Självständigt arbete på avancerad nivå

*Independent degree project - second cycle*

Datateknik
*Computer Engineering*

**Robust Issue Tracking Application (RITA)**
Developing an issue tracker using modern web technologies

Stefan Åhman
MID SWEDEN UNIVERSITY
The Department of Information Technology and Media (ITM)

Examiner: Mikael Gidlund, mikael.gidlund@miun.se
Supervisor: Stefan Forsström, stefan.forsstrom@miun.se
Author: Stefan Åhman, stam1000@student.miun.se

Degree programme: International Master in Computer Engineering, 120 points
Main field of study: Computer Engineering
Semester, year: 06, 2017
Abstract

Issue tracking is one of the vital parts in maintaining computer systems. It is utilized in anything from small independent companies to large enterprises. The tracking does not just provide developers and other personnel with crucial information regarding their systems current state, but additionally stores useful documentation if any error reoccurs in the future. However, if the tracking issue would be deficient in some way, the complete process of developing or maintaining a system could affected negatively in great extent. This thesis work has looked into a scenario where the tool has been to slow, overly complicated and obsolete. It has made a large negative impact on the work force that uses the tool and made tracking issue to a discouragement. The thesis work has thence investigated features of the existing tool, suggested a better solution to use based upon these findings, followed by the development of a web application. When the application was finished, its usability was tested by the intended staff and performance tests were conducted. The test results showed that the application had been implemented successfully in many aspects. Unfortunately, due to deficient technical choices, the project did struggle with implementing all features as expected. The thesis work did consequently learn the hard way the importance of a choosing development techniques very thoroughly.

Keywords: issue tracker, issue tracking, Javascript, Node.js, AngularJS, EmberJS, Sequelize, ORM, MVC, SPA
Foreword

Firstly, I would like to thank my supervisors Johan Moritz, and Michael Andersson for their highly appreciated guidance and support. I also thank Jonny Nordin for his excellent technical consultation.

Thanks also go to the remaining staff at the CRT division and the CGI office in Sundsvall for bringing good energy and laughter into this project.

Lastly, I would like to express my deepest and sincere gratitude to my parents. My interest in computer science and engineering would not have been if not for them.
Table of Contents

Abstract ................................................................................................................... iii
Foreword .................................................................................................................... iv
Table of Contents .................................................................................................... v

1 Introduction .......................................................................................................... 1
  1.1 Background and problem motivation ........................................................... 1
  1.2 Overall aim .................................................................................................... 2
  1.3 Concrete and verifiable goals ...................................................................... 3
  1.4 Scope ............................................................................................................. 3
  1.5 Outline .......................................................................................................... 3

2 Theory .................................................................................................................. 5
  2.1 Issue tracker ................................................................................................. 5
  2.1.1 Related work ............................................................................................ 6
  2.2 Usability tests .............................................................................................. 7
  2.2.1 Potosnack’s three steps ......................................................................... 7
  2.3 Performance tests ....................................................................................... 8
  2.4 Modern web development ......................................................................... 9
  2.5 Model-View-Controller ............................................................................. 10
  2.5.1 Model ...................................................................................................... 10
  2.5.2 View ....................................................................................................... 10
  2.5.3 Controller ............................................................................................... 11
  2.6 Object-relational mapping ....................................................................... 11
  2.7 Package manager ...................................................................................... 12

3 Methodology ....................................................................................................... 13
  3.1 Investigating features ................................................................................ 13
  3.2 Finding appropriate techniques ................................................................ 13
  3.3 Development ............................................................................................... 14
  3.4 Tests ............................................................................................................. 14
  3.4.1 Usability tests ......................................................................................... 14
  Performance tests .............................................................................................. 15

4 Design ................................................................................................................... 18
  4.1 Issue list and search feature .................................................................... 18
  4.2 Add issue ..................................................................................................... 19
  4.3 Single issue .................................................................................................. 19
Robust Issue Tracking Application
(RITA)
Stefan Åhman

Table of Contents

2017-06-16

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Implementation</td>
<td>21</td>
</tr>
<tr>
<td>5.1 Tools</td>
<td>21</td>
</tr>
<tr>
<td>5.2 Overall architecture</td>
<td>22</td>
</tr>
<tr>
<td>5.3 Front end</td>
<td>22</td>
</tr>
<tr>
<td>5.3.1 Model</td>
<td>23</td>
</tr>
<tr>
<td>5.3.2 Views</td>
<td>23</td>
</tr>
<tr>
<td>5.3.3 Controllers</td>
<td>24</td>
</tr>
<tr>
<td>5.3.4 Additional modules</td>
<td>24</td>
</tr>
<tr>
<td>5.4 Back end</td>
<td>25</td>
</tr>
<tr>
<td>5.4.1 Database</td>
<td>26</td>
</tr>
<tr>
<td>5.4.2 File handling</td>
<td>27</td>
</tr>
<tr>
<td>6 Result</td>
<td>28</td>
</tr>
<tr>
<td>6.1 Investigating features</td>
<td>28</td>
</tr>
<tr>
<td>6.2 Finding appropriate technique</td>
<td>29</td>
</tr>
<tr>
<td>6.3 Development</td>
<td>30</td>
</tr>
<tr>
<td>6.3.1 Issue list and search feature</td>
<td>31</td>
</tr>
<tr>
<td>6.3.2 Add issue</td>
<td>32</td>
</tr>
<tr>
<td>6.3.3 Single issue</td>
<td>33</td>
</tr>
<tr>
<td>6.4 Tests</td>
<td>34</td>
</tr>
<tr>
<td>6.4.1 Usability tests</td>
<td>34</td>
</tr>
<tr>
<td>6.4.2 Performance tests</td>
<td>36</td>
</tr>
<tr>
<td>7 Discussion</td>
<td>38</td>
</tr>
<tr>
<td>7.1 Investigating features</td>
<td>38</td>
</tr>
<tr>
<td>7.2 Finding appropriate techniques and development</td>
<td>38</td>
</tr>
<tr>
<td>7.3 Tests</td>
<td>39</td>
</tr>
<tr>
<td>7.3.1 Usability tests</td>
<td>39</td>
</tr>
<tr>
<td>7.3.2 Performance tests</td>
<td>40</td>
</tr>
<tr>
<td>7.4 Ethical and integrity aspects</td>
<td>40</td>
</tr>
<tr>
<td>7.5 Future work</td>
<td>41</td>
</tr>
<tr>
<td>References</td>
<td>43</td>
</tr>
<tr>
<td>Appendix A: Requirements specification</td>
<td>46</td>
</tr>
<tr>
<td>Mandatory features</td>
<td>46</td>
</tr>
<tr>
<td>Additional preferred features</td>
<td>46</td>
</tr>
<tr>
<td>Appendix B: Usability test scenarios</td>
<td>47</td>
</tr>
</tbody>
</table>
1 Introduction

As the IT society grows rapidly, bigger and more sophisticated computer systems are introduced. These systems must be maintained and even though the ambition is to have as few errors or issues as possible, they will still exist in some degree. When they occur, it is crucial to report and provide solutions for them. This is often accomplished by using an issue tracking tool. These tools many times serves as the key to finding solutions fast and assist in resolving faulty systems with previous knowledge of common problems.

1.1 Background and problem motivation

One of the major telecommunication provider in Swedish is called TeliaSonera, or more commonly just Telia. The company holds a great amount of computer systems that must interplay with each other. One of these systems is called CRT (Customer Recommendation Tool). The tool and its surrounding data is maintained by the IT consulting company CGI, more accurately their dedicated division located in Sundsvall, Sweden.

The CRT division has severe problems with their issue handling. The division is currently using an older issue tracker provided by the client. Unfortunately, the tools general impression is that is far too complicated. For instance, the reporting tool has too many input fields, check boxes and radio buttons to fill in and many fields are unnecessary according to the staff at the division. Another example is the inadequate search feature. It is to slow, complicated and operator-based, meaning that if more than one word is to be searched for, it has to be divided with operator such as “OR”, “AND” et cetera. Due to this fact, the search functions are often avoided completely and the staff scrolls though lists of issues instead.

As expected, these flaws make the tool not far as adequate as the staff would like it to be and has a big negative impact regarding the everyday work flow. Since the tool can’t obtain vital data about an issue in an efficient manor, it causes delays in the error handling. Delays that eventually costs money, due to the time being spent on finding reported issues instead of actually resolve them.
Some staff has even started to avoid the tool completely which means that many of the issues remains completely undocumented. If they remain undocumented, chances are that only one employee knows the solution, making the entire division dependent of a single person. A scenario that leaves the division unaware of critical information if the employee decides to quit or isn’t capable of working. Information that could also make the introduction of new employees more difficult since it puts more responsible on existing employees rather than detailed and easily accessed documentation regarding pre-existing issues.

1.2 Overall aim

This thesis works aims to improve the division’s current situation by providing a new, more efficient solution. The new issue tracking tool is to be called Rapid Issue Tracking Application (RITA). It is meant to facilitate the current struggle of retrieving, composing and expanding issues vital to their everyday activities.

To begin the thesis work, the current tool and the work flow is to be studied in detail to hopefully get a greater understanding of the current situation and what features that the division feels that they are missing in the current tool. This is crucial since the project has a limited time frame and that requires a good insight in what features is required and optional or “nice to have”. Although that ambition is to include as many features as possible.

Based on these findings, along with possible requirements from the division and the research regarding different technical options, a suggestion concerning a new solution will be presented to the staff of the division. If it gets approved, it will directly take the project into the next phase, development. It will have features already present in the current tool as well as new features if sought. The project strives toward providing a solution that lives up to its expectations and helps the division to find a better workflow. To be certain of this, verifications must be made alongside the staff.

When finished, the developed application intends to simplify the issue reporting. Moreover, increase the overall number of issues that is reported with the new tool. With the result of making the division less dependent on certain individuals and allows new staff to educated themselves more rapidly on their own.
1.3 Concrete and verifiable goals

The RITA project pursues four main goals. These goals spans from the very beginning to the end of the projects timeline and will act as milestones throughout it. The following goals are:

1. Successfully obtain crucial information about if and if so, which existing features needs improvement according to the staff and what new feature they may want to see in the new tool?

2. Being able to suggest a valid solution based on the gathered information from previous goal and studied techniques. A valid suggestion means that the supervisors from the division approves the suggestion and lets project proceed

3. With given solutions from preceding goal, develop the RITA tool and implement its sought features. It should at least consist of all features that has been stated as “must have”, and preferably include some, if not all “nice to have” features

4. Examine if the application and its features has been successfully implemented, both in regards of performance and usability. Also, verify that the tool resolves and eliminates the problems that the division has previously encountered.

1.4 Scope

No particular demands of a specific technique have been stated except that the application should be web-based and for convenience open-source.

The report will not discuss any detailed sensitive information regarding any of the companies involved in this thesis work. This is mainly a precaution so that no business secrets can be used to in any way harm or exploit neither of the companies.

1.5 Outline

Within chapter 2, central concepts used within the project will be explained. This will be followed by a chapter 3, in which the projects methods will be described. Chapter 4 shows and explains the overall design of RITA, whereas the succeeding chapter 5 explains how the application has been implemented. Chapter 6 contains compilations of test result and a presentation of findings that has been made. Last but not least,
chapter 7 will bring the conclusions and offers a discussion regarding the RITA and the project as a whole
2 Theory

To better understand the many concepts of this project, a theory chapter is present to bring clarity into some of the more central ones. It is sought to aid when reading succeeding chapters in this report. To begin with, the project is created due to the need of a more sufficient issue tracker. That is also where the main focus will be set. Additionally, technical ideas and test procedures will also be explained.

2.1 Issue tracker

Modern computer systems have been around for approximately half a century now, and so has programs. And as we all know, programming could also result in bugs that needs resolving [1]. But how can this be handled sufficiently? Commonly, the programmers and other people that are involved in the development process need a mutual way to report these issues in some way, or track them if you will, hence the name issue tracking.

Issue tracking is frequently used in everything from small scaled projects to large enterprises and can be performed in several different ways. Traditional mailing lists have been used more often before the introduction of specific applications. They can provide a more efficient organization and can enable e.g. testers to more easily contact the developers regarding present errors [2].

There are a variety of ways to report an issue, but the most basic ones usually consist of a name, identifier and description with vital information such as the environment that the issue has been discovered at, instructions on how to reproduce it if possible and other diagnostic data. When created, issues can have different states to distinguish it status. An issue can commonly have the status open, fixed, and duplicate or won’t fix depending on where in the process it is at. Open means that it is still not solved, fixed and duplicate is self-explanatory. Won’t fix can mean that no one has been able to reproduce the steps given and therefore may be unique for the issue author. [1]

So now that the different statuses an issue have been established, what makes a good issue report and what challenges exist in creating one? Lotufo, Passos and Czarnecki discusses this in their paper [1]. According to them, misleading reports can have a great impact at the general workflow. These issues may not result in any changes to the
development after its initial glance and wastes precious time from actually relevant errors. One other aspect is the redundancy. Even though it may have a positive effect on the issue at hand, perhaps by providing a more thorough description or resolution, it can also easily just be set to fixed and abandoned. This results in a loss of valuable information and the people that writes the issues may avoid sending any more issues reports since they are assumed to be ignored. Comments is one other key factor to a successful issue tracking procedure, especially to clarify and extend an already present issue. Although it helps, it may also hinder the process if the comments become great in numbers and the most relevant bits is spread out. Therefore, some kind of rating or highlighting is preferred.

2.1.1 Related work

There exists a number of issue trackers currently on the market. Most of them requires a payed license which makes it more difficult to do a well based comparison. There still are some articles written regarding these tools. There main purpose is often issue tracking, but can often be combined with project management and other important features that help with the complete workflow of a project from its initial state to its finish.

It is only natural that the issue tracker is integrated with the rest of the projects management. Today, these applications can also be closely integrated with e.g. integrated development environments (IDE) that developer use [3]. That means that they can create new issues automatically and with little overhead for documentation and extra time spent in a second project management environment.

One of the more frequently used solutions that adopts this principle is named Jira and is developed by Atlassian. The product is being presented as Way more than a bug and issue tracker on products website. Overall it is very advanced and focuses on many other aspects than plain issue tracking, although it initially had issue tracking as its major features [4]. Furthermore, as agile development has been more frequently incorporated into modern project, Jira has evolved into a full scaled project management solution. It can offer several different boards such as Scrum and Kanban, a wide variety of reporting tools and advanced tools for planning [5]. One other feature provides a specific classification for bugs in its environment and they can also be directly connected to an existing user story that the project holds [4].
Redmine is another project management tool that provides similar functionality as Jira. It comes with a simpler user interface and can be, according to Coney and Tunnell at the MICE experiment, used by both students, professors and engineers with ease. The simple interface does also result in minimal system requirement. The classification of an issue is either a bug or a feature. Every issue can have watchers attached to it. They serve the purpose of making people able to contribute with useful information regarding the issue or they may have added solely by pure curiosity. Some cases do however require discretion, which Redmine has solved by having a private or public state of an issue. The restriction can be customized to e.g. member of a very specific project that does deal with company secrets and other sensitive information. [4]

2.2 Usability tests
Whenever an application or any other aspect of the computer interaction is implemented, it is vital to ensure that the general perception regarding usability is positive. Formally, applications were commonly very specific to e.g. a company or institution. Some were not even used outside a certain geographic location. Judy Kay discusses the evolution of usability in her paper and points out that the introduction of Internet and portable devices such as laptops and smart phones truly forced usability testing into new areas [6]. She also suggests that one of the major changes in contrast to earlier application is the importance of interface likeability. These days, a poorly developed interface is not just preferred, it is often crucial to maintain a high level of users as possible.

2.2.1 Potosnak’s three steps
There are numerus ways to conduct tests that verifies that the user experience is adequate. One of them has been presented by Kathleen Potosnak. She points out three distinguishing factors that is crucial to conduct proper usability tests.

Firstly, the user group that participates in the study should also be the group that eventually will be using the tool, or at least will come in contact with it in some extent [7]. Or in a larger scope, what kind of e.g. culture background and general perception does that intended group have regarding usability? The ethnology that is describe in Aaron Marcus article states that ethnology is the key to building future applications [8]. Since the great expansion and availability, usability has turned to more im-
important than many other factors, therefore, it has also been grown in importance how to test them and make detailed studies of the possible user group or groups beforehand, regardless of its size. If the usability is not there, the user can ignore to use the expected or simply choose another, convenient solution.

Secondly, the population should be tested in complete scenarios that may occur to them in their everyday work. It should be a realistic scenario and not a test of isolated features that is separated from the rest of the application [7]. This means that a casual demand to just sit with the application and see what happens isn’t sufficient and concrete enough. In order to measure the usability, it need to have some kind of statistics, from number of mouse or keyboard presses to the actual time spent with a given task. Statistics does also play a key role in standardizing the tests, enabling a wider set of test population, environments and perhaps futures application [7].

Lastly, it is vital that the test users do actually test existing functionality in the application. No test person can’t predict all of her of his actions toward something that is unseen or unheard of. If some functionality is missing, it is better to exclude them the test so that the population does not get confused and searches for thing that isn’t present when conducting the tests [7]. Due to this fact, is can also be preferable to conduct several tests as the development progresses instead of a single test.

2.3 Performance tests

One other major aspect of testing applications is its performance. As one can imagine, there are a very many ways to conduct tests on application, and even if is being narrowed to Javascript-based languages the list still remains long. Nevertheless, some test methods are more frequently heard of in scientific articles.

One of the mentioned tool is called JSWhiz, which is integrated into the compiler while testing. Its purpose is to detect memory leaks, which is a big aspect of Javascript programming since it may cause a discomfort toward users and puts strain on the hardware that the application is running on, even though there are fully functional and well written code. This is present although Javascript comes with a garbage collector similar to Java. The reason being the many variations of frameworks, different wrappers and other abstractions of the code. Some abstractions is needed
to withhold a good backward-compatibility toward older browsers [9]. This is not just a pain for developers but may also be painful for the memory of the browser if executed poorly.

There are also examples of a more native approach called Mutation Testing which both Mirshokraie, Mesbah and Pattabiraman, has contributed with their findings in several recent articles. It is based upon the concepts of mutants, which is a set of operators to be applied on the actual code [10]. This could be e.g. missing semi-colon and braces, typos and other syntactic errors. After it has been applied, the ability to manage and avert these errors is being measured resulting in a mutation score which then can be set into a statistical context. Although this is a good test method for robustness, it also comes with a great amount of required calculations, especially in large-scaled systems. One other issue is the fact that many times different implementations appears differently but will act equivalently in the actual code [11].

Loading time is one of many factors that can reflects on the overall impression of how well an application works. It may be one of the major considerations that web applications must be kept to a minimum. But it is not just actual bandwidth that is of importance here, although it lightens some burdens. The actual time for an interaction to give feedback and appear on screen can be separated into various categories based on programming language and setup that is uses. In Google Chrome developer tools, it is latency and downloading times that is measured. This has been used for different types of tests, most commonly regarding 3D rendering, where it truly provides an elaborate measurement. One of many areas where it has been used is in medical imaging during the presentation of detailed MRI images [12]. One other area that benefits from using the developer tools for testing time is interactive visualizations. It can be used as in Bostock’s et al. articles which conducts test of the time-efficient way to present large-scaled maps and graphs in a web application. The three contestants D3, Adobe Flash Player and Protovis was benchmarked in respect of initialization time [13]. I both cases, the tool proved excellent to measure performance in regards of speed in a web browser.

2.4 Modern web development

The never-ending demand after more sufficient and visually appealing web applications has set the development and its tools in the fast lane.
Many of the most recognized alternatives to date is based upon the programming language Javascript, accompanied with HTML and CSS in some way.

One of these concepts is called Single-page applications. SPA, which it is abbreviated to, is supposed make the experience of interacting with a web application or page more fluid due to more client-side computations. The basic principle relies on request that isn’t sent to the server side, but instead the general load on the server will be reduced since more logic is placed on the front end and client system. This results in a decreased amount of executions and data transfers to the server as well. Much of the code can also be reused, hence the fact that much functionality can be added through e.g. Javascript frameworks. [14]

### 2.5 Model-View-Controller

There numerous approaches and techniques to succeed with SPA concept. Some of the more popular way is to use with Javascript frameworks such as AngularJS and EmberJS that uses the Model-View-Controller (MVC) design pattern [15]. MVC is a design pattern commonly used in software and web development with distinct user interaction. It’s characterized by its separation of user interface and underlying data into three different categories [16].

#### 2.5.1 Model

The models can be responsible for several different tasks, but most noticeable is the management of data structures. Is also commonly have some sort of data abstraction to more easily access and manipulate it [17]. Whenever data is needed, the model will be called by the controller to return its data from e.g. a database or the local storage on the client system. In other words, the model both holds and changes its data based on requests from the controllers [18].

One of the biggest advantages with this approach is that the code complexity is being reduced due to less variations in the code. The database interaction will not have its logic in the same file as code responsible for providing the templates with correct appearances or any functions that handles things that are completely unrelated to it.

#### 2.5.2 View

In contrast to the model, the view is responsible for the visual aspects of an application. It presents templates, applies different types of styling and
sets up the entire grid that the users will navigate in. Everything from buttons, to menus and dialogs is structured in the view category. Whenever the views become more than a few, they can be separated into several different files. By separating the code, it makes the files more readable and easier to debug. However, the code can occur as inline HTML and styling if that is preferred.

With this basis, it is also vital that none of the elements in the view is related to the overall logic of the application. This is to make the work for a designer or usability expert easier since there are not interested in how the data is retrieved or sent, but instead how it is presented to the user [16]. So, try to keep the code as separated as possible, especially in large-scaled applications.

2.5.3 Controller

Lastly, the controller binds the two other categories with each other. Whenever e.g. a key is pressed or any other type of event, the controller registers it and runs whatever implementation it is supposed to do [19]. Perhaps a mouse click on a button shows more text, or a URL path that changes to /index.html tells the controller to change template to a main page, requesting new data and visual styles.

On the other side, controllers are in charge of deciding what actions will be triggered in the model category. It could be anything from deleting a row in a database to updating the cookie for how long the user is allowed to be logged into the application. Additionally, controllers hold the responsibility to ensure that the model receives data that is well formed, correctly ordered and be of the right file format, for example when uploading files to a server. [16]

2.6 Object-relational mapping

As mentioned in previous chapter 2.5.1, data abstraction commonly used in models. These are used whenever a traditional database that relies upon a query language, such as SQL. Although having SQL queries inlined has worked sufficiently, it can be hard to grasp what every query does. Further, the SQL code will commonly be injected into other programming language as strings. This could be considered bad practice by some and avoided by using better solutions.

Object-relational mapping (ORM) is a frequently used option. As seen in Figure 2.1, the technique creates objects in its own language and builds a
virtual database with these objects. All of which is later translated into SQL queries and sent to chosen database. The biggest advantage with this approach is that it does not require any SQL in the models at all, making the development time for database-related work a lot shorter and by that, cheaper [20]. Depending on ORM, it can’t always offer the exact functionality as native database queries, but this could both be intentionally or a result of high complexity that isn’t easy to abstract.

![Figure 2.1: Basic principle of ORM.](image)

### 2.7 Package manager

This is a type of software that often consists of command-line interfaces. Its main purpose is to install, uninstall, upgrade and configure software packages [21]. These can be seen in many operating systems, especially Linux- and Unix-distributions. In is popular in software and web development as well. For example, Maven (Java), NuGet (.NET), npm (Node.js) and Bower (web languages) are some popular choices. In order to add them to Node.js, most often a package manager is used. For RITA, npm is used. It is fully CLI-based and makes adding functionality much easier than manually find all packages.
3 **Methodology**

In this chapter, a detailed description of the projects methods will be presented. The methodology is structured such that they can correlate with the given goals that has been mentioned in section 1.3.

### 3.1 Investigating features

In order to obtain the information regarding the features present today in CRT’s old tool and fulfil the first goal in section 1.3, several interviews will be scheduled with CRT staff. The staff will be given questions regarding their experience with the current issue handler and its features in general. They will later be encouraging to leave their options on specific features and whether the respondents think that it works as expected. Lastly, each CRT staff member will be allowed to leave their own suggestions on what non-existing features they would like to see in RITA.

In addition to interviews, the tool will be observed while in use of any CRT staff. During the observation, notes will be taken on how the tool and its features behaves. This will give an idea what the staff is familiar with and if there are any way to make the application in some way similar in it interface. It is also important to see how the user reflects on the usage. Will there be signs of frustration, stress, et al. and what triggers these possible feelings.

### 3.2 Finding appropriate techniques

An initial literature review will be conducted using valid web sources, such as scientific articles and conference notes from several large scientific databases together based sources with less scientific validity if no articles has been written on that specific topic. For example, as development forums, blog posts and new paper articles. The topics that will be investigated concerns e.g. the most popular issue trackers to this day, programming languages, design patterns, databases, front- and backend solutions and communication interfaces. Criteria such as speed implementation easiness and minimizing workload will be taken into consideration while making the final choices and successfully finish the second goal from section 1.3.

To better understand one e.g. framework, smaller “Hello World” applications may be written. These could help deciding which framework is easiest to start off with or if it resembles something that is
already know to the developer. These decisions must, however be taken with caution, since free hands where given regarding techniques to use. These loose restrictions can be both freeing and risky. Freeing since it doesn’t force one to use techniques that may be obsolete and completely unknown to the developer. In contrast, the risk of having several choices it that they can be chosen poorly and hence consume time that could be better spent. That means that these choices must be studied and taken carefully.

3.3 Development

As a key part in this project, the development phase will span through the majority of projects time. Its features will be implemented based upon the information retrieved from the CRT staff and requirements provided by the supervisors. The sought-out techniques will be implemented from scratch but some key elements may remain the same, such as legacy data from the previous tool. As of the third goal of section 1.3, it is vital that the interface remains easy to use for more senior member and also newly hired staff.

While developing, manual tests of all features will be made. Automated tests will not to be created in this project due to time shortage and not fundamental knowledge in how to conduct these types of tests.

Since the project consists of only one person, it won’t be possible to use methods such as Extreme Programming (XP). Nevertheless, consultations regarding the company’s own system developers is likely to occur if needed.

3.4 Tests

The last goal that has been stated concerns tests, both from a usability and performance point of view. The tests will ensure that the development has been produced something rigid and reliable solution to the CRT division.

3.4.1 Usability tests

As mentioned in the theory chapter, the test will be formed from the article by Kathleen Potosnak called “Recipe for a usability test”. She suggests that test should be conducted on realistic subjects and not just random populations. That means the usability tests should be performed on as many CRT employees as possible instead of e.g. the entire CGI Sundsvall office. Continuing, the subject isn’t meant to “play
around” with the application. This is motivated by the need to draw conclusion and statistics based upon each user’s actions during a specific task. This won’t be possible if there all doing completely different things with no structure.

Regarding the application state of development, RITA should be near finish rather than in its crib. The most obvious reason is that not all features can’t be tested when they haven’t been implemented yet. It should however not be tested on a completely finished product either. This approach would leave out necessary room for improvements based upon the feedback from those who participates in the test.

Each subject will be given six scenarios and a seventh, more subjective question. Scenarios are used so that subject isn’t guided be the conductor of the test. Even though there is a structure, the subject must move freely in the user interface [22]. The duration of each scenario will be noted as well as the number of time assistance is requested. Any unusual behavior or verbal feedback will also be written down. All of which will help provide a foundation for the result and conclusion of the test. The complete test sheet can be found in Appendix B.

Performance tests

There are many ways to measure the performance of web pages and its Javascript libraries. Unfortunately, these test methods do commonly concentrates on network or partial Javascript performance [23]. Methods that can measure the entire life-cycle of a request to finishing rendering the page is not as many, but they do exist. The reason why these tools is sought after is because it will provide a more accurate view of how the user percepts the interaction with the web application. Measuring only parts of the life-cycle may result in misleading result. For example, discovering that the data retrieval is fast omits the possibility that rendering incoming data can have a negative effect on the user experience due to delays.

The method that will be used in this thesis work is Google’s development tools, previously mentioned in chapter 2.3. It has been used in several areas, which the theory chapter showed, however, it will in this case be used for solely benchmarking Javascript performance.
The integrated developer tools consist of a large set of tools that help diagnostic the performance of web applications [24]. One of the tools is located in a tab called timeline as shown in Figure 3.1 when finished, is can plot its measurements in a convenient pie chart, seen in figure 3.2.

![Figure 3.1: Google’s Chrome’s Developer Tools overview.](image)

The timeline tool can record and measures the duration of a complete load sequence of a web application. This includes aspects memory consumption, present frames and different events types [13]:

- **Loading**: events related to network events such as send request, receive response, receive data and finish loading (complete network request).

- **Scripting**: this involves garbage collection, function calls and events such as mouse clicks and key presses captured from the user input.

- **Rendering**: invalidated layout, calculates element styles and scroll nested views.

- **Painting**: decodes, renders and resizes images.

Altogether, this provides a uniform view of how fast the application loads and is being visible for the user. Other values like other and idle will not be a part of the result due to relevance and inability to track what other time is.
### Summary

<table>
<thead>
<tr>
<th>Range</th>
<th>0 - 7.15 s</th>
</tr>
</thead>
</table>

#### Aggregated Time

- **7.15 s**
  - 10.877 ms Loading
  - 804.917 ms Scripting
  - 219.914 ms Rendering
  - 15.225 ms Painting
  - 115.944 ms Other
  - 5.98 s Idle

---

**Figure 3.2:** Duration summary in the timeline tool.
4 Design

This chapter will illustrate and explain the design of RITA. The design sketches have been drawn digitally to more easily demonstrate the design. The design is separated into three different subchapters containing each main view that is supposed to be used in RITA.

4.1 Issue list and search feature

The first sketch shows the view which provides the list of all issues available in the system. The list should be able to be ordered by column in both ascending and descending order. The columns in this image have been limited, but may hold a greater number of columns. The search should be able to search all issues in any field. A single click on an individual issue should result in a new view that shows all information regarding that specific issue.

![Mockup of the issue list and search field.](image)

When first sketching RITA’s interface, it was important to design it so that users were encouraged to use the search feature extensively. As seen in Figure 4.1, it does not have any “advanced search” option since the search should be able to filter out quickly enough and minimize the number of mouse movement and clicks. The search in this application is sought to use multiple parameters without the requirement that it must come in the same order as the actual string it compares. To more easily explain this, an example is provided. The issue called “Worst case ever” will be found using RITA’s search filter by typing “worst ever”. The default search will only find the issue with the terms “worst case” or “case ever”. Otherwise, the string will break and return no matching issues.
4.2 Add issue

Adding a new issue should be straight forward. Is should contain a title, description and possibly tags to make it less hard to find. The list of tags should be supposed to be presented as suggestions whilst the field is filled with letters that matches any given tag. If it does not exist, it should be saved to be later used in future additions of issues.

![Mockup of the add view](image)

Figure 4.2: Mockup of the add view.

Naturally, an issue should be able to have attachments as well. In this case it should be as simple and time efficient as possible. Therefore, a drag-and-drop feature should be present as seen in figure 4.2. Even though the traditional “browse” approach is proven to work, it still takes a lot of time to navigate through a long folder hierarchy and at last find the file to upload. Instead, the drag-and-drop functionality provides the user with a quick way to drag one or several files into the web browser windows. This particular mentioned, it is sought to make the user interact with as little mouse movement as possible.

4.3 Single issue

When clicking the link of an issue, a new view is show that summarizes the information from that specific issue. Since the legacy data does have more information than the newly added issue, some information is not present. These headings are supposed be hidden whenever a value is not available so that it doesn’t confuses the user. The same approach is used in the comment section. If the issue has no comments, the section will be
hidden as well. In Figure 4.3, no legacy information I present, but comments are however visible.

![Mockup of the single-issue view.](image)

Figure 4.3: Mockup of the single-issue view.

The drag-and-drop feature previously mentioned in Figure 4.2 is also used in the comment sections to more easily upload files to the comment. This is designed so that user won’t be force to edit existing issue but rather complement with new information such as screen captures and code snippets.
5 Implementation

This project has been blessed with no restrictions regarding its architecture and programming languages. This makes it possible to choose more wisely and make decisions based on experience and curiosity rather than being restricted to solutions that might not be beneficial for the projects overall success. This chapter will in detail describe how the project has been developed and what solutions has been used.

Note that this chapter presents the final development and therefore contains nothing in regards of EmberJS and its surrounding frameworks or libraries.

5.1 Tools

In order to implement RITA in a modern and innovative fashion, a trustworthy development environment is all but necessary. In this project, Sublime Text is being used. It is a more sophisticated text editor with its own built-in package manager. This feature makes it really easy to install further functionality that might come of use during the development. This includes real-time syntax checkers for numerous programming languages, different syntax highlighting features and automatic beautifiers, meaning that the code will be indented and follow coding conventions to a specific language. This is often the beauty of web programming; it does most often require nothing but a suitable text editor.

To be able to run the application, firstly, the software stack WAMP is being used, partly because it supports for the database administration tool phpMyAdmin and it sets up MySQL databases with ease. It could however be set up without a bundled software stack. Alongside phpMyAdmin, a small MySQL plugin for Microsoft Office Excel called MySQL for Excel is being used to export the legacy data more easily.

Alongside WAMP, Node.js is also required to run the RITA. It relies heavily upon Node Packet Manager (npm). As explained in the theory section 2.7 package managers assists in providing further functionality to any given environment or software. In this case, npm plays a big role in installing features responsible for communicating with the database, maintaining attachments and serving the front end in which all visual templates is being presented.
Since the development takes place in a Windows-based operating system, it’s built in command prompt PowerShell are being used to command Node.js to install packages and run the app. This is nevertheless easy to do on a Unix/Linux-based operating system as well. As long as a command prompt is present.

### 5.2 Overall architecture

RITA is built upon a number of key components which is separated by a traditional back end and front end. The communication between them is handled with Representational State Transfer (REST). This architecture is almost entirely based on Javascript, making it easy to deploy on any given environment. The only things that differ is the file system and database that can be interchanged to other suitable options if necessary.

The next chapters will go into detail in what each box contains in Figure 5.1.

![Diagram](image.png)

**Table 5.1:** The components of RITA.

### 5.3 Front end

RITA has a front end that is mainly based on the popular framework AngularJS. It has really become one of the go-to solution when developing new and fast web front ends. Since RITA has Angular as its main framework, it does rely on the previously mentioned Model-View-Controller (MVC) model, which also delegates how the code structure of this project is supposed to look like. It means that the code is split up to be more easily understand it and scale it more efficiently. Each part has a corresponding sub chapter, but before this, it is good to know about the core.
Angular applications are commonly built with a central core file from which all other modules et al. originate from. The core file is responsible for several tasks such as managing the routes in the application. Angular has a module called ngRoute that in turn has a routeProvider. This component can control all paths in the application and it can also tie a specific controller to that specific path. The core is furthermore responsible for the configuration of modules that will be used globally throughout several controllers.

Even though AngularJS has the main responsibility for the front end, some minor additions is present as well to improve the overalls style of the application. Bootstrap is a popular CSS library that provides predefined styling features. It contains good solutions for styling and presentation of data. These styles have then been change to better match RITA’s interface. Some of Bootstraps more advanced features does involve actions where jQuery is needed.

5.3.1 Model
Angular has no predefined way of structure model data. The scope object plays a central role when it comes to managing the data and provide the controllers and views with its content. Whenever data is received from the server using REST, it is handled by the scope and stores its data within that scope. It can also be parsed and manipulated before its inserted if needed. When finished and put into place, it can then be reached from other modules as well.

Naturally, the model does also withhold data that is sent to the back end, and not just from it. This can be things such as objects containing text, JSON data, and attachment like pictures or documents. If parsing is needed, this can be made in this case as well, directly invoke changes on the object at hand. Changes that can be triggered by actions from the different views.

5.3.2 Views
The scope does not just have the ability to provide data, but can also be responsible for calling functions from the template. In RITA’s case, submit buttons does have submit functions that is directly tied to the scope. When pressed, it directly calls the submit function in the controller that the view is tied to.
Even though functions can be called from the view to the controller, some logic remains directly inside the actually HTML. One example is the sorting of the issues in the list. Much of the logic has already been implemented in Angular already which makes these trivial tasks possible to do inside the template. However, it has been kept to a minimum in order to withhold the MVC model and not having more logic in the view than the controller.

5.3.3 Controllers

The most of the remaining logic is written in the controller. Inside it, all request to the back end is handled and responses is pushed to Angular’s scope object. This object can access all parts of the actual website and have its data called upon using different functions.

For instance, while iterating through all posts retrieved from the back end, a built-in Angular attribute `ng-repeat` is used. It acts like a basic for-loop. Inside the function, several options can be set. In this project, these options consist of an order option, specifying what column in the table the data should be ordered by. This is however triggered from the view that can access this function.

Controllers will also make all communication with the back end via the REST API. Thanks to JSON, the communication can use Javascript own objects easily. And since RITA’s backend is based on Javascript as well, the retrieval of requests is likewise easy.

Each controller does have injected modules that might be used both in the view directly or inside corresponding controller. For instance, a filter module is injected, this module is not provided from any module resource but rather developed by own hand. Furthermore, several additions have also been used in RITA

5.3.4 Additional modules

Angular provides some modules that isn’t included by default in the start of the project. They can be retrieved from Angular own repository and injected in any given controller. One of the most frequently used modules is `http`, it provides necessary functionality to use the REST toward the backend. This is complemented with `routeParams` that can handle specific data field within the REST request and response.
Whenever Angular and its related modules is missing some functionality, there may exist modules created outside Angular’s own library that provides that functionality. For instance, one of the modules that has been more frequently used during this project has been helping simplifying the tedious task of handling file uploads with help of using multipart/form-data. The library is called *angularFileUpload*. Its main purpose simplifies the overall handling of attachments. RITA sends the attachments in another request than with the comment or addition of an issue that the attachment is tied to. The upload event is instead made by *Fileuploade*s modules function *uploadAll()* after that the previous event has succeeded.

One other module is *toastr* that is being used in RITA to let a small bar being shown whenever a request is successful or not as feedback for the user. Whenever something is sent or received this bar will be shown for an optional amount of time.

Lastly, Angular’s standard library does also include a filter function which is usually used in search fields. Unfortunately, the filter does not support multiple words searching. If several words are put in the field, it must remain the right order to find a match. A behavior that isn’t optimal for the sought search in this application. Since RITA’s search is supposed to search multiple keyword at once, as stated in the design chapter 4.1, an own implementation is required and then tie it to its suitable controller.

## 5.4 Back end

RITA’s back end is heavily based upon the platform Node.js. This platform is highly capable and it can expand its functionality even further using external frameworks and modules much like Angular do. The modules are brought into the application with help from the command prompt based packet manager npm.

One of the most popular additions to Node is the web framework ExpressJS. It holds a great number of features which is used in the application, starting already in the very core of the application; the app.js file. It contains a series of declarations that is vital for the overall application, many of them with help from ExpressJS that acts as a middleware. One of the crucial parts to configure is the path to the files that is responsible for the front end. This is made by ExpressJS that specifies the folder `/public` where all front-end content is stored. In other
words. Express redirects the client to that certain folder on the server. It will hence provide the client necessary files required for viewing the front end of RITA.

Other uses of the middleware are for instance, specifying a path to the attachments storage, declarations of encoding variables and the setup of the listening port for on which the applications responds to. The remaining parts of the application is placed in separate files within a well-structured file structure. Overall, the separation makes it easier to handle dependencies and makes sure that modules isn’t used in any unnecessary files.

One of Node’s main tasks is to act as web server and has the responsibility to serve the user with the front end and the routes that the REST make use of. This functionality is built into ExpressJS and its router object. These routes will act as the channel between the back and front end for sending data. It retrieves all possible routes from separate files based on functionality. RITA has separate files for issues, comments and its main (public) route mentioned above.

5.4.1 Database

When the routes interact, it’s is most commonly with the database. The database used in RITA is the traditional relational database management system (RDBMS) MySQL. To interact with it, an ORM, which was touched upon in chapter 2.6 was been chosen. It is called Sequelize which also is provided via the npm and hence integrated with the Node.js platform.

With Sequelize, each table can have its own file and model tied to it. The ORM will then use these models to generate the tables and makes it possible to use any attributes in the model as objects inside Node.js. Model object has function such as find(), findAll() and create() tied to them and they can be called wherever the model object is included. For instance, in the route files where many paths interact with the database. These function is what Sequelize uses to call the database and send its SQL queries to the database. Naturally, the functions accept parameters, inclusion of other models to the query and it can specify what attributes that is should return just like ordinary SQL.

Since the legacy data present in this project is strictly structured the models must correlate to that data in preferably in the same order and naming
as close as possible. The datatypes on the other hand is almost exclusively of the type TEXT.

5.4.2 File handling

RITA has the feature to upload attachments to the back end. On the receiving side, the node module Multer is used. Multer is package designed for Node to handle incoming files and store them at the servers local file system. Multer can specify its own route in the router object. The route will then also be the folder where the files are stored on the back-end file system. The file has then rows in the database that represents it, alongside an `issueld` or `commentId` depending on where the attachment is placed. Furthermore, since the nature of this project is to develop a proof of concept, there is no limitations towards file types and sizes.
6 Result

It is now time to present the results that was emerged as the project has progressed. Overall, the result of this project has had several issues which has delayed the development and affected the general success of the project. Even though, the goals stated in chapter 1.4 has all been reached in some extent.

6.1 Investigating features

Firstly, as the analysis of Telia’s own tool and features was finished, it stood clear that the tool heavily lacked usability and was very slow. It could even more often than not freeze or shut down completely. There have been numerous reports of people being locked out from the system for more than three hours. These flaws do affect the CRT employees work negatively. The unstable system steals time from other tasks, makes the staff frustrated and not keen to use the tool at all.

The staff at the CRT division also experienced a lack of features, or the existing too inadequate. One of the major flaws is the search feature. One of the interviewed employees clarified that the search acts like shell upon database. This means that operator such as AND, OR, NOR must be used to separate specific search terms, much like a database query. Almost all interviewed had tried the search at some point, but due to its slow speed and tedious design the instead choose other approaches to find their issues.

Having operator-based search is fairly uncommon in today applications. It seems like the tool is really old, making it less similar to the application that is used in today’s IT world. Today, many of the sites implements search field such that they can search multiple terms without operators. This provides less mouse clicks and a more rapid user experience.

When adding a new issue, there were dozens of tabs that each shows certain fields. Far from every field is used by CRT leaving many fields blank. There are numerous options, radio buttons and drop-down menus, many of which remains untouched by the CRT staff.

As for the expansions of an issue, a work log field exist. The field does not provide a automatic timestamp or indication other than the ones Telia provides themselves. Consequently, it is up to every user that uses the
work log to provide time and date of when the addition was made. Improvement on these features was highly desired by the staff. When it came to new features, the CRT staff had less to say. There were suggestions to implement the login system with the staff’s current user accounts instead of being forced to login all the time. But the common perception has that anything new would actually be better than the current tool. With that being said, it still required the same vital features as in the current tool.

From what investigation showed, it seems like the number of new features isn’t as sought after as for improving the ones that already exist in the current tool. Although some new features were suggested, it came clear that they were comprehensive and had many of the wanted features already built in along with many that aren’t used at all. This made the task of reporting or searching for an issue so overwhelming that the CRT staff avoided it partially or completely.

The ability to properly comment existing issues has several advantages that help an issue be revised and corrected if new solution would emerge. The newly improved feature can also add date and improved user format that is needed to put the comment in the right context. Additionally, adding files to comment would make description of a problem more effectively. Screen captures and code files can be many time more descriptive than trying to compose it in text.

### 6.2 Finding appropriate technique

After the investigation has finished, a suggestion was made to the supervisors which approved the technical choices. As stated earlier, the project did not have any tight restrictions which made the approval even easier.

The project did suffer from several changes that did cost the project a great deal of time and did affect the overall success of the project. One of the techniques that did remained through the project was NodeJS and many of its modules. The choice was more based on previous knowledge. The communication with the front end was based on proven techniques that has a lot of documentation in case errors occurred. The overall front end framework and database in contrast did not. At the start of the project, it has supposed to be built with EmberJS, alongside the database MongoDB. Unfortunately, the latter options had to be dropped, approximately one month into the project, and has then
replace with MySQL. The motivation being that MySQL had a relational structure that correlated with the legacy data and hence could more easily migrate the data.

The front-end framework on the other hand was replace in middle of the project due to lacking performance and continuous change in its syntaxes from release to release. This fact forced the decision to introduce another framework, and it had to be made hastily. Fortunately, AngularJS had initially been one of the candidates alongside EmberJS and hence not completely unknown. The choice was made based on the great amount of documentation, examples and demos, in other word aid that could have the potential to speed up the delayed development significantly.

All new consideration was off course informed to the supervisors that had full understanding on the matter and helped in the extent they could to make the changes easier.

6.3 Development

The overall development of this project has been struggling with finding appropriate techniques, hence misdirecing much of the focus from completing some of the features that where sought initially. The EmberJS framework was still in development which meant that a lot could change between releases. For instance, during the development of RITA, Ember switched to a command-based interface. This resulted in a much more difficult way to gather relevant and up-to-date documentation. The choice of Ember forced the project to be more about finding the right version of a library instead of implementing actual functionality. A fact that was not just frustrating but indeed inefficient. Additionally, it had too few possibilities to extend its functionality further. This fact made even the most trivial tasks as setting up a table of all present issues difficult. The one of few options available did do the job, but unfortunately performing poorly. The web browser did freeze frequently and resulted in an unacceptable user experience.

The transition to AngularJS did put the project back to almost its initial state, with exception of a working database and back end environment. The templates and overalls code and file structures was completely different and all pervious code had to be rewritten. Fortunately, the common design pattern did make the transition at least a bit more manageable.
Even though their structure differs, both frameworks deal with separating specific logic and are closely integrated with traditional HTML. When the framework was changed to AngularJS, it led to a more rapid development that was more in line with the knowledge of the developer. One other factor that made Angular more appropriate to develop was the great amount of well written and updated documentation, much of it from Angular’s official website. All these factors led to a higher paced development process. With a higher pace, it also resulted in more functionality successfully being developed than before the transition.

The issue with the database did affect the projects progress less than the front-end framework, although it did force the data model to be rewritten from scratch. The introduction of MySQL and a Sequelize-based ORM made the overall database management easier to understand and maintain. Sequelize has far less documentation than the front-end networks, but it was sufficient enough to perform the tasks at hand.

6.3.1 Issue list and search feature

The main view, seen in Figure 6.1, of RITA is its issue list and search field. In comparison to the sketch present in chapter 4.1, there are no crucial differences. Here, as expected, all issues are listed. The list consists of the columns id, subject, system, originator and date. The subject acts as the link to the single-issue view. The columns of the table can be ordered by every column attribute and show a range of 15, 30, 60 and 100 issues each. Please note that actual data can’t be presented in the figures due to secrecy toward the company and the division responsible for that information.

The search feature did become fully functional, being able to use separate keywords. The feature also does search continuously without requiring any press at enter or search button. As shown in Figure 6.2 a pagination component is also present whenever the total number of rows exceeds the preferred number of rows. The row count is also seen in the far left of the page (not visible in the figure).
6.3.2 Add issue

The issue to add new items is also similar to its original sketch. It has maintained all components except for the tagging system. These features were dropped due to the fact that the search functionality was sufficient enough that a tag system only seemed redundant. The information that issue hold hopefully be enough to find the issue by solely using the search field. The sketch shows two field, namely title and description. But in Figure 6.3 the solution field has been added because it was suggested by the supervisors. It was stated that some issues could be resolved before it was documented, but should nevertheless be put in the application in the event of the issue being occur again.

The drag-and-drop was also implemented. Additionally, a delete button was added if it landed in the drop area by mistake. The areas are mainly explanatory, but the supposed file can be dropped whenever the drop rectangle is lit red. For browsers that isn’t supporting drag-and-drop, a simple browser option is available. Whenever an issue is submitted, the, a small, coloured bar is seen next to every single file, visualizing its progress. When finished, a small toast is shown to inform the user if the submission has succeeded or not with a green or red bar.
Table 6.3: Add issue view.

6.3.3 Single issue

The view that has been changed the most is the one presenting a single issue. It was supposed to have tags and not shown any other data than the ones seen in chapter 4.3. The additional legacy data fields present in Figure 6.4 was added mainly because it made it possible for the user to search specifically for a unique Request ID that the legacy data provided. It also had e.g. information regarding the author and administrator tied to the issue. Fields that are empty should be valued N/A, meaning not available, although some field remains empty due to time shortage.

The comment section remains the same as the sketch. It will be hidden whenever comments aren’t tied to the issue. Comments does not have any relation to each other, e.g. users can’t reply on a comment, but must submit a new comment that will be shown in under the latest comment. The attachments that can be added uses the same implementation as in previous view.
6.4 **Tests**

RITA has successfully gathered a set of results that now will be presented in the following two chapters. Some of the results and other related material can be found in the appendices in the end of this report.

6.4.1 **Usability tests**

The test was successfully conducted with all employees at CRT. The test population had mixed experience with the current tool and had a large age gap as well. This could certainly be seen in some of the scenarios in Table 6.1. For instance, the first scenarios did tell the subject to find an issue with a certain id. Two of the subject choose to scroll through the list...
instead of searching for the id just like in the old tool. This resulted in a longer time till the issue was found.

Table 6.1: Time chart of the usability test.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Usability test (sec/task)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>max 10 5 10 15 20 25 30 35 40 45 50 55 60 65 70</td>
</tr>
<tr>
<td>2</td>
<td>max 10 5 10 15 20 25 30 35 40 45 50 55 60 65 70</td>
</tr>
<tr>
<td>3</td>
<td>max 10 5 10 15 20 25 30 35 40 45 50 55 60 65 70</td>
</tr>
<tr>
<td>4</td>
<td>max 10 5 10 15 20 25 30 35 40 45 50 55 60 65 70</td>
</tr>
<tr>
<td>5</td>
<td>max 10 5 10 15 20 25 30 35 40 45 50 55 60 65 70</td>
</tr>
<tr>
<td>6</td>
<td>max 10 5 10 15 20 25 30 35 40 45 50 55 60 65 70</td>
</tr>
</tbody>
</table>

In scenario 2 and 3, clarification had to be made. Some participant chose to count the issues listed instead of inserting an additional search term that would not only list only the sought issues but also gave the exact sum of all listed issues in the top left corner of the page.

Also note that the eldest of the subjects did have the longest time in many of the scenarios, in contrast to the two youngest which performed a lot faster results. The general impression was nevertheless uniformly positive, with very fast performance being the biggest improvement according to the test subjects.
6.4.2 Performance tests

The tests between the both frontend frameworks AngularJS and EmberJS had some similar traits. Both frameworks performed similarly in both their loading, rendering and painting duration. Angular did render and paint more slowly than EmberJS. While the loading events where more similar and only differentiate by 2-3ms. All of these events did however not influence end result as much as the scripting time. This is where Angular shines and Ember does not.

As seen in Table 6.2 even at the first small row count of n=2000, Angular is more than twice as fast while scripting. When n=10 000, the difference is even more distinguishable with an average scripting time for Angular being three seconds in contrast to Ember’s fifteen seconds. A time that is not even acceptable for more sophisticated single-page applications that can suffers from a slow first load.

Table 6.2: Performance chart of both frameworks.

When investigating the reasons behind the great difference between the two frameworks, it all was tracked down to two function calls. Remember
that this is possible to track since the timeline records all functions in the scripting view. As seen in Figure 6.5 and Figure 6.6 the difference in these two function calls is noticeable.

The first function is called within the jQuery code present in both the Angular and Ember implementation. Here, Ember uses jQuery to collect all incoming data rows from the REST API. Please also note the amount of data in each GC (Garbage collection) event.

In Angular’s case, the function call is made to the own framework and it results in almost no big GC events, nor high memory consumption. A closer look inside both function calls did reveal the reason behind this significant difference in terms of scripting.

Both frameworks need to get their data from the server, but this can be performed in several different ways. The developers of EmberJS has relied on jQuery’s XHR handling. This handling is using a XHR property called \textit{responseText}. When analysing Angular’s XHR handling, it did instead rely on a newer property called \textit{responseType}. This format does not need to parse the incoming content and is therefore faster and lighter to script.
7 Discussion

The thesis work has now come to an end and many conclusions can be drawn from the different parts of the project. Although it has been a struggle in many aspects, new discoveries have been made. Some aspects have resulted in time shortage and has hence made the prioritizing process even more important. Overall, RITA has brought new realizations and can aid in future projects.

7.1 Investigating features

On a whole, this goal was the easiest to fulfil and the fastest to finish. The initial investigation was made quickly, mostly to maximize the amount of time intended for finding productive techniques and development with them. Fortunately, the company that wanted a new tool had already laid much of the ground work for the study and it didn’t take more than a few meetings before next stage of the project could be initialized. During the meetings, much of the topics discussed was sketched and written down so that it wouldn’t be forgotten. If could also been sufficient to record the conversation, to that the whole conversation could be stored for future use. Some sketches were also drawn at whiteboards, which did get photographed if they could not fit in a notebook.

It would have been optimal to get more input from the whole staff, but this is not possible in larger groups since they are working on many different project and have limited time to spend at these kinds of projects. This may perhaps be one of the reasons why some major organizations struggle with their everyday tools, they may be insufficient, but no one have the time to reflect over this matter and hence, it won’t be improved or exchanged to a better tool. In this case, the current tool is maintained by a client, making these problems even more prominent.

7.2 Finding appropriate techniques and development

The development has been the most challenging and time-consuming part of this project. It has applied many new concepts and brought new knowledge regarding developing both the back end and front end. The biggest challenge being to choose proper techniques, since they had a great impact on the overall result. Some of the choices were made with too much haste, leading to a delayed development process due to late changes. It is perhaps one of the most crucial parts of a project of this
nature to choose techniques that are reliable and well documented. EmberJS lacked both unfortunately. The complications with the framework did nevertheless show the importance of making decisions right from the very start, but also to cancel in time if the techniques fail to perform as expected. Many of the performance issues may not be discovered from the start, but if they surface it is important to have a second alternative that can be introduced directly. The project RITA did not do this fully, since the development continued even after these issues had been discovered, but instead of switching technique, EmberJS remained and caused big problems. A lesson well learned.

These problems could have been discovered earlier if a more sophisticated tests process had been present. It could have stressed tested the application more easily and hence sent alarm of the poor performance perhaps just a couple of weeks into the project. The lesson being; never underestimate continuous testing. The issue with lacking documentation should have been made more thoroughly from the start. If possible, ask the community and more experienced developers for aid.

### 7.3 Tests

The tests made in the project where made in the final stage of the project, which, in hindsight would have been made continuously instead. The possibility to adapt this approach was however difficult due to the limited time that the staff could offer and a development process that had to suddenly take another direction.

#### 7.3.1 Usability tests

The usability test of this thesis work was conducted in a successful way. It had however room for improvements. More detailed scenarios and what the expected output would be could have made the subject more certain if they passed the scenario. Other than that, it would probably have resulted in fewer questions and clarifications needed from the test supervisor.

One question that arise during the tests was whether or not old fields should have reintroduced in the new tool. Some of the test subjects expected some features to be more similar to the ones present in old tool. One example being a separate input field for the attributes *system* and *country*. This is something what can be added, but is the right way to go? Do the division really like to see more fields now that they have been drastically decrease? The supervisors at CRT is certain that fewer fields
will have a positive effect on the number of issues that is being reported. They meant that is would be like going back to the start with too many fields to fill in and a general avoidance of even using the tool.

But this might not always be the case. If too many old features are removed, staff might feel too limited. So, it is important to find the sweet spot between old and new, especially when migrating to a new tool that apparently leave out a lot of features. This also shows the importance of the studies made on the current tool and trying to track what the staff uses frequently and seldom.

7.3.2 Performance tests
The discovery shows a major flaw in jQuery, a very popular library when developing these types of application. The performances drop that this result in may be one of many reasons why Ember hasn’t been as popular as Angular. Ember should be able to implement a solution that doesn’t require jQuery for basic features as those used in this project.

Due to time shortage and an inflexible test procedure, the number of tests conducted has too few. It should have used a better scaling possibility to more easily add a great number of test issues automatically, or even a separate database for these tests. Perhaps an introduction of Angualrs’s own test modules could have helped in this process.

These problems really visualized the importance of having at least a test and a production environment, especially if several developers et al. is involved in the process.

7.4 Ethical and integrity aspects
Now that CRT has exported the legacy data into their own tool, new obligations is introduced. Since the client no longer can administrate the information regarding their issue in RITA, the division must insure that the data remains in their possession and won’t be leaked and jeopardizing customers and integrity of employees. The data must therefore not be shared with other division within CRT or unauthorized persons outside of the office.

Depending on the nature of the information that the CRT division shares internally in the tool, a new question is raised. Is it in their interest to keep
the internal information to themselves or share it with Telia with no restrictions? The first option may influence the business strategy of the company whilst the latter option could strengthen the trust as a business partner. Due to business secrecy, the questions will remain unanswered, that is, if it's considered a valid question.

7.5 Future work
Since the project suffered from major problems mentioned previously, there is room for improvements and additions to the RITA application and the methods used throughout the project. The most prominent the left out features of the actual application.

Although much of the functionality has been implemented successfully, some remains. One of the major being any kind of status indication, much like other issue trackers, it is vital to know what status an issue has to more effectively sort and filter issues that are of importance. The legacy data had a status attribute, but it was never used in any other way except presenting the information in the single-issue view. The attribute would have been changeable directly in the database for this feature to work.

The search feature could have been extended with a highlighting feature and perhaps show related suggestions whenever the user search for an issue. Even though the search actions are useful and without any advanced search options it would draw good use of options such as letting only specific field be search and maybe whole word search among others. Of course, it should be possible to search issue from a specific author or a time range as well. Basically, all general-purpose options to bring an extra edge to the search result, still without and search operators.

As of now, RITA does not track a specific user, it is mainly an attribute from the attribute that indicates the author. Naturally, a user system should be present to better keep track of all authors and perhaps have a role based approach, meaning that some issues is not visible for all users. Some may require a staff member high higher authority to ever comment or remove it. There where talks of this, but it has not realized in the time given. One or several administrating roles should also be present to manage both issues, the attributes of the application and users and their access restrictions. The option of using any predefined access manager that is already in use and implement it into RITA instead. It would minimize the number of accounts that users need to keep track of.
Even though RITA is focused to migrate already existing data to a new, more efficient platform, it would have full support to work standalone from this data and perhaps be integrated into other divisions and share issues between them, if possible. The database would also be able to dynamically remove and add new attributes and columns if that ever should be needed. A simple expansion of a business area to other countries or client will most definitely lead to an increase in issues and that requires a sophisticated solution to scale the app to meet the new situations that may occur over time.
References


Robust Issue Tracking Application (RITA)

Stefan Åhman

References

2017-06-16


“Reporting system based on the university project of the MVC design pattern”, Ye Yinlan. 2013 8th International Conference on Computer Science & Education, April 2013, pp.931-935.


Appendix A: Requirements specification

Here, the list of requirements concerning the application is presented.

Mandatory features

- Web interface and open-source if possible.
- Easily accessible interface.
- Quick issue registration and being able to comment existing.
- Quick issue search that one all attributes.
- Being able to add attachments.
- Automated import to the tool from a fixed place.
- Flexible admin interface in case attributes must be created or removed.
- Reporting feature, such as generating diagrams.

Additional preferred features

- Email feature, being able to emails directly in the tool
- Export feature, for instance to Excel sheets.
Appendix B: Usability test scenarios

The usability test was based on the following scenarios:

1. A specific issue with identity number ___ is being mentioned in a mail correspondence, please look it up in RITA and confirm that it exists.

2. There have been reports that _____ _____ has had problems logging into system _____ ____. Please consider finding the issue(s) regarding the incident.

3. You need to know how many issues concerns system ____ in Sweden. How many are there listed in RITA?

4. A new error has occurred, it’s time to send an issue to RITA with files attached to it.

5. You found new information on a relevant case, please share it in a comment.

6. You co-worker just attached an important file to his comment “Worst Error Ever”, please take a look at it.

As stated before, the time for each scenario has measured and the number of help requests counted.

The last section of the test is more of a subjective question, whether RITA is easily used and how it improves the work compared with Telia’s tool.

Note that the blank areas are not set due to business secrecy.