DICOMmunicator

An Online Medical Image Transfer Service

J. Aué
R.H. van Staveren

Delft University of Technology

TU Delft Delft University of Technology

Challenge the future

DICOMmunicator

An Online Medical Image Transfer Service

by

J. Aué
R.H. van Staveren

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Computer Science

at the Delft University of Technology

Supervisor:	Dr. Z. Erkin,	TU Delft
	Dr. W. Winterbach,	Clinical Graphics
Thesis committee:	Dr. Z. Erkin,	TU Delft
	Dr. M. A. Larson,	TU Delft
	Dr. W. Winterbach,	Clinical Graphics

TU Delft Delft University of Technology

Clinical Graphics See Beyond

PREFACE

This is the final report which describes the development of DICOMmunicator, an online medical image transfer service.

In the report we discuss the problem that is faced when transferring medical images. The approach used to solve the problem is explained and a list of requirements is formed. Research on the problem and the requirements concluded in a design. The implementation is described and the application and process of the project are evaluated. Finally the report is concluded with recommendations for future development of DICOMmunicator.

The project was completed for the Bachelor Project course for the Computer Science Bachelor program at the Delft University of Technology and Dr. P. Krekel from Clinical Graphics issued the assignment.

During the course of the project we were supported by the Delft University of Technology and the team of Clinical Graphics. We would like to thank the following people.

- Zeki Erkin, from the Delft University of Technology, for supervising the project, stimulating us to think critically and providing us with feedback.
- Peter Krekel, from Clinical Graphics, for the project assignment and providing us with all the necessary support.
- Wynand Winterbach, from Clinical Graphics, for supervising the project, sharing his ideas on functionality and providing us with support.
- Korijn van Golen, from Clinical Graphics, for providing us with support on best practises and testing as well as answering our programming questions.
- Martha Larson, from the Delft University of Technology, for coordinating the Bachelor Project and taking place in the thesis-committee.

Contents

1

INTRODUCTION

1.1. PROBLEM DEFINITION

Medical doctors and researchers have to send medical image files to each other. This may be for a consult, because a patient is treated by a different doctor or because the files are to be used for research. The files contain personal health information, and privacy rules have to be satisfied before the files may be sent [1]. To securely send the files the data should be encrypted and anonymized.

Section 1.1.1 explains the difficulties of anonymization. Section 1.1.2 describes how doctors currently send medical files to each other. Security threats that come with the transfer of medical files are listed in Section 1.1.3. The problems that arise in securing a file in terms of encryption are described in Section 1.1.4. Existing applications that are possible solutions to the problem are discussed in Section 1.1.5. Finally in Section 1.1.6 the problem is summarized.

1.1.1. ANONYMIZATION

Medical image files use the data exchange standard called Digital Imaging and Communications in Medicine (DICOM) [2]. A DICOM file consists of medical images and information about a patient. In order to meet the privacy rules concerning patient information a DICOM file needs to be anonymized before transfer. This is done by sending the medical images files to the Radiology department to have the patient data removed. However, the anonymization of files by the Radiology department takes up to two weeks, which is the reason for a doctor to neglect this step.

1.1.2. FILE TRANSFER

For the actual sending of the data CD-ROMs are used regularly. This is time consuming and not always secure because CDs can be lost or are not destroyed after the information it contains is used. If the data is not anonymized before being burned on a CD it is very unlikely this will be done later.

Beside CDs, services like WeTransfer [3], a platform to send large files from A to B, are used. WeTransfer is easy to use, but is not built for sensitive medical data that should be anonymized and securely sent.

Another issue is that doctors want to send their files without putting effort into it. They do not want to register an account or make a payment to be able to send data to another party.

1.1.3. SECURITY THREATS

Malicious parties would like to access a patient's medical data. For example, insurance companies can take advantage of the medical records of a patient and charge the patient a higher monthly fee for their health insurance.

Another malicious user can be a journalist that is after an interesting story on a celebrity whose medical files are being sent to another doctor for a consult.

1.1.4. ENCRYPTION

Doctors, nurses and researchers often do not have the knowledge to secure patient data sufficiently. Generally medical data is not encrypted and in the other cases a simple pass-phrase is often used.

In the situation that the data is encrypted a doctor is faced with the underlying problem of key distribution. The key used for decryption cannot be communicated to the receiver over communication channels that allow eavesdropping, like email or the phone.

1.1.5. INADEQUATE SOLUTIONS

Services exist that focus on the secure sending of information. These services are either specifically for medical data, or for data in general. The following sections list these services and discuss why these services are not a solution to the problem.

CLINICAL GRAPHICS' LIGHTBOX

Clinical Graphics' Lightbox [4] is a web application used by the company's customers. The upload of the files is done using Silverlight [5]. Files are anonymized in the user's browser and are transferred over an Secure Socket Layer (SSL) connection.

The application requires the user to have Silverlight installed. However, most hospitals will not have this framework installed and doctors often do not have permission to install software on hospital computers. Since the framework often is not available on hospital computers it is not an adequate solution to the problem.

IMAGE32

Image32 [6] is an application used to share medical data. Users can upload and download medical data securely and the service covers all rules associated with the transfer of medical data.

The drawback is that Image32 requires a fee after a 15 day trial. This is a hurdle to users that would not use the service frequently. Beside that, the hospital would need to approve the payment, which on its own will take at least a week. Not the ideal choice for quick uploads with minimal efforts.

DICOMLIBRARY

DICOMLibrary [7] is an online DICOM file sharing service. This service is used for educational and scientific purposes and anonymizes DICOM files before uploading. A link to the download location is provided and the files can even be viewed in an embedded viewer.

A drawback of this system is that after uploading, the files are accessible for everyone. Perfect for scientific purposes, but unacceptable for sending patient data to specific recipients. Another downside is that the system uses the Adobe Flash Player [8], which may not be installed on every computer, and doctors are normally not allowed to install software on hospital computers. In this case not every doctor could use the service, meaning that the service cannot be used to solve the problem.

POPBOX

Popbox [9] is a tool used to send encrypted messages or files. The files and messages are encrypted before leaving the browser and a URL is generated which can be used to access the file.

However Popbox is not specialized in DICOM files and thus does not anonymize the files. It is an example of client-side encryption, but unfortunately Popbox allows only a maximum file size of 5 MB and the user needs to deliver the URL to the addressee himself.

PICTURE ARCHIVING AND COMMUNICATION SYSTEM

A Picture Archiving and Communication System (PACS) [10] is used for archiving and transmitting medical images. PACSs can pre-fetch images to a short-term server to provide a fast retrieval before usage the next day. PACSs exists that pre-fetch pictures taken earlier for comparison.

A PACS however is not accessible by everybody. A user can only use a PACS if he is registered to it. Furthermore special software like DCMTK [11] or Powertools [12] is necessary to connect to a PACS, which not all doctors have access to. Besides, PACSs can differ between hospitals and a therefore a connection between them is not always possible.

1.1.6. PROBLEM SUMMARY

In summary a service is missing that facilitates the anonymizing, securing and sending of DICOM files from one doctor or researcher to another without barriers to use the service, like registering or making a payment.

1.2. ASSIGNMENT AND CONTRIBUTION

The problem resulted in the following assignment [13] issued by the company.

> The amount of medical image data being transferred via the internet increases rapidly. Medical image data deviates from normal data in that privacy sensitive patient information is typically encapsulated in the image data. Regardless of this privacy issue, clinicians tend to use existing data transfer services such as WeTransfer and YouSendIt to transfer the data, in lack of a better alternative.
>
> In this assignment we propose to develop an easy-to-use file transfer service for medical image data. The usability of the service should be comparable to the WeTransfer-like services, as we should prevent creating a barrier for clinicians to use this service. In addition to this usability requirement, functionality requirements include file encryption to guarantee safe transfer of data, file compression to make the transfer of data faster, and finally file anonymization to optionally remove personal information from the data before transferring it. The front-end and back-end of this system should be separate systems, allowing us to easily add or remove functionality to/from the system (e.g. online viewers for medical image data).

The problem led to the development of DICOMmunicator, a single page web application. It allows clinicians to use the self-descriptive application to send DICOM files to another doctor. DICOMmunicator removes patient information from the files that are uploaded, encrypts the files and stores them in a database. The recipient of the files is sent an email containing the key used for decryption allowing only that person to download the files.

1.3. OUTLINE

The establishment of DICOMmunicator is described in this report.

In Chapter 2 the project methodology is described. Then, in Chapter 3 the requirements gathered are listed. Chapter 4 elaborates on the choices made during the design phase. Furthermore it gives an overview of how the system behaves, discusses the system architecture and describes the components of the user interface. The implementation is covered in Chapter 5 by listing the frameworks used, and explaining the functionality and testing of the application. Chapter 6 evaluates the process and product, draws a conclusion and mentions recommendations for future work on the application.

2

METHODOLOGY

This chapter describes the project methodology used during the project. Section 2.1 enumerates the different phases of the project. Section 2.2 explains the agile software development method used. The choice of a version control system is given in Section 2.3 and Section 2.4 discusses the role division in the team.

2.1. PLANNING

The project was planned to have separate phases. A timeline of the project per week is given in Figure 2.1.

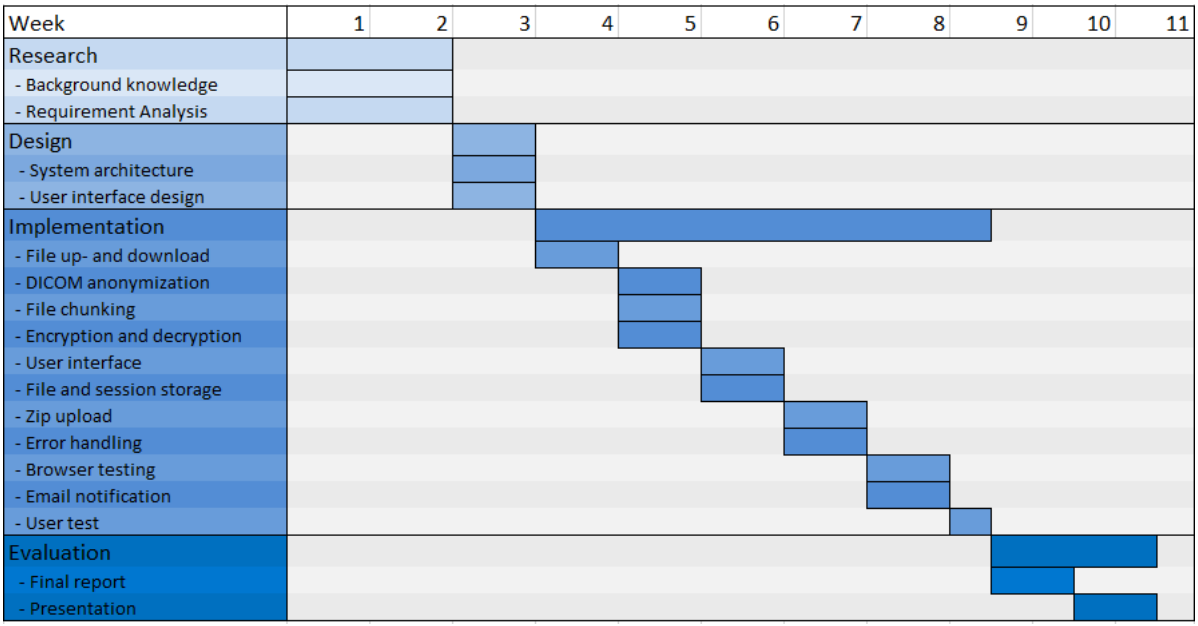

Figure 2.1: The project timeline per week.

Research phase *(two weeks)*
The problem was defined in the research phase. The requirements were gathered and analysed. Furthermore the frameworks and programming languages used were studied.

Design phase *(one week)*
During the design phase the architectural design was made and the user interface was designed.

Implementation phase *(five-and-a-half weeks)*
During this phase the application was implemented according to the design.

Evaluation phase *(two weeks)*
In this phase user testing was done to evaluate the application. Furthermore this final report was writ-

ten, which contains an evaluation on the process and product. This phase has also been used to prepare a presentation to demonstrate the process and product.

During the project weekly meetings were held with both the TU coach and the client advisor. The meetings were held to give demos of the implemented features, have discussions on problems and were used to brainstorm on solutions. All meetings were summarized and the summary was emailed to the participants. This way all decisions were documented and this allowed the participants to look back at the result of previous meetings.

2.2. AGILE DEVELOPMENT

For this project the agile software development framework called Scrum [14] was used. Scrum is iterative and incremental, which means that after each iteration a shippable product can be released. Iterations of one week would suit the project planning best. This is due to the fact that a five week period was planned for programming. Besides that there would be weekly meetings with the client and coach, and the features to implement were estimated to take at most one week of programming. For these reasons a one week iteration period was chosen.

Pivotal Tracker [15] was used to keep track of the product backlog containing all features to be added. At the start of the week the backlog would be looked at to decide which features would be implemented that week.

2.3. VERSION CONTROL

GitHub [16] was used for version control and this allowed the creation of working branches for each new feature. When one team member finished a feature, a pull request was created and another team member would test the feature by hand to make sure everything behaved as intended. If a mistake was found, a comment would be left at the pull request for the initiator to look at and to apply a correction to the code. When new code behaved as it should the code would be merged with the working version of the system.

2.4. ROLE DIVISION

Due to the scale of the project it was not useful to have one team member in charge of different aspects like, for example, testing. All team members were equally responsible for every aspect of the project.

Furthermore the work was split at the beginning of each week. Besides dividing the total work evenly the tasks were also divided on difficulty, interest and potential learning experiences.

3

REQUIREMENTS

The requirements for the application were gathered by talking to the company employees and were split into the following three groups.

- Overall system requirements
- Front-end requirements
- Back-end requirements

Front-end is defined as the client-side of the system; in this case the user's browser. By back-end the server-side of the system is implied including any external databases that are linked to the server. This approach was chosen as it gives a clear overview of which requirements belong to which component.

Section 3.1 explains the methodology used to prioritize the requirements. The overall system requirements are listed in Section 3.2. The requirements for the front and back-end are shown in Section 3.3 and Section 3.4 respectively.

3.1. MoSCoW methodology

Due to the size of the project and the tight deadlines, the scope of the implementation had been limited. In order for a product to be delivered, which fit the needs of the client best in the given timespan, a prioritisation method was needed. The MoSCoW prioritisation methodology [17] was used to prioritise the requirements according to the company's needs.

3.2. Overall system requirements

The requirements of the overall system are given below.

- The user *must* not need prior knowledge of the system to use the application.
- The user *must* be able to use the application without installing new software.
- The system *must* not send any recognizable patient data over a network.
- The data *must* be encrypted during transfer over a network.
- The system *must* have a module based structure.
- The system *must* be able to upload hundreds of files whose combined size reaches up to 1 GB.
- The system *must* comply with European and American regulations for medical data.

3.3. FRONT-END REQUIREMENTS

The front-end requirements were split into two categories.

- User interaction requirements - This category lists the requirements that cover usability and utility.
- Functional requirements - This category lists the functionality requirements that are part of the front-end. The processing that occurs before uploading is covered by these requirements.

3.3.1. USER INTERACTION REQUIREMENTS

- The system *must* not require the user to register for use of the application.
- The user *must* be able to drag and drop a file that the user wants to send onto the application.
- The system *must* not require any user input after the upload is configured.
- The sender *must* be able to send multiple files at once.
- The system *should* work on all popular browsers; Firefox 30+, Chrome 33+, Safari 7+ and Internet Explorer 8+.
- The system *should* have the interface designed according to Norman's principles of design for usability [18].
- The sender *could* send zip archives with one or more DICOM files.
- The interface *could* give a preview of the images that are in the DICOM file.
- The user *won't* be able to register an account after uploading a file for future use of the application.

3.3.2. FUNCTIONAL FRONT-END REQUIREMENTS

- *Must* anonymize the DICOM files by removing the patient fields in the DICOM files.
- *Could* compress DICOM files before sending over a network.
- *Could* decompress DICOM files locally in the receiver's browser.
- *Could* send, in case of duplicates, files only once.
- *Won't* detect full face photographic images and remove them, as they can identify the patient.

3.4. BACK-END REQUIREMENTS

- The system *must* inform the addressee how the sent file(s) can be retrieved.
- The system *must* store files encrypted.
- The system *should* log the uploads and downloads of files.

4

DESIGN

In order to design the system a general idea of how the system should work was needed. Section 4.1 lists the key features of the system. The definition of secure is given in Section 4.2 as well as the security assumptions made. Design choices that have been made are elaborated on in Section 4.3. A flow diagram of the system was created, which is explained in Section 4.4. The architecture of the system will be discussed in Section 4.5. Section 4.6 concludes with a design description of the user interface.

4.1. KEY FEATURES

The system was designed by first determining the key features the system should have according to the requirements. By listing the key features it was possible to gain a general idea of how the system should work. These key features are:

- File transfer
- Anonymization of DICOM files
- Compression of files
- Encryption and decryption of files
- File storage
- Large file support
- Notification

From this list was deducted that the anonymization of DICOM files should take place before compression and encryption, as the application would not be able to read and change values of a file in an encrypted or compressed state. The files should be stored after encryption has taken place.

4.2. SECURITY DEFINITION AND ASSUMPTIONS

The definition of secure is to be determined if it is to be used to describe the application. Secure can be defined in terms of confidentiality, integrity and authenticity [19] as is explained below. The section will conclude with a list of assumptions that determine the definition of secure in the scope of this application.

CONFIDENTIALITY

The confidentiality of data is defined as the protection of the data from disclosure by unauthorized parties. In other words, making sure that data is only read by those who are supposed to read the data.

By using Hypertext Transfer Protocol Secure (HTTPS) as the transfer protocol for the communication between the server and the client confidentiality is ensured during the transfer. The protocol is used to prevent wire-tapping, which is when a malicious user listens in on the communication between two parties. HTTPS uses the SSL protocol, which uses certificates distributed by trusted certificate authorities. These certificates

are used to exchange a key between both ends of the communication line. The key is then used by the communicating parties to encrypt their data before transferring it. The level of protection however depends on the browser that is used, the implementation of the server software and the encryption algorithm.

On a storage server, confidentiality is ensured by storing the data in encrypted form. When a malicious user gains access to the data it will be unreadable and therefore unusable as long as the user does not have the decryption key. The key for decrypting the data should not be stored on the same server, as this defeats the purpose of encryption.

INTEGRITY

Data integrity means that data is protected from modification by unauthorized parties. Data in this case may refer to medical data, but also to website scripts, which are downloaded to the user's computer for execution.

Integrity can be improved by using the HTTPS protocol as well. The protocol is used to prevent man-in-the-middle attacks, where a malicious user makes it look like two parties are communicating directly with each other. However, in reality, both parties communicate with the malicious user and the person can decide to send a malicious script instead of the original script. A harmful situation where files are sent to the malicious user instead of the server can be the result. When the connection is encrypted using the HTTPS protocol a man-in-the-middle attack cannot take place, since the man in the middle does not possess the encryption key to, for example, encrypt a malicious script.

Integrity on a server is compromised when an intruder gains either physical access or remote access to the server. Therefore the physical server should be located in a secured environment where unauthorized access is prevented. Companies that provide web servers usually ensure this type of protection. Remote access to a server by an unauthorized user can occur when this user exploits a security flaw in the server's software. To minimize this risk server software should be kept up to date.

Another security breach occurs when a malicious user gains access to the server's password, which allows the person to access the contents of the server. To avoid this breach a strong password should be created, which cannot be guessed, neither by man and computer.

AUTHENTICITY

For authenticity it is important to validate that each of the parties involved in a transfer are who they claim to be.

This is achieved by using digital signatures provided by a certificate authority. Assumed is that both the owner of the certificate, the web server, and the party relying upon the certificate, the user, trust the certificate authority.

ASSUMPTIONS

The assignment is to develop an easy-to-use application comparable to WeTransfer-like services. Besides that, the application should make use of file encryption to allow a secure transfer of files. Furthermore anonymization of the files should take place as well. To fulfil these requirements assumptions had to be made on which parties to trust, in order to define what secure means within the scope of the application. The decisions to use Microsoft Azure and Mandrill are given in Section 4.3.

Clinical Graphics Assumed is that Clinical Graphics makes sure that the authorization details of the system are not compromised.

Microsoft Azure Assumed is that Azure provides physical security of their server and uses up-to-date server software.

Certificate Authority The certificate authority that supplies the certificate, used by the application for HTTPS, is trusted to not share the private key that can be used to sign the application's certificate.

Mandrill Assumed is that Mandrill protects email from being read by anyone unauthorized.

Sender Assumed is that a person using the application to send files has taken precautions to prevent unauthorized access to the person's computer.

Receiver Assumed is a person that uses the application to download files uses an email service that supports email encryption during transfer.

4.3. DESIGN CHOICES

In order to design a system design choices had to be made. Section 4.3.1 explains the size DICOM files come in and discusses the memory problem that arises when processing DICOM files. Decisions on encryption are given in Section 4.3.2. Anonymization technicalities and processing are discussed in Section 4.3.3. Furthermore the storage service that will be used is explained in Section 4.3.4. Finally in Section 4.3.5 the choice of using email as means of communication is substantiated.

4.3.1. FILE PROCESSING

By searching for DICOM files online [20] and by contacting the company for samples the file size DICOM files come in was determined. A single DICOM file's size varies from 50 KB to 4 MB. However, multiple DICOM files are part of a single study and these files are saved in archives. The size of these archives varies from 5 MB to 280 MB as, for example, multiple 2D image slices of a 3D model are contained in the archives.

The limit of the total size of the files is set at 1 GB. There is no particular reason for this limit besides that in extreme cases, where an archive is larger than 280 MB, the archive can still be sent. The system will be able to handle files with a combined size of 1 GB.

Throughout the scope of this section a file, or multiple files, with a total size of 1 GB is/are referred to as a large file or large files.

A memory problem that is faced when dealing with large files in the browser is discussed next .

MEMORY PROBLEM

Browsers do not have access to a user's file system [21], so before a DICOM file can be processed it needs to be read into the browser's memory. Tested was whether this would work for large files in the latest versions of Firefox, Chrome and Internet Explorer by reading dummy files of various sizes into the browser's memory. Unfortunately only Firefox was able to deal with files larger than 250 MB in size. Beside the problem that browsers have with large files in memory, hospital computers are often outdated and do not have these amounts of memory available.

In order to avoid these problems input files will be split into chunks of 1 MB as the total overhead that is created by 1MB chunks is negligible in contrast to the total overhead of smaller chunks of, for example, 100 KB chunks.

Client-side processing in the sender's browser in terms of anonymization, compression, encryption and uploading can be done using chunks. The problem of this approach is faced when the file chunks are sent to the receiver of the file. The decryption and decompression can take place on the chunks, however for the file to be downloadable it will need to be reconstructed from the chunks in the browser's memory. But because browsers cannot handle large files in memory the reconstruction is not possible. In order to solve the problem the receiver must be able to directly download the file, without the receiver's browser having to read the file into memory. Decryption therefore has to be done on the server, just before downloading starts.

4.3.2. SERVER-SIDE ENCRYPTION

Section 4.3.1 explained why the decryption of a file client-side is infeasible and that uploading should take place in chunks.

Decided was that encryption should take place on the server as well. By using the HTTPS protocol the system is strictly not less secure when the files are encrypted before storage on the server instead of on the client's computer before transfer. When the HTTPS protocol is used for the communication between, in case of the application, the client and the server the communication is encrypted. This means that any file that is sent from the client to the server is encrypted during the transfer. Unfortunately, there are two moments in time when a file is unencrypted on the server; when the file is decrypted after the transfer and just before the download takes place as files are then decrypted. If the server is compromised files may be read at arrival and departure.

However, if client-side encryption was used and the server were to be compromised by a malicious user the following situation could take place. The script responsible for encryption could be changed by the malicious user to not perform encryption at all or worse, be replaced by a malicious script to send the files without encryption to another server.

From this reasoning was concluded that server-side encryption would not make the system less secure, under the assumption that the certificate used by the HTTPS protocol can be trusted and that the server uses a state-of-the-art algorithm for the encryption during transfer.

The encryption specification used by the application is described next.

SYMMETRIC-KEY CRYPTOGRAPHY

Symmetric-key encryption methods have the property that the same key is used for both encryption and decryption. In contrast to asymmetric-key encryption that makes use of a public key for the encryption and a private key for the decryption of files. Advanced Encryption Standard (AES) [22] encryption, a National Institute of Standards and Technology specification, will be used for the encryption of files. AES is fast compared to public-key encryption methods, but it also comes with a disadvantage. Since the same key is used for both encryption and decryption, the sender and the receiver will need a secure channel to transmit the key.

4.3.3. SERVER-SIDE ANONYMIZATION

The anonymization of DICOM files has both legal and technical aspects which are discussed in this section. Furthermore the anonymization process is discussed.

LEGAL ASPECTS

In order for DICOM files to be transferred and used in research for example, information that can be used to identify the patient must be removed. The information that needs to be removed is part of the protected health information (PHI) of a patient [23]. From the US Health Insurance Portability and Accountability Act (HIPAA) [1] is deduced that there are 18 identifiers to remove in order be sure there is no PHI present in the data [24]. The identifiers that are of interest for DICOM files are [25]:

- Names
- Dates related to an individual
- Phone numbers
- Email addresses
- Medical record numbers
- Health insurance beneficiary numbers
- Device identifiers and serial numbers
- Full face photographic images and any comparable images
- Any other unique identifying number, characteristic, or code except the unique code assigned by the investigator to code the data

These identifiers are to be removed for a file to be anonymized. Among the identifiers on the list there is one which yields a problem. Since a DICOM file may contain any image, a DICOM file can contain a full face photographic image or any comparable image. Face recognition can be implemented to filter these images out, however this was not done during this project.

THE DICOM FILE FORMAT

A set of medical images together with information, relevant for a clinical study, can be contained in a DICOM file. Patient information, study details and equipment that was used are examples of the extra data that may be added. In order to anonymize DICOM files the identifiers of interest as described in the previous section need to be removed before uploading. In order to do so a deeper understanding is required of the DICOM file structure.

A DICOM file is made up of multiple data elements. Each of these elements contain a special tag, which is used to identify the element, a value representation (VR), for example 'Unsigned Short', a value length and a value. The VR may not be present, since it can be derived from the tag. Figure 4.1 illustrates the structure of a DICOM file. By consulting the DICOM standard [25] a list of tags was composed that complies with the identifiers of interest. This list is used for the anonymization process and is given in Appendix B.

A DICOM file is encoded using three of the following syntaxes:

- **Explicit VR Little Endian** The DICOM file is stored in Little Endian and each element contains a VR.
- **Explicit VR Big Endian** The DICOM file is stored in Big Endian and each element contains a VR.
- **Implicit VR Little Endian** The DICOM file is stored in Little Endian and the elements do not contain a VR; it is derived from the element's tag.

The result is that the application should be able to process DICOM files that use any of these syntaxes.

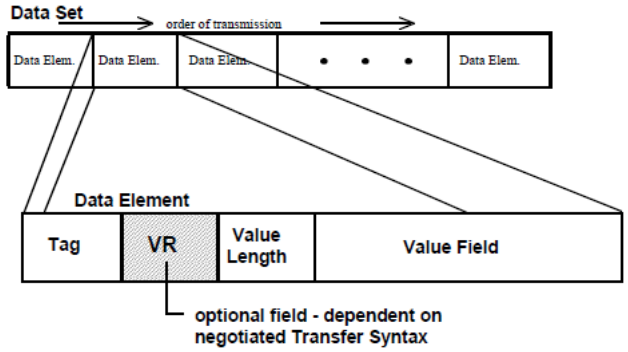

Figure 4.1: Illustration of DICOM element encoding in a DICOM data stream [25].

ANONYMIZATION PROCESS

Anonymization on the client-side of the system will ensure that no patient information in transferred over a network at all. Microsoft Silverlight [5], Adobe Flash [8] and the Java plug-in for browsers [26] can take care of this, however these options all require the respective platform or framework to be installed on the client's device. Hospital computers may not have the required software installed and doctors are usually not allowed to install applications. Therefore, the options described cannot be used for the application.

A combination between HTML5 and JavaScript is a better solution, which requires no installation. Feature detection is to be applied in order for the application to work in all required browsers as well as the implementation of a polyfill, providing facilities that are not built into a browser, if there is no native support for a certain feature. Internet Explorer 8 and 9 however, do not support the functionalities of HTML and JavaScript that are required for file processing. To support anonymization in these browsers anonymization will have to be implemented in a platform like Flash.

Unfortunately, the solution would imply that there will still be users who have an incompatible browser and do not have Flash as well. Therefore the decision was made that server-side anonymization is adequate for this project instead of client-side anonymization. Section 4.3.2 has explained why server-side encryption is not less secure than the client-side variant. The same reasoning is applied to anonymization. In the case of client-side anonymization a malicious user that has access to the server could change the anonymization algorithm to not perform any anonymization at all.

The requirement that stated client-side anonymization was adjusted after consultation and agreement with the company.

4.3.4. STORAGE

The uploaded files will need to be stored, so that they can be retrieved later. File storage will be provided by the Microsoft Azure cloud platform [27]. Azure's advantage over maintaining a private database server is that Azure takes care of the maintenance, scaling and security. Besides that Azure is cheaper than renting a private server, since with Azure one only pays for the service usage. Azure's SQL Database and Binary Large Object (BLOB) Storage service will be used for file and session storage.

The SQL Database server is used for the relational database solution. It can log uploads and downloads, manage file locations and keys and user information can be stored in the future.

The BLOB Storage service will be used for storing the actual files. An unlimited amount of file containers is featured which can each contain an unlimited number of BLOBs.

4.3.5. EMAIL SERVICE

When a user has finished uploading, the addressee of the data has to be notified and instructed on how to retrieve the files. Furthermore the receiver will need a password or key to prove that he or she indeed is the legitimate addressee.

There are two constraint to be taken into account:

1. The receiver does not need a user account

2. The user is not required to install any software other than a browser to use the system.

According to constraint 1, the user does not need a user account. Therefore email and telephone are the only ways to remotely contact a receiver as it can be assumed that every receiver has access to these means of communication. Both ways of communicating allow eavesdropping by third parties. However since there is no way to verify that the legitimate addressee is indeed legitimate, one of the communication channels described will need to be trusted. Nowadays the amount of email sent over an encrypted connection increases every day [28]. Email therefore seems to be the best solution, also because making a phone call would require the sender and receiver to use the service at the same time, which is not feasible.

Using email to notify the addressee will limit the effort the user needs to make to contact the receiver and does not require any extra software to be installed, meaning that constraint 2 is covered.

The reasons above are why the application will use an email service in order to instruct the recipient of DICOM files. The recipient will receive an email with an URL and key which can be used to retrieve files uploaded by a sender. Instead of setting up an SMTP server and handle email, the Mandrill service will be used by means of its API [29]. Emails can be tracked in terms of opens and clicks, which is a great accessory because it allows the monitoring and analysis the email traffic.

As is explained, email imposes a security threat. Encrypted email, protecting the information inside the email from malicious persons, is supported by the Mandrill mail service. The secure delivery of an email to the recipient's email service, for example Gmail, is accounted for by Mandrill. Encrypted email is supported by Gmail and HTTPS is used to transfer an email from the Gmail server to the user's browser. However, there are email services that do not provide this type of security, meaning that the email transfer from server to user might be in plain-text. This means that a malicious person eavesdropping on the transfer can get a hold of the content and download the files.

4.4. FLOW DIAGRAM

The determining of the key features, the research and the design choices had led to a clear understanding of what the application should do. Reasoned were the states the application could go through and the reasoning was used to create the flow diagram given in Figure 4.2.

The user loads the website used to select files and fills in the contact details of the recipient. After the input is validated, the user can press upload, whereafter the files are uploaded one by one in chunks. When all of a file's chunks are uploaded without errors the file is assembled. Before storing, the file is anonymized and encrypted. After all files have been anonymized, encrypted and stored an email is sent to the recipient containing the decryption key and an upload identifier. The key and the identifier may be used by the recipient to download the files as a ZIP-file using the website.

4.5. SYSTEM ARCHITECTURE

A module based structure has been designed where each module has its distinctive functionalities. Maintainability is increased when a module base structure is used as it allows functionalities to be added or removed without requiring changes to another module. A Model View Controller (MVC) architecture was chosen as MVC encourages the separation of concerns.

The system is divided in the three components below.

Services implement the server side functionalities and are used to communicate with external services like Azure storage and Mandrill.

Controllers handle incoming API requests from the client-side of the system and use the services to handle the requests.

Angular is the client-side web application framework used to handle user input and make API requests to the server.

An overview of the system architecture can be found in Figure 4.3.

Each of the components makes use of multiple models that will be described in Section 4.5.1. The different services used by the system are explained in Section 4.5.2. Section 4.5.3 will describe the controllers' tasks and Section 4.5.4 will cover the Angular client-side of the system.

4.5.1. MODELS

There are several models used in this system. These models are used to communicate between the components and contain grouped information on for example an email. Below an overview of the models is given:

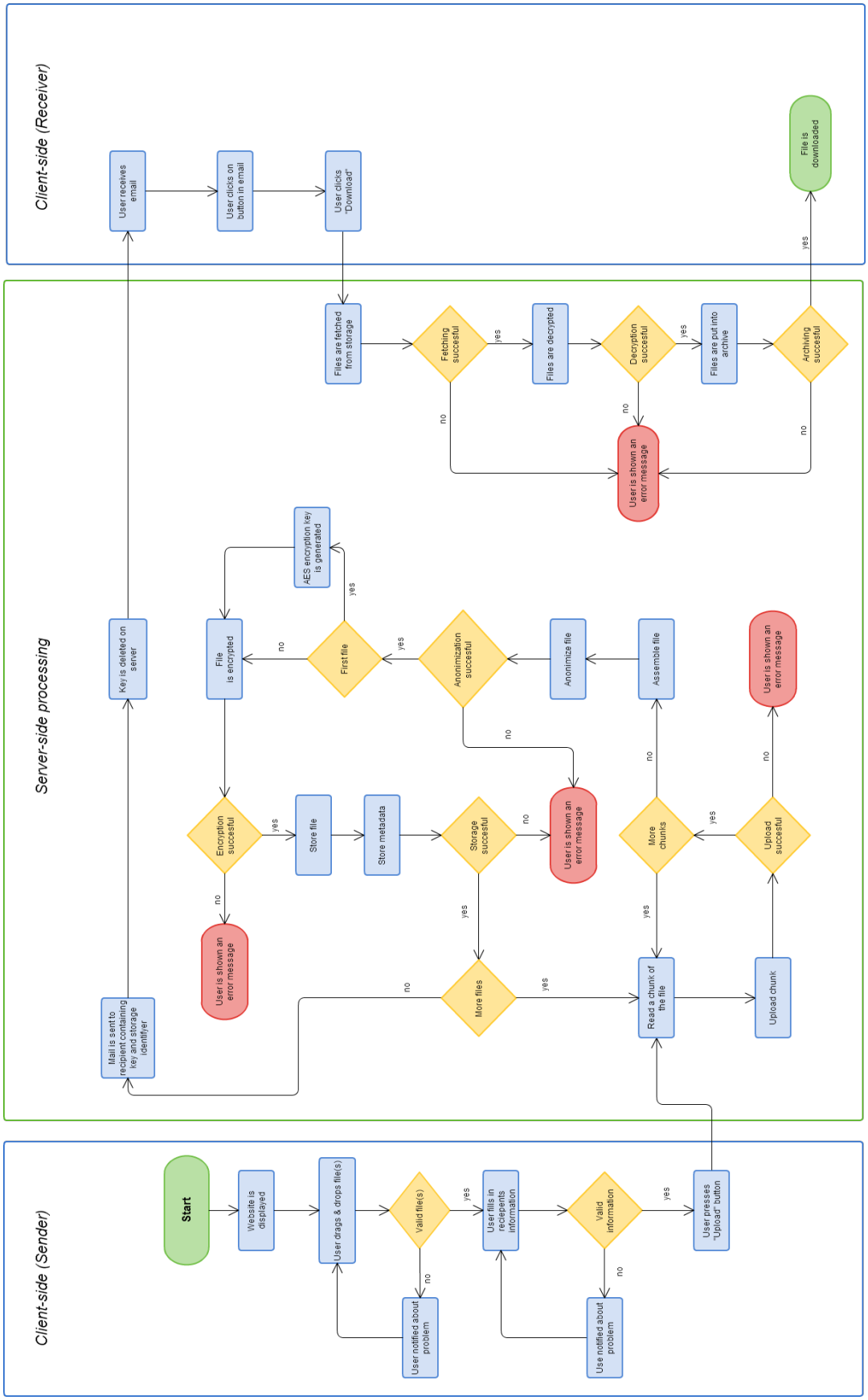

Figure 4.2: A flow chart describing the system's usage.

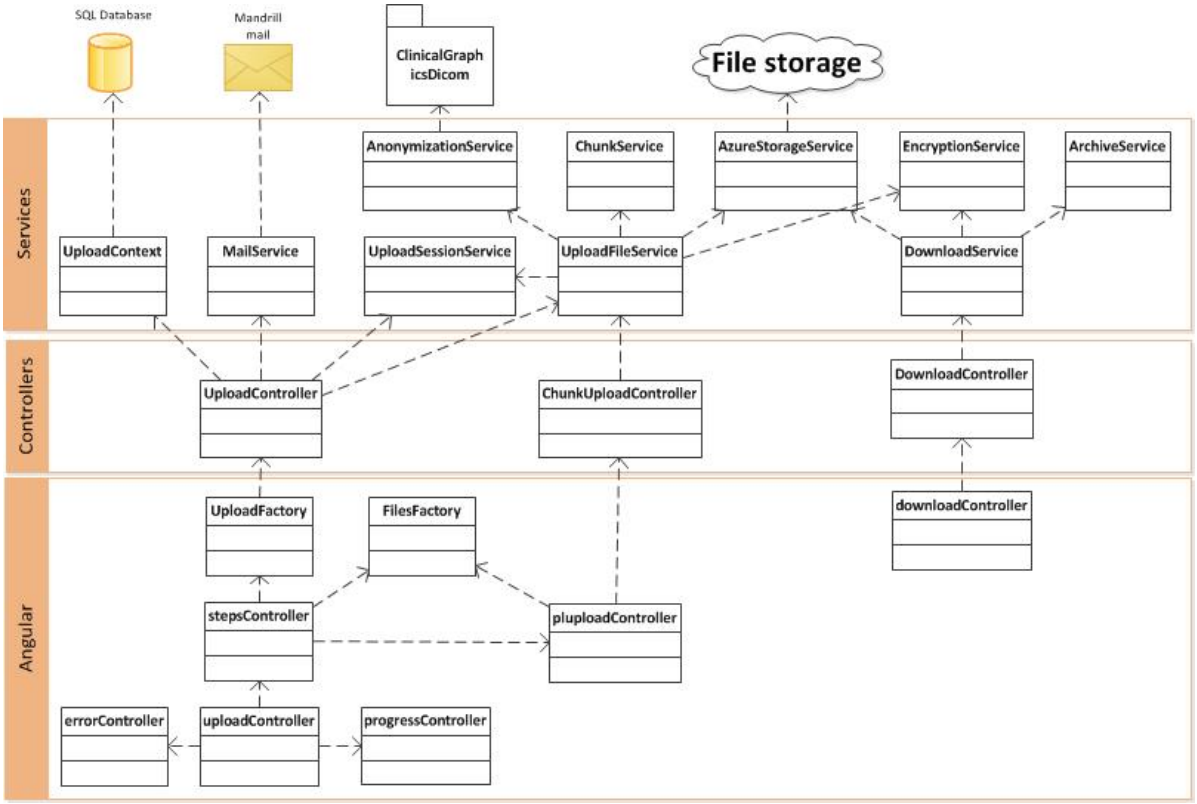

Figure 4.3: System architecture

UploadSession The UploadSession model contains a Globally Unique Identifier (GUID) and an encryption key used for the upload session of a user.

UploadInfoModel Information about an upload is contained in the UploadInfoModel. It contains the GUID, the recipient name and email address, the sender's name and an optional message. The information is sent by the browser to the server when a user starts the transfer.

MailModel The contents of the email to send to a recipient are contained in the MailModel. It is based on the UploadInfoModel, but instead of the GUID it contains a download link. The MailModel and UploadInfoModel are separated as this allows each of the models to change and not influence the other.

ChunkModel When a file is uploaded it is split into chunks. The file name, the consecutive number of the chunk, total amount of chunks and the GUID of the corresponding upload session are contained in the ChunkModel. The model is sent to the server from the browser.

4.5.2. SERVICES

Server-side functionalities like encryptions are implemented by services. Each service implements one functionality as this results in a modular architecture. The services are divided in three categories: storage, process and management services.

STORAGE SERVICES

Each of the following services implements a storage functionality.

AzureStorageService implements the functionality needed to communicate with Azure's Blob Storage and allows files to be stored.

UploadContext is used to store the UploadInfoModel's information when an upload is initiated.

UploadSessionService is used to store the GUID and the generated encryption key of an ongoing upload session. The service is also used to delete the GUID and key when a session expires or when the upload is complete.

PROCESS SERVICES

The files uploaded to the server are modified using the process services before being stored.

AnonymizationService checks whether an input file is a DICOM file and changes the values to be anonymized as listed in Appendix B.

ChunkService is used to store chunks temporarily on the server and assembles the file when all chunks have been uploaded.

EncryptionService is used to generate an encryption key and initialisation vector and encrypt or decrypt an input file.

ArchiveService combines files into a ZIP archive to allow the download of a single file instead of multiple DICOM files.

MailService is used to notify the recipient when the upload, anonymization, encryption and storage of the uploaded files is complete.

MANAGEMENT SERVICES

Functionalities of other services are called by the management services and all operations to be completed are initialized by these services.

UploadFileService handles the steps to be taken to process uploaded chunks in terms of file assembly, anonymization, encryption and storage.

DownloadService handles the steps to be taken to download a ZIP archive with decrypted DICOM files.

4.5.3. CONTROLLERS

The communication between the client and the services is done through the controllers. API calls are made by the client to a controller and the controller is then used to call the services. Three controllers will be used by the application: DownloadController, UploadController and ChunkUploadController.

DownloadController is sent a container name and decryption key by the client and it calls the DownloadService to prepare a ZIP file that is sent back as a response.

UploadController is used to handle an incoming upload request and returns an upload GUID that is used by the client to upload files.

ChunkUploadController is used to handle incoming file chunks and uses the ChunkService to handle the chunks.

4.5.4. ANGULAR CLIENT-SIDE

The client-side of the system will consist of two interfaces: the upload interface and the download interface.

The download interface will make use of the *downloadController* and is used to handle the user input and initiate the download.

The upload interface makes use of the following five controllers: uploadController, stepsController, progressController, pluploadController and errorController.

uploadController is used to manage the content of the web page. When the upload is started the uploadController will make sure to display the progress, but will also show errors and warnings when they occur.

stepsController is utilised to manage the three steps a user will have to go through to start an upload. The controllers applies validation to the input form and an API request is made when the user starts the upload.

progressController is put to action by the uploadController when the uploadController has told the upload is started. The progress of the upload is shown to the user by the controller.

pluploadController is used to update the list of selected files and implements the Plupload polyfill [30] for file uploads.

errorController listens for warnings and errors and shows them to the user.

Furthermore the upload interface makes use of the FilesFactory. A list of the selected files is maintained in the FilesFactory, and stepsController and pluploadController use it to acquire an updated list of the files. API requests like an upload request are made using the UploadFactory.

4.6. USER INTERFACE DESIGN

The application makes use of two separate interfaces. The upload of files is facilitated by one of the interfaces and the retrieval of files by the other. Both interfaces contain no more than is required for a user to upload files. By this is meant that there will be no excessive information, advertisement or buttons. The interfaces are explained below in detail and the framework that is used for styling is talked about.

4.6.1. INTERFACE ELEMENTS

The user should be assisted in completing an upload by using the sender's interface by describing the steps to be taken be the user. Components that must be contained by the interface are given below.

- A contact form that requires the contact information of the receiver.
- A file upload component, which allows drag and drop and a browse option.
- An upload button that starts the anonymization, compression, encryption and file upload.
- Information that guides the user in steps to complete an upload.
- Instant feedback of the upload progress.

The interface for the receiver should not contain more than necessary to complete the download. After a user clicks the link in the notification email the interface is loaded by the browser. A list of components that must be contained by the interface is given below.

- A download button that starts the download without the user having to supply any information; the URL from the email is enough.
- Information that guides the user to complete the download.
- Instant feedback of the download progress.

4.6.2. INTERFACE LAYOUT FRAMEWORK

The application will need a simple interface, in terms that no unnecessary information, images or options are present in the interface. Twitter Bootstrap [31] will be used for styling instead of styling buttons and input fields from scratch. The usage of Bootstrap will result in an appealing interface and will be compatible with all browsers to be supported according to the requirements. Furthermore the framework is included in the application without a problem and has a step-by-step documentation.

5

IMPLEMENTATION

The implementation of DICOMmunicator is explained by listing the frameworks used in Section 5.1. Section 5.2 illustrates the functionality of the application. Finally in Section 5.3 the way the application was tested during the implementation phase is described.

5.1. FRAMEWORKS

The client-side and server-side of the system can be seen as separate systems that communicate through a Web API. Both sides of the system were implemented using a framework encouraging separation of concerns and code maintainability.

5.1.1. SERVER-SIDE FRAMEWORK

The ASP.NET framework [32] using the MVC 4 architectural pattern that ASP.NET comes with was used to implement the application. Separation of concerns is encouraged by the MVC 4 pattern and an extensive documentation on ASP.NET is available. The same functionality is provided by other frameworks, but because the ASP.NET framework is used by the company it is a convenient choice.

5.1.2. CLIENT-SIDE FRAMEWORK

A JavaScript framework was needed to structure the client-side code. The three most popular JavaScript frameworks were looked into: Angular [33], Ember and Backbone. Ember is a framework that works best on a multi page, navigational website [34]. However, the application was going to be a single page application, so Ember was dropped. Angular and Backbone could both have been used, but Angular was chosen because developers using it are encouraged to write testable code. Furthermore Angular is supported by a larger community. [35].

5.2. FUNCTIONALITY

The functionalities of DICOMmunicator are shown throughout this section. First the application's upload web page is shown in Section 5.2.1. In Section 5.2.2 the server-side processing is explained. The application's email notification feature is looked at in Section 5.2.3. Finally in Section 5.2.4 the download web page is shown and describes how the receiver can obtain the files sent.

5.2.1. UPLOAD WEB PAGE

The upload web page is displayed in Figure 5.1. Three columns corresponding to the steps to be taken by the user are displayed, as well as a banner containing the name and key features of the application.

The steps are numbered to guide the user in the progress. A paragraph is included with each step. The essence of the paragraph is written in bold and instantly shows the user what to do. Detailed information is given by the full paragraph.

The selected files are listed in a drag and drop area, which allows files to be dropped from a folder on the user's computer. An optional browse button is supplied that can be used instead of the drag and drop functionality. Files can be deleted by clicking the bin icon. The bin will be recognized by the user as a bin and

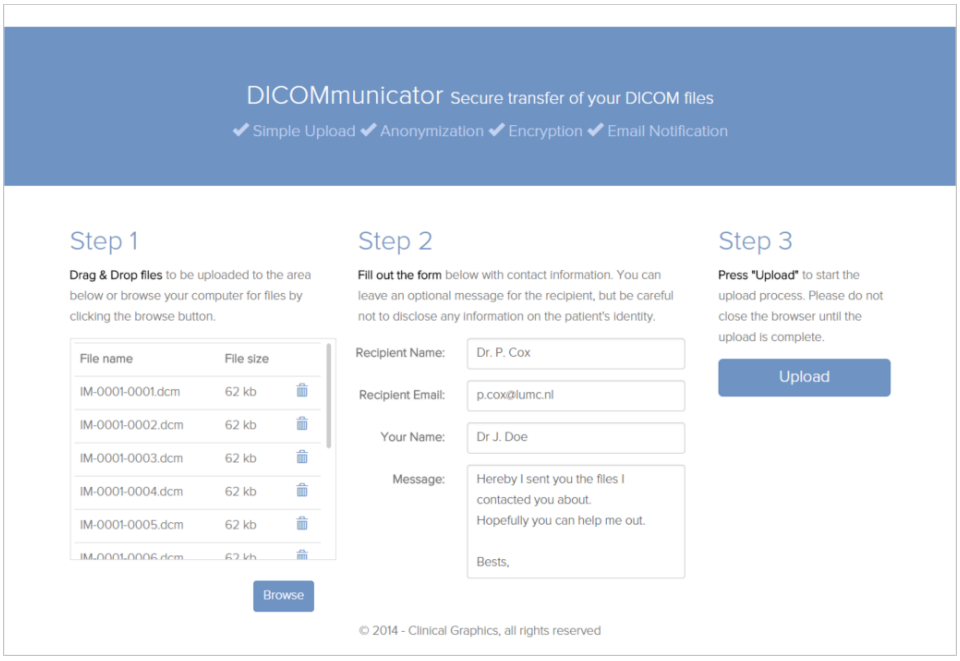

Figure 5.1: The upload page of DICOMmunicator

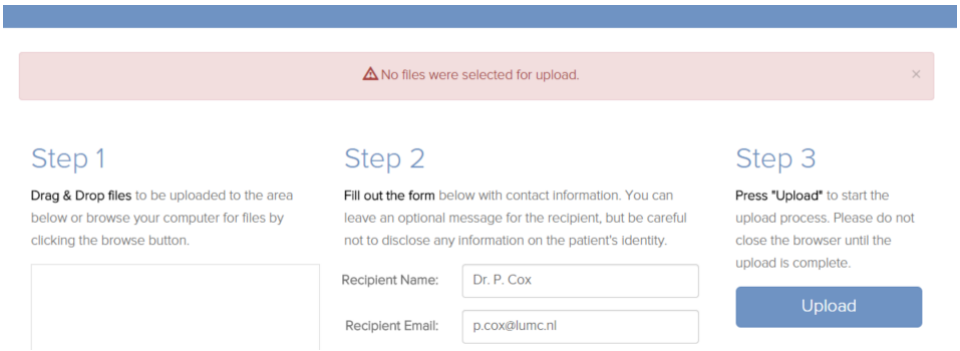

Figure 5.2: A file selection warning given by the application

will be linked to a physical bin, making it obvious a file is deleted by clicking the icon. The mapping of a real world object to a icon was used to reduce the amount of text and explanation needed to clarify the controls. Therefore improving the application's usability.

The user is warned after clicking the input button in case of invalid input. An example is shown in Figure 5.2 in which a warning is given because no files were selected to upload.

After the upload is initialized, by clicking the upload button, the web page switches to the progress view used to inform the user about the upload progress. Figure 5.3 shows an example of what the progress view may look like. The upload progress is illustrated by telling the user about the upload initialization, upload progress, whether the mail is sent and whether the transfer was successful. While the files are being uploaded the user is shown a progress bar and time remaining for the upload to be complete. Furthermore an option to cancel the upload is present.

5.2.2. SERVER PROCESSING

When an upload request is made to the server, a GUID is generated and sent back to the client. The GUID is used by the client to tag file chunks it will be sending. On arrival at the server the chunks are temporarily stored. The original file will be reassembled when the last chunk has arrived on the server. The file is then anonymized. Anonymization was implemented by using the company's existing Silverlight anonymization project. The project was converted to an ASP.NET project and the algorithm performing the anonymization was changed to handle numeric, date and time values without corrupting the file. Furthermore the list of tags

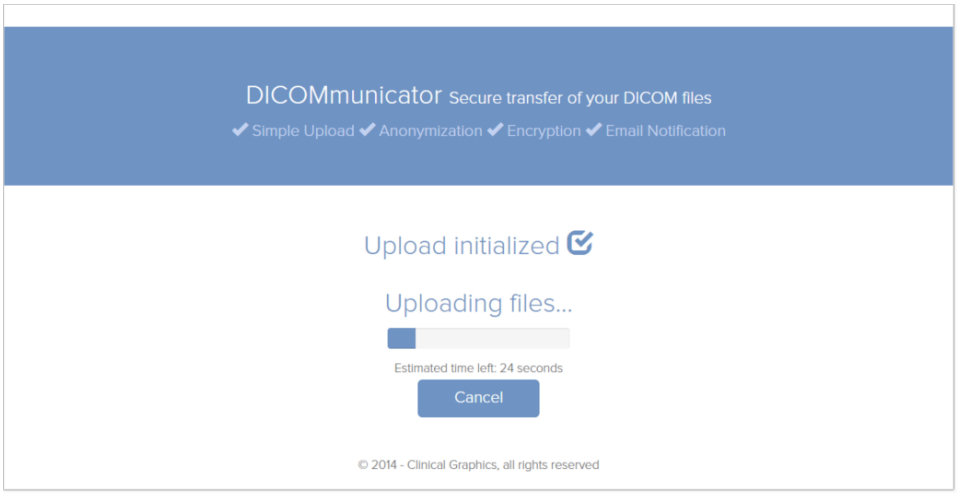

Figure 5.3: The progress view of an upload

to be anonymized was implemented to be recognized by the algorithm. The values corresponding to the tags are altered by replacing the value's characters with replacement characters; 0's and 1's. After anonymization the file is encrypted using the AES encryption specification, whereafter the file is stored in the Azure BLOB Storage. When a file is stored the next file can be uploaded. After all files have been uploaded the server is instructed by the client to send an email to the receiver.

5.2.3. EMAIL NOTIFICATION

The recipient is informed by email, illustrated by Figure 5.4, that files have been uploaded. The email contains the optional message the sender provided and a button that directs the receiver to the download page.

5.2.4. DOWNLOAD WEB PAGE

A small paragraph of explanation is given on the download web page as well a an download button. The download page is illustrated by Figure 5.5. By using the download button a ZIP archive is assembled by the server and automatically downloaded to the user's computer.

5.3. TESTING

During the implementation phase unit tests were written to make sure the application works as intended. In Section 5.3.1 the approach to writing unit tests is given. Furthermore the integration of the application's features were tested in Section 5.3.2.

5.3.1. UNIT TESTING

To be sure every feature of the application worked as intended unit tests were written for every feature. The client-side of the application, written in JavaScript using Angular, required a different approach of testing than the server-side, which was written in C#.

For the client-side Jasmine [36] was used, a behaviour-driven development framework, which allowed every front-end feature to be tested as a scenario. The Chutzpah test runner [37] for Visual Studio was used to execute the tests for JavaScript together with the tests written for the server-side.

C# unit tests were written for the server-side services and controllers to verify that they behave as intended. Wrappers were implemented for the Azure library as the classes were sealed and could not be mocked for unit testing. NCrunch [38] was used; an automated concurrent testing tool, that runs tests as code is written and gives feedback when a test fails due to editing. Furthermore it shows lines that are not covered. NCrunch came with a coverage tool of which the coverage results are shown in Figure 5.6.

5.3.2. INTEGRATION TESTING

One of the requirements was that the application should work on all popular browsers; Firefox 30+, Chrome 33+, Safari 7+ and Internet Explorer 8+. The required browsers were installed. Microsoft's Modern.ie [39],

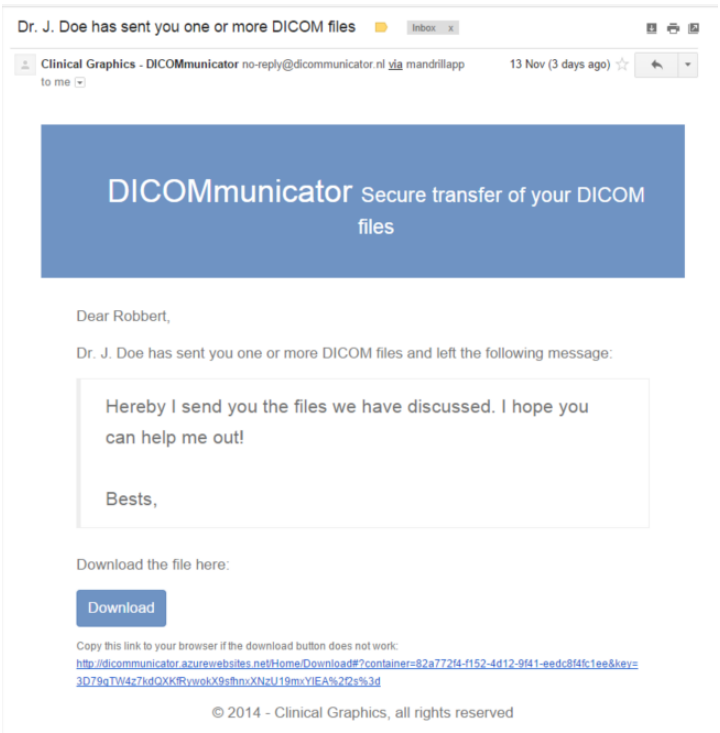

Figure 5.4: An email informing the receiver of files

Figure 5.5: DICOMmunicator's download web page.

Name	Code Coverage %	Compiled Lines	Covered Lines	Uncovered Lines	Code Lines
Entire Solution	89.38%	565	505	60	1,729

Figure 5.6: Coverage of the C# code using NCrunch

which provides Virtual Machines that run a version of Internet Explorer on operating systems like Window XP and Windows Vista, was used to install Internet Explorer 8 and 9. The functionality of the application was tested in each of the browsers by hand. Browser testing tools like Selenium [40] allow the writing of tests that automate browser testing. However, decided was that for the scope of this project manual testing was adequate.

6

Discussion and Recommendations

In this chapter the project is discussed and recommendations are given for further development of the application. In Section 6.1 the process and product are evaluated. Concluded is in Section 6.2 whether the problem described by Chapter 1 is solved by DICOMmunicator. Recommendations for future development of DICOMmunicator are listed in Section 6.3.

6.1. Evaluation

The project is evaluated in terms of process and product. In Section 6.1.1 the project methodology is discussed. The requirements that have not been met by the application are listed in Section 6.1.2. Furthermore a user test was carried out to determine the usability of DICOMmunicator and the results are described in Section 6.1.3. The application's code was assessed to ensure code quality. The quality is discussed in Section 6.1.4 .

6.1.1. Methodology

The project's methodology as given in Chapter 2 is reflected on in the following sections.

Planning

When the original planning made at the beginning of the project is compared to the actual time there are differences to note. The actual timeline of the project is given in Figure 6.1.

The research phase was expected to be finished in the first two weeks of the projects. However, the design phase required research on best practises in order to come up with a solid system. Coming to a best solution in terms of file encryption and key distribution turned out to take more time than expected as well. This resulted in the fact that during the design phase research was conducted on the side.

During the first week of implementation the ASP.NET MVC 4 principles were to get used to, which made the design change in the first week. This is the week in which was decided that anonymization on the server-side was preferred over client-side anonymization. Furthermore, the implementation's planning had been changed in the course of the phase.

Due to personal issues the evaluation phase started half a week later and finished two days later than originally planned.

Making use of an agile software development framework was a good choice. The incremental and iterative aspects suited the project well. The implementation of features like file upload, encryption support and file chunking support all took roughly a week as was expected. This allowed one aspect of the application to be implemented during a week and allowed a demo to be ready every week during the meeting as well.

Agile development

Looking back, the Scrum ways of development like daily stand-ups were not really appropriate for the project as the team consisted of two people. A meeting at the beginning of each week was sufficient to determine what would be completed that week. Furthermore, during the working days the team was updated regularly as the team sat together.

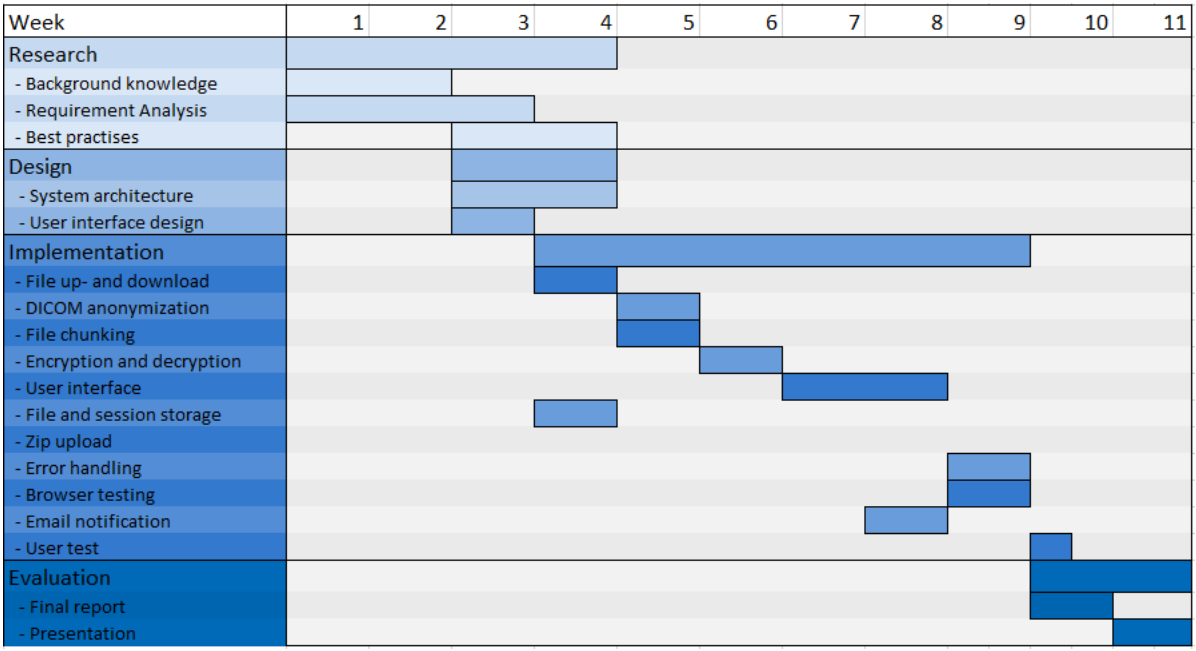

Figure 6.1: The actual timeline of the project

Initially Pivotal Tracker was used to keep track of the product backlog, which had given a clear view of what to do each week. However, refining the backlog and keeping it up-to-date was found to take more time than it yielded. Maintaining a list of to-dos on paper turned out to be more convenient. We are convinced that using an agile management tool like Pivotal Tracker improves the development process in larger teams, but keeping the tool up-to-date cost more time than it saved.

VERSION CONTROL

GitHub was an excellent choice to complete the project with. First of all looking back at all changes in the code was possible. When an unexpected bug was found which had not occurred earlier, it could be found by checking the history of the file the bug originated from. Furthermore, GitHub saved time writing the report as it would automatically merge most of the written work and give a warning when automatic merging could not take place. Each of us working on a different branch gave a clear understanding of the structure and how the application was evolving. Furthermore branching allowed faulty code to be removed without touching the working application's code. One thing that would be done differently during another project is to make use of the issues feature GitHub provides. When a bug in the system is found an issue is created and GitHub automatically manages a list of issues that are not solved yet. Besides the bugs, the feature is also useful for questions or saying a piece of code is invalid. Furthermore, specific issues can be assigned to the responsible developer.

ROLE DIVISION

Having each team member responsible for every aspect of the project was a good choice. We believe that we would have been less critical if for example one of us was responsible for the code testing. During the course of the project it was unwritten that at one point one of us was lead developer of a feature. One team member for example knew most about the encryption and decryption and worked on that more than another member, but that did not mean that the other member was responsible for it. We however feel that in a larger development team having one person responsible for aspects of a project, code quality for example, is a good choice. This is because a member of a larger team feels less responsible for certain aspects as the person subconsciously assumes someone else will take care of the aspect. This was not the case in this project as the team consisted of two people.

6.1.2. PRODUCT REQUIREMENTS

The requirements that have not been or have been partially fulfilled are discussed in this section. The MoSCoW methodology is used to order the requirements starting with the most important requirements.

PARTIALLY FULFILLED REQUIREMENTS

The following requirements are partially fulfilled.

- *The user must be able to drag and drop a file that the user wants to send onto the application.*

  The drag and drop functionality could not be implemented in browsers that do not support HTML5, like Internet Explorer 8 and 9. The browse button can be used in case of an incompatible browser. No check has been implemented to see whether a browser supports the drag and drop functionality or not. Browsers that do support HTML5 do feature the dragging and dropping of files.

- *Should log the uploads and downloads of files.*

  The upload sessions are logged, but the download sessions are not. This functionality was lower priority due to time constraint, but can be implemented in the future by storing download information when a download is initiated.

UNFULFILLED REQUIREMENTS

The following *could* haves have not been implemented.

- *The sender could send ZIP archives with one or more DICOM files.*

  Input support for ZIP archives was not implemented due to time constraints. Only DICOM files are supported by the system.

- *The interface could give a preview of the images that are in the DICOM file.*

  Due to time constraint a preview feature was not implemented.

- *Could compress DICOM files before sending over a network.*

  Compression before transfer was to be used to reduce file size and bring the time for transfer down. The feature was given a low priority as it would require benchmarks to be ran to determine whether compression would save time. Due to time constraints compression was not implemented.

- *Could decompress DICOM files locally in the receiver's browser.*

  Decompression was given a low priority on the list and could only be implemented in Google Chrome, since Chrome is the only browser with a supported FileWriter. Furthermore in Section 4.3.1 was explained that processing files in the user's browsers would not work.

6.1.3. USER TEST

A user test was carried out to determine whether the application is self-descriptive and does not raise any questions. Three test subjects were asked to use the application by completing two assignments. They were provided with a scenario in which they were a doctor that needed to send DICOM files to another doctor. DICOM files were given to the subjects and the first assignment was to upload the files using DICOMmunicator. The second assignment was to download the files after a DICOMmunicator email was sent to then.

FEEDBACK

The feedback from the test subjects included the following. Originally a red 'x' was used as delete icon for files. One test subject however thought the selection had gone wrong, before realising the button could be used to remove files. The icon was replaced by a trash bin icon, which is associated with discarding. Furthermore the progress messages were found to be changing too quickly as the system performs operations like initializing instantly.

The progress information was found clarifying as it showed the subject what the system was doing. Also, the application looks appealing.

No problems occurred during the uploads and the downloads took place without any problems as well.

6.1.4. QUALITY ASSURANCE

To allow the application's code to be maintainable the code was assessed by the Software Improvement Group (SIG). In the duration of the project the code was assessed twice. Once when the 75% mark of the project was passed and a second assessment at the end of the project. The feedback of the first evaluation is given in Appendix A.1 and the section below explains how the feedback was handled.

At the end of the project the code was evaluated again, in which SIG also checked how the first feedback was handled. The results from the second evaluation are given in Appendix A.2.

PROCESSING THE FEEDBACK

The first evaluation showed that the code had a maintainability level that was above average. There were two aspects which led to the score not being the maximum of 5 stars. These aspects were unit size and unit interfacing.

Unit size refers to the fact that there is code that was relatively long compared to other code. The issue was addressed by splitting the Angular code into multiple files and creating a folder structure that allowed grouping of the controllers, directives, filters and services. Instead of having all controllers in one file, which was the case at first they were placed in separate files together in a *controllers* folder. Furthermore one file contained a large part of JavaScript compared to the other code. The file however was used for experimenting on a feature and was not to be assessed. It was deleted as it was no longer required.

Unit interfacing means that a relatively large part of the code makes use of too many parameters compared to other code, which can cause confusion. For example methods were using the following parameters: mediaType, inputStream, name, chunkNumber, chunkTotalAmount and uploadGuid. The code quality was improved by refactoring the parameters into a *ChunkModel* which was passed into the methods instead of all of the previous parameters.

Furthermore, SIG suggested that existing libraries used by the application should be placed in a folder called *ext* or *lib*, to make clear that these files are existing libraries used by the application. This issue was addressed by adding *ext* folder for the CSS code.

Finally, ReSharper, a development productivity extension for Visual Studio was used to clean up the code according to C# coding conventions.

6.2. CONCLUSION

An application was needed to allow the secure transfer of medical image files. This can be achieved by removing sensitive patient information that can be used to identify the patient, and by encrypting the files during transfer. Existing solutions however, are not easy-to-use and therefore services that do not support anonymization and encryption are used by doctors. Easy-to-use in this case means that the usability is comparable to WeTransfer-like [3] services.

DICOMmunicator was developed to solve this problem. A self-describing interface is provided by the single page application which guides the user in three steps to a successful file transfer. It is available to users using one of the major browsers; Firefox 30+, Chrome 33+, Safari 7+ and Internet Explorer 8+. Only having to select the files to upload and filling out the recipient contact information makes the application easy-to-use. The files are uploaded automatically and anonymized on arrival on the server and are stored in encrypted form. The recipient is sent an email with information on how to download the files using the download page.

DICOMmunicator has implemented all the requirements that were necessary to securely send medical image files. Therefore it can be concluded that DICOMmunicator solves the problem defined in Chapter 1.

6.3. RECOMMENDATIONS

A list of recommendations has been formulated that contains features that may be implemented in the future. This list is given below.

Client-side form validation The validation of upload data is currently done by the server. For this reason there is a delay after pressing the upload button. This delay can be prevented by client-side form validation. The server side validation should be kept to prevent malicious usage.

Browser detection Drag and drop is currently not supported by Internet Explorer 8 and the application does not take browser versions into account. However, the upload information saying to drag and drop files is present when the application is loaded in Internet Explorer 8. Browser specific scripts should be implemented to take care of these incompatibilities and in this case change the upload information. For detection of drag and drop support it is not necessary to implement a full browser detection. The Plupload runtime can be used, since drag and drop is currently only supported in html5.

ZIP file upload Currently only DICOM files can be uploaded. Doctors may want to upload a ZIP archive with DICOM files. The implementation requires a modification in the Plupload file filter and a method on the server that checks whether the uploaded file is an ZIP archive. The DICOM files will have to be extracted from the archive and can then be processed using the existing logic. A decision is to be made about non-DICOM files.

Integrated DICOM viewer To view DICOM files a receiver must download the files and open them with a DICOM viewer. If a doctor wants to take a quick look at the files or does not have a DICOM viewer installed an integrated viewer would reduce the actions the user has to take. This feature will improve user experience. Open source JavaScript DICOM viewers are available and can be used to support this feature.

Improved anonymization The characters of a value to anonymize are currently replaced by a replacement character. If the receiver has information on the sender's patients the informations can be used to identify the patient. For example, a patient's name which has a length of 20 characters will be anonymized to a string of 20 characters, possibly allowing a receiver to find out the patient's name. The anonymization algorithm can be improved by also shortening the value.

Other security options Besides using a AES for encryption a developer could store the files in two or more different databases. By taking the bit difference of a file A and a random bit sequence B of the same size A can not be derived without knowing both A and B. By storing A and B in different databases the security of both storage servers has to be breached to risk the unauthorized exposure of files.

Client-side functionality In order to improve transfer security of DICOM files, encryption and anonymization can be implemented on the client-side. The encryption and anonymization will only work in the supported browsers. However, the server will need a way to validate that an incoming file is anonymized and encrypted correctly.

A

CODE EVALUATION FEEDBACK (DUTCH)

A.1. FEEDBACK EVALUATION I

De code van het systeem scoort ruim 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Size en Unit Interfacing.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Wat binnen dit systeem opvalt is dat zowel bij de JavaScript als bij Razor langere units te vinden zijn. De oorzaak binnen de JavaScript code ligt bij het samenvoegen van alle Angular code binnen 1 bestand, terwijl de best practice is om per controller een apart bestand aan te maken. Zeker bij het groeien van de applicatie zal het opsplitsen van de code het onderhoud makkelijker maken. Binnen de Razor valt op dat de file 'plupload.cshtml' met name JavaScript bevat. Het verplaatsen van deze code naar een eigen JavaScript file maakt het duidelijk wat de functie van deze code is. Het is aan te raden kritisch te kijken naar de huidige locaties van de JavaScript code en deze waar mogelijk apart onder te brengen.

Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden. Wat binnen dit systeem opvalt is dat er op meerdere plekken een combinatie van 'fileName' en 'directoryName' in de parameter lijsten voorkomt, hier kan wellicht gebruikt worden gemaakt van een (versimpelde) 'File' representatie. Ook het paar 'chunkNumber' en 'chunkAmount' lijken bij elkaar te horen, wellicht kan ook hier een concept zoals 'Upload' voor worden geïntroduceerd samen met 'uploadGUID'. Om bij toekomstige aanpassingen duidelijker te maken wat er precies meegegeven moet worden aan deze methodes is het aan te raden een specifiek type te introduceren voor deze concepten.

Daarnaast nog een opmerking over de code-organisatie van de CSS. Op dit moment staat de code van de Bootstrap library en de eigen style-informatie naast elkaar in 1 directory. Naar alle waarschijnlijkheid moet de Bootstrap code niet aangepast worden. Om dit duidelijk te maken is het aan te raden de code van libraries en de eigen code goed te scheiden, in dit geval bijvoorbeeld door de Bootstrap library in een 'lib' of 'ext' directory te plaatsen.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase. De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code (zowel voor C# als voor de JavaScript) ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

A.2. FEEDBACK EVALUATION II

In de tweede upload zien we dat de omvang van het systeem is gestegen met 31% en dat daarbij de score voor onderhoudbaarheid nagenoeg gelijk is gebleven.

Bij de Unit Size zien we dat binnen de Razor de JavaScript code is verwijderd en dat de 'uploads' code is uitgesplitst over meerdere files. Deze files zijn wel aan de wat langere kant. Het opsplitsen van de JavaScript

heeft ook voor verbetering gezorgd, het merendeel van de units zijn nu ook kleiner.

Wat betreft de Unit Interfacing zien we een lichte stijging, dit lijkt met name te komen door de introductie van het 'ChunkModel'. De bootstrap library lijkt verplaatst te zijn en het aantal tests is gestegen in zowel JavaScript als C#.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject. Het is goed om te zien dat naast een functionele uitbreiding er ook nog steeds een stijging in het volume van de test-code voor beide technologieën te zien is.

B

ANONYMIZATION VALUES

DICOM tag	Description
(0008,0020)	Study Date
(0008,0021)	Series Date
(0008,0030)	Study Time
(0008,0031)	Series Time
(0008,0050)	Accession Number
(0008,0080)	Institution Name
(0008,0081)	Institution Address
(0008,0090)	Referring Physician's Name
(0008,0096)	Referring Physician Identification Sequence
(0008,1010)	Station Name
(0008,1040)	Institutional Department Name
(0008,1048)	Physician(s) of Record
(0008,1049)	Physician(s) of Record Identification Sequence
(0008,1050)	Performing Physicians' Name
(0008,1052)	Performing Physician Identification Sequence
(0008,1060)	Name of Physician(s) Reading Study
(0008,1062)	Physician(s) Reading Study Identification Sequence
(0010,0010)	Patient's Name
(0010,0020)	Patient ID
(0010,0030)	Patient's Birth Date
(0010,0032)	Patient's Birth Time
(0010,1000)	Other Patient IDs
(0010,1001)	Other Patient Names
(0012,0010)	Clinical Trial Sponsor Name
(0012,0030)	Clinical Trial Site ID
(0012,0031)	Clinical Trial Site Name
(0012,0040)	Clinical Trial Subject ID
(0012,0042)	Clinical Trial Subject Reading ID
(0012,0050)	Clinical Trial Time Point ID
(0012,0060)	Clinical Trial Coordinating Centre Name
(0018,1000)	Device Serial Number
(0018,1011)	Hard copy Creation Device ID
(0018,1801)	Time Source
(0020,000D)	Study Instance UID
(0020,000E)	Series InstanceUID
(0020,0010)	Study ID
(0040,A120)	DateTime
(0040,A121)	Date

DICOM tag	Description
(0040,A121)	Date
(0040,A122)	Time
(0040,A122)	Time
(0040,A123)	Person Name
(0040,A123)	Person Name
(0040,A124)	UID

BIBLIOGRAPHY

[1] *Health insurance portability and accountability act of 1996,* United States Congress Public Law **104**, 191 (1996).

[2] Roni, *Dicom is easy - software programming for medical applications,* http://dicomiseasy.blogspot.nl/2011/10/introduction-to-dicom-chapter-1.html (2011).

[3] R. Hans, R. Visser, and B. Beerend, *Wetransfer - an easy and free to use platform for sending large files,* https://www.wetransfer.com/ (2014).

[4] *Clinical graphics' lightbox,* https://apps.clinicalgraphics.com/ (2014).

[5] *Microsoft silverlight,* http://www.microsoft.com/silverlight/ (2014).

[6] *image32,* https://www.image32.com/ (2014).

[7] *About dicom library,* http://www.dicomlibrary.com/about/ (2014).

[8] *Adobe flash player,* http://get.adobe.com/nl/flashplayer/ (2014).

[9] D. Saltman, C. Turner, and F. Winetown, *popbox,* https://popbox.io/about (2014).

[10] S. Dzik, D. Hill, and J. Schwebel, *An overview of picture archive and communication systems,* (1983).

[11] *Dcmtk - dicom toolkit,* http://www.dcmtk.org/dcmtk.php.en (2014).

[12] *Powertools,* http://www.laurelbridge.com/powertools.html (2014).

[13] P. Krekel, *An online medical image transfer service,* http://bepsys.herokuapp.com/projects/view/67 (2014).

[14] *Homepage of the scrum development framework,* http://www.scrum.org/ (2014).

[15] R. Mee, *Simple, collaborative project management from the experts in agile software development,* http://www.pivotaltracker.com/ (2014).

[16] *Github - build software better, together,* https://github.com/ (2014).

[17] K. Waters, *Prioritization using moscow,* http://www.allaboutagile.com/prioritization-using-moscow/ (2009).

[18] K. Matz, *Donald norman's design principles for usability,* http://architectingusability.com/2012/06/28/donald-normans-design-principles-for-usability/ (2012).

[19] *Security: Cia triad and parkerian hexad,* http://cyberseecure.com/2012/07/basic-information-and-network-security-objectives-cia-triad-and-parkerian-hexad/ (2012).

[20] Osirix, *Dicom sample image sets,* http://www.osirix-viewer.com/datasets/.

[21] C. Hoffman, *Sandboxes explained: How they're already protecting you and how to sandbox any program,* http://www.howtogeek.com/169139/sandboxes-explained-how-theyre-already-protecting-you-and-how-to-sandbox-any-program/ (2013).

[22] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard* (Springer, 2002).

[23] *Hipaa 'protected health information': What does phi include?* http://www.hipaa.com/2009/09/hipaa-protected-health-information-what-does-phi-include/ (2014).

[24] *Glossary of common terms health insurance portability and accountability act of 1996 (hipaa)*, http://healthcare.partners.org/phsirb/hipaaglos.htm (1996).

[25] ACR NEMA, *Digital imaging and communications in medicine (dicom) part 3: Information object definitions*, (2011).

[26] *Java plug-in for browsers*, http://www.oracle.com/technetwork/java/index-jsp-141438.html (2014).

[27] *Azure cloud-based media services*, http://azure.microsoft.com/ (2014).

[28] *Email encryption in transits*, https://www.google.com/transparencyreport/saferemail/ (2014).

[29] *Mandrill api*, https://mandrillapp.com/ (2014).

[30] *Plupload*, http://www.plupload.com/ (2014).

[31] *Bootstrap framework for front-end web development*, http://getbootstrap.com/ (2014).

[32] *Download asp.net | the asp.net site*, http://www.asp.net/downloads (2014).

[33] *Angularjs: Html enhanced for web apps!* https://angularjs.org/ (2014).

[34] L. Orsini, *Angular, ember, and backbone: Which javascript framework is right for you?* http://readwrite.com/2014/02/06/angular-backbone-ember-best-javascript-framework-for-you (2014).

[35] U. Shaked, *Angularjs vs. backbone.js vs. ember.js*, http://www.airpair.com/js/javascript-framework-comparison (2014).

[36] *Jasmine: Behavior-driven javascript*, http://jasmine.github.io/ (2014).

[37] *Chutzpah - a javascript test runner*, https://chutzpah.codeplex.com/ (2014).

[38] R. Mulder, *Ncrunch - concurrent testing tool for visual studio.* http://www.ncrunch.net/ (2014).

[39] *modern.ie, dev tools essential for building the modern web*, https://www.modern.ie/en-us/virtualization-tools#downloads (2014).

[40] *Selenium - browser automation*, http://docs.seleniumhq.org/ (2014).