



^b
**UNIVERSITÄT
BERN**

Archiving in Small and Medium Enterprises

Bachelor Thesis

Marcel I. Chavez Panduro

from

Wattenwil BE, Switzerland

Philosophisch-naturwissenschaftliche Fakultät
der Universität Bern

8. Januar 2019

Prof. Dr. Oscar Nierstrasz

Software Composition Group

Institut für Informatik

University of Bern, Switzerland

Abstract

Nowadays every company has to store almost every sheet of paper that is produced during its day-to-day business. In almost every process one or multiple documents have to be printed out to ensure correct book keeping. All these various papers have to be stored for over ten years.

The amount of paper constantly grows over each working day and after a short period of time, this mass of files takes up a lot of space already and most of the companies need an extra storageroom for these files. This kind of archiving could be avoided with the current technical possibilities.

In this bachelor thesis I evaluate a valid approach and try to implement a simple digital archive. A lot of processes could be optimized especially when the company is already using an enterprise resource planning system. During the process of planning and developing I encountered multiple challenges and tried to describe different approaches for a solution. The result of this bachelor thesis was a working solution, however with some drawbacks and unfinished requirements.

Contents

1	Introduction	1
1.1	Archive Requirements	3
2	Concept	4
2.1	WebArchive	4
2.1.1	OAIS Model	4
2.1.1.1	Ingest	7
2.1.1.2	Data Management	8
2.1.1.3	Archival Storage	8
2.1.1.4	Access	8
2.1.1.5	Administration	9
2.1.1.6	Preservation Planning	9
2.1.2	Web Access and Definition	9
2.2	Database	11
2.2.1	Database CRUD	13
3	Implementation	15
3.1	MVC in the Context of the WebArchive	15
3.2	Authentication and Authorization	16
3.3	Security	17
3.3.1	Access Points	18
3.3.2	WORM	19
3.3.3	File Encryption and Certification	19
3.4	File Validation	20
3.5	Storing Structure	22
3.6	User Interface	22
3.6.1	Search	22
3.6.2	Authorization	23
3.6.3	Administration	24
3.7	Summary	25

4	Conclusion and future work	27
4.1	Lessons Learned	27
4.2	In Process	28
4.3	Enhancements	28
5	Anleitung zum wissenschaftlichen Arbeiten	29
5.1	Architectures	30
5.1.1	Monolithic Architecture	30
5.1.2	Self-Contained System Architecture	30
5.1.3	Microservice Architecture	30
5.2	Targeted Architecture	31
5.3	Consequences for the WebArchive	32
5.3.1	User Interface	32
5.3.2	Data Set	33
5.3.3	Business Logic	33
5.3.4	Technology	33

1

Introduction

Every company listed in the commercial register is obliged to manage an archive. The guidelines and requirements for an archive are regulated in different legal texts. In Switzerland these laws are written down in the following books: Obligationenrecht (OR), Verordnung über die Führung und Aufbewahrung der Geschäftsbücher (GeBüV), Datenschutz (DSG), Mehrwertsteuer Gesetz (MWStG), Strafgesetzbuch (StGB), to mention some of the most important ones. These legal texts target the “classical” archiving (as in the storage of paper; physical storage) first and foremost and not explicitly digital archiving. One thing that we can take away from these legal texts is the definition about what should be stored in an archive: All books that are necessary to prove the value and represent the debt and accounts receivable of the company have to be stored for about ten years (for more detailed information about what the company is obliged to store please consult the mentioned legal texts). Not every document in a company is valued the same. Some files are more important and still have to be stored in paperform despite having a working digital archive. In this bachelor thesis I will concentrate on the day-to-day business. This includes bills, delivery orders, offers and other files which are produced daily.

Depending on the size and business sector of the company the daily mass of produced paper is extremely high. The company often has to rent an additional room to store this data. Considering the state of technology this is a cumbersome process and sometimes unnecessary. Nowadays companies are already working with an enterprise resource planning (ERP) system. That would mean that almost all documents are created and stored in digital form in the environment of the ERP system. Despite the digital availability the companies are obliged to print out all of the documents, transport them

to the archive storage room and archive them correctly. The reason for this process is that most of these companies don't have a digital archive that is legally accepted. The company I'm currently working for is aware of this problem. A simple solution for an archive system already exists but it just works as a file storage and would not withstand an assessment of the government.

So it is not surprising that the idea for the subject of this bachelor thesis comes from the company where I work. I have the possibility to develop a prototype of the digital archive in cooperation with my company. This is a first step to offer our customers a valid alternative to the "classical" archive. The company where I work already has a lot of running systems for their different customers and this project should be embedded in the current environment. The centerpiece of the company is an ERP system with multiple modules for finances, addresses, materials, material planning and many more. The wider usage of internet applications drove the company to implement parts of the ERP in web applications. In the near future customers will be able to use some of the ERP modules online. In order to achieve this, the applications must have direct access to the data of the ERP. This implies that the customer already uses the ERP system and has the necessary licenses. In future it should be possible to deploy one or multiple web applications independently from the central data management of the ERP system. The goal is to present a flexible solution on different platforms. The intended result of this bachelor thesis is to develop a functioning digital archive system, known as WebArchive, that will be deployed as one of many web applications and cooperate with the existing ERP system.

This bachelor thesis is a prototype for a digital archive in the environment of an ERP system and allowed me to conduct some research in the subject of archiving. I will present a possible approach to develop a valid and useful digital archive. It will address the arising challenges and suggest possible technologies or solutions to overcome these problems. In the scope of this bachelor thesis I will always refer to the implementation of the digital archive as WebArchive.

In the course of this bachelor thesis I will often refer to an ERP system by which I mean the ERP system that is developed by the company I'm working at. Otherwise I will clearly indicate when I'm referring to a different ERP system or to the term ERP itself.

Because the Bachelor thesis is written in cooperation with the company where I work, I'm not able to present the code or snippets of code in this documentation.

1.1 Archive Requirements

The digital archive would bring a lot of value to companies but it also brings a lot of challenges. It requires a complex system that ensures the integrity of the files during all stages of archiving (upload, download and storage). The system should also be able to process various user requests. All automated functionalities and each user access should be supervised to ensure a correct procedure. The government itself has some requirements about how to implement an archive system. Unfortunately these requirements are specified in a completely theoretical manner. This leads to a lot of interpretation for the developer in the actual implementation of a digital archive system. Following this I will present the specifications of the government of Switzerland¹. I considered which part of the system is targeted with each requirement and have listed them below.

Integrity The archive system has to ensure the data integrity over the whole duration of storage. It should not be possible to alter or delete files or part of the data without the application or the administrator knowing. This targets the security aspects of the system and also the process of uploading, storing and downloading of the data.

Understandability The document should be understandable for a long period of time. This requires from the digital archive that additional metadata for the purpose of understandability has to be extracted. The most basic information would be to extract the format of the file. This way the user receives a document with the format ending and can therefore open the document with a suitable program (e.g. PDF can be opened with any PDF reader).

Originality The structure and the overall appearance of a file does not change during the time of storage. The data that has been uploaded looks the same after a later download regardless of the time passed and amount of accessing.

Authenticity The author and the origin of the document is available and coupled with the data. Clean authentication processes are necessary and for traceability, each request has to be registered and stored. Each interaction with the application should be logged.

Accessibility The stored documents have to be accessible all the time or at least at short notice. The system or administrator should be able to provide the necessary data (single access, packet of multiple files or the entirety of files). Even in the instance that a file is damaged it should be possible to reproduce the file (e.g. from a backup).

¹www.egovernment.ch/de/dokumentation/rechtliche-fragen/elektronische-archivierung accessed 2018-12-14

2

Concept

In the first stage of the bachelor thesis the goal was to create a concept. In a theoretical manner I wanted to plan and outline the architecture of the WebArchive. This required a lot of reading and gathering of information. From the beginning I decided to split the concept into two different parts. In section 2.1 the general structure and organisation of the web application WebArchive is discussed. Section 2.2 is about the setup of the database.

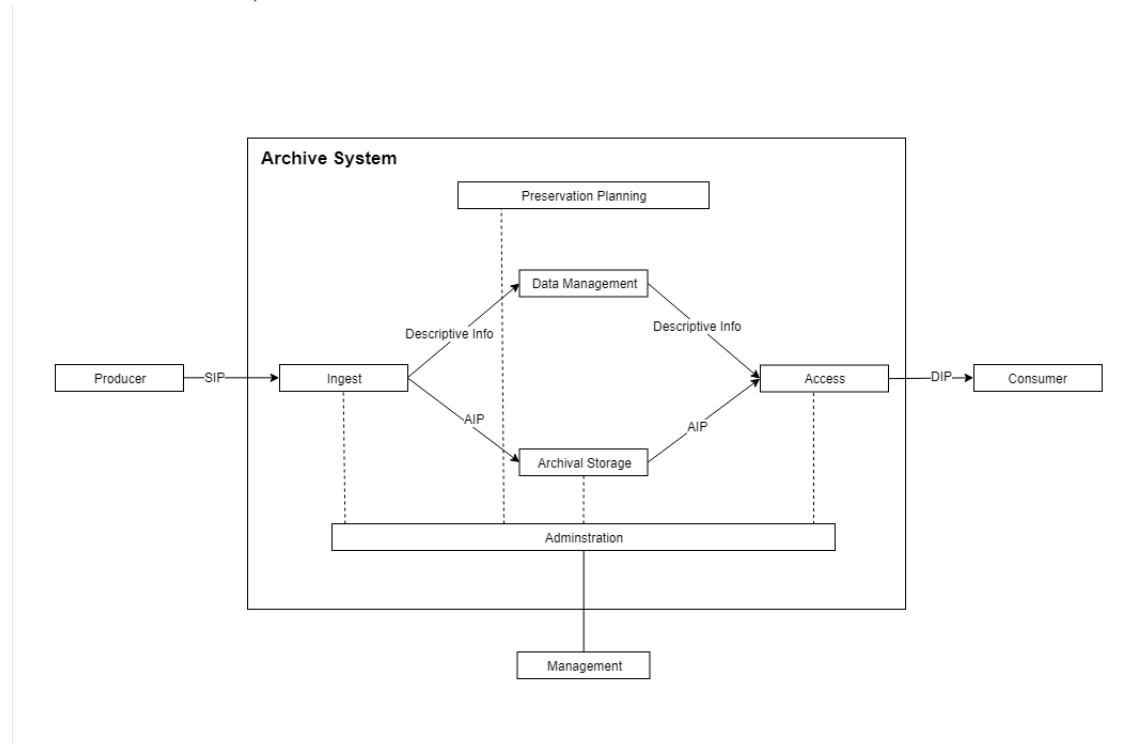
2.1 WebArchive

During my research, I encountered the OAIS (**O**pen **A**rchival **I**nformation **S**ystem). The OAIS is a defined ISO-Standard 14721 since 2003 and has been extended to version 2 in 2012. The concept and architecture used in this bachelor thesis is largely derived from the Magenta Book from June 2012 [1]. The OAIS is a reference model for digital archiving and especially targets long-term archiving. It offers no concrete implementation or technology for developers but provides concepts and approaches for the architecture of an archive system. For a more detailed knowledge and understanding I refer to the actual book (see bibliography [1]).

2.1.1 OAIS Model

The OAIS model provides a framework for understanding, terminology and concept of a long-term archive system. It offers an abstract view on how to organize and theoretically

Figure 2.1: OAIS model (based on the Magenta Book [1] p. 4-1, figure 4-1: OAIS Functional Entities)



build / implement an archive system. It especially targets archiving for the long term. This bachelor thesis focuses more on short-term archiving and therefore some parts of the OAIS model have been disregarded because they would have been too time consuming and wouldn't bring immediate value. That doesn't mean that these subjects will be forgotten and never appear in a final version. I have to point out that in this bachelor thesis I built version 1.0 with only the most important functionalities. Further development will be needed to improve the functionality of the WebArchive.

First the Magenta Book [1] defines all used terminologies for understanding. Some of these terms will be used in this document. The most important ones are listed and shortly explained afterwards in own words based on the definitions from the Magenta Book (p. 1-8, section 1.7.2 [1]). Other less frequently used terms will be explained as needed.

Submission Information Package (SIP) Set of data that is submitted by a Producer to construct or update one or several AIPs.

Archival Information Package (AIP) The AIP consists of the information that is the

target of archiving and the Preservation Description Information (this will be elaborated in more detail in section 2.2)

Dissemination Information Package (DIP) The DIP is the package that is returned upon a successful consumer request. It consists of the information of one or more AIPs.

Descriptive Information Information that is provided mainly from the producer but also contains information from the file itself. This set of data helps to find, order and / or retrieve data. Generally speaking this is the data that is finally stored to the database.

Content Information Part of any Information Package (AIP, SIP, DIP, Descriptive Information) that contains the target of preservation and the Representation Information.

Representation Information Part of the Content Information that helps to make the target of preservation understandable to the Consumer.

Figure 2.1 represents the abstract OAIS model of an archive system. It especially visualizes the interface and the environment of an archive system. Each path of communication is characterized by a defined information package (SIP, AIP, DIP, Descriptive Information). The dotted lines indicate an implicit data flow from the Administration to distribute information. Only Ingest, Access and Administration offer an interface to external actors. All other communication should be internal. The OAIS model splits the external sources which interact with the archive system in three groups:

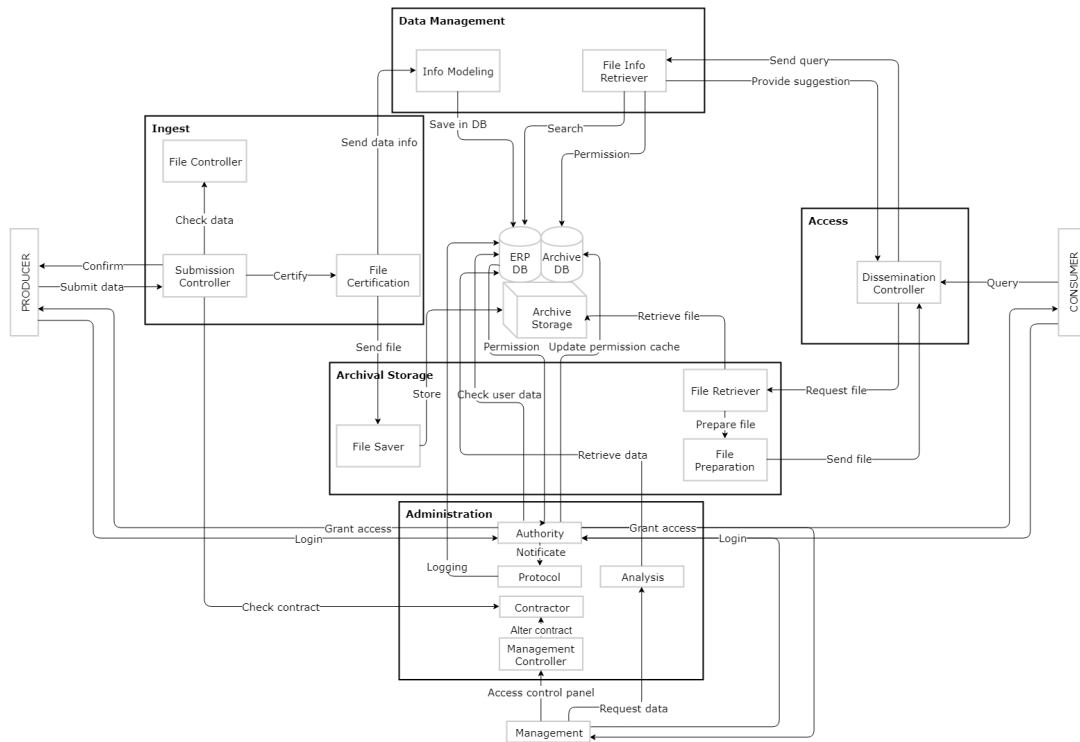
Producer This entity is an external participant who is providing data to the archive. This can be an user, another archive or information system.

Consumer A user (or entity) with the goal to retrieve data from the archive system. OAIS defines a specific class of consumer: “A special class of Consumers is the Designated Community. The Designated Community is the set of Consumer who should be able to understand the preserved information.”(The Magenta Book p. 2-2 [1])

Management Sets the policy for the archive system. For example access rights, user management or data structure of the AIP and DIP.

We now know all the stakeholders that are interacting with the archive system and have some definitions for the most frequently used terms in the OAIS model. All these interactions and the communication happens through interfaces. The OAIS model provides a detailed explanation for each access and communication channel. Furthermore the internal interfaces and information flows are described. The process and the information distribution is split into six functional entites. In figure 2.2 we see a possible and concrete

Figure 2.2: Class structure according to OAIS



implementation for an archive system accordingly to the OAIS model. This figure is the blueprint of my implementation for the WebArchive. The blueprint from figure 2.2 is a more detailed view for the OAIS model in figure 2.1. The terms used in figure 2.1 and during this bachelor thesis are according to the OAIS model from the Magenta Book ([1] p. 4-1,4-2,4-3 section 4.1: Functional Model). Each of the six parts is reference as a “Functional Entity” (Ingest Functional Entity, Data Management Functional Entity, Archival Storage Functional Entity, Access Functional Entity and Administration Functional Entity). For convenience the term “Functional Entity” is omitted.

2.1.1.1 Ingest

The entity Ingest provides the entry point for every SIP and handles the communication with the Producer. Every SIP will first be submitted to the “Submission Controller” who handles the communication with the Producer (request and response). The “Submission Controller” extracts from the SIP (and the additional header data) the needed information and transforms it into an AIP. The AIP will be forwarded to the “File Controller” and

will be validated according to the specifications given by the Administration (more on the subject file validation see section 3.4). After the validation process is successfully terminated the “Submission Controller” immediately calls the “File Certification” along with the validated AIP. The “File Certification” extracts the Content Information and creates the certificated file (for some thoughts on certification and encryption see section 3.3.3). As soon as this process is terminated the Descriptive Information is separated from the AIP and sent to the Data Management.

2.1.1.2 Data Management

The Data Management works under the assumption that the Descriptive Information is complete and formatted correctly. All database transactions concerning the Descriptive Information are handled entirely in this entity. The Ingest entity sends the Descriptive Information of the currently processed AIP to the Data Management where the statements for the database are prepared and executed. The part “File Modeling” is there to correctly fill in the needed models and database fields. The Data Management is not only responsible for storing the data but also has to prepare queries from the Consumer in the “File Info Retriever”. The search statements are provided by the search engine and sent to the “File Info Retriever”. As soon as a result for the search is available the “File Info Retriever” returns a proper formatted response.

2.1.1.3 Archival Storage

Archival storage does the actual storing and retrieving of the AIP packages. It’s also responsible for disaster recovery and surveillance of the storage environment. From the Ingest entity the Archival Storage receives a valid and prepared AIP. The “FileSaver” only searches the right path based on the AIP and stores the Content Information. On the other side the Archival Storage has to retrieve the Content Information for any Consumer requests. The Content Information has to be checked and validated before returning to the Access entity. These steps are handled by the “File Retriever” and “File Preparation”.

2.1.1.4 Access

The communication with the Consumer is carried out by the Access entity. All incoming requests from the Consumer are sent to the “Dissemination Controller”. Search requests are forwarded to the Data Management and if the Consumer eventually decides to download a file, then the request is handled by the Archival Storage. The response from the Archival Storage will be wrapped up in a DIP and sent to the Consumer.

2.1.1.5 Administration

Only the Management has access to the Administration and is therefore responsible for the administration of the archive system. This includes user management, surveillance, (if needed) migration and contracting. The Administration takes the control over the authentication and authorization in “Authority”. Furthermore all responsibilities of creating and surveilling the protocol is handled by the Administration. The protocol is created when Ingest or Access make an authentication request and also the actions of the Producer or Consumer will be logged as soon as the process is terminated. Additionally the Management is able to manage the contract between Producer / Consumer in the entity Administration. This contract describes the process of uploading / downloading a file and defines the data set that has to be provided by the Producer.

2.1.1.6 Preservation Planning

This part of the OAIS model targets the long-term archiving. The Preservation Planning ensures the accessibility and understandability of all the files stored in the archive over the long term, even if technologies to represent stored data are deprecated or no longer supported. Periodic analysis reports, constant surveillance and recommended updates are part of the Preservation Planning entity. A simple way of implementation is adding a documentation to the archive system with guidelines and rules. This requires manual reviews by the developer to update the documentation.

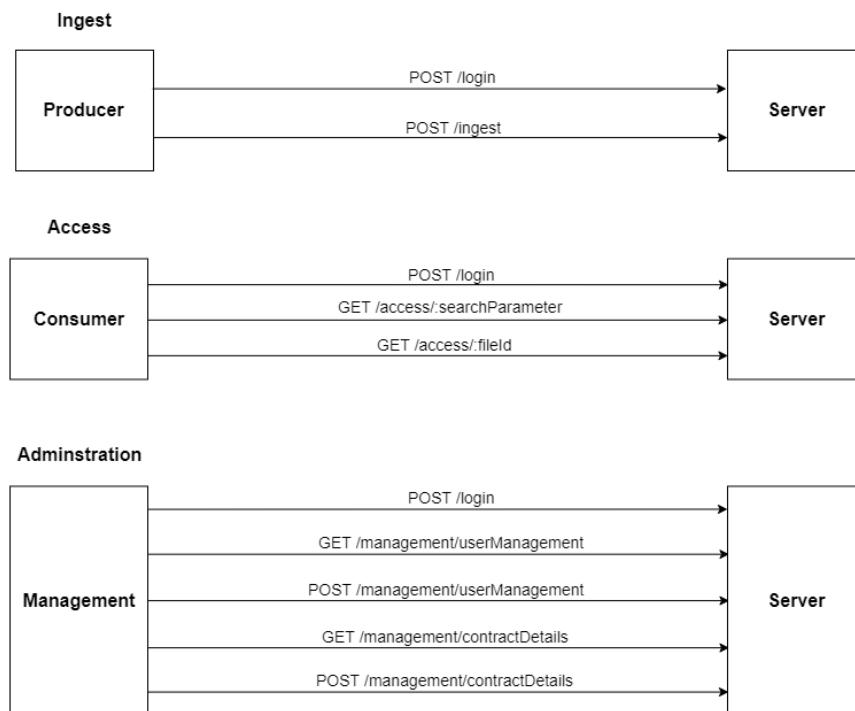
2.1.2 Web Access and Definition

Producer, Consumer and Administration need a reliable way to communicate with the WebArchive. It should enable the users to do their job and at the same time restrict them from malicious access. I had to define a standard interface and routes where the Producer, Consumer and Administration can reach the system. To offer a simple and intuitive solution, the WebArchive makes use of the RESTful API architecture. The communication with the environment of the WebArchive happens with the HTTP protocol (or rather the HTTPS protocol to ensure additional security measures). A RESTful API has to meet six constraints which are first asserted by R. T. Fielding¹. The following criteria are based on the dissertation “Architectural Styles and the Design of Network-based Software Architectures” of R. T. Fielding (p. 76-86, chapter 5 [4]) and written in own words:

Client-Server Clear separation between client and server. This enables both parts to evolve independently.

¹www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm accessed 2018-01-17

Figure 2.3: Webaccess



Stateless Each request from the client is clear and does not need additional status information stored on the server. Each request with the same data should result in the same response.

Cache It should be possible to mark data as cachable (explicit or implicit) to provide more efficient access to already retrieved data.

Uniform Interface The server provides a uniform interface to which the client is able to “talk”. This requires that the server implement generalizations of communication like making use of the HTTP actions GET, POST, PUT and DELETE.

Layered System The system is a multi layered system where on each layer different representations of data can be obtained. This is mostly useful for legacy services.

Code On Demand This allows the client to download and execute code in form of applets and scripts. This requirement is optional and doesn’t have to be implemented.

In figure 2.3 a simple representation of the available interface is visible. For each user group a set of routes are available and only the authentication (POST /login) is shared

between users. Naturally, it is possible for a user to be a Producer and Consumer. This allows one user to access two sets of access routes with proper authentication. All paths in the section Administration require additional user authorization and are only accessible to the Management.

Ingest takes all archiving requests. To make use of this route, the user has to be properly authenticated and authorized. As soon as the user is logged in, he is able to upload a file which he wants to store on the WebArchive. Along with the file, additional header data has to be submitted. In several processes the posted data is validated. If inconsistencies happen in this process, all data is dropped and the user will be notified with a “BadRequest” response.

Like with Ingest, the user has to be authenticated and authorized to use the platform from the Access. On this platform, the user is allowed to search and download data from the archive. Similar to the Ingest, invalid queries end in a “BadRequest” response. If an error occurs in the WebArchive an “Internal Server Error” is returned.

Management needs more connections because the administrator has to be able to survey the allowed users and the corresponding contract between client and server (see section 2.1.1.5). This set of access routes is a first attempt to give the Management some rights to manage a digital archive. The access routes for the administration can be (and eventually has to be) further extended to offer more flexibility to the management.

In this section we should also address the risk of sensitive data exposure. It is not advised to put sensitive data in an URL string, not even when we make use of the HTTPS protocol. The URL is saved in the browser history and everyone with access to this workstation could read the sensitive data. A common way of use is to add parameters to the URL for saving user requests. If the URL carries IDs from models, any user would be able to simply add some numbers to get access to other documents. Sometimes it is necessary or useful to just send the ID with the URL to avoid saving the state of the user on the server side. In this case it would be advised to use UUID instead of normal ID. The users would still see the UUID of the model but they can't exploit it.

2.2 Database

The OAIS model always uses Information Packages (AIP, DIP, SIP, Descriptive Information) for communication. The Information Package is split into two parts: Content Information and Preservation Description Information (PDI). I already explained the term Content Information briefly. For the database, the PDI is more important and is absolutely necessary to store. Although it could be possible to store the complete Content Information in the Database and for long-term archive it becomes necessary eventually. Representation Information can be an especially important part of the Information Package. If this part is stored in the database, it would be possible to open files even when the format is out-of-date. In the WebArchive I assume that the formats I allow to be stored

are existing at the time of storage and can be opened by the Consumer without further information. In this section we take a closer look at the PDI. The OAIS model gives a good overview of the values that should be gathered and stored in the database. PDI can be categorized in five parts. Following, these parts are shortly explained.

Provenance relates to the origin of the Content Information and the processing history.

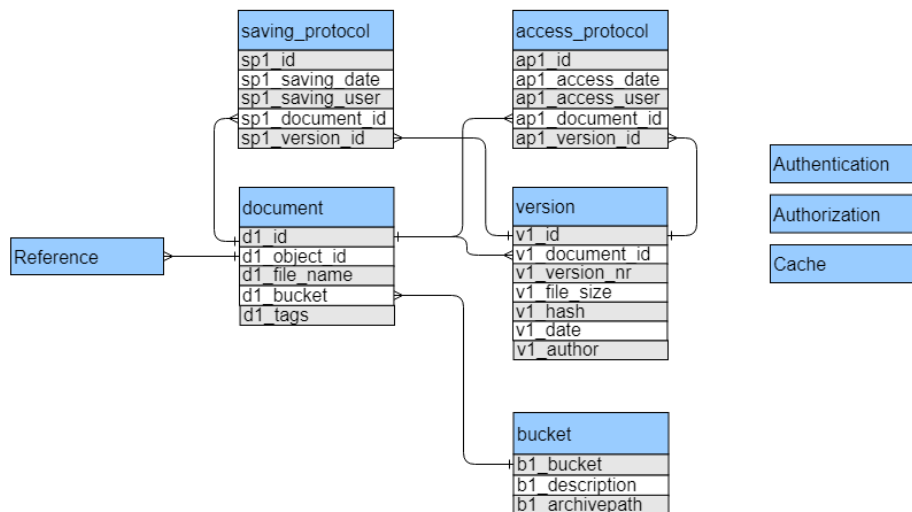
Context describes the relation of the document to the outside.

Reference is a set of identifiers with which the information content is uniquely identified.

Fixity protects the document from malicious alteration (hash value, certification, encryption)

Access rights define the distribution and usage of the preservation data.

Figure 2.4: Database Model



Based on the provided structure of the OAIS model, I created the database model in figure 2.4. This is the simplest way to implement the database model and can surely be extended with additional values and tables. The focus lies on the tables that are essential for the archive system and how they are associated with the OAIS model. Naturally, each system works in a different way and therefore more tables are necessary. Additional information concerning authentication, authorization or cache has to be stored in separate tables.

Saving / Access Protocol For each login and the following action (upload or download) a corresponding entry will be made. This is a requirement from the government to offer traceability and indicates the originality of the document. It also relates to the Provenance of the OAIS model. Each protocol entry is related to a version and document that was uploaded or downloaded.

Document The table Document holds data that is unchanged for every version of a file that is uploaded. As soon as a file is uploaded, a new entry will be made with a unique ID. The ID for the Document table is an UUID because I want to use this ID as a parameter in the URL to retrieve corresponding versions. The Document holds a reference ID called `d1_object_id`. This is the ID used in the ERP system. Furthermore the Document stores the filename, bucket and tags.

Version To every Document entry a new version is created. For every change to a document that should be archived, a new version is created. Versions will never be replaced or altered. The database entry Version stores the file-specific data: version number, file size, hash value, date and author. This data set targets the requirement Fixity and fulfills a part of Provenance by storing the author.

Bucket This is a helper table. The purpose of this table is to store the absolute path to folders where the files are being stored. The concept of the bucket will be explained later in section 3.5.

Reference, Authentication, Authorization, Cache These tables are suggestions about what a system also has to include for proper functionality. However, the Authentication and Authorization have to be implemented in some way to guarantee the point Access Rights from the OAIS model.

2.2.1 Database CRUD

CRUD stands for **C**reate, **R**ead, **U**ppdate and **D**eleate. These are the four basic functions of persistent storage². In a relational database, each function can be assigned to an operation (create - INSERT, read - SELECT, update - UPDATE, delete - DELETE). In each database the root database user is able to create new users with different rights. With this simple functionality (that is provided by the database) I am able to ensure some protection for the data. In an archive system I can limit and control the work of an application by assigning a user with specific rights. In reference to the figure 2.4 shown above, the application should not be allowed all rights to ensure that important data is not altered or deleted by accident or through malicious attacks. My first approach suggests to create two different database users and grant them different access rights. For one, the

²https://en.wikipedia.org/wiki/Create,_read,_update_and_delete accessed 2018-12-14

web application WebArchive would receive a set of permissions (referred to as DB-user) and the administrator of the WebArchive would receive a separate set of permissions (referred to as DB-administrator).

The protocols for example should only be inserted once and the DB-user would have no authority to alter or change such a protocol. The SELECT command should be reserved for the DB-administrator only. This would prevent accidental or malicious alterations of the protocols from the daily workload of the WebArchive.

Based on the OAIS model the storing of the Descriptive Information happens after the validation of the file. The DB-user should be able to create the Document and Version entry for the corresponding AIP. Afterwards, the DB-user has no reason to alter or delete these values. However, the user must be able to search for specific names or even tags that are stored in the database. The DB-user has therefore the right to INSERT and SELECT all values on the tables Document and Version.

Only the DB-administrator should be permitted to alter the content of the Bucket table. Accidental deletion or changing through the DB-user would not erase all stored files but the files would be lost on the storage unit. The administrator would have to manually search the bucket and restore the bucket entry. However the DB-user must have access to this table because the absolute path to a bucket is stored there. Therefore permission to SELECT is sufficient for the DB-user.

The approach of two different roles for database users would add an additional layer of protection to the data. Nevertheless both of these roles would be embedded in the same application. The question arises if this implementation is worth the effort. Depending on the used framework it could be a cumbersome process to add an additional user and clearly separate these two users. With the authentication and authorization of the WebArchive I already define different access rights for separate entities and therefore I don't see the advantage of two separate database users. The documentation of the WebArchive includes a set of rules about how the database should be used and which tables should be accessed by the WebArchive user and the WebArchive administrator. For further development of the WebArchive, these guidelines should ensure proper use of the data and prevent developers from misusing the database.

Eventually it could be a possibility to extract the part of Administration completely from the WebArchive and develop a separate application. In this case the permissions for the WebArchive database user could be reduced to a minimum and simultaneously it would be possible to enrich the set of permissions for the administrator. Depending on the development in the company where I work, a new concept would be required for the Administration.

3

Implementation

The next step was to implement the WebArchive according to the concept. The OAIS provides a well defined structure and organisation to an archive. The organisation of the system is visible in figure 2.2 and I tried to stick to it as closely as possible. I did that by organising the package and class structure as shown in figure 2.2.

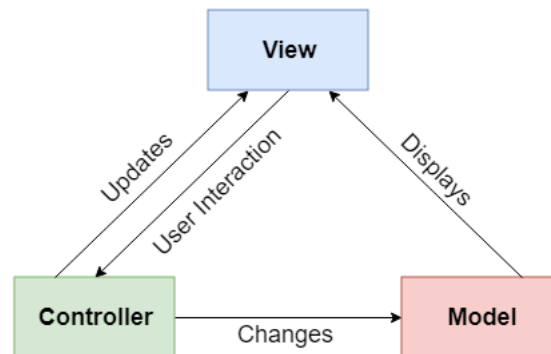
In almost every stage of development a problem surfaced or I had to decide on a specific technology. I would like to highlight these problems in this chapter and explain why I classified them as problems. I also would like to present some solutions or approaches to fix these problems. Most of these suggestions are theoretical only and have not been implemented in the actual version of WebArchive 1.0. Nonetheless I tried to take all possibilities (or at least many of them) into account and explain how they could be useful or provide additional usability.

3.1 MVC in the Context of the WebArchive

Generally, the implementation of the WebArchive follows the rules and guidelines of the MVC (**M**odel, **V**iew, **C**ontroller) pattern. The implementation of a system is split into three different parts. The **C**ontroller takes responsibility for initializing and updating the **V**iew. Based on the user interactions on the **V**iew, the **C**ontroller is notified and eventually has to change the **M**odel. Changes to the **M**odel can influence the presentation of the **V**iew, therefore the **M**odel itself notifies the **V**iew.

In figure 2.2 the class structure of the WebArchive shows three **C**ontrollers: Submission Controller, Management Controller and Dissemination Controller. With the help

Figure 3.1: MVC pattern



of various helper classes these **Controllers** have the responsibility over an entity. Each of them need a different **View** to display their content. Each of them offer one or more **Views** to allow user interaction. Based on the input from the user the Information Packages AIP, SIP, DIP or Descriptive Information are created or updated. Also additional tables from the WebArchive are representations of models. The entity Administration has therefore the possibility to update models, if needed.

3.2 Authentication and Authorization

First thing that comes to mind when we are talking about security is authentication. There are different ways to implement authentication in a web application. I will focus on three possible approaches.

Centralized Credentials The most straightforward way is to implement a login page for the user. A normal user would submit his username and his password. Based on this information the system would be able to verify the user and also assign an authorization level to the user. Such a system is implemented relatively fast and offers some security for the system. In the context of multiple web applications, that would restrict the system to a monolithic architecture (for more on this architecture see chapter 5) because all attached systems must have direct access to the credentials. If this is not possible each web application needs to implement

centralized credentials on his own. In this case the user is obliged to login multiple times although the web applications are in the same environment.

Authentication with AuthServer In this scenario we would use an additional server which would only handle the authentication request for the users. Users and external applications would be redirected to this authentication server for a proper login. With this solution you would be able to create multiple self contained systems (for more on this architecture see chapter 5) with a central authentication authority. This approach is somewhat similar to the oAuth2 but does not implement the standardized protocol.

oAuth2 [3] This is a more complex approach but offers more possibilities and ensures proper authorization. Applications can be authenticated through a registration on the resource server (server hosting the protected data). At the same time it would also be possible to allow a login mask for users. The authentication will still require a username and password. The great advantage is the authorization. A user can request access for all the applications which are hosted in the resource server and the resource owner can then decide to grant or dismiss the request. If the access is granted, the user receives an access token and with it he can access the requested application without logging in again.

As said before, these scenarios of authentication and authorization are a process of an evolving environment of different web applications. At the moment everything is managed through a single database. So the simplest way for some sort of authentication and authorization happens based on centralized credentials. This is the current implementation of the WebArchive to ensure authenticity. In the WebArchive authorization is implemented with three different authority levels. Each user has to be assigned an authority level at creation. By the time the user logs himself in, the authority level is retrieved from the database. The version of a document and each bucket also has an authority level assigned. If the authority level of the user is too low, he can't access the bucket or version of the file. This is a very simple solution which fulfils authorization on a very basic level. The goal would be to have a server that handles permissions for multiple services and web applications. To accomplish this, the company where I work is setting up an oAuth2 environment.

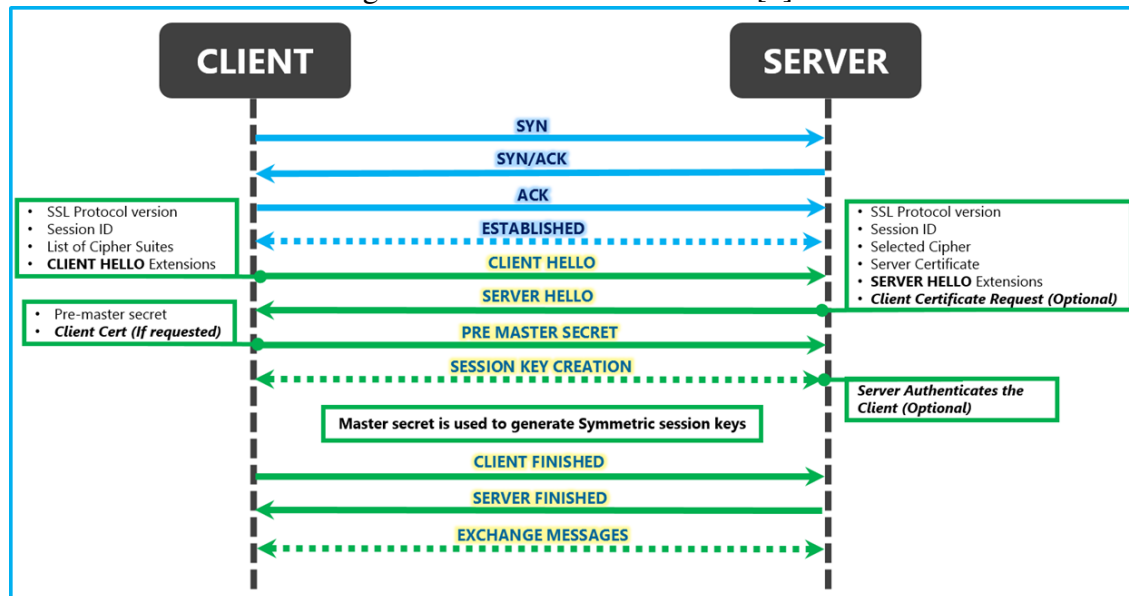
3.3 Security

In the environment of a company, the archive always contains a lot of sensitive data. To protect this data, authentication and authorization is only one part of the security. As soon as the WebArchiv knows who the user is and if he is authorized the focus changes to the target of preservation. This section addresses the security of the file and all access

points. The whole process of archiving should be secure. I have to make sure that the file is uploaded correctly, stored safely and can be downloaded unchanged.

3.3.1 Access Points

Figure 3.2: SSL/TLS Handshake[2]



To ensure that no one is able to tamper with the data on the way to and from the archive, it is necessary to implement the SSL/TLS [5] protocol. With this protocol the client and server are able to establish a secure connection.

In figure 3.2 we see an illustration of an SSL/TLS handshake. Before the essential messages are exchanged, the server and the client define the way of communication. For websites it is usually the client who initiates the contact. With his first request he asks for the wished SSL/TLS version and cipher suites (a combination of several cryptographic algorithms; one suite uses different algorithms for key exchange, authentication, bulk encryption and message authentication¹). If the server can fulfill the requirements it will answer with its certificate. To be able to do this, it is important that the server owns a certificate from a trusted Certification Authority (CA). When this is the case, the client is able to make sure that the server is a trusted communication partner by checking with the validation authority (VA) if the certificate is valid. As soon as the client has verified the certificate, a pre-master secret is created with the agreed cipher suite and encrypted

¹www.acunetix.com/blog/articles/establishing-tls-ssl-connection-part-5 accessed 2018-12-30

with the public key from the server. Now the server is able to decipher the secret with its private key and both can establish a secure connection. With this technology we can ensure a secure communication between client and server. In the WebArchive every connection has to be HTTPS. So I force the user to establish a secure channel to the archive system for whatever service the user is requesting.

An additional measurement is to limit the available access to the system. The goal is to minimize all accessible URL connections to the archive. For the normal user only two access points exist. This access is restricted to GET and POST requests (see figure 2.3). They should only be able to login, upload, download and nothing more. DELETE and UPDATE requests are specifically forbidden so that no accidental or malicious changes can happen. The concept of the access routes can be seen in section 2.1.2. For the administrator I have to offer a wider set of accessibility so that he is able to fulfill his duty. Eventually the administrator needs additional rights for updating or even deleting but this is not yet defined. In a newer version of the WebArchive a more elaborate concept for the administrator access routes should be created.

3.3.2 WORM

To ensure the data integrity it should absolutely not be possible to delete or alter any stored data. The best way to enforce this would be to use specialized hardware like WORM (Write Once Read Many). This type of storage only allows one to store a file once and is then only readable until the defined retention date. When the retention date expires, the file will be removed from the disk. This way it would be possible to ensure the data integrity on a very low level of the system. The downside is that a special hardware component is necessary which would be very expensive.

It is also possible to implement a WORM on a software basis. Although this way it is not possible to ensure the same security and data integrity as with a hardware WORM. In my project I concentrate on software-based WORM because I don't have access to resources to use hardware WORM. For the user it is not possible to alter or delete any files (see database concept in section 2.2.1) with the URL access (see Access points 3.3.1) but the files are all available on the server and any user with administration rights and access to the server is able to change existing CRUD permissions and delete or alter any file.

3.3.3 File Encryption and Certification

Each file that submitted to the archive is stored unchanged. An additional security measure would be to encrypt the file on the client side. During the time of storage no one would be able to view and alter the file without damaging it. In the process of decryption the client would realize the alteration. This would require an universally used desktop application which contains the secret and handles the encryption and decryption of the file.

The file would be sent to the server in an encrypted form and the corresponding metadata would be provided in the header or in a separate JSON body. This approach has some drawbacks and that's why I didn't use this implementation. The process of validation would only be possible if the server also has access to the encryption key, otherwise the WebArchive has to trust the user or application that uploads the file. Holding the key on the server would greatly reduce the effectiveness of the client side encryption. An attacker who has access to the server would in turn also have access to the encryption key. Although this would increase security on the server side, the server would be too reliant on the provided data of the client. I decided to store the data in the original representation in the server storage. This allows the WebArchive to validate all incoming files (more on file validation in section 3.4) and extract meta data directly from the file if possible.

A more common way to ensure the integrity of a file would be to add a certificate to the file. There exist different services and companies that offer a Public Key Infrastructure (PKI) which is able to issue certificates and also check issued certificates. A PKI is split into three main services. The Registration Authority (RA) handles all requests for a certificate. Users or machines call the RA for a certificate whereupon the RA checks the validity of the caller. If the user or machine is valid, the RA sends the request to the Certification Authority (CA). The CA returns the certificate. If another user or machine wants to check the certificate, he calls the Validation Authority (VA). The VA checks the certificate and lets the user or machine know if the certificate is trusted or not.

However, not all file formats support certificates. Sometimes it is not possible to embed the issued certificate directly in the file. For PDF this is easily possible and there exist various tools to achieve this. For formats like JPG it would be advised to use XMLDsig². With XMLDsig it is possible to store an additional XML file with the signature in it. This would be a detached signature and has to be stored close to the original. In the context of the WebArchive, this would be part of the AIP.

For the time being I decided not to use any PKI. The costs for such a service would be too high and most of our customers wouldn't be willing to pay for it. An approach we are currently exploring at the company is to issue a self-signed certificate. This would require for our company to build a self managed PKI. Nevertheless it would be a difficult process to build such an environment and we most likely would not be able to get a trusted certificate. That would mean that every tool and also browser (for SSL/TLS) would show a warning that this certificate is not officially trusted.

3.4 File Validation

According to the OAIS model it should be possible to archive all sorts of files with different formats. In every archiving system it is essential that the information to format

²www.w3.org/TR/xmlsig-core1 accessed 2018-01-08

and represent a file is also part of the AIP (and DIP). This allows the Consumer to read and make a file understandable. In the scope of this bachelor thesis it was not possible to implement such a system that would be able to support all kinds of file formats. The accepted file formats have to be narrowed down to the most popular and most used ones at the time of development. To validate the format I used an additional library called JHOVE³. JHOVE is a “format-specific digital object validation API written in Java”⁴. It was developed by the Harvard University Library. The tool checks a file in three steps:

Identification This is the process of determining which format the file has. It does this with different modules, one module for each format. Standard supported formats and corresponding modules are: AIFF, ASCII, bytestream, GIF, HTML, JPEG, JPEG2000, PDF, TIFF, UTF-8, WAV and XML. It would also be possible to write your own modules to extend the library and so check for additional formats with those.

Validation The prerequisite is that the format is known which is taken care of by the Identification. As soon as the format is clear, JHOVE tests syntactical correctness. If this test returns a successful response the file is well formed. The tool can to some extent also test for semantic requirements. If they are also correct the file is deemed valid. For some formats there exist standard representations from external sources and if the internal represented format is consistent with this specific standard then the file is deemed consistent.

Characterization When the file is at least well formed JHOVE extracts predefined values from the file. The standard characterization consists of: file pathname, last modification date, byte size, format, format version, MIME type, format profiles (optional: CRC32, MD5 and SHA-1 digests)⁵

The first step in the Ingest (see process Ingest in section 2.1.1.1) JHOVE takes initiative and identifies the format and verifies the file. Currently the WebArchive is restricted to the formats PDF, TIFF and JPEG. The reason for that is the popular usage of these formats and because the corresponding ERP system uses mostly these formats. It is possible to extend the WebArchive with additional formats, assuming that a JHOVE module for this format is available or can be created.

Currently not all standard characterizations of JHOVE are extracted and used. The WebArchive makes use of the last modification date, byte size, format and MIME type. The hash values are computed with the available Java library MessageDigest⁶. Despite this very useful tool the WebArchive relies on additional information. Authorship, file name and bucket name have to be provided by the Producer.

³www.jhove.openpreservation.org accessed 2019-01-08

⁴<https://en.m.wikipedia.org/wiki/JHOVE> accessed 2018-12-15

⁵<https://jhove.openpreservation.org/getting-started> accessed 2018-12-15

⁶<https://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html> accessed 2018-12-15

3.5 Storing Structure

The WebArchive implements a simple bucket system. The Management is responsible for maintaining the bucket system and creates (if necessary) new buckets. For every module in the ERP system at least one bucket is necessary (e.g. the material planning module has its own bucket for delivery orders). In figure 2.4 we see the database model. The table Bucket holds the actual path to the files and the corresponding bucket name and description. If all files were stored directly at the path where the bucket points, that would cause a chaos of unsorted files, so the system creates additional folders accordingly to the object ID of the file (see table Document in figure 2.4). For most of the files this object ID is given by the corresponding ERP system, otherwise the system assigns one to the file. The object ID will be formatted as a ten-digit number and split into three parts (e.g. object ID 3 will be formatted to 0000000003 and then split to 0000 + 000 + 003). This more structured system only allows us to store 1'000'000'000 - 1 files.

When we are talking about storing we don't want to forget about the backup storage. It is absolutely crucial that the archive system has a backup and is able to restore files in case of emergencies. The easiest way of creating a backup would be to set up a backup server which replicates the storage units of the archive system, so we wouldn't have to implement a backup service inside the archive system but would still be able to retrieve and access restored files if necessary. At the company where I work I have the possibility to use a backup server to ensure data security.

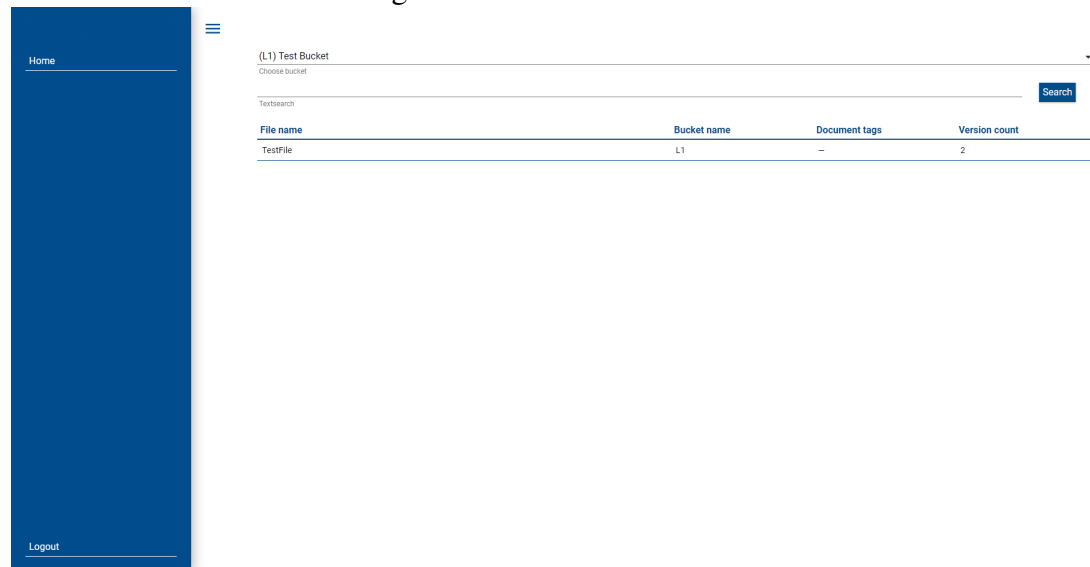
3.6 User Interface

Although I mostly focused on the machine to machine interaction for the archive, it was important to me to implement a simple user interface. This allowed me to see what is needed and how to proceed from there. A simple access platform didn't consume a lot of time but it revealed how much work still has to be done to present a useful and practical access panel. There's a lot you have to keep in mind and just small errors can drive a user crazy and can lead to never using the application again.

3.6.1 Search

The simplest search includes a text search so the user can type in a filename or part of it and with a simple LIKE command on the database, possible matches can be retrieved. This is the current implementation of the WebArchive (see figure 3.3). Based on the query and the chosen bucket a list of suggestions is shown. A list entry can obtain one or more versions. When an user clicks on a list element all available versions are shown (see figure 3.4). The user has then the possibility to download a file. The search is still missing a lot of elements: What if a user has a typo or wants also to search for similar

Figure 3.3: WebArchive Search



notations (eg. Meier and Mayer)? Should the results be loaded instantly or only on confirmation? What if the user searches for a category or tags? Does the search also already order and filter the results and how far can the user control this?

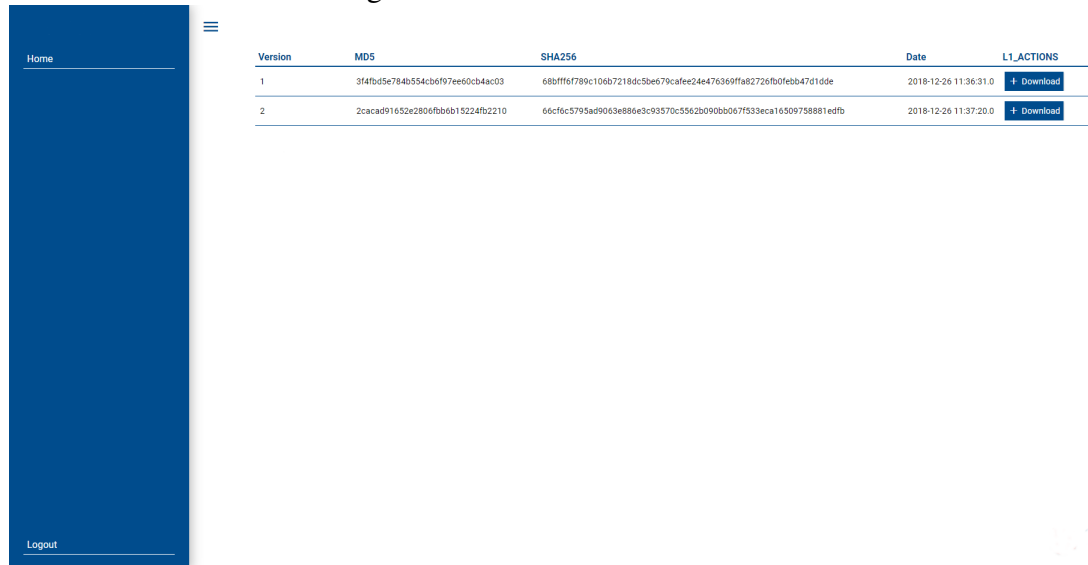
It would be possible to manually implement some of the requirements but it would probably be better to use an already existing search library. The company I am working at uses Elastic Search⁷ most of the time which is why I will probably also implement this search engine for the WebArchive. A search library brings a lot of value to the system but needs a lot of time to read into and a lot of effort to implement. That is the reason why in WebArchive 1.0 no such library is integrated. This answers the questions about typos, autocompletion and search performance but if the search engine should be able to search for tags or categories, the system has to gather information from the user and correctly store them in the database. The implementation of the search engine is the second step but first we should define the data that has to be provided to the WebArchive. To get an overview of what data we should store, we would have to meet with different customers and analyze the requirements.

3.6.2 Authorization

I already dedicated a subsection of this bachelor thesis to this subject (see 3.2). This part is more about the influence on the user interface. Like already mentioned, not every user should be able to see and download all files. The file system is structured

⁷<https://www.elastic.co/de/products/elasticsearch> accessed 2018-12-15

Figure 3.4: WebArchive Download



Version	MD5	SHA256	Date	L1_ACTIONS
1	3f4fb5e784b554cb97ee60cb4ec03	680ff1b789c106b7218dc5be679cafee24e476369ffa82726fb0febb47d10de	2018-12-26 11:36:31.0	Download
2	2cacad91652e2806fbb6b15224fb2210	66cf6c5795ad9063e88e63c93570c5562b090bb067f533eca16509758881edfb	2018-12-26 11:37:20.0	Download

and built according to a bucket system (see Storing Structure in section 3.5). The first layer of authorization is the bucket. The user is only able to select and search in those buckets where he is authorized. Each bucket is assigned to a module and not every user should have access to the documents of each module. The dropdown in figure 3.3 only contains the buckets which the user is authorized. The second layer includes the version of the file. Every version can be separately marked with a higher authorization level than the bucket itself, so it is not absolutely necessary to store delicate files in a separate bucket. That way I am able to rule out a lot of unallowed access already because the system only loads the content which the user is allowed to see. This is a pretty rough implementation of authorization and eventually has to be improved in a later stage of development. Especially when authorization will happen with OAuth2 or over an external authorization server.

3.6.3 Administration

The implementation of the WebArchive only allows the administrator to change settings in the background and directly in the database. A user interface doesn't exist at the moment and is planned for the next iteration of the WebArchive implementation. As you can see in section 2.1.2 some functionalities are already ready to be implemented. With these available routes the administrator should have access to a platform where he can manage all users and the contract for the submitted data. The goal would be for this platform to be extendable with additional features. The administration should also be

able to surveil the database and ongoing procedures.

The administration is a more complex platform so I didn't implement this part of the WebArchive yet because a more precise concept is needed. The embedding of the WebArchive in a web application environment could possibly change the management possibilities of the Administration. For example, the user management would not be a part of the WebArchive Administration as this would be handled through an external authentication and authorization server.

3.7 Summary

The result of this bachelor thesis is a simple but working solution for an archive system. Based on the guidelines and structure of the OAIS model I built the WebArchive. Single files (PDF, JPG or TIFF) can be uploaded through a UI. Also machine to machine interaction is possible through different URL access routes. After completing the file validation with the help of JHOVE, the file is stored on the harddrive and the meta data is stored on the database. The WebArchive operates as an independent web application and therefore has an own database. The concept for the database and the initialization also was part of the implementation. Along with the UI to submit files, an additional simple access platform is available to search for a file and eventually download it.

The documentation in this bachelor thesis and the prototype serve as proof of concept for an independent web application in the vicinity of the ERP system. During the process of this bachelor thesis it became more and more clear that it is currently not possible to embed the WebArchive in the environment of the ERP system. The important link to the ERP system and also a general interface for communication with other web applications is currently missing. The missing interface between web application is due to the lack of other web application in the environment of the ERP system. It is not yet clear what kind of web applications will be deployed and therefore no concept for the interface exist. The implementation of the URL routes is based on the features the WebArchive uses. Therefore other web applications and the ERP system are able to make use of the features of the WebArchive. However adjustments in the ERP system are necessary to ensure automatical exchange between ERP system and the WebArchive.

However the main focus remains on the digital archiving. The implementation of the prototype was only useful for myself, to proof that it is possible to build a digital archive based on the OAIS model. Furthermore the bachelor thesis showed that in theory it is possible to implement a valid digital archive system. Every section of chapter 3 point out different approaches to solving problems and describe the current implementation of the WebArchive. Not only do these sections help focus on most needed implementations in the next iteration of the WebArchive, they also provide approaches for general solutions in web application. The validation of the functionalities of the WebArchive were made be myself only. In this stage of development no other user tested the features of the

WebArchive. As soon as the connection to the ERP system exist, the prototype will be used as a demo project for interested customers and eventually I will be able to start a pilot project at one of the customers of the company where I work.

4

Conclusion and future work

4.1 Lessons Learned

The first thing I realized is that I've never designed a concept for such a big and complex system. The first struggle was where and how to start. I began to create a lot of documents and in a later state of my work I realized that some of the work was unnecessary or the information got lost in the documents. I think the beginning of the project was the hardest part because I was reading a lot and made no visible progress during this stage. In retrospect it was absolutely necessary to read about the subject because I had only a little knowledge about what archiving is about. Moreover I never implemented a web application on my own. This is why the creation of a concept consumed a lot of time and still had some flaws or wasn't thought through. It is really hard to create a meaningful and helpful concept, but thanks to this bachelor thesis I now understand better on what I have to focus on. I hope it is not the last time I designed a concept, as next time I could use my newly gained knowledge on the subject.

Another thing I realized was that I missed a co-worker or a team who also deals with the same problem. Sometimes I became lost in unnecessary details. That's when it would have been really helpful if someone said that I was on the wrong track, or if there was someone I could just share or discuss some ideas with.

All in all I expanded my knowledge about archiving and all the challenges that arises during the implementation of a digital archive. Furthermore I learned a lot about web application implementation and specifically about the framework we use at our company. Until now I barely worked with this framework and only touched the surface

of it. Developing the WebArchive forced me to dive deeper in the code and understand basic concepts of web applications. This knowledge now empowers me to extend my work on the web application development and contribute to it.

4.2 In Process

Right now the company where I work is setting up a more complex environment for web applications. The goal of the company is to implement a Self-Contained-System architecture for all their web applications. The WebArchive is one of the first projects that is implemented with this intention. Because this bachelor thesis was written during a time of active development and planning it is not in a definite state, it has to be extended and improved for future use. In chapter 5 I will further explain what is planned and what the final product should look like.

4.3 Enhancements

For almost every presented part in the section 3 there is at least one point I want or I have to improve. The current WebArchive is in Version 1.0 and can handle simple requests and offers a simple access view for users. This should not and can't be the final version of the WebArchive. For now I have to wait to see in which direction the company is evolving, and then adjust the WebArchive to the given circumstances. It would be best if the company can offer an external authentication and authorization server. This would allow me to completely delete the login mask from the WebArchive and could redirect the user to the login server. Apart from the environment of the WebArchive, the whole UI and the functionalities available to the user have to be improved. Especially the UI for the Consumer and the Management require a lot of changes. For the Consumer the search should be improved and for the Management the platform has to be created in the first place. Because of the flaws in the UI, I didn't make any user test and validation. I by myself tested the basic functionalities of the WebArchive. This is surley something I have to improve and enforce in near future for a more detail assesement of the WebArchive.

5

Anleitung zum wissenschaftlichen Arbeiten

The architecture of a system is the most important element to define. It is possible to start a project without a clear concept but soon the process of developing will be cumbersome and time-consuming. The structure of the system will become increasingly unclear. The scalability of the system will suffer from this kind of development. It isn't possible to make changes without affecting other parts of the program. That's why a concept is created in advance to the actual development. In my opinion we have to differentiate between two sorts: the architecture where an application is embedded and the architecture of the application itself. In the main part of the bachelor thesis I focused on the architecture of the application and pointed out some challenges that arose while developing a web application. I relied on the OAIS model for the architecture of the WebArchive. In this chapter I want to focus on the architecture for the web application environment. As soon as I started with the bachelor thesis the company I'm working at began to restructure the environment where the WebArchive was to be embedded. In this chapter I would like to explain some of the most common (and currently most discussed) architectures and try to show the consequences for the WebArchive and for a web application in general.

5.1 Architectures

All three of the following architectures can be used to organize and structure multiple systems in one environment. There exist many different approaches to set up an environment of different but logically related systems. I picked three of the most discussed approaches and will explain their architecture. I also try to point out the advantages and disadvantages of each architecture.

5.1.1 Monolithic Architecture

In a monolithic architecture all components are in one application and run on a single platform. The goal of such a system is that all requests can be handled in this single application and don't need additional services. All the information is centralized and the application is able to access this data from everywhere and at any time. Also UI and business logic have direct access to the data storage and are implemented in the same application. This allows a developer to add features and functionalities everywhere he likes and he doesn't have to concern himself too much with messaging or data availability. A monolithic system tends to grow and get tightly coupled in the process. This results in difficulties when replacing or maintaining parts of the system. It becomes harder and harder to keep an overview. Even small changes can trigger major problems somewhere else.

5.1.2 Self-Contained System Architecture

The self-contained system architecture consists of multiple SCS (self-contained systems). Each SCS is an independent application with an own domain, data set, UI and logic. An SCS is able to operate in a single instance and can handle requests on his own. Generally a SCS doesn't share the UI (only hyperlinks to other applications are allowed) or even data with other SCS. However, it is sometimes needed to send messages or gather information from another part of the environment. This communication is always through interfaces and it always has to be asynchronous. This architecture offers high resilience and maintainability. If one SCS is down all other SCSs are not affected and can continue working. This also allows developers to easily update or replace single parts of the system without interrupting the other SCSs from working¹.

5.1.3 Microservice Architecture

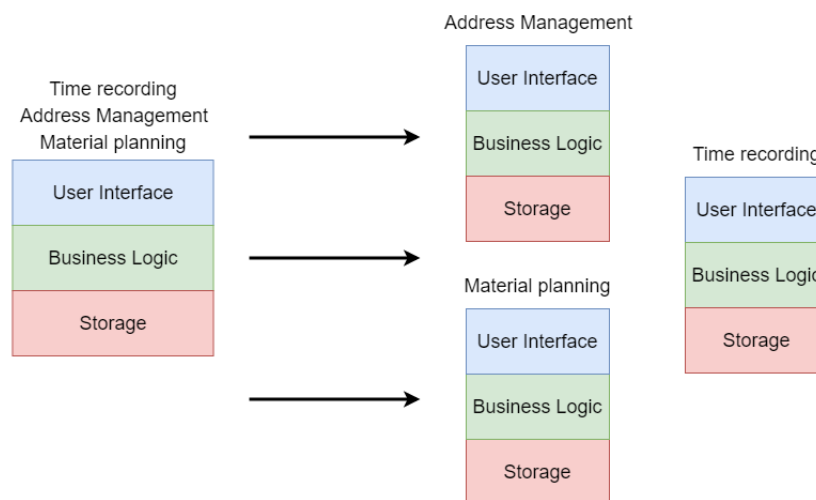
There are a lot of similarities to the self-contained system architecture in a microservice architecture. The system is modularized and the functionalities are distributed to multiple

¹<https://scs-architecture.org/> accessed 2018-12-13

microservices. The main difference to a SCS is that the modules are smaller. SCS can span a whole system (e.g. WebArchive) whereas a microservice only would take a small part of the WebArchive (e.g. Ingest) and communicate with other microservices. The architecture consists of two parts: Macro Architecture (decisions that concern all microservices) and Micro Architecture (decisions that only concern a single instance). The advantage of microservices (and SCS) is in this differentiation macro and micro architecture. Each microservice can be implemented with the needed technology and specifically target difficulties of this microservice. If each microservice is implemented differently, that means that the communication with other microservices has to be solid and standardized. Each microservice has an interface to share data with other microservices. Unlike the SCS architecture, the microservice architecture relies heavily on communication between microservices².

5.2 Targeted Architecture

Figure 5.1: Evolving process



The ERP system that is used is a purely monolithic system. It provides multiple modules but they're all packed into one program, access the same data set and are made visible through the same UI. In the near future, this architecture for the ERP system has

²Further reading under <https://www.innoq.com/en/articles/2018/12/microservices-oder-doch-nicht/> accessed 2018-12-13

to change gradually. The company where I work wants to expand their web application products. More and more customers would like to access some of the functionalities through a browser and don't want to use just the desktop application. A problem that arises with this requirement is that the customer must already be in possession of the ERP system because at the moment the ERP encapsulates and provides all data. To enable a customer to buy only what he really needs, we have to move away from the monolithic solution step by step. The first move is to arrange all web applications in a SCS architecture. The ERP already separates functionalities in different modules which would be great to adopt into web applications. That is the reason for the company to focus on SCS architecture. This measure would allow for continuous module development and separate deployment without affecting already deployed web applications.

5.3 Consequences for the WebArchive

The implementation and restructuring from a monolithic to a SCS architecture has a great impact on all web applications and also on the subject of this bachelor thesis: the WebArchive. All applications have to be embedded in a SCS architecture. In this case the WebArchive represents an SCS. With the definitions in mind for a SCS I had to consider some points that will influence the future work of this project.

5.3.1 User Interface

Because the WebArchive is one of the first web applications that is implemented in the environment of the company, the user interface is uniquely used for the WebArchive and doesn't use shared UIs. Naturally, the UI shares a corporate design for an uniformed appearance. The appearance is currently part of a standardized framework that is used for every new web application. Eventually we have to move away from a standardized framework and instead create guidelines for future projects. The goal should be to provide a uniformed appearance but not interfere with the basic structure and choice of technology. Each SCS should be able to use whatever technology or structure that is suited best for the underlying logic. This makes shared UIs difficult because under certain circumstances multiple SCSs don't implement the same technology. Nevertheless it is allowed and probably best to work with hyperlinks to redirect directly to another SCS. In one approach of implementing the authentication and authorization I only used a hyperlink to an external server for the login. For future applications OAuth2 is planned and absolutely necessary so that the user doesn't have to login for each SCS.

5.3.2 Data Set

In the SCS architecture each SCS is obliged to store the necessary data itself. For this purpose the WebArchive has its own database. The generated data and the cache is stored internally. In the current implementation, the WebArchive relies on the user information from the ERP system. We have to implement an external authentication and authorization service, optimally with the OAuth2 protocol. That would eliminate the synchronisation of user data with the ERP. Otherwise the WebArchive doesn't rely on other SCS because right now, no other SCS exist. Despite the lack of other SCSs the WebArchive already has a defined interface and can handle incoming requests. The data is not distributed to other systems and is kept on the database. In the first phase of planning, it was decided that the archive would just handle the files. The metadata would be stored on the centralized database from the ERP system. Relatively early it became clear that bigger changes are coming so I decided to move the data to the WebArchive completely. This saved me a lot of refactoring later on and it brings more value to the WebArchive because it can operate more independently.

5.3.3 Business Logic

A SCS should only use code that is part of the implementation itself. No code or procedures of other SCSs should be used. This approach avoids tight coupling but can lead to duplicated code. In section 5.3.1 I suggested to eventually reduce the content of the given framework. This is a trade-off because in this situation it might be worth it to have a package to avoid having to write duplicated code. In the WebArchive we use a framework which makes some useful functionalities available. Despite the usefulness of this framework we have to refactor a lot and perhaps create a completely new concept. Something that is probably worth standardizing is the handling of requests from other SCSs. The framework should be able to handle requests based on syntax and the SCS handles the semantic translation. However we should consider that not every SCS is built with the same technology. A standardized library may be the best solution if we think that a change of technology will be unlikely or that a project with different technology will be very rare.

5.3.4 Technology

With the introduction of SCS I would be free in choosing an appropriate technology. The single requirement for this technology would be that the communication with other SCSs should be provided (if necessary). It may be a good idea in further development of the WebArchive to take some time to evaluate different technologies (e.g. for database, coding language, libraries). Because the project was very ambitious, I took advantage of most of the tools we use currently. This gave me a little edge in the development process

because I was already more or less familiar with them. A complete evaluation about whether or not the appropriate database or coding language is used for the WebArchive was not done.

Bibliography

- [1] CCSDS(The consultative Committee for Space Data Systems. *Reference model for an open archival information system (OAIS)*. CCSDS Secretariat; Space Communications and Navigation Office; 7L70; Space Operations Mission Directorate; NASA Headquarters, 2012.
- [2] Microsoft Corporation. SSL Handshake [graph]. Website, accessed 2018-12-25. <https://blogs.msdn.microsoft.com/kaushal/2013/08/02/ssl-handshake-and-https-bindings-on-iis/>.
- [3] Ed D. Hardt. *The OAuth 2.0 Authorization Framework*. Internet Engineering Task Force(IETF), 2012.
- [4] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. University of California, 2000. Doctoral dissertation.
- [5] S. Chokhani T. Polk, K. McKay. *Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations*. U.S. Department of Commerce; National Institute of Standards and Technology, 2014.