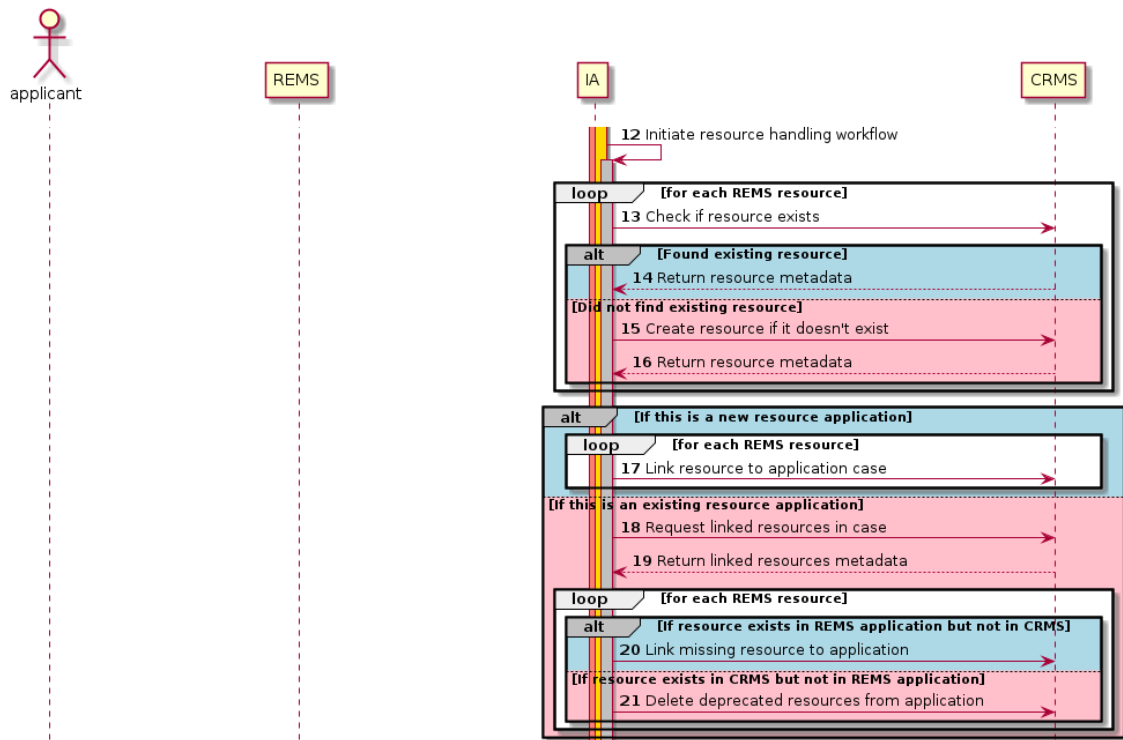


Figure 18. Workflow diagram for resource application submissions, part 2/3.

For each resource in the REMS application, the IA checks from CRMS if the metadata already exists. If it does, the metadata is returned, in other cases, the IA generates the necessary metadata from the REMS resource for later use (different applicants can request access to the same data). Fig.18(12-16) Next the workflow diverges based on if it's an existing application that is updated, or a totally new application submission. New application submission handling is simple: resources are linked to the case, and then this step is complete. Fig.18(17)

For existing application submissions there are some additional steps to be taken. First the existing resources that are linked to the CRMS case are requested, then for each resource entity in the REMS application, the IA checks and matches resources that are on the application, but not in CRMS, and vice versa (resources that are in CRMS, but not in REMS). The resources that are missing from CRMS are linked to the case, and resources that were found from CRMS but not from the REMS application, are removed from the case, because they are deprecated. Fig.18(18-21)

The final step in resource application submissions is to handle the participants. The participant entities contain personal details such as the applicant's name and organisation affiliation, and means of contacting (phone, email, etc.).

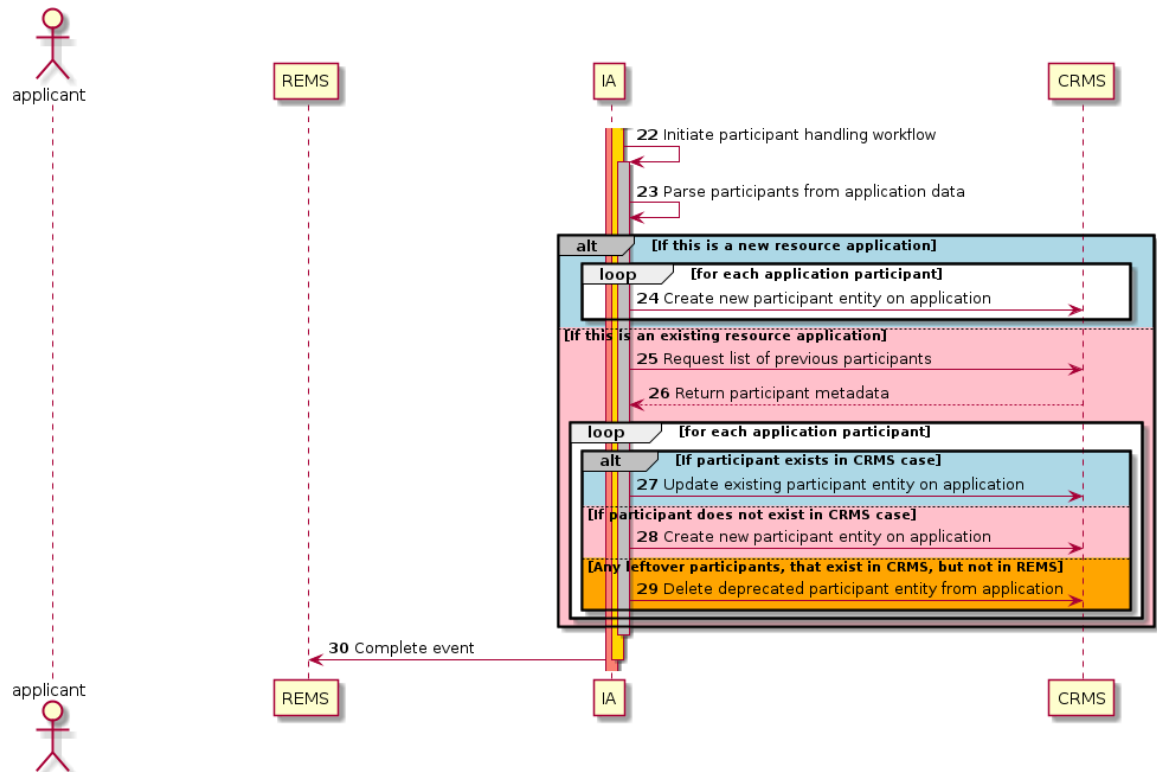


Figure 19. Workflow diagram for resource application submissions, part 3/3.

The participant processing is performed similarly to the resource handling, but whereas resources are distinct entities in the REMS application, the participant information must be parsed from the structured data within the REMS e-form. Fig.19(22-23)

For new resource application submissions, a new participant entity is created in CRMS. Fig.19(24) For existing cases the process is again slightly longer. First the existing participants are requested from CRMS, and then for each participant listed in the REMS e-form, if a participant exists in the CRMS case, the existing personal details are updated. If a participant does not exist in CRMS, a new participant entity is created for them. And finally, any other participants are

removed from the CRMS case, because the REMS application represents the current situation. Fig.19(25-29)

### 3.4.2 Archival

Similarly, to the resource application submission, the archival request workflow consists of so many steps, that for reading assistance, the workflow diagram has been split to 3 parts, and each section will be covered sequentially. The complete, unsliced, workflow diagram can be seen in Appendix 3.

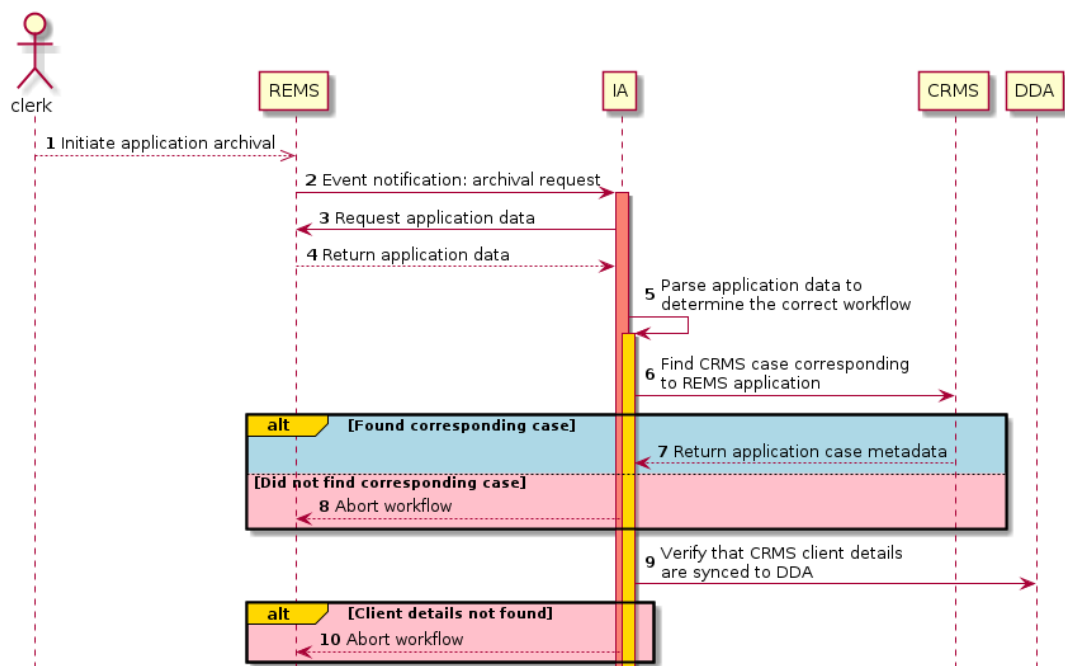


Figure 20. Workflow diagram for archival request, part 1/3.

Contrary to previous workflows, an application archival workflow is initiated by a clerk, and not by an applicant. A clerk will visually approve the filled application e-form in REMS, and either return it for correction, or approve it for the archival workflow. Fig.20(1)

The archival workflow begins with the IA looking for an existing CRMS case corresponding to the identifier received in the event notification from REMS. If no case metadata is found, the workflow is aborted, and the clerk is notified of the

issue. The archival process depends on the proper execution of the submission workflow prior to the archival request. Fig.20(6-8)

Between the CRMS and DDA there exists a data integration that syncs up client details. After receiving case metadata, the IA attempts to verify, that the client details received from CRMS exist in DDA. In case they don't, the workflow is aborted, and the clerk is notified of the issue. Fig.20(9-10)

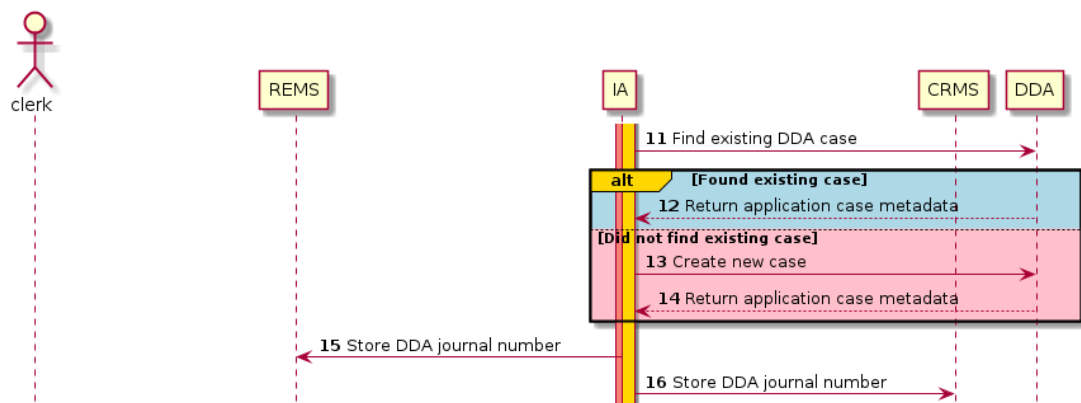


Figure 21. Workflow diagram for archival request, part 2/3.

Next the IA looks for an existing application case from the DDA. If a case was found, the metadata is returned for later use. If no existing case was found, then a new archive case is issued. Archive cases are legal entities, and they hold a permanent identifier “*journal number*” which is used to refer to the case. The journal number is stored to both REMS and CRMS for book-keeping. Fig.21(11-16)

The next and final steps of the archival request describe the actual archiving process of documents and files.



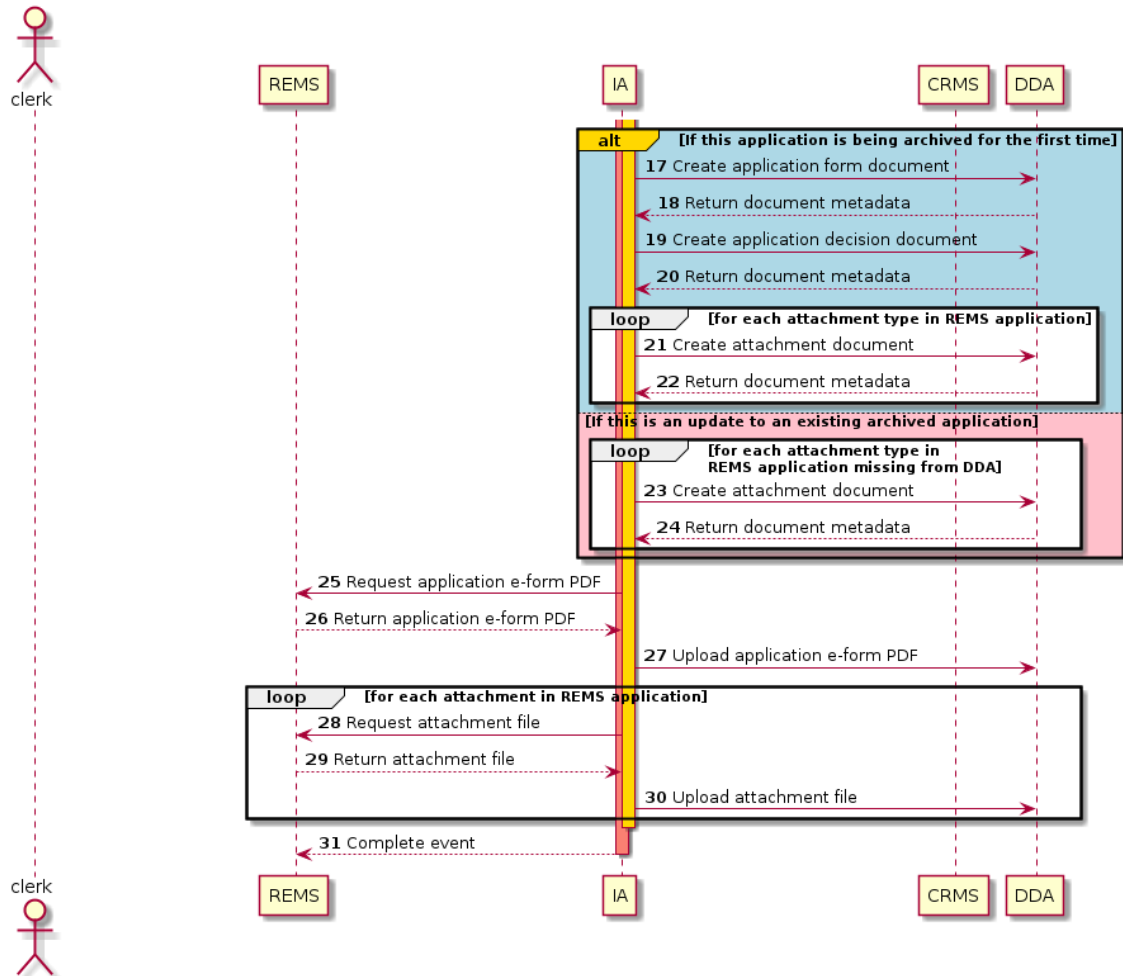


Figure 22. Workflow diagram for archival request, part 3/3.

If an application is being archived for the first time, new documents must be created. Documents in DDA represent folders in the physical world and they can hold multiple files inside of them. The IA begins with creating a document slot for the application e-form, there is always only one application e-form coming from REMS. The second document slot created is for the application decision. This document is left empty, and it will be filled in by the clerks. When an application has been approved, a digitally signed file is uploaded to the decision document. Finally, for each attachment type, a document slot is created. As of the writing of this thesis, there exists two types of attachments: public and confidential. Public attachments are as their name suggests, in public domain. Confidential attachments are handled with more care, and access to them is limited. Fig.22(17-22)

For existing cases only missing attachment documents are created, as application and decision documents are always created regardless of the application contents. Fig.22(23-24)

Next the application e-form is downloaded from REMS and uploaded to the document slot in DDA. After this, each attachment is downloaded and uploaded to the correct attachment type documents. As mentioned before, the decision document will be left empty at this stage. Finally, the event is marked as completed to REMS. Fig.22(25-31)

### 3.4.3 Decision

The final workflow works slightly differently from the previous ones, in that, it's not initiated by human interaction, but by a timed (cron) job. This automated workflow is responsible for closing the previously opened cases.

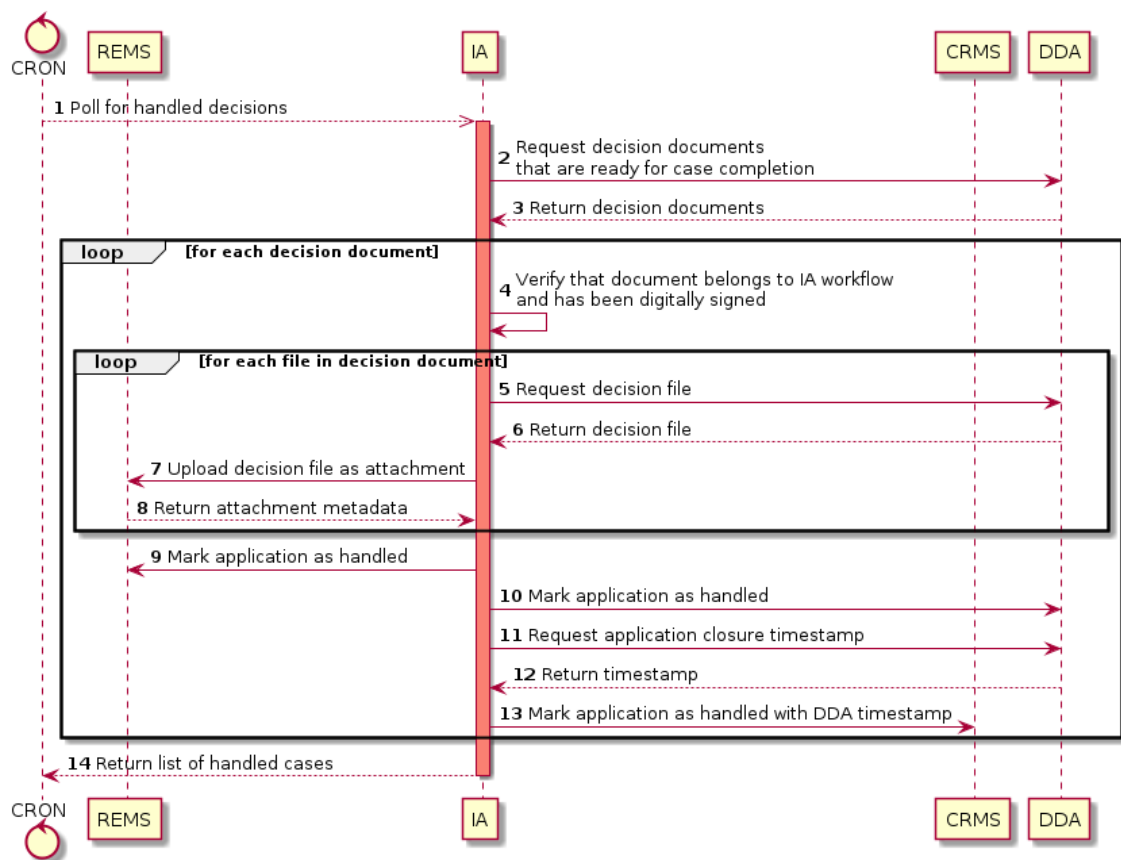


Figure 23. Workflow diagram for requesting decision documents.

When the timed event is fired, the IA requests decision documents from the DDA that are ready for further processing. Each decision document is verified, that it belongs to the IA workflow and that the decision document has been digitally signed, any other documents are discarded from the workflow.<sup>Fig.23(1-4)</sup>

For each file in the decision document, the file is downloaded from DDA and uploaded to REMS as an attachment to the application e-form. This is the method which is used for delivering the legal decision documents to the applicant via REMS.<sup>Fig.23(5-8)</sup>

Finally, the application is marked as completed in each system. When the application is closed in REMS, the applicant is notified, and they will find the decision document from their application. Next the case is closed in DDA, which produces a closure timestamp. This timestamp is retrieved from DDA and sent to CRMS along with a case closure request. Lastly, the list of processed decisions is returned as a response to the job manager.<sup>Fig.23(9-14)</sup>.

## 4 Application Development

The product of this thesis (the IA) is an HTTP web service that exposes a REST API for making requests for carrying out tasks. HTTP stands for Hypertext Transfer Protocol, and it dictates how requests and responses are conducted between servers. REST stands for Representational State Transfer and it provides guidelines and constraints for building a web service in a deterministic way to ensure interoperability and stability. These details are industry standards and don't require further investigation.

The following subchapters provide insight into the technological selections and implementation details of the IA.

### 4.1 Technologies Selection

While starting the development of the IA, there were two options for the programming language selection based on use case and previous experience: Python and Go. Both are high level programming languages that can be used to solve the issue in this project: to develop a web service that complies with HTTP and REST principles.

Python is a multipurpose programming language that can, and has been, used to create anything and everything. One of its strong points are its simple syntax that is close to reading English text. Due to Python being so simple to understand it's a popular language for learning programming. Python is a dynamically (or loosely) typed language, that allows variables to change meaning during runtime. Due to the flexibility of Python it's a very convenient prototyping tool and has great developer productivity aspects from the quick development-loop of writing code, to testing, and getting feedback.<sup>11(c.1.2), 12</sup>

Go is also a multipurpose programming language, however, it's younger than Python, and was developed with a single purpose in mind: systems programming. Go was developed by Google to support their expanding cloud service needs. It's a statically typed and slightly more convoluted language, not as beginner friendly

as Python. The great thing about Go for web service development is, that it was developed with concurrency in mind. Web services need to be able to serve multiple requests and responses at the same time, and this feature has been built into the core of Go. The same functionality in Python requires the use of special asynchronicity packages.<sup>13(c.1), 14</sup>

Python is an extremely popular language due to its simplicity and availability of premade packages to be used in nearly any use case. According to Statistics Times, Python's global share of programming projects in 2021 was 30.21%. Go's share for the same time period was a meager 1.52%.<sup>15</sup>

Had I been given the authority to decide the language of implementation for the IA, I had gone with Go for its superior performance, stability, scalability, and use case specialisation. However, similarly to the global share, the competence at CSC strongly favours the use of Python over Go. As such, Python was the language of choice, and improvements on it had to be made using additional packages to build functionality over the base features.

## 4.2 Application Implementation

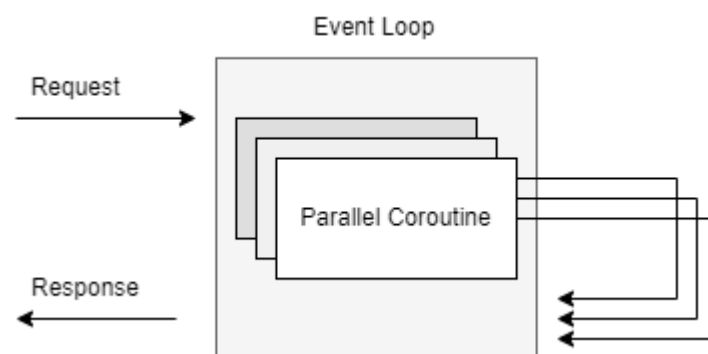


Figure 24. Event loop handles requests and responses asynchronously. Parallel non-blocking coroutines carry out tasks.

## **Asynchronicity**

Python is traditionally a synchronous programming language, in that, it executes orders procedurally one after the other. To be able to serve multiple requests coming to the web service simultaneously, an additional `asyncio`-package had to be leveraged. `Asyncio` provides new features to Python in the form of an event loop, coroutines, and futures.<sup>16</sup> Using these features the web service can simultaneously receive requests and send back responses without blocking the processing of other clients.<sup>Fig.24</sup>

## **Managing a Growing Codebase**

When starting with the implementation, the dynamic typing nature of Python was convenient for rapidly testing different use cases, and iteratively building the application to work as intended. However, as the project grew, I started to have trouble in remembering the contents of variables being passed around from function to function in the program. In Go this would have been a non-issue, as the statically typed variables would self-document their contents and purpose. For Python, optional type hints can be built using the `Typing`-package.<sup>17</sup> This brings similar constraints to Python that exist in statically typed languages, where a program refuses to build if conflicts are detected. Adding typing to large Python projects is virtually mandatory, as the vague nature of missing variable and function types will make a project unmaintainable as the codebase grows.

## **Stateless Architecture**

When designing a web service, a decision must be made for state and data management. If a service stores data and operates according to the shape of the data, then it's a stateful service. A stateful service may behave differently from time to time depending on the form of data it's storing. Another style, in a stateless service, is to store no data. A stateless service behaves idempotently; it always does the same thing when executed, as it is operating independently from the context of previous data. This style of architecture greatly simplifies a program in both operation, deployment, and maintenance.<sup>18</sup>

The IA's architecture style is a stateless web service. This decision was possible because the subsystems in the HAS-model, where the IA serves as the central relaying hub<sup>Fig.5, Fig.8</sup>, are already equipped with databases of their own that store all relevant information needed for their operation. This reality relieved the IA from needing to have its own database for tracking transactions, as the systems are bolstered with APIs that allows the IA to push and pull contextual data during its workflows. As the IA has no database of its own, there are no states that can get jumbled up in confusion and later hamper its operation. Each request to the IA starts from nothing, and context is gathered from the subsystems' databases.

### Configurability

An important aspect in creating long lasting software is to make them configurable. With a great amount of configurability, a program can be tweaked using simple text files instead of making code changes, that then require new versions to be built and deployed.

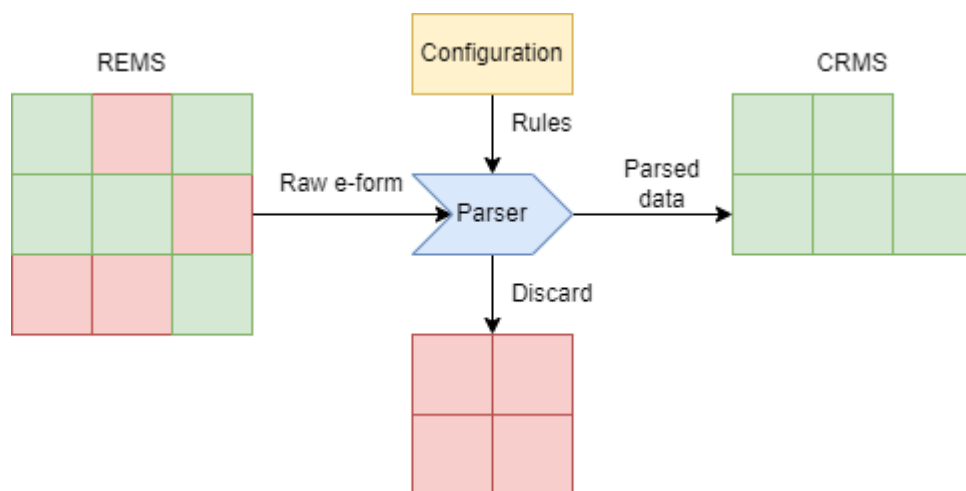


Figure 25. REMS e-form parser.

One of the key features of the IA is extracting specific data from REMS application e-forms to be pushed into CRMS. Application e-forms can be updated on demand as needed, as opposed to traditional paper forms. The IA must be able to respond to changes swiftly, and so, an intricate e-form parser was developed as part of the application submission workflow. The e-form parser takes the REMS e-form

contents and a configuration file as input and parses the raw data into the desired format in which the CRMS can process it. Fig.16-19, Fig.25

### 4.3 Application Deployment

There are many ways for applications to be deployed to production with. The simplest method that was used in the past was to just start the application on some machine and expose its port to the internet. This method is not very scalable of course, since the application that was being run on a single machine would consume resources directly from other processes and the operating system (OS) itself. It was also cumbersome to start new replicas on separate machines, as doing so required manual labour each time for setting up the environments. Since then, new virtualisation techniques have been developed that allowed proper resource handling to be performed on the host hardware.<sup>19</sup>

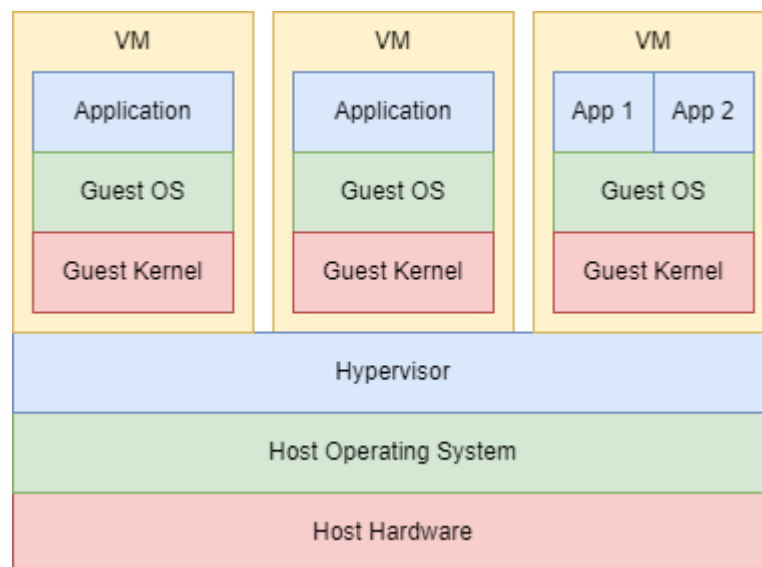


Figure 26. Virtual machine deployment scheme.

Virtual machines (VM) were the first step to improving the scalability of deployments through the emulation of physical computers using software called a hypervisor. The hypervisor is a program that administers these virtual environments. VMs contain all the same elements as a normal physical computer, but instead of being run on dedicated hardware (motherboard and processor),



the process is emulated in a virtual environment that is running on top of another, existing, OS, with its own hardware. VMs also provided isolation and environment separation which increased the security of applications, as access and visibility could be restricted to the virtual environments only. Another positive aspect that came from virtualisation was the possibility to automatically provision new services in order to scale deployments horizontally.<sup>19, Fig.26</sup>

VMs, while being a step in the right direction, are not without issues themselves. Running them requires allocating static resources to the individual VMs, so that even if the emulations don't require all the resources they are allocated, they are still reserved for those VMs, and can't be utilised by other processes. A more granular approach has been developed with the use of containers, that provide a more minimalistic approach to virtualisation.<sup>19</sup>

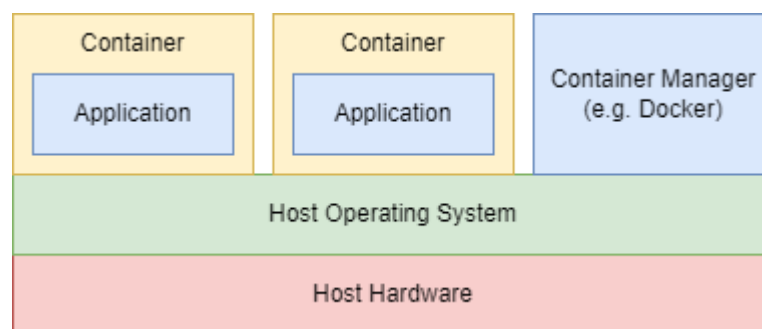


Figure 27. Container deployment scheme.

While VMs emulate a complete physical computer environment, containers are a step down in complexity: they aim to only provide an isolated environment from the host machine where a single application can be run without interference. Because containers have the possibility of sharing the host OS and kernel, their resource management is much more powerful. Containers can be configured to only use the resources they need at a single moment. When a container application is not processing anything, the resources are freed back to the host system for use by the host OS, or other containers. This feature allows more applications to be run on dedicated hardware compared to the heftier VMs.<sup>19</sup>

Containers are administered by a container manager runtime, of which one of the most popular ones, also the one used in this project, is Docker. Docker provides extensive tooling for building and deploying applications while using minimal host system resources. The IA was built using a multistage docker image that packs the business logic into a very small image, that can then be run in an application container. The size of the image is in the magnitude of dozens of megabytes, while VM images tend to range in the magnitude of hundreds of megabytes to several gigabytes.

## 5 Application Testing

Testing is a mandatory step in software development in order to create quality code. Feathers<sup>20</sup> suggests, that any code without tests is automatically legacy code, and in order for code to be of good value, it is to be well maintained, which includes having rigorous test cases.<sup>20(c.2, c.7)</sup> The operation of the IA is intended to last indefinitely, thus, it also requires a set of testing tools to aid in development as well as automated regression tests (introduced later in Chapter 5.2) to help with feature development and other code changes.

### 5.1 Testing Utilities

Proper testing of software often requires tools of convenience to aid in mimicking real world scenarios or to ease laborious manual tasks. During the development of the IA, two helper tools were birthed in tandem: a tool for copying REMS resources from one environment to another, and a tool for mocking digital signatures of electronic documents in DDA.

#### **REMS Copy Tool**

*The intricate data model of REMS was introduced in Chapter 3.1.1 with Figure 11.*

Software testing should always be carried out in testing environments which are separated from the real production environments. For testing REMS functionalities, it is beneficial to create an initial testing environment where everything is laid out and all aspects are tested rigorously. After tweaking and fine tuning, the resources can then be copied over to a production environment. This method of testing allows the production environment to stay free of the clutter of deprecated items that were rejected during testing.

The amount of REMS resources varies from user to user, but they are often quite numerous. In the case of the Customer, their catalogue contains over a hundred different resources, not to mention their supporting forms, licenses, and

workflows. In total, there are so many items, that transferring them between the testing and production environments every time new versions are created, would be too time consuming to be done by hand. For this purpose, a REMS copy tool<sup>21</sup> was created that can be used to copy only the changes between environments, similarly to how git handles code changes.

### **DDA Digital Signature Mocking Tool**

When the Customer's clerks handle cases, the end result is always a legal decision document, which must be digitally signed by the clerks to ensure authenticity and validity. The signature is verified by the IA during the decision retrieval.<sup>Fig.23(4)</sup> The digital signing and certification of documents is done in the DDA, however, this feature is missing from the test environment, so a tool had to be devised to help the clerks play out the full workflow in the test environment. The tool mocks the digital signature process, so that the decision document may travel onwards from the DDA, via the IA finally to REMS.

## **5.2 Regression Testing**

Regression testing is a method of quality assurance for detecting and preventing bugs from being introduced with new features and code changes. A regression in a codebase happens when changes are introduced and something that used to work before now suddenly doesn't anymore. Regression testing is usually an automated process that can be executed quickly to verify that small incremental changes have not broken the software. When breaking changes are introduced, it requires updating the tests cases accordingly.<sup>20(c.2), 22</sup>

### **Unit Testing**

The characteristics of unit tests are small and isolated test cases that test a single functionality in a code, for example, the execution of a single operation or a function call. The advantage of unit tests is that they are often quick to run, and they can swiftly test multiple different cases of a function's operation. The disadvantage is, that their number is often large, and updating a small piece of

code requires updating many test cases, however, the value they bring in preventing bugs is often much greater. Unit tests are usually used to calculate a testing coverage of the codebase, which tells the amount of code that has been run and verified. This value can be used as a metric of how well maintained a codebase is, and it is generally said that programmers should strive to maintain a coverage of over 80%.<sup>20(c.2)</sup>

It is important to understand about testing code functionality, that tests should not only be made for ideal cases where the operation works perfectly from start to end, but also to verify, that error cases are handled appropriately.

Table 2. Codebase metrics of the IA (calculated with Linux command *find* and python package *unittest*).

<i>Category</i>	<i>Metric (n)</i>
<i>Business logic (codebase)</i>	2082 lines
<i>Testing logic</i>	2176 lines
<i>Test cases</i>	128
<i>Configuration files</i>	690 lines
<i>Statements</i>	1076
<i>Tested statements</i>	1048
<i>Missed statements</i>	28
<i>Coverage</i>	97%

The business logic of the IA (the codebase) contains 2082 lines of code, including comments. The business logic relies on configuration files, which add another 690 lines on top of it. The test suite with all the 128 individual test cases covering this codebase is written with 2176 lines of code. In the codebase there are 1076 individual statements of which 1048 are covered in the unit tests, bringing the code coverage to 97%. A statement in a code is any operation that the stack executes, for example, a variable assignment, an if-else-clause, a function call, etc.

## Typing Enforcement

Another vector to regression testing is typing enforcement. While unit tests ensure that functions work as intended, typing enforcement ensures, that the variables that are passed between the functions retain their correct forms. This aspect is not inherently covered by python's dynamic typing, but this was achieved by adding type hints with the *typing*<sup>17</sup> package, and type validation with the *mypy*<sup>23</sup> package.

## Style Enforcement

*There are as many programming styles as there are programmers programming programs.*

While unit tests and style enforcement may fail due to syntactic reasons, it's possible for horribly written code to pass testing and usage requirements. Software developers often need to agree on common conventions when they are working together on projects, and the Python Software Foundation has also devised such guidelines in their Python Enhancement Proposal (PEP) article 8.<sup>24</sup> Using the rules of PEP-8 developers can impose strict code quality requirements to ensure that code remains consistent and readable in the future. This style enforcement for the IA was conducted using the *black*<sup>25</sup> package, which also provides automatic formatting as means of improving developer productivity.

## 5.3 End to End Testing

End to end (E2E) testing is a framework of testing the whole pipeline of functionalities from start to end. It often involves laying out a plan for each step to be tested and verified for functioning correctly.<sup>26</sup>

For the IA, E2E testing was conducted manually, with a plan devised for automated testing. Manual testing is simple, but it only allows to test the successful operation and some error cases. To be able to test all error cases, mock services will need to be built of each subsystem in network.<sup>Fig.8</sup> Automated

E2E testing is a laborious task and is thus excluded from the topic of this thesis and will be developed at a later date as deemed necessary.

## 5.4 Acceptance Testing

Acceptance testing is a set of final testing to verify that software meets a set of essential criteria before being passed on to production. This stage is conducted as black box testing, where only the outcome is under scrutiny.<sup>27</sup> Acceptance testing for the IA was conducted during meetings with the Customer, with having representatives of each system present to note down flaws, while the Customer's clerks attempted to execute various operations in the different systems.

Table 3. Acceptance testing meetings, findings, and fixes.

<i>Meeting number</i>	<i>Findings</i>	<i>Fixes</i>
1	<ul style="list-style-type: none"> <li>Some DDA documents were stored under a wrong classifier</li> </ul>	<ul style="list-style-type: none"> <li>Edit configuration values in IA</li> </ul>
2	<ul style="list-style-type: none"> <li>Bug in DDA that prevents document metadata from being formed</li> <li>Typo in IA caused wrong resource translations to be transferred to CRMS</li> </ul>	<ul style="list-style-type: none"> <li>Bug fixed in new version release</li> <li>Fixed typo</li> <li>Edit configuration values in IA</li> </ul>
3	<ul style="list-style-type: none"> <li>Bug in REMS that caused forms to disappear when copying a previous application and adding a new resource to application</li> </ul>	<ul style="list-style-type: none"> <li>Bug fixed in new version release</li> </ul>
4	<ul style="list-style-type: none"> <li>Error in CRMS when linking resources to applications</li> </ul>	<ul style="list-style-type: none"> <li>Human error, resources were set as deactivated</li> </ul>
5	<ul style="list-style-type: none"> <li>Bug in DDA causes decision documents to not be findable</li> </ul>	<ul style="list-style-type: none"> <li>No solution to bug</li> </ul>

6

- Bug in DDA causes decision documents to not be findable
- Modify the way IA queries decision documents from DDA to be able to find faulty decision documents (temporary fix)

A total of six acceptance testing sessions were held, before all issues were taken care of. In total there were 4 bugs found across the systems and a couple of minor configuration errors. The bugs in IA were misclassifying documents for DDA and a typo in reading resource translations from REMS to be pushed to CRMS. As configuration errors, the fixes were issued swiftly. REMS had a share of one bug where an application would lose an e-form when the application was copied to a new application and the resources were changed. A fix was issued for this on code level and a new version was released. CRMS was the best performing system of the bunch, as no flaws were found from it, only human errors that were corrected with process guidelines. DDA had the majority of issues from the four systems. The first issue was related to document metadata not being formed correctly based on underlying rulesets. This issue was fixed with a new version release. The second issue of decision documents not being findable was caused by an unknown conflict in the database, and its cause is still unknown to this date. The issue hasn't been fixed, but a workaround was devised to it from the IA's side, by querying decision documents in a different manner. This fix was deemed to be a temporary solution by the DDA's personnel while they investigate the issue.



## 6 Conclusion

Digitalisation is quickly transforming the service industry landscape by being able to provide services more rapidly to a wider audience around the clock. The most notable advantage digital media provides is the service availability, as most things that previously required queueing at an office during daytime, can now be done in the middle of the night, leisurely from one's home. The second advantage comes from the newfound accessibility from the nature of digital media. Paper forms and human clerks at a reception can only serve a very narrow clientele: those who can read, and who speak the same language as the clerks. With digitalisation, services can be used by virtually anyone using the right accessibility tools, be it translation, text-to-speech, or any other form of aid.

The goal of this thesis was to create a systems integration application, that not only connects independent systems together, but also automates work tasks for the clerks. The application was developed to be highly customisable and free of regular maintenance. These aspects were achieved using elaborate configuration schemes and stateless architecture. The finished application was positively received by the Customer, with the following quote received from the project manager after one of the acceptance testing sessions:

*"I have been in the IT industry for 25 years, and this is the best performing systems integration that I have witnessed to date."*

Overall, I am very happy with the results of the project and the integration application that I created. It was my first outside-company project that I largely designed, developed, and maintained communications with the end customer by myself. My only grievance comes from the choice of the programming language, where I would have rather used Go over Python.

The IA has now finished acceptance testing and has been deployed to production. The expected application volumes linger somewhere around 500 per year. The next steps for the application are to prepare for new features to be added and to contemplate on the creation of automated E2E testing scenarios.

## References



- 1 Digitalisation [Internet]. Ministry of Finance; [cited 22 Jan 2022] Available from: <https://vm.fi/en/digitalisation>
- 2 Regulation 30.4.2010/310 [Internet]. Helsinki: Republic of Finland; 30 Apr 2010 [cited 1 Jan 2022]. 2§ Agencies, departments, companies and other institutions of the Ministry of Education and Culture. Available from: <https://finlex.fi/fi/laki/ajantasa/2010/20100310#P2>
- 3 The board's proposal to parliament for a law on a limited liability company called CSC – IT Center for Science Ltd [Internet]. Helsinki: Ministry of Education and Culture; 19 Oct 2018 [cited 1 Jan 2022]. Available from: [https://api.hankeikkuna.fi/asiakirjat/d9ce7f26-6255-41e4-ba52-3dce39f6d90f/08e96958-f5b5-4020-8600-07087d2daa5c/KIRJE\\_20181113110425.pdf](https://api.hankeikkuna.fi/asiakirjat/d9ce7f26-6255-41e4-ba52-3dce39f6d90f/08e96958-f5b5-4020-8600-07087d2daa5c/KIRJE_20181113110425.pdf)
- 4 Law 1397/2016 on public procurement and concessions [Internet]. Helsinki: Republic of Finland; 29 Dec 2016 [cited 1 Jan 2022]. 17§ Exclusive service contracts. Available from: <https://www.finlex.fi/fi/laki/alkup/2016/20161397#Pidm45237816340688>
- 5 Law 306/2019 Digital Services Act [Internet]. Helsinki: Republic of Finland; 15 Mar 2019 [cited 4 Jan 2022]. Available from: <https://www.finlex.fi/fi/laki/alkup/2019/20190306>
- 6 Resource Entitlement Management System [Internet]. Espoo: CSC – IT Center for Science Ltd.; 2017 [cited 4 Jan 2022]. Available from: <https://github.com/CSCfi/remS>
- 7 What to know when planning system integration [Internet]. Cleo; [cited 23 Jan 2022]. Available from: <https://www.cleo.com/blog/system-integration>
- 8 Hou, Z.X. Guest editorial: Special Issue on Network Integration Technology [Internet]. Telecommunication Systems; New York. May 2013 [cited 23 Jan 2022];53(1):1-2. Available from: <http://dx.doi.org/10.1007/s11235-013-9669-2>
- 9 Agile 101 [Internet]. Agile Alliance; [cited 22 Jan 2022]. Available from: <https://www.agilealliance.org/agile101/>
- 10 Spackman, D. Speaker, M. Enterprise Integration Solutions. Redmond, Washington, USA: Microsoft Press; 2005. 368 p.
- 11 Lutz, M. Programming Python. 3<sup>rd</sup> edition. Sebastopol, California, USA: O'Reilly; 2009. 400 p.
- 12 General Python FAQ [Internet]. Python Software Foundation; [cited 18 Mar 2022] Available from: <https://docs.python.org/3/faq/general.html>

- 13 Chisnall, D. The Go Programming Language Phrasebook. Ann Arbor, Michigan, USA: Addison-Wesley; 2012. 264 p.
- 14 The Go Programming Language Specification [Internet]. Google; [cited 18 Mar 2022] Available from: <https://go.dev/ref/spec>
- 15 Top Computer Languages [Internet]. Statistics Times; [cited 18 Mar 2022] Available from: <https://statisticstimes.com/tech/top-computer-languages.php>
- 16 Asyncio – Asynchronous I/O [Internet]. Python Software Foundation; [cited 18 Mar 2022] Available from: <https://docs.python.org/3/library/asyncio.html>
- 17 Typing – Support for type hints [Internet]. Python Software Foundation; [cited 18 Mar 2022] Available from: <https://docs.python.org/3/library/typing.html>
- 18 Stateful vs. Stateless Architecture: Why Stateless Won [Internet]. Dwyer, G., Virtasant [cited 19 Mar 2022] Available from: <https://www.virtasant.com/blog/stateful-vs-stateless-architecture-why-stateless-won>
- 19 P. K. Singh, M. Kumari. Containers in OpenStack. Packt Publishing; 2017. 176 p.
- 20 M. C. Feathers. Working Effectively with Legacy Code. 1<sup>st</sup> edition. Pearson; 2004. 464 p.
- 21 REMS Copy – A helper tool for copying items from one REMS instance to another. CSC – IT Center for Science Ltd. [cited 26 Mar 2022] Available from: <https://github.com/CSCfi/rems-copy>
- 22 What Is Regression Testing? Definition, Tools, Method, And Example [Internet]. Software Testing Help. [cited 26 Mar 2022] Available from: <https://www.softwaretestinghelp.com/regression-testing-tools-and-methods>
- 23 Mypy [Internet]. The Mypy Project; [cited 26 Mar 2022]. Available from: <http://mypy-lang.org/>
- 24 PEP 8 – Style Guide for Python Code [Internet]. Python Software Foundation; [cited 26 Mar 2022]. Available from: <https://peps.python.org/pep-0008/>
- 25 Black [Internet]. Python Software Foundation [cited 26 Mar 2022]. Available from: <https://github.com/psf/black>
- 26 End To End Testing: A Detailed Guide [Internet]. S. Bose. BrowserStack; 2021 [cited 26 Mar 2022]. Available from: <https://www.browserstack.com/guide/end-to-end-testing>

- 27 What Is Acceptance Testing (A Complete Guide) [Internet]. Software Testing Help; [cited 26 Mar 2022]. Available from: <https://www.softwaretestinghelp.com/what-is-acceptance-testing/>

## Appendices

### Appendix 1, Screenshot of REMS demo environment

Catalogue Applications Administration About EN FI SV  Owner  Sign out

---

# REMS

## Application 2022/14

Application: Success

Fill out and submit the application. Several Applicants can be invited as members of the application, and each applicant has to accept the terms of use to gain access.

**State**

➤ Apply
Approval
Approved

Application

Title


State

Latest activity

**Events**

2022-02-05 15:04 **Owner created application 2022/14.**

**Actions**

Save as draft
Send application
Delete draft...
  
Copy as a new application
 PDF

**Applicants**

**Applicant**

Name

[Show more](#)

Invite member...

**Resources**

Auto-approve workflow – [More info](#)

Change resources...

**Terms of use**

Each member is required to accept the terms of use to access the resources.

[CC Attribution 4.0](#)

[General Terms of Use](#)

Accept the terms of use

**Example form with all field types**

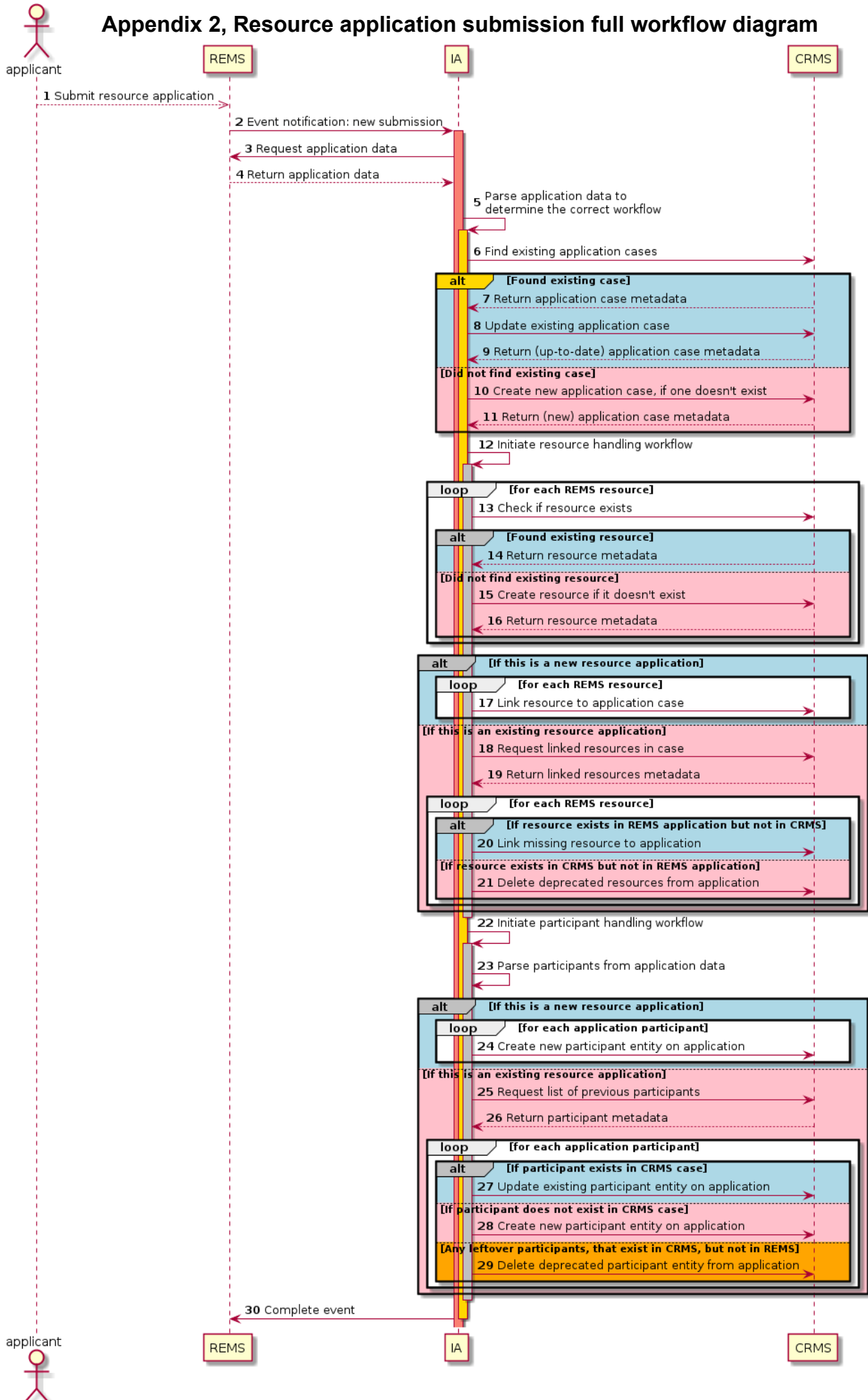
This form demonstrates all possible field types. (This text itself is a label field.)

**Application title field**

**Text field**

**Text area**

## Appendix 2, Resource application submission full workflow diagram



### Appendix 3, Resource application archival process full workflow diagram

