



Faculty for Economical Science, Communication and IT

Joakim Carlsson, Christian Lindeström

Incident Management System

Computer Science
C-level thesis

Date/Term: 11-06-09
Supervisor: Kerstin Andersson
Examiner: Donald F. Ross
Serial Number: C2011:08



Computer Science

Joakim Carlsson, Christian Lindeström

Incident Management System

Bachelor's Project

2011:08

This report is submitted in partial fulfillment of the requirements for the Bachelor's degree in Computer Science. All material in this report which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Joakim Carlsson

Christian Lindeström

Approved, 2011-06-09

Advisor: Kerstin Andersson

Examiner: Donald F. Ross

Abstract

This dissertation for a bachelor project in computer science at Karlstad University will describe how to conquer a challenge suggested by Tieto: How to create a secure Incident Reporting System with a high level of confidentiality and security for the contents. The system should be easy to use and encourage incident reporting, open for changes and statistics gathering, for those with the relevant authority. The result will be a requirement specification and a prototype incident management system that matches those requirements. Any employee of Tieto will be able to submit an incident report and the system will notify a security administrator at Tieto who will solve the problem. The system will be able to gather information and statistics regarding incidents which can be used as decision support.

Table of Contents

1	Introduction	1
2	Background	2
2.1	Introduction	2
2.1.1	Initial Specification.....	2
2.1.2	SharePoint	3
2.1.3	Existing applications.....	4
2.1.4	Scrum	4
2.2	Specification and design suggestions	5
2.2.1	Diagrams.....	5
2.2.2	User interfaces.....	7
2.3	Final Specification and design adjustments	10
2.3.1	Report states	11
2.3.2	Database system and Security.....	12
3	Project Development Process	14
3.1	Time planning	14
3.2	Sprint introduction	15
3.3	Sprint 1	15
3.3.1	Overview	16
3.3.2	Results.....	20
3.3.3	Retrospective.....	22
3.4	Sprint 2	22
3.4.1	Overview	23
3.4.2	Results.....	26
3.4.3	Retrospective.....	27
3.5	Sprint 3	28
3.5.1	Overview	29
3.5.2	Results.....	32
3.5.3	Retrospective.....	34
3.6	Sprint 4	34
3.6.1	Overview	34

3.6.2	Results.....	36
3.6.3	Retrospective.....	36
3.7	Sprint 5	36
3.7.1	Overview	36
3.7.2	Retrospective.....	38
4	Results and evaluation	39
4.1	System	39
4.2	Code	40
4.2.1	The Graphical User Interface	43
4.2.2	The Manager Layer	44
4.2.3	The Accessor Layer	45
4.2.4	The Independent Layer	46
4.2.5	LINQ to SQL Classes Layer	48
4.2.6	The Database Layer.....	48
4.3	Web Interface	49
4.4	Security.....	53
4.4.1	Access hierarchy	53
4.4.2	Report encryption.....	54
5	Conclusions	56
5.1	Specification	56
5.2	This project	56
5.2.1	Security.....	56
5.2.2	Scrum	57
5.3	Future work	57
5.4	General conclusions	58
6	Bibliography	59
7	Appendix A – Google chart API, bar chart	61
8	Appendix B - Drafts	62

Table of Figures

Figure 2.1 - Product backlog.....	5
Figure 2.2 - A use case giving a brief overview of the reporting system.	6
Figure 2.3 - Flowchart of how incidents are handled after they have been reported.	7
Figure 2.4 - Incident Reporting Form	8
Figure 2.5 - Incident Management Form	9
Figure 2.6 - Incident Search Page Example, chart generated by Google chart API	10
Figure 2.7 – Report states	12
Figure 3.1 - Time table with focus on “pre”	14
Figure 3.2 – Backlog, sprint 1	15
Figure 3.3 - Database Structure, Sprint 1	17
Figure 3.4 - Submit Form.....	18
Figure 3.5 - Meta-Report Form.....	18
Figure 3.6 - Dispatching Form.....	19
Figure 3.7 - LINQ to SQL Classes interpretation of a few tables from a SQL database	20
Figure 3.8 - Web page design	21
Figure 3.9 –Backlog, sprint 2	23
Figure 3.10 - Databse structure, sprint 2	26
Figure 3.11 - Sprint 2 Page	27
Figure 3.12 – Backlog, sprint 3	29
Figure 3.13 - Web Page, Sprint 3	33
Figure 3.14 - Search Performance, worst case scenario	37
Figure 3.15 - Database structure, sprint 5	38
Figure 4.1 - Use case diagram.....	40
Figure 4.2 - Diagram of the layers and classes in our system and how they relate to each other ..	42
Figure 4.3 - Incident Reporting Form	50
Figure 4.4 - Incident Management Form	51

Figure 4.5 - Incident Dispatching Form	52
Figure 4.6 - Incident Statistics Form	53
Figure 4.7 - Access hierarchy	54
Figure 7.1 - Example bar chart.....	61
Figure 8.1 - Early system draft.....	62
Figure 8.2 - Early state and sequence diagram draft	63

Table of Code Extracts

Code 3.1 - LINQ Select Example.....	20
Code 3.2 – LINQ query to derive report statistics from the database.....	31
Code 3.3 – Custom property example	32

1 Introduction

For a company that offers services and solution for customers, a reliable Incident Management System (IMS) is essential. Some customers even demand it. Customers need to easily be able to contact the service provider if any problems occur. On the company side the IMS could also provide valuable information. For example statistics on how many incidents customers report regarding specific services or subjects. A big issue when handling incidents from different companies is security.

This report will describe one way, the best way within the provided timeframe, of how to implement a complete incident management system in SharePoint. Some challenges we will face include: how to develop web parts for SharePoint since that is a completely new development environment for us, how to secure information without sever performance hits, and how to gather and present statistics about incident reports.

The project touches several subjects in computer science: Web design, database systems, security, C# programming, SharePoint, agile development and more. If you are interested in any of these you should read on.

We have chosen to write this report in a fairly unusual way. The background chapter does not only contain an introduction to concepts brought up later; it also explains how we determined the specification out of the initial problem statement. We think this belongs in the background chapter because it is a preparation, of sorts, to the main project. We could have extracted it as a separate chapter but the initial specification serves as a ground for why we need to explain SharePoint and other subjects. The third chapter is written almost like a journal or logbook. It explains the development process, what we did, how, why and when we did it. Chapter four describes the finished product in detail. The last chapter contains our reflections of the project, what we did well and what we could have done better.

2 Background

The background will mostly consist of the evolution of our specification. Section 2.1.1 will cover the initial requirements. In Section 2.2 we will explain our own interpretation of the project, and in Section 2.3 we will describe the final specification.

2.1 Introduction

This chapter will cover four fundamental parts of the background: the initial specification, SharePoint, existing systems and Scrum. The Initial specification is an elaboration of the problem statement Tieto wrote about the project. Since SharePoint is such a big part of the project we will describe some important aspects of it. Before the project started we investigated if there was any existing system that would fit the requirements. We also took the opportunity to look for features that could be useful for our system. We used a personalized version of Scrum as a development process and some of the words in this report are part of the Scrum vocabulary so we will explain Scrum briefly.

2.1.1 Initial Specification

Tieto wants a prototype of a reporting system that handles incidents (virus outbreaks, intrusion, system errors, etc.). The incident reports can contain sensitive information. This information needs to be properly protected. There need to be access restrictions in order to insure that employees cannot access information outside their scope of responsibility. Traditional issue management systems are not appropriate because of their lack of information protection and segmentation. Incidents are going to be reported by employees of Tieto and possibly companies associated with them. It should be easy to report an incident while still allowing all relevant information about the incident to be conveyed. It should be possible to generate meta-reports containing information about several incidents. This type of report could be used as decision support for company policies.

The key problem in this project is the reporting availability versus internal security constraints. In order to not discourage users from submitting reports it needs to be easy to submit reports without sacrificing the ability to convey important details. The main goal of this system is to allow meta-reports to be generated from submitted incident reports on a need to know basis. Traditional incident and issue management systems do not provide strong enough protection for the sensitive information that Tieto wants to store.

Some initial security concerns are:

- *Encryption of data storage; encrypt sensitive data,*
- *Segmentation of information; separate information so that it is harder to gain unlawful access to sensitive data,*
- *Hierarchical access principles; roles should inherit access rights,*
- *Dynamic report engine; generate statistics about incident reports.*

2.1.2 SharePoint

Tietos intranet is built on a SharePoint foundation so it would be beneficial to implement the incident reporting system as a SharePoint web part. SharePoint [1] is a family of products created by Microsoft. It is supposed to facilitate collaboration, organization and sharing of files and website management.

Web parts are components that in SharePoint can be dragged and dropped to add functionality to a page. They can provide a wide range of functionality, for example news feeds. They are created and behave similar to ASP.NET user controls. A web part consists of four main files:

- Elements.xml; an element manifest that describes the functionality of the web part,
- MyWebPart.cs; is the main code file and startup point [2],
- MyWebPart.webpart; an XML file describing the web part that makes it visible in the web part gallery on SharePoint websites [3],
- MyWebPartUserControl.ascx; an ASP.NET user control.

It is possible to create user configurable web part properties. SharePoint servers interpret some properties as user configurable if you add a few specific attributes to them. Attributes needed for user changeable properties are [4]:

- Category; determines under which category the property in SharePoint will be visible,
- WebPartStorage; determines how the property will be stored, personal means that each user can specify their own value,
- DefaultValue; the default value of the property,
- WebDisplayName; the name displayed to users,
- WebDescription; a description of the property,
- WebBrowsable; is the attribute that tells the SharePoint server that the property is a custom one that can be accessed via a browser.

SharePoint handles the user identification of the user that is logged in and web parts can utilize this.

2.1.3 Existing applications

It is possible to use a common issue management system like JIRA [5] and BugZilla [6] but they are too open for database administrators and do not satisfy our security needs.

MySafeWorkPlace [7] is a website based incident reporting system with anonymous reporting options. It supports a wide range of incident types to choose from when reporting incidents. A few of these are accounting error, discrimination and sabotage. MySafeWorkPlace's focus is too general and does not provide the security required. As a separate website it would be complicated to seamlessly integrate it into Tieto's intranet.

Karlstad University uses a free open source system called Request Tracker [8] to keep track of requests. It is mostly used internally to keep track of requests made to the university's IT department. As the name implies it is designed to provide a way of keeping track of requests. As such there are limited security options and no clear way of setting up hierarchical access. Request Tracker is written in Perl and can only run under a Linux or Unix server. This would make the integration with Tieto's windows based SharePoint servers cumbersome.

2.1.4 Scrum

We had the chance to plan our work hours and workflow. Our supervisor suggested going with scrum, which we were quite familiar with, so we decided to do so. In Scrum it is important to personalize the work process to the group and project. The basic idea of Scrum is to implement iteratively and dividing up the development time into several shorter so called sprints. After each sprint you should have a working, even if lightweight, product. We chose to have two week sprints because that would give us enough interaction with our supervisor and enough time to implement quite a chunk of features. At the end of each sprint there is a demonstration of the application and planning for the next sprint with the supervisors. We, ourselves, also have a retrospective to pin point what we could improve on in the next sprint.

In Scrum, a project specification is separated into small parts called tasks collectively known as a product backlog. An example of a product backlog can be seen in Figure 2.1, where a lower priority number means the task is more important. Tasks should be small in regard to the time needed to complete them.

Priority	Name	Description
1	Visual Studio	Install
50	SQL Database	Gain access to a SQL database.
100	Report organizer UI	(Get familiar with SharePoint) Make simple user interface in SharePoint for outputting information from data storage via search field. A textbox and a button.
150	Client Incident report UI	Make simple user interface in SharePoint for input of data. Ex: Title, description.
160	Incident Management UI	Checkbox to mark field OK
180	Investigate Security Models	Investigate different methods of securing the data storage.
200	Secure storage of data	Secure the data storage without limiting functionality.
300	Make ROUI work with encrypted data	
310	Make CIRUI work with encrypted data	
350	learn how to handle user id	In SharePoint
380	create hierarchy	client, managers, admin (RO).
400	connect user to report	Optional (anonymity available)
410	connect report to managers	N-1. Keep track of who managed the report
420	Expand ROUI	Add functionality; field search,
430	Expand CIRUI	Add functionality; anonymity, categories?(physical incidents, software, other)
	Configurability (web parts)	

Figure 2.1 - Product backlog

Before each sprint, tasks from the backlog are selected into a sprint backlog. Each task is then given a priority and time estimation. The time estimation is relative to another task of known time consumption and is not always translatable to work hours. Tasks in the sprint backlog are also known as stories and the time estimation is called story points.

For each sprint a certain number of tasks are chosen collectively called a sprint backlog. The goal is to choose as many tasks as can be fully completed as possible. Work on a task should not begin if it cannot be completed in the same sprint.

For a more in depth description of Scrum, read “Scrum and XP from the Trenches” [9].

2.2 Specification and design suggestions

Part of the assignment was to establish a software requirement specification with use case and state diagrams. Early drafts of figures in this chapter can be found in Appendix B.

2.2.1 Diagrams

A use case diagram is used to show different users in the system, their roles and interaction with each other and the system itself. We use a flowchart to show how a report travels through the system and which state actions can occur to it.

There are 3 groups of users in the system; Submitters who report incidents, managers who evaluate reports and handle them and administrators who can generate meta-reports. Figure 2.2 is a simplification of how these three groups interact.

Figure 2.3 shows how reports are handled after they have been submitted. First the report is evaluated and more information is gathered if necessary, if gathering information fails the incident remains unresolved. Once all information is available an attempt at handling the incident is made, this can of course fail and the incident remains unresolved.

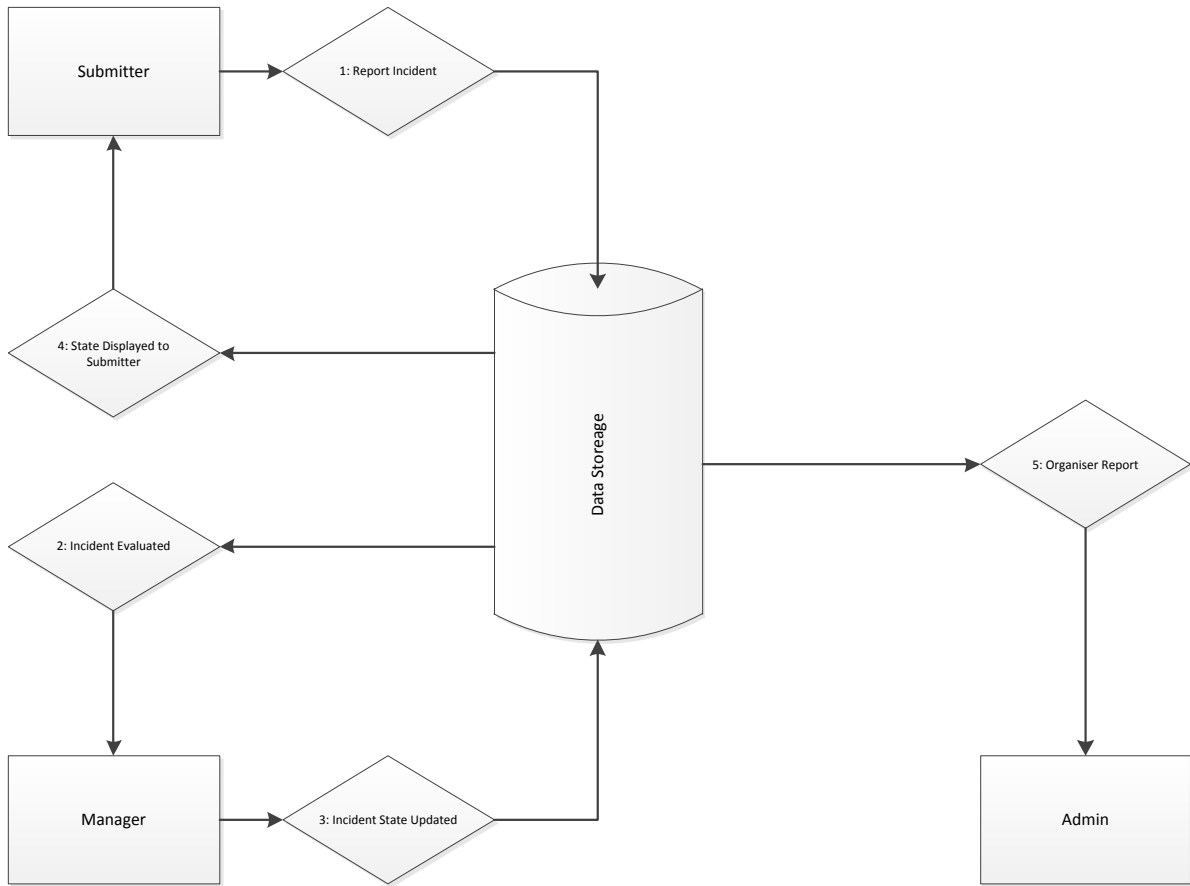


Figure 2.2 - A use case giving a brief overview of the reporting system.

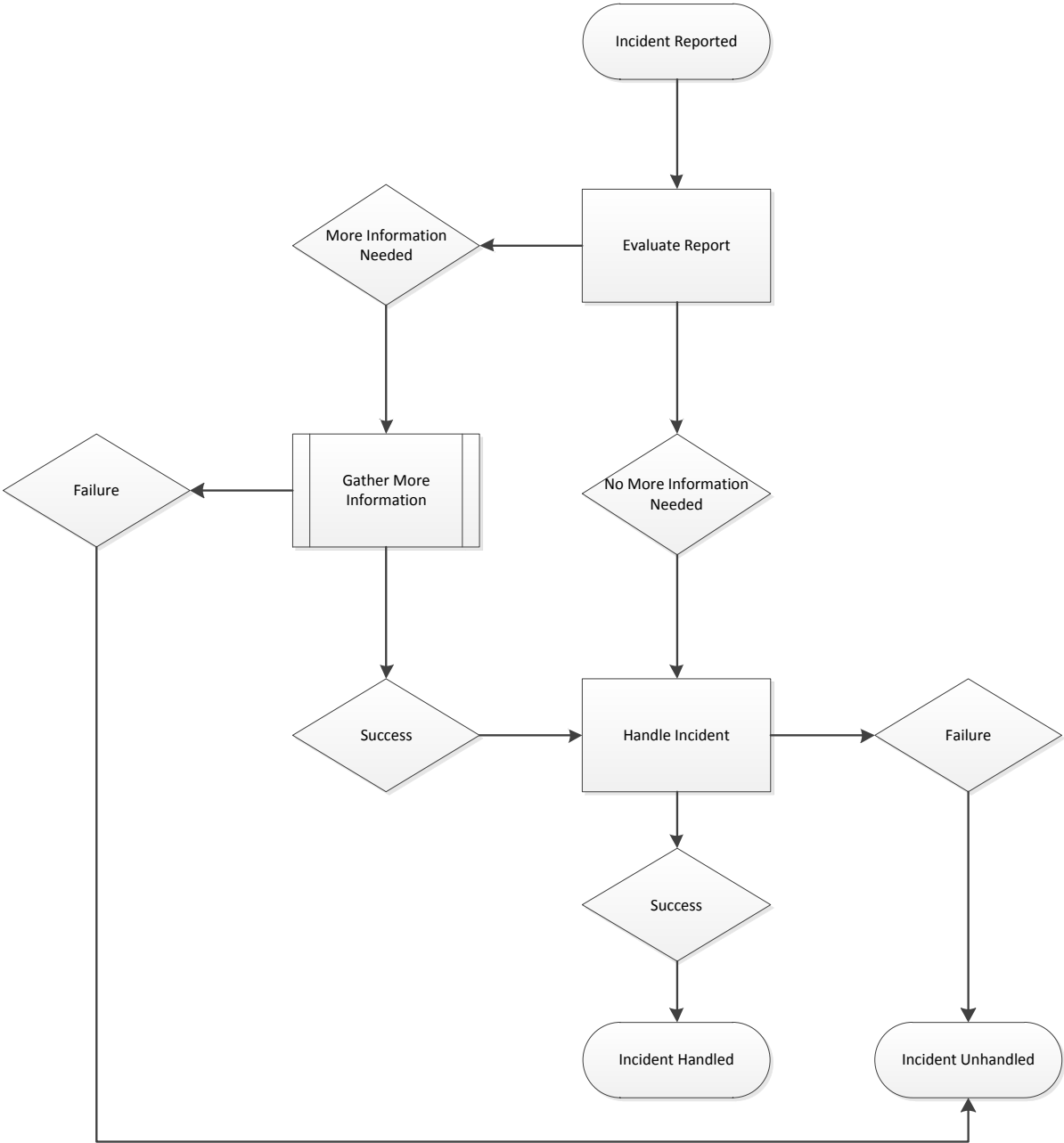


Figure 2.3 - Flowchart of how incidents are handled after they have been reported.

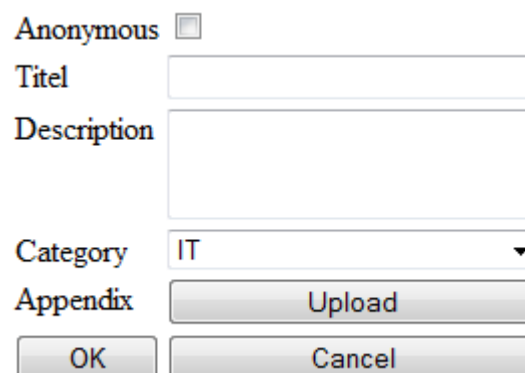
2.2.2 User interfaces

The different user interfaces of the incident management system are the reporting form, the management form and the meta-report (statistics about incident reports) page. The images that follow in this chapter are just simple HTML representations of our sketches and are used as visual aid. They are not supposed to show how the design will look later.

2.2.2.1 Incident Reporting Form

The user interface for incident reporting, in Figure 2.4, consists of a form with several input fields and buttons:

- A title field where the user can enter a subject or short description of the incident with a maximum of 100 characters.
- A text area where the user should describe the incident more in depth.
- A drop-down list where a category can be chosen. A category can be software related incidents; building, legal etc. “Other” should also be a choice if the user do not know or care to choose a category.
- A button for uploading appendixes, for example images that can help clarifying the problem or documents that relates to the incident,
- An option for the user to be anonymous is available to protect users’ identity if they feel they could be in trouble for submitting the incident.



The screenshot shows a form with the following elements:

- An "Anonymous" checkbox, which is currently unchecked.
- A "Titel" text input field.
- A "Description" text area.
- A "Category" dropdown menu with "IT" selected.
- An "Appendix" section containing an "Upload" button.
- At the bottom, there are "OK" and "Cancel" buttons.

Figure 2.4 - Incident Reporting Form

2.2.2.2 Incident Management Form

The incident management form, pictured in Figure 2.5, contains in addition to the fields in the incident reporting form:

- A text area for inputting feedback to the submitter whose main purpose is to request additional information.
- An action drop-down list is used to specify which state the report should transition to next. The different actions are: Read, Supplement Required, Attended To and In Progress.
- Also, instead of an upload button for the appendices there is a view button.

The manager should not be able to edit the title and the description of the report but should be able to change the category.

The form contains the following elements:

- Anonymous**: A checkbox that is currently unchecked.
- Titel**: A text input field.
- Description**: A larger text area for detailed input.
- Category**: A dropdown menu with "IT" selected.
- Appendix**: A button labeled "View".
- Feedback**: A text area for providing feedback.
- Action**: A dropdown menu with "Read" selected.
- Buttons**: "OK" and "Cancel" buttons at the bottom.

Figure 2.5 - Incident Management Form

2.2.2.3 Meta-Report Page

We designed what we called the report organizer pictured in Figure 2.6 with a search field, categories and a button that would show the title, submission date, state, and possibly the date the incident was handled. We would also like to add a graph, using for example something like Google chart API [10] which is a service that draws charts, which shows the number of reported incidents containing the subject over time. This would give a good overview of how many and when incidents with a given search term occurred. It would also be possible to review the graph after new policies are enacted and see if the overall trend has changed.

Software
 Hardware
 Security
 Other

Title	Description	reported time	state	last state change time
example	Blah blah blah	2011-01-02 13:15	Done	2011-01-03 08:30
example	Blah blah blah	2011-01-05 09:15	Waiting for dispatcher	2011-01-05 09:15



Figure 2.6 - Incident Search Page Example, chart generated by Google chart API

2.3 Final Specification and design adjustments

After conferring with our supervisors at Tieto we chose to remove some and change other features because they would needlessly complicate the project if they were left in their current state. The user interfaces from Section 2.2.2 are used but with somewhat reduced functionality. User anonymity is an example of a feature that would complicate things and therefore was removed. If the user is truly anonymous, it would complicate the system. It would be complicated to request more information from them or inform them of the status of the report in a secure way. Another feature we removed is the ability to upload appendices. The storage and encryption of files and the issue of how to handle uploaded files if the report is never submitted would be too complex.

We also needed to rethink design choices. The submitter should not have to take responsibility to make sure the report is in the correct section in the right category. With this feature gone we

cannot make sure the correct manager with the appropriate knowledge and responsibility is automatically assigned to the incident. This leads to the addition of a user class that receives all incident reports and assigns them to appropriate managers, which we decided to call dispatcher. This led us to rethink the states a report can be in detailed in Section 2.3.1. We also started to think about how we are going to store information and how to secure sensitive data in Section 2.3.2. The forms in Section 2.2.2 need some modifications to accommodate these changes. The reporting form will only have a title and description field. The management form will no longer have an anonymity check-box or an appendix view button. We need a new form for the dispatcher. It would be similar to the management form except the addition of a field where the dispatcher can choose whom to assign the report. The incident statistics page will need more advanced filtering choices, as advanced as we can make them at least.

2.3.1 Report states

The state diagram in Figure 2.7 contains all the states we envisioned was needed in our system. It also shows what actions can cause a transition to another state. This is described in detail in the following list:

- **Waiting for dispatcher;** is the state a report is in when it is pending evaluation by a dispatcher. When it has been evaluated it can transition into five different states Closed, Undoable, Assigned to manager or Info required.
- **Info required;** this state allows the submitter to supplement the report with more information. Once more information is submitted it changes state to Waiting for dispatcher or Assigned to manager depending on if the request came from a dispatcher or manager.
- **Closed;** reports in this state describe a problem that does not exist.
- **Undoable;** this state means that the problem detailed in the report cannot be fixed.
- **Assigned to manager;** this state means that the report has been assigned to a manager who now is responsible for solving the situation. It can then transition into five different states Closed, Undoable, Pending, Done or Info required.
- **Pending;** this state means that the problem is about to be solved.
- **Done;** this is the state that a report that has been solved transitions into.

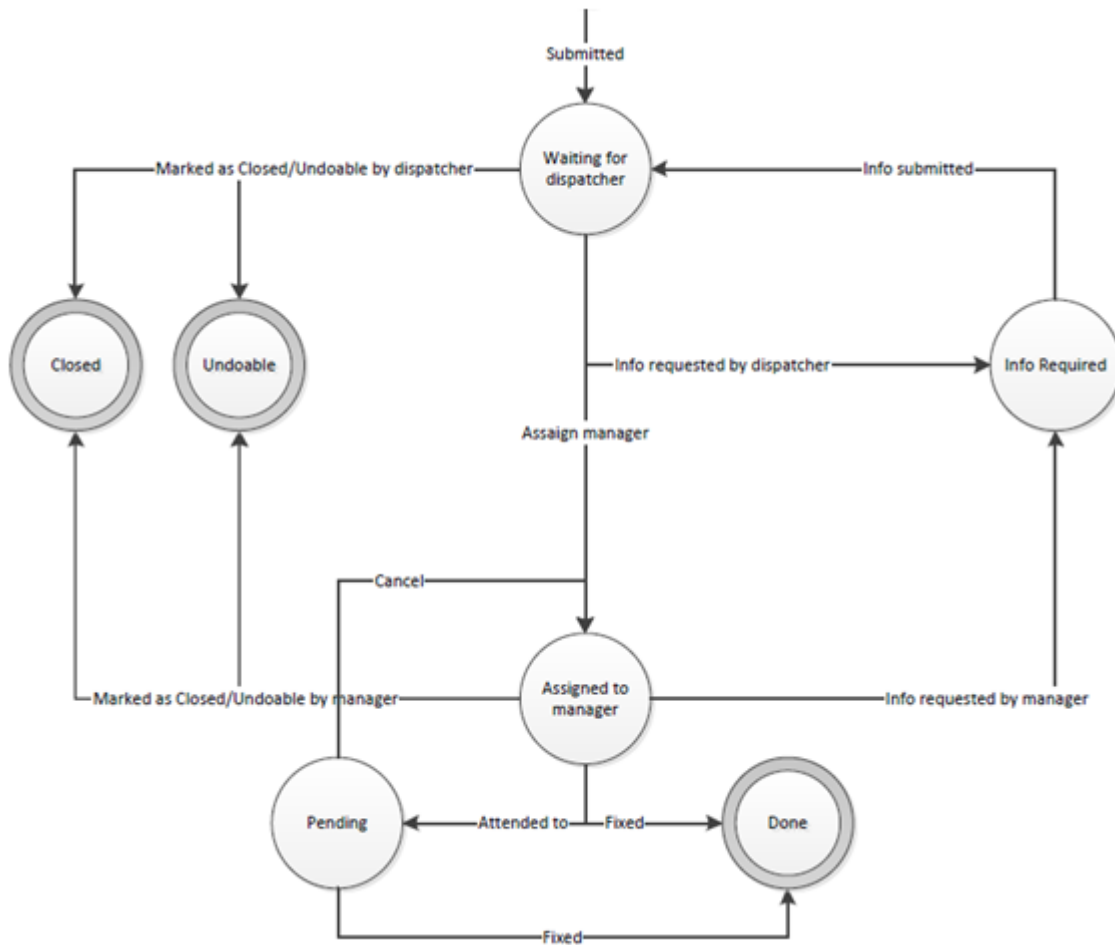


Figure 2.7 – Report states

Closed and undoable are very similar and could very well have been a single state, however we choose to have two states as to be able to provide better filtration capability. The same can be said of pending and done.

2.3.2 Database system and Security

In addition to information about reports and their connections to submitters and managers, a history log needs to exist that stores information about each state a report has transitioned through.

There are some security features that are essential to the system. For example: a user should not be able to read other users' incident reports because the users can work for different companies and their reports could contain valuable information. Managers should not have access to other managers' reports because they may contain information that is outside of their scope. The dispatcher however should be able to read every report because he or she needs to be able to assign it to managers. So the system needs a hierarchical access principle, where the person at

the top has access to everything under it but the different branches do not have access to other branches.

3 Project Development Process

In this chapter, we will describe how we planned our project in Section 3.1. We will document each sprint starting in Section 3.3. Before that, Section 3.2 will introduce the sprint documentation.

3.1 Time planning

We called the first time portion of the project *Pre*; one week where we thought up a specification for the project, described in detail in Chapter 2. The specification was later improved on after a meeting with our supervisors at Tieto at the end of the *pre*-period. After this followed five sprints consisting of two weeks each. At the beginning of each sprint was a planning event where the sprint backlog was created. The sprints all ended with a demo event where we demonstrated the product so far. The last time portion was called *Post* where we finalized the documentation about this project.

In order to keep track of what was supposed to be done each week we created a simple HTML page pictured in Figure 3.1. This page contains a list of all weeks that are part of our planning. If a week is clicked a list of all weekdays are displayed along with any meetings or other important events for those days. The weeks are also paired into sprints. If a sprint is clicked, its backlog is displayed as seen in Figure 3.2. In the sprint backlog we color coded each story; green means it is completed, yellow means it has been started and red means it has not been started. The story points for each story are also listed, in parentheses after the stories name.

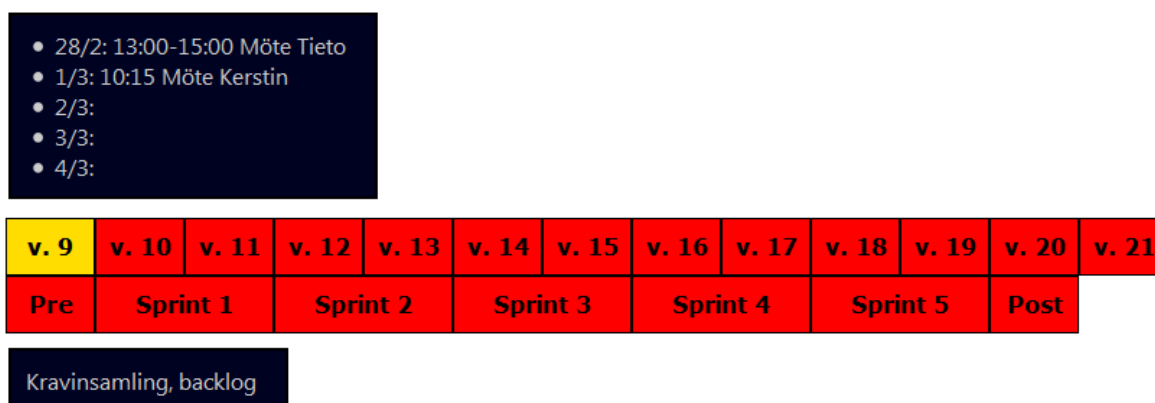


Figure 3.1 - Time table with focus on "pre"

3.2 Sprint introduction

The sprint documentations will consist of four parts that will explain the work process of each of the sprints. They will begin with a brief introduction of what we planned to achieve during the sprint. Then they will continue more in depth with a detailed overview. There we will explain what was done, how it was done and why we did it the way we did. This section is followed by a results part in which we will describe concrete features of the application, mostly focused on features added in that sprint. Lastly a retrospective view of the sprint will bring forth realizations and learning experiences. Hopefully this way of writing will convey the evolution of the project to the reader.

3.3 Sprint 1

In the first sprint we decided to install all tools needed and to get familiar with developing SharePoint web parts. To get familiar with web parts we created the different user interfaces, but not all of them could do something useful. We wanted at least one web part that was able to add information to a database, and one that could perform searches and retrieve information. The complete sprint backlog can be seen in Figure 3.2.

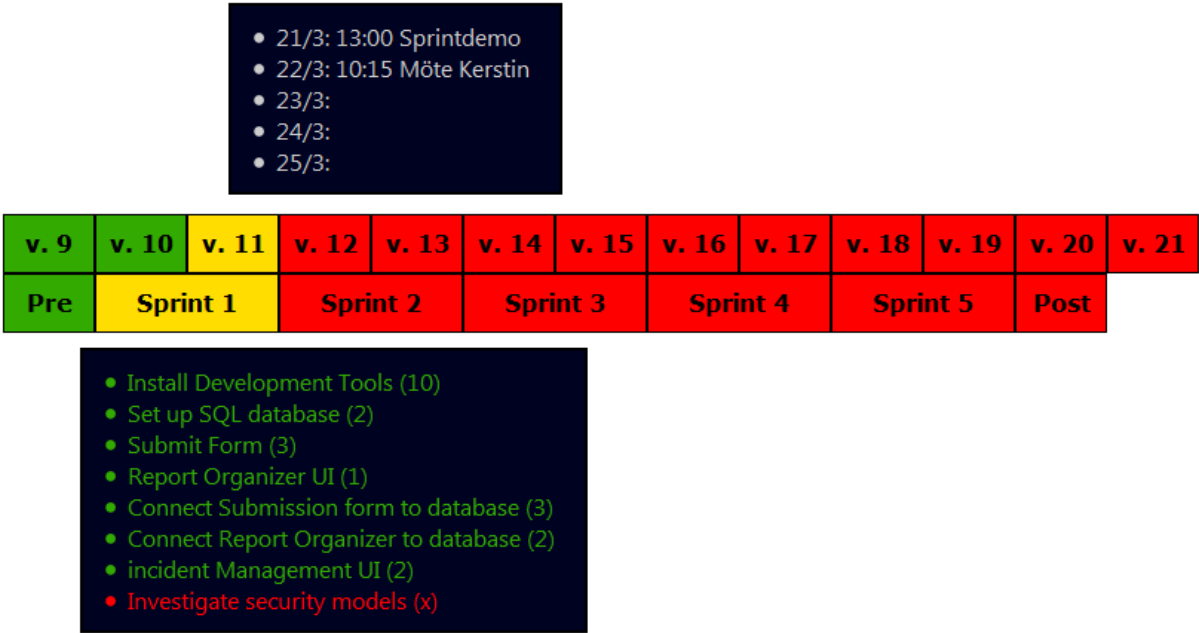


Figure 3.2 – Backlog, sprint 1

3.3.1 Overview

The first story in the backlog was to install development tools. The development tools we chose to use were Microsoft Office 2010, Visual Studio 2010, SharePoint 2010 Server and Microsoft SQL 2008 Server. The first computer that Tieto provided us with only had Microsoft Office 2010 installed so the first order of business was to install the rest of the tools. The computer ran Windows 7 32-bit and SharePoint 2010 Server cannot be installed on a 32-bit Windows installation. A series of attempts were made to bypass this, we tried to install SharePoint 2007 Server but it only supported Windows 2003 Server. We made a few attempts to get around this but only made limited progress. When we finally managed to get it installed, Tieto provided us with another computer with a Windows 7 64-bit installation. With the 64-bit version and the help of a guide from Microsoft's website [11] all the installations were done with no problems aside from a few performance issues.

The second story was to install and setup a database. We opted to use an SQL database to store data. An SQL database was chosen because of the flexibility available, such as the wide range of data types that can be stored. Microsoft SQL Server 2008 was the one we chose. It was mostly because it was included with the SharePoint installation. It was determined through discussions with our supervisors at Tieto that reports need titles, descriptions and it should be possible to connect reports to managers. Managers should be capable of writing comments to submitters and to change the state a report is in. A log is needed to keep track of each time a report changes its state.

To accommodate these requirements we determined a solution where the database consists of four tables handling users, reports, comments and states would be most convenient. The Users table stores the ID of users together with their user roles describing what security access the users have. The Reports table stores unique identifiers for each report and stores titles, descriptions and a reference to users who are manager. The Comment table should contain comments, or feedback, and references to reports. It acts as an extension of the Reports table. We thought it would be a good idea to separate this data because all reports will not always have a comment. It is unnecessary to have fields that often are empty. The states table stores references to reports, the new states, the time the reports changed their states, the users who caused it to happen and a unique id. Figure 3.3 shows how we implemented the database. PK means that a field is a primary key and FK means that it is a foreign key. Primary keys are used to

ensure that each tuple (unordered set of data) in the table is unique. Foreign keys are used to reference primary keys of other tuples.

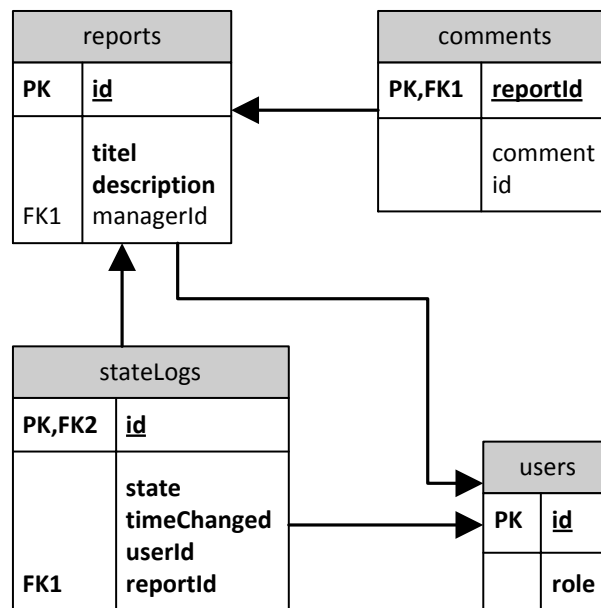


Figure 3.3 - Database Structure, Sprint 1

To get used to the development environment and have a way of adding data, which we could later retrieve, to the database we started with the incident submission web part.

Figure 3.4 shows the design for the submitter form that was derived from Figure 2.4. The appendix and category fields were omitted because we wanted to start out simple and get used to SharePoint Web Parts. As the figure suggests the first field is the title field and can only hold hundred characters. The second field is the description field which does not have any limit of the length of its content.

Incident Reporting

Title, subject or a short description. (Max 100 Chars)

Describe the incident here.
* list item

Submit

Figure 3.4 - Submit Form

To be able to retrieve data from the database we implemented the meta-report web part. Figure 3.5 shows the meta-report page that provides rudimentary search functionality. A search term can be entered and any report that contains that term as a substring will be listed. Each row in said list has a link that if pressed will show the title and description of a report. To get the result shown in the figure below we first pressed the search button to bring up the report table. We then clicked the *show* link to bring up the description underneath.

Report Organizer

Show all reports containing 1 Search

Title	
test1	show

test1

description1

Figure 3.5 - Meta-Report Form

We designed a manager/dispatching web part to visualize how it could look in the end but we did not add any functionality to it. The dispatching form in Figure 3.6 allows reports to be processed before they arrive at a manager. A text field named comment is provided so that a dispatcher can give feedback to submitters. There is also a list that will allow dispatchers to

assign managers to reports. This form also acts as the manager form but if you are logged in as a manager the *assign to* list box will not be shown.

The screenshot shows a web form titled "Dispatching". The form contains the following elements from top to bottom:

- The title "Dispatching" in blue text.
- The identifier "test1" in bold black text, followed by "description1" in a smaller font.
- A label "Comment:" followed by a large, empty rectangular text input area.
- A label "Assign to:" followed by an empty list box with a vertical scrollbar on the right side.
- An "Action:" label, a dropdown menu currently displaying "Request info", and an "OK" button.

Figure 3.6 - Dispatching Form

Any differences between the current design and the initial design are because the current design is not complete. It will continue to evolve and the functionality in the two should converge as the sprints go on.

The connection between the data layer and the graphical user interface layer is established through a LINQ to SQL Classes-layer. LINQ stands for Language-Integrated Query [12] and is a really useful resource in C# .NET. It provides an easy way of searching and sorting collections.

LINQ to SQL Classes is a small framework inside Visual Studio that makes it possible to work against a SQL database in an object oriented way. Each row in a database table is interpreted as an object with the same attributes as the row and also provides direct access to other rows referenced by foreign keys; both to and from that object. LINQ to SQL Classes provides a class that gives access to the interpreted objects of all database tables that has been chosen for use with LINQ to SQL Classes. LINQ allows lambda expressions that look and behave very similar to SQL queries and are translated into SQL by LINQ to SQL Classes when the expression is run. Figure 3.7 shows how LINQ to SQL Classes interprets the database pictured in Figure 3.3.

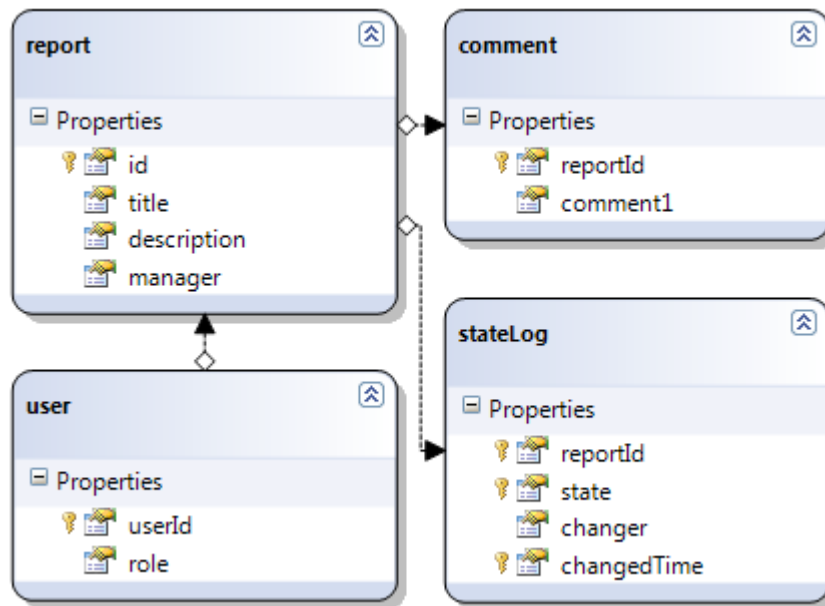


Figure 3.7 - LINQ to SQL Classes interpretation of a few tables from a SQL database

For example if you want to get a collection of reports that contains a specific search term the LINQ query would look like the one in Code 3.1.

```
var result = from report in database.reports
             where report.title.Contains("term")
             select report;
```

Code 3.1 - LINQ Select Example

3.3.2 Results

On the local SharePoint web page on our development computer you can now drag out three custom web parts on any SharePoint page; *Incident reporting* for submitting reports to the system, *Report Organizer* for searching, and *Dispatching* for assigning incidents to managers. In Figure 3.8 these web parts are shown laid out after each other for the convenience of not having to switch page to see all of them.

With the incident reporting web part you can enter a title and a description of an incident. When pressing submit it will be stored in a database. With the Report organizer one can search through the database and the report organizer will display a list with the title of all reports containing said search word together with a link to display the report in a more detailed view. The Dispatching web part is just a design and has no connection to the database so far.

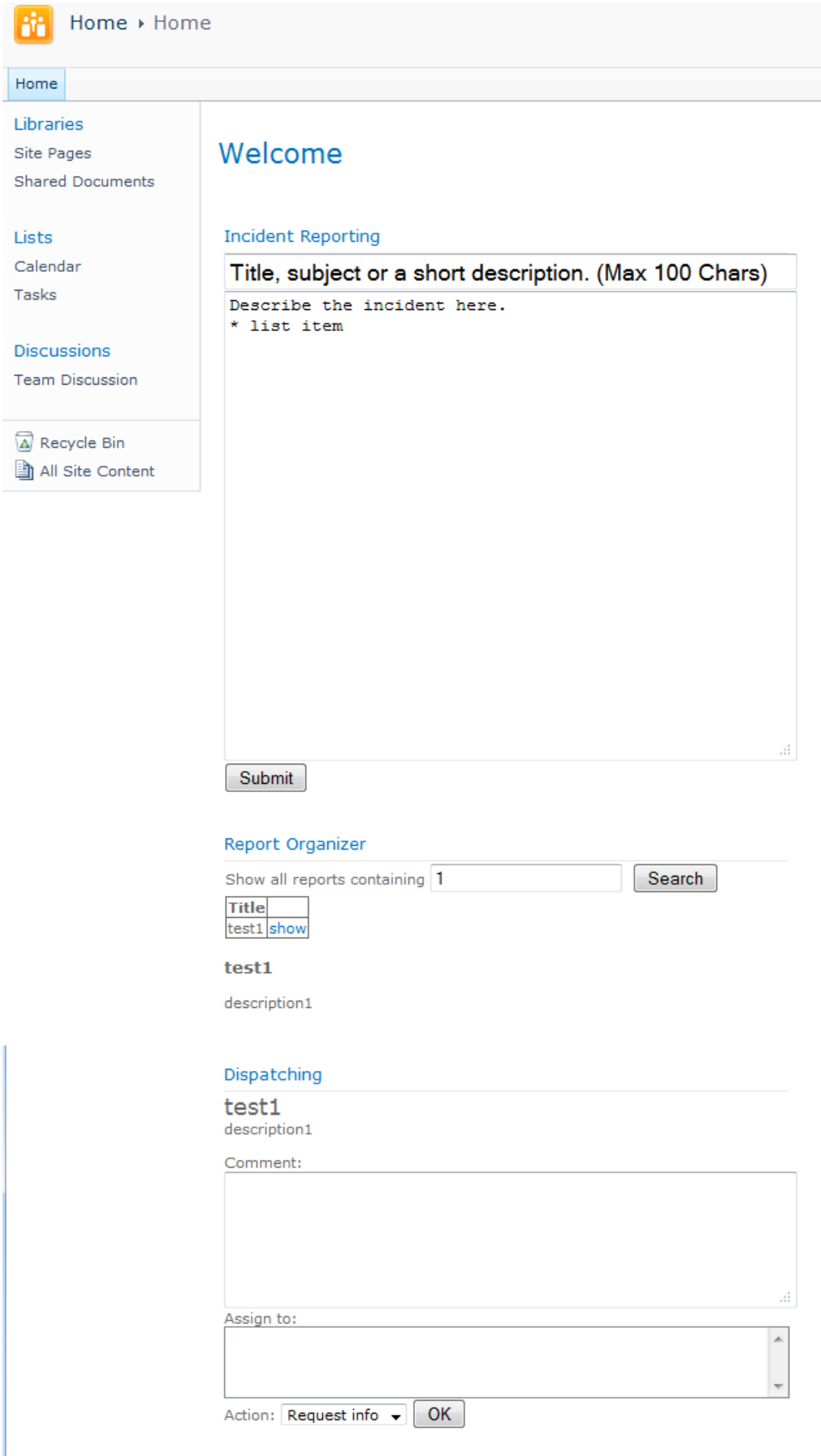


Figure 3.8 - Web page design

3.3.3 Retrospective

Two days were wasted trying to get first SharePoint 2003 then SharePoint 2007 to run on the 32bit version of Windows 7. By the time we were able to get it to run we had already been given a new workstation with the 64 bit version of Windows 7. On the new workstation we could install SharePoint 2010 without much trouble, though we needed to install quite a few prerequisites manually because the installer that was supposed to install them only works on Windows 2008 R2 Server. We used the instructions found on MSDN [11].

SharePoint Server was never intended to run on an average workstation. The SharePoint and Microsoft SQL 2008 Express server consumes up to 3GB of RAM. The SQL server was part of the SharePoint installation. We were given a workstation with 4GB of RAM and that was barely enough to run SharePoint alongside Visual Studio. Remote debugging was not available for the SharePoint Server so running the server on a separate machine was not possible.

Because of this unforeseen problem we realized we would not have time to investigate how to secure the data storage, which was the last and remaining item in our sprint backlog. This taught us an important lesson; Setting up a development environment has a tendency to take more time than anticipated.

We were not entirely pleased with the performance of the development machine so we decided that we would, in the next sprint, move the entire development process to a different computer.

3.4 Sprint 2

In the second sprint we chose to have our main focus on investigating and implementing security for reports. Secure the storage of reports and setting up user access restrictions. Some time was spent further investigating existing systems to see if they had anything we could get inspiration from. We also separated a lot of code away from the graphical user interface.

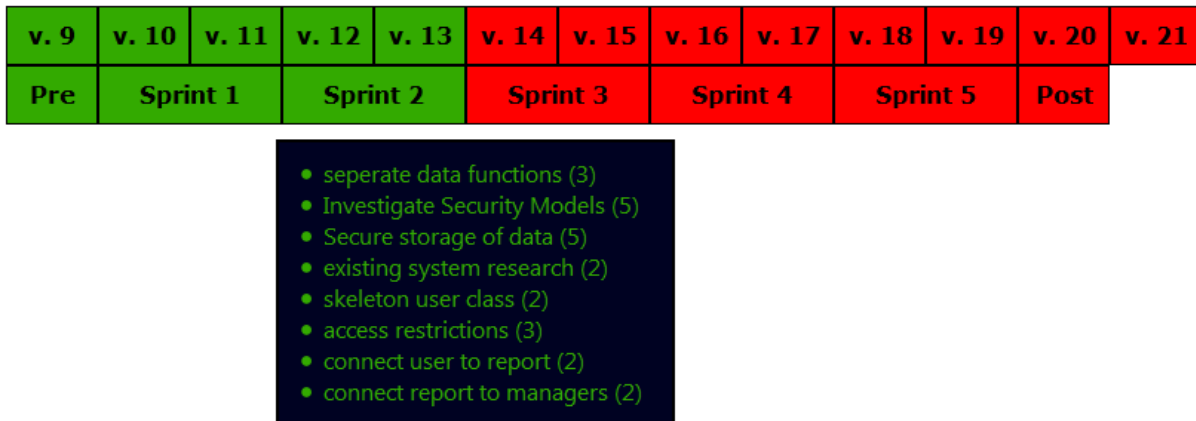


Figure 3.9 –Backlog, sprint 2

3.4.1 Overview

We brought our own laptop which we installed Windows 2008 R2 Server on. This computer had access to 512 MB more RAM and a more powerful processor: a modern 2.4 GHz dual core processor versus an old 3.2 GHz single core processor. When we did this we decided to rewrite the project from the ground to create a more stable foundation to build on. We separated the graphical user interface from the data accessing and processing parts of the program. We split the Visual Studio project into two, one that housed the web parts and one with everything else as a dynamic-link library (DLL). This file led to permission problems, it was solved by simply changing a setting in how a DLL file was deployed. We then started to introduce users to the system.

The code in the project was structured into five different layers:

- The Graphical User Interface Layer; formats user input into a form which the manager layer understands and formats the output from the manager layer into a form users can understand,
- The Manager Layer; provides error and validity checks on top of what the accessor layer does,
- The Accessor Layer; provides the capability to create and retrieve data from the data storage,
- The LINQ to LINQ Classes Layer; translates LINQ queries into SQL and provides a way to interact with the database in an object oriented way,
- The Independent Layer; is a heap of loosely connected classes, mostly container classes,
- The Database Layer; consists of an SQL database.

We were interested about what kind of incident management Karlstad University was using, if any. We visited the IT department and found out they were using Request Tracker. More information about Request Tracker can be found in the background chapter.

We introduced the concept of users by creating a user class that returns the currently logged in user. Currently it always returns the same user. It was done this way because we needed users but did not want to spend much time implementing them and we did not have insight into how Tietos intranet handles users. The addition of the user class made it possible to add several user based features. We added access restrictions to web parts some users should not be able to read. We also connected submitters and managers to reports in the database. Users are connected to reports through the state log. When a report is created a state log item is also created. So to get the creator of a report it is only a matter of checking which user caused the first state log item for a report to be created.

There are four user classes; administrators, dispatchers, managers and users. They are hierarchical in nature so that administrators are considered as dispatchers and dispatchers as managers. Access restrictions have been implemented so that administrators can view the Incident Statistics form, dispatchers can view the Incident Dispatching form, managers can view the Incident Management form and users can view the Incident Reporting form. If a user tries to access a form it is not supposed to, the user will be redirected to another user control named NoAccess. This user control displays text explaining that the user does not have access to that form.

There are several ways to ensure security of data. You just have to weigh time, both in the sense of work hours and application responsiveness, against performance. There is a vast amount of available encryption algorithms so we chose for our own convenience to limit ourselves to the ones in the .NET Framework 3.5. There are two main groups of encryption methods; One-way, also known as hash, and two-way encryption. Hashes are not possible to decrypt so we could not use them because we need to be able to read the contents of the submitted reports. The algorithms that were most interesting to us were advanced encryption standard (AES), data encryption standard (DES), and Triple DES. DES is a 56-bit encryption method [13]. Triple DES is DES applied up to three times which is equivalent to 56, 112 or 168-bits [13]. AES can use 128, 192 or 256 bits in the key [14]. The keys are used when encrypting and decrypting. Longer keys provide better security, because they provide more possible key values.

We chose to use AES because of the higher security option. We made a simple performance test to determine if AES encryption was a viable option. The test revolved around splitting up a large text file into different sizes and then looking at how much time was required to decrypt. The test showed that decryption time was reasonably fast, so we started implementing encryption with AES. At first we tried to store the encrypted byte arrays as strings. This was done by converting bytes to characters and merging them into strings. This did not work as intended; in short the bytes were changed when they were converted to characters. In the end we solved this by storing the encrypted byte array as varbinary in the database. A varbinary column stores variable-length binary data [15].

AES encrypts data by using a key and an initialization vector (IV). We randomized sixteen integers between 1 and 255 for the IV and thirty-two for the key with an online randomization service called Random.org [16]. We stored the result in a byte array.

These changes and additions resulted only in a few changes to the database structure seen in Figure 3.10:

- A new table named categories has been added. It consists of two fields, an id and a name.
- The reports table now has an id reference to the categories table.
- The description and feedback fields of the reports table have been changed from text to varbinary(max).
- The title field of the reports table has been changed from nchar(100) to varbinary(100).
- The comment table is now a field called feedback in the reports table. This was done because we did not think that it would be common for a report to not have feedback. At least not enough to warrant a separate table.

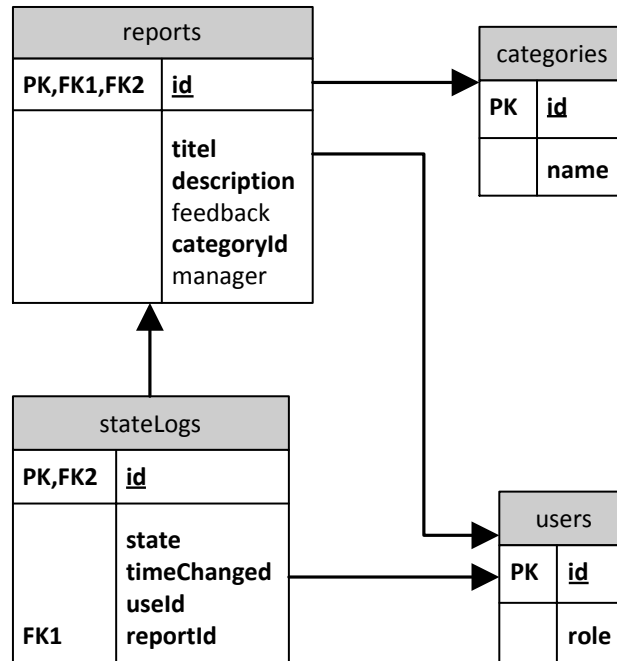


Figure 3.10 - Database structure, sprint 2

3.4.2 Results

Reports are now stored encrypted in the data storage. In Figure 3.11 the updated user interfaces can be seen. The submitter form titled Incident Reporting now allows submitters to choose a category by selecting one in a drop down list when reporting an incident. The dispatcher form titled Incident Dispatching now allows dispatchers to assign reports to managers via the managers' drop down list and change the state a report is in via the actions drop down list. The meta-report page titled Incident statistics functionality still remains the same. The Incident Management form does not have any functionality.

If users try to access a form which is outside their scope they will only see a text explaining that they cannot access said form.

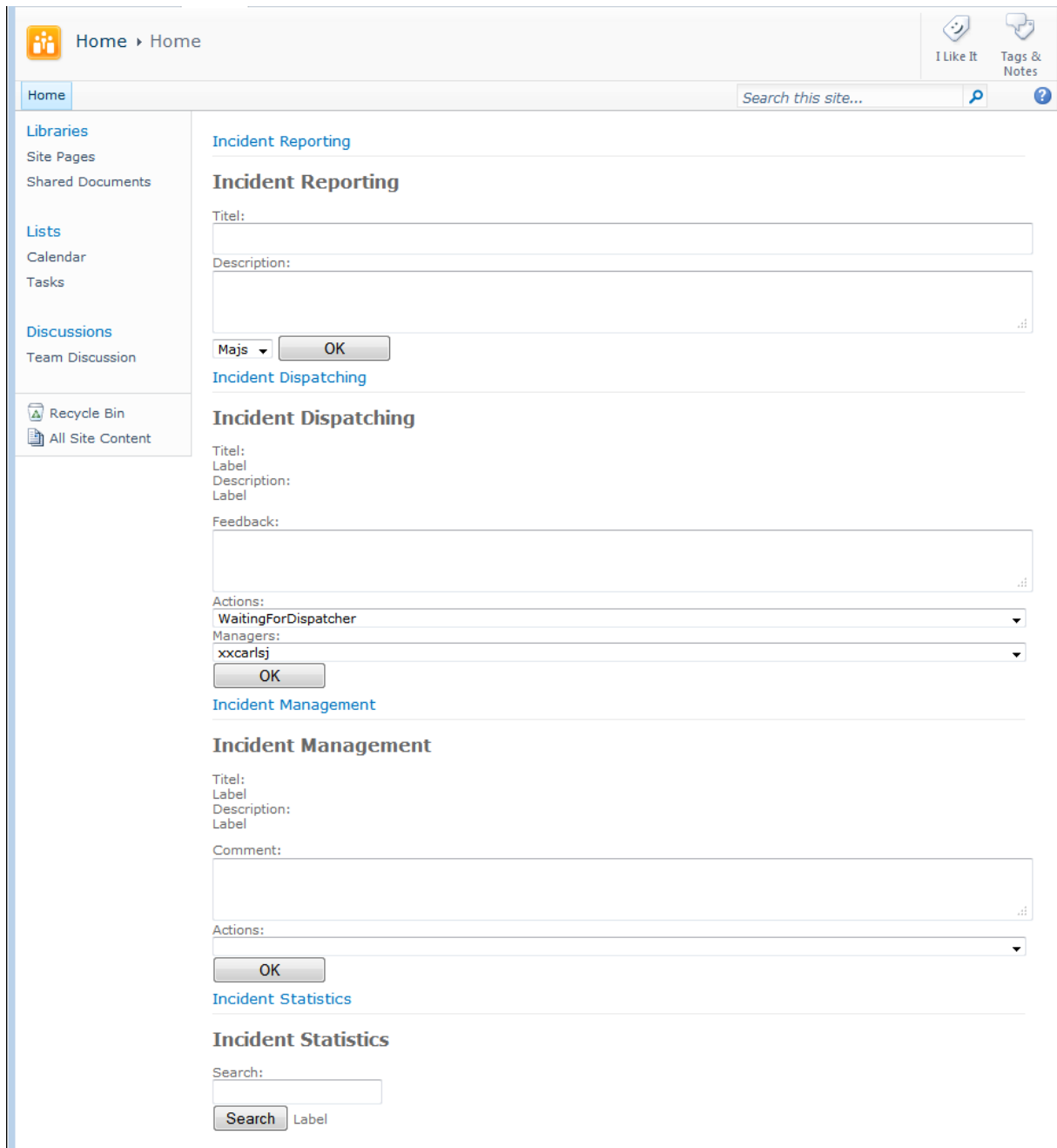


Figure 3.11 - Sprint 2 Page

3.4.3 Retrospective

Moving the development from the workstation computer to the much faster laptop and the complete rewriting of the code resulted in some irritating problems. When we tried to keep the main bulk of the code in a DLL file separated from the user interface we got a series of problems. The first was that the DLL file simply was not deployed with the rest of the code. After some digging around we found out that it is possible to manually specify files to deploy. This manual way had two different options for where to deploy the DLL file, Global Assembly Cache or Application. At first we tried Global Assembly Cache deployment but that made it problematic to

use the DLL file in user controls, because you then needed to provide an absolute path to the DLL file. So we changed to Application deployment. This made it possible to easily use the DLL file in user controls. This led us to problem two; we got a lot of access permission errors when using LINQ to SQL Classes. DLL files deployed to Application are less trusted than ones deployed to Global Assembly Cache. We solved this by not using the DLL file in user controls and deploying to Global Assembly Cache. If a DLL is to be used in user controls and in classes it would have to be deployed both to the Application and Global Assembly Cache.

When we started to encrypt and decrypt reports we tried to save the resulting byte arrays to strings. This was not a good idea due to C# and SQL Server tendency to error correct and pad strings. This was solved by us discovering that SQL Server has a data type called *varbinary* that allows storage of byte arrays. Another related issue was that the SQL Server padded *nchar* fields with whitespaces. This resulted in strings with a lot of whitespaces where there should not be any. This was solved by changing to *nvarchar* which does not pad.

We separated the key away from the code by hiding it inside a file. It worked aside from some strange behavior where the file was locked by another process. This caused the web parts to crash. We did not find out what locked the file. Nothing in our code should have locked the file. We only opened the file with read access which should not lock it.

All in all, we were really pleased with the progress we made in this sprint. Every task we put in the sprint backlog was completed. We expect to be able to run test cases in or after the next sprint.

3.5 Sprint 3

Our main focus for the third sprint was to implement three different news feeds for the dispatchers, managers and users. The Overview will explain how we separated the key, implemented the news feeds, expanded the search engine, and how configurability was achieved in web parts. The goal is that after this sprint, we will have a system in which we can run test cases. The backlog for sprint 3 can be seen in Figure 3.12.

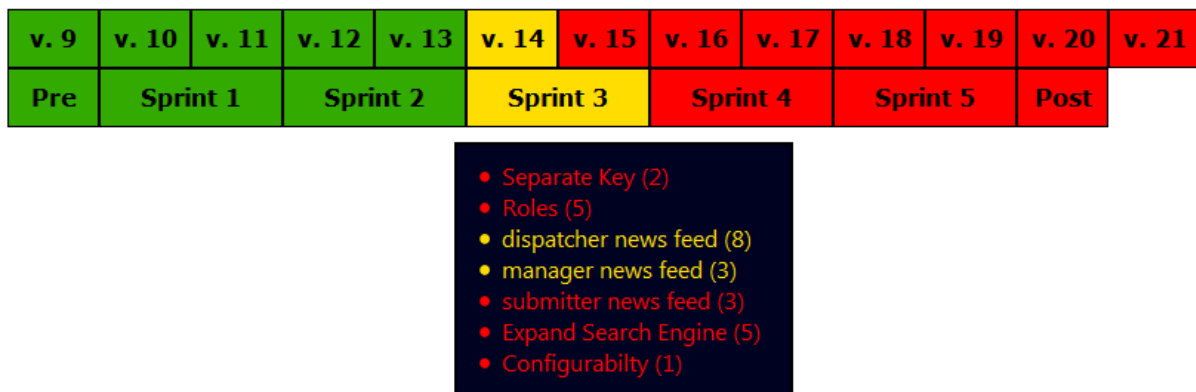


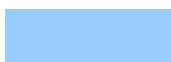
Figure 3.12 – Backlog, sprint 3

3.5.1 Overview

We began the sprint with the separation of the master key and initialization vector from the code into files. We created a function that could pick them out of the file. It was done by reading a byte array from the file. The function takes an offset value to determine where to begin reading the file. This is made as a security measure because you need the database, the code and the file. You cannot decrypt the database with just the key file because you do not know where the key is or how it is generated from the file. Furthermore the stored key is modified by using data from the database before it is used.

When we implemented the news feeds we created a server control, hereafter referred to as ReportNews feed, which would print out any list of reports we sent to it. A server control [17] is an object that is rendered as HTML code when called and can be used in ASP.NET pages. Our server control has a click event that registers each time a report is clicked. ReportNews feed is used for all four feeds; dispatcher, manager, user and admin feed. All the news feeds behave the same, when a report is clicked said report is loaded and shown except for the admin feed which does not have any clickable links. Reports in the news feed are color coded based on what state they are in. This is done to make it easy to see which state a report is in. The states are color coded as following:

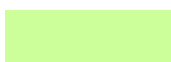
- AssignedToManager; a light blue

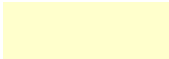
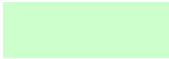




- Closed; a light orange



- Done; a light lime



- InfoRequired; a light yellow

- Pending; a light green

- Undoable; a light red

- WaitingForDispatcher; a light purple


Done and pending are similar colors as both states imply almost the same thing. “Pending” means that the incident is being fixed but that it might take some time. It acts as a confirmation to the submitter that someone has read the report and the problem will soon be solved.

To track which report is clicked we created a class named ReportLinkButton that inherited from the built in LinkButton server control and overrode two functions: RenderBeginTag and RenderEndTag. These functions are called before and after the server control is rendered. In the RenderBeginTag and RenderEndTag we wrapped the link in HTML code to display it in a table. This is easier to show rather than to explain. The end result can be seen in Figure 3.13. The ReportNews feed class instances contain a list of ReportLinkButton.

We expanded the search engine by adding a bar graph that shows how many incidents were reported each month and how many of them were resolved. To derive this from the database we used LINQ, see Code 3.2. First we took the creation time and current state for each report in the reports list. Then we filtered out the reports that were outside the time span of the graph, ordered them in descending order by creation time, and grouped the reports by the month they were created. Lastly we selected year, month, and number of reports submitted and resolved for respective month. We displayed the result in a bar chart generated by Google chart API. An example of a chart generated in the API can be found in Appendix A.


```

var results = from reportInfo in
    (from report in reports select new {
        Date = report.StateLogAuxes.OrderBy(w => w.timeChanged).First().timeChanged,
        State = report.StateLogAuxes.OrderByDescending(w => w.timeChanged).First().state
    })
where reportInfo.Date >= graphTimeStart &&
    reportInfo.Date <= graphTimeStart.AddYears(years)
orderby reportInfo.Date ascending
group reportInfo by reportInfo.Date.Month into month
select
    new MonthStatistics(
        month.First().Date.Year,
        month.First().Date.Month,
        month.Count(),
        month.Count(w => w.State == (int)ReportStates.Done)
    );

```

Code 3.2 – LINQ query to derive report statistics from the database

Configurability was investigated and implemented. The SharePoint server interprets some properties as user configurable if you add attributes to them in a specific way [4]. An example of this is shown in Code 3.3:

- Category; determines under which category the property will be visible under in SharePoint,
- WebPartStorage; determines how the property will be stored, personal means that each user can specify their own value,
- WebBrowsable; is the attribute that tells the SharePoint server that the property is a custom one that can be accessed via a browser.

The result of this would be that a new setting would be shown under settings for our web part in SharePoint.

```
[Category("Report Filter")]
[WebPartStorage(Storage.Personal)]
[DefaultValue(default_showDoneReports)]
[WebDisplayName("Show resolved incidents")]
[WebDescription("Uncheck the box if you do not want reports for resolved incidents to
show up in your news feed.")]
[WebBrowseable(true)]
public bool ShowDoneReports { get; set; }
```

Code 3.3 – Custom property example

3.5.2 Results

All the web parts now have a news feeds with reports. These news feeds allow reports to be clicked:

- In the incident reporting web part it causes a report to be loaded into the submit form. But only if a dispatcher or manager has requested more information.
- In the incident dispatching web part a report is loaded into the dispatching form. It can then be dispatched.
- In the incident management web part a report is loaded into the management form. It is then ready to be managed.
- The incident statistics web part does not have any clickable reports.

The news feeds color code reports based on which state they are in as can be seen in Figure 3.13.

Incident Reporting

Incident Reporting

[Report Incident](#)

Report Feed

Flooding Damage Windows on the 24th floor were impaled by rods and broken.	WaitingForDispatcher	04/13/2011 15:22:34
Window Broken A window on the first floor has collapsed.	WaitingForDispatcher	04/13/2011 15:20:12
Development Tools I cannot access the SQL server with my domain credentials! <i>You shouldent.</i>	Closed	04/13/2011 14:58:34
Broken Window On the 98th floor a window has been broken by a seagul. <i>Sure... :/</i>	Closed	04/13/2011 14:57:21
System Failure The system refuses to start! <i>Your own damn fault.</i>	Undoable	04/13/2011 14:51:31
Sharepoint Need more RAM to properly run SharePoint 2010 Server and Visual Studio 2010. <i>Majs, korv, vete och mjöl!</i>	Done	04/13/2011 14:49:49
Isen smälter Isen ute i älven(Klarälven) smälter! <i>Bah</i>	Done	04/13/2011 12:31:24

Incident Dispatching

Incident Dispatching

Report Feed

Clouds The clouds are blocking the sky.	WaitingForDispatcher	04/13/2011 15:00:33
Annanas It grows wrong!	WaitingForDispatcher	04/13/2011 15:00:51

Incident Management

Incident Management

Report Feed

Incident Statistics

Incident Statistics

Search:

Report Feed

Isen smälter Isen ute i älven(Klarälven) smälter! <i>Bah</i>	Done	04/13/2011 12:31:24
Vatten Det är vatten i floden!?, är du galen?, vadåda? <i>?, whatever..., I dont even!</i>	Done	04/13/2011 14:35:24
System Failure The system refuses to start! <i>Your own damn fault.</i>	Undoable	04/13/2011 14:51:31
Broken Window On the 98th floor a window has been broken by a seagul. <i>Sure... :/</i>	Closed	04/13/2011 14:57:21
Development Tools I cannot access the SQL server with my domain credentials! <i>You shouldent.</i>	Closed	04/13/2011 14:58:34
Clouds The clouds are blocking the sky.	WaitingForDispatcher	04/13/2011 15:00:33
Annanas It grows wrong!	WaitingForDispatcher	04/13/2011 15:00:51

t reports by month
ar from 4/26/2010
ontaining ""

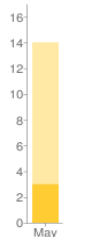


Figure 3.13 - Web Page, Sprint 3

3.5.3 Retrospective

We did not have any major problems in this sprint with the exception of the sprint demo. The sprint demo was delayed for about one and a half week.

At the end of the sprint we were able to run a test case where we simulated the lifetime of a report. The test case was done as following:

1. First an incident was reported.
2. Then a dispatcher requested more information from the submitter.
3. The submitter responds with more information.
4. Then a dispatcher assigns a manager to the report.
5. The manager starts with requesting more information from the submitter.
6. The submitter once again responds with more information.
7. Then the manager sets the report as done.

A feature that was originally scrapped was revived after the sprint demo with our supervisors at Tieto. The feature was the ability to separate incident reports based on departments, locations and customer from each other. This will be described in the next sprint.

3.6 Sprint 4

This sprint's main focus would have been the addition of customers. Customers need to know that information regarding their company does not fall in the hands of someone without permission to touch it. So we need a way to classify reports, users, managers and even dispatchers with access restrictions. Unfortunately we did not have time to implement it because Easter and delays cut the sprint short by more than a week. The addition of a customer would have meant changes to all parts of the system. Instead we decided to explain in detail how we would have implemented the feature.

3.6.1 Overview

We initially planned to include this functionality in the initial specification but it was scrapped at that time. This was done because we wanted to start out with a simple system.

In a company, as large as Tieto, there exist several different departments on varying locations. Customers for Tieto do not want incident reports about them to be visible to affiliates of another customer of Tieto.

This means that each department, location and customer needs to be isolated in some regard from each other. And the whole should only be accessible by a few. However the changes required for this addition of functionality would be quite time consuming. Due to the projected time requirements we chose not to start implementing it.

We would make departments, locations and customers act as groups. In order to read an incident from a group you would need to be a member of that group, or be the submitter of an incident. The submitter of a report should always be able to see it regardless of which group it is in. This is because a dispatcher or manager could move the report from one group to another. If the submitter would have to be in the same group as the report to be able to read it, it would be impossible for a manager or dispatcher to have a dialog with the submitter.

Two new tables would be added to the database: one that would contain information about groups and one that would connect users to groups. The groups table would contain two fields; an id and a name for the group. The table that connects users to groups would need two fields: one that references a group and one that holds a user id. Another addition to the database would be the addition of a foreign key to the reports table that would reference a group.

Each of the four web parts incident reporting, incident dispatching, incident management and incident statistics would need to be connected to groups in some manner. We thought of two distinct ways of doing this. The first would be to make the web parts configurable so that they always connected to a group. This would require a lot of different pages with web parts. The second was to display a drop down list of all the groups a user is connected to. This would allow the user to choose which group the incident belongs to.

Dispatchers and managers must somehow be able to register for a group. There are different ways of accomplishing this, each with varying levels of difficulty:

- A group administrator that manually adds every eligible user to the database,
- Users receive personalized one-time-use-passwords,
- Every group has one password; it would have to be changed fairly often though. This is because it might be leaked; a user can quit or be relocated to another department or location.
- The easiest way would be if the groups somehow could be automatized by using user information the system already has. We do not know what information is stored about users or how to access it so this is a hypothetical solution.

The passwords themselves would not be stored; instead hash values of the passwords would be stored. This in order to protect the passwords of the database would be breached.

The accessor layer would need to be updated so that the new fields are loaded when retrieving and creating objects. The select statements would also need to be updated so that they do not retrieve reports that the current user is not supposed to be able to read.

3.6.2 Results

Reports would belong to groups and administrators would be able to generate statistics based on groups. Dispatchers and managers would only get incident reports that belong to the same groups as they are. Users would only be able to submit reports to groups they belong to but would be able to read their own reports even if they are moved to another group.

3.6.3 Retrospective

Our thoughts was that we would have had time to implement this if the sprint had not been cut short due to scheduling problems.

3.7 Sprint 5

This sprint's main focus was writing this report and generating some performance measurements of the search functionality.

3.7.1 Overview

The main bottleneck in our system is the decryption process. Though it only affects the incident statistics web part. In order to be able to search through a report it has to be decrypted first. This can be quite time consuming and memory intensive when the number of reports goes up.

There are some ways the search process could be sped up. The reports could be filtered before encryption occurs. The filtering could be based on categories and timespan. This would result in that only reports matching a chosen category and timespan are decrypted and searched.

As seen in Figure 3.14 the search time is a linear function of the number of reports the search is performed on. The peaks and valleys are caused by the use of a page file. This test was a worst case scenario where the search term where at the end of each report and each report contained the search term. Each report consisted of a 100 char long title and a 1,000 char long description. The title and description were randomly generated.

The test was performed in a virtual machine. A virtual machine behaves as an ordinary machine but cannot run on its own and has to be hosted by another machine. This was chosen because we had a virtual machine with everything needed to run the test. The machine running the virtual machine did not. The virtual machine was configured to have one processor with four cores and 1928 MB of RAM. It was running the 64-bit version of Windows 7. The virtual machine was running under VMware Workstation 7.1.4. The host operating system was running the 64 bit version of Windows 7. It had 4 GB of RAM and an Intel Core 2 Quad processor (Q6600) clocked at 2.4GHz.

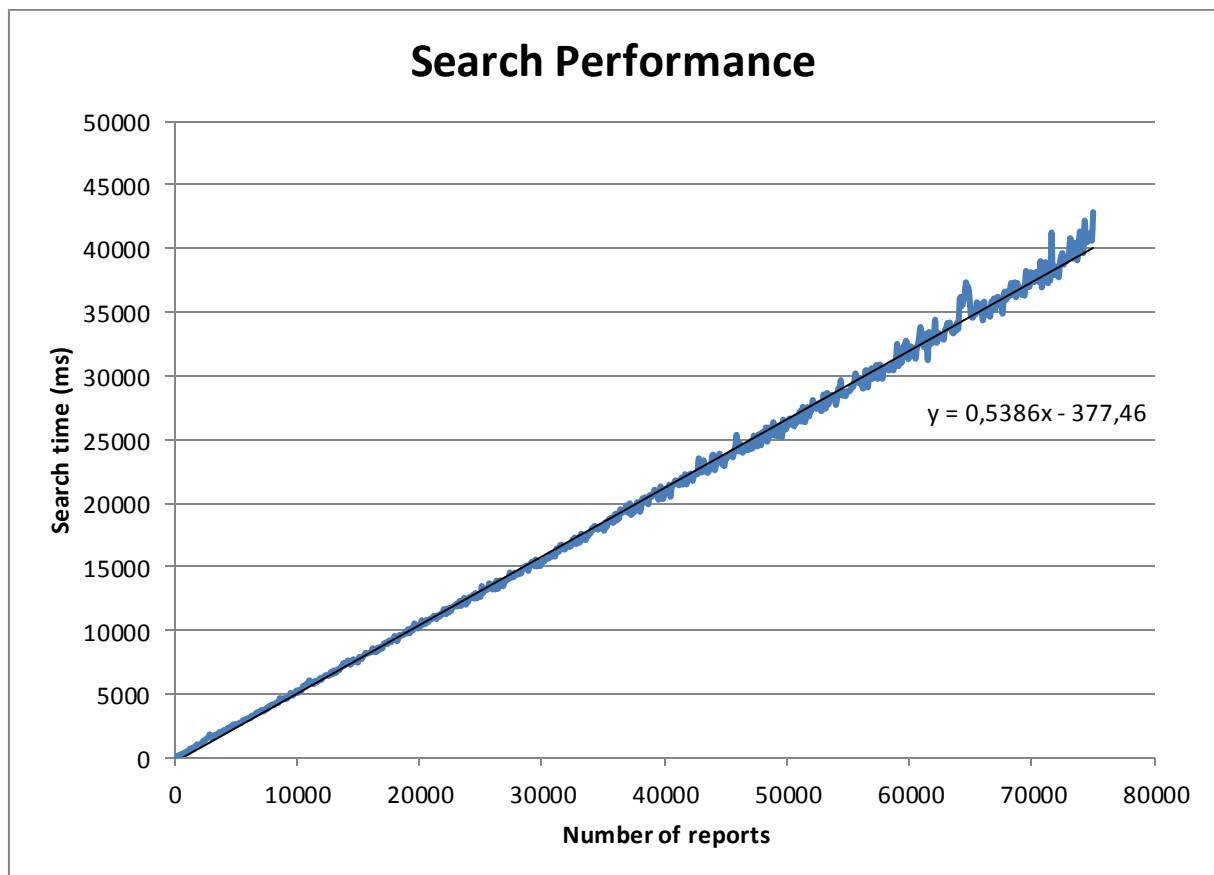


Figure 3.14 - Search Performance, worst case scenario

Currently all reports are loaded into memory at once and decrypted one at a time. This requires a lot of memory compared to alternatives. One alternative would be to only keep a few reports in memory at a time. Another solution would be to filter the reports that are loaded, so that fewer are kept in memory. The memory usage will of course vary depending on how many reports are in the system and how big each report is.

It was discovered that a one hundred characters long titles could no longer be stored in the database after they had been encrypted. About 15 % more in our case, this number comes from

adding the encrypted title lengths from the test in Figure 3.14 and dividing it by the total number of reports. It was solved by increasing the number of bytes that can be stored in the title field of the reports table.

There has been only one change in the database seen in Figure 3.15; the title field of the reports table has been changed from varbinary (100) to varbinary(max). This was done because a title of 100 characters could not be stored in 100 bytes after it is encrypted.

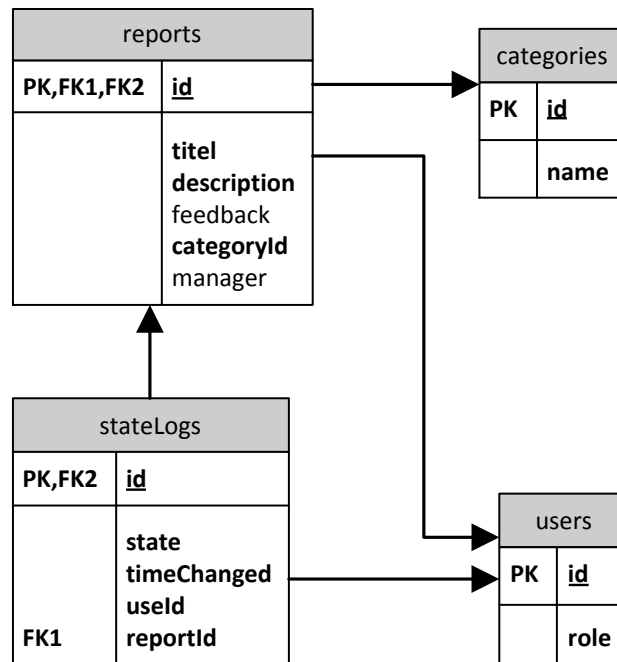


Figure 3.15 - Database structure, sprint 5

3.7.2 Retrospective

We came to the conclusion that our choice of cryptographic algorithm was a good one. It performs reasonably well when searching through a large amount of reports. If performance issues are encountered when the system goes live, the algorithms could be tweaked.

4 Results and evaluation

This chapter is a complete rundown of the finished system. In Section 4.1 we will briefly describe the system as a whole; Section 4.2 will cover the code structure; Section 4.3 will provide screen shoots of the web interface and we will explain what users can do and how; and In Section 4.4 we will describe security aspects such as user access restrictions and data encryption.

4.1 System

Users can report incidents. They can help managers and dispatchers by providing additional details about the incident if requested. Managers manage reports that they receive from dispatchers; solving the incident. Dispatchers dispatch reports that are waiting to be dispatched, by solving the incident themselves or assigning it to a manager. Managers and dispatchers can request information from the submitter of a report to gain more insight into what the incident was about. This can make it easier to manage and dispatch the reports. Administrators can generate statistics from reports. Dispatchers and managers can determine that the incident is not a real problem or that the incident cannot be solved.

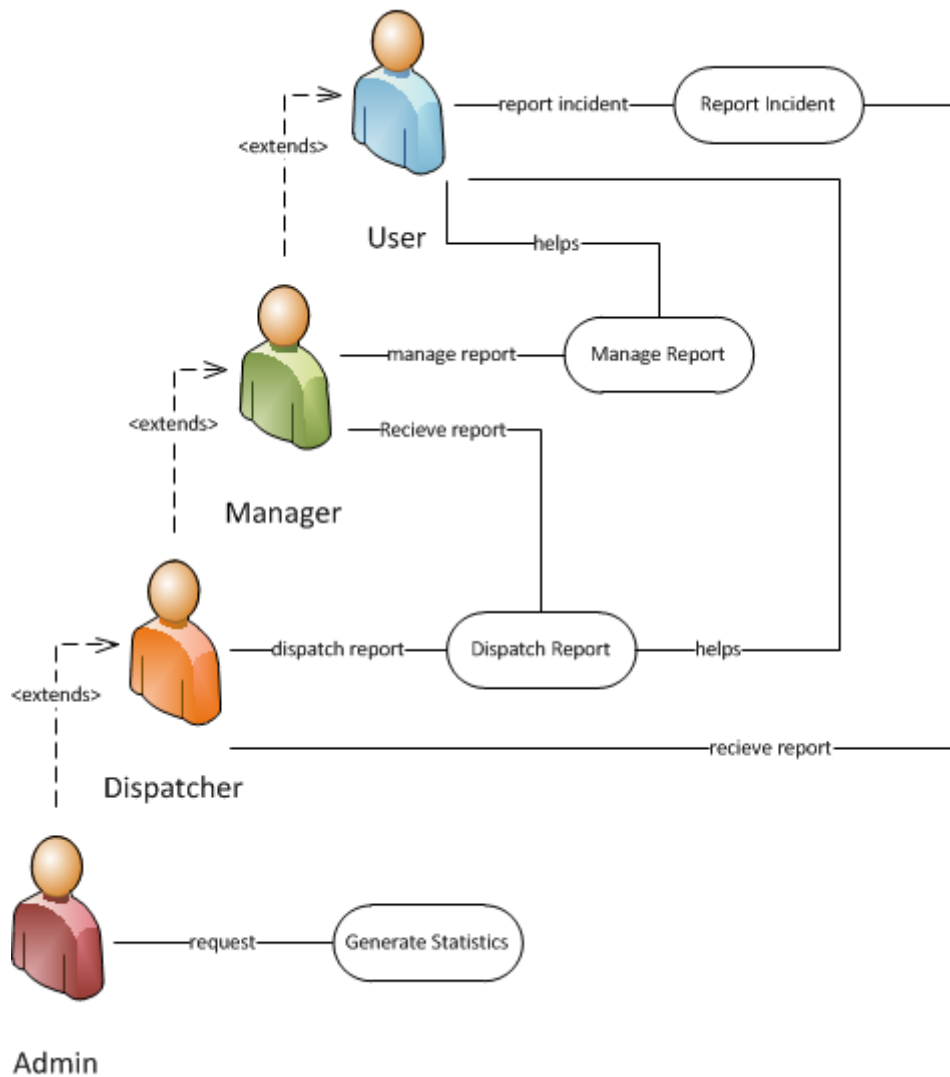


Figure 4.1 - Use case diagram

4.2 Code

We chose to make the Incident Reporting System as modular as we could, to make it adaptable to changes. This philosophy led to the creation of several different layers:

- The Graphical User Interface Layer; formats user input into a form which the manager layer understands and formats the output from the manager layer into a form users can understand;
- The Manager Layer; provides error and validity checks on top of what the accessor layer does;
- The Accessor Layer; provides the capability to create and retrieve data from the data storage;

- The LINQ to LINQ Classes Layer; translates LINQ queries into SQL and provides a way to interact with the database in an object oriented way;
- The Independent Layer; is a heap of loosely connected classes, mostly container classes,
- The Database Layer; consists of an SQL database.

The relationship between these layers and thereby our classes will be explained below and can be seen in Figure 4.2. Each layer only access classes from the same layer, the layer beneath it or to the right. This makes it relatively easy to replace layers.

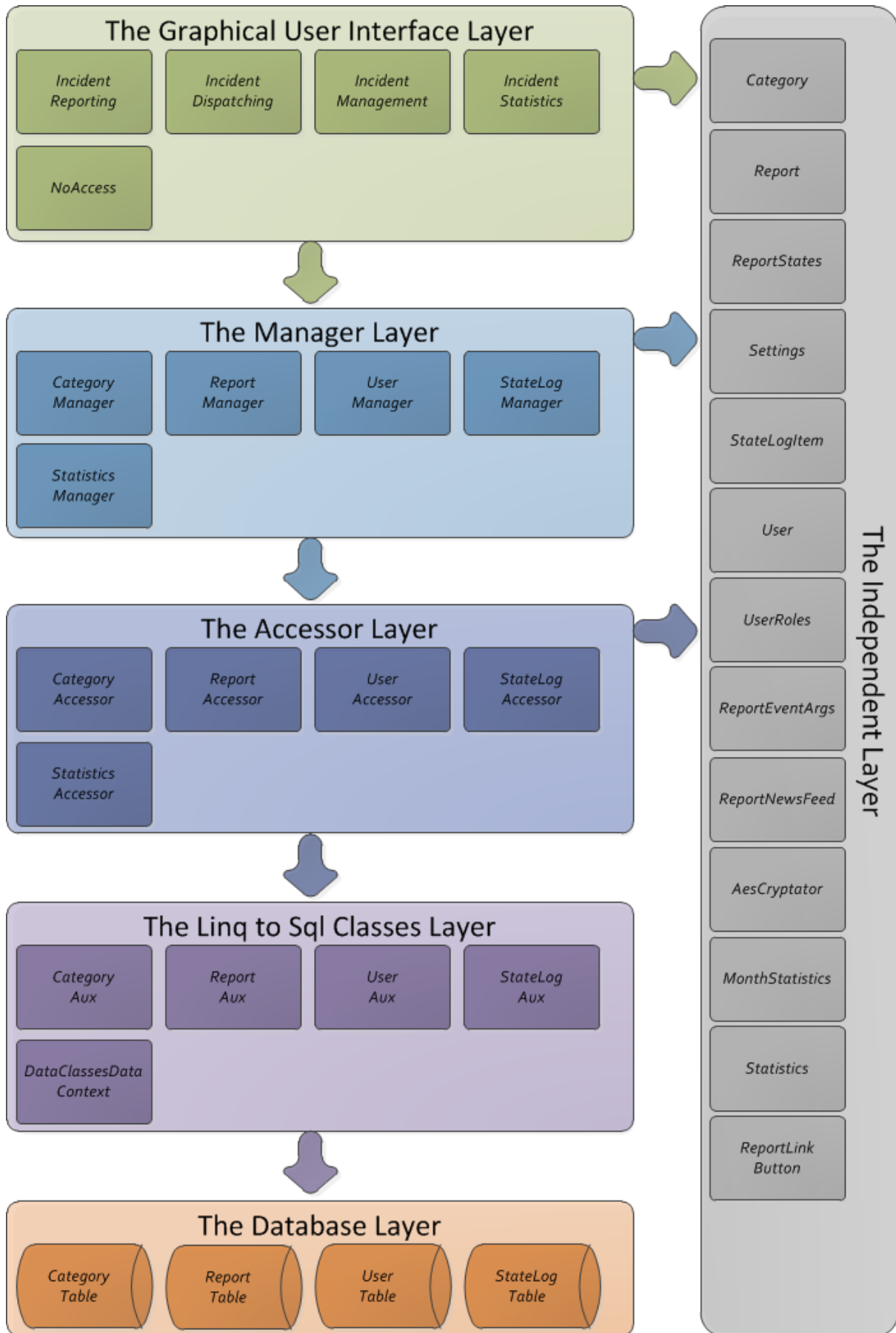


Figure 4.2 - Diagram of the layers and classes in our system and how they relate to each other

4.2.1 The Graphical User Interface

The graphical user interface layer consists of four web parts and a user control. The web parts are incident reporting, incident dispatching, incident management, incident statistics and the user control is NoAccess.

The incident reporting web part allows users to report incidents through an HTML form. The form is made up by a few input elements; a textbox where a title can be entered, a text field where a description can be entered, and a drop down list where a category can be chosen. The category drop down list is populated by an instance of CategoryManager. The web part also has a news feed that allow incidents reported by a user to be managed by said user. This news feed gets a list of reports from an instance of the ReportManager. Reports can be supplemented with more information upon request. This is done by selecting the report in question. The report is then loaded into the HTML form mentioned earlier and can be edited freely. When reporting incidents the currently logged in user is accessed via the User class. Reports, users and categories are stored in Report, User and Category containers while processing occurs. If a user without permission attempts to access this web part they will be redirected to the NoAccess user control.

The incident dispatching web part allows dispatchers to handle incoming incident reports through an HTML form. The form is made up by a few labels and input elements. There are two labels one in which the title of incident is displayed and one where the description is. There is a text field in which a message to the user who submitted the report can be written. There are two drop down lists; one which allows a manager to be selected for an incident and one in which an action can be selected. The manager drop down list is populated by an instance of the UserManager. In order to populate the action drop down list the enumerator ReportStates is used. A dispatcher is provided with a news feed which gets a list of reports from an instance of the ReportManager. This news feed allows reports to be selected. A selected report will be loaded into the form mentioned earlier. A loaded incident report can be assigned to a manager, and an action can be chosen. When a report is dispatched a new state log item is created by an instance of the StateLogManager. The state log item records the new state and which user caused it to be changed. The currently logged in user is accessed through the User class. Reports and users are stored in Report and User containers while processing occurs. If a user without permission attempts to access this web part they will be redirected to the NoAccess user control.

The IncidentManagement web part has the same functionality as the incident dispatching web part except for two key differences. Managers can only see incident reports assigned to them and are waiting to be managed. And managers cannot assign incident reports to other managers. If a user without permission attempts to access this web part they will be redirected to the NoAccess user control.

The incident statistics web part allows administrators to generate statistics. This is done through an HTML form. This form has one textbox where a search term can be written. A statistics search is limited to a one year period prior to the search date. Once a search has been done all matching reports are displayed in a news feed which is populated by an instance of the ReportManager. A box diagram displays the number of solved incidents versus the number of unfinished ones. This chart is generated by the Statistics class. The currently logged in user is accessed through the User class. Reports and users are stored in Report and User containers while processing occurs. If a user without permission attempts to access this web part they will be redirected to the NoAccess user control.

The user control NoAccess displays a text explaining that access was denied to another form.

4.2.2 The Manager Layer

The manager layer consists of five classes that can retrieve, save and/or create objects in the data storage; CategoryManager, ReportManager, UserManager, StateLogManager and StatisticsManager. The manager classes validate their input before calling accessors and will never throw exception if given invalid input.

The CategoryManager class can retrieve categories in two different ways. It can retrieve all categories. Or it can retrieve a specific category by using its id. Categories are retrieved by an instance of the CategoryAccessor class. Categories are stored in Category containers while processing occurs.

The ReportManager class can create, change, and retrieve reports. When reports are created an initial state is created using an instance of the StateLogManager class. The reports themselves are created using an instance of the ReportAccessor class. When creating reports the parameters are inspected. For example one parameter is an id of a category, it is checked that the referenced category exist. This is done using an instance of the CategoryManager. In order to change a report, an instance of the Report class is required. The information in this instance is then saved to the data storage using an instance of the ReportAccessor class. A new state for the

report is also created using an instance of the StateLogManager. There are five different ways of retrieving reports and all are using an instance of the ReportAccessor class. All reports can be retrieved for a user, dispatcher or manager. When retrieving for a user, only incidents reported by said user are retrieved. Dispatchers can only retrieve reports that are waiting to be dispatched. And managers can only retrieve reports that are waiting to be managed. A specific report can also be retrieved. It is possible to retrieve all reports that contain a search term. Reports that contain the search term are retrieved. Reports, categories and state log items are stored in Report, Category and StateLogItem containers while processing occurs.

The UserManager class can retrieve all users whom are managers using an instance of the UserAccessor class. Users are stored in User containers while processing occurs.

The StateLogManager class can create state log items using an instance of the StateLogAccessor. When creating a state log item the parameters are inspected. For example it is checked that the new state exists in the enumerator ReportStates. State log items are stored in StateLogItem containers while processing occurs.

The StatisticsManager class can generate statistics about a one year period if given a date and a search term. The search term is used to filter the reports before statistics are made. These statistics are generated on a per month basis using an instance of the StatisticsAccessor class. Month statistics are stored in MonthStatistics containers while processing occurs.

4.2.3 The Accessor Layer

The accessor layer consists of five classes that can retrieve, save and/or create objects in the data storage; CategoryAccessor, ReportAccessor, UserAccessor, StateLogAccessor and StatisticsAccessor. The accessor classes do not validate their input and are likely to throw exceptions if given invalid input.

The CategoryAccessor class can retrieve categories in two different ways. It can retrieve all categories. Or it can retrieve a specific category by using its id. Categories are retrieved using the class collection DataClasses. The DataClasses class collection provides CategoryAux class instances. They are used to create instances of the Category class. Categories are stored in CategoryAux or Category containers while processing occurs.

The ReportAccessor class can create, change, and retrieve reports. When reports are created an initial state is created using an instance of the StateLogAccessor class. The reports themselves

are created using the class collection DataClasses. In order to change a report, an instance of the Report class is required. The information in this instance is then saved to the data storage using the class collection DataClasses. There are five different ways of retrieving reports and all are using the class collection DataClasses. Reports can be retrieved for a user, dispatcher or manager. When retrieving for a user, only incidents reported by said user are retrieved. Dispatchers can only retrieve reports that are waiting to be dispatched. And managers can only retrieve reports that are waiting to be managed. A specific report can also be retrieved. It is possible to retrieve all reports that contain a search term. All reports that contain the search term are retrieved. The DataClasses class collection provides ReportAux class instances. All text in these instances is encrypted and are decrypted and instances of the Report class are created. Reports are stored in ReportAux or Report containers while processing occurs.

The UserAccessor class can retrieve all users whom are managers using the class collection DataClasses. The DataClasses class collection provides UserAux class instances. They are used to create instances of the User class. Users are stored in UserAux or User containers while processing occurs.

The StateLogAccessor class can create state log items using the class collection DataClasses. State log items are stored in StateLogAux or StateLogItem containers while processing occurs.

The StatisticsAccessor class can generate statistics about a one year period if given a date and a search term. The search term is used to filter the reports before statistics are made. These statistics are generated using the class collection DataClasses. Month statistics and reports are stored in MonthStatistics or ReportAux containers while processing occurs.

4.2.4 The Independent Layer

This layer consists of eleven classes and two enumerators. Six of the classes are containers; Category, Report, StateLogItem, User, ReportEventArgs and MonthStatistics. We called them container classes because they act as containers and mostly only hold information without any data manipulation methods. One class is used when encrypting and decrypting (AesCryptator). One holds global settings (Settings). One that builds URLs to charts (Statistics). Two that are ASP.NET user controls, ReportNews feed and ReportLinkButton. The two enumerators are ReportStates and UserRoles.

The Category class is a container. It holds the name of a category and its unique identity.

The Report class is a container class. It holds the unique identity of a report, the title, the description, the feedback, the manager of the report, an identity reference to a category and the current state of the report. The current state of the report is stored in an instance of the StateLogItem class.

The Settings class is a static class and the point where all the globally configurable settings in our system can be found.

The StateLogItem class is a container class. It holds an identity reference to a report, the current state as a ReportStates enumerator, the time the report took the new state and the user who caused the state to change.

The User class is a container and static class. It holds the identity of the user and which role the user has as a UserRoles enumerator. The static part of the class can be used to retrieve the currently logged in user.

The ReportEventArgs class inherits from EventArgs. It holds an instance of the Report class. The Report is used to hold information about the report that caused the event.

The MonthStatistics class is a container class. It holds the year in which the statistic occurred, the month of the year the statistics occurred, the number of incidents reported, and the number of solved incident reports.

The ReportNews feed class is a server control and can be placed in ASP.NET pages and user controls. It is used to create lists of reports. Reports are color coded based on what state they are in. There is a click event for when a report is clicked. This event uses the ReportEventArgs class for event arguments.

The ReportLinkButton class inherits from LinkButton is a server control and can be placed in ASP.NET pages and user controls. It is used to create lists of reports. It holds a reference to a report and provides some formatting. The formatting renders a HTML table around the link button. The table contains information about the referenced report.

The ReportStates enumerator contains all the seven states a report can be in; these are described in Section 2.3.1.

The UserRoles enumerator contains all the user roles that exist in our system; User, Manager, Dispatcher and Admin.

The Statistics is a static class and can generate links to charts if given statistics.

The AesCryptator class is a wrapper for the built in .NET AesCryptographicService class. It provides a simple way to encrypt and decrypt strings and byte arrays.

4.2.5 LINQ to SQL Classes Layer

This layer consists of five classes; DataClassesDataContext, ReportAux, CategoryAux, UserAux and StateLogAux. The DataClasses class collection is used to give an object oriented way of interacting with the database tables. It consists of several classes we called aux classes and a data context class.

The DataClassesDataContext class is used to connect and access the different tables of the database.

The ReportAux class is a wrapper for the reports table. It holds a reference to a CategoryAux for the category referenced by the report. A list of all the StateLogAux that reference the report is also available.

The CategoryAux class is a wrapper for the categories table. It holds a list of references to ReportAux to keep track of all reports that reference the category.

The UserAux class is a wrapper for the users table.

The StateLogAux class is a wrapper for the stateLogs table. It holds a reference to a Report to keep track of which report a state log item belongs to.

In order to save new instances of an aux classes, just set their properties and then use an instance of the DataClassesDataContext class to save them to the database.

4.2.6 The Database Layer

The database-layer consists of a SQL server with four tables; reports, users, stateLogs and categories. A diagram of the SQL database can be seen in Figure 3.15.

The reports table has six fields:

- Id; is a unique integer and the primary key;
- Title; is a varbinary(max);
- Description; is a varbinary(max);
- Feedback; is a varbinary(max);

- CategoryId; is a foreign key to the id field of the categories table;
- Manager; is a reference to the id field of the users table;

Of these six fields two are voluntary; feedback and manager. Title, description and feedback are strings that have been encrypted. Feedback allows dispatchers and managers to have dialogs with the submitter.

The categories table has two fields; id which is a unique integer and name which is an nvarchar. The id is the primary key. Both of these fields are mandatory.

The stateLogs table has five fields:

- Id; is a unique integer and the primary key;
- State; is an integer;
- TimeChanged; is a date time;
- UserId; is a reference to the id field of the users table;
- ReportId; is a foreign key to the id field of the reports table.

All five fields are mandatory. The state field is the state a report was in at the time specified in TimeChanged. It should exist in the ReportStates enumerator. TimeChanged should be the time the state log item was created. The user reference should be to the user who created the new state log item.

The users table has two fields; id which is a unique integer and role which is an integer. The id is the primary key. The role is a user role and should exist in the UserRoles enumerator. Both of these fields are mandatory.

4.3 Web Interface

There are four web parts in our system; Incident Reporting, Management, Dispatching and Statistics.

Anyone is allowed to view the Incident Reporting web part seen in Figure 4.3. In this web part users can report an incident by entering a short description or subject in a title field, and describe the incident in a text area. They can choose to categorize the incident within a category but it is not necessary; it defaults to a general category and can be changed by dispatchers and managers later on. The web part also has a list of incident reports showing all reports the user has submitted.

Incident Reporting

Incident Reporting

Report Incident

Titel:

Description:

Vårflod

OK

Report Feed

Flooding Damage Windows on the 24th floor were impaled by rods and broken.	WaitingForDispatcher	04/13/2011 15:22:34
Window Broken A window on the first floor has collapsed.	WaitingForDispatcher	04/13/2011 15:20:12
Development Tools I cannot access the SQL server with my domain credentials! <i>You shouldnt.</i>	Closed	04/13/2011 14:58:34
Broken Window On the 98th floor a window has been broken by a seagul. <i>Sure... :/</i>	Closed	04/13/2011 14:57:21
System Failure The system refuses to start! <i>Your own damn fault.</i>	Undoable	04/13/2011 14:51:31
Sharepoint Need more RAM to properly run SharePoint 2010 Server and Visual Studio 2010. <i>Majs, korv, vete och mjöl!</i>	Done	04/13/2011 14:49:49
Isen smälter Isen ute i älven(Klarälven) smälter! <i>Bah</i>	Done	04/13/2011 12:31:24

Figure 4.3 - Incident Reporting Form

The user must have manager privileges to view the Incident Management web part seen in Figure 4.4. A manager can write feedback back to the submitter and can request more information. The manager can perform a few actions:

- Assign the report to themselves; does nothing, can be used to write feedback to the submitter;
- Mark the report as solved;
- Request more information from the submitter;
- Mark the report as pending to be solved;
- Close reports; reports that are not describing real problems should be closed and not solved;
- Mark reports as undoable; a problem is described that cannot be solved.

The web part also has a list of incident reports showing all reports that are assigned to the manager and are waiting to be managed.

Incident Management

Incident Management

Titel:

Clouds

Description:

The clouds are blocking the sky.

Comment:

Actions:

AssignedToManager

Report Feed

Clouds The clouds are blocking the sky.	AssignedToManager	05/23/2011 20:57:41
Windows whissing! Yesterday while i was programming chickens a few of them were accedently given firmware destined for hawks. The pavement will never be the same again.	AssignedToManager	05/23/2011 20:58:01
Borders in Borderlands Are there many borders in borderlands 1.4.1	AssignedToManager	05/23/2011 20:58:11

Figure 4.4 - Incident Management Form

If the user has dispatcher privileges it can view the Dispatching web part seen in Figure 4.5. The dispatcher can perform a few actions:

- Assign the report to a manager,
- Mark a report as waiting for dispatcher; can be used to write messages (feedback) to other dispatchers or the submitter,
- Mark the report as solved,
- Request more information from the submitter,
- Close reports; reports that are not describing real problems should be closed and not solved,
- Mark reports as undoable; a problem is described that cannot be solved.

A dispatcher can write feedback back to the submitter and can request more information. The web part also has a list of incident reports showing all reports that are waiting to be dispatched.

Incident Dispatching

Incident Dispatching

Titel:
Annanas
Description:
It grows wrong!

Feedback:

Actions:
WaitingForDispatcher

Managers:
xxcarlsj

OK

Report Feed

Annanas It grows wrong!	WaitingForDispatcher	04/13/2011 15:00:51
Entropy Its approaching!	WaitingForDispatcher	04/13/2011 15:12:48
Window Broken A window on the first floor has collapsed.	WaitingForDispatcher	04/13/2011 15:20:12
Flooding Damage Windows on the 24th floor were impaled by rods and broken.	WaitingForDispatcher	04/13/2011 15:22:34

Figure 4.5 - Incident Dispatching Form

Users with administrator privileges can view the Statistics web part seen in Figure 4.6. Administrators can generate statistics based on a search term that can be entered.

Incident Statistics

Incident Statistics

Search:

Report Feed

<u>Isen smälter</u> Isen ute i älven(Klarälven) smälter! Bah	Done	04/13/2011 12:31:24
<u>Vatten</u> Det är vatten i floden!?! , är du galen?, vadåda? ?, whatever..., I dont even!	Done	04/13/2011 14:35:24
<u>System Failure</u> The system refuses to start! Your own damn fault.	Undoable	04/13/2011 14:51:31
<u>Broken Window</u> On the 98th floor a window has been broken by a seagull. Sure... :/	Closed	04/13/2011 14:57:21
<u>Development Tools</u> I cannot access the SQL server with my domain credentials! You shouldnt.	Closed	04/13/2011 14:58:34
<u>Clouds</u> The clouds are blocking the sky.	WaitingForDispatcher	04/13/2011 15:00:33
<u>Annanas</u> It grows wrong!	WaitingForDispatcher	04/13/2011 15:00:51

t reports by month
ar from 4/26/2010
ontaining ""



Figure 4.6 - Incident Statistics Form

4.4 Security

The security consists of two major parts, user access restriction and protection of sensitive information. User access restriction will be described in Section 4.4.1, followed by a description of how we protected incident report information in Section 4.4.2.

4.4.1 Access hierarchy

There are four different user types in our system; administrator, dispatcher, manager and user. Administrators can generate statistics from reports based on search results. Dispatchers can

dispatch new reports to make sure that they have relevant information and that the report gets to a manager who can solve it. Managers solve incident reports assigned to them. Users can report incidents. The different user types are hierarchical in nature as can be seen in Figure 4.7. Administrators expand from the access rights that dispatchers have. Dispatchers do the same from managers and managers from users.

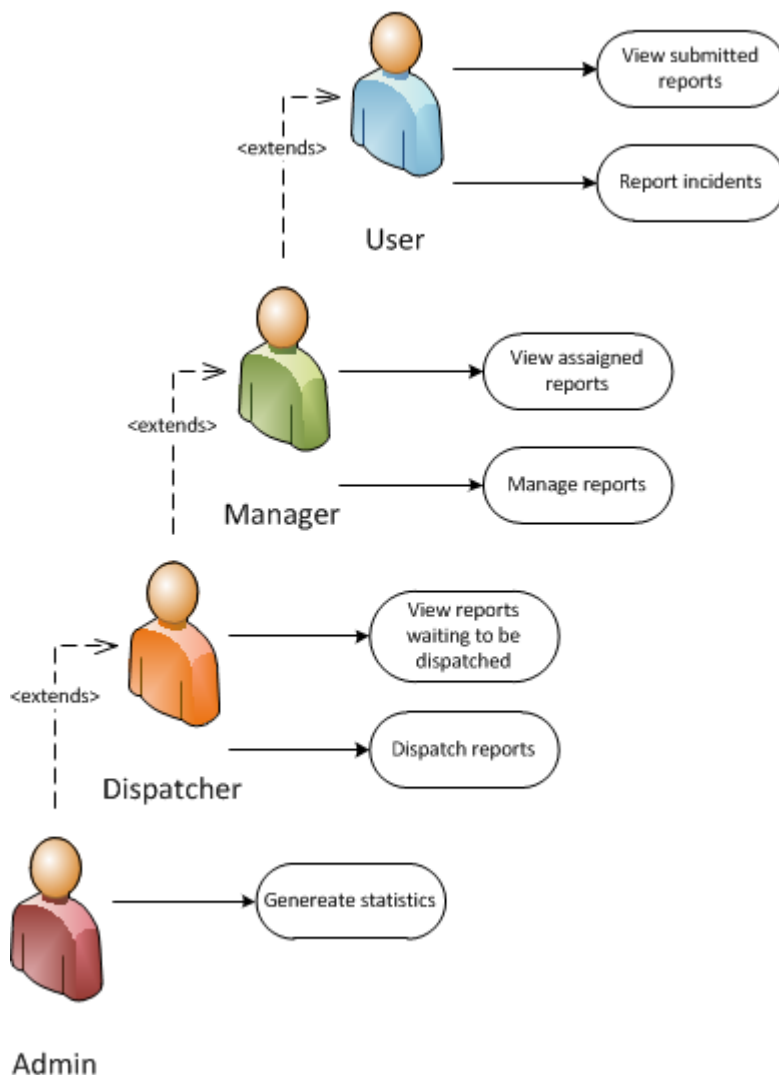


Figure 4.7 - Access hierarchy

4.4.2 Report encryption

Sensitive information in Incident reports is the titles, the descriptions and the feedbacks. This information needs to be protected. We protected it by encrypting it with AES. The added protection comes at a cost of course. In order to access the information it needs to be decrypted. This is especially true if the information needs to be searched, in which case the information needs to be decrypted before search can be done. This however does not pose a

serious problem in our system because the time penalty is small and only affects a small part of the system.

To protect the encryption process we partly separated the key and initialization vector from the code and database. It is hidden partly in the code, database and the file system. In other words one needs access to the database, the code and two special files in order to encrypt and decrypt reports. Unfortunately we cannot go in to too much detail about the concrete method of which we accomplished this. But it is very easy to secure the database even against us by just editing a few variables in the code before deploying the project.

5 Conclusions

We will review the specification and describe an improved version of it with all the learning experiences the project has given in Section 5.1. An overview of the project can be found Section 5.2. A discussion of possible future development opportunities can be seen in Section 5.3. The chapter will end with a general overview in Section 5.4

5.1 Specification

There are some aspects that we realized during and after the development process. Some things that could have been handled better from the start or simply removing redundant functionality. We would include groups in the specification, for more details about groups see Section 3.6.1. We would merge the two states closed and undoable into a single state.

5.2 This project

We are really happy with what this project has given us. Learning how to develop SharePoint web parts was overall a fun experience. Getting familiar with the cryptographic options in C# .NET framework 3.5 was interesting. Working with Scrum on a real project was a good experience.

We managed to satisfy everything in the final specification described in Section 2.3. There was a late addition of groups (detailed in Section 3.6.1) that we were unable to implement. Our supervisors at Tieto did seem reasonably pleased considering the time we had to implement the system. But they still need to evaluate the code. The lack of groups is a major drawback.

5.2.1 Security

Encrypting the sensitive information in reports with AES worked well. It was fast enough at decrypting to be able to decrypt up to a thousand reports in a reasonable amount of time; see Figure 3.14. Some performance problems arise when the number of reports searched at once is greater than one thousand or if many administrators try to search at the same moment. See Figure 3.14 for the time requirements needed to search different amount of reports.

The access hierarchy isolates the different type of users and makes sure users only have access to the web parts they need in order to fulfill their roles.

5.2.2 Scrum

Scrum worked out quite well. We were able to have a working system after each sprint. It was useful to have the sprint demos where our supervisors at Tieto could point out if we were going in the wrong direction.

There was however a problem that occurred in sprint four where a holiday and scheduling conflicts caused the third sprint demo to be delayed over a week. This cut the fourth sprint short and ended up being just two days long. We chose to just investigate and not implement due to the lack of time available. Though the time was not wasted, it was used to write this report.

This highlights a problem in Scrum, the need to have the customer (in our case our supervisors at Tieto) readily available for meetings throughout the development process.

5.3 Future work

Our prototype system was not integrated into Tieto's intranet. In order to do this the web parts would need to be deployed to the SharePoint server. The cryptographic algorithm would need to be configured. Meaning that a key and initialization vector has to be chosen and stored in files. An SQL database would need to be created and configured; it can be done by using the LINQ to SQL Data Classes. SharePoint users need to be integrated into the system. Currently a hardcoded user is returned when the User class is used to access the logged on SharePoint user. Groups need to be implemented for the system to fit the specification properly.

There are some features we would like see added to the system in the future to make it even more convenient.

Reloading the entire page just to keep the news feeds updated is wasteful and could disturb other web parts. It would therefore be positive to reload the web parts independently from the page. This could be achieved by using the ASP.NET update panel user control. The update panel causes everything in it to reload independently from the page the control is on. It is done by using AJAX (Asynchronous JavaScript and XML).

The web parts could be made highly customizable. This would allow users to personalize their web parts. An example would be to make it a choice if the forms and news feeds in the web parts should be displayed or hidden by default. At the moment the news feeds are displayed while the forms are hidden and a link has to be pressed to show them.

The search process can be streamlined quite a bit. How this can be done is explained in Section 3.7.1.

Currently certain things can only be managed with direct access to the SQL database. Categories cannot be managed and users cannot be given roles without direct SQL access. It would be a lot more convenient if web parts existed that would allow such things to be edited via the intranet.

5.4 General conclusions

Overall, the project went well. We set up a development environment for developing SharePoint web parts. The main goal was to create an Incident Management System which uses a capable security model. Reported incidents can contain sensitive information that should not be accessible by even database administrators. Existing systems expose sensitive information to database administrators and do not sufficiently protect reports from external attacks. Therefore the data is encrypted in the database, and there are checks that make sure that only users with proper access rights can read reports. We are very pleased with how we solved the security issue. Reports are well protected in a three pronged approach, in order to encrypt and decrypt one needs access to the database, the code and two secret files. The access rights need to vary depending on which state a report is in. Another goal was to make it as easy as possible for the users to report incidents and encourage them to do so. Both of these goals were completed.

Another feature we are pleased with is the box chart in the statistics web part. It gives a good visual representation of trends and feedback for post decision evaluation of policies.

Our supervisors at KaU and Tieto have been really helpful. Our supervisor at KaU helped us a lot by reading this report regularly and suggesting changes. Our two supervisors at Tieto helped us create the specification and had a lot of valuable and intelligent input when we implemented it.

6 Bibliography

- [1] Wikipedia. (2011, Feb.) Wikipedia. [Online]. [http://en.wikipedia.org/w/index.php?title=Microsoft SharePoint&oldid=416182459](http://en.wikipedia.org/w/index.php?title=Microsoft_SharePoint&oldid=416182459)
- [2] Tyler Holmes. (2008, Mar.) System.What? [Online]. <http://blog.tylerholmes.com/2008/03/walkthrough-creating-sharepoint-feature.html>
- [3] Microsoft. MSDN. [Online]. <http://msdn.microsoft.com/en-us/library/ms499606%28v=office.12%29.aspx>
- [4] Microsoft. MSDN. [Online]. <http://msdn.microsoft.com/en-us/library/dd584174%28office.11%29.aspx>
- [5] Atlassian. Atlassian. [Online]. <http://www.atlassian.com/software/jira/>
- [6] Bugzilla. Bugzilla. [Online]. <http://www.bugzilla.org/>
- [7] My Safe Workplace. [Online]. <http://www.mysafeworkplace.com/>
- [8] Best Pratical. [Online]. <http://bestpractical.com/rt/>
- [9] Henrik Kniberg, *Scrum and XP from the Trenches.*: C4Media, 2007.
- [10] Google. (2010, Feb.) Google Code. [Online]. <http://code.google.com/intl/sv-SE/apis/chart/>
- [11] piscetes1012000 and Senthilrajan Kaliyaperumal. (2010, May) MSDN. [Online]. <http://msdn.microsoft.com/en-us/library/ee554869.aspx>
- [12] Don Box and Anders Hejlsberg. (2007, Feb.) MSDN. [Online]. <http://msdn.microsoft.com/library/bb308959.aspx>
- [13] Wikipedia. (2011, Mar.) Wikipedia. [Online]. [http://en.wikipedia.org/w/index.php?title=Triple DES&oldid=418332025](http://en.wikipedia.org/w/index.php?title=Triple_DES&oldid=418332025)
- [14] Wikipedia. (2011, Apr.) Wikipedia. [Online]. [http://en.wikipedia.org/w/index.php?title=Advanced Encryption Standard&oldid=422489156](http://en.wikipedia.org/w/index.php?title=Advanced_Encryption_Standard&oldid=422489156)
- [15] Microsoft. MSDN. [Online]. <http://msdn.microsoft.com/en-us/library/ms188362.aspx>
- [16] Random.org. RANDOM.ORG. [Online]. RANDOM.ORG - True Random Number Service
- [17] Microsoft. MSDN. [Online]. <http://msdn.microsoft.com/en-us/library/bs302eat%28v=vs.80%29.aspx>
- [18] Google. (2010, Feb.) Google Code. [Online]. http://code.google.com/intl/sv-SE/apis/chart/docs/gallery/bar_charts.html

- [19] Wikipedia. (2011, Apr.) Wikipedia. [Online].
<http://en.wikipedia.org/w/index.php?title=RSA&oldid=422735691>
- [20] Wikipedia. (2011, Apr.) Wikipedia. [Online].
[http://en.wikipedia.org/w/index.php?title=Data Encryption Standard&oldid=421969035](http://en.wikipedia.org/w/index.php?title=Data_Encryption_Standard&oldid=421969035)
- [21] Wikipedia. (2010, Nov.) Wkipedia. [Online].
[http://en.wikipedia.org/w/index.php?title=Microsoft Search Server&oldid=399335297](http://en.wikipedia.org/w/index.php?title=Microsoft_Search_Server&oldid=399335297)

7 Appendix A – Google chart API, bar chart

Google chart API [18] generates charts if given a few parameters in the URL, for example <https://chart.googleapis.com/chart?>

- `chs=600x300&`; chart size in the `<width>x<height>` form,
- `cht=bvo&`; chart type,
- `chd=t:40,38,37,35,37,41,43,41,42,39,37,21|41,39,37,36,39,43,44,45,43,42,40,41&`; chart data series separated by `|`,
- `chco=FFCC33,FFE9A4&`; chart color for the different data series,
- `chxt=x,y&`; sets how the axis are aligned in the `<horizontal>`,`<vertical>` form,
- `chxr=1,0,70&`; chart axis range in the `<axis index>`,`<min value>`,`<max value>` form,
- `chds=0,70,0,70&`; chart data scaling,
- `chxl=0:|Oct|Nov|Dec|Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|&`; chart axis labels
- `chtt=Incident+reports+by+month+last+year|containing+%22term%22;` chart title | is used as line break.

If the link above is concatenated with everything before the semi colon in the bullet list the result will be as seen in Figure 7.1.



Figure 7.1 - Example bar chart

8 Appendix B - Drafts

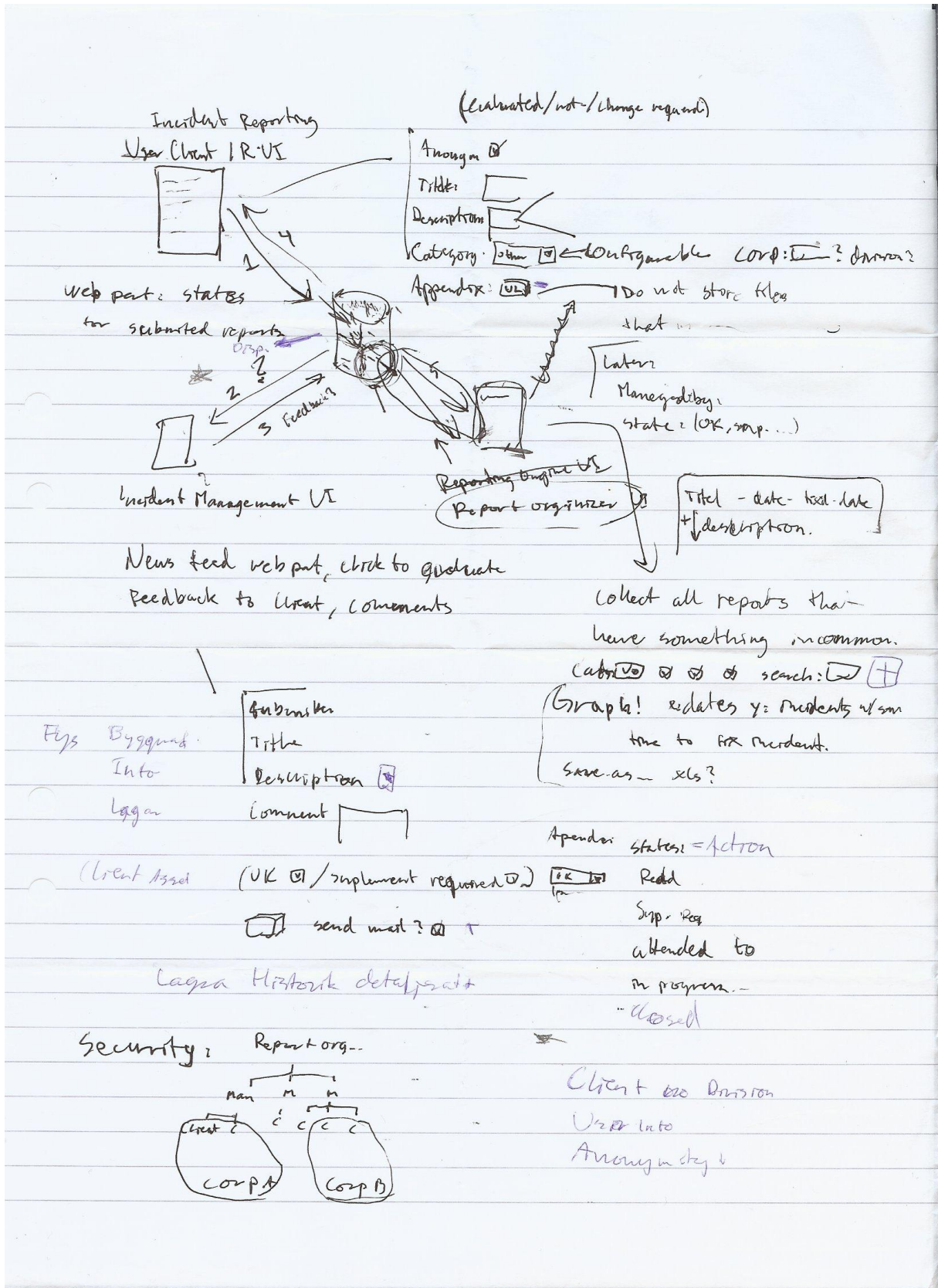
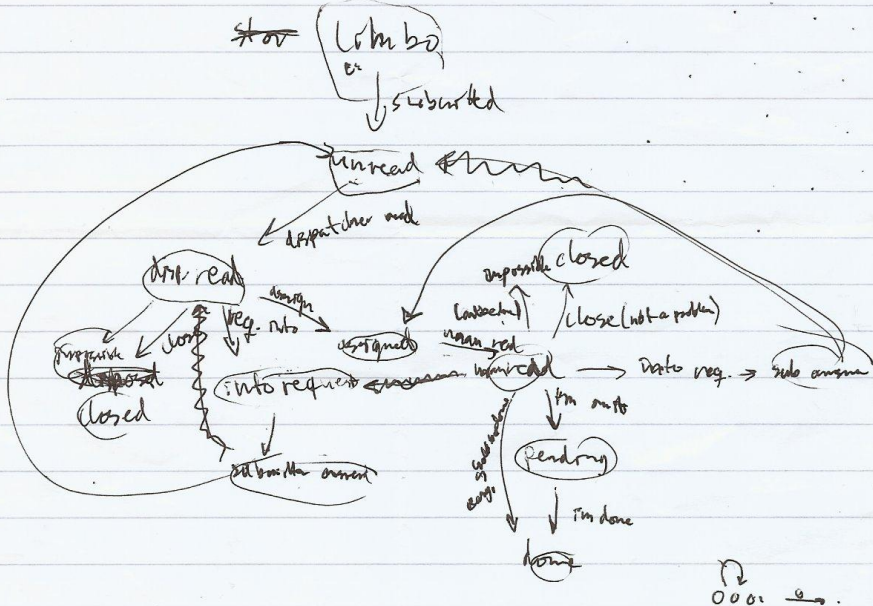


Figure 8.1 - Early system draft

State diagram for en rapport



sequence diagram

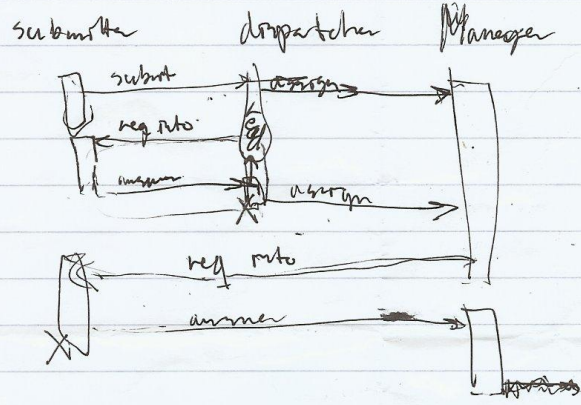


Figure 8.2 - Early state and sequence diagram draft