



OULUN YLIOPISTO
UNIVERSITY of OULU

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Joona Rahko

Web Client for a RESTful Clinical Data Management System

Master's Thesis
Degree Programme in Computer Science and Engineering
May 2015

Rahko J. (2015) Web client for a RESTful clinical data management system. University of Oulu, Department of Computer Science and Engineering. Master's Thesis, 69 p.

ABSTRACT

With the emergence of computers, the fashion in which clinical trials are conducted has been revolutionized. Traditionally, most clinical trials have been run on paper based systems, which is inefficient in the light of today's technology. Computerization of clinical data management has improved clinical trial processes in many ways, such as by reducing the cost of collecting, exchanging, and verifying information. Moreover, readily available data has also greatly improved subject safety, as physicians are faster aware of any adverse events.

This thesis depicts the requirement elicitation, design, implementation, and evaluation of a web client for Genesis, a web-based clinical data management system developed primarily for the LIRA-Study. In the LIRA-Study, the software will be used at various study sites in Finland as well as in Sweden. The usability testing, presented in this thesis, indicated that the engineered application was user-friendly and that its development should be continued. In addition to serving the LIRA-Study, the secondary goal of the developed software is to be easily portable to other studies. Initial plans have already been made to deploy Genesis in two other large-scale studies, one of which is the largest type 1 diabetes study in the world.

In addition to presenting the developed software, both the current state and the history of clinical data management are also discussed. After illustrating the software development process, the results and future prospects of Genesis are pondered.

Keywords: Clinical data management, Web client, Programmable web, REST, Usability

Rahko J. (2015) Web-asiakasohjelma REST-pohjaiseen kliinisen tiedon hallintajärjestelmään. Oulun yliopisto, tietotekniikan osasto. Diplomityö, 69 s.

TIIVISTELMÄ

Kliinisten tutkimuksien toteuttamistapa on muuttunut valtavasti tietokoneiden yleistyessä. Ennen tietokoneiden valta-aikaa kliiniset tutkimukset ovat toimineet paperipohjaisilla järjestelmillä, jotka ovat nykyteknologian valossa olleet tehottomia. Kliinisen tiedonhallinnan tietokoneistuminen on parantanut kliinisen tutkimuksen prosesseja monilla tavoin, kuten pienentämällä informaation keruusta, jakamisesta ja tarkistamisesta aiheutuvia kustannuksia. Nopeasti ja helposti käytettävissä oleva data on lisäksi parantanut tutkimuspotilaiden turvallisuutta, sillä tutkimuslääkärit ovat nopeammin tietoisia lääkkeiden mahdollisista sivuvaikutuksista.

Tämä diplomityö esittelee web-pohjaisen kliinisen tiedon hallintajärjestelmän Genesiksen web-asiakasohjelman vaatimukset, suunnittelun, toteutuksen ja evaluoinnin. Kehitetyn järjestelmän ensisijainen tarkoitus on palvella LIRA-tutkimusta. LIRA-tutkimuksessa ohjelmistoa käytetään monella tutkimuspaikkakunnalla sekä Suomessa että Ruotsissa. Tässä työssä esiteltävän ensimmäisen version käyttäjättestaus osoitti, että kehitetty asiakasohjelma on käyttäjäystävällinen ja sen kehittämistä kannattaa jatkaa. Ohjelmiston toissijainen tavoite on olla helposti siirrettävissä muihin tutkimuksiin. Ohjelmiston käyttöönottoa onkin alustavasti suunniteltu myös kahdessa muussa ison mittakaavan tutkimuksessa. Toinen näistä tutkimuksista on maailman suurin tyypin 1 diabeteksen kehittymistä selvittävä tutkimus.

Tässä diplomityössä keskustellaan ohjelmiston esittelyn lisäksi kliinisen tiedon hallinnoinnin historiasta ja nykytilasta. Ohjelmistokehitysprosessin kuvaamisen jälkeen tässä työssä pohditaan Genesiksen jatkonäkymiä ja onnistumista ohjelmistoprojektina.

Avainsanat: Kliinisen datan hallinnointi, Asiakasohjelma, Ohjelmoitava web, REST, Käytettävyys

TABLE OF CONTENTS

ABSTRACT	
TIIVISTELMÄ	
TABLE OF CONTENTS	
FOREWORD	
ABBREVIATIONS	
1. INTRODUCTION.....	9
2. CLINICAL DATA MANAGEMENT	11
2.1. History of clinical data management.....	11
2.2. Clinical data management today	12
2.3. Data capture methods	14
2.3.1. Paper-based data capture.....	14
2.3.2. Electronic data capture	15
2.4. Clinical data management systems.....	16
2.4.1. Local clinical data management systems	16
2.4.2. Central web-based clinical data management systems.....	17
2.5. Common challenges in developing and deploying a CDMS	18
2.5.1. Fear of change	18
2.5.2. Conflicting user-group requirements.....	19
2.5.3. Conflicting workflows.....	19
2.5.4. Safety, security & privacy	19
2.5.5. Analyzability of the data	20
3. SOFTWARE REQUIREMENTS.....	21
3.1. Gathering requirements	21
3.2. Stakeholder interviews	22
3.3. Use cases	22
3.4. System requirements	25
4. DEVELOPMENT METHOD, TOOLS AND APPROACHES.....	29
4.1. Agile software development.....	29
4.2. Client-side versus server-side rendering	30
4.3. JavaScript framework selection.....	31
4.4. Web service architecture selection	32
4.5. Web service data interchange format selection.....	33
4.6. Version control system selection.....	34
5. WEB CLIENT DESIGN AND IMPLEMENTATION.....	35
5.1. System architecture	35
5.2. User interface navigation flow	36
5.3. User interface design	37
5.4. CRF design.....	38
5.4.1. Label positioning.....	38
5.4.2. Validation	39
5.4.3. Icons	40
5.5. CRF compilation design.....	41
5.6. CRF entry and verification flow.....	42
5.7. Interaction with backend	43
5.8. Information security	43
5.8.1. Securing communication.....	43

5.8.2.	Preventing cross-site scripting	44
5.8.3.	Preventing cross-site request forgeries.....	44
5.8.4.	Password requirements.....	44
5.8.5.	Authentication	45
6.	WEB CLIENT EVALUATION	46
6.1.	Automated testing.....	46
6.2.	Usability testing.....	47
6.2.1.	Usability questionnaire results	47
6.2.2.	User feedback.....	49
6.2.3.	Observations during usability testing.....	49
6.3.	Customer satisfaction	51
7.	DISCUSSION	52
7.1.	Attainment of the primary goal	52
7.2.	Attainment of the secondary goal.....	53
7.3.	Reflecting on development.....	53
7.4.	Genesis compared to existing solutions	54
7.5.	Future prospects of the developed software	55
8.	CONCLUSION	57
9.	REFERENCES.....	58
10.	APPENDICES.....	64

FOREWORD

This Master's Thesis was done in the year 2015 for the data management purposes of the LIRA-Study under the guidance of Professor Tapio Seppänen. While crafting this thesis, my professional knowhow has increased tremendously.

I would like to extend my sincere gratitude to Tapio Seppänen for the acrobatics and the sorcery he pulled off in the sculpturing of this thesis. Additionally, I would like to thank Olli Okkonen for being such a delightful colleague and friend to work with. Further, I would like to thank our DIPP-mother Professor Riitta Veijola for the brilliant personal growth opportunities and the support she has provided me with during the past 5 years that I have worked in her various research projects. I would also like to thank Lloyd 'Shiny Rock' Luck for reading this thesis.

Finally, I would like to thank my parents for the support they have provided throughout my life.

Oulu, 9.5.2015

Joona Rahko

ABBREVIATIONS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BaaS	Backend as a Service
CDASH	Clinical Data Acquisition Standards Harmonization
CDISC	Clinical Data Interchange Standards Consortium
CDM	Clinical Data Management
CDMS	Clinical Data Management System
CORS	Cross-Origin Resource Sharing
CRF	Case Report Form
CSP	Content Security Policy
CSRF	Cross-Site Request Forgery
CSS	Cascading Style Sheets
CVS	Concurrent Versions System
EDC	Electronic Data Capture
GUI	Graphical User Interface
HIT	Healthcare Information Technology
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IT	Information Technology
JSON	JavaScript Object Notation
JWT	JavaScript Object Notation Web Token
MedDRA	Medical Dictionary for Regulatory Activities
MIT License	Massachusetts Institute of Technology License

MITM	Man-in-the-middle attack
MVC	Model-View-Controller
OCR	Optical Character Recognition
REST	Representational State Transfer – architecture
SDLC	Software Development Life Cycle
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SRS	Software Requirements Specification
SSL	Secure Sockets Layer
SUS	System Usability Scale
SVN	Apache Subversion
TLS	Transport Layer Security
UI	User Interface
URI	Unique Resource Identifier
VCS	Version Control System
XML	Extensible Markup Language
XSS	Cross-Site Scripting

1. INTRODUCTION

The purpose of this Master's Thesis is to design and implement a web client for a clinical data management system (CDMS). Overall, this software project has two goals. The primary goal is to serve the data collection and management purposes of the LIRA-Study. The secondary goal is for the software to be portable to other studies.

Among illustrating the software design, this thesis addresses the general difficulties in designing and implementing a CDMS. Additionally, this thesis enlists principles that should be followed when designing and developing aforementioned systems. Furthermore, this thesis introduces some of the modern development tools such as AngularJS that can be leveraged to develop modern rich web applications.

A CDMS is a tool used by clinical trials to manage the acquired research data. An alternative to a CDMS would be a paper based system. Paper based data management often leads to problems such as delayed research data, inability to successfully track and schedule patients, and missing milestones [1].

Among other benefits, a CDMS enables the management of a study to make better resource allocation decisions and to improve inter-study communication [1]. Additionally, a CDMS offers many benefits over its paper predecessor such as input validation, which helps to reduce traditional human transcript errors. Despite the benefits, 30% of active trials still solely rely on the traditional method of capturing and managing data on papers [2].

Before the digital revolution and the existence of healthcare information technology, paper based trial data collection and management were the norm. With the ongoing era of digitalization however, there has been an accelerating trend to utilize computers for research use. While some of the clinical study personnel have embraced the reformation of how clinical trials are conducted, others have not. Most of the resistance toward technology results from the perceived threat of loss of control and from the fear of change [3].

In order for a piece of software to be deployed, the user must perceive it to be useful. It is the developer's responsibility to communicate value of software to the users and help them realize the utility and benefits of information technology. In addition to communicating value, developers need to understand that there is a connection between perceived usefulness and perceived ease of use [4]. The easier the system is to use, the higher the perceived usefulness. The higher the perceived usefulness, the more likely is the utilization of the system [4]. Based on the aforementioned grounds, usability of the system is of vital significance, and therefore a focal point of this thesis.

The developed CDMS will initially be utilized in the LIRA-Study, which investigates the effect of trial drug liraglutide on type 1 diabetes prevention. The trial will begin in 2015 and will take three years to complete. The trial will be conducted in Oulu, Tampere, Turku and Malmö. If the system proves itself to be useful, it will be deployed in other trials as well. To accommodate this, the system will be designed to be easily portable to other trials.

The development of this software project was split into two distinct components. One of these components was the web client, and the other one was the server. With the right technology choices, we were able to establish a separation of concerns, which allowed us to design and develop each component separately. My role in the

software project was to engineer the web client for the application while Olli Okkonen was responsible for the server.

There are various reasons why this software was built in-house. First, off-the-shelf software often does not conform to unique needs of a complicated study. Therefore, tailor-made software can better fit the specific and unique needs of a given study. Second, we had already developed a similar in-house software project for the same customer which proved successful. The success of the earlier project indicated that a customizable in-house solution would be an excellent choice. Moreover, we had already established a clear understanding of the workflows and processes that occurred at the study-site. This knowledge base could be leveraged in the design of a new system. Third, tailor-made software solutions can be designed to be swiftly modifiable and extendable. This is because familiarizing yourself with the source code of an open source application for instance can be a tedious task, never mind extending it with your own features. Fourth, after investigating the costs of deploying ready-made software, it was estimated that the cost of developing a tailor-made solution would likely be lower. Fifth, by developing the software ourselves, we would be in the best position to provide support and training to the users.

The organization of this thesis is as follows; Chapter 2 provides background information about clinical studies, electronic data capture and clinical data management systems. In addition, it will address the general challenges of developing and deploying clinical data management systems. Chapter 3 introduces the initial functional and non-functional requirements that were placed on the system, as well as how those requirements were collected. Chapter 4 depicts the development method and the technology choices that were made for this project. The reasoning for the made choices is also shown. Chapter 5 illustrates the design and implementation of Genesis. In chapter 6, the ways to evaluate software quality and performance are presented. Chapter 7 is assigned to the discussion. Chapter 8 is allocated for concluding and summarizing the thesis.

2. CLINICAL DATA MANAGEMENT

Of late, it has been the ambition of the biopharmaceutical industry to increase its efficiency – both internally and externally [5]. In the industry, clinical data is globally seen as a key asset, as it acts as proof of medicine’s efficacy, safety and utility [5]. Clinical data management systems (CDMS) are the modern solution to manage all this valuable data.

This chapter acts as an introduction to clinical data management and electronic data capture (EDC). The chapter starts off by illustrating the history of clinical data management and then moves onto talking about the current state of clinical data management. After the general overview, this chapter compares local and web-based CDMS systems. Finally, some of the common challenges to developing a CDMS are discussed.

2.1. History of clinical data management

Roughly 50 years ago, the medical industry began its quest of transforming paper based clinical data collection into an electronic form. The motivation for this transformation was the uncontrollable rate at which the paper based systems were growing [6]. Information technology was seen as the silver bullet to the problem as computers were excellent in performing the required functions of sending information from place A to place B, storing data, and printing the data when it was required [6]. In the 1960’s, most medical professionals already utilized paper forms for storing data [6]. Moreover, the sufficient knowledge base for conducting clinical trials had been established [6]. Harnessing all this know-how for the use of a computerized solution was seen as “a simple matter of programming” [6]. Later, this initial assumption was regarded as a joke [6].

The use of healthcare information technology (HIT) has been influenced by various factors. First, as technology has evolved, computers have transformed from extensive, highly expensive computing machines into highly interactive and relatively commonplace everyday tools. Second, the HIT industry is relatively new, and it has gone through a phase of trial and error, as developers have had to first learn how to resolve the challenges, and then, how to market their solutions [6]. Third, during the past 40 years, the medical industry’s demand for clinical data has grown exponentially. This demand has been met with a tremendous increase in the supply of clinical data. The increase in both supply and demand has necessitated computers to take an active role in information management. The need for computers was not only created by the quantity of the data, but also by the high-cost. In the 1960’s, the estimated cost of managing clinical data varied between 25 and 39% of the total cost of the US healthcare system [7].

While HIT held the promise of cutting down costs and enhancing health-care, its benefits were overestimated simultaneously as its costs were underestimated [6]. This was because system developers failed to accurately gauge the complexity of healthcare information management systems [6]. Most of the systems that made it through the development phase failed. One of the identified reasons was lack of user-centered design. Failed HIT solutions left an everlasting scar of skepticism and cynicism on health-care personnel, which still affect the health industry today [6]. The reason why the ambitious project of computerizing healthcare was not

completely put to halt were a couple of the successfully developed systems that shed some hope for the future of HIT. One of such projects occurred at El-Camino hospital where the developed system was actively maintained and updated in a user-centered manner [6]. Without some success stories, such as the one that unfolded in El-Camino, it is unlikely that HIT would be so highly intertwined with medicine.

2.2. Clinical data management today

With the emergence of computers and the ongoing era of digitalization, there has been a gradual increase in the use of electronic data capture and managing technology. One such example would be the use of a CDMS; a technological tool, utilized by clinical studies, for the purpose of clinical data management (CDM). CDM is a vital component of clinical trials, as its purpose is to guarantee that all data has been captured accurately on case report forms (CRF). A CDMS helps in verifying the reliability and quality of the stored data, as it is possible to track any changes done to the CRF's. These changes are tracked using an audit log. In addition to tracking changes, CDMS's often come with the functionality to validate user input, which helps in reducing human errors.

The use of CDM technology has had a prominent impact on how clinical trials are being conducted today. Before, when all clinical trials were paper based, a lot of extra effort had to be put into archiving all of the collected data as well as into verifying it. For example, it used to take roughly 5-8 days to get a query resolved in a traditional paper based study [8]. Today, the same task takes roughly 15 minutes [8]. In the paper-based system, the cost of a single query revolved around US\$80-120, which goes to show how much the paper based system wasted both financial and time resources [8]. Other benefits of EDC are reduced human errors such as unclear handwriting, interpretation errors, and lost or damaged CRF's. Moreover, EDC implementations have lowered the paper consumption of studies. In addition, as clinical trial personnel are no longer required to manage endless piles of paper, huge time savings can also be achieved [8]. Despite all of the aforementioned benefits, 30% of all active clinical trials still solely operate on a paper-based system [5].

Another massive change that has occurred in the field of clinical data management has been the emerging utilization of clinical data management standards such as the ones set by Clinical Data Interchange Standards Consortium (CDISC). CDISC's aim is to establish "global, platform-independent data standards that enable information system interoperability to improve medical research and related areas of healthcare" [9]. Before the rise of clinical data standards, there were a lot of discrepancies revolving around data collection and analysis. Today, most of these complexities can be at least minimized with the use of standards. Another benefit of standards relates to information sharing, as harmonizing data between two different trials is no longer as difficult. In addition, CDISC standards reduce the amount effort that has to be put into the trial design phase, as CDISC offers an existing set of domains and variables that can be used in any study [9]. For this project, we will pre-build the CRFs according to the CDISC standards, so that future studies already have a base they can modify according to their own specific needs.

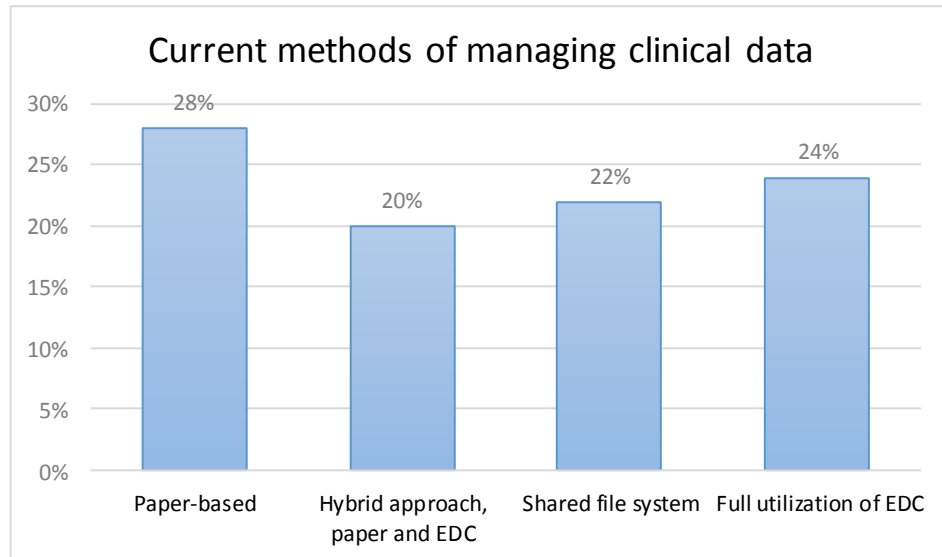


Figure 1. Utilization rates of different ways of managing clinical data in 2013 [10].

Figure 1 illustrates the current utilization rates of CDM technology. These rates indicate that while technology has been deployed in some clinical trials, we are still in a transition phase, where paper is still actively being used. This means that clinical data management has a lot of untapped potential that is yet to be realized.

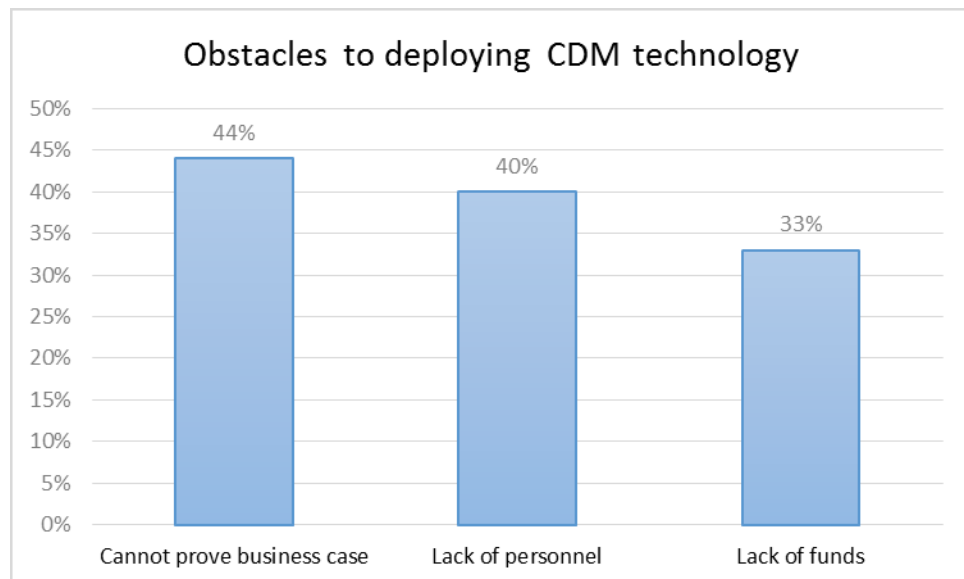


Figure 2. Obstacles reported by clinical trials to deploying data management technology in 2013 [10].

Figure 2 indicates that the biggest obstacle to deploying CDM technology is that IT companies have failed to prove that the clinical trials need their software. This indicates that CDM technology has not been marketed properly. Further, clinical trials identified the lack of personnel and funds as other key determinants of not being able to transition into using CDM technology.

2.3. Data capture methods

Clinical trials have undergone major reform in the way they practice data acquisition. While the traditional method of writing entries on paper CRF's is still being utilized today, electronic data capture methods have become more prevalent in clinical trials. In order to provide a user friendly way to electronically capture data, various methods have been developed. Each of these data-collection regimens has its unique pros and cons.

2.3.1. *Paper-based data capture*

The traditional method of writing entries on paper CRFs is still actively being used. There are various reasons for this – both acceptable and unacceptable ones.

Paper based data collection has many deficiencies. First, a longer study will accumulate a lot of paper CRFs, which require a safe physical storage unit, to which no external entities have access to. Second, in paper based systems, only one copy of the entered CRF usually exists. Third, paper entries are usually deciphered and entered into statistical analysis tools such as Excel or SPSS much later. This delayed process results in ambiguity as the data entry clerk or the statistician has to resort to making judgement calls due to unclear handwriting or illogical entries. Fourth, paper based data collection does not have any kind of immediate validation. This means that the one collecting the data can mistakenly enter insensible data, which eventually makes its way into the actual dataset. Fifth, with paper based data collection, the policy of strict structural data entry cannot be attained. This is because that the data collector can, for example, enter the data on the backside of the paper, leaving it to the data entry clerk to notice and interpret any anomalies.

Despite all of the downsides of paper based data collection, there are some benefits and good reasons to utilize them. Each clinical trial varies in its duration and scope. When a clinical trial is not expected to accumulate a lot of data, and needs to be initiated in a quick timeframe, a paper based solution can be an acceptable way to conduct the trial. Further, if the investment into a CDMS for instance would cause undue harm to the budget of the trial, paper based data collection can be considered to be a viable option. However, it is important to note that even if a paper based solution feels more agile or cheaper, utilizing it can result in unpredictable data digitalization costs and unreliable data.

Some data collectors also claim that computer utilization renders the subject interview sessions impersonal, thus resulting in a bad visit experience for the subject. This is a significant issue to consider, as subjects should be at the center of each clinical trial. Without the subjects, accumulating clinical data would be impossible. Some studies have supported the aforementioned claim that HIT deteriorates the quality of the interview [11]. However, there have also been a lot of studies that indicate that the physician-subject interaction could in fact be enhanced with the use of HIT [12]. In the end, it is a matter of how well HIT is utilized by the physician in the interview process.

2.3.2. *Electronic data capture*

Electronic records are computerized versions of the systematically collected data. In clinical studies, multiple different methods have been developed to digitize data.

The traditional way of entering data is using a keyboard. When the data is entered directly into a CDMS, it can be verified through various means. One way is to use independent double data entry verification, in which two people enter data and a third party resolves any discrepancies between the entries [13]. Another way is to use blind double data entry verification, where two people enter data, unaware of what the other person entered, and they cross-validate each other's entries, correcting any mistakes [13]. A third way is to use double data entry with interactive verification, which is otherwise the same as blind double data entry verification, but in this the other is aware of the others' entries [13]. A fourth option is to use single data entry, where the entered data is later reviewed against the source data [13]. Each of these methods results in a different level of accuracy, as well as costs, which is why it is up to the prime investigator to gauge what is required.

Another way to enter data is to use a voice recognition system. Voice recognition technology has been available for longer than a decade, but the utilization of this technology in clinical settings has been minuscule [14]. Studies comparing voice recognition to human transcriptions have been conducted, and the results of these studies have shown promise for the use of voice recognition technology [14]. In fact the accuracy of voice recognition technology has in some studies been rated as high as 99% [14]. Despite the accuracy, the overhead costs of setting up a voice recognition system are substantial, often acting as a deterrent to taking advantage of voice recognition technology [14].

The use of optical character recognition (OCR) has also picked up amidst clinical studies. OCR systems scan paper entries, and decipher them into a computerized format, rendering them immediately available for analysis. In a perfect world, OCR would provide a seamless solution for clinical studies to capture data, as clinical personnel could hold onto their traditional workflow of using pen and paper, not having to worry about the digitalization of data. However, studies have shown that the current OCR technology does not meet the level of accuracy required by clinical studies, and is therefore, at least for the time being, out of the picture [15].

With the emerging use of tablets, clinical studies have also begun to use handwriting recognition systems in their endeavor to computerize their data collection process. Handwriting recognition mirrors the traditional workflows of physicians. Studies have shown that the use of tablets and digital pens as data collection methods were nearly as fast as using the conventional pen and paper [16]. Additionally, the same studies indicate that tablets and digital pens provide a higher user satisfaction rating than the traditional pen and paper method [16].

To summarize, various means of EDC are already being utilized in clinical studies. However, EDC is often coupled with the traditional pen and paper based practices. This perhaps indicates that we are still living in a transition phase, where old practices are being replaced with new ones. It remains to be seen, whether clinical trials will ever completely discard their old habits. It is clear however that EDC offers a way for studies to considerably cut down their data collection and management costs. Due to the benefits, EDC will be deployed in the LIRA-Study. The LIRA-Study is initially scheduled for 3 years, and due to the overhead costs, we decided to go for a basic keyboard solution.

2.4. Clinical data management systems

Another issue that all of the clinical trials have to grapple with is choosing the right CDMS. Selecting the correct CDMS is affected by the availability of financial resources, as well as by the size, scope, and duration of the clinical trial. One option is to store and enter the data locally, which is known as a local CDMS [13]. Another option is to deploy a local centralized CDMS [13]. A third alternative is to use a centralized web-based CDMS [13]. The unique strengths and weaknesses of each one of these options have to be weighed when deciding which one to deploy.

2.4.1. Local clinical data management systems

In local CDMSs, all of the data is both collected and computerized on-site. This means that physical copies of the data are never mailed to a central location, which helps to save resources. Additionally, because there is no physical transmission of data, the paper copies do not get lost as easily. Moreover, using a local CDMS enables a swift resolution of missing and erroneous data. Another benefit of using a local CDMS is information security. Due to the data being stored locally, local CDMS's are not nearly as vulnerable to being accessed by unauthorized third parties. Local CDMS's are also usually managed by the on-site staff, which enables the study center to tailor the system to suit their own site-specific needs.

Despite all of the benefits, there are downsides to using a local CDMS as well. In most cases, the ones collecting the data are physicians and nurses, who have not been trained in keying in data. Data entry creates extra work for the staff, which is why a need to hire additional trial personnel is often raised. Hiring more workers takes away from the already scarce resources, which is why it is often avoided. Moreover, new version releases of local CDMS's have to be manually installed at each location, inflating the maintenance costs. This is true especially when the released version contains unnoticed bugs, and the install process has to be repeated. Moreover, in local CDMSs, the data sharing between sites often happens through email, which has its own security holes, and creates some delay in information sharing.

One example of a local CDMS is DIPP-Portal, a system developed for the Diabetes Prediction and Prevention – study. The goal of DIPP-Portal was to computerize a paper-based data collection process, and to save money by eliminating the need for data entry clerks. DIPP-Portal was an in-house-solution, done in collaboration with the local trial staff. DIPP-study had been ongoing for over 20 years, which is why the deployment of DIPP-Portal initially faced resistance. Users' fear of change and loss of power are common challenges that software developers have to face. In DIPP-Study, this resistance was notably reduced by involving the primary users in both the design and development phases of the project.

Centralized local CDMSs work in a similar manner. However, the key difference is that in centralized local systems the data is stored and keyed in a single location, usually by professional data entry clerks. This means that before the data is entered, it needs to be physically delivered to the central location. This results in high transportation costs and increases the risk of losing CRFs. However, as data is entered by professional data entry clerks, the entries themselves are done faster and with fewer mistakes. These benefits might help to offset the transportation costs.

2.4.2. Central web-based clinical data management systems

Many larger trials require collaboration and information sharing between various study centers. This has always been true for clinical trials, but with the globalization of nearly everything, the role of information sharing has been set on a pedestal. Exchanging information was previously done through physical media. This required a lot of financial resources, as sending physical copies of data was expensive. Moreover, this traditional information sharing process severely slowed down clinical trials, as they had to wait for information to be delivered. Central web-based CDMSs were the key to unlocking the door to a new way of conducting clinical trials and managing clinical data.

Central web-based CDMSs offer remarkable benefits. First, information sharing becomes instantaneous, as the data is available to all study centers as soon as it is entered. Further, tremendous financial savings can also be achieved, as data exchange no longer requires the delivery of physical media. Further, immediate availability of data also improves the safety of the subjects, as physicians are faster aware of any adverse events. Moreover, maintaining a web-based system is easy, as new releases only have to be installed to the web-server, from which it is directly available to the users. Another benefit of web-based solutions is that it is possible for study coordinators to ensure that all of the data is being stored in a uniform manner. Uniformity of data means that there are no differences in, for example, variable naming or entry conventions between two different centers storing the same information. Late detection of non-uniform data collection often results in unnecessary problems and expenses.

While helping in cutting down costs, web-based systems suffer from certain deficiencies. Firstly, as the data is stored on the Internet, information security becomes a major concern. There are many aspects to information security, some of which are completely independent of the software. However, the first gatekeepers of confidential information are the precaution methods, which software developers place in their software to prevent unauthorized third parties from accessing the information. These include, but are not limited to, having strict minimum password requirements, using a firewall protected database, and sanitizing inputs to prevent SQL injections and cross site scripting. While software developers can take many precautions to prevent information leaks, the truth is that most threats to information security come from the end-users themselves [17]. Often unintentionally, end-users compromise information security by accessing the system from a computer that has been hijacked by malware. These end-user-related threats are addressed in the LIRA-Study by providing information security training to the users.

ClinCaputre, developed by Clinovo, is one example of a centralized web-based CDMS [18]. Clinovo offers various clinical data related services, such as clinical data management, biostatistics, clinical and statistical programming, and medical writing [18]. Clinovo also offers a standard version of clinCapture with limited features for free [18]. With clinCapture, it is possible for end-users such as prime investigators to design their own electronic CRF's [18]. This reduces the need for a clinical trial to hire an IT professional to create the forms. Unfortunately, free versions of the available clinical data management systems were not enough to fulfill the specific demands of the LIRA-Study, which is why they were not utilized. In addition, the cost of purchasing off-the-shelf software was estimated to have been too high.

2.5. Common challenges in developing and deploying a CDMS

CDMS holds the promise of better clinical trial management. As aforementioned, CDMS offers many benefits, including financial and time savings. However, these cannot be realized if the CDMS is not being utilized [19]. Like with technology in general, in developing a CDMS, there are many hurdles that a developer needs to jump over. Some of these obstacles are discussed below.

2.5.1. Fear of change

CDMS is expected to improve clinical trial management and patient safety. However, the utilization of CDM technology often faces resistance from the same professionals, who ought to benefit most from its use [3]. Studies have shown that the resistance for technology is often caused by the general resistance to change [3]. This is especially true, when the users perceive that the technology threatens the status quo, or would potentially reduce their own power and control over the organizational resources [3]. Fear of change is one of the most prominent detriments to information technology usage, which is why developers should have a basic understanding of managing change in an organizational setting.



Figure 3. Kurt Lewin's change model.

Kurt Lewin, a psychologist, argued in his work that change is a process, rather than an event [20]. Lewin's change model helps us to comprehend the three different steps of this process. The first step, which is the unfreezing phase, is focused on preparing the users for change [20]. In the unfreezing phase, developers should communicate to the users, why the software is necessary, and help the users move outside of their comfort zones. The better the developers are able to convince the users that the change is necessary, the more motivated the users are to make the change [20]. One way to brace the users is to demo the software with them throughout the development. This helps the users to familiarize themselves with the software.

The second step in Lewin's model is change, which is focused on actually making the alterations. This step is usually the hardest, as people are unsure of the new methods of conducting affairs and organizing their workflows [20]. In this phase, the main focus of software developers should be to encourage users and provide support through training sessions and one-on-one coaching.

Once the transition phase has been completed, the next step in the model is the refreezing phase [20]. In the refreezing, changes become the new norm [20]. In this step, the focus should be on sustaining the results of the transformation [20]. In the modern world, change is often rapid and continuous, which is why refreezing is probably not the best term to describe the third step. Rather, we should think of the last step as something more flexible, so that the organization can go through the 'unfreezing' phase faster in the future [20].

2.5.2. Conflicting user-group requirements

When there are many user-groups using a given piece of software, the requirements of these user-groups might conflict. In CDMSs, the requirements of the users entering the data, and the users analyzing it, are at a dissonance. The users entering the data often want the CRFs to be very lightweight. To explain, they want to enter the data in an unrestricted manner, and as little of it as possible. This need contradicts the needs of the prime investigators, who wish to collect as much data possible. However, if we give precedence to the quality of the data, the user experience might be heavily influenced. Thus, the software might become cumbersome to use for the users entering the data. Moreover, in the LIRA-Study, all of the subjects are volunteers, which is why the visits cannot be made overly burdensome and lengthy. If the trial visits take up too much of the subjects' time, the withdrawal rate of subjects will increase, thus reducing the trial sample size. Reduced sample size can undermine a trial, which is why finding an appropriate level of collected data is vital.

Software developers have to balance out and prioritize all of the conflicting user-group requirements. This can be done by discussing with all of the user-groups, and communicating the needs of one group to another. In addition, software developers should try to find ways to automate as much of the data entry as possible. Simply put, the focus should be on not having the users enter data that could be collected programmatically.

2.5.3. Conflicting workflows

Another issue that software developers need to address are workflows. Ideally, a given CDMS should be designed to support the existing workflows of the trial staff, rather than replace them with new ones. Each clinical trial is different, which is why designing an all-purpose CDMS is difficult. The benefit of in-house solutions is that all of the user-specific needs can better be taken into consideration. Additionally, if the users have no experience in EDC, it might be necessary to train the users on how to utilize the system to support their tasks. According to studies, if the CDMS conflicts with the existing workflows of the trial management by too much, it can lead to the abandonment of the system [21]. This is why consideration of the existing workflows is paramount to a successful implementation and deployment of a CDMS [21].

In order to address the existing workflows, the developers should do on-site observing and analysis of the existing workflows. It is always easier to change software, than people, which is why the CDMS should be designed to suit the needs of a given trial, instead of trying to get the trial staff to suit the framework set by the CDMS.

2.5.4. Safety, security & privacy

Studies have shown that information security is one of the factors that affect the general acceptance of information technology [22]. With CDMS, none of the data should be accessible by an unauthorized third party. Compromising the information security of subject related data causes financial, psychological, and social harm to the

subject [23]. This is why information security should be a focal point in designing CDMS systems.

According to statistics, information security is compromised more often in systems, where the subject data can be accessed remotely or in systems where the data is physically transported to a centralized data storage [23]. However, by utilizing modern information safety precautions and methods, a higher level of information security can be achieved, than what could be with the traditional paper-based method [23]. In addition, it is important to train the users on information security as most threats to information security are posed by the users [17].

2.5.5. Analyzability of the data

In clinical trials, there are various factors that result in contingencies of the data. First, a big portion of the entered data, such as illnesses or medication, is based on the information elicited from the subjects. Subjects' self-diagnosed illnesses and medication cannot be medically verified, which results in inaccuracy of the data. Second, data is keyed in by a trial staff member, who uses his or her own discretion on how to enter the subject's story into the available input fields. Entry formats often vary from one staff member to another, which is why statistical analysis of the data becomes cumbersome.

These problems can be alleviated with the use of a standardized international medical terminology. These standards can be used both for regulatory communication and evaluation of data pertaining to medicinal products for human use [24]. One such standard is the Medical Dictionary for Regulatory Activities (or MedDRA), which provides a library of standardized medical terminology. By using the standardized terminology, the data does not need to be converted from one terminology to another, which often results in distortion of data [24]. Moreover, with the use of single terminology, data can easily be statistically analyzed later [24]. On the other hand, using the library requires an extra effort from the trial staff, as they are required to search for the correct terms from the library. In the LIRA-Study, the decision whether to use MedDRA is yet to be made. However, we will integrate the support for the MedDRA library as part of the system so that a given trial can deploy it at will.

In addition to utilizing MedDRA, the analyzability of the data can further be improved by deploying a data entry standard such as the Clinical Data Acquisition Standards Harmonization (CDASH) developed by CDISC [9]. CDASH describes the basic recommended data collection fields and formats for 18 different domains [9]. These domains include aspects such as demographics, adverse events, and other common domains that are common to most therapeutic areas and phases of clinical research [9]. CDASH also includes implementation guidelines and best practice recommendations for any CDMS project [9].

3. SOFTWARE REQUIREMENTS

Chapter 3 starts by talking about requirement elicitation on a general level. Thereafter, this chapter illustrates the means of how the requirements were collected in this software project. Lastly, this chapter depicts both the functional and the non-functional requirements that were placed on Genesis.

3.1. Gathering requirements

The first step in the software development life cycle (SDLC) is to gather the business requirements for the software. In this phase, all of the stakeholders will come together and attempt to find answers to questions such as:

- Who are the users of the system?
- Can the users be separated into specific groups?
- What kind of functionality should the software have?
- Where will the users use the system?
- When should the system realistically be available?
- What is the estimated lifespan of the system?

The requirement elicitation phase necessitates all parties to participate, because nowhere else in the SDLC do the interests of all of the stakeholders intersect to such an extent. An enormous portion of problems in the software industry originate from the shortcomings in the ways that people discover, document, discuss and modify the system requirements [25]. Most frequently problems arise from informal information gathering, implied functionality, miscommunicated assumptions, and poorly specified requirements [25]. The requirement elicitation phase should be of vital significance for software developers as statistics show that roughly 50% of all errors found in testing can be attributed to defects in requirements [26]. In addition, the cost of changing functionality in the software increases exponentially as the project moves further in its development life cycle.

Requirement elicitation and analysis is an iterative process that can be split into three distinct phases. The first step is the actual elicitation of the requirements, which can be done through interviews and questionnaires. The second step is to analyze the elicited requirements and make sense of them. In this step it is important to determine whether the elicited requirements are understandable and unambiguous. Use cases and process diagrams can be used for the aforementioned analysis phase. The third step is to document all of the analyzed requirements, which can be done in various forms such as by using summary lists, mockups, use cases, data models, and process flow diagrams.

In this software project, the requirements were collected using one-on-one interviews, group interviews, questionnaires, use cases, and mockups. Some of these are described below.

3.2. Stakeholder interviews

This project started with one-on-one interviews with the customer, who in this case was the prime investigator of the research. In the initial interviews, the customer described what kind of functionality was needed from the software, when it should be available, and who would be its users. During these interviews, specifying questions were made to the customer in order to properly map out the specific requirements. There is often a communication disconnect between the developer and the customer, which is why it is important to ensure that both parties are on the same page in regards to the project. This can only be done via communicating and resisting the urge to make assumptions about the nature of the software.

After the initial interviews, a first version of the software requirements specification (SRS) was made. This document enlisted the basic functionality that the software should have. The SRS was presented to the customer and all of the enlisted requirements were examined. This was in order to affirm that we had truly understood the customer's needs. After re-examining the requirements, the requirement gathering and analyzing process was iterated until a solid view of what needed to be done had been established.

In addition to talking to the actual customer, a communication channel was also established to the primary users of the software. This was done in order to elicit any user-specific requirements. Involving the users in the requirement collecting phase is significant, because it helps to build rapport between the users and the developer. In addition, it helps to prepare the users for change, which was identified in chapter 2 as one of the difficulties to developing a CDMS.

In this software project, we recorded most of the requirement interviews. By recording the interviews, we could solely focus on communicating with the customers and the users, instead of making notes on a paper. By having the recordings, we could later revisit some specific details of the interview as well as to evaluate the quality of the interviewing process itself.

3.3. Use cases

A use case depicts a list of actions that an actor performs when interacting with a system. Use cases provide a lot of benefits in analyzing the elicited requirements. To illustrate, they help in managing the complexity of a given system by focusing on a single usage aspect at once [27]. Use cases are an excellent tool for user-centered design because they are based on the viewpoint that a system should be designed first and foremost for its users [27]. Additionally, by utilizing use cases, software developers are encouraged to visualize all of the alternative solutions before deciding on the final one.

Despite all of the benefits, there are some downsides to using use cases. While excelling at illustrating functional requirements, use cases do not account for non-functional requirements such as usefulness, user experience, and usability. Another drawback of use cases is the fact that they look at each process individually, which often leads to disregarding the interactions between multiple processes [27].

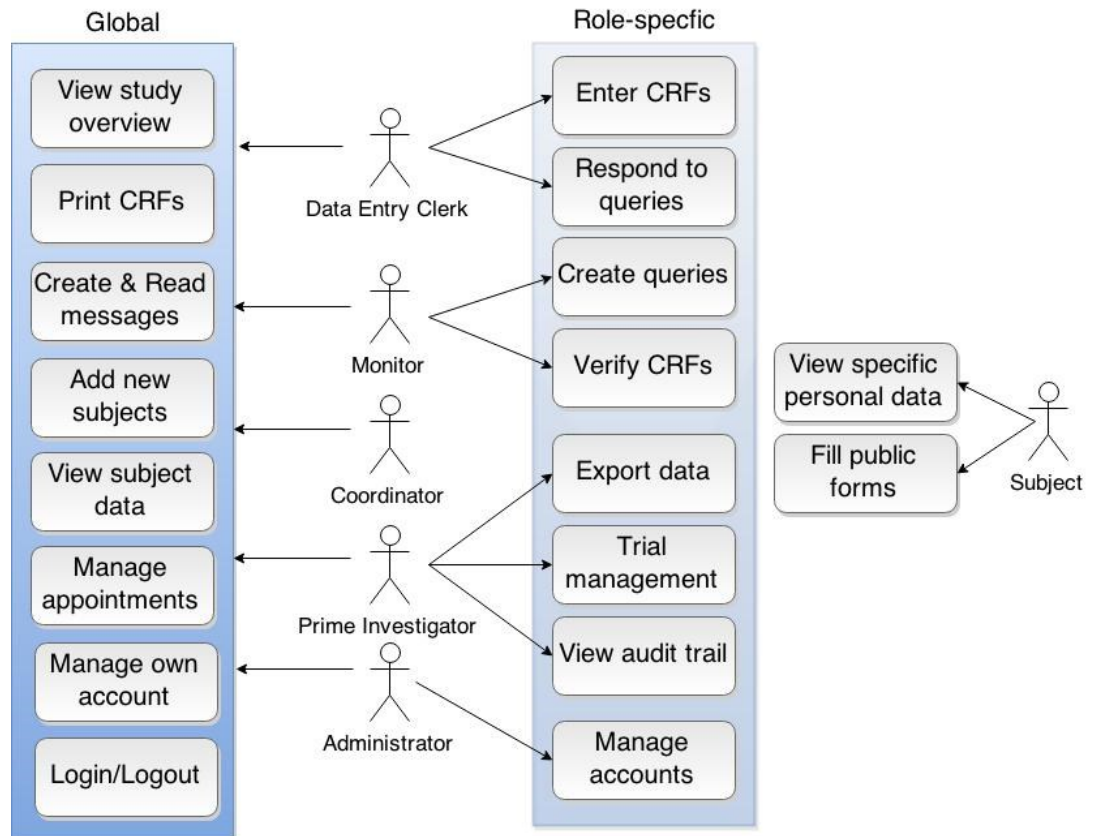


Figure 4. Use case diagram showing actors and main processes.

The use case diagram, illustrated in Figure 4, shows the primary actors and main processes of Genesis. In our system, we have various user groups and roles. One such group is data entry clerks whose job is to enter CRFs and respond to queries. These queries are made by a monitor in case the submitted CRFs are invalid or need clarifying. If the data is entered correctly, the monitor has the power to accept the entry. Coordinators are general managers of the trial whose job is to ensure that all tasks are being completed in a timely manner. Role specific tasks for the prime investigator are to export data, manage the trial, and to view audit trails. System administrators are responsible for managing the existing user accounts. As depicted in the use case diagram, in addition to the role specific duties, all of the users have some global rights such as to view general information about the study, to print data, to create and read messages, and to view subject data. In addition to the primary actors, we have the subjects. Subjects are allowed to fill out some CRFs and to view certain data about themselves.

Table 1. Example of a detailed use case of form submission.

Use Case Element	Form submission
Use Case Number	3
Application	Genesis
Use Case Name	Form submission sequence
Use Case Description	Describes how user can open subject info and submit forms
Primary Actor	Data Entry Clerk
Precondition	Trial ongoing, subject exist
Trigger	-
Primary Flows	<p>User clicks subjects button or tasks to open the subjects protocol</p> <p>Subjects button:</p> <ol style="list-style-type: none"> 1. User opens a view of all subjects in it 2. User finds the correct subject and opens its view 3. User identifies the correct visit and opens it 4. User selects a form from the visit, opening the form 5. User fills the form and clicks the submit button 6. On submit button click, both the client and the server verifies data <p>Search bar:</p> <ol style="list-style-type: none"> 1. User searches for user by their code 2. User finds the correct subject and opens its view 3. User identifies the correct visit and opens it 4. User selects a form from the visit, opening the form 5. User fills the form and clicks the submit button 6. On submission, client and server validates the data. <p>Task menu:</p> <ol style="list-style-type: none"> 1. User opens a view of all tasks on it 2. User finds the correct task and opens the form associated with it 3. User fills the form and clicks the submit button 4. On submit button click, both the client and the server verifies data
Alternative Flows	-

Table 1 is an example of a specific use case. Specific use cases were created for any activities that were considered to be even a bit ambiguous. All of the use case elements were given a name and a number. Each use case was also given a brief description that describes what happens in the use case. Additionally, for each use

case, the primary actor was identified as well as any preconditions that had to be met for the use case to occur. Primary flow depicts the most common way the scenario could unfold. Alternative flow illustrates what happens if something goes wrong with the primary flow. There are no alternative flows in Table 1, but for example, if a user is unable to login to the system, an alternative flow would be to first retrieve a new password and then to login.

3.4. System requirements

System requirements are the blueprints that a developer utilizes in engineering the software. System requirements can be separated into functional and non-functional requirements. A functional requirement states what a single functional piece or part of a system is supposed to accomplish. Non-functional requirements illustrate all of the requirements that are not included in the functional requirements. They are requirements that describe how a piece of software will accomplish the enumerated functional requirements. Non-functional requirements are often used as quality metrics to judge the performance of a system. Tables 2 and 3 describe what high-level system requirements were placed on the developed software. Functional requirements that were decided to be excluded from the first version are marked as (ex).

Table 2. Functional requirements of Genesis.

FUNCTIONAL REQUIREMENTS	
1. Users	<ul style="list-style-type: none"> a. User should be able to be created/deleted. b. Forgotten user passwords should be able to be retrieved. c. Users should be able to change their password. d. Users should be able to be banned from accessing the system.
2. Authentication (Access Management)	<ul style="list-style-type: none"> a. Users should be able to authenticate themselves. Authentication is done via username and password. Correct credentials provide the client with a token, which is used to access resources.
3. Authorization (Access Management)	<ul style="list-style-type: none"> a. On each request, the system needs to verify that the user is authorized to access the requested resource.
4. Manage user privileges (Access Management)	<ul style="list-style-type: none"> a. User roles should be able to be set. Roles define access rights. b. User group should be able to be set to a user. Groups have group specific privileges.
5. Trials (Study Management)	<ul style="list-style-type: none"> a. Trials should be able to be created/modified/deleted. b. Trials should be able to be given details such as description and duration.

<p>6. Visits (Study Management)</p> <ul style="list-style-type: none"> a. Visits should be able to be created and linked to a protocol. Visits contain the CRFs. b. Visit details such as description and date should be able to be set.
<p>7. Forms/CRFs (Study Management)</p> <ul style="list-style-type: none"> a. CRFs should be able to be created/modified/deleted. b. CRFs should be able to linked to visits. c. Each CRF should have an audit log that shows the modifications done to a given CRF.
<p>8. Variables (Study Management)</p> <ul style="list-style-type: none"> a. Variables should be able to be created/modified/duplicated /deleted. b. Variable validation rules should be able to be set.
<p>9. Study sites (Study Management)</p> <ul style="list-style-type: none"> a. Study sites should be able to be created/deleted. b. Study site is a consortium of users, users should be able to be linked and unlinked to study sites.
<p>10. Subjects (Study Management)</p> <ul style="list-style-type: none"> a. Subjects should be able to be created/modified/deleted. b. Subjects should be able to be mass-imported from a data source. c. Subjects should be able to be linked to a study site.
<p>11. CRF entry</p> <ul style="list-style-type: none"> a. Data should be able to be entered into CRFs. CRFs should be able to be submitted. b. CRFs' should be editable for <i>t</i> amount of time after the form has been submitted, which after it is locked and pending for monitor approval.
<p>12. CRF validation</p> <ul style="list-style-type: none"> a. Entry validation (client) Client should validate the entries in real-time according to the set validation rules. b. Form submit validation (client) Client should validate that all of the required fields have been entered on clicking the submit button. c. Form validation (server) Server should validate that all of the required data has been received, and that all of the data is in correct format. d. Form validation (monitor) Monitor should be able to validate that the CRF entry.
<p>13. CRF verification</p> <ul style="list-style-type: none"> a. Monitor should be able to flag invalid CRFs. b. Monitor should be able to verify a CRFs.
<p>14. Audit trail</p> <ul style="list-style-type: none"> a. System should store what changes have been done to a CRF. b. System should store who made the changes, and when.

<p>15. Calendar</p> <p>a. Calendar events should be able to be created/modified/deleted.</p>
<p>16. Metadata collection</p> <p>a. Metadata should be collected on who enters or validates the CRF, how long it took to enter or validate the CRF and so forth.</p>
<p>17. Printing (ex)</p> <p>a. User should be able to print out a completed CRF, as well as an empty CRF.</p>
<p>18. Communication (ex)</p> <p>a. Users should be able to send messages to each other.</p> <p>b. Users should be able to read messages from one another.</p> <p>c. Administrators should be able to send messages to all system users.</p>
<p>19. Data export (ex)</p> <p>a. All data should be able to be exported to Excel, CSV, SPSS or similar formats.</p>
<p>20. Study Overview (ex)</p> <p>a. Authorized users should be able to view, at what phase the trial is at (how many tasks have been completed and so forth)</p>

Table 3. Non-functional requirements of Genesis.

NON-FUNCTIONAL REQUIREMENTS
<p>1. Usability</p> <p>a. The system should use language that is intuitive to the users.</p> <p>b. The system should have workflows that mirror those of real-life.</p> <p>c. The system should use widgets that are intuitive to the users.</p>
<p>2. Reliability</p> <p>a. The system should not have downtime during regular working hours.</p> <p>b. The system should not lose data after CRF submission.</p> <p>c. The system should make sure that all data is stored in correct format; strings as strings and numbers as numbers.</p> <p>d. The system should make sure that required data has been entered.</p> <p>e. The system should disallow users from modifying locked CRFs.</p> <p>f. The system should handle exceptions and announcing them to the user.</p>
<p>3. Performance</p> <p>a. The system should be responsive.</p> <p>b. The system should support at least 30 concurrent users.</p> <p>c. The system should have high scalability.</p>
<p>4. Supportability</p> <p>a. The system should be able to be ported later to a mobile phone or tablet.</p> <p>b. The system should be maintained by a person with a proficiency in information technology.</p> <p>c. The system should easily be extendable.</p>

5. Implementation

- a. Software should be web-based as users will be at various different locations.
- b. The users' browsers should have JavaScript enabled.
- c. The users should have up-to-date browsers.
- d. The source code should be commented and understandable for easy maintainability.
- e. All crucial functions of the software should be unit and e2e tested.

6. Operation

- a. The system will be maintained by Joonas Rahko & Olli Okkonen.

7. Packaging

- a. The software will be delivered to end-users in a software-as-a-service based model. This means only modern browsers, Internet connection and JavaScript is required.

8. Safety, Security & Privacy

- a. The system should not be able to be accessed by unauthorized users.
- b. The system should have high minimum password requirements and users should be trained on selecting unique passwords.
- c. The system should be immune to most common security threats such as CSRFs and XSS.
- d. Data should be maintained on a separate database server, protected by a firewall.

4. DEVELOPMENT METHOD, TOOLS AND APPROACHES

This chapter begins with the description of the development method that was chosen for this project. Afterwards, modern web development approaches and tools that were considered for this project are discussed comparatively. The underlying reasoning behind the selection of each development tool and approach is also illustrated in this chapter. Lastly, the way version control was conducted in this project is described.

4.1. Agile software development

CDMSs are people focused software. To explain, they are used by people to store data about other people. The information elicitation, an important process and art of its own, is based on the interaction between the nurse and the subject. A well designed CDMS should support this interaction, which is why the developers need to understand how the research staff and the subjects interact with each other. This understanding can only be gained by actively interacting with the research nurses.

Agile software development is a development method, which is best suited for people focused software. This is because agile software development is based on rapid cycles, in each of which a new version is first developed, and then tested with the users [28]. Afterwards, the user feedback is analyzed and the process continues [28]. By continuously interacting with the users, a developer can better grasp how a CDMS should be built to support the interaction between the subject and the research nurse. According to the Manifesto for Agile Software Development, developers should focus on individuals and interactions over processes and tools. This statement demonstrates the people-oriented nature of agile development [29].

Among the aforementioned reasons, agile methodology was adopted in Genesis development for another reason. Our initial end goal was not clear, which is why the classical waterfall approach was not the best choice for this project. By keeping the iterations short, the software could be steered towards a better, yet unknown direction. Moreover, agile development helps to reduce the time to the market because the software can be worked on meanwhile requirements are still being gathered [30].

Another benefit of agile development is active communication with the users, which helps to build ownership of the software for the users. Additionally, as the users see that their ideas are valued, trust is also established between the developers and the users. Moreover, active testing also functions implicitly as training, as the users slowly familiarize themselves with the software while it is still being implemented.

We were able to deploy agile software methodology in the development of Genesis because our workstations were at the study site, enabling us to constantly gather user feedback. Without the users to test the software, there can be no agile development, as user feedback is paramount.

4.2. Client-side versus server-side rendering

In developing web-applications, an important architectural decision has to be made on where to render the HTML. There are two ways to conduct rendering, either on the server-side or the client-side [31]. The current paradigm is to utilize a mixture of both methods. The traditional approach has been to render templates on the server-side [32]. This is because front-end web development tools have not been around for long and historically browsers used to have compatibility issues [32]. With the release of JQuery, and introduction of various JavaScript based frameworks, web front-end development has been revolutionized [32]. Both client- and server-based rendering have their unique strengths and weaknesses that need to be evaluated in order to find the best option for a given project. The pros and cons of both client-side and server-side rendering are illustrated in the Table 4. The ✓ symbol indicates a positive aspect whereas the ✗ symbol indicates a negative aspect.

Table 4. Comparison between pure server- and client-side rendering [32][33].

Server-side rendering	Client-side rendering
✓ Performance is independent of users hardware and software because data is rendered on the server	✗ Performance depends on user hardware and software because data is rendered on user browser
✓ Page rendering is faster, no extra DOM manipulation	✗ Page rendering is slower, extra DOM manipulation
✓ Browser independent, server deals with rendering	✗ Older browsers might not support all client-side rendering features
✓ Faster initial paint time, slower last paint time. Better if page needs to be shown immediately	✓ Faster last paint time, slower initial paint time. Better if all data is needed before view can be shown
✗ Despite going with server-side rendering, client-side programming language (JS, HTML, CSS) knowledge still required	✓ No knowledge of server-side programming languages is required. Makes developing easier for front-end developers
✗ Server needs to worry about data visualization, no decoupling	✓ Decoupling, separation of concerns, client visualizes data
✗ A lot of traffic between server and client, view alterations need to be done at the server. User interactions needs to confer with server	✓ Helps to reduce traffic between client and server. Server only responds to API calls and serves static HTML templates

Client-side rendering was chosen for this project because it was found to be more favorable for the following reasons:

1. Users had modern browsers, negating one of the client-rendering weaknesses.
2. Users are expected to have longer usage sessions, which is why initial load time does not matter. Interaction after the initial load has to be optimized.
3. Both the web client and the server-side could be developed separately. This was very important as we had two developers working on the whole system.

4.3. JavaScript framework selection

After having decided to go with client-side rendering, the next decision, which was one of the key ones, was to choose the client-side JavaScript framework. There were various decent options available to choose from, namely AngularJS, Backbone.js, and Ember.js, which is why some research had to be put into mapping out the pros and cons of each alternative. One of the key factors in choosing a framework is community support. Strong community support is especially beneficial because strong communities often have a lot of existing modules that a developer can use in an application. In addition, big communities often offer an existing broad library of information that a developer can tap into. Table 5 illustrates that AngularJS has an exceptionally strong community compared to the other frameworks.

Table 5. Community support metrics and results on August 16th, 2014 [34].

Metric	AngularJS	Backbone.js	Ember.js
Stars on Github	~ 27.2k	~ 18.8k	~ 11.0k
Third-Party Modules	~ 800	~ 236	~ 21
StackOverflow Questions	~ 49.5k	~ 15.9k	~ 6k
YouTube Results	~ 75k	~ 16k	~ 6k
GitHub Contributors	928	230	393
Chrome Extension Users	~ 150k	~ 7k	~ 38.3k

In addition to having a strong community, AngularJS seems to be growing in interest among developers. This is an important statistic, as it is vital that a framework is expected to have support in the long run. A comparison of AngularJS, Backbone.js, and Ember.js interest can be seen in Figure 5.

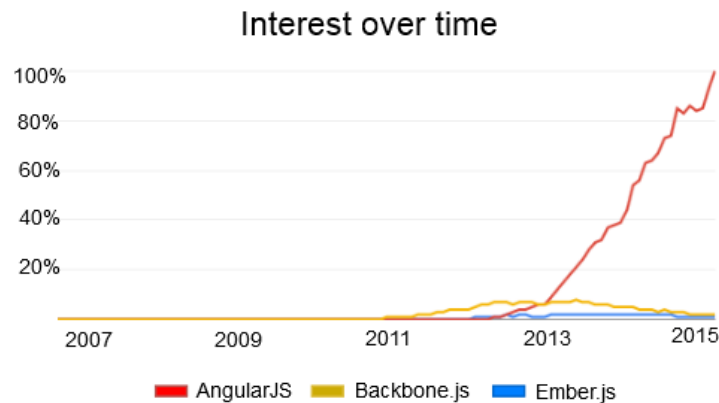


Figure 5. Graph of JavaScript framework popularity based on Google trend on 4.10.2015. Y-axis represents number of searches relative to the highest number of searches [35].

Moreover, Google has helped to develop and has used the framework to build many of its applications. Due to this, it is unlikely that Google would migrate away from AngularJS in the near future, helping to create stability for the framework [34]. AngularJS's future prospects are further improved with the oncoming release of AngularJS 2.0 [36]. Due to the aforementioned reasons, the AngularJS framework was chosen over the other available options.

4.4. Web service architecture selection

A web-based CDMS is a web service. A web service is defined as “a software system designed to support interoperable machine-to-machine interaction over a network” [37]. Representational State Transfer (REST) and Simple Object Access Protocol (SOAP) are two of the mainstream options to offer web services [38]. REST is an architectural approach, which gives certain rules on how a web service should be developed, whereas SOAP is a protocol, which strictly defines a set of rules for XML-based message exchange [38]. REST and SOAP are similar in the sense that they both use HTTP as their transport layer [38]. Both of the aforementioned approaches are actively used on the web, which is why an analysis had to be made on which one to deploy in Genesis [38]. REST and SOAP are compared in the Table 6 below.

Table 6. Comparison of REST and SOAP approaches [38][39][40][41][42].

REST	SOAP
✓ Permits many data formats, not just XML. While adding an extra layer of complexity, newer formats can fare better compared to XML	✓ Permits only XML as a data format. Beneficial because of reduced complexity due to not having to support many formats
✓ Client-side coding is less complex, server-side becomes more complex	✓ Server-side coding is less complex, client-side becomes more complex
✓ REST is more scalable due to statelessness, which also allows caching	✗ SOAP based web services often maintains state for clients, using states prevents caching
✓ REST has better performance: higher throughput of clients and kilobytes. Faster response time with number of clients	✗ SOAP has worse performance, loses to REST in throughput of clients and kilobytes. Slows down as number of clients increases
✓ REST has been found to be 9 to 30 times faster on mobile applications	✗ Much slower in mobile applications compared to REST
✗ No in-built security mechanisms, relies mainly on SSL, additional layers of security need to be built-in	✓ Supports WS-Security, which has some enterprise security features, such as tracking identity through intermediaries
✗ REST relies on clients to handle communication errors, retrying etc.	✓ Has in built messaging system with successful/retry logic built-in

One of the goals of our software is to serve multiple clients on different platforms, such as tablets and phones. Further, REST offers a much higher overall performance, especially in mobile applications. This is why in Genesis we chose to deploy REST over SOAP.

4.5. Web service data interchange format selection

Because REST was chosen as our web service type, a choice could also be made on what communication language to use on the client-side to communicate with the server. While other options are available, the most common formats that are used on the web are JavaScript Object Notation (JSON) and Extensible Markup Language (XML). While a REST API can support both languages, there is no reason to use multiple data interchange formats on the client-side. In determining, whether to use JSON or XML, some research had to be conducted in order to reach a decision on which one to harness. The results of the literature review can be seen in Table 7.

Table 7. Comparison between JSON and XML formats [43][44][45][46].

XML	JSON
✓ XML is simple, open and interoperable	✓ JSON is simple, open and interoperable
✓ Self-describing data and internationalization, Unicode support	✓ Self-describing data and internationalization, Unicode support
✓ XML allows for storing any type of data, making it a bit more flexible	✗ JSON is limited to classical data types (strings, integers, numbers...)
✓ XML has inbuilt XML document validation	✗ JSON does not have inbuilt validation
✗ Increased data type flexibility of XML creates an additional layer of complexity, and makes XML less human readable	✓ JSON is easier for humans to read because JSON supports less data types. In addition, the structure of the data itself is standardized
✗ While XML supports any file of any format, it creates security threats, as an executable malicious file can be transferred (XML-RPC)	✓ Simplicity of JSON data structures makes it impossible to send malicious executable files
✗ Slower format for transporting large quantities of data compared to JSON	✓ Faster format for transporting large quantities of data compared to XML
✗ Uses more hardware resources than JSON	✓ Uses less hardware resources than XML
✗ More verbose compared to JSON	✓ Less verbose compared to XML
✗ Not natively supported by JavaScript, needs to be parsed on the client-side	✓ Natively supported by JavaScript, no need to be parsed on the client-side

Through investigation it was found that transferring JSON encoded data is generally faster compared to transferring XML encoded data [46]. In addition, JavaScript natively supports JSON parsing and deserialization, which makes client development considerably easier. For the aforementioned reasons, JSON was chosen as the client-server communication language.

4.6. Version control system selection

Version control is very important in software development because it allows developers to keep track of changes that have occurred during the development. Version control systems (VCS), such as Git, Apache Subversion (SVN), and Concurrent Versions System (CVS) make version control much easier. They allow developers to revert files or even an entire project back to a previous state, compare changes over time, see who has made modifications to the source code, and much more. A traditional way of handling version control is to copy files into timestamped directories. However, this process is prone to human errors, as wrong files can be overwritten and actual development files can be lost. Moreover, without a VCS two people cannot work efficiently and safely on the same file at once. [47]

In this project, we knew that we would be working from various locations such as the hospital, the university, and our homes. Moreover, in each of these locations, we would not necessarily have access to the Internet. Due to the nature of our work, we needed our version control system to be able to operate locally without the need for the Internet. Git supports this functionality by default, which is why it was deployed in this project. Git icon can be seen below in Figure 6.



Figure 6. Icon of Git version control system.

5. WEB CLIENT DESIGN AND IMPLEMENTATION

This chapter begins by illustrating the overall architecture and navigation flow of the system. Thereafter, the general UI of Genesis is illustrated. After discussing the UI, we discuss how the CRFs in Genesis were designed and built. Thereafter, the different states of CRFs and how those states affect the UI are shown. Additionally, this chapter shows how the information exchange between the client and server is handled. The means to ensure information security are also depicted in this chapter.

5.1. System architecture

Figure 7 depicts the system architecture for Genesis. The web client uses the model-view-controller (MVC) architectural software pattern. Local storage can also be utilized for temporary data storage. All of the web client communication goes through AngularJS \$http service, which enables the user to perform basic http operations. All of the http operations are intercepted using an http interceptor. The interceptor adds a JSON Web Token (JWT) to the header of each http request. The JWT lets the server know that the client has the right to access the data. All of the data transmissions between the server and the client are done using JSON. The server side uses the Django REST framework and a PostgreSQL database.

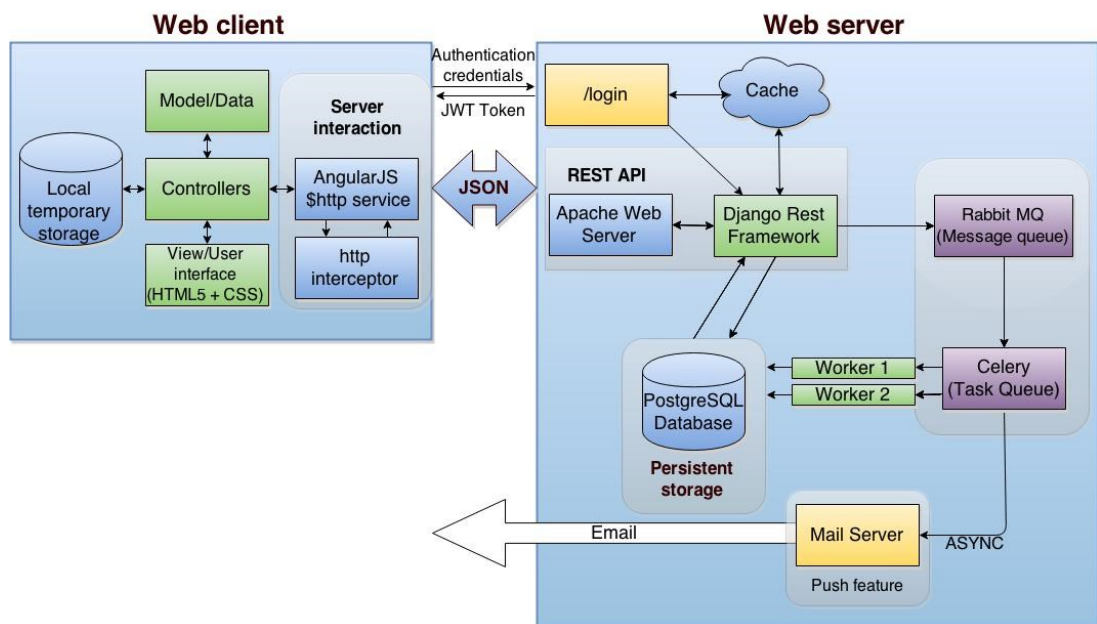


Figure 7. System architecture diagram for Genesis.

5.2. User interface navigation flow

The user interface navigation flow is illustrated in Figure 8. When users enter the trial website, they are first taken to the login page (Appendix 1). On the login page, users can enter their credentials or retrieve a forgotten password. Users might also be prompted for a new password if the previous one has expired. After inputting correct credentials, users are redirected to the dashboard state (Appendix 2). The dashboard is one of the states of the main view, which is the root state for the client. In the main view, users can enter the account state (Appendix 3). In this state, the users can change their password and other account details. In the subjects state (Appendix 4), users can add new subjects to the trial or view subject related information. Subject specific information is contained in CRFs, all of which are generated from the JSON schema. In the main view, users can also view their tasks (Appendix 5), which act as links to different parts of the system. For example, the user might have a task to complete a CRF for a subject. In the messages (Appendix 6), users can communicate with the personnel of the study. The calendar state (Appendix 7) is reserved for managing appointments. In the study overview (Appendix 8), the progression of the study can be viewed. The dashboard is a general state where users can see if they have received new messages or tasks. In addition, the changelog for new versions of the software can also be viewed from the dashboard.

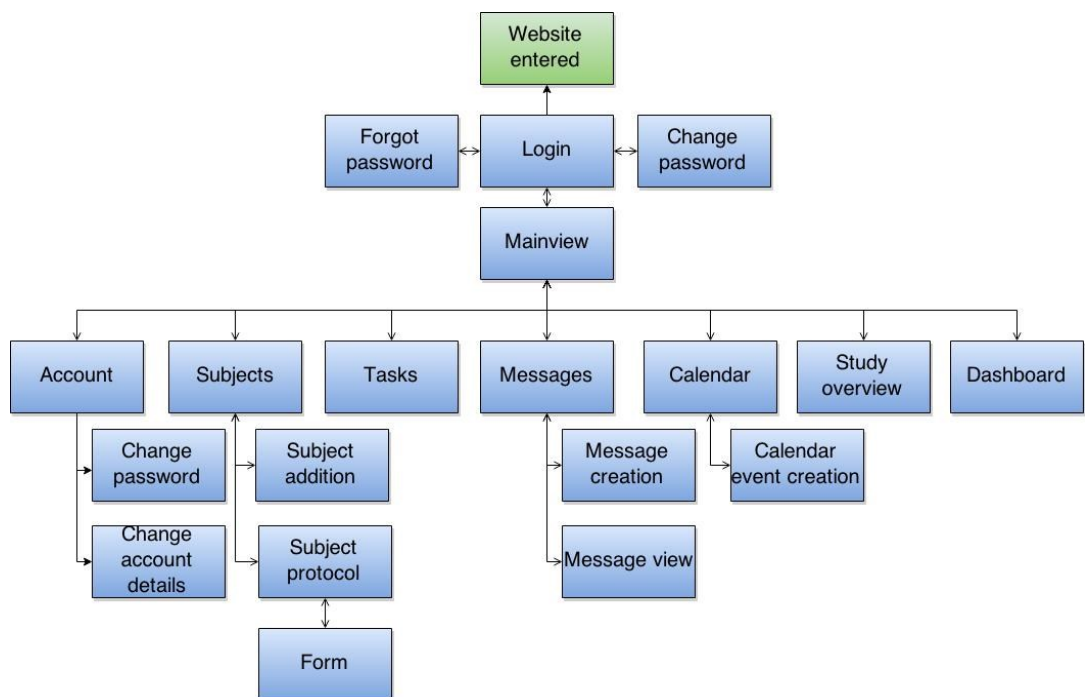


Figure 8. Flow chart of user interface navigation.

5.3. User interface design

The wireframe illustrated in Figure 9 was the base for the user interface design for Genesis. On the left, we have the static state navigation, which is the user's primary mean of navigating within the application. The static state navigation contains all of the primary states, such as subjects, messages, and tasks. Dynamic sub-state navigation is also used to aid navigation within the parent state. This sub-state navigation is implemented using a bread-crum approach, where previous states are shown and are accessible. The dynamic content area contains all of the dynamically changing content, which depends on the current state, as well as on the data provided by the backend. Graphical layout of the user interface is illustrated in Figure 10.

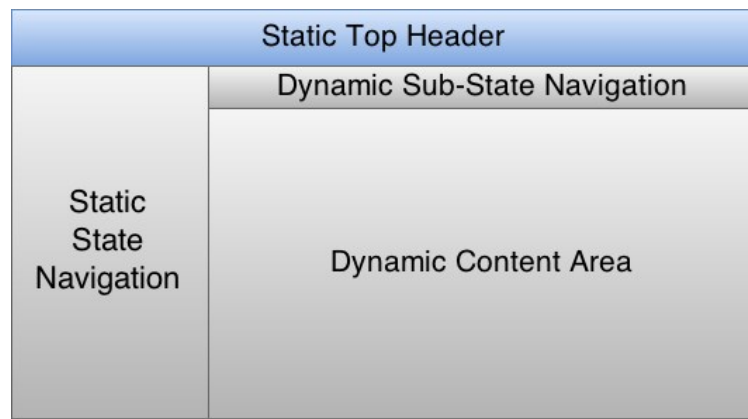


Figure 9. Wireframe of user interface design.

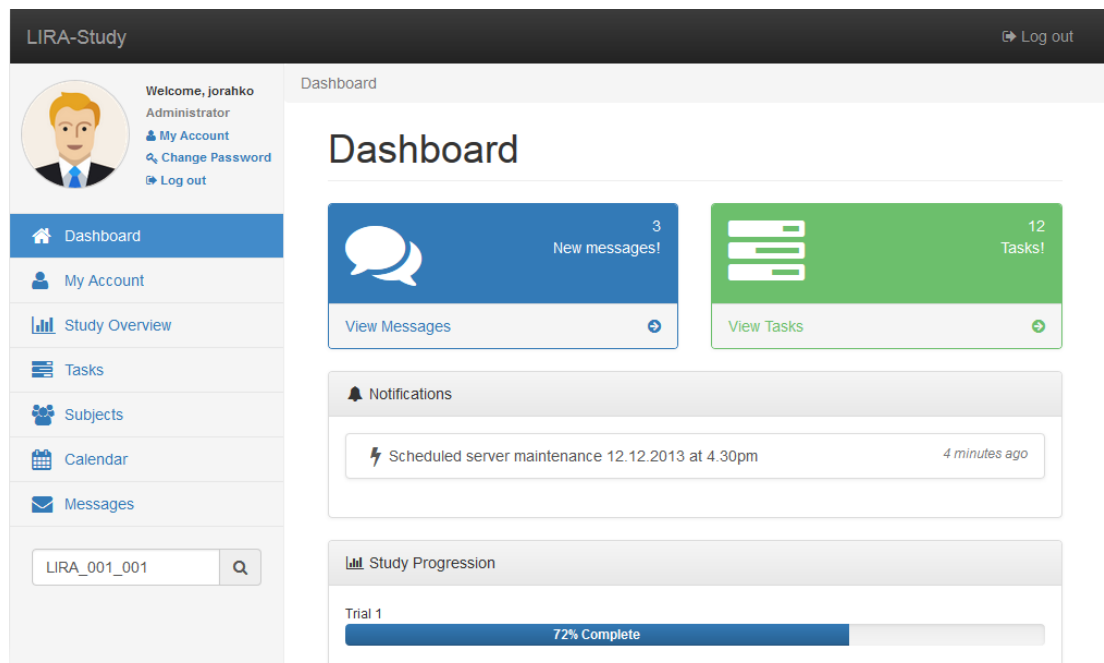


Figure 10. Graphical layout of the user interface.

5.4. CRF design

In CDMSs, most of the system-user interaction happens through CRFs. This is why CRF design should be one of the focal points when developing a CDMS. In Genesis, we strive to offer the best possible user experience through an efficient CRF design. Excellently devised CRFs do not only improve user experience, but they also help to reduce human related entry errors. Moreover, with good design, the time for data entry can be reduced, and thus, scarce resources can be saved. To achieve the aforementioned benefits, good CRF design principles must be utilized. Here are some of the factors that were taken into consideration in the CRF design for Genesis.

5.4.1. Label positioning



Figure 11. Illustration of label positioning options.

There are various legitimate ways to position field labels inside a form. These are shown in Figure 11. The pros and cons of each option are depicted in Table 8.

Table 8. Comparison of label placements options in a CRF [48][49].

Option 1	Option 2	Option 3
~500ms saccade duration	~200ms saccade duration	~50ms saccade duration
<ul style="list-style-type: none"> ✓ Requires less vertical space ✓ Labels can be longer than fields ✓ Labels are easier to scan, no field separation ✗ Heavy cognitive load ✗ No localization support, long labels push the input fields ✗ Form requires more horizontal space ✗ Far from the corresponding field, hard to associate 	<ul style="list-style-type: none"> ✓ Requires less vertical space ✓ Labels can be longer than fields ✓ Labels easy to scan, no field separation ✓ Close proximity to corresponding field, easier to associate ✗ Heavy cognitive load ✗ No localization support, long labels push the input fields ✗ Form requires more horizontal space ✗ Long labels that warp are harder to read 	<ul style="list-style-type: none"> ✓ Requires less horizontal space ✓ Smallest cognitive loading ✓ Allows easy localization, label lengths can change ✓ Longer labels are easier to read ✓ Easy to associate field with label ✓ Best for mobile devices ✗ Requires a lot of vertical space ✗ Labels can be harder to scan due to field separation

In Genesis, we decided to use option 3 of placing the label on top of the input field. One of the goals of Genesis is to be able to provide localization support in the future. Placing the label on top of the input field allows for the labels to vary in length without affecting the layout of the form. Another reason option 3 was chosen, was to reach fast CRF completion times. Option 3 had been proven to perform the best in this department, which is why it was deployed.

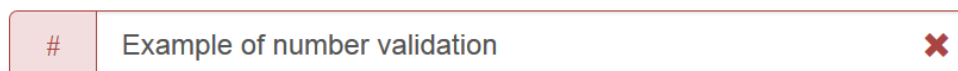
5.4.2. Validation

In CDMSs, data entries can be validated both on the server-side and on the client-side. Any CDMS system should at least deploy server-side validation in order to ensure the reliability and quality of the collected data. While server-side validation is enough for reliable CRF validation, relying only on it does not provide the best user experience as the user has to wait for the server [50]. Therefore, a well-designed CDMS should also be accompanied with client-side validation. With client-side validation, it possible to use rules and conditions to validate user input instantaneously [50]. Despite the benefits, implementing a client-side validation requires some extra effort from the developer. However, the improved user experience is worth the cost. After deciding to perform client-side validation, decision must be made on what to validate, when to validate, and how to validate.

There are three ways to schedule validation. To illustrate, the first way is to validate the web-form upon submit. With on-submit validation, the user does not know whether the fields are correct or incorrect until the form has been submitted. While this is not the optimal way of performing validation, it is sometimes the only available option if, for example, JavaScript is not enabled on the client-side [50]. If we add client-side validation into the mixture, the fields can also be validated in either real-time or on-blur. The result is a more responsive and visually richer validation [50]. Real-time validation means that the input fields are being validated continuously, with each character or number input. However, by showing an error message prematurely, the user experience might be negatively affected. Real-time is the fastest way of giving feedback to the user about field validity. On-blur validation means that a given field is validated after the field specific entry has been completed.

In Genesis, both server- and client-side validation are utilized. On the client-side, the web-form is being validated using the on-blur method. If the user enters erroneous data to an entry field, the user interface tells the user that the entry is invalid, and shows an example of a correct entry. This is illustrated in Figure 12.

Sample number input



You did not enter a valid number. For example: 5.0

Figure 12. Example of a number validation message.

In Genesis, all of the input fields have been designed to allow for many ways to input the same data. An example of various acceptable ways to enter data would be the date field. Genesis accepts period, dash, and slash as date separators. In addition to using on-blur validation, the client also utilizes on-submit validation. On-submit validation is used for checking that all of the required fields have been filled before submitting the CRF. If any of the required fields have not been completed, the system notifies the user and highlights the required fields. This can be seen in Figure 13. In addition to letting the user know when an entry is incorrect, Genesis also gives the user feedback on a successful data entry. Example of this is shown in Figure 13.

Sample Form

Sample date input

Required

Sample number input

Required

Sample text input

There are missing required fields in the form!
The form cannot be submitted until the fields have been filled.

Figure 13. Validation example of different fields.

5.4.3. Icons

As we can see in Figure 13, icons are also actively being used in Genesis web-forms. The purpose of the icons is to improve the affordance of the UI. Affordance means how intuitive using the UI elements is [51]. By using a calendar icon for date fields, a pound sign for number fields, and a paragraph icon for text fields, the user can faster realize what type of data the entry field requires. Icons are also used on the right of the input fields to indicate the validity of the field.

5.5. CRF compilation design

In order for the web client to serve various trials, all of the CRFs need to be created based on data received from the backend. In Genesis, CRFs are built based on a JSON Schema, provided by the backend. JSON Schema is a JSON based media type used to define the structure of JSON data [52]. The purpose of the JSON Schema is to define validation, documentation, hyperlink navigation, and interaction control of JSON data [52]. According to the JSON Schema standard, implementations can define their own unique keywords, but these should not override the ones set by the standard [52].

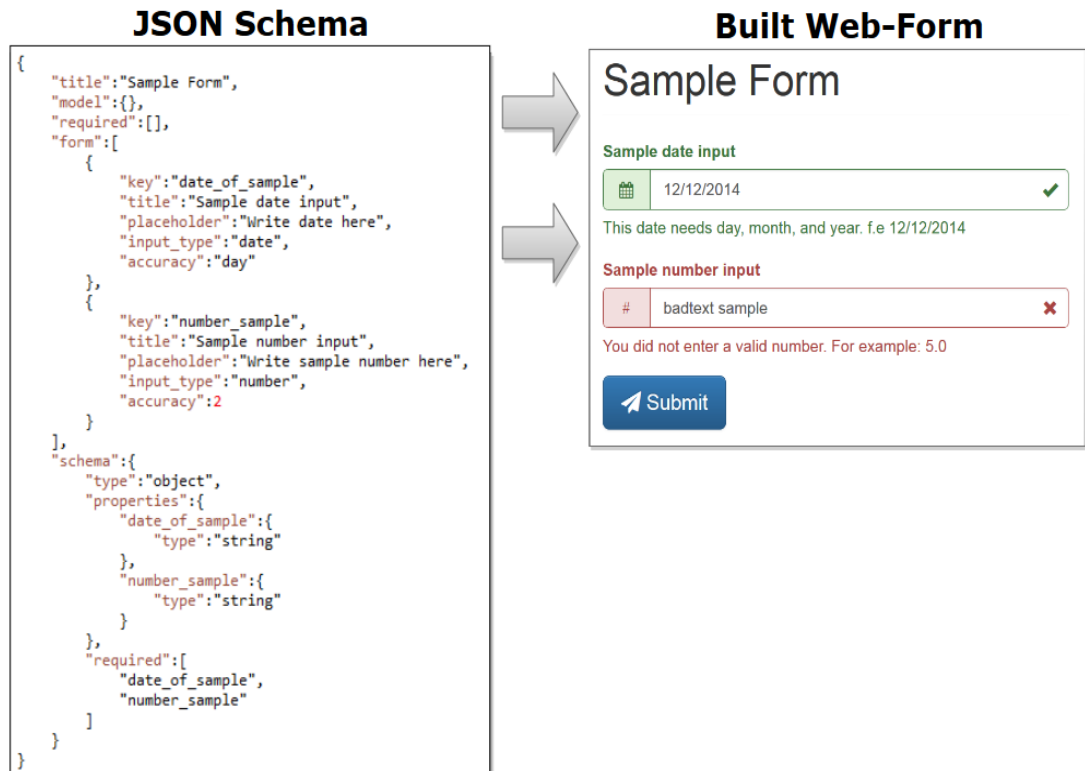


Figure 14. Example of interpreting JSON schema into an HTML form.

Figure 14 illustrates how the JSON schema is interpreted by the client and converted into an HTML based web form on the client side. In order to make the building of the JSON Schemas a lot more effortless, a GUI based form builder will later be developed. Currently, Genesis uses a tool that creates the CRFs from a comma-separated values (CSV) file. Having a GUI based web-form creator and manager, would enable the prime investigators to build their own variables and CRFs, as well as to define their own validation rules.

5.6. CRF entry and verification flow

In addition to the automatic entry validation, most CRFs are verified by a monitor in the LIRA-Study. The monitor's role in a trial is to verify and query forms. Query is an error message generated by the monitor when there is a discrepancy in the data entry. The process of CRF verification is illustrated in Figure 15. Due to the different CRF states, the same CRF has to be shown in multiple different formats depending on both the CRF status and the role of the user viewing the CRF. The way different states of CRFs can change the GUI can be seen in Figure 16.

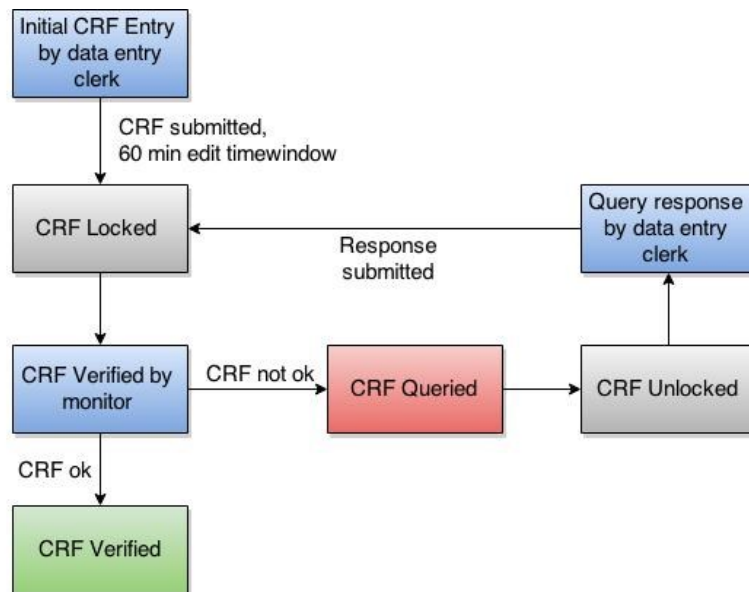


Figure 15. CRF entry and verification flow.

Visit 1 - 01.01.2014	Example Form	
Visit 2 - 01.01.2014	Example Form	31 minutes of edit-time remaining
Visit 3 - 01.02.2014	Example Form	Locked
Visit 4 - 01.03.2014	Example Form	Queries
Visit 5 - 01.04.2014	Example Form	Locked Validation-required
Visit 6 - 01.05.2014	Example Form	Locked Validation-required
Visit 7 - 01.06.2015	Example Form	Locked Validated

Figure 16. Different CRF states in the protocol.

5.7. Interaction with backend

In Genesis, the backend as a service (BaaS) approach is utilized. BaaS offers a way for clients to link their frontend to a backend by an application programming interface (API). In Genesis, the client utilizes a REST API. In REST, resources and basic HTTP methods are utilized to exchange and modify data. Resources are individual collections of data, identified in requests by using a unique resource identifier (URI). These resources are usually offered in either JSON or XML formats. The aforementioned methods, or verbs, utilized in the REST architecture are GET, PUT, POST, and DELETE. For example, the verb GET would be utilized to retrieve a collection of data from the server. [53]

We decided to utilize a REST API in Genesis because the same API could be reused to serve other clients such as mobile devices. Additionally, REST helps to build a clean separation of concerns between the client and the server as the implementation of all of the resources is invisible to the client. Moreover, the separation of concerns allowed us to develop the client and server as separate components. Because the client consumes the resources through an abstract interface, it is possible for us to alter the backend without influencing the client.

5.8. Information security

One of the key aspects of a browser-based CDMS is security. Web browsers are the most commonly used client-side application, which is also why they are often one of the main targets for hacking [54]. In addition to being a prime target for malicious endeavors, web-browsers are being used by users with little-to-no experience in information security [54]. Moreover, with new web-technology being released daily, and with new vulnerabilities being discovered all the time, merely keeping up with web-security is a time consuming task. All of these factors combined make information security a difficult, yet a vital aspect to consider.

Information security has been heavily considered in the design of Genesis. In order for the implemented information security measures to be efficient, interplay between the server and client is necessary. A basic approach to web-application security is that the server should never trust anything from the client [55].

5.8.1. Securing communication

In Genesis, all of the content is served over Hyper Text Transfer Protocol Secure (HTTPS). With HTTPS, all of the communication between the client and the server is insulated using either Secure Sockets Layer (SSL) or Transport Layer Security (TLS) encryption [56]. Both SSL and TLS protocols deploy an asymmetric Public Key Infrastructure system [56]. Without the encryption, all of the data travelling between the client and server would be in plain text, making the data vulnerable to man-in-the-middle (MITM) attacks [56]. In MITM, a third, unauthorized party hijacks the connection and is able to secretly relay and alter the communication between the server and the client. HTTPS encryption renders the data impossible to decrypt, even if somebody breaks into the connection [56].

5.8.2. Preventing cross-site scripting

Cross-site scripting, or XSS, is a vulnerability in web applications where the attacker injects client-side script into web pages viewed by real users. The malicious scripts are injected into fields that are used to generate output, and can be used to bypass access controls such as the same-origin policy. In Genesis, we prevent script injections by sanitizing all of the inputs given by the user. This is done using AngularJS's ngSanitize module. NgSanitize first parses the website HTML into tokens [57]. Then, after parsing the HTML in to the tokens, all of the tokens that are deemed safe according to a whitelist, are then serialized back to HTML strings [57]. Using this method, no unsafe scripts make it through. Input sanitation helps to protect the application from cross-site scripting attacks (XSS), as well as from SQL injections. In order to further immunize the system from XSS attacks, a Content Security Policy (CSP) is deployed. CSP tells the client from which location and what type of resources it is allowed to use [57].

5.8.3. Preventing cross-site request forgeries

Cross-site forgery (CSRF) is an attack where a harmful web application forces the user's browser to perform unwanted actions on another trusted application running in the browser [58]. CSRF attacks cause damage, because they trick the victim to send a malicious request on their behalf [58]. CSRF attacks aim to cause a state change on the server. There are various prevention measures to protect against CSRFs. One method is to use same-origin policy [58]. Same-origin policy ensures that a document retrieved from location A, cannot access a document retrieved from location B [58]. This functionality is implemented by the web browsers. The strict same-origin policy can be relieved with the use of cross-origin resource sharing (CORS), which makes cross-domain communication possible.

Application developers can also take action to prevent CSRFs by utilizing an unpredictable challenge token on each HTTP request and associating it with the users unique session. Tokens verify that all requests come from the user [59]. In Genesis, we have deployed tokens, which help to build a strong immunity against CSRF attacks.

5.8.4. Password requirements

Passwords are the first-line of defense against malicious users. In Genesis, information security is also improved with the use of good password standards. To illustrate, Genesis enforces a minimum password length requirement, as well as requires the user to use special characters in the password. Moreover, dictionary words are not permitted in the password, as they are susceptible to dictionary attacks [60]. In addition to these standards, each password is set to expire after a certain amount of time. Brute force attacks are prevented by only allowing for a limited number of attempts to login into the system until the account is locked.

Studies have shown that while users accumulate more accounts for different web-services, the number of passwords the use remains roughly constant [61]. This indicates that many of the users practice password reuse. Specifically, one study showed that the amount of password reuse per account is on average four accounts

per password [61]. Recycling passwords creates many security vulnerabilities. If an unauthorized person gets a hold of the user's password, he or she most likely has access to the user's other accounts using that password as well. Because selecting a unique password is significant for the overall security of a given system, we will train the users on the importance of a unique password when Genesis is deployed.

5.8.5. Authentication

Because HTTP by its nature is both stateless and session-less, the client must authenticate itself on each HTTP request by providing an authentication token or authenticator [60]. Authentication in Genesis is done using a token-based approach. This is a modern method of conducting authentication, and is contrary to the traditional method of using cookies. Both of these methods are illustrated in Figure 17.

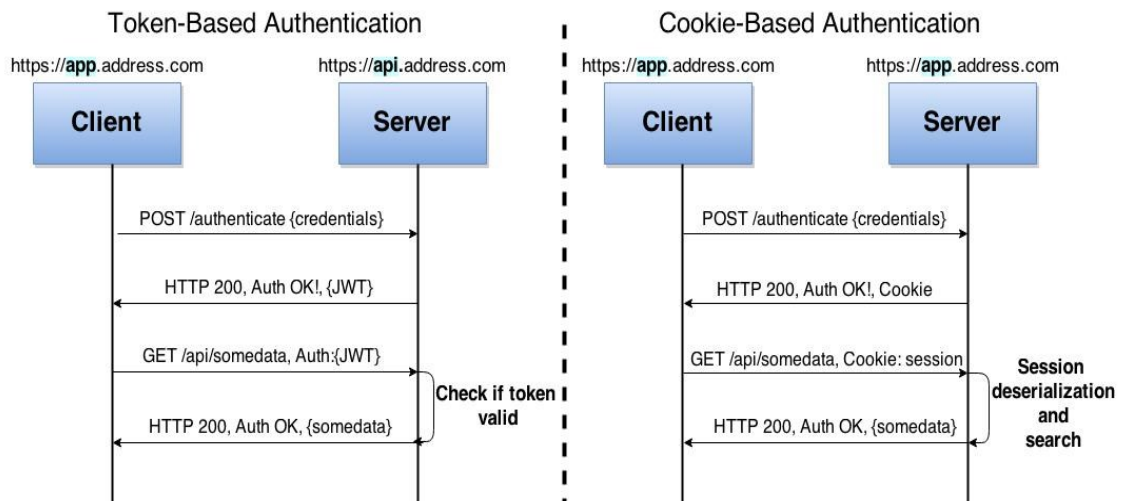


Figure 17. Illustration of token- and cookie-based authentication.

There are various reasons why token-based authentication is used instead of cookie-based authentication. First, with cookies, all of the sessions have to be stored somewhere, and therefore, as the number of users grow, so does the amount of sessions. With tokens, storing sessions is not required, which allows for server-side scalability [62]. As Genesis endeavors to serve multiple users, server-side scalability is of vital importance. Second, using tokens makes authentication a lot easier for mobile devices, because by using tokens, you do not have to deal with cookie containers [62]. Third, using tokens allows for looser decoupling as tokens do not force you to use the framework specific authentication scheme [62]. Another key benefit of using tokens is that cross-origin resource sharing (CORS) is made effortless [62]. This is due to being able to make AJAX calls to any server or domain, which is the result of storing the user information in the header of each HTTP request [62].

6. WEB CLIENT EVALUATION

This chapter illustrates some of the aspects of Genesis's performance that were evaluated. First, we discuss automated testing, which has been deployed to ensure the robustness and functionality of the system. Thereafter, usability testing and the results of the first testing iteration are illustrated. Additionally, the satisfaction of the primary customer is evaluated.

6.1. Automated testing

In Genesis, automated testing was deployed to ensure the functionality of significant pieces of the application, such as the robustness of an input field or successful transmission of data. Test automation helps to reduce the chance of regression bugs that might occur due to refactoring the code [63]. In Genesis, two different types of automated testing were used: unit testing and end-to-end testing.

Unit testing was used to test the low-level functionality of the system. Low-level functionality testing means that the tests are run on a very limited set of code such as a function. For unit testing the client code, Jasmine was utilized. Jasmine is a behavior-driven development (BDD) framework for testing JavaScript code [64]. In order to run the unit tests automatically on changing code, Karma was used. Karma spawns a web server that executes source code against the unit tests for each of the browsers connected [65].

End-to-end testing is used to test the high-level functionality of the code. End-to-end tests are often used to test entire use case scenarios, such as logging-in or entering data. In Genesis, we used Protractor for end-to-end testing. Protractor is an end-to-end test runner, which simulates user interactions helping to verify the functionality of an Angular application [66]. Some of the existing end-to-end tests and their results are shown in Table 9.

Table 9. Example list of existing end-to-end tests.

Test description	Result
User enters the website and is taken to the login page. User writes incorrect credentials and clicks login button. Invalid login message should be shown and user should stay on login page. After entering the correct credentials and click the login button, valid login message should be shown and invalid login message should be hidden. Afterwards, the user should be taken to the dashboard.	PASS
When a user is on the dashboard, the dashboard button class should be active. When the user clicks tasks, the dashboard button's active class should be removed and the tasks button should now have an active class. In addition, the current URL route should reflect the tasks state.	PASS
When a user tries to submit a CRF with missing required fields, an error message should show up. If there are no missing fields, a success modal should pop-up and the user should be taken to the protocol state. On clicking the ok button, the modal should disappear.	PASS
When users click either logout button, they should be taken to login.	PASS

6.2. Usability testing

Genesis's usability testing was conducted in collaboration with the research nurses of DIPP-Study in Oulu. Users at the other study sites are likely to have a similar background and experience in healthcare, which is why our test users were considered likely to simulate the users at other sites. In addition to testing the software with the actual users, hallway testing was also utilized. In hallway testing, the test users are random people who have no prior knowledge of the software [67]. Hallway testing helped us in identifying any flaws in the flow of the G6denesis UI.

Studies have shown that the best and the most economical way to conduct usability testing is to do plenty of short testing iterations and to test the UI with no more than five people [68]. This is because the chance of finding a new UI flaw reduces as the number of test users grows [68]. For the enumerated reasons, each evaluation round of a new version will purposefully be kept short and will not be tested with more users. On each iteration, the user interface will be improved according to the user feedback.

Use case scenarios, which the users were requested to complete on the first testing iteration are described in Table 10 below. Full use case descriptions can be found in Appendix 9. These use case scenarios were chosen, because they were regarded as most important for the functionality of the first version.

Table 10. Use case scenarios that were tested with the users.

#	Scenario description
1	User enters subject data to a CRF
2	User finds a previously stored CRF
3	User validates a CRF
4	User responds to a CRF query
5	User validates a queried CRF
6	User changes password
7	User creates an appointment for subject

After completing all of the scenarios, each user completed a questionnaire. In addition to the questionnaire, the users were interviewed to elicit any other suggestions for improvement. Each evaluation session lasted roughly 40 minutes.

6.2.1. Usability questionnaire results

For the usability testing, we decided to deploy the SUS questionnaire. SUS stands for system usability scale, and is a standardized measuring tool to easily rate the usability of a system [69]. It has been referenced in over 600 publications, thus validating its efficiency. In the 10 item questionnaire below, the score contribution of odd numbered questions runs from 0 to 4. For the even numbered questions, the point contributions run from 4 to 0 [69]. All of the points are added together and then multiplied by 2.5 to get a score ranging from 0 to 100 [69]. This score does not reflect a percentile, but rather it can be used to calculate the actual percentile. The questions and the results of the first usability testing iteration can be seen in Table 10.

Table 11. Distribution of the SUS questionnaire answers with five users.

#	Statement	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Points
1	I think that I would like to use this system frequently.				3	2	17
2	I found the system unnecessarily complex.	1	3	1			15
3	I thought the system was easy to use.				3	2	17
4	I think that I would need the support of a technical person to be able to use this system.	2	1		2		13
5	I found the various functions in the system were well integrated.				5		15
6	I thought there was too much inconsistency in this system.	2	2	1			16
7	I would imagine that most people would learn to use this system very quickly.				4	1	16
8	I found the system very cumbersome to use.	2	3				17
9	I felt very confident using the system.		1	1	3		12
10	I needed to learn lots things before I could get going with this system.	1	3	1			15
Total points = 153, Point Average = $153 / 5 = \mathbf{30.6}$ Total score = $30.6 * 2.5 = \mathbf{76.5}$							

500 studies have indicated that the average SUS score is 68 [70]. According to the distribution of the SUS results, Genesis is placed in the top 25 percentile. In order to place in the top 10 percentile, a score of 80.3 or above would be needed. Only one of the test users had some previous knowledge of using Genesis. Therefore, the usability score will likely increase in the future as the users learn to use the software.

6.2.2. User feedback

In the interviews it was found that the users really liked the UI and considered it aesthetic and modern. All of the users stated that the color choices were well-thought-out and that the UI was not too noisy. Despite the UI itself looking good, many of the test users raised the question of why the software was in English. These users indicated that the language barrier was the most detrimental issue to the usability of the software. Furthermore, the same users also stated that they would have rated the usability higher in the survey if the UI had been in their native language. This illustrates our presupposition that localization might be required for the best user experience.

When we asked the users, whether they would use email or the software for communicating, it was found that the users prefer their email over any software solutions. Users explained that this was mainly because they use their email for other work related communication, and therefore using other mediums would likely result in confusion. On this observation, the implementation of the messaging functionality is low on our priority list.

One user asked if they could link queries to specific fields in the future. This would allow for the monitor to submit field-specific queries. With field-specific queries, we could, for example, highlight the queried fields with a red color. Highlighting would make it easier for the users to recognize the queried fields. We regarded that this feature would add value to the software, which is why it will be implemented in the future.

One of the most important pieces of feedback that we received was related to the dashboard. When we asked the users what they would like to see on their dashboard, the users told us that they would like to have direct links to their daily tasks. In addition, they would like to see any actions that they have taken recently. Because of space limitations, the dashboard cannot be made to conform to the needs of all of the user roles simultaneously. Therefore, we could perhaps make the dashboard change dynamically depending on the role of the user. By doing this, we can show information most relevant for a specific user role.

The most prominent usability aspect that we were interested in was the success of the CRF design. In the interviews, all of the users indicated that the CRFs were easy to complete. Despite using all of the possible data collection methods, users did not find any of the collection methods hard to comprehend. In the LIRA-Study, regular users will spend most of their time entering and verifying CRFs, which is why it was significant that we succeeded in designing user-friendly CRFs.

6.2.3. Observations during usability testing

All of the users started the completion of the scenarios from the login screen. None of the users had issues in logging in or logging out of the system. When the users arrived to the dashboard, they were told to complete a CRF for a subject. Some of the users had initial difficulties in figuring out where to go. However, without any outside help, after roughly 30 seconds, even the last user was able to move into the subject's protocol state. Only one of the five users used tasks to move into the protocol state while all of the other users found the subject from the subjects state. No one used the search bar to find the subject. When asked, the users indicated that they did not see the search bar, which is why it needs to be made more visible.

When the users entered the protocol state, all of them were easily able to spot the visit. However, three of the users did not initially realize how the accordion widget on the protocol functioned. Some of them clicked on the visit date closing and reopening the visit a couple of times. Despite this, all were eventually able to move into the CRF state without help.

In the data entry scenario, while completing the fields, two of the users clicked the date icon assuming that it was a drop down calendar widget. This indicates that some of the users associate the date icon with a calendar widget. In order to improve the intuitiveness of Genesis, we probably need to implement this functionality. It was also noticed that two of the users attempted to write dates without separators. After seeing the error message however, they added the separators. On this observation the system needs to be altered to support this way of entering the date as well.

Aside from the date input, three of the users made an entry error during the CRF completion scenario. On two of the occasions, the users were immediately able to see the error message and fix the input accordingly. This indicates that the error messages are visible and work as intended. One of the users entered a different height value than what was told in the scenario. The validation did not fire on this instance. This is because the entry itself was in the correct format, but the entered value was insensible. This indicates that the users can currently key in erroneous data that cannot be verified in real-time. In order to ameliorate this, we need to develop an automated server-side validation, where similar subjects are compared to each other in order to find any data anomalies.

When entering the array elements, two of the users could not immediately differentiate between the different levels of an array. To illustrate, when we asked the user to add a medication for a medical condition, they added a new medical condition instead. However, both of the users noticed their error and were able to spot the delete button at the top-right corner of the array widget. To address this problem, we should probably use a different button color for adding an inner array item.

It was noticed during testing that each user followed the entry guidelines given in descriptions underneath the fields. This indicates that these descriptions play a huge role on how the users enter the data, which is why effort needs to be put into formulating both concise and unambiguous entry descriptions.

After having finished entering all of the fields, all users were easily able find the submit button and submit the CRF. After submitting the CRF, none of the users had problems logging out of the system.

Overall, all of the users were able to complete all of the use case scenarios without outside help. During testing, it was noticed that two of the users often wanted a confirmation that what they had done was correct. These users also happened to be the oldest of the testers. The fact that the users required confirmation indicates that the users were somewhat unconfident using the software, which was also indicated by the SUS questionnaire. However, it might also denote that there are differences in the use behavior between various cohorts.

6.3. Customer satisfaction

After testing the usability and the functionality of the client, a customer satisfaction survey was also given to the customer who ordered the project. The purpose of this survey was to determine whether the customer was content with the first version and the development process so far. The survey questions and its results can be seen in Table 12.

Table 12. Customer satisfaction survey results.

#	Statement	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
1	The existing feature set fulfills the needs of this trial.				X	
2	I am happy with user interface of Genesis.					X
3	The invested time and resources in the development of Genesis have been worth it.				X	
4	I am happy with the results of the usability testing.				X	
5	The existing functionalities have been implemented well				X	
6	The underlying design of Genesis has been thought out well.				X	
7	I am satisfied with the current version.				X	
8	I would like to deploy Genesis in other trials.				X	
9	The development of the software has been agile.				X	
10	I see a lot of potential in Genesis.					X

7. DISCUSSION

This project had two major goals, the first and primary goal of this work was to design and implement a client for a REST-architecture based clinical data management system that would suit the needs of the LIRA-Study. The secondary goal for the developed software was to be easily portable to other studies. This chapter discusses how well these two goals were achieved. In addition, this chapter reflects on the development process, and discusses the future prospects of Genesis.

7.1. Attainment of the primary goal

The primary goal of Genesis was to serve the data management and collection purposes of the LIRA-Study. Roughly 6 months since the commencement of the project, all of the client-side functionalities that were considered vital for the operation of the LIRA-Study have been implemented and evaluated. These functionalities included the entry and verification of CRFs, as well as scheduling and managing appointments. There are still components missing, but the trial is not hindered by the lack of them.

Fulfillment of the required functionalities was evaluated with real users. Testing results demonstrated that all of the required workflows could be completed effortlessly. In addition to being functional, the conducted usability testing indicated that the first version of the web client offered an excellent user experience. Our discussions with the trial staff, prior to development, had revealed that users found some of the existing HIT technology to be unfriendly. On this observation, we decided to focus heavily on usability in this project. Despite doing relatively well on the usability aspect, there is always room for improvement.

Initially, the customer would have liked the software to of have been online and running in 5 months. We considered this time schedule to be extremely rigorous considering the large scope of the project. Despite this, we estimated that, if necessary, we would have been able to provide an operational version with minimal functionality by this deadline. However, mid development, the LIRA-Study was postponed by four months due to bureaucratic reasons, thus giving us more time for development. Given the extra time, we decided to spend more time on designing instead of rushing the implementation of the software. In our design, we aimed to ensure the extensibility and the portability of the software, as well as find ways to automate manually conducted trial processes.

In order to affirm the attainment of the primary goal, we also measured the satisfaction of the primary customer through a survey. The results of the survey indicated that the customer was very content with the development process so far. The customer was especially happy that she could also deploy Genesis in her future trials. Due to the portability, Genesis is expected to offer a lot of utility and financial savings to the primary customer in the long run. In addition to the system itself, the customer also benefitted immensely from our investigation of the existing clinical data management standards. Our acquired knowledge of the CDISC and data management standards was capitalized in designing the LIRA-Study.

7.2. Attainment of the secondary goal

The secondary, yet perhaps the most prominent goal of this project was to be easily portable to other studies. This requirement was not directly espoused by the customer, but we regarded that the portability of the CDMS would offer the most utility in the long run. This is why in our design we focused heavily on finding solutions that would universally suit the needs of any given study.

Portability of Genesis is achieved in multiple ways. The first and the most important factor in considering the portability of Genesis is the data model design. This thesis did not discuss the underlying data model in great detail. This was mainly because its implementation was mostly the concern of the backend developer. However, the web client has been designed to work in accordance to the underlying data model. The second way Genesis's portability is ensured, is by using JSON schemas as blueprints to create the CRFs. By deploying JSON schemas, web-forms do not need to be hard-coded on the client side. Third, Genesis is based on REST architecture, which renders the software scalable, allowing for us to serve massive amounts of users at once. In addition, REST architecture allows for Genesis to be deployed on mobile devices such as phones and tablets. There have already been inquiries to deploy Genesis on a tablet for the purpose of off-site recruiting. Fourth, Genesis has been built to be compatible with CDISC standards. The use of these standards greatly enhances the quality of the entered data. Moreover, we have used CDISC guidelines to pre-build a list of CRFs so that a given study can easily pick which ones to utilize in their study. This is expected to cut down the time required by the database planning phase of a trial.

In order for Genesis to be easily portable to other studies, it should be able to be deployed by a person with little-to-no information technology experience. As we initially assumed, we did not have enough time to implement a trial builder and managing tool. However, Genesis has been designed with the idea that a GUI based tool will later be designed and implemented to allow for the studies to be created and managed by regular users. In this tool, the user will be able to initialize a trial, plan protocols, drag and drop variables into CRFs, design validation rules for variables, and much more. When this functionality is completed, Genesis can be expanded rapidly into other studies and its full potential can be realized. To summarize, while the goal of Genesis being easily portable has not yet been achieved, we regard ourselves as being successful in designing and implementing the underlying framework that will allow this ambitious goal to be achieved.

7.3. Reflecting on development

In the initial stages of development, a lot of research had to be done in order to understand the current state of web development. In addition, we had to explore the many existing technology options and choose the best ones that would allow for both rapid development and longevity of the software. I regard that we succeeded in our choices thanks to the performed research.

One prominent decision, which had to be made on the client-side, was to choose the JavaScript framework. As previously shown, AngularJS was chosen as the JavaScript framework for Genesis development. In hindsight, this choice proved itself to be extremely successful. The strong community around AngularJS has given rise to a broad selection of MIT licensed modules that we could relatively effortlessly

deploy in Genesis. These modules helped immensely in cutting down the development time of the web client. While using third-party add-ons helped to shear off development time, it can lead to unnoticed bugs that are induced by the modules. In order to ascertain that the modules were compatible with Genesis, time had to be spent on reviewing the source code of the modules. In the future, if we wish to upgrade the module versions, we need to re-verify that the newer version remains compatible with our existing software.

In the implementation phase, time should have been spent on writing automated tests, instead of rapidly adding on new pieces of functionality. Automated tests were not written in the initial stages of development, which resulted in time having to be spent on manual testing. By using automated tests, time will be saved in the long run, as the functionality of the software does not need to be manually retested. Therefore, the plan is to extend the currently scant code coverage of Genesis, and to focus on writing more unit and end-to-end tests in the future. This is especially vital for the extensibility of the software, as the larger the functionality set becomes, the more manual testing would need to be done.

In addition to writing automated tests earlier, we should have started to do user testing earlier. The idea was to start testing the UI once we had the first functional version of the web client ready. However, this could have resulted in a lot of excess work, if the first design would have been unusable. In that sense, we were lucky that the design that was deployed ended up being liked by the users.

Another issue that should have been taken into consideration in the web client development was browser JavaScript compatibility. Older browsers do not support all of the newer JavaScript functions that have been used in Genesis. Browser compatibility will not be a problem in the LIRA-Study, as all of the users are using modern browsers. However, if we wish to extend Genesis's compatibility to older browsers, some of the functions will need to be rewritten.

7.4. Genesis compared to existing solutions

While the current version of Genesis does not offer any groundbreaking ideas that would revolutionize the CDMS industry, it fulfills the primary needs of the LIRA-Study. Furthermore, Genesis still lacks the broad feature set that the existing commercial CDMS systems such as OpenClinica or Oracle InForm possess. Most of the existing systems have been under development for many years, which is why surmounting them in number of features is likely to be an impossible task.

Despite lacking a broad feature set, Genesis excels in conforming to the specific needs of the LIRA-Study. Ready-made commercial CDMS solutions often strive to cater to large audiences. On the contrary, by developing Genesis specifically for the LIRA-Study, we were able to take user-specific needs into consideration, and thus generate an excellent user experience. This was proven by the conducted usability testing. Moreover, before Genesis, we had already developed an in-house solution for the same user group, which allowed us to leverage the already established trust between the users and us.

Overall, because Genesis is currently being developed by just two developers, it is unlikely that we would be able to compete with the largest CDMS providers in the near future. Despite this, we can strive to excel in being chameleon-like and adapt swiftly to different trials.

7.5. Future prospects of the developed software

While the first version of Genesis's web client could be considered successful, it by no means is the final version. Our next step is to keep iterating and polishing the existing version until the commencement of the LIRA-Study. Afterwards, we will start to gradually implement the missing features. All of the functionalities can be deployed on the system as components without interfering with the existing functionality, thus reducing the chance of regression bugs.

One prominent component that we did not have time to engineer yet is a GUI based trial management tool. This trial management tool will be a separate module that can be used to create and manage trials, protocols, visits, CRFs, and variables. The goal for this tool is to be highly user friendly so that Genesis could be deployed in any research without the help of IT professionals. The need for this tool was recognized early in the design phase, which is why the current version is engineered to be compatible with the to-be-developed tool.

Another plan of ours is to utilize the metadata acquired from the LIRA-Study to analyze the completion times of CRFs. The insight gained from the analysis will be used to estimate the completion time of any CRF. Because visits are formed of CRFs, we can further estimate the entry time for a trial visit. Moreover, by knowing the total amount of visits within the trial, we can estimate the labor hours that need to be invested into entering the data. Labor hours can be directly used by the prime investigators to estimate the costs of data entry. Moreover, we can use the CRF completion metadata to analyze the correlation between CRF length and completion time. This data can later be used for optimizing the user experience. Another method that we plan to utilize in order to improve user experience is to start collecting metadata on the use of the UI elements. If the metadata indicates that some of the UI elements are not being used, we will remove them to reduce the cognitive load.

We also plan to improve the validation of CRFs by comparing the data of subjects with each other. By doing this, we can spot any anomalies in the dataset. This modification will help to enhance the quality of the data. While the automatic validation can be used to prevent most human mistakes, the users can still induce errors that our in-built validation safety net does not cover. To illustrate, a data entry clerk could enter data for a wrong subject. In order to resolve a problematic situation such as the one described above, help would be needed from an IT professional. In the future, we need to generate ways for the users to easily recover from these kinds of situations themselves.

Due to the rigorous schedule of the project, remote testing had not initially been one of our plans. Given the extra time however, we plan to do some remote testing with the research nurses from other study sites. This will likely give rise to a completely new set of ideas, as the research nurses at the other sites have used different pieces of software compared to the nurses that had originally tested the software. In addition to doing more testing with the users, the current code coverage of the automated tests is scant and needs to be greatly improved.

As aforementioned, our current technology choices should be long-lasting. Despite this, we need to keep our eyes and ears open, as there are various interesting changes that will occur in the field of web development in the near future. One such change is the release of AngularJS 2.0 [36]. The new version of the framework is going to bring about radical changes. To illustrate, AngularJS 2.0 is focused on optimizing performance for mobile devices rather than web clients [36]. In addition, some of the

modules of the existing version will be removed from the core in order to improve performance [36]. Because our goal is to serve Genesis on mobile devices, we might want to consider transitioning to the new framework when it becomes more well-established. Further, in the coming years, current HTTP-based applications will likely begin to gradually migrate to HTTP/2 [71]. Compared to HTTP, HTTP/2 offers various new features that result in faster overall load times and improved information security [71][72]. One major change of HTTP/2 is that the format of data transmission is now in binary rather than text [71]. While the text-based protocol is certainly more intuitive to developers and better for debugging, the binary format allows for faster computer processing [73]. Despite the changes, the new HTTP/2 protocol is out-of-the-box compatible with existing API's [73]. Therefore, adopting HTTP/2 on Genesis should not induce any problems and should be looked into if we later deem that a performance boost is necessary.

As it was mentioned earlier in this thesis, we are currently living in an era where computers are actively being deployed virtually everywhere. Genesis is a product of this era as well as a part of a broad movement that strives to computerize data collection and management of studies. In our study site alone, we have already introduced and proved the business case for the use of information technology in clinical studies. Once we deploy Genesis in other trials, it is to be expected that we will also further fortify the paradigm that IT is a beneficial tool that can be leveraged to conduct better science.

Discussions have already been had to deploy Genesis in two other large-scale trials. One of these trials is the world's largest type 1 diabetes study. If Genesis shows promise in the LIRA-Study and the other two trials, it will be commercialized and marketed to other trials as well. As it was illustrated in chapter 2, the market for CDMS systems is still quite unsaturated. In addition to the plans to commercialize Genesis, we will publish a paper on Genesis in either a technology or healthcare related journal. Further, Genesis will be referenced in future publications that are based on the trials that run on Genesis.

8. CONCLUSION

In this thesis, both the design and the implementation process of Genesis's web client were illustrated. In addition, the current state of clinical data management and web development were mapped out. As the result of this thesis, the first version of the clinical data management system was released.

This thesis started with the study of clinical data management's history and current state. Thereafter, some of the common challenges in developing health information technology were investigated. After reviewing the current state of clinical data management, both the functional and nonfunctional requirements of the developed system were depicted. In addition to the requirements, the requirement elicitation process and methods were also discussed. The benefits of each requirement collection method were also considered. The gathered requirements partially guided the development method and tool selections, which were also illustrated in this thesis. The selection process, and the justification for the made choices were also presented.

Based on the requirements and the selected tools, the design for the web client was created. The design was illustrated in the design and implementation chapter, which started with the representation of the underlying system architecture. Genesis's user interface design and flow were additionally portrayed. Further, the means to ensure client-side information security were discussed as well.

After developing the first version of the web client, it was evaluated with real users who were both interviewed and given a questionnaire. The acquired feedback was used to gauge the quality and usability of the application, as well as for making changes for the next version. In addition to testing the usability, satisfaction of the customer who ordered the work was also evaluated.

Finally, the results of the project were evaluated. In addition to analyzing the achieved results, future prospects and extensibility of Genesis were also contemplated.

9. REFERENCES

- [1] Pal A. (2014) Data Capture Challenges in Clinical Trials. URL: <http://thedoctorweighsin.com/data-capture-challenges-in-clinical-trials/>. Accessed on 9.4.2015.
- [2] Survey Reveals Technology Gap in Managing Clinical Trial Documentation. URL: <http://www.centerwatch.com/news-online/article/5444/survey-reveals-large-technology-gap-in-managing-clinical-trial-documentation#sthash.T0A2awxk.dpbs>. Accessed on 9.4.2015.
- [3] Bhattacharjee A & Hikmet N. (2007) Physicians' Resistance toward Healthcare Information Technologies: A Dual-Factor Model. In 40th Annual Hawaii International Conference on System Sciences, 2007. HICSS 2007. January 3-6. DOI: <http://dx.doi.org/10.1109/HICSS.2007.437>.
- [4] Seckman C. (2008) Clinicians' Perceptions of Usability of an Electronic Medical Record Over Time. DOI: <http://dx.doi.org/10.1097/01.NCN.0000360477.08025.80>.
- [5] Lu Z & Su J. (2010) Clinical data management: Current status, challenges, and future directions from industry perspectives. Dove Press Journal, p. 93-105. DOI: <http://dx.doi.org/10.4103%2F0253-7613.93842>.
- [6] Hammond W. (1987) Patient Management Systems: The Early Years. In HMI '87 Proceedings of ACM conference on History of medical informatics. ACM, p. 153-164. DOI: <http://dx.doi.org/10.1145/41526.41541>.
- [7] Jackson G. (1969) Information handling costs in hospitals. *Datamation* 15:56, p. 15-56.
- [8] Babre D. (2011) Electronic data capture - Narrowing the gap between clinical and data management. *Perspectives in Clinical Research* 2011;2:1-3. DOI: <http://dx.doi.org/10.4103%2F2229-3485.76282>.
- [9] CDISC | Strength Through Collaboration. URL: <http://www.cdisc.org/>. Accessed on 9.4.2015.
- [10] State of Trial Master Files in 2013. NextDocs. URL: <https://rammellconsulting.files.wordpress.com/2013/10/rpe-state-of-trial-master-files-oct-2013.pdf>. Accessed on 9.4.2015.
- [11] Rouf E, Whittle J, Lu N & et al. (2007) Computers in the Exam Room: Differences in Physician–Patient Interaction May Be Due to Physician Experience. *J Gen Intern Med.* 2007 January 22(1): p. 43–48. DOI: <http://dx.doi.org/10.1007%2Fs11606-007-0112-9>

- [12] Irani J, Middleton J, Marfatia R & et al. (2009) The use of electronic health records in the exam room and patient satisfaction: a systematic review. *J Am Board Fam Med* 2009;22: p. 553–562. DOI: dx.doi.org/10.3122/jabfm.2009.05.080259.
- [13] Huong N. (2012) *Clinical Data Management (Process and practical guide)*. URL: <http://www.gfmer.ch/SRH-Course-2011/Geneva-Workshop/pdf/Clinical-data-management-Huong-2012.pdf>. Accessed on 9.4.2015.
- [14] Kang H, Sirintrapun J, Nestler R & et al. (2010) Experience With Voice Recognition in Surgical Pathology at a Large Academic Multi-Institutional Center. *American Journal of Clinical Pathology*. 2010 Jan;133(1):p. 156- DOI: dx.doi.org/10.1309/AJCPOI5F1LPSLZKP.
- [15] Wahi M, Parks D, Skeate R & et al. (2008) Reducing Errors from the Electronic Transcription of Data Collected on Paper Forms: A Research Data Case Study. *Journal of the American Medical Informatics Association*. Vol 15. p. 386–389. DOI: <http://dx.doi.org/10.1197%2Fjamia.M2381>.
- [16] Elodia C, Pisano E, Clary G & et al. (2005) A comparative study of mobile electronic data entry systems for clinical trials data collection. *International Journal of Medical Informatics*. Vol. 75(10-11). p. 722-729. DOI: <http://dx.doi.org/10.1016/j.ijmedinf.2005.10.007>.
- [17] Rossi B. (2014) Educating the end user and eliminating the biggest security risk. *Information Age*. URL: <http://www.information-age.com/technology/security/123458150/educating-end-user-and-eliminating-biggest-security-risk>. Accessed on 9.4.2015.
- [18] Clinovo – cloud based eClinical software. URL: <http://www.clinovo.com/>. Accessed on 9.4.2015.
- [19] Ulinski D. (2013) *Computerized Physician Order Entry: Reluctance of Physician Adoption of Technology Linked to Improving Health Care*. Dissertation. Walden University.
- [20] The Kurt Lewin Model of Change. URL: http://www.change-management-coach.com/kurt_lewin.html. Accessed on 9.4.2015.
- [21] Krohn R. (2003) In search of the ROI from CPOE. *Journal of healthcare information management*, Vol. 17(4), p. 6-9.
- [22] Thompson C. (2013) *Benefits and risks of Electronic Medical Record (EMR) An interpretive analysis of healthcare consumers' perceptions of an evolving health information systems technology*. Dissertation. Robert Morris University.
- [23] Barrows R. (1996) Privacy, Confidentiality, and Electronic Medical Records. *Journal of the American Medical Informatics Association*, Vol 3(2), p. 139-148. DOI: <http://dx.doi.org/10.1136/jamia.1996.96236282>.

- [24] Vision for MedDRA – MedDRA. URL: <http://www.meddra.org/about-meddra/vision>. Accessed on 9.4.2015.
- [25] Wiegers K, Beatty J. (2013) Software Requirements.
- [26] Schwaber C, Leganza G & Daniels M. (2006) The Root Of The Problem: Poor Requirements.
- [27] Lee J & Xue N. (1999) Analyzing User Requirements by Use Cases: A Goal-Driven Approach. Software, IEEE Vol 16(4). p. 92-101. DOI: <http://dx.doi.org/10.1109/52.776956>.
- [28] What is Agile model – advantages, disadvantages and when to use it?. URL: <http://istqbexamcertification.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/>. Accessed on 9.4.2015.
- [29] Manifesto for Agile Software Development. URL: <http://agilemanifesto.org>. Accessed on 10.4.2015.
- [30] Agile Methodology – Agile Software Development – versionOne. URL: <http://www.versionone.com/agile-101/agile-development-methodologies-scrum-kanban-lean-xp/>. Accessed on 10.4.2015.
- [31] Dale T. (2015) You're Missing the Point of Server-Side Rendered JavaScript Apps. URL: <http://tomdale.net/2015/02/youre-missing-the-point-of-server-side-rendered-javascript-apps/>. Accessed on 10.4.2015.
- [32] Fabric V. (2014) Server vs client side rendering (AngularJS vs server side MVC). URL: <http://technologyconversations.com/2014/07/10/server-vs-client-side-rendering-angularjs-vs-server-side-mvc/>. Accessed on 10.4.2015.
- [33] Slatkin B. (2015) Experimentally verified: "Why client-side templating is wrong". URL: <http://www.onebigfluke.com/2015/01/experimentally-verified-why-client-side.html>. Accessed on 10.4.2015.
- [34] Shaked U. (2014) AngularJS vs. Backbone.js vs. Ember.js. URL: <https://www.airpair.com/js/javascript-framework-comparison>. Accessed on 10.4.2015.
- [35] Google Trends – Web search interest: AngularJS, Backbone.js, Ember.js – Worldwide, 2004 – present. URL: <https://www.google.com/trends/explore#q=AngularJS%2C%20Backbone.js%2C%20Ember.js&cmpt=q&tz>. Accessed on 10.4.2015.
- [36] Esteva S. (2014) AngularJS 2.0 Status and Preview. URL: <http://ng-learn.org/2014/03/AngularJS-2-Status-Preview/>. Accessed on 10.4.2015
- [37] Web Services Glossary. URL: <http://www.w3.org/TR/ws-gloss/>. Accessed on 10.4.2015.

- [38] Potti P. (2011) On the Design of Web Services: SOAP vs. REST. UNF Theses and Dissertations.
- [39] Prescod P. (2002) Roots of the REST/SOAP Debate. 2002 Extreme Markup Languages Conference, Montréal, Canada.
- [40] Francia S. (2010) REST vs SOAP, The Difference Between Soap and Rest. URL: <http://spf13.com/post/soap-vs-rest>. Accessed on 10.4.2015.
- [41] Nene A. (2014) Web Services Architecture – When to Use SOAP vs REST URL: <http://java.dzone.com/articles/web-services-architecture>. Accessed on 10.4.2015.
- [42] Davelaar S. (2015) Performance Study – REST vs SOAP for Mobile Applications. URL: <http://www.ateam-oracle.com/performance-study-rest-vs-soap-for-mobile-applications/>. Accessed on 10.4.2015.
- [43] JSON: The Fat-Free Alternative to XML. URL: <http://www.json.org/xml.html>. Accessed on 10.4.2015.
- [44] Laurent S. (1998) Why XML?. URL: <http://www.simonstl.com/articles/whyxml.htm>. Accessed on 10.4.2015.
- [45] Mikoluk K. (2013) JSON vs XML: How JSON Is Superior To XML. URL: <https://blog.udemy.com/json-vs-xml/>. Accessed on 10.4.2015.
- [46] Nurseitov N, Paulson M, Reynolds M, et al. (2009) Comparison of JSON and XML Data Interchange Formats: A Case Study. Scenario, p. 157-162.
- [47] Git – About Version Control. URL: <http://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>. Accessed on 10.4.2015.
- [48] Penzo M. (2006) Label Placement in Forms. URL: <http://www.uxmatters.com/mt/archives/2006/07/label-placement-in-forms.php>. Accessed on 10.4.2015.
- [49] Enders J. (2014) The Definitive Guide to Form Label Positioning. URL: <http://www.sitepoint.com/definitive-guide-form-label-positioning/>. Accessed on 10.4.2015.
- [50] Jovanovic J. (2009) Web Form Validation: Best Practices and Tutorials. URL: <http://www.smashingmagazine.com/2009/07/07/web-form-validation-best-practices-and-tutorials/>. Accessed on 10.4.2015.
- [51] Usability First – Usability Glossary – Affordance. URL: <http://www.usabilityfirst.com/glossary/affordance/>. Accessed on 10.4.2015.
- [52] JSON Schema: core definitions and terminology. URL: <http://json-schema.org/latest/json-schema-core.html>. Accessed on 10.4.2015.

- [53] Using HTTP Methods for RESTful Services. URL: <http://www.restapitutorial.com/lessons/httpmethods.html>. Accessed on 10.4.2015.
- [54] Nilkanth H. (2010) Client-Side Security. Thesis. National Institute of Technology, Karnataka.
- [55] Toomey P. (2010) How NOT to build your client-server security architecture. URL: <http://labs.neohapsis.com/2010/02/26/how-not-to-build-your-client-server-security-architecture/>. Accessed on 10.4.2015.
- [56] HTTP to HTTPS – What is a HTTPS certificate? URL: <https://www.instantssl.com/ssl-certificate-products/https.html>. Accessed on 10.4.2015.
- [57] AngularJS: API Reference. URL: <https://docs.angularjs.org/api>. Accessed on 10.4.2015.
- [58] Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet – OWASP. URL: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet. Accessed on 10.4.2015.
- [59] Same Origin Policy – Web Security. URL: https://www.w3.org/Security/wiki/Same_Origin_Policy. Accessed on 10.4.2015.
- [60] Fu K, Sit E, Smith K & et al. (2001) Dos and Don'ts of Client Authentication on the Web. In Proceedings of the 10th USENIX Security Symposium, Washington, D.C. August 2001.
- [61] AlFayyadh B, Throsheim P, A Jøsang & et al. (2012) Improving usability of password management with standardized password policies. In 7th Conference on Network and Information Systems Security.
- [62] Pose A. (2014) Cookies vs Tokens. Getting auth right with AngularJS. URL: <https://auth0.com/blog/2014/01/07/angularjs-authentication-with-cookies-vs-token/>. Accessed on 10.4.2015.
- [63] Testing in the Software Lifecycle. URL: <https://msdn.microsoft.com/en-us/library/jj159342.aspx>. Accessed on 20.4.2015.
- [64] Jasmine: Behavior-Driven JavaScript. URL: <http://jasmine.github.io/>. Accessed on 20.4.2015.
- [65] Karma – How It Works. URL: <http://karma-runner.github.io/0.12/intro/how-it-works.html>. Accessed on 20.4.2015.
- [66] Protractor – end to end testing for AngularJS. URL: <http://angular.github.io/protractor/#/>. Accessed on 20.4.2015.
- [67] OpenHallWay Blog – Blog Archive – What is Hallway Usability Testing? URL: <http://blog.openhallway.com/?p=146>. Accessed on 6.5.2015.

- [68] Nielsen J. (2000) Why You Only Need to Test with 5 Users. URL: <http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>. Accessed on 10.4.2015.
- [69] System Usability Scale | Usability.gov (SUS). URL: <http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>. Accessed on 20.4.2015.
- [70] Sauro J. (2011) Measuring Usability with the System Usability Scale (SUS). URL: <http://www.measuringu.com/sus.php>. Accessed on 20.4.2015.
- [71] HTTP/2. URL: <http://http2.github.io/>. Accessed on: 8.5.2015.
- [72] Orsini L. (2015) Everything You Need To Know About HTTP2. URL: <http://readwrite.com/2015/02/18/http-update-http2-what-you-need-to-know>. Accessed on: 8.5.2015
- [73] Sidey T. (2015) What REST API Services Can Expect from HTTP/2? URL: <https://www.api2cart.com/blog/rest-api-services-can-expect-http2/>. Accessed on: 8.5.2015.

10. APPENDICES

- Appendix 1. Login state
- Appendix 2. Dashboard state
- Appendix 3. Account state
- Appendix 4. Subjects state
- Appendix 5. Task state
- Appendix 6. Message state
- Appendix 7. Calendar state
- Appendix 8. Trial overview state
- Appendix 9. Use case scenarios

Appendix 1. Login state.

Sign In

Username


Password

[Login](#)

[Forgot your password? Retrieve password](#)

Appendix 2. Dashboard state.

LIRA-Study Log out




Welcome, **jorahko**
Administrator

- [My Account](#)
- [Change Password](#)
- [Log out](#)

Dashboard


- [Dashboard](#)
- [My Account](#)
- [Trial Overview](#)
- [Tasks](#)
- [Subjects](#)
- [Appointments](#)
- [Messages](#)

Dashboard



New messages! **3**


[View Messages](#)



Tasks! **12**

[View Tasks](#)

Notifications

 Scheduled server maintenance 12.12.2013 at 4:30pm 4 minutes ago

Study Progression

Trial 1

72% Complete

Appendix 3. Account state.

LIRA-Study Log out

Welcome, jorahko
Administrator

[My Account](#)
[Change Password](#)
[Log out](#)

- [Dashboard](#)
- [My Account](#)
- [Trial Overview](#)
- [Tasks](#)
- [Subjects](#)
- [Appointments](#)
- [Messages](#)

Search for a subject...

My account

My Account

Account Information

Username	jorahko
Name	Joona Rahko
E-mail	jrahko@gmail.com
Account created	24.01.2015
Role	Administrator
Location	Oulu
Phone Number	040-777-9970

[Change contact details](#)
[Change password](#)

Appendix 4. Subjects state.

LIRA-Study Log out

Welcome, jorahko
Administrator

[My Account](#)
[Change Password](#)
[Log out](#)

- [Dashboard](#)
- [My Account](#)
- [Trial Overview](#)
- [Tasks](#)
- [Subjects](#)
- [Appointments](#)
- [Messages](#)

Search for a subject...

Subjects

Subjects

+ Add a subject

Search for a subject:

Subjects (25)

Subject Code	Date of Birth	Previous Visit	Status
LIRA_001_001	01.01.1990	01.01.1990	ONGOING
LIRA_001_002	01.01.1990	01.01.1990	ONGOING
LIRA_001_003	01.01.1990	01.01.1990	ONGOING
LIRA_001_004	01.01.1990	01.01.1990	ONGOING
LIRA_001_005	01.01.1990	01.01.1990	ONGOING

Appendix 5. Tasks state.

LIRA-Study Log out

Welcome, Jorahko
Administrator
[My Account](#)
[Change Password](#)
[Log out](#)

Dashboard
 My Account
 Trial Overview
Tasks
 Subjects
 Appointments
 Messages

Search for a subject...

Tasks

Search from tasks:

Show task type
 Entry Query Schedule Verification

Show task type
 Complete Incomplete

Tasks (20)

Type	Status	Date	Details
Query task	Incomplete	17.4.2015	Answer a query for CRF 'HbA1C'
Entry task	Incomplete	17.4.2015	Enter data for 'LIRA_001_002'
Schedule task	Incomplete	17.4.2015	Schedule visit for 'LIRA_002_014'
Verification task	Incomplete	18.4.2015	Verify CRF 'Measurements' for 'LIRA_002_014'

Appendix 6. Messages state.

LIRA-Study Log out

Welcome, Jorahko
Administrator
[My Account](#)
[Change Password](#)
[Log out](#)

Dashboard
 My Account
 Trial Overview
 Tasks
 Subjects
 Appointments
Messages

Search for a subject...

Messages

[+ Compose a message](#)

Search:

Inbox (19) **Sent (16)**

Sent By	Title	Date Received
Riitta	Hoitajapalaute Hoitajilta oli tullut hyvää palautetta...	14.04.2015
Olli	Test Testi viesti...	14.04.2015
Riitta	Testataan viestitystä Testataan viestitystä...	14.04.2015
Olli	Super message	13.04.2015

Appendix 7. Calendar state.

LIRA-Study Log out

Welcome, jorahko
Administrator

- My Account
- Change Password
- Log out

- Dashboard
- My Account
- Trial Overview
- Tasks
- Subjects
- Appointments
- Messages

Search for a subject...

Appointments

Appointments

+ Schedule an appointment

Search from appointments:

Search for an appointment

Show:

Upcoming
 Complete
 Postponed
 Cancelled

Appointments (36)

Status	Meeting Date	Subject	Location	Start Time	Duration
<input type="button" value="Upcoming"/>	2012-01-20	LIRA_001_001	Oulu	11:46	1 h 11 min
<input type="button" value="Upcoming"/>	2012-01-20	LIRA_001_001	Oulu	11:46	1 h 11 min
<input type="button" value="Upcoming"/>	2012-01-20	LIRA_001_001	Oulu	11:46	1 h 11 min

Appendix 8. Trial overview state.

LIRA-Study Log out

Welcome, jorahko
Administrator

- My Account
- Change Password
- Log out

- Dashboard
- My Account
- Trial Overview
- Tasks
- Subjects
- Appointments
- Messages

Search for a subject...

Trial Overview

Trial Overview

Trial All

Study site All

Subject Search for a subject...

Trial Progression

Trial 1 72% Complete

Tasks	147,841	Unresolved Queries	171
Tasks Completed	115,982	Unverified CRFs	52
Tasks Remaining	32,982		

Trial Statistics

Total Subjects	1,713
----------------	-------

Appendix 9. Use case scenarios.

All use case scenarios start from the log-in screen

Use case scenario 1 – User enters subject data to a CRF

Subject code LIRA_001_002 comes for a regular study visit on 01.01.2014 with his mother. The subject is named Erkki Esimerkki and he has been born on 15.8.2010. You measure his height, and find out that his height is 80.7 cm. After measuring the height, you measure his weight and find out that he weighs 35.4 kg. Lastly you measure the circle of head, which is 49 cm. After entering the measurements, you enter the date of the measurement, which is today. Afterwards, you write, whether Erkki has type 1 diabetes or not. After asking Erkki's mother, he tells you that he does not have diabetes. You also enter Erkki's study site, which is Oulu. Thereafter, you inquire the mother, has Erkki had any medical conditions. Erkki's mother tells you that Erkki has an active Asthma, which was diagnosed in June 2010. Flixotide 250µg and ventoline 100µg are used to treat the condition. Erkki also has celiac disease, which was diagnosed in November 2012, which is active, but is not being treated in any way. Afterwards, you write into the other information field that nothing out of the ordinary occurred during the visit.

Use case scenario 2 – User finds a previously stored CRF

LIRA_001_002 had a visit on 01.02.2014, you decide to view the CRF from that visit.

Use case scenario 3 – User validates a CRF

A data entry clerk has submitted a CRF for LIRA_001_002 for visit 3 that was held on 01.03.2014. Your task is to validate the submitted CRF. While validating the entry, you notice that the name is in format first name + lastname, while it actually should be in format lastname + firstname. You also know that subject LIRA_001_002 study site should be Turku instead of Oulu.

Use case scenario 4 – User responds to a CRF query

A monitor has queried a CRF that you entered for visit 4 that was held on 01.04.2014. Your task is to make the proper fixes and respond to the query.

Use case scenario 5 – User validates a queried CRF

A data entry clerk has responded to a query that was made. Your task is to check the CRF again and make sure that the requested changes have been made.

Use case scenario 6 – User changes password

Your task is to log into the system and change your password. Your old password is password1 and you need to change it into fancypassword2.

Use case scenario 7 – User creates an appointment for a subject

Your task is to create the next visit appointment for subject LIRA_001_002 for date 14.07.2015.