Faculty Code: EAR
Project Number: 0112

Hyreminder: A Hand Hygiene Tracking System with Mobile Sensors

A Major Qualifying Project Report:
Submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE

WPI

In partial fulfillment of the requirements for the
Degree of Bachelor of Science

Submitted by:

Leah Greer
Christian Mortensen
Corey Phillips
Hanxiong Shi
Jeffery Stokes

Date: March 12, 2013

Approved by:

Professor Elke A. Rundensteiner

Abstract

The purpose of this project is to extend the functionality and improve the performance of the hand hygiene monitoring system (Hyreminder). This system is in place at the University of Massachusetts Memorial Hospital in the intensive care unit, where it helps prevent the spread of infection in the hospital. Our reengineering of the system under the Java Spring framework improves the usability, performance, and maintainability of the system. This allows for faster webpage loading times and improves code readability. It also enables a more scalable system now ready to deploy at hospitals of varying size. Additionally, the system adds support for tracking moving beds via hardware sensors. Thoroughly documenting and testing the code ensures that others can develop the system further.

Acknowledgements

Our team would like to thank the following individuals for their help and support throughout our project.

- Our advisor from Worcester Polytechnic Institute, Professor Elke Rundensteiner, for her guidance and support throughout our project.
- Doctor Richard Ellison, director of Infection Control at the University of Massachusetts Memorial campus. Thanks for his creation of the project and for his ideas and suggestions on feature implementation.
- Lei Cao, graduate student at Worcester Polytechnic Institute and member of the Database System Research Group, for his valuable opinions through development and his assistance with answering our questions.
- Di Wang, graduate student at Worcester Polytechnic Institute, for her work creating the original system.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

Maintaining good hygiene in hospital settings has been repeatedly shown to be critical in preventing the spread of infection. In 2006 a study involving 168 hospitals discovered that hospital-acquired infections (HAI) extend an average hospital stay from five to twenty-one days, and multiply the cost of the stay by a factor of six.[1] Infections acquired while in a hospital can result in extended stays and exacerbated illnesses, as well as cost the hospital and the patients more time and money. A system to monitor and improve hygiene is therefore beneficial to both patients and hospital staff alike. To that end, the Hyreminder system was developed by the Worcester Polytechnic Institute Database System Research Group in collaboration with UMass Memorial Hospital. The system is designed to electronically monitor hygiene and report compliance statistics via web interface that is accessed by head nurses, doctors, and supervisors (see Section 2.1).

The goal of this project was to improve the Hyreminder system. We were to improve the maintainability and performance of the Hyreminder web application, and to add new functionality. The team approached maintainability by improving code readability and reusability, in code, developer and user documentation, and automated testing. We increased performance by looking into database query structure, system architecture, and tools used for data visualization. We also upgraded the system's security and made several improvements to the interface's usability.

The first step to achieving our goal was to examine the current Hyreminder web application. The original system was inefficient, with several large blocks of commented out code, duplicated code, and insufficiently documented functions (see Section 3.1). Additionally, the code was built using the Google Web Toolkit[2] framework, which is useful for smaller applications, but can cause issues for larger, more complex applications (see Section 3.2.1). After this analysis we decided that it would be in the best interest of the project to reengineer using the Spring Web MVC Framework[3]. The Spring framework is designed to allow for simplicity and rapid deployment of enterprise level solutions. Spring Core, the foundation package of the framework, has a heavy emphasis on managing the "plumbing" code of an application allowing the developers to focus on the important business logic of the application. This allows for clean, focused code that is easily maintained. Spring does this while allowing the developers to make use of dependency free plain-old-java-objects (POJOs) that are not heavily coupled with interfaces or APIs common to other frameworks. Thus, it is trivial to add or change packages or frameworks at some point in the future if needed. These all come together to deliver clean, maintainable, scalable code as quickly as possible.

While reengineering we took it upon ourselves to work on the maintainability and performance of the web application. As we wrote code, we added in-line documentation. We revised code we

[1] Lee, Christopher. (2006, November). Studies: Hospitals Could Do More to Avoid Infections. The Washington Post.

[2] Google Developers. (2012). Google Web Toolkit. https://developers.google.com/web-toolkit/

[3] Spring Source Community. (2013). http://www.springsource.org/

wrote to be more reusable, for example, much of the code involved in reporting is similar. By pulling the similarities out into their own functions, we decreased the amount of code that henceforth would need to be maintained. We evaluated the queries in use, consolidated them where possible, and added stored procedures in order to increase their performance and reduce load times.

When we changed from using Google Web Toolkit to Spring, we needed to completely rebuild the front end of the web application. We kept the look of the system the same, but were able increase the usability by rearranging user interface elements on each page to better suit the workflow. We also added more browser support from only Internet Explorer 8 to Chrome, Firefox, Internet Explorer 8 and 9, and Safari. We added enhancements that make it easier for the user to use the web application, such as easier switching of ICUs and URLs for each of the tabs. With the new front end, we had to find a new way to display charts. After exploring a number of options (see Section 6.2 for more detail), we chose to go with JQuery Visualize.[4] This library had the basic features we needed, however we put extensive amounts of work into adding more features to the library to more fully accommodate our requirements. For example, we added support for charts with two y-axes and to allow a small pop-up window on mouse hover for more information about a particular data point. The two y-axes were required for the compliance charts which have sensors hits on the left y-axis and percent compliance on the right y-axis. Mouse hovers were a feature in the original Hyreminder system that we wanted to preserve.

During development we tested the system in several ways. We performed correctness testing to verify that the system's output matched what the original system was outputting (See Section 7). We also ran performance testing to determine how much we gained from our redesign in terms of time taken to execute various tasks (results in Section 7.2). Lastly, we implemented and performed stress testing to test the limits of the system, for instance, how many sensors could be hooked up to one ICU before the system began to slow down (results in Section 7.1).

After the new framework was swapped in, and the web application was back to running to the original level of functionality, we added new features. We now provide complete reporting in the ID Based and the Multi-ICU types. This involved writing the queries and processing for both, and creating a user interface for Multi-ICU (see Section 4.1). The second feature we added was commenting on sensors in the Anomaly system. Users can now add and remove comments on a sensor.

The major feature we added was the tracking of mobile beds. This feature was requested to be able to track the beds as they move through the ICUs, enabling better maintenance of the system, to aid in replacing malfunctioning sensors. The sensors are an expensive resource to lose. Because the beds are mobile, and the only fixed points that the system can use are the soap dispenser sensors, we had to develop logic to connect sensor hits with locations. By measuring the time between dispenser and bed sensor hits, we derive an estimate of a bed's probable locations. (see

[4] Jehl, Scott. (2013). JQuery Visualize. https://github.com/filamentgroup/jQuery-Visualize

Section 4.3.2).

Tracking beds involved adding more screens to the Anomaly system's user interface (see Section 5.4 for full details). The first screen is used for viewing a map of the current sensor locations. The second screen is for map maintenance, where the user will set up locations on the map and which locations are close to each other. The last screen is used for setting up which sensor is at a location for the stationary sensors, while the mobile sensor locations are determined by our system. Maps on all of these screens are implemented using Raphal,[5] a JavaScript drawing library (see Section 6.3).

While implementing the moving bed functionality, we solved another maintainability issue. We improved the system of replacing sensors with new ones. Previously this was done by replacing the old ID with the new ID. Now we have an abstraction from the sensor and the physical device. We created a logical sensor with a unique ID for use in the sensor hit tables and each logical sensor will have a list of physical sensors IDs, only one active at a time. Section 5.4 explains in more detail.

Our team came up with a good workflow early on in the project. We met together three times a week: one meeting we held shortly after our weekly meetings with Professor Rundensteiner to discuss work to be completed the following week, and two other meetings to catch up with each other and discuss any issues that had arisen. As a team, we divided up the project requirements into smaller tasks, and each person chose one or more tasks to complete each week. Anytime a team member encountered sufficient difficulty so as to slow progress, other group members were willing and able to step in and assist so that the project could proceed as close to schedule as possible. Some members were focused on particular areas of the project, but in general we all covered multiple parts of the project.

In conclusion we have added onto and improved the Hyreminder and Anomaly web applications. With the framework change, we created a more efficient and maintainable software. We have added documentation and testing, both essential components for a maintainable and expandable project. Performance gains were seen in the largest two features: report generation and chart creation. We added the the interface and backend for setting up ICUs and their sensor networks, and created a basic algorithm for tracking moving sensors such as bed.

In the future, teams will have our documentation to aid them through learning the system. The system itself is modular due to the use of the Model View Controller pattern, thus pieces can be replaced without affecting the system as a whole. New features can be added easily, again without affecting other pieces. While there are still some user interface issues that need to be resolved (see Section 9), the Hyreminder system is in a solid state to which others can add on to in the future.

[5] Baranovskiy, Dmitry. (2012). Raphaël. http://raphaeljs.com/

2 Background

Since the goal of this project was to improve and extend the Hyreminder system, it is useful to describe the reasons for Hyreminder's existence, and to describe how the Hyreminder system works.

2.1 Importance of Hand Hygiene

With nurses and doctors treating large numbers of patients daily, controlling the spread of infection is a critical component to effective medical care. Infections can be spread several ways: through physical contact, by surgical tools such as scalpels or thermometers not being cleaned properly, or via airborne transmission. Although a person's immune system can protect against many infections, patients tend to have weaker immune systems due to the nature of being sick or recovering from surgery, and are therefore at a higher risk of getting infected.[6] Infections acquired while in a hospital can extend a patient's stay by several days, and the increased stay results in additional costs and less space available for incoming patients. A study conducted in 2006 involving 168 hospitals and 1.6 million patients discovered that hospital-acquired infections extended the average stay from five to twenty-one days, and correspondingly the cost was about six times more than those that did not acquire an infection during their stay.[7]

The basic principle to reduce infection risk is through proper hygiene. This can include using gloves and masks, covering up coughs, keeping immunizations up to date, and proper hand washing.[8] A study on a program to increase hand washing in the United Kingdom that started in 2004 and ran for six years, found that soap use nearly tripled and rates of notable infections such as MRSA dropped by approximately 50%.[9] The efforts of this drive are estimated to have saved 10,000 lives, in addition to the cost savings from increased hospital stays.

Several hospitals have tried different strategies in attempts to increase hand washing compliance. The strategies employed in these initiatives vary greatly. A study at Duke University Hospital in 2009 found that by using observers to record hygiene compliance and report that data to a central web service, compliance ratios soared to over 90%, far exceeding the national average of approximately 40%. This study was so successful that the program was expanded to cover more areas of the hospital.[10] In a similar but more electronic way, an Alabama hospital piloted a more

[6] Prüss, A., Giroult, E. & Rushbrook, P. (1999). Hospital Hygiene and Infection Control. In *Safe management of wastes from healthcare activities*. World Health Organization. http://www.who.int/water_sanitation_health/medicalwaste/148to158.pdf

[7] Lee, Christopher. (2006, November). Studies: Hospitals Could Do More to Avoid Infections. The Washington Post.

[8] MedlinePlus. (2013, February). Infection Control. http://www.nlm.nih.gov/medlineplus/infectioncontrol.html

[9] Hospital Hygiene Drive 'Saved 10,000 Lives'. (2012, May). The Guardian. http://www.guardian.co.uk/society/2012/may/04/hospital-hygiene-drive

[10] Duke Medicine News and Communications. (2010, March). Technology-Based Hand Hygiene Monitoring Improves Compliance at Duke Hospital. Duke Medicine. http://www.dukehealth.org/health_library/news/technology_based_hand_hygiene_monitoring_improves_compliance_at_duke_hospital

automated system in 2010. Giving each person a badge that triggered when a soap dispenser was used, the system tracked hygiene activity over seven months and found that by the end of the study, there was a 22% drop in infections and a correlation between increased soap usage and decreased infection rates.[11]

The Hyreminder system is one implementation of a hygiene monitoring program that uses sensors to gather hygiene related information electronically. The Hyreminder system is a multi-part system, involving a screensaver for reporting the short term degree of success to nurses and doctors in the unit; a web interface for reporting the short and long term degree of success to supervisors, government and the boardroom; and several supplementary programs for getting and processing data from the sensors.

2.2 About Hyreminder

Hyreminder is a real-time monitoring system created by the Worcester Polytechnic Institute Database System Research Group, in collaboration with several partners, notably UMASS Memorial Hospital and a sensor hardware device company. Hyreminder tracks "hand washes performance with different time and space granularity, including group based, ID based, Multi-ICU and Anomaly functionality."[12] The system is made of five parts: the hardware sensor network, the complex event processing (CEP) engine, the database, the application server and multiple clients, refer to Figure 1.

[11] Brazzell, Brena Edwards, et. al. (2011, June). Efficacy of an Electronic Hand Hygiene Surveillance and Feedback Monitoring Device Against Healthcare Associated Infections. *American Journal of Infection Control*. 39, 5, E172-E173.

[12] Worcester Polytechnic Institute Database System Research Group. (2011). Hyreminder IDE Setup & User Manual.

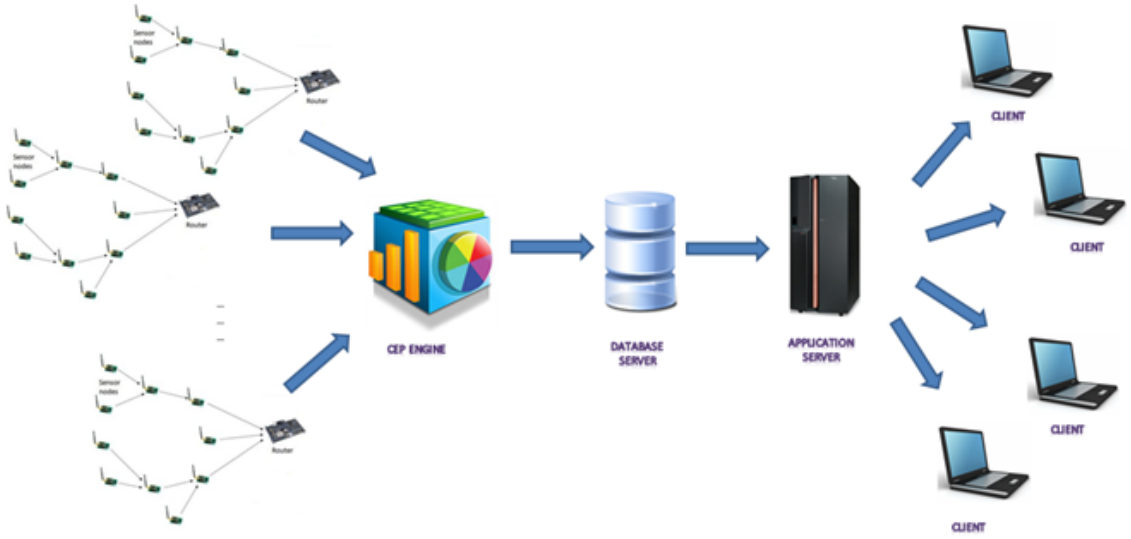


Figure 1: The Hyreminder system shown in five parts: the hardware sensor network, the complex event processing (CEP) engine, the database, the application server and multiple clients.[13]

The hardware sensor network is created by a company, name withheld. They currently have sensors installed in selected intensive care units (ICUs) of UMASS Memorial Hospital. There are sensors on the doorways of rooms tracking enters and exits, and sensors on hand wash stations. The WPI Database System Research Group's CEP engine collects data from their network, analyzes the data to detect violations and compliances and sends them to database at the end."[14] Head nurses, doctors and other users send requests using the web interface, the application server retrieves data from the database server and returns it back to the requesting user.

2.3 Hyreminder System and Usage

The Hyreminder system was originally designed to track two basic types of events – hand washes and enters/exits from rooms – at a group-based level. Some ICUs added functionality for using ID badges that corresponded to the events, so that patient contacts could be tracked. For this, two different functionalities are required, so Professor Rundensteiner's team developed the Group Based and ID Based functionalities. The Group Based solution uses physical sensors that monitor the hand washes and enters/exits from rooms to compute the hygiene compliance. The ID Based solution also uses the hand washes to compute the hygiene compliance, however the patient contacts are also used. Each employee would have a sensor attached to their ID badge, so that each individual's hygiene compliance could be monitored. A patient contact is triggered when a hospital staff member

[13] Ibid.
[14] Ibid.

approaches the patient's bed.

When we got the system, the Group Based functionality was more fully featured, while the ID Based lacked some features. The two functionalities exist in the same web application, called Hand Hygiene Performance, and the user is seamlessly switched between the two functionalities depending on the ICU being viewed. Professor Rundensteiner's team also developed the Multi-ICU functionality for comparing the compliance between ICUs, which has also been incorporated into the same web application. After seeing some faulty data from sensors that weren't working correctly, the team developed a separate Anomaly application, which consists of two parts: the Anomaly Detector and the web application. The Anomaly detector is used for offline correction of the data, to avoid reporting incorrect data created by faulty sensors. The Anomaly web application displays which sensors had errors and is used for maintenance purposes.

Upon loading any of the applications, the user is prompted to login with username and password at the Login Screen (see Figure 2). The password string is then encoded into a sequence of bytes and run through SHA-1 encryption before it is compared to the value in the database. Once successfully verified, a unique session ID is generated for the user. This ID along with the user's access level is added to the current HttpSession. By doing this, the system can easily verify the authenticity of the session when the user accesses other parts of the website. When the user logs out, the session is invalidated.

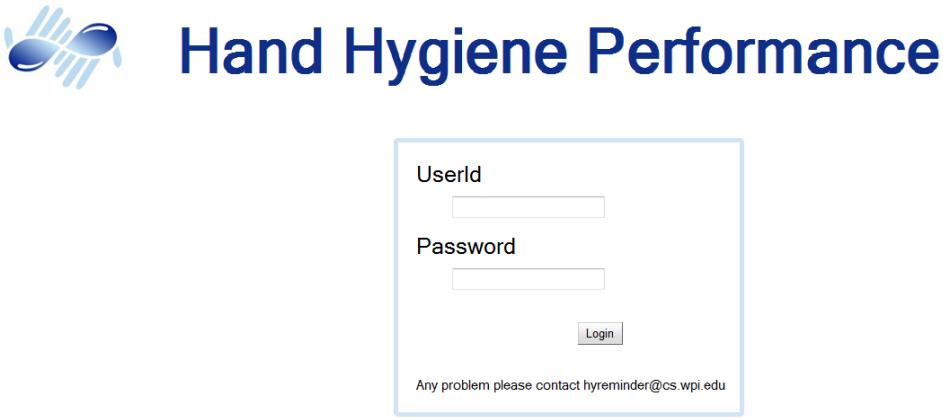


Figure 2: Login - a screenshot from the original Hyreminder application. The user must authenticate before accessing the application.

2.3.1 Group Based Functionality

The Group Based interface is composed of tabs. The first tab is Overall-Compliance View (see Figure 3). The purpose of this tab is to give users a brief, easy to understand overview of how the selected ICU is performing. There are counts of hand washes and enters/exits from a room, and a

percent compliance. These numbers reflect the data from the past hour and is refreshed from the database every twenty seconds.

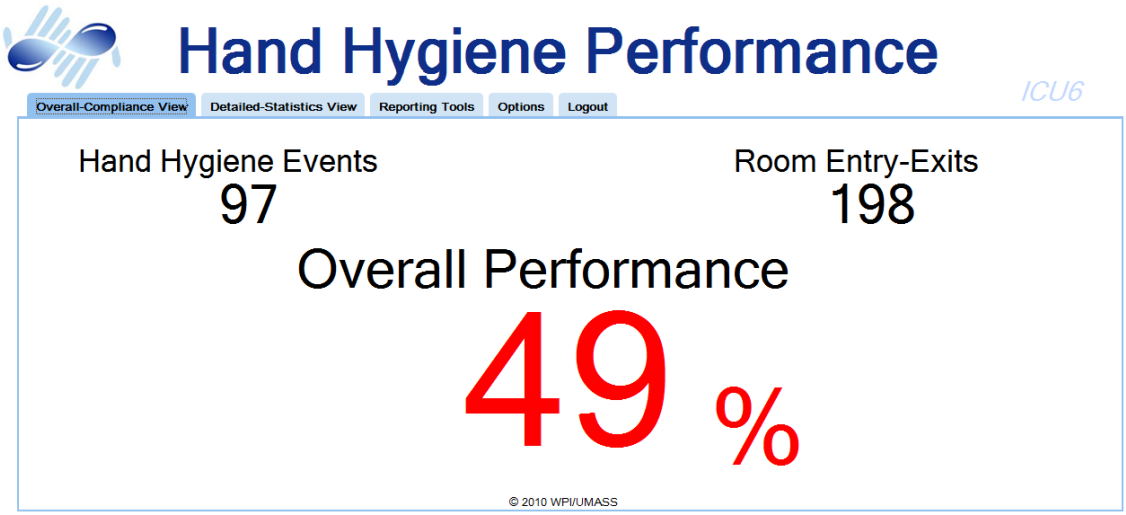


Figure 3: Group Based Overall-Compliance View - a screenshot from the original Hand Hygiene Performance application. Used for short term statistics on the ICU's performance.

The second tab is Detailed-Statistics View (see Figure 4). The purpose of this tab is to give users a more detailed understanding of past compliance. On the screen are four charts that display compliance rates over different time periods: the last twenty-four hours, seven days, eight weeks, and twelve months. The charts show the counts of hand wash and enter/exit events and are overlaid by a compliance ratio. There are two y-axes: the one on the left shows the number of sensor hits, and the one on the right shows the percent compliance. The charts are created at the time of page request, and were created as .swf files (shockwave-flash) on the server and then sent to the client, taking between one and five minutes, on average, to fully load (see Section 7.2.1 for more details).

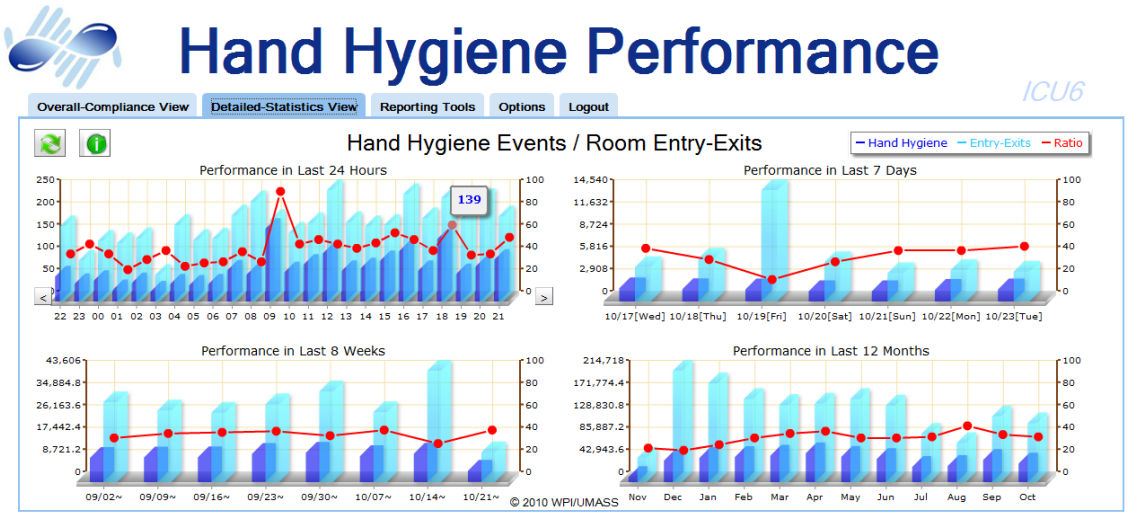


Figure 4: Group Based Detailed-Statistics View - a screenshot from the original Hand Hygiene Performance application. Used for longer term statistics on the ICU's performance.

The third tab is Reporting Tools (see Figure 5). The purpose of this tab is to allow users to download statistics for showing at staff meetings. The user can retrieve more detailed statistics than they can on the previous tab. At the top of the screen are four preset buttons, for downloading charts with the same information from the previous tab (compliance rates in the past twenty-four hours, seven days, eight weeks, and twelve months). At the bottom of the screen are drop-downs for selecting a time unit (e.g. month, day, shift) and time range (e.g. last 6 months, custom time range), and an option for aggregation.

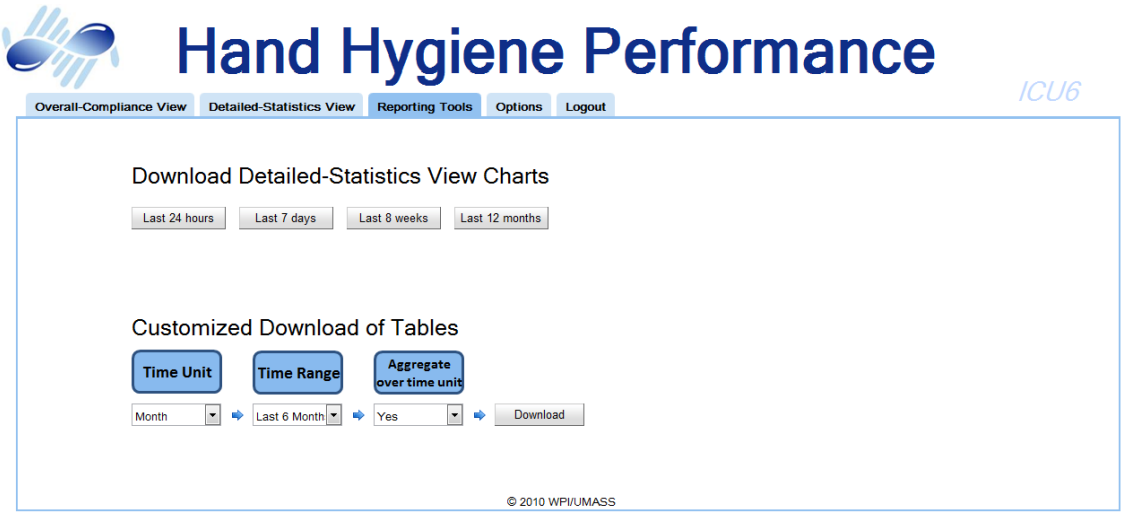


Figure 5: Group Based Reporting Tools - a screenshot from the original Hand Hygiene Performance application. Used for downloading long term statistics on the ICU's performance.

The data is downloaded as an Excel file. The charts are rendered using Excel's charting capabilities. See Figure 6 for an example download.

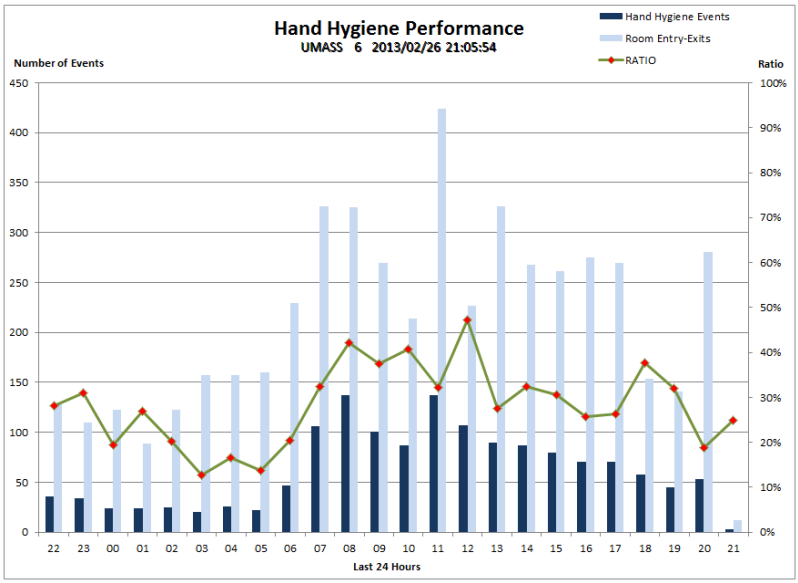


Figure 6: Group Based Reporting Tools - a screenshot from the original Hand Hygiene Performance application.

The fourth tab is Options (see Figure 7). The purpose of this tab is to allow users to change their password, using a standard change password form. This tab also allows users to switch which ICU's statistics they were viewing. If the user switches to a different type of ICU (e.g., from ID Based to Group Based) they are brought back to the Overall Compliance tab. If they stay in the same type of ICU they remain on the same tab.

Figure 7: Group Based Options - a screenshot from the original Hand Hygiene Performance application. Used for changing the user's password and switching ICUs.

2.3.2 ID Based Functionality

The ID Based interface is similar to the Group Based interface. It too is composed of different tabs. This interface is filled by dummy data, so some tabs may show strange readings. Again, the first tab is the Overall-Compliance View (see Figure 8). The difference from Group Based is that ID Based uses patient contacts instead of enters and exits for determining compliance.

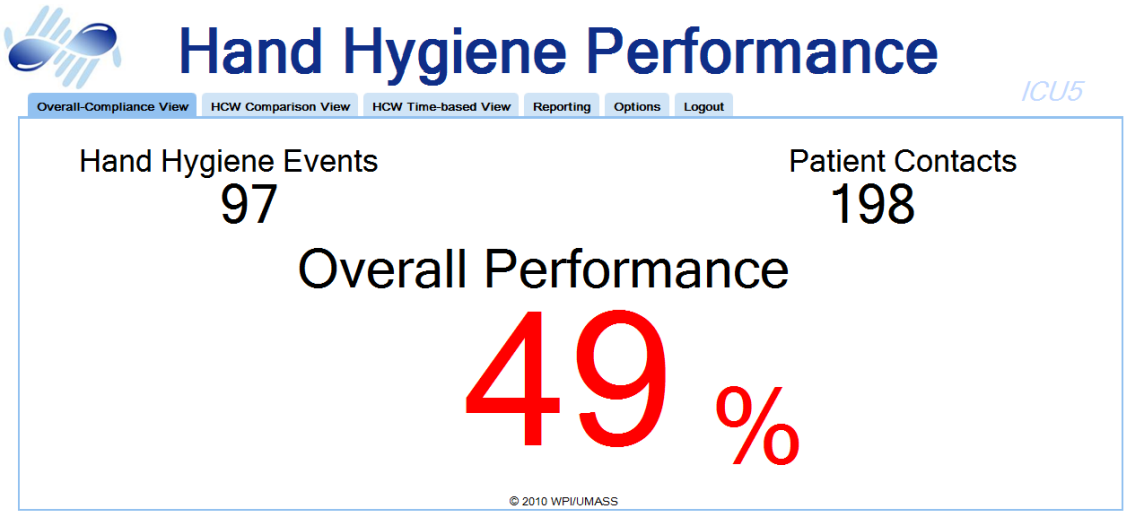


Figure 8: ID Based Overall-Compliance View - a screenshot from the original Hand Hygiene Performance application. Used for short term statistics on the ICU's performance.

The second tab is a new tab, HCW Comparison View (see Figure 9). The purpose of this tab is to give users the ability to review compliant versus non-compliant patient contacts within an ICU. The user applies filters on the right side of the screen to generate a chart on the right. The user first picks which ID badge group they wanted to review (e.g. all, occupation like RN, shift or individual ID) and the time range (the same four default time ranges used throughout the Hyreminder system).

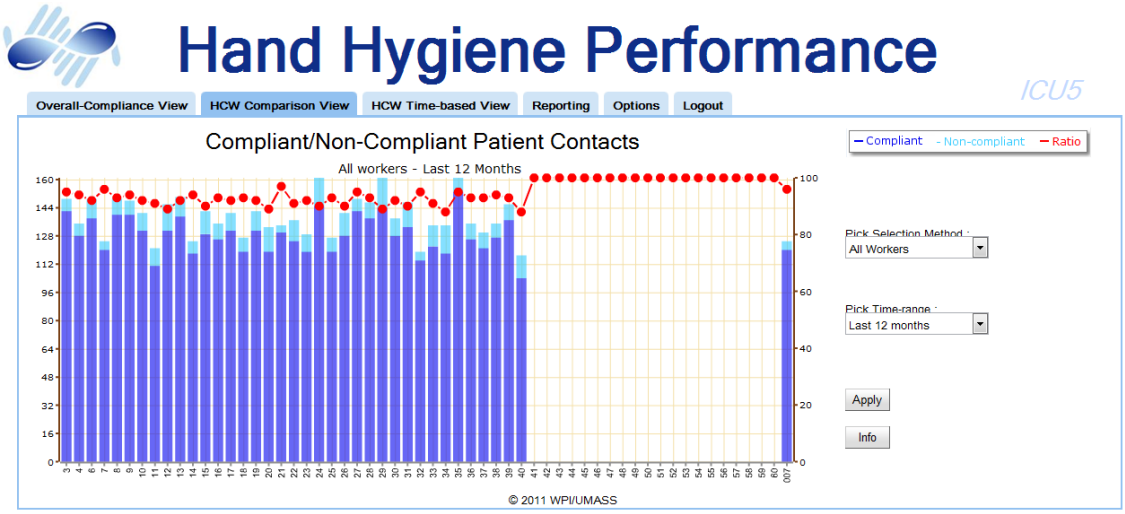


Figure 9: ID Based HCW Comparison View - a screenshot from the original Hand Hygiene Performance application. Used for comparing the performance of workers in the ICU. There is no data for some workers because this system was never live, and those workers didn't get any dummy data.

The third tab is HCW Time-based View (see Figure 10). This tab has the same purpose as Group Based's Detailed Statistics, and is similar in that it has four charts with compliance rates over the same four time ranges. The difference is that the ratio is between compliant and noncompliant patient contacts. The user selects which ID badge groups, using the same options as on HCW Comparison (Figure 9), to see charts for.

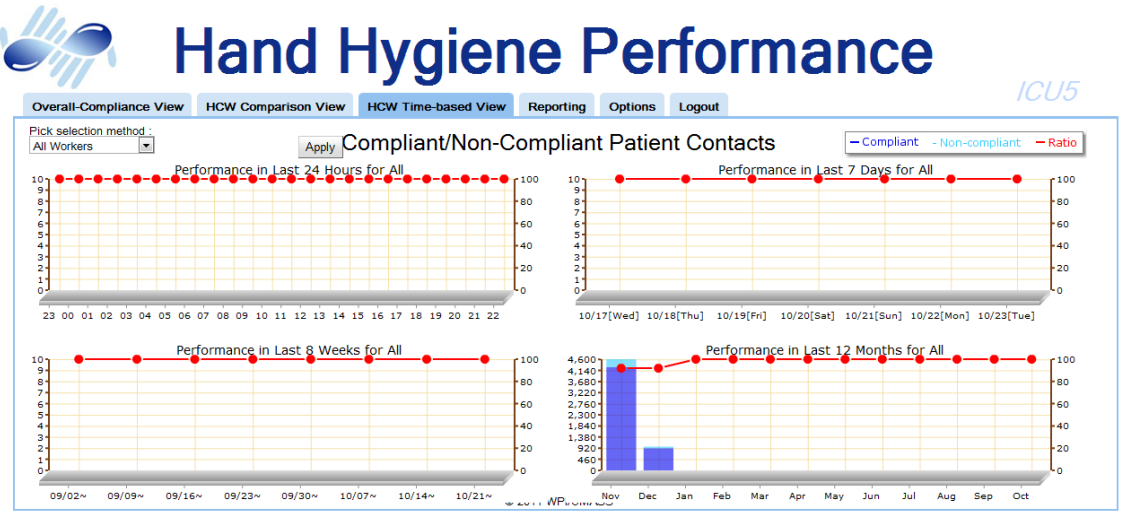


Figure 10: ID Based HCW Time-Based View - a screenshot from the original Hand Hygiene Performance application. Used for longer term statistics on the ICU's performance.

The fourth tab is Reporting (see Figure 11). This tab is similar to Group Based's Reporting, however the ID Based Reporting doesn't have any preset buttons. The user can use the drop-down menus to build a custom report. First they select which ID Badge group to use, with the same options from HCW Comparison, and then select a time unit, time range and aggregation, same as Group Based's Reporting.

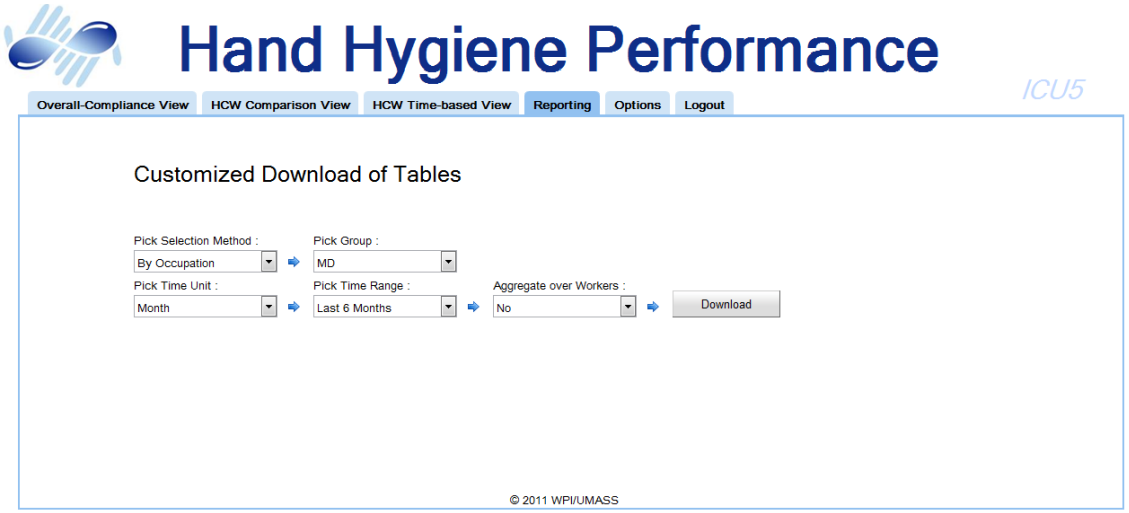


Figure 11: ID Based Reporting - a screenshot from the original Hand Hygiene Performance application. Used for downloading long term statistics on the ICU's performance.

The fifth tab is Options (see Figure 12). The tab is identical to the Group Based Options tab.

Figure 12: ID Based Options - a screenshot from the original Hand Hygiene Performance application. Used for changing the user's password and switching ICUs.

2.3.3 Multi-ICU Functionality

The Multi-ICU interface has similar tabs to both the Group Based and the ID Based interfaces. The first Multi-ICU tab is the Overall-Compliance View (see Figure 13). This tab allows users to see the compliance of all units. Displayed on the screen is the hand wash count, the total room enters/exits, and the patient contacts. Again displayed in a large font is the overall performance.

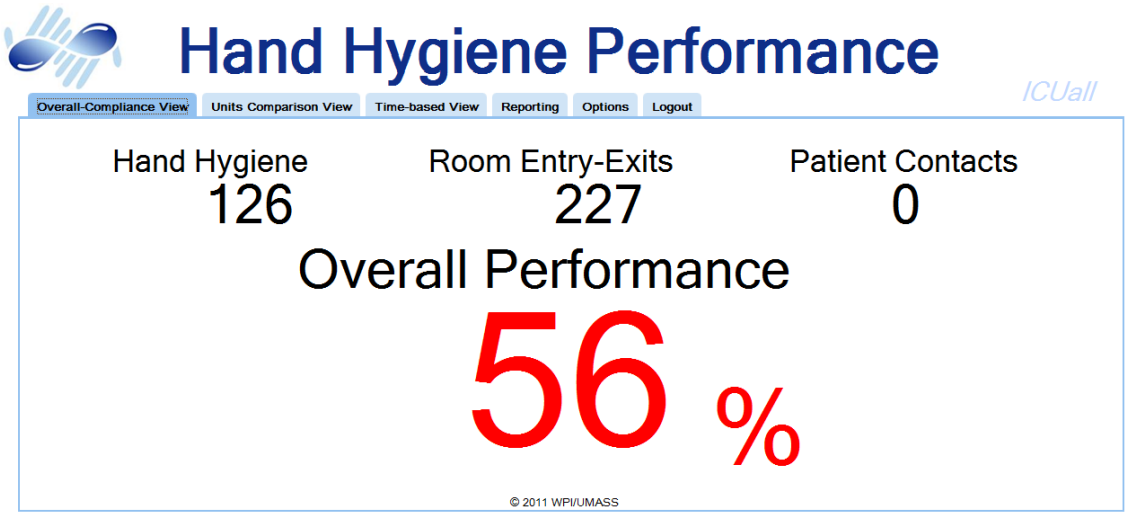


Figure 13: Multi-ICU Overall-Compliance View - a screenshot from the original Hand Hygiene Performance application. Used for short term statistics on all ICUs performance.

The second tab is the Units Comparison View (see Figure 14). This tab allows the user to select ICUs for comparison and the time range to compare by (the same four time ranges). After selecting ICUs and a time range, a chart is populated on the left side of the screen. The chart shows ICUs across the x-axis and event counts on the left y-axis, with a ratio overlaid on top (right y-axis).

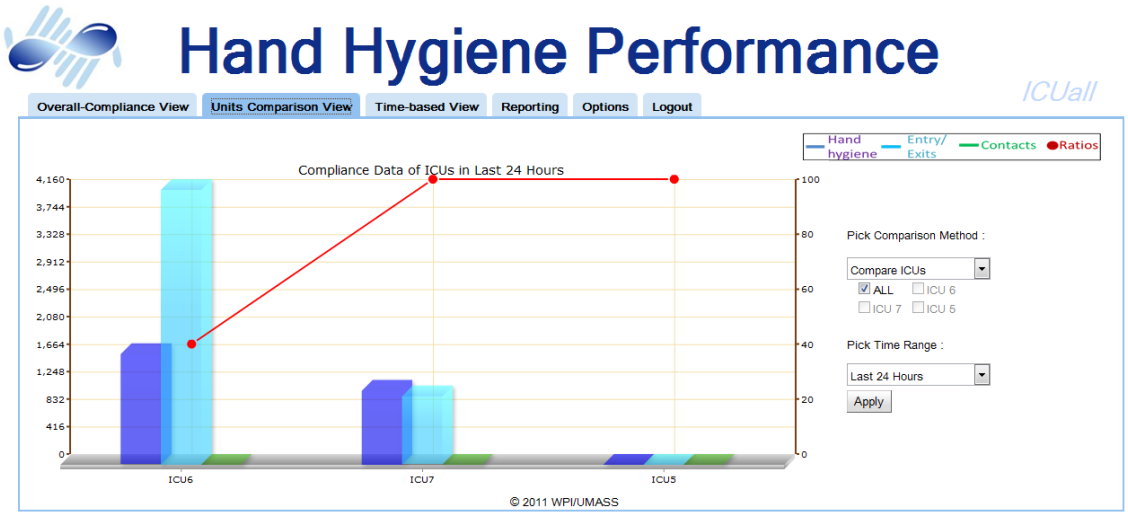


Figure 14: Multi-ICU Units Comparison View - a screenshot from the original Hand Hygiene Performance application. Used for comparing performance between ICUs.

The third tab is the Time-based View (see Figure 15). This tab is similar to the Group Based Detailed Statistics and the ID Based HCW Time-Based interfaces. On the left of the screen, the user selects ICUs to compare and four charts are populated in the standard time ranges.

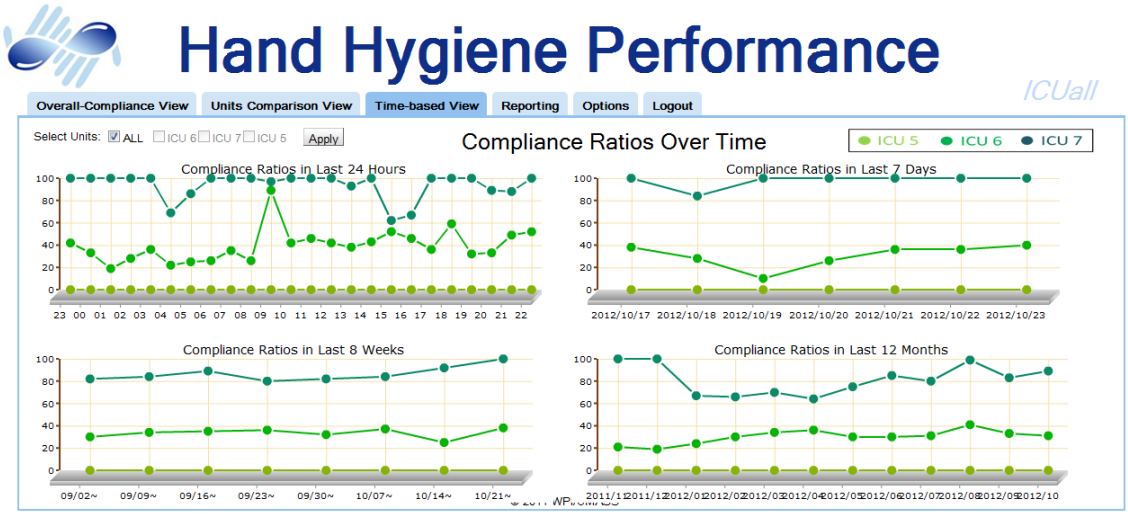


Figure 15: Multi-ICU Time-based View - a screenshot from the original Hand Hygiene Performance application. Used for longer term statistics on all ICUs performance.

The fourth tab is Reporting (see Figure 16). This tab was not completed, and thus is shown to

the users as a blank tab. The fifth tab is again Options (see Figure 17), same as the Group Based and the ID Based interfaces.

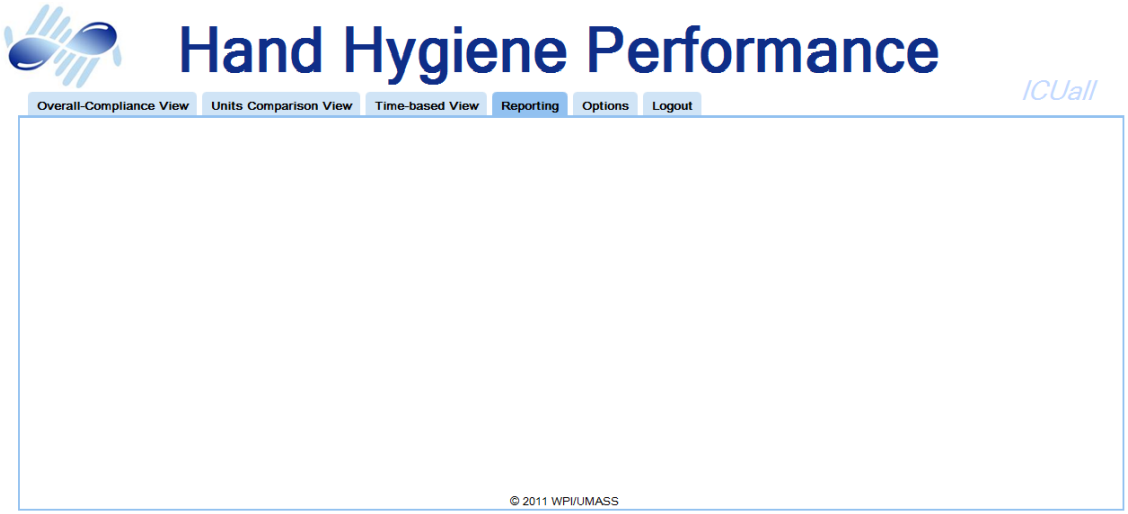


Figure 16: Multi-ICU Reporting - a screenshot from the original Hand Hygiene Performance application. Would be used for downloading long term statistics on all ICUs performance.

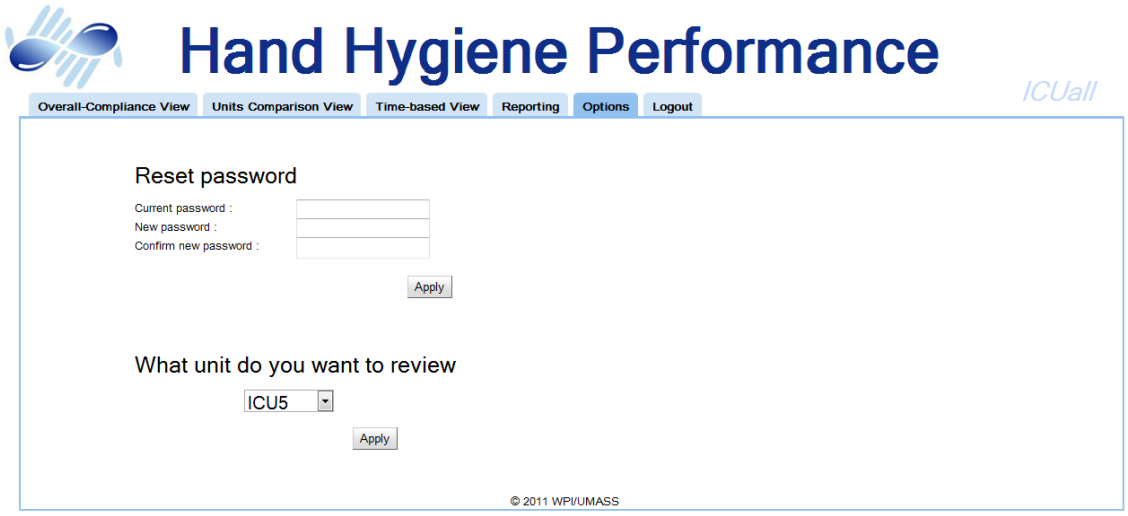


Figure 17: Multi-ICU Options - a screenshot from the original Hand Hygiene Performance application. Used for changing the user's password and switching ICUs.

2.3.4 Anomaly Functionality

As described above in Section 2.3, the Anomaly interface allows an IT staff the ability to troubleshoot the system by providing them the needed information on malfunctioning sensors. The Anomaly basic interface is the same as the Hand Hygiene Performance application's interface, however the tabs are different. The first tab of the Anomaly application is Current Errors (see Figure 18). The purpose of this tab is to see all of the sensors in an ICU with errors. The user first selects an ICU from the drop-down menu on the right side of the screen. A list of sensors with errors is populated on the left. Each sensor has a location, type, and error type and reported time. The user can then locate the sensor in the hospital and attempt to fix it.

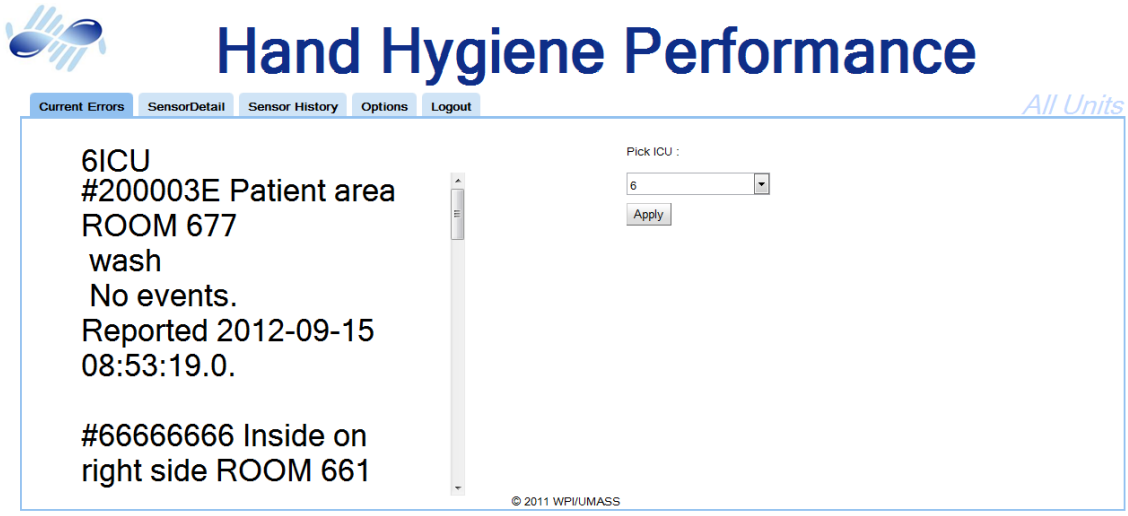


Figure 18: Current Errors - a screenshot from the original Anomaly application. Used for viewing which sensors have errors in the selected ICU.

The second tab is Sensor Detail (see Figure 19). This tab gives the user details on a particular sensor. The user first selects a sensor from the drop-down menu of the right side of the screen. The details of the sensor appear on the left. Details include the name, location, and current status. The page also contains the history of the sensor – including events in the past month and a count of each type of error – located in the middle of the screen. This screen wasn't properly finished, as text goes off the screen and there is no scroll bar.

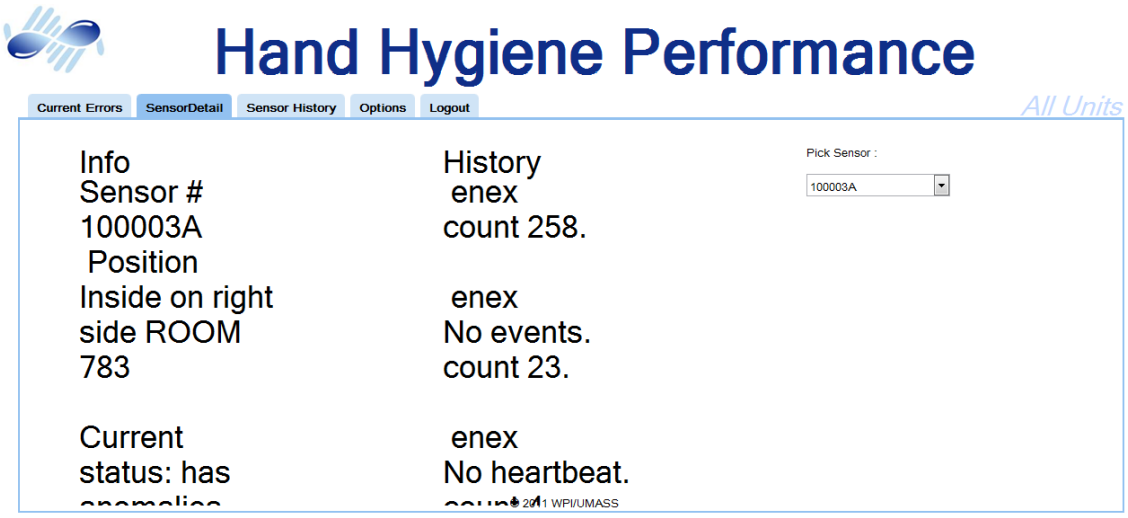


Figure 19: Sensor Detail - a screenshot from the original Anomaly application. Used for viewing details about a specific sensor.

The third tab is Sensor History (see Figure 20). This tab gives a chart with the hits on a particular sensor in the past twenty-four hours. The user selects a particular sensor from the drop-down on the right, and a chart is populated on the left. The user can scroll backward and forward in time using the arrows at the bottom of the chart.

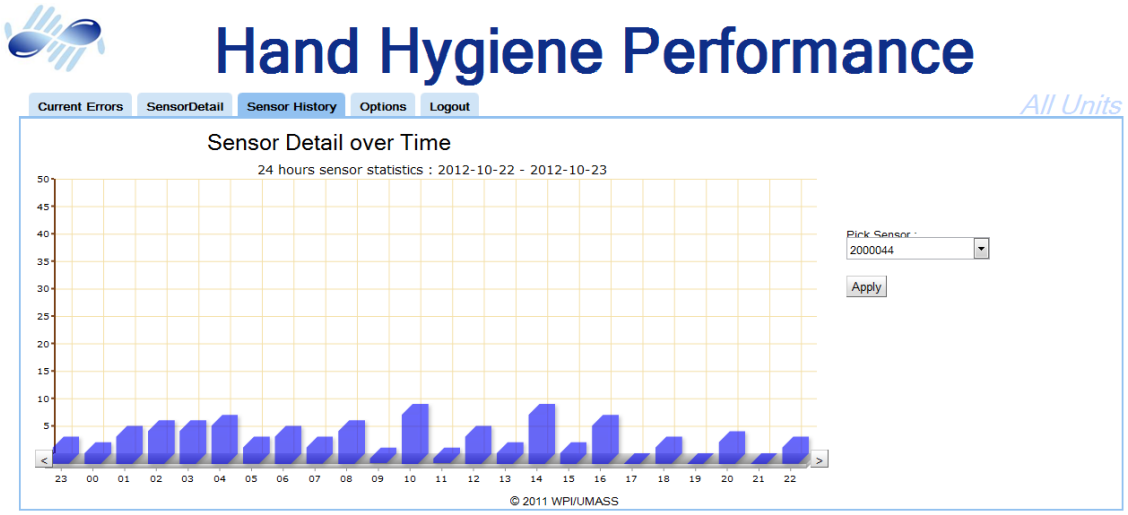


Figure 20: Sensor History - a screenshot from the original Anomaly application. Used for viewing hits on the selected sensor.

The fourth tab is Options (see Figure 21). This tab is nearly identical to the Options tabs in Hand Hygiene Performance, however there is no drop-down for switching between different ICUs – the user is always in all ICUs. The typical user for the Anomaly application is a system administrator who takes care of the full system, and didn't need to get a view into a particular subspace configuration like an ICU.

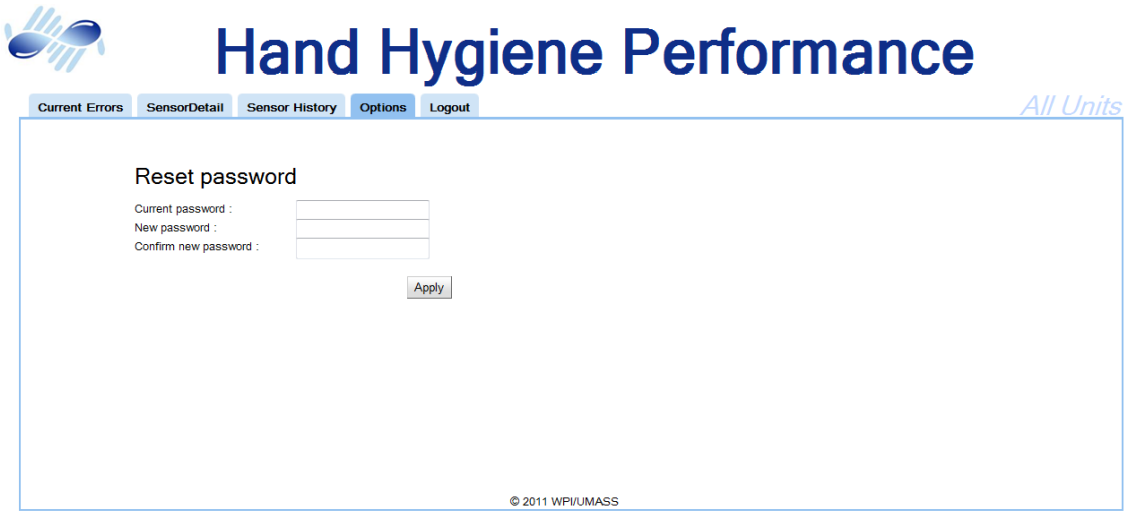


Figure 21: Options - a screenshot from the original Anomaly application. Used for changing the user's password.

2.3.5 General Architecture

The general architecture of the original Hyreminder system with the Google Web Toolkit framework consists of two main packages: com.mycompany.project.client and com.mycompany.project.server. The client package has the GroupView class, that contains all of the user interface elements (com.google.gwt.user.client.ui). Also in the client package are interfaces that contain methods to be called by the client, for example, there's the GetHCWList and GetHCWListAsync. The server package contains implementations for these methods, for example GetHCWListImpl. These implementations contain code for getting information from the database and perform any needed calculations before the client is able to display the information. All database calls are handled in the DB_Operations class in the com.mycompany.project.server.db package.

2.3.6 Database Tables

The data from the sensor hits are stored in three tables: enexlive (Figure 22), washtablelive (Figure 23), and contact1 (Figure 24). These tables are populated by the Data Grabber program (see

Section 2.3.7). These three tables contain the majority of the information used in the Hyreminder web application.

```
mysql> describe enexlive;
+------------+-------------+------+-----+---------------------+-------+
| Field      | Type        | Null | Key | Default             | Extra |
+------------+-------------+------+-----+---------------------+-------+
| txid       | varchar(50) | NO   | PRI |                     |       |
| time       | timestamp   | NO   |     | 0000-00-00 00:00:00 |       |
| badgeid    | varchar(50) | YES  |     | NULL                |       |
| event_type | varchar(10) | YES  |     | NULL                |       |
| sensorid   | varchar(50) | YES  |     | NULL                |       |
| shift      | varchar(50) | YES  |     | NULL                |       |
| unitid     | varchar(50) | YES  |     | NULL                |       |
+------------+-------------+------+-----+---------------------+-------+
7 rows in set (0.00 sec)
```

Figure 22: The enexlive table schema. This tables contains the door sensor hits.

```
mysql> describe washtablelive;
+----------+-------------+------+-----+---------------------+-------+
| Field    | Type        | Null | Key | Default             | Extra |
+----------+-------------+------+-----+---------------------+-------+
| txid     | varchar(50) | NO   | PRI |                     |       |
| badgeid  | varchar(50) | YES  |     | NULL                |       |
| time     | timestamp   | NO   |     | 0000-00-00 00:00:00 |       |
| shift    | varchar(50) | YES  |     | NULL                |       |
| sensorid | varchar(50) | YES  |     | NULL                |       |
| unitid   | varchar(50) | YES  |     | NULL                |       |
+----------+-------------+------+-----+---------------------+-------+
6 rows in set (0.00 sec)
```

Figure 23: The washtablelive table schema. This tables contains the hand wash sensor hits.

```
mysql> describe contact1;
+-----------+-------------+------+-----+-------------------+-------+
| Field     | Type        | Null | Key | Default           | Extra |
+-----------+-------------+------+-----+-------------------+-------+
| txid      | varchar(50) | YES  |     | NULL              |       |
| badgeid   | varchar(50) | YES  |     | NULL              |       |
| time      | timestamp   | NO   |     | CURRENT_TIMESTAMP |       |
| shift     | varchar(50) | YES  |     | NULL              |       |
| compliant | tinyint(1)  | YES  |     | NULL              |       |
| sensorid  | varchar(50) | YES  |     | NULL              |       |
| unitid    | varchar(50) | YES  |     | NULL              |       |
+-----------+-------------+------+-----+-------------------+-------+
7 rows in set (0.00 sec)
```

Figure 24: The contact1 table schema. This tables contains the patient contact sensor hits.

In addition to the sensor hits tables, Hyreminder also uses the user table (Figure 25). This table is used for checking log in details. Additionally, the table is used in the ID Based system to get details about the staff responsible for the patient contact.

```
mysql> describe user;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| userid   | varchar(20) | YES  |     | NULL    |       |
| fname    | varchar(50) | YES  |     | NULL    |       |
| lname    | varchar(50) | YES  |     | NULL    |       |
| role     | varchar(20) | YES  |     | NULL    |       |
| level    | varchar(20) | YES  |     | NULL    |       |
| icuCode  | varchar(20) | YES  |     | NULL    |       |
| password | varchar(50) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

Figure 25: The user table schema. This tables contains the user information

In the Anomaly application, three tables are used to store information about sensors: sensor_real (Figure 26), sensorErrorCur (Figure 27), and sensorErrorHistory (Figure 28). Discovery of these errors, identification of the error type, and anomaly correction are all challenging issues that are beyond the scope of this MQP.

```
mysql> describe sensor_real;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| sensorid  | varchar(20)  | NO   |     |         |       |
| location  | varchar(100) | YES  |     | NULL    |       |
| gatewayid | varchar(20)  | YES  |     | NULL    |       |
| function  | varchar(100) | YES  |     | NULL    |       |
| unitid    | varchar(20)  | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Figure 26: The sensor_real table schema. This tables contains sensor information.

```
mysql> describe sensorErrorCur;
+------------+-------------+------+-----+---------------------+-------+
| Field      | Type        | Null | Key | Default             | Extra |
+------------+-------------+------+-----+---------------------+-------+
| sensorid   | varchar(20) | YES  |     | NULL                |       |
| event_type | varchar(10) | YES  |     | NULL                |       |
| starttime  | timestamp   | NO   |     | 0000-00-00 00:00:00 |       |
| status     | varchar(20) | YES  |     | NULL                |       |
| count      | int(11)     | YES  |     | NULL                |       |
| reported   | char(1)     | YES  |     | NULL                |       |
+------------+-------------+------+-----+---------------------+-------+
6 rows in set (0.00 sec)
```

Figure 27: The sensorErrorCur table schema. This tables contains currently with an error.

```
mysql> describe sensorErrorHistory;
+------------+-------------+------+-----+---------------------+-------+
| Field      | Type        | Null | Key | Default             | Extra |
+------------+-------------+------+-----+---------------------+-------+
| sensorid   | varchar(20) | YES  |     | NULL                |       |
| event_type | varchar(10) | YES  |     | NULL                |       |
| starttime  | timestamp   | NO   |     | CURRENT_TIMESTAMP   |       |
| endtime    | timestamp   | NO   |     | 0000-00-00 00:00:00 |       |
| status     | varchar(20) | YES  |     | NULL                |       |
+------------+-------------+------+-----+---------------------+-------+
5 rows in set (0.00 sec)
```

Figure 28: The sensorErrorHistory table schema. This tables contains the history of sensor errors.

2.3.7 Additional Programs

In addition to the Hyreminder web interface we worked on, there are additional parts to the system, including the Performance Screensaver, the Data Grabber, the Anomaly Detector and the Alerting System. All of these programs are beyond the scope of this MQP, however a brief overview can be useful to understand context for the work that we completed.

The Performance Screensaver is on display in the ICUs where all of the nurses and doctors can see it. It's used to display real time feedback on how the staff are performing. It appears similar to the Overall Compliance tab, displaying counts of hand washes, enters and exits, and patient contacts, as well as the compliance rate.

The Data Grabber is the bridge between the physical sensors and the Hyreminder system. At periodic intervals, it grabs in near normal time new events sensed in the physical environment and pushes them into the Hyreminder system database. The Data Grabber, in addition to detecting events on-the-fly, will also collect statistics about expected behavior of sensors. The Data Grabber takes care of filtering some outliers, such as the large numbers of hits that could result from a malfunctioning sensor, to give the Hyreminder system cleaner input. Last but not least, this module also inserts the relevant event data in the above Hyreminder System's tables for use by the Hyreminder web application.

The Anomaly Detector reads the stream of data from the sensors and detects information that varies from the norm. Once an anomaly is detected, the information displayed is corrected, so that performance is not skewed by malfunctioning hardware. The anomaly is noted in a table in the database, where the Anomaly web interface pulls from. Additionally, the Alerting System alerts the admin staff of the Hyreminder system when an anomaly is detected in a sensor.

3 Reengineering Hyreminder

The Hyreminder System had a number of issues with the code itself, as we will discuss in detail below. These issues then lead our team to research alternative frameworks, ultimately deciding to rebuild the project in the Spring MVC Framework.

3.1 Analysis of Existing Hyreminder System

In this section we will discuss how we evaluated the Hyreminder system and what shortcomings we discovered.

3.1.1 Comments and Class Files

Several issues with the code base were found by performing static analysis of the code. Using the PMD[15] plugin for Eclipse, approximately 12% of the lines of code had a significant warning or error. Some of these errors included a class with only a private constructor, variables that could be declared final to allow more compiler optimizations, and variables that were declared but then never used. While many of these found violations would be trivial to fix, some, such as classes being too large or program flow issues, would require much more time and effort to clean up.

The code base also had very little commenting, either in the form of inline comments or Javadoc style class methods. Additionally, large portions of unused or inactive code were present in several files as block comments.

3.1.2 Testing

The Hyreminder system had very little testing attached to it. There were no unit tests, and the only method to determine if the system was working correctly was to perform manual testing in the browser itself. From a maintainability standpoint, this was very bad, since if a developer decided to make a change there was very little the developer could do to ensure that the change didn't break something else other than by manual testing, which is time consuming and unreliable.

3.1.3 Usability

To test the usability of the applications, we simply used them ourselves. As we had little interaction with the application before, we were perfect to test the system as new users. We noticed several aspects that were difficult or confusing to use. One usability issue was the fixed size of the browser window. Due to the lack of scaling, the Overall Compliance tab, for example, would always be rendered as if it was on a large screen, which caused scrollbars to appear if the browser window was resized or the screen resolution was too low. Additionally, there was a lack of cross-browser

[15] PMD. (2013). http://pmd.sourceforge.net/

compatibility. The system would work only with Internet Explorer 8; any other browser (e.g. Firefox, Chrome, Internet Explorer 9, or Safari) would result in a message stating the incompatibility instead of bringing up the normal login screen. As the system was built with Google Web Toolkit, it was confusing as to why it could not be displayed on Google Chrome.

Another usability issue was the navigation. To change ICUs, a user would have to first go to the Options tab, then select an ICU from a drop-down menu, which would then redirect the user to the new ICU's Overall Compliance tab. For the ordinary user, a head nurse with only one ICU, this wouldn't be a problem. However, for administrators with access to multiple ICUs, this would make it a tedious process to download reports for multiple different ICUs. Another aspect of poor navigation was in the Anomaly web application. There was no connection between pages with reference to a sensor ID number, so the user would have to continuously relocate the number in a drop-down.

Some other issues that we noticed were problems were the speed at which the charts were loaded and the reports were generated. In some cases, these tasks were so slow that we wondered if they were functioning or not. We made note of all of these usability issues – browser and navigation issues and the speed of charts and reports – and knew we wanted to fix them.

3.2 Options for Going Forward

After reviewing the current state of the code, we knew we needed to make some major changes to it, in order to align with our main goal of creating a flexible and maintainable system. The most obvious option would be to clean up the shortcomings in the existing code. The second option would be to to rebuild the system from near scratch, either using the existing framework, Google Web Toolkit, or a new framework such as Spring MVC.

3.2.1 Why Not Google Web Toolkit?

GWT attempts to bring the desktop application paradigm to the web and causes more problems than solutions. One feature of GWT is the abstraction of Javascript into Java. According to Jeff Atwood, "all good programming abstractions are failed abstractions."[16] In his article Atwood brings up the fact that heavily abstracted frameworks result in unnecessary overhead on simple tasks. GWT not only abstracts Javascript, but it also abstracts away HTML. The commonly used markup language is interwoven into Java objects resulting in a complicated series of parent-child relationships. With multiple developers the task of deciphering this code at a later date could prove problematic.

The ThoughtWorks Technology Advisory Board has stated its opinion of GWT by saying "GWT

[16] Atwood, Jeff. (2009, June). All Abstractions Are Failed Abstractions. http://www.codinghorror.com/blog/2009/06/all-abstractions-are-failed-abstractions.html

is a reasonable implementation of a poor architectural choice. GWT attempts to hide many of the details of the web as a platform by creating desktop metaphors in Java and generating JavaScript code to implement them.'[17] They go on to agree with the statements of Atwood by saying that it is impossible to hide such complex abstractions without some problems eventually popping up. Javascript is a useful language by itself for client-side scripting. When abstracted into Java, you end up with a union of the problems and an intersection of the benefits.[18] GWT is a good choice for building a simple, desktop-like application on the Web, however when the application becomes sufficiently complex, any advantages brought by GWT become moot.

3.2.2 Why Spring?

The Spring Web MVC Framework[19] is a well-designed, robust and flexible framework especially for those rapidly developing web applications using the Model-View-Controller (MVC) design pattern. Spring is a lightweight framework that enables developers to build enterprise-level applications with Plain Old Java Objects (POJOs). The Spring Framework itself is open-source with numerous developers. However it is also supported commercially by VMware who makes profit through selling books and consulting on Spring. This means that Spring has a large development and support network which makes it ideal for a long-term project. Much of Spring is designed with the idea of "convention-over-configuration" to allow for rapid development. In other words, Spring by default will be configured the way that the majority of the developers will use it, allowing those developers to skip the necessary configuration. Despite this, it is still possible to configure Spring to fit specific needs that may not be the convention, a developer would be required to take the configuration steps that would be standard with a different framework. Spring also provides an easier testing experience for unit testing by injecting test data into JavaBeans[20] and building mock classes to simulate Java HTTP objects.

3.3 The Reengineering

After getting permission to reengineer the Hyreminder web applications we first created the new Spring project. An early simple Spring MVC project with bare bones functionality was created for us to use as a base as we added more content. The CSS and general Hyreminder page template was copied over from the GWT system to be used as a starting point.

We started re-coding with the Group Based functionality. Each of us took different sections to work on. We reused the original code where possible, cleaning it up as we went. The database queries

[17] ThoughtWorks Technology Advisor Board. (2011, July). Technology Radar.

[18] C, Nick. (2011, January). When not to use Google Web Toolkit?. http://programmers.stackexchange.com/questions/38441/when-not-to-use-google-web-toolkit

[19] Spring Source Community. (2013). http://www.springsource.org/

[20] Vogel, Lars. (2009, August). Dependency Injection with the Spring Framework - Tutorial. http://www.vogella.com/articles/SpringDependencyInjection/article.html

were reused, modifying them where necessary to make them run more efficiently (see Section 4.1). After the Group Based functionality was brought up to the original system's level of functionality, we moved on to the ID Based and Multi-ICU functionalities. These two functionalities had some similarities to the Group Based functionality, so we were able to refactor and reuse some of the code we previously wrote. Both the ID Based and Multi-ICU versions had some incomplete functionality in the original system, such as reporting, so we finished those off (see Section 4.1). Lastly, we worked on the Anomaly web application, which also had an uncompleted feature: sensor commenting (see Section 4.2).

3.3.1 The Model-View-Controller Pattern

Spring uses the Model-View-Controller design pattern, which is an effective way to develop a Web Application.[21]

We created numerous models, such as a Room, Sensor and SensorLocation. These models hold information that we pulled from the database, so they can be easily manipulated and displayed. The original code had no notion of models, so all of these were created from scratch. In general, these models are simple Java objects with just constructors, getters and setters.

Views are .jsp (JavaServer Pages) files that dynamically create an HTML document. We needed to rebuild the user interface as none of the original code was compatible with the .jsp files. We used the styles defined in the original code to create the .css (Cascading Style Sheets) files and we modeled the placement of HTML elements off the original. This preserved the look and feel of the original Hyreminder web application.

Controllers link the models and views. In Spring, the controller receives the page request and orchestrates getting the data from the database, manipulated as needed, and given to the view for display. The controllers make use of services for retrieving and manipulating the data.

For example, when the OverallComplianceController receives the request from the client for /ICU6/hygiene it calls methods from the OverallComplianceService for the appropriate data. The controller puts the data into a Spring Model for use in the view: hygiene.jsp.

3.4 Static Code Analysis Statistics

To compare the old code with our new, reengineered code in a concrete and objective manner, we used CodePro Analytix[22] to analyze both systems for quality metrics and rule violations. The quality metrics measure statistics such as lines of code, average number of lines of code per method, and the abstractness of the code. The rule violations flag segments of code that break design

[21] msdn. Model View Controller. http://msdn.microsoft.com/en-us/library/ff649643.aspx
[22] CodePro Analytix. https://developers.google.com/java-dev-tools/codepro/doc/

practices based on a ruleset. These flags are different colors based on severity, and range from the minor warning of a blue flag up to the critical issue red flag.

3.4.1 CodePro Metrics

There were several metrics generated by the CodePro tool that we used to compare the two systems. This section first shows how our code compares to the original GWT version in several metrics in Table 1, and then describes what each of those metrics mean.

Table 1: CodePro Metric comparison of the old and new systems.

Metric	**Old (GWT)**	**New (Spring)**
Abstractness	19.9%	15.5%
Average Block Depth	1.42	0.92
Average Cyclomatic Complexity	3.59	1.48
Comments Ratio	17.4%	7%
Lines of Code	19,169	11,372

Abstractness - Abstractness measures the ratio of abstract classes and interfaces to total types. In general, more abstractness is better.

Average Block Depth - This averages the nested block levels over each method. A getter or setter would have a block depth of 0, while a complicated nested for loop would have a much higher depth. Having low average block depth is desirable.

Average Cyclomatic Complexity - This metric represents the average cyclomatic complexity of each method in the project. The cyclomatic complexity of a method measures how many distinct execution paths exist in it. For example, a simple getter would have a complexity of 1, while a method with many if statements would have a much higher complexity. In general, the smaller the average cyclomatic complexity, the less each individual method has to do, and thus a small value for this metric is ideal.

Comments Ratio - This is computed simply as the number of comment lines divided by the total number of lines of code. This is a rough metric for measuring how much documentation is in the code. This can be misleading, as this does not measure how useful the comments are, however in general a higher ratio is better to have than a lower ratio.

Lines of Code - This is not very useful in and of itself for comparing projects, but it does give an indication of the scale of a project.

Overall, these metrics provide some interesting results. It is important not to only look at the numbers but also what the numbers mean. For example, with the comment ratio, our system has about half the original system's ratio. However, large sections of the old code are commented out, and there are multiple instances of unnecessary comments that do not contribute much to understanding the code. By contrast, when developing our code we strived to make it self-documenting,

so that the code structure and variable names would explain themselves, reducing the need for comments on every line. The reason that our code has a much lower average cyclomatic complexity is because it is much more streamlined into methods. We have many private helper methods that do one task only. Similarly, with our code having a lower average block depth, we designed our methods to be as simple as possible, avoiding nested conditionals or loops where possible.

3.4.2 CodePro Code Audit

The ruleset we used for the code audit was a modified version of the standard ruleset used by CodePro. Our version was scaled down from the normal ruleset, as we removed rules that we as a group disagreed with based on our programming experiences. Our final ruleset included 325 rules. By applying the same rules to both systems, we were able to compare the frequency of moderate yellow and severe red flags in both systems, as seen in Table 2.

Table 2: CodePro Audit results, as total flags and percent of total code.

Flag Type	**GWT Total**	**GWT Percentage**	**Spring Total**	**Spring Percentage**
Yellow (Medium)	98	0.51%	9	0.08%
Red (High)	1568	8.18%	0	0%

In the process of completing our reengineering, we addressed all of the rule violations in our code, fixing nearly all of them. Some of the red flag violations on the original system mark critical areas where the application's security could be compromised, such as code that allows SQL injections. However, the real difference between our code and the original is that, since we wanted to make a well-built and reliable codebase, we made sure we went through the code and cleared any critical violations. The remaining yellow violations in our code are actual violations of the rules, but for which there is no easy solution. For example, in one method in the stress test framework, we used the thread.stop() method, which is deprecated. Although this is a violation and flagged as such, our reasons for use of the method are documented thoroughly right before the statement itself.

4 New Features

After converting the Hyreminder system from Google Web Toolkit to Spring and the web application was back to running to the original level of functionality, we implemented more features for the system. We started by completing the reports for ID Based and Multi-ICU. We added commenting on sensors. Lastly, we tackled the issue of tracking and mapping moving sensors.

4.1 More Reports Available

The original Hyreminder system had the Group Based reporting completed with unstable performance which varies from less than ten seconds to several minutes to generate the report (see Section 7.2.2 for more details). ID Based was only a user interface with no database calls, and Multi-ICU had neither. We implemented the reporting functionality for the ID Based and Multi-ICU systems. While similar to the existing reporting functionality, the different nature of the ID Based system means that there are more options available to the user. For example, the user can choose to group by employee ID or by work shift, and get reports based on those criteria instead of the entire ICU.

In general we preserved the original user interface design from the Google Web Toolkit version. The ID based and Multi-ICU versions both implement drop-down menus for users to select different time units, time ranges and group types. In the ID based version we added parameters of occupation, shift and ID number. In the Multi-ICU version we added checkboxes for each individual ICU unit, enabling users to generate flexible reports based on their needs by indicating their desired settings.

As we updated and added queries associated with the reporting module, we improved performance in reducing the time cost for downloading reports. This was accomplished by combining multiple queries into large queries so that there were less round trips to the database, greatly decreasing the time cost. All queries were moved from database operations class to the database as stored procedures to reduce the queries transmission time cost. See code clips from the two versions: Google Web Toolkit in Figure 29 and Spring in Figure 30.

```
int count = 8;  // 8 weeks
ArrayList<String> listEnExAll = new ArrayList<String>();
ArrayList<String> listHHY = new ArrayList<String>();
while (count > 0) {
    start = sqlRange.remove(0);
    end = sqlRange.remove(0);
    DB_Operations.getEnAndEx(stmt, start + " 00:00:00", end + " 23:59:59", listEnExAll, icuCode);
    DB_Operations.getHHY(stmt, start + " 00:00:00", end + " 23:59:59", listHHY, icuCode);
    count--;
}
```

Figure 29: Code clip in Google Web Toolkit version. This code shows the query for getting the Group Based past 8 weeks report data.

```
// get data for 8 weeks
    inp = new ClassPathResource("ExportWeek.xls").getInputStream();
    dateGroupFormat = "%Y-%u";
    startDate = DateTime.now().minusWeeks(8);
// Get data set
    set = op.getHygieneGroupedGB(dateGroupFormat, icuCode, startDate,
    endDate);
```

Figure 30: Code clip in Spring version. This code shows the query for getting the Group Based past 8 weeks report data.

From the code comparison example above, the reengineered hyreminder system is able to generate data of all requests by calling a query only once instead of looping and calling the query eight times to generate the data for the past eight weeks. This has a major impact on the performance by minimizing round-trip transmission costs.

See the developer documentation in Appendix D for more information.

4.2 Sensor Commenting

One desired feature was to be able to add comments on a sensor, such as "battery changed." Since this action is being performed on one particular sensor, we added the commenting interface onto the Sensor Details tab (see Figure 37). After the user submits a comment, the comment details are stored in the newly created sensorComments table (see Figure 31).

```
mysql> describe sensorComments;
+----------+--------------+------+-----+-------------------+-------+
| Field    | Type         | Null | Key | Default           | Extra |
+----------+--------------+------+-----+-------------------+-------+
| sensorid | varchar(50)  | YES  |     | NULL              |       |
| time     | timestamp    | NO   |     | CURRENT_TIMESTAMP |       |
| comment  | varchar(255) | YES  |     | NULL              |       |
| userid   | varchar(50)  | YES  |     | NULL              |       |
+----------+--------------+------+-----+-------------------+-------+
4 rows in set (0.00 sec)
```

Figure 31: The sensorComments table schema. This tables is used to hold comments on a sensor.

4.3 Mapping Sensors

Doctor Ellison asked us to display where within which ICU each static and moving sensor was located. For the moving sensors, such as those on beds and equipment, this requires storing locations of each sensor. The recent history of a moving sensor is used to determine its current or last known location. Additionally, having the map of where sensors are believed to be will help with maintenance of the system. Sensors could go missing for a number of reasons - either the bed has

moved outside of the sensor net area, for example to an operating room without no sensors, or the sensor is malfunction due to battery or other hardware issues.

4.3.1 Restructuring Sensors

When we started to discuss how to implement mapping of sensors, we learned that physical sensor IDs could be recycled after being deactivated. To account for this, when a sensor was replaced in the past, all instances of the old ID were replaced with the new ID in the database. As the database grows this will become increasingly resource consuming. To deal with this problem, we implemented a logical sensor concept in addition to the physical sensor concept. A logical sensor has a unique ID that will be used as a primary key in the tables, which will be connected to one "current" physical ID that represents the actual sensor hardware unit. In addition, the logical sensor will have a list of physical sensor IDs that have been connected in the past to the logical sensor. Only one physical sensor at a time will be active per logical sensor. Lastly, a logical sensor will have a location. For static sensors, this will be set at creation of the logical sensor. However, mobile sensors will be attached to hospital beds and will have their locations calculated based on proximal sensor hits logic, explained in more detail below.

4.3.2 Tracking Moving Sensors

The sensor tracking system keeps a record of the time difference between a worker triggering a handwash event and triggering a patient contact event - which is a nurse approaching a bed sensor. Since the handwash sensors are fixed in location, by setting distances of each handwash sensor to several nearby rooms during the initial setup, the travel times for an employee to get from a handwash sensor to a bed sensor will allow the system to get a fairly accurate location for the bed, mostly due to the relatively large number of encounters a bed will have with one or more employees throughout any given day. Though it will be impossible to achieve 100% accuracy about the precise physical location of a bed, a good enough estimate provides a good starting point to search for a missing bed.

The logic for determining a bed location comes in two parts. The first part generates a number of separate confidence events for a given bed and location based on information about bed contacts and handwash sensors. These are generated using the distances between the sensors and locations as described above. When a hand wash sensor and a bed sensor have events logically close to each other, in time, this will generate and store a confidence event for each configured location between the location and the bed. The confidence is calculated using an exponential scaling on the time difference between the two events. If there is no difference, the confidence is 100%, whereas if the time difference is equal to or greater than the configured distance then the confidence 0%. If a sensor has no configured distances it will not be considered for these calculations.

The second part of the logic uses the confidence events generated by the first part to determine a list of location confidence pairs for a given bed. It aggregates all of the recent confidence events for the bed within a certain threshold. Each confidence is decayed slightly based on how long it has been since the event has happened. Next, all of the decayed confidence events are weighted exponentially based on how recent they are and then averaged together to produce a total confidence for a given location and bed. We used one minute as a default threshold, however experimentation would need to be used to determine a more accurate threshold for an active ICU.

For example, given a threshold of one minute, if there was one confidence event of 100% 30 seconds ago, we would report a 96% confidence due to the confidence decay. We are still pretty sure it is there, but given it has been 30 seconds we are not 100% sure. For another example, if we had two events, the first one second ago with 100 confidence, and one 50 seconds ago with 1% confidence, we would report 99% confidence as the 1% confidence event is weighted so negligibly in comparison to the 100% confidence event.

In the case where we don't have any events within our one minute threshold, the system will attempt to find the most recent event for a given bed and use just that, no matter how far into the past it will need to go and stretch the threshold accordingly. This will at least give us the last known location of the bed with a relative confidence. As an example, if the last event we had was 3 hours ago with 100% confidence, we would report that we are 38% confident that the bed is in the stated room.

This system is not perfect, as we have had no real data to test it with. Realistically, many of the constants would need to be configured through rigorous data collection and user testing in order to produce accurate estimates. What we have provided is a system that would allow for a future team to easily implement such a system once the data is available.

5 Graphical User Interface Design

This next section describes changes to the graphical user interface. We start with general improvements such as browser compliance, scaling to screen size, and navigation. Next we discuss the changes to the anomaly interface. Lastly, we explain the new user interface used for mapping sensors.

5.1 Browser Based Improvements

One noticeable improvement regarding the user interface is that users can access the application using more browsers. The original system used only Internet Explorer 8. However now users can access the site using additional browsers including Internet Explorer 8 and 9, Chrome, Firefox and Safari. We developed and tested on a number of operating systems including Windows 7, Windows XP and Mac OS.

In addition to cross browser support, there is now better support for window sizes. The original system had the browser use the full screen size instead of the window size. Now users can keep the window at their desired size without needing to horizontally scroll on the page. In Figure 32 the user has the window at their full screen width (1681px x 652px) and in Figure 33 they have reduced the window size to 1020px x 652px. In both versions, they do not need to horizontally scroll. This is accomplished by not setting elements on the page to reside in specific pixel locations based on the screen size.

Figure 32: An example of how the web application acts at a resolution larger than the minimum: at a resolution of 1681px x 652px.

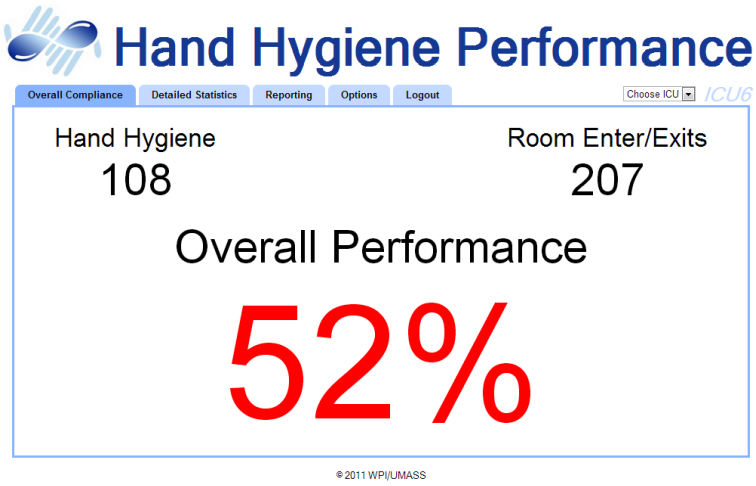


Figure 33: An example of the same screen as Figure 32 reduced the the minimum resolution width: at a resolution of 1020px x 652px.

While this flexibility is good, we still need to keep in mind that layouts can become messed up on small screens. The W3 Consortium advises that many countries still use 800px by 600px screens, but the U.S. typically has larger than 1024px by 768px screens.[23] We have set a minimum width of 1000px, which is the smallest we could set due to the horizontal space needed for the logo and title. With window sizes less than 1000px, the user will still need to horizontally scroll. We deemed this an acceptable trade for keeping the layout looking clean.

5.2 Work Flow Improvements

We improved the user experience by reducing the effort it takes to change the ICU. In the old system, they needed to go to the Options tab, select a new ICU, and be redirected to the Overall Compliance tab. Now there is a drop-down menu at the top of every screen, in the tab bar next to the current ICU (see Figure 34). Changing the ICU keeps the user on the same tab, assuming that the current tab exists in both ICUs, otherwise a similar tab is selected. For example, when switching from an ID Based "HCW Time Based" tab to a Group Based, which has no "HCW Time Based" tab, the system will redirect the user to the Group Based equivalent, the "Detailed Statistics" tab. We added this navigation to both the Hyreminder and Anomaly web application.

[23] W3 Consortium. (2010, November). Display capabilities. http://www.w3.org/International/questions/qa-display-capabilities.en.php

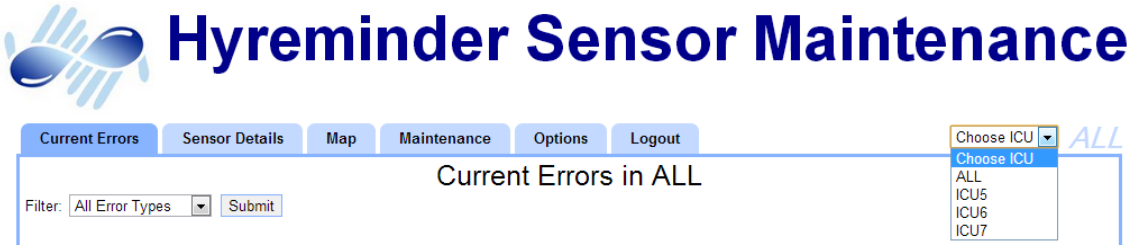


Figure 34: A screenshot showing the ICU switching interface. The user selects the desired ICU from the drop-down and they are automatically redirected to the appropriate ICU.

Another workflow improvement we accomplished through the switch from the Google Web Toolkit framework to the Spring framework was adding different urls for different screens. In the original system, all pages of the application were at: `/liveGroupbased/`. With our system the main page is at: `/SpringHyreminder`. However the reporting is at `/SpringHyreminder/ALL/reporting` and ICUs have different URLs: `/SpringHyreminder/ICU6/reporting`. This becomes extremely useful in the Anomaly interface where the individual sensor ID is used as in: `/SpringAnomaly/ICU6/sensor/1000009`. The ability to have the sensor ID number in the URL is helpful for switching pages. A user can click on a link on the Sensor Details page to either the Map or Maintenance pages and the sensor number will be remembered for them.

5.3 Anomaly Interface

The Anomaly system underwent a number of changes. The font size was decreased throughout this system due to the fact that this system is used for maintenance and doesn't need to be read from a distance. The smaller font makes the screen easier to read, allows more text to fit on a screen and reduces the need for scrolling.

The first fix made was on the Sensor Details tab. Observe how in Figure 35 the text goes off the screen and is covered by the copyright text, without a scrollbar to remedy the issue.

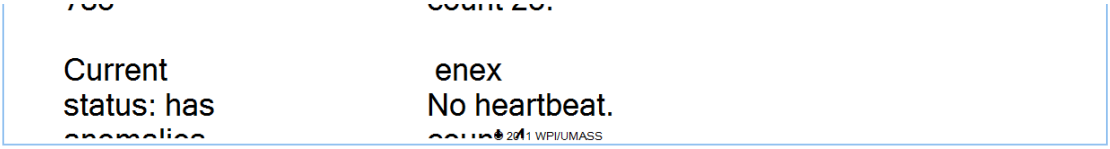


Figure 35: A screenshot of overlapping text in the original Hyreminder system.

With the reengineering of the Hyreminder system, we were able to fix this. In the original system the text was inside of a container with a predefined size. The new system allows the text to take as much room as needed, however the smaller font generally removes this issue (see Figure 36).

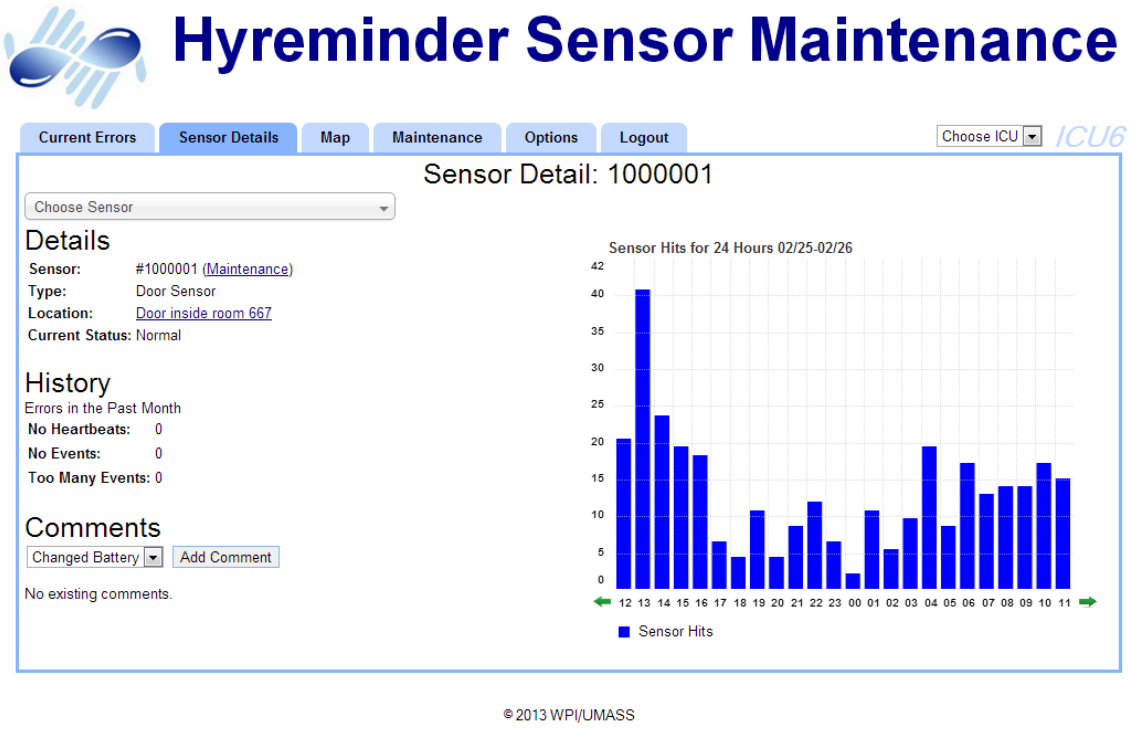
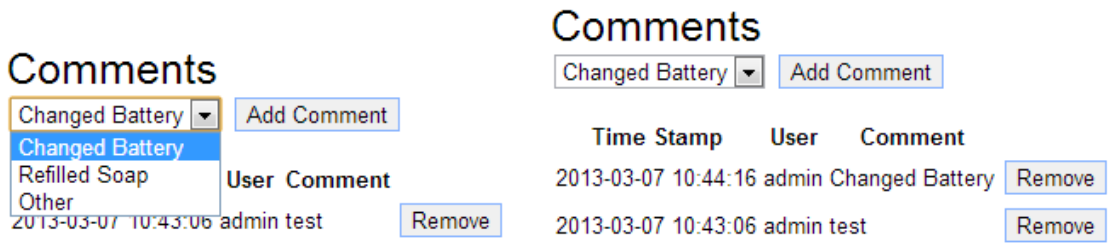


Figure 36: Sensor Details - a screenshot from the revised Anomaly application.

The second change made was to combine the Sensor Details and Sensor History tabs into the Sensor Details tab. It made sense to combine these tabs as they both contain information on one sensor. With the font size decreased, it was easy to fit the chart from the Sensor History onto the Sensor Details tab.

The last change to the Sensor Details tab was the addition of Comments (see Figure 37). The user first chooses an existing comment or types in their own. After adding the comment the page is reloaded and the new comment appears. Comments are sorted with the most recent first. Lastly, comments can be removed in case they had been erroneously added.

Figure 37: Commenting interface on the Sensor Details tab.



(a) Adding a comment

(b) After adding a comment

The Current Errors tab had improvements to the layout of the reported errors to maximize the screen space (see Figure 38). The sensor errors are filtered by which ICU the user is currently in. We added a drop-down for filtering by the types of errors. With the smaller font size, it was harder for the user to tell where the description of one error ended and the next began, so we added a thin box around each error. The box is not of a predefined size, so it will always be big enough to fit the text. Lastly, we utilized white space by allowing the boxes containing errors to fit as many horizontally across the screen as possible, before going to the next line.

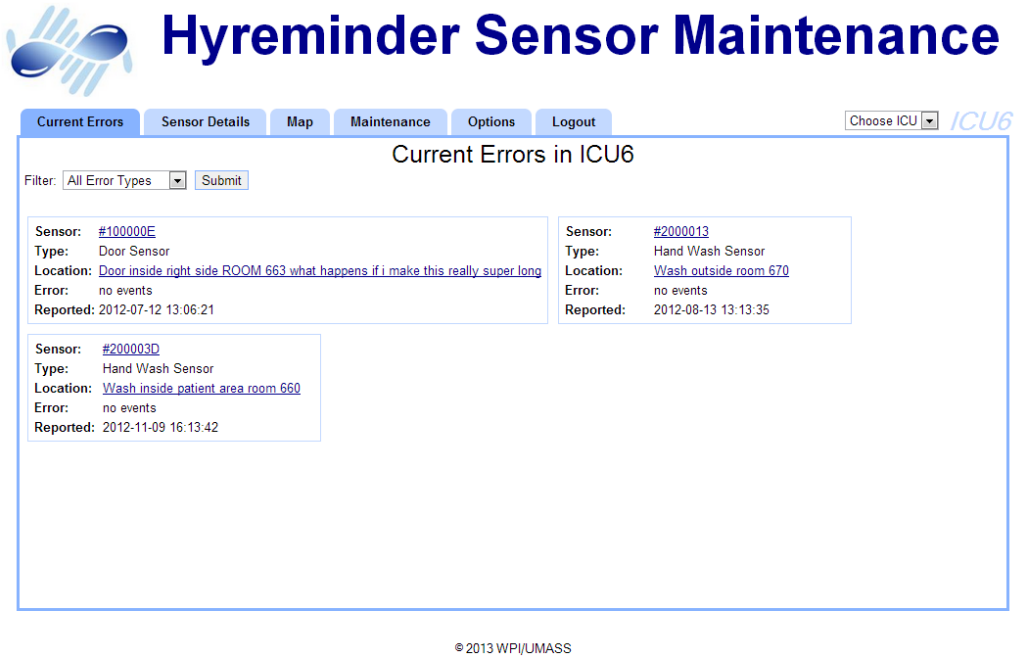


Figure 38: Current Errors - a screenshot from the revised Anomaly application.

To the Options tab we added a form for editing a hospital subspace details, such as one ICU's details (see Figure 39). This includes a display name and the type of subspace: Group Based or Id Based. Additionally there's a file upload dialogue for adding the floor plans for the specific subspace. This floor plan should be an image file, for example .jpg or .png. This floor plan should just have the physical aspects of the subspace: individual rooms, doors and hallways and labels as needed, and doesn't need to have sensors as they may change. Full details for setting up a new subspace can be seen in Appendix A.

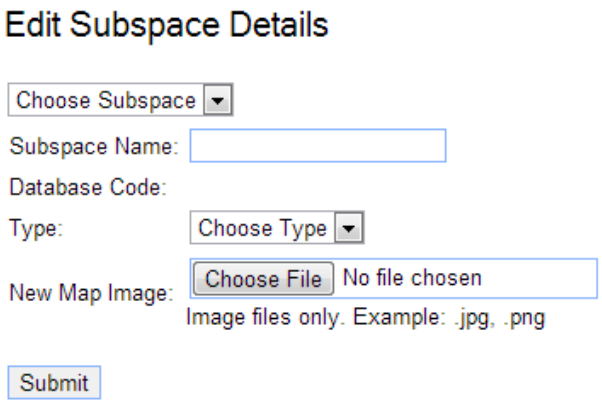


Figure 39: Edit Room Details section on Options page

5.4 Mapping Interface

The first screen used for mapping sensors that we designed is a screen for viewing sensors on a map. There are numerous ways to access this page: clicking on a link from either the Current Errors tab or the Sensor Details tab, or clicking directly on the Map tab. The Map tab (see Figure 40) allows the user to quickly see where a sensor is located and its status. Using a link from either the Current Errors tab or the Sensor Details tab will instantly show a map of the correct ICU with only the selected sensor highlighted. Accessing the Maps screen directly from the maps tab will show the map of the ICU with no sensors. A user can then choose which sensors to show on the map: all sensors in the ICU, all sensors with errors in the ICU or select a subset of sensors. The user can hover over a sensor to see basic details such as its ID, location and status. Additionally they can click on a sensor to go to the corresponding Sensor Details page.

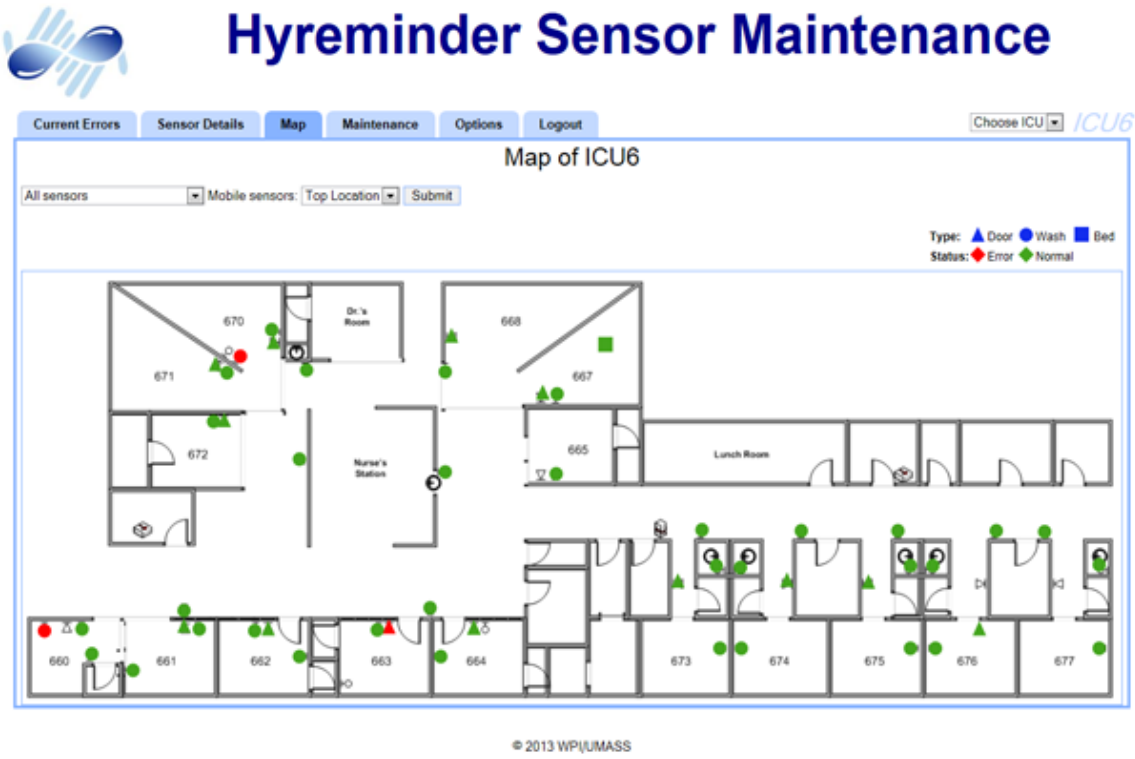


Figure 40: Map- a screenshot from the new screen in Sensor Maintenance.

We considered a couple of ways for the system administrator to set up the locations on a map, including a configuration file and a user interface. Once we added the sensor abstraction by separating the concept of logical and physical sensor identification, we had to decide how a user could add or modify existing sensors. After considering using a configuration file or a user interface, we decided that a user interface would be the most straightforward for the user. A user interface would allow more flexibility in making sure a sensor was in the correct location since we could show a map of the ICU where the sensor was being placed. We determined two main tasks that a user would need to accomplish: creating and editing the logical sensors; and setting up the coordinates of where a sensor will appear on the map. We created a new tab, the Maintenance tab, that is divided into two sub-tabs: Sensor Maintenance and Map Maintenance.

The Sensor Maintenance sub-tab (see Figure 41) is used for creating and editing the logical sensors. After choosing to create a new logical sensor or edit an existing one, the user must fill in the details of the logical sensor. This includes tasks such as adding new physical sensors, activating and deactivating physical sensors, and updating its location when applicable. The steps for this process is numbered out on the screen, starting with choosing a sensor or adding a new

one, then updating the physical sensor information, and lastly specifying whether the sensor moves or is stationary at a location. If the sensor is stationary, fields for entering a location appear (see Figure 42).

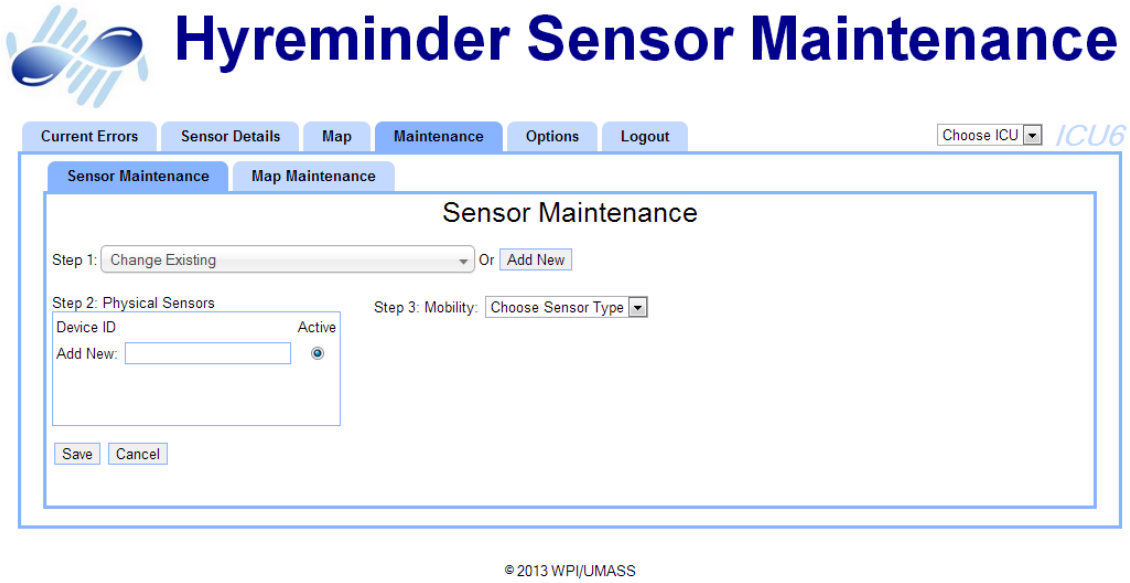


Figure 41: Sensor Maintenance on page load - a screenshot from the new screen in Sensor Maintenance.

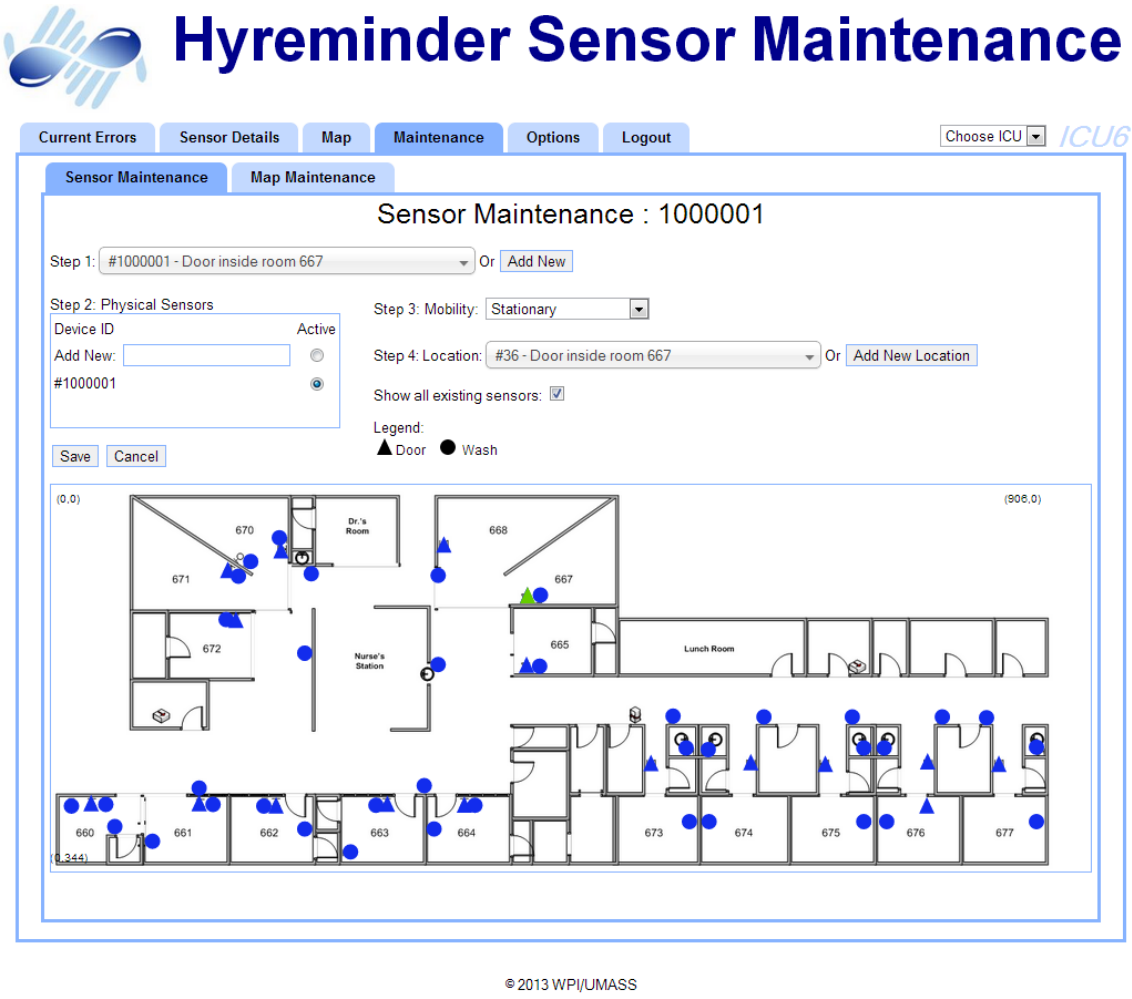


Figure 42: Sensor Maintenance after selecting a Stationary sensor - a screenshot from the new screen in Sensor Maintenance.

To enter a location for the sensor, a user can choose to select an existing location (e.g. Outside Room 663, see Section for more information of logical vs physical sensors) or add a new one to the map. If they pick an existing logical sensor, its location will show up on the map as a green sensor. Alternatively, the user can click "Add New Location" and will be taken to the Map Maintenance page (explained below). Also in the location section of the screen, there is a checkbox for showing all existing sensors. The user can hover over a sensor on the map to see the description of its location.

The Map Maintenance sub-tab (see Figure 43) can be used for creating new sensor locations or editing existing ones. There are several fields with information for a sensor location, including

sensor type (wash, door, or bed), description, and x- and y-coordinates. A user can click and drag a sensor on the map to update the coordinates.

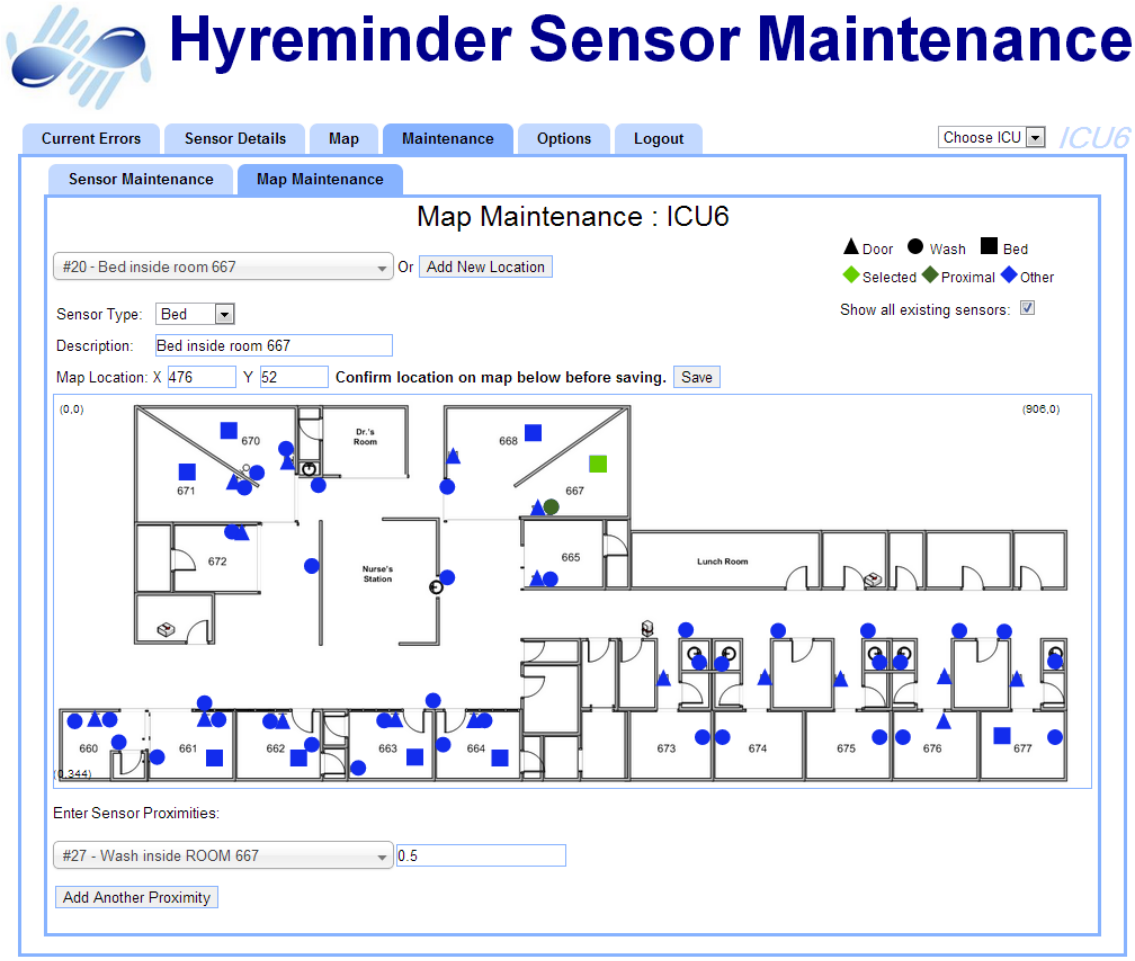


Figure 43: Map Maintenance - a screenshot from the new screen in Sensor Maintenance.

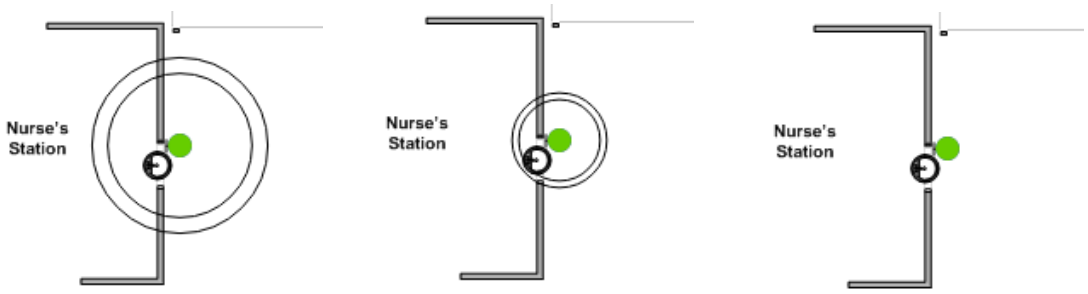
The last thing a user needs to do on this page is enter the proximity from a bed sensor to another sensor. These proximities are used for determining the location of the bed as explained in Section 4.3. The user can add as many proximities as needed by clicking on the "Add Another Proximity" button and filling in the sensor and proximity. After saving the form the proximities go into the proximity table (see Figure 44).

```
mysql> describe proximity;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| locA      | int(11) | NO   | PRI | 0       |       |
| locB      | int(11) | NO   | PRI | 0       |       |
| proximity | float   | NO   |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Figure 44: The proximity table schema. The table is used for storing the proximities between sensors.

When selecting a sensor location from either the drop-down at the top of the page, or from one of the proximity sensor location drop-downs, the selected sensor location will be highlighted on the map. The sensor location is highlighted by two concentric circles zeroing in on the icon on the map (see Figure 45).

Figure 45: Highlighting the selected sensor location.



(a) Start (b) Middle (c) End

The full steps for using these new screens can be found in the User Documentation, Appendix A.

5.5 User Interface Revisions

The interface shown in the previous sections are the final interfaces we created. As we developed we got feedback at our weekly meetings with Professor Elke Rundensteiner and Lei Cao. They made suggestions on wording and consistency with titles. One example is with the Reporting screen. In the original system the user was asked if they wanted the data Aggregated or not (see Figure 46). While this term made sense to us as computer scientists, it is not a clear term for the average user. We ended up renaming the option to Group Values and spelling out exactly what the user will receive as a report (see Figure 47). While this is more verbose and unnecessary after a user understands the intended meaning, we decided this was a clearer way for a new user to understand.

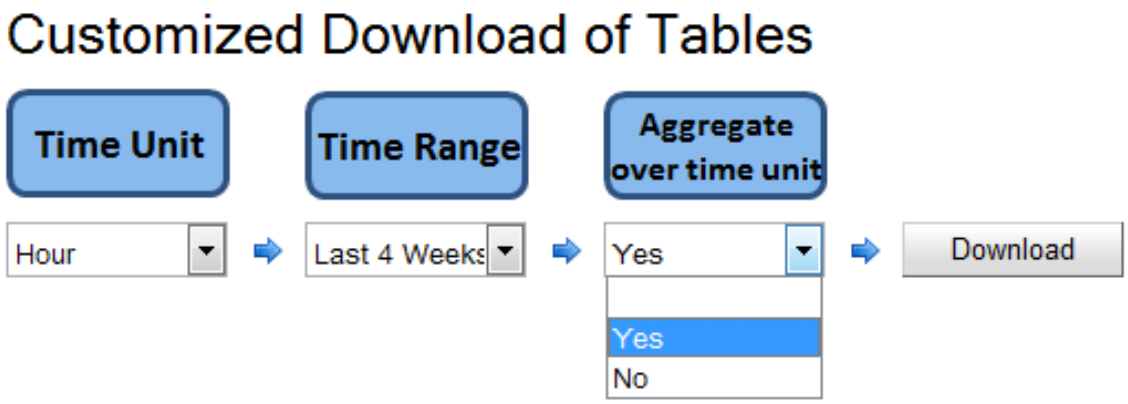


Figure 46: Aggregate option in original Hyreminder system

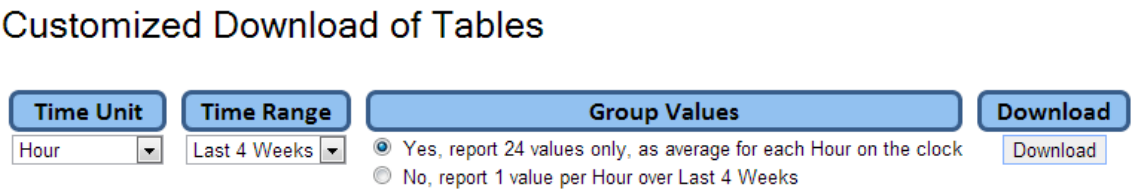


Figure 47: Grouping option in the reengineered Hyreminder system

Professor Elke Rundensteiner also had suggestions for creating a more consistent layout, for example having filters on the same place on all screens, which previously had different spacing around them. She also had us condense white space on all screens, so that more would fit on a screen reducing the need for scrolling on a page.

In addition to feedback from ourselves and our weekly meetings with Professor Elke Rundensteiner, we performed some informal user testing. This involved us asking several peers to look at and use the Maintenance interfaces, and requesting their feedback. We gave them a brief introduction of the Hyreminder system – there are sensors on hand wash units and beds and we are tracking hand hygiene in doctors and nurses. Then we told them that they are the system administrator and are responsible for the maintenance of the sensor system. We asked them to add a new sensor to the system and took notes on what they said and did. At the end, we asked them for any general feedback they hadn't yet said and if they had any ideas on how to improve the usability.

One peer said there was a lot of information on the Sensor Maintenance screen, and wasn't sure where to start. She suggested that we number the steps, and that would have helped guide her through the screen. As she was filling out the form she skipped entering a physical device I (due to being given a poor explanation of the task). When asked about it, she hadn't even realized there

was a section for it, despite having found the save button underneath.

On the Map Maintenance screen, one peer got confused with the legend next to the drop-down for type selection, and tried to click the legend. She suggested making it clearer as to which type of sensor has been selected. She really liked the click and drag on the sensors.

Another peer had a number of comments on the layout, saying that it doesn't feel like a form, and that spacing between elements would make it less of an information explosion. He didn't even realize it was a form right away, because he didn't see the Save button; he suggested that the save button should always be on the left of the screen, not the right, and that it should always be at the bottom of a form. He felt that we should move the physical ID section underneath the map. After explaining why we placed it above the map he said "scrolling is not the end of the world" but agreed that it might be missed.

After hearing their feedback, we were able to implement some of their suggestions. We numbered the steps in Sensor Maintenance to aid the user in working through the slightly unusual form. We moved the physical ID section to the left of the screen and the location section to the right; this allowed the screen to better follow the "F-Shaped pattern."[24]

[24]Nielsen, J. (2006). F-Shaped Pattern For Reading Web Content. http://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/

6 Technologies Used

Over the course of this project, we employed several different technologies to achieve our objectives. Each choice we made regarding the use of a particular technology was thought through and considered before we began using it, to ensure that we were making the best possible decision given our requirements and the capabilities and limitations of the technology in question.

6.1 Accounts and Security

With the reengineering of the Hyreminder system to incorporate the Spring framework comes a change to the way user accounts and security issues are handled. The original Google Web Toolkit Hyreminder system offered no easy way to handle accounts and security so it had to be custom made. This included the addition of code to verify passwords on login and to issue session tokens to maintain login sessions. The Spring framework offered a simpler way to accomplish these tasks, plus more, with the use of Spring Security. [25]

Spring Security is a customizable module for the Spring framework that simplified the handling of logging into the system and preventing unauthorized access to parts of the website. Unlike other Java standards such as Java Authentication and Authorization Service[26] or Java EE Security,[27], Spring Security combines everything into one concise solution for application security.[28] By configuring Spring Security with an XML config file (see Appendix C), Spring Security will automatically verify passwords, create a session cookie, and log in the account without the need for any additional Java code like with the old Google Web Toolkit system. If, in the future, the default methods used by Spring Security are no longer viable, custom Java classes can be made to override the defaults. It is both simple to use to accomplish easy tasks and heavily customizable to handle more complex tasks.

Security should always be a concern for modern day web sites. Associated with each Hyreminder account is an account role that denotes what parts of the website the account should have access to this. Spring Security has another feature that has allowed for specific URLs to require certain account roles in order to be accessed. For example an account with the ICU5 Nurse role can gain access to the data from ICU5, but will not be able to access any other ICUs unless those roles are also assigned to the account. By limiting access based on account roles, those accounts with the correct roles will be able to access their data easily. Yet any unauthorized accounts will not be able

[25] SpringSource. Spring Security Documentation. http://static.springsource.org/spring-security/site/reference.html

[26] Oracle. (2011). JAAS Reference Guide. http://docs.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html

[27] Oracle. (2013). Introduction to Security in the Java EE Platform. http://docs.oracle.com/javaee/6/tutorial/doc/bnbwj.html

[28] Mularien, Peter. (2010). *Spring Security 3: Secure Your Web Applications against Malicious Intruders with This Easy to Follow Practical Guide.* p18. Birmingham UK: Packt Publishing.

to view that data.

For a more detail about the database table structure relating to user accounts, how to modify data in the accounts, and how to secure URLs, see the developer documentation in Appendix B.

6.2 Charting Libraries

Hyreminder makes heavy use of charts in its pages. Previously, the charts were created with a library designed for the Google Web Toolkit called Open Flash Charts for GWT.[29] Due to the move over to the Spring Framework, usage of the Open Flash Charts was no longer an option. Thus, it was necessary to find a framework that would facilitate the creation of charts. After some initial research, four libraries were considered in greater depth: Highcharts[30], JQuery Visualize[31], and JavaScript InfoVis Toolkit (JIT)[32], all client-side javascript libraries, and JFreeChart[33], a server-side java library. Based on discussions with both Doctor Ellison and Professor Rundensteiner on what information they wanted to display in those charts, we put together the following list of design requirements to aid us in deciding which charting library to use:

- Ability to make both bar and line charts.
- Ability to plot a bar and line chart on the same chart.
- Ability to have two different value scales on the y axis, such as a number and a percent.
- Ability for mouseover popups of datapoint values.
- A free and unrestricted licensing option.
- In the absence of any of the above features, the ability to add them if possible.

In addition to these required features, we also considered performance and usability of the libraries as well as the quality of the appearance of the produced charts. Ultimately, none of the four libraries satisfied all of the requirements, so we were forced to make a choice based on the strengths and weaknesses of each library. The pros and cons of each library are detailed below:

HighCharts

- Pros
  - Has all of the desired features except for a free and unrestricted licensing option.

[29] Open Flash Chart for GWT. http://code.google.com/p/ofcgwt/
[30] Highsoft Solutions JS. http://www.highcharts.com/
[31] Jehl, Scott. (2013). JQuery Visualize. https://github.com/filamentgroup/jQuery-Visualize
[32] Belmonte, Nicolas Garcia. (2013). SenchaLabs. JFreeChart. http://philogb.github.com/jit/
[33] Viklund, Andreas. (2012). Object Refinery Limited. JFreeChart. http://www.jfree.org/jfreechart/

  - Is well documented and easy to use.
  - Produces aesthetically pleasing and highly customizable charts.

- Cons
  - The only free licensing option is an educational license which would not work due to the end goal of this project being commercial viability. Highcharts would either need to be replaced or licensing would need to be purchased for its continued use.

JQuery Visualize

- Pros
  - Simple and easy to understand library, in particular, it is easy to modify.
  - Is able to produce line and bar charts.
  - Licensed under the MIT license.[34]
- Cons
  - Does not have the ability to plot bar and line charts on the same chart.
  - Does not have the ability to have two different y axis scales.
  - The only satisfied requirements are the ability to make line and bar charts, as well as the licensing option.

JavaScript InfoVis Toolkit

- Pros
  - Well documented API.
  - Is able to produce line and bar charts.
  - Also produces high quality graph and tree charts if those should be desired.
  - Allows for mouseover hover popups
  - Licensed under the new BSD license.[35]
- Cons
  - Does not have the ability to plot bar and line charts on the same chart.
  - Does not have the ability to have two different y axis scales.

[34] MIT License. http://opensource.org/licenses/MIT
[35] BSD 2 License. http://opensource.org/licenses/BSD-2-Clause

  - Very limited customization of charts.
  - While the API is well documented, it is hard to use.
  - The library is large and has many features we would not use, it would be particularly difficult to modify.

JFreeChart

- Pros
  - Satisfies all requirements except the ability for mouseover hover popups.
  - Well documented API.
- Cons
  - It is a serverside chart generation library, this means that the server instead of the client would generate the charts causing a heavier load on the server.
  - The web based implementation of JFreeChart only allows for generated images to be provided to the client. Not only does this take several times more bandwidth than the other options, it makes it almost impossible to add mouseover hover to the library itself. The mouseover hover would need to be a completely separate feature on top of it.
  - Since the framework is server-side, it would take quite a bit of configuration to properly integrate with the Spring server model.
  - The library is large and has many features we would not use, it would be particularly difficult to modify.

Of these four libraries, the option of using HighCharts was discarded rather quickly due to the licensing option. We decided that it would be better to use one of the free options even if they would take more work. JavaScript InfoVis Toolkit was discarded mostly due to the rigidity and complexity of the framework. It was incredibly difficult to figure out how to customize charts to look the way we wanted them too, and any changes to the framework would not be trivial due to the complexity. JavaScript InfoVis Toolkit is good assuming you want to produce charts that are very similar to the charts that it produces. However, the graph and tree charts for JavaScript InfoVis Toolkit seemed to be easier to work with than the bar and line charts. Due to it's quality, it may be a good option if a future project wishes to add charts of this type. The last option to be discarded was JFreeCharts. While this would be an ideal solution for a desktop application, it is not as good an option for a web based applications due to the above mentioned limitations of server-side related challenges. However, it did not meet our needs well. Since it would be a significant investment to add hover effects to this library, we decided against using it.

Having eliminated three of the four options, this left us with JQuery Visualize as the remaining charting solution available to us. While, of the four options, it has the fewest of the requirements that we wanted, it is also the easiest library to extend due to its straightforward API and simplicity. In order to deal with this key limitation, we wrote a significant extension to the library to add functionality that was required for our project. In particular, we added support for plotting both bar and line charts on the same graph, functionality for displaying two different scales on the y axis, and functionality for mouseover hover effects. Since this is an open source library released under the MIT license, these extensions were released back to the community as a fork of the original library[36]. A screenshot of the charts used for the detailed statistics page, created using this charting library, can be seen in Figure 48. For full information about the details of this extension, please refer to Appendix **??**.

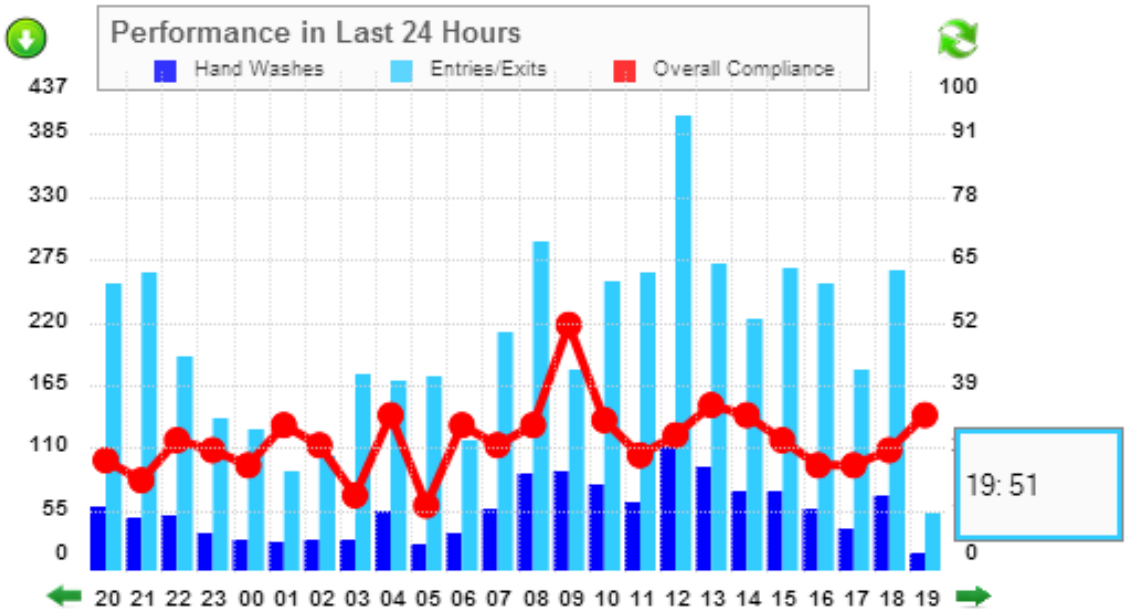


Figure 48: The modified JQuery Visualize[37]charts displaying the Hand Hygiene Events / Room Entries-Exists for ICU6 in the last 24 hours. See Figure 4 for a comparable chart.

6.3 Mapping Labrary

With the addition of the sensor tracking, the Hyreminder web application needed a way to visually display the location of sensors to the user. We wanted to host just one map image per ICU on

[36]The library can be found on github here: https://github.com/newdog/jQuery-Visualize
[37]Ibid.

the server, but we also had to be able to display different sets of sensors depending on the page and options selected by the user. To satisfy these requirements, we looked for a client-side drawing library, not a server side one for the same reasons described in Section 6.2. As with charts, we needed an open license.

We found the JavaScript library Raphaël.[38] After looking at demos using the tool and reading some of the documentation, we added it to Hyreminder. Raphaël allows you to add images to the canvas, draw shapes, and add text. Elements added to the canvas can be animated, and have mouseover and click effects. All of these aspects were needed for displaying a map that is dynamic to the user's needs. Additionally, the Raphaël library is released under the MIT license.[39]

When we use the library on a page, we first add the image of the floor plan, stored in the database, to be the background of the map. Then with information about sensors, also stored in the database, we can draw them on top of the floor plan. This allows us to have many dynamic maps, with various sensors shown with a different status, and yet we only need to host one image file on the server. Because the map is generated with JavaScript, the user can show and hide sensors without needing to go back to the server for a new map every time.

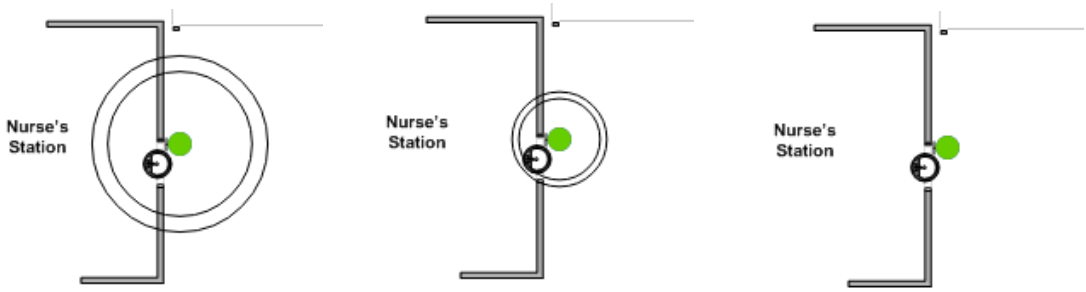
In addition to adding the floor plan image and drawing sensors, we took advantage of animations to highlight sensors with two circles zeroing in on the sensor and to click and drag a sensor when editing the location. We used mouseover effects to show more details about the sensor, and click effects to direct the user's browser to the sensor's page. See Figures 49, 50 and 51 for example of the Raphaël library used in the Hyreminder application.

Figure 49: Example of Raphaël. This is a map with many sensors (blue shapes) and only one selected (green).

[38] Baranovskiy, Dmitry. (2012). Raphaël. http://raphaeljs.com/
[39] MIT License. http://opensource.org/licenses/MIT

Figure 50: Example of Raphaël. This is a map with many sensors, with two different statuses: normal (green) and error (red).

Figure 51: Example of Raphaël's animation capabilities, used to highlight a sensor that has been selected.



(a) Start (b) Middle (c) End

6.4 Additional UI Libraries

In addition to the two main graphical libraries that we used, JQuery Visualize and Raphaël, we used a number of other UI libraries to achieve our desired user interface. These libraries include jQuery Datepicker for picking custom date ranges, jQuery Dialog for user friendly dialog boxes, and Chosen for searchable drop-down menus.

6.4.1 jQuery Datepicker

The Report tabs allow for custom date ranges to be picked by the user (see Figure 52). To aid the user in entering valid dates, we used the jQuery Datepicker.[40] This datepicker is an improvement on the date picker in the old system for several reasons. First, the user can use drop-downs to pick month and year instead of needing to scroll back months. The datepicker shows three months at a time. Lastly, the order of dates is enforced, so that the start date will always be before the end date.

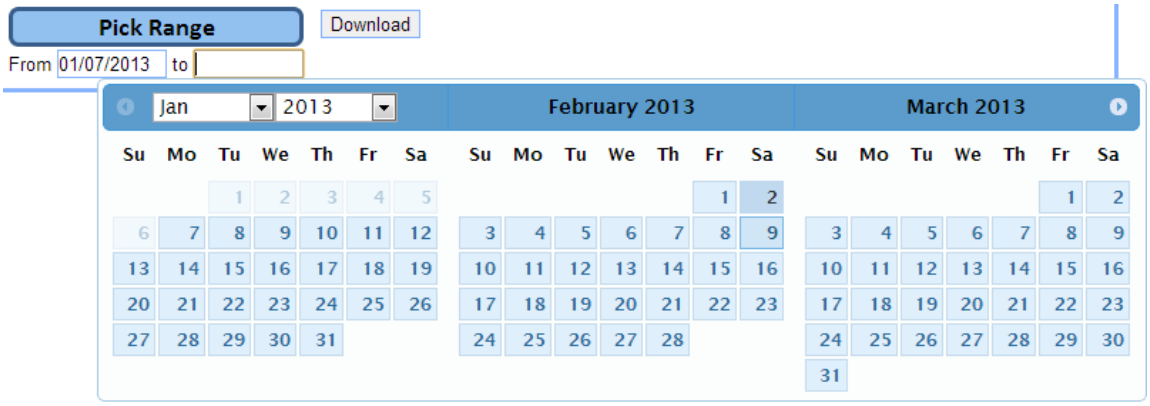


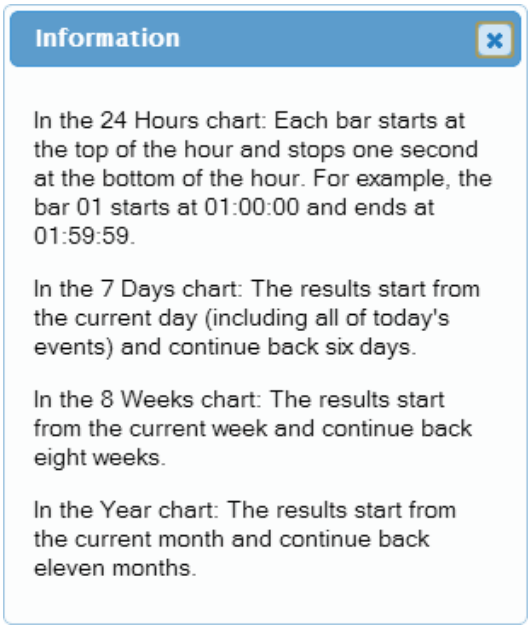
Figure 52: jQuery Datepicker[41] with Jan 7 selected as the beginning, so no date before then can be selected for the end date.

[40] jQuery Foundation, The. (2013). jQuery UI Datepicker. http://jqueryui.com/datepicker/
[41] Ibid.

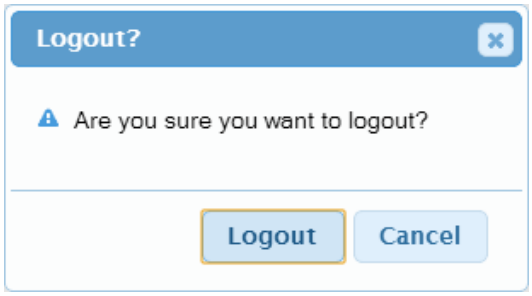
6.4.2 jQuery Dialog

One user feedback that we received was that an information button created a dialog box that made a warning system sound, making her think something was wrong with the webpage. To improve the user's experience, we added the jQuery Dialog[42] library. This enables us to create user friendly dialog boxes, that don't create a negative sound (see Figure 53 for examples).

Figure 53: Two different example of jQuery Dialog boxes used in the Hyreminder web applications.



(a) An information dialog



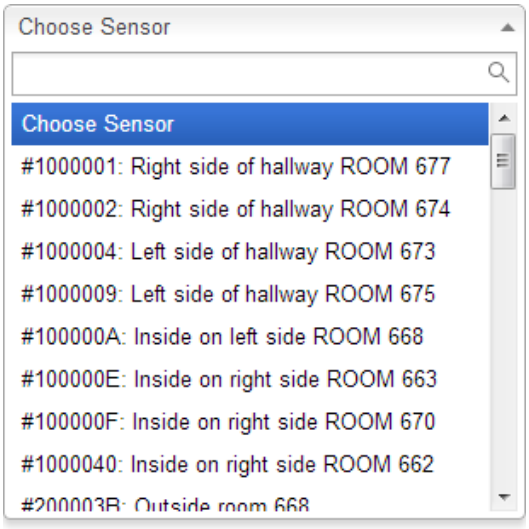
(b) A logout prompt

[42] jQuery Foundation, The. (2013). jQuery UI Dialog. http://jqueryui.com/dialog/

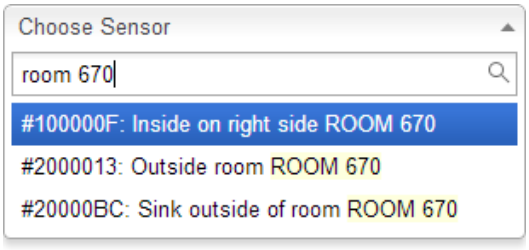
6.4.3 Chosen

One difficulty with using drop-downs for a long lists of items, such as sensor IDs, badge IDs and locations, is that it is difficult for the user to find a particular item. To help with this problem, we used Chosen,[43] a JavaScript library that allows the user to search the drop-down. In Figure 54a a user has clicked on a long drop-down list of sensors. As they begin typing, the list is shorted to only entries matching what the user has typed, as seen in Figure 54b.

Figure 54: An example of the searchable drop-downs in the Hyreminder application.



(a) Long drop down list of sensors.



(b) A filtered list of sensors based on user's search text.

[43] Harvest. (2011). Chosen. http://harvesthq.github.com/chosen/

7 Testing

As mentioned in our analysis of the Hyreminder system (see Section 3.1), the state of testing in the previous code base was nonexistent. One of our goals was to provide multiple forms of testing that can sufficiently test the functions and features of the new Hyreminder web application. These tests not only help to verify that the current code works as intended and is as efficient as possible, but they also verify that future development will not inadvertently break any functionality. Our tests can be divided into four main categories: standard JUnit[44] unit tests, performance tests by both stopwatch and the not-so-standard Selenium WebDriver[45] tests, correctness testing to verify that our system performs as desired, and a series of stress tests that push the system to its limit. For documentation on how to run the tests, see Appendix E.1.

JUnit is an open source framework designed for the writing and running of tests in Java. The unit tests for Hyreminder are designed to test single units of code such as a function or a class and verify that that code is functioning as desired. The unit tests are the first layer of defense in preventing faulty code from being implemented. After making a change to the code, unit tests are run to verify that nothing broke.

Selenium is a framework that supports web browser automation. The primary benefit of Selenium is that tests can be written to actively navigate a website in multiple web browsers. Selenium tests offer a huge advantage to verifying correct functionality since it emulates the exact tasks that the average user would be doing, and thus helps us to identify and fix bugs before a user experiences them. The tests that we have implemented include verifying page loads and verifying report downloads for each ICU. To verify that all pages of the website load without error, our Selenium test will open an instance of Internet Explorer, Chrome, and Firefox and proceed to navigate to every part of the website. Once it opens a page the test checks that it receives a status code of 200, meaning that the page loaded correctly, and then moves on to the next page. This test ensures that the system is compatible with the three major browsers. The second set of Selenium tests open the same three browsers and download every combination of time unit, time range, and aggregation reports. This makes sure the download links work and acts as a measurement for how long each download takes. Not only did we test the download speeds of our new reengineered system, but also the GWT system as well. The test *TestGWTReporting* measured the time it took to download all iterations of reports from the GWT system to use as a comparison. These download tests were used to automate the process of recording times found in the Efficiency Improvements Section 7.2. A full list of Selenium tests include the following:

- Page Load
  - SeleniumPageLoadTests

[44] JUnit. http://junit.sourceforge.net/
[45] Selenium Browser Automation. http://docs.seleniumhq.org/

- Report Downloads
  - SeleniumICU5DownloadTests
  - SeleniumICU6DownloadTests
  - SeleniumICU7DownloadTests
  - TestGWTReporting

The correctness testing was done to ensure that our reengineered system output correct data. Since our reengineered system was supposed to perform the same tasks as the original, we ran several correctness tests on our database queries and reporting functions. These reports were generated on both the old system and on ours, and the data was checked to ensure consistent results. Any errors we found were investigated and fixed, until we had tested and confirmed that our system was as accurate as the original.

7.1 Stress Testing

In addition to the testing performed above, we also conducted a series of stress tests on our system. This has two purposes. The first is that it allows us to identify portions of the system that are bottlenecks and thus good candidates for performance improvements. Secondly, this allows us to discover the limits of the system in terms of variables such as hospital size or employee numbers as well as time. In other words, this helps us to determine at what point these variables become too high for the system to perform within a reasonable amount of time. For example, the system may take longer than is acceptable to generate a report if there are more than some threshold of beds in use in a given ICU. The Hyreminder system has the potential to be deployed at a hospital of any size. It is crucial that our system is scalable to avoid any bottlenecks or slow loading speeds when processing large amounts of data.

7.1.1 Framework

For stress testing we went in with the goal of not only producing a set of stress tests, but also providing an easy way for stress tests to be initialized and used in the future. To achieve this we developed a simple stress test framework that would allow us and future developers to create and run their own stress tests with minimal effort. The framework consists of individual test units, which are written by the developer, and a runner that runs them as per an xml configuration file. Inside of the xml file are configurations for setting the number of times the test is repeated, a list of all the parameters as well as the starting value and step value of each, and the number of times to increment the parameters. The runner creates a report with the results of the test in a number of possible formats such as text, xml or html format. As of the end of our project the only currently supported format was html. More information about the framework can be found in Appendix E.2.

7.1.2 Supported Tests

When choosing which parts of our system to test, we focused on both database queries and service methods that are either used frequently or that require the most time to execute. It is at these two points where a bottleneck would be most problematic. We selected tests that would cover the following categories: Chart generation, database operations, reporting, and sensor details. Each of these areas represent potential weak spots in the system. Significant load times here would limit the usability of the entire system.

Generating charts is one of the most time consuming portions of our system. Thus is only natural to stress it to see how it performs with a large number of events in the database. In this series of tests, we stress the service methods that are called to generate Group Based, ID Based, and Multi-ICU charts. The test will scale the number of tuples stored in a local database and simulate the time it takes to generate a single chart of the given type. Thus it effectively judges the performance of pages such as the Detailed Statistics page which generates four of these charts.

The database operation stress tests target frequently used database queries that do not take much time to run. Rather than tests the service methods, these tests focus on testing the raw queries run on the database. The queries selected are those that are run very frequently and often run multiple times by other parts of the system. The most important of these queries is the one that retrieves a count of all hand wash events, enter/exit events, and patient contact events and calculates the compliance ratio. Because of the importance of this query, we've tested it with two varying values. *Overall Compliance (Incrementing Rows)* stresses how the query performs when the number of event rows in the data is incremented. The second version of this test, *Overall Compliance (Incrementing Rooms)*, will measure how the query performs when it retrieves the compliance ratio across multiple ICUs. It is important to separate out these two variables so that if there is a problem, the true source can be found.

The reporting stress tests call the service method used to generate an excel spreadsheet while incrementing the number of events in the database. The execution of the test is similar to the chart stress tests, we keep adding more events to the database and see how long it takes for the data to be gathered and inserted into an excel document.

In order to cover every part of the system, we also dive into the Sensor Maintenance component with the final test. This sensor stress test adds additional sensors with a static amount of sensor history and sensor comments and tests how long it takes to gather the details of a single sensor. This type of information would be requested when visiting the Sensor Details page (see Figure 36).

A full list of all stress tests include:

- Charts
  - Group Based
  - ID Based

  - Multi-ICU
- DB Operations
  - Overall Compliance (Incrementing Rows)
  - Overall Compliance (Incrementing Rooms)
  - MICU Compliance (Incrementing Rows)
  - Unit Comparison
  - Worker Comparison
- Reporting
  - Aggregate
  - Non-aggregate
- Sensors
  - Sensor Details

7.1.3 Results and Analysis

The stress tests listed below were ran ten times at the given limit and the average execution time was recorded. The computer used to run the tests had the following specifications:

- Operating System - Windows 7 x64 Home Premium
- CPU - AMD Phenom II x4 3.4Ghz
- RAM - 8 Gigs
- Application Environment - Spring Tool Suite 3.1.0
- Java Version - 1.7.0_15

Table 3: Stress Test Results

Stress Test	50,000 Limit	500,000 Limit	1,000,000 Limit	5,000,000 Limit
Overall Compliance (Incrementing Rows)	0.678 sec	6.268 sec	16.082 sec	79.698 sec
Overall Compliance (Incrementing Rooms)	FAILED @ 1.500 ICUs	N/A	N/A	N/A
MICU Compliance	0.314 sec	3.469 sec	10.102 sec	57.680 sec
Group Based Charts	0.669 sec	3.545 sec	10.280 sec	52.084 sec
ID Based Charts	0.185 sec	2.000 sec	3.739 sec	19.257 sec
MICU Charts	1.442 sec	16.448 sec	43.890 sec	210.058 ms
Worker Comparison	196.059 sec	N/A	N/A	N/A
Unit Comparison	0.357 sec	3.929 sec	8.192 sec	39.028 sec
MakeExcel Day Aggregate	0.345 sec	3.591 sec	10.916 sec	53.316 sec
MakeExcel Day NonAggregate	0.346 sec	3.587 sec	10.516 sec	51.953 sec

Table 3 explains the amount of time it took to execute each test at a given maximum limit. For eight out of ten[46] tests, the limit represents the number of tuples inserted into each of the following tables: washtablelive, enexlive, contact1. The maximum limits were chosen in order to cover the low, average, and high ranges. To put it into perspective, over the past year the two ICUs the system is deployed in at UMass Memorial had 2,087,988 enter/exit sensor hits, 953,588 hand wash sensor hits, and 0 contact sensor hits[47] for a total of 3 million rows across the three tables. Our 1 million limit tests will add 1 million rows to each of the three tables resulting in a similar number of rows as UMass. To test our system on an even larger amount of data, we chose an even higher limit of 5 million rows of data in each of the three tables. The remaining two tests that were not mentioned, Overall Compliance (Incrementing number of rooms) and Worker Comparison, have incrementing number of ICUs and workers respectively rather than sensor hits.

With the exception of one issue that is discussed later, the system performed very well with a large amount of data. As we increased the maximum limit, the execution time increased at a linear rate. For example, running the Overall Compliance (Incrementing Rows) test at the 50,000 limit compared to the 500,000 limit resulted in a 9.24x increase in execution time, which is close to the 10x limit increase. This proves that our system backend is capable of handling a growing amount of data. When dealing with a realistic[48] amount of data, we found no bottlenecks in the areas tested that could halt the system or result in errors.

[46] Overall Compliance (Incrementing Rows), MICU Compliance, Group Based Charts, ID Based Charts, MICU Charts, Unit Comparison, MakeExcel Day Aggregate, and MakeExcel Day NonAggregate

[47] There were 0 hits in the past year because the contact sensor is used in ID based which is not implemented at the hospital yet.

[48] Thousands of ICUs or tens of thousands of workers is considered unrealistic for a single hospital.

When attempting to run the overall compliance query with a list of 1500 ICUs, we received a Data Integrity Violation Exception. The stored procedure being tested takes in a list of ICU codes as one of its parameters. This exception was caused because the comma separated values being passed in was too long resulting in a MySQL Data Truncation error. This limits the number of ICUs in our system to under 1,500. Such a high number of ICUs is very improbable and it is unlikely that a single hospital will ever reach that number.

Another data point that stands out is the large execution time of Worker Comparison at the low limit. The reason behind this is that Worker Comparison is incrementing the number of workers in the system.Looking at Table 3, the low limit added 50,000 workers to the database. Such a high number of workers is not likely to occur in a hospital so it is unlikely that it will ever see execution times as high as the low limit. Setting the limit to 1,000 workers showed a more reasonable execution time of 2.426 sec. Table 4 shows the run speeds of the two problematic tests at a much lower limit.

Table 4: Stress Test Results with Lower Limits

Stress Test	**100 Limit**	**500 Limit**	**1,000 Limit**
Overall Compliance (Incrementing Rooms)	0.87 sec	0.288 sec	1.042 sec
Worker Comparison	0.49 sec	0.625 sec	2.426 sec

Due to time constraints, there was no data collected from the Sensor Details stress test. The task of running and monitoring this test has been moved to the Future Works section (see Section 9).

7.2 Efficiency Improvements

One of the goals of this project is to enhance performance by reducing the time cost caused by chart creation and report downloading. Generating charts and reports involves querying data from the database, performing calculations on the data, and putting the data into a user friendly format.

We used three different computers to measure the performance of the Hyreminder application and averaged the times. In the first trial, we used one computer to run all of the possible tests, timing them with a stopwatch. In the second trial, we used another computer to run the tests using Selenium. After the first two trials, we reviewed the times we got for variability. The tests with high variability were marked for a third trial run, in order to make the average as correct as possible. The third trial was run on a third computer, again using a stopwatch. After all three trials were completed, we reviewed the times and removed outliers. For example, one test – the Spring 12 Month Button in Group Based Reporting – had 14 times, thirteen of which were between 9.0 and 10.3 seconds. The fourteenth time was 15.2 seconds, a clear outlier.

From the averages on both the old application and the reengineered application, we calculated one value to represent how well our system did. We used the percent formula below. A negative

number indicates a performance gain (faster), while a positive number indicates a performance loss (slower).

$$Percent = \frac{New - Original}{Original} * 100$$

7.2.1 Chart Creation Performance

For the chart creation performance testing, we used the three computer procedure described above. First we tested the charts in the main Hand Hygiene system. This includes:

- the four charts on the Detailed Statistics page, one test per type of ICU (Group Based, ID Based and Multi-ICU);
- comparing all ICUs on the Units Comparison page, one test per time frame;
- comparing workers on the HCW Comparison page, one test per grouping of workers.

See our results summarized in Table 5.

Table 5: Hand Hygiene Charting Performance Comparison

Chart Type	GWT System	Spring System	Time Difference	Percent Change
Detailed Statistics (IC6)	41.71 sec	24.78 sec	–16.92 sec	–40.58%
Detailed Statistics (IC5)[1]	40.07 sec	2.51 sec	–37.57 sec	–93.74%
Detailed Statistics (ALL)	199.38 sec	40.28 sec	–159.11 sec	–79.80%
Units Comparison, 24 Hours	31.51 sec	4.82 sec	–26.69 sec	–84.70%
Units Comparison, 7 Days	30.18 sec	4.87 sec	–25.31 sec	–83.86%
Units Comparison, 8 Weeks	12.06 sec	5.94 sec	–6.12 sec	–50.74%
Units Comparison, 12 Months	15.83 sec	15.87 sec	0.04 sec	0.25%
HCW Comparison, All Workers, 12 Months	4.21 sec	1.47 sec	–2.74 sec	–65.07%
HCW Comparison, MDs, 12 Months	2.69 sec	1.18 sec	–1.51 sec	–56.24%
HCW Comparison, By Shift, 12 Months	9.43 sec	1.04 sec	–8.39 sec	–88.97%
HCW Comparison, Individuals: 3, 12 Months	N/A[2]	0.93 sec	N/A	N/A

[1] These charts have no current data. Thus this comparison shows how long the trip to the database and any formatting performed takes to complete.
[2] This chart rarely completed on the original system, and when testing the trial was stopped after 5 minutes of waiting. When the chart did complete, the x-axis showed all workers instead of the one selected.

All but one of the charts has improved performance, and most have performance gains of more than 50%. The “Charting Revised Time Taken(%)” Chart (Figure 55) shows the performance gains in total time taken for each reporting query as a percent of the original query time. In this chart,

a bar at 0% means there was no change in the report's performance, while a bar below 0% means the report saw a performance gain.

The "Charting Revised Time Taken (sec)" Chart (Figure 56) shows the absolute difference in seconds for all but one of the chart types. The missing type, Detailed Statistics (All), saw a 159 second decrease in actual time, which would have made the chart very skewed and difficult to adequately show the other tests' improvements.

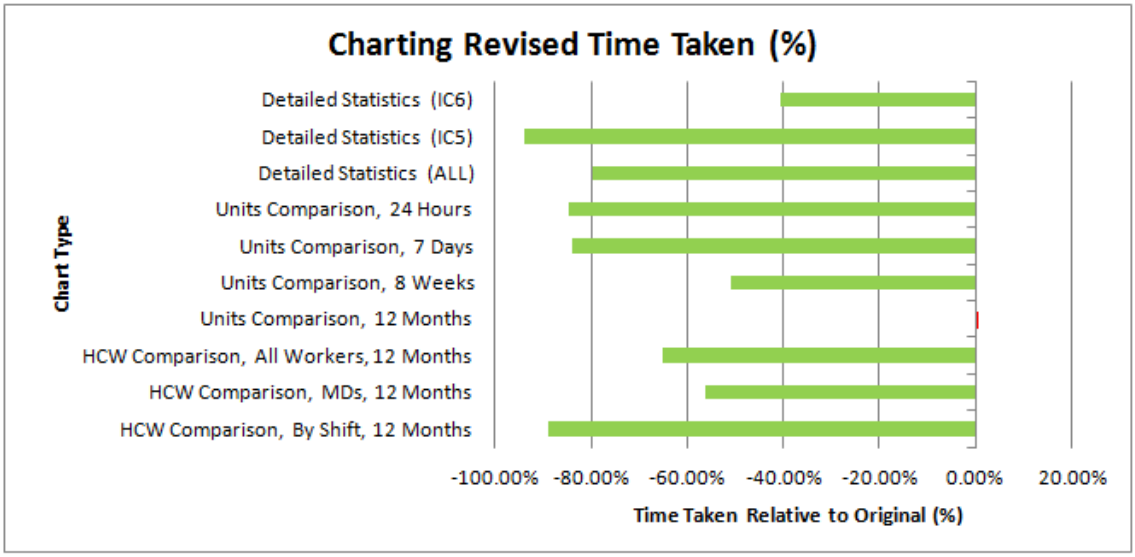


Figure 55: Charting Revised Time Taken (%) The y-axis shows the charts tested. The x-axis shows the percent difference in time taken from the original. Green bars to the left of 0% had improved performance. Red bars to the right of 0% had reduced performance.

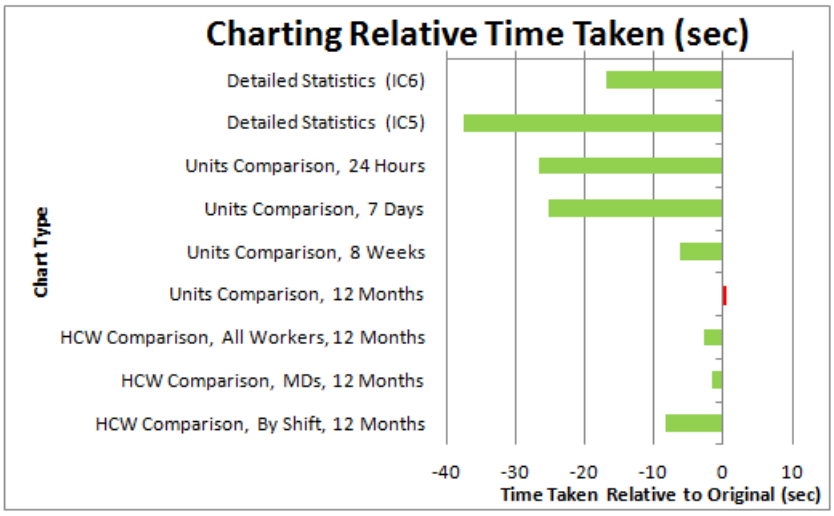


Figure 56: Charting Revised Time Taken (sec) The y-axis shows the charts tested. The x-axis shows the difference in time taken from the original. Green bars to the left of 0 sec had improved performance. Red bars to the right of 0 sec had reduced performance.

The charting performance improvements are mainly due to the better charting library, JQuery Visualize (see Section 6.2). This charting library allows the client to do processing of the charts, instead of the server creating a .swf file. Letting the client handle the processing reduces the load on the server and is much faster than generating a .swf file. This is especially evident in the case of the charts relating to ICU5. Since there is no data for ICU5 because the physical system for ID based reporting is not yet implemented, the ICU5 charts give a fairly good representation of the overhead required to run the chart queries. The old system took 40 seconds on average to query the database, determine that no data existed, build the .swf that shows an empty chart, and finally send that file to the client. Our system eliminated almost 94% of the overhead, displaying an empty chart in 2.5 seconds.

While the times we achieved were much better than the original system, we weren't pleased with the 25 to 40 seconds it took to load the Detailed Statistics page. So we implemented server side caching on the charts that change infrequently and/or are commonly accessed: 24 Hours, 7 Days, 8 Weeks and 12 Months. This greatly improved our times as seen in Table 6. The testing procedure differed slightly than the process used to compare chart performance between old and new systems. For this test, we ran multiple trials using the detailed statistics for the past 24 hours. By moving into the past on that chart, new data was generated that had not been cached, and these times were recorded. To get cached times, the time movement was reversed, so the system retrieved the data it had already generated and was still in the server's cache.

Table 6: Hand Hygiene Cached Charting Performance Comparison

Chart Type	Date Generation (Uncached)	Data Retrieval (Cached)	Time Difference	Percent Change
ICU6 - Group Based	5.28 sec	0.82 sec	–4.45 sec	–84.40%
ICU5 - ID Based	1.00 sec	0.79 sec	–0.213 sec	–21.28%
ALL - Multi-ICU	9.20 sec	0.79 sec	–8.41 sec	–91.38%

The caching system implemented drastically cut down on chart load times. For the two ICU types that had data, ICU6 and ICUALL, the caching system resulted in the load times being reduced to an almost unnoticeable one second, compared to the original time of several seconds. ICU5 showed less improvement because there was no data to retrieve from the database, but the caching still cut down on the load time by eliminating the need to go to the database. The caching system stores charts for a period of time depending on chart type and cache size. Chart expiration is set as part of the server configuration, and can be different for each type of chart. For example, the Last 24 Hours chart expires very quickly, usually after one minute, while the Last 12 Months chart expires much less frequently, after approximately four days. Due to the nature of the limited space for the cache, the charts may be expired more frequently, as in the event of a full cache the least recently used charts will be removed to make more room.

Comparing the chart creation performance in Anomaly is a little more challenging, as we combined tabs and added more functionality (see Section 4.2 and 4.2 and Section 5.3). Any comparison we do is not a true comparison due to these differences in supported functionality. Additionally, any test we would have performed would have contained so little data, it would have been difficult to see any change in time.

7.2.2 Report Creation Performance

For the report creation performance testing, we used the three measurement methodologies procedure described in Section 7.2. The tests are all run on ICU 6, as it is a Group Based ICU and the original system only supported reports in a Group Based ICU. We tested all four of the preset buttons under “Download Detailed Statistics View Charts” and all combinations of the drop-downs under “Customized Download of Tables” with the exception of custom date ranges. See our results summarized in Table 7.

Table 7: Hand Hygiene Reporting Performance Comparison

Report Type	GWT System	Spring System	Time Difference	Percent Change
24 Hours Button	4.64 sec	4.19 sec	–0.45 sec	–9.76%
7 Days Button	6.33 sec	4.30 sec	–2.03 sec	–32.04%
8 Weeks Button	26.62 sec	4.78 sec	–21.8 sec	–82.03%
12 Months Button	8.79 sec	9.59 sec	0.80 sec	9.05%
Month, 6 Months, Yes	N/A[1]	N/A	N/A	N/A
Month, 6 Months, No	5.43 sec	6.80 sec	1.37 sec	25.17%
Month, 12 Months, Yes	N/A[1]	N/A	N/A	N/A
Month, 12 Months, No	8.43 sec	9.66 sec	1.23 sec	14.59%
Week, 4 Weeks, Yes	N/A[1]	N/A	N/A	N/A
Week, 4 Weeks, No	17.51 sec	4.91 sec	–12.59 sec	–71.94%
Week, 8 Weeks, Yes	N/A[1]	N/A	N/A	N/A
Week, 8 Weeks, No	32.96 sec	4.66 sec	–28.30 sec	–85.85%
Day, 4 Weeks, Yes	4.28 sec	4.36 sec	0.08 sec	1.83%
Day, 4 Weeks, No	4.36 sec	4.34 sec	–0.01 sec	–0.29%
Day, 8 Weeks, Yes	4.50 sec	5.31 sec	0.80 sec	17.82%
Day, 8 Weeks, No	4.67 sec	5.35 sec	0.68 sec	14.61%
Hour, 4 Weeks, Yes	109.85 sec	5.33 sec	–104.53 sec	–95.15%
Hour, 4 Weeks, No	4.83 sec	4.53 sec	–0.30 sec	–6.30%
Hour, 8 Weeks, Yes	103.04 sec	4.65 sec	–98.38 sec	–95.48%
Hour, 8 Weeks, No	4.64 sec	4.89 sec	0.25 sec	5.30%
8–Hour Shift, 4 Weeks, Yes	14.17 sec	8.99 sec	–5.18 sec	–36.58%
8–Hour Shift, 4 Weeks, No	N/A[2]	9.15 sec	N/A	N/A
8–Hour Shift, 8 Weeks, Yes	18.46 sec	9.33 sec	–9.13 sec	–49.46%
8–Hour Shift, 8 Weeks, No	N/A[2]	10.80 sec	N/A	N/A
12–Hour Shift, 4 Weeks, Yes	16.32 sec	11.05 sec	–5.27 sec	–32.29%
12–Hour Shift, 4 Weeks, No	N/A[2]	10.84 sec	N/A	N/A
12–Hour Shift, 8 Weeks, Yes	16.21 sec	9.62 sec	–6.59 sec	–40.65%
12–Hour Shift, 8 Weeks, No	N/A[2]	11.30 sec	N/A	N/A

[1] These reports were removed from the new system due to changed design requirements.
[2] These reports rarely completed on the original system, and when testing these the trial was stopped after 5 minutes of waiting if no report was generated.

In our reengineered system, some of the simpler queries had increased time from the original system. These increases are due to the Spring Framework, which adds additional layers for the report request to go through before it is delivered to the user. However, the benefit of these layers result in massive gains in the queries that were improved, such as the "Hour, 8 Weeks, Yes" report saw a 95% decrease in the time taken, as the average time dropped from 103 to 4.6 seconds.

The major reason for the performance gains was the query optimization we performed. Some of the reports generated a query for each unit of time. Other queries were doing data manipulation

on the server rather than letting the database do the necessary joining, especially in the reports that aggregated the data (see Section **??** for more information). As an example, in generating the report, the original system sent several queries to the database. For example, a report on the past 8 weeks would result in 8 queries being sent to the database. Our system consolidated the queries into one larger query, which eliminated the need for multiple trips to the database and thus greatly improved performance.

The "Reporting Revised Time Taken (%)" Chart (Figure 57) shows the performance gains in total time taken for each reporting query as a percent of the original query time. In this chart the green bars to the left of 0% are improvements in speed, and the red bars to the right of 0% are decreases in performance. The report types where there are no entries in the chart are those reports that were either removed in our system due to design changes, or reports that did not work on the original system. The removed reports in our new system were those reports that were meaningless. For example, when looking at the last 12 months, with a time unit of month, the aggregate option does not make sense, as there is no way that a report will extend far enough to need the aggregation. In this example, the same data would be put in the report regardless of whether aggregate was selected or not, so to make it easier for the user we removed the aggregate option on that report and others like it.

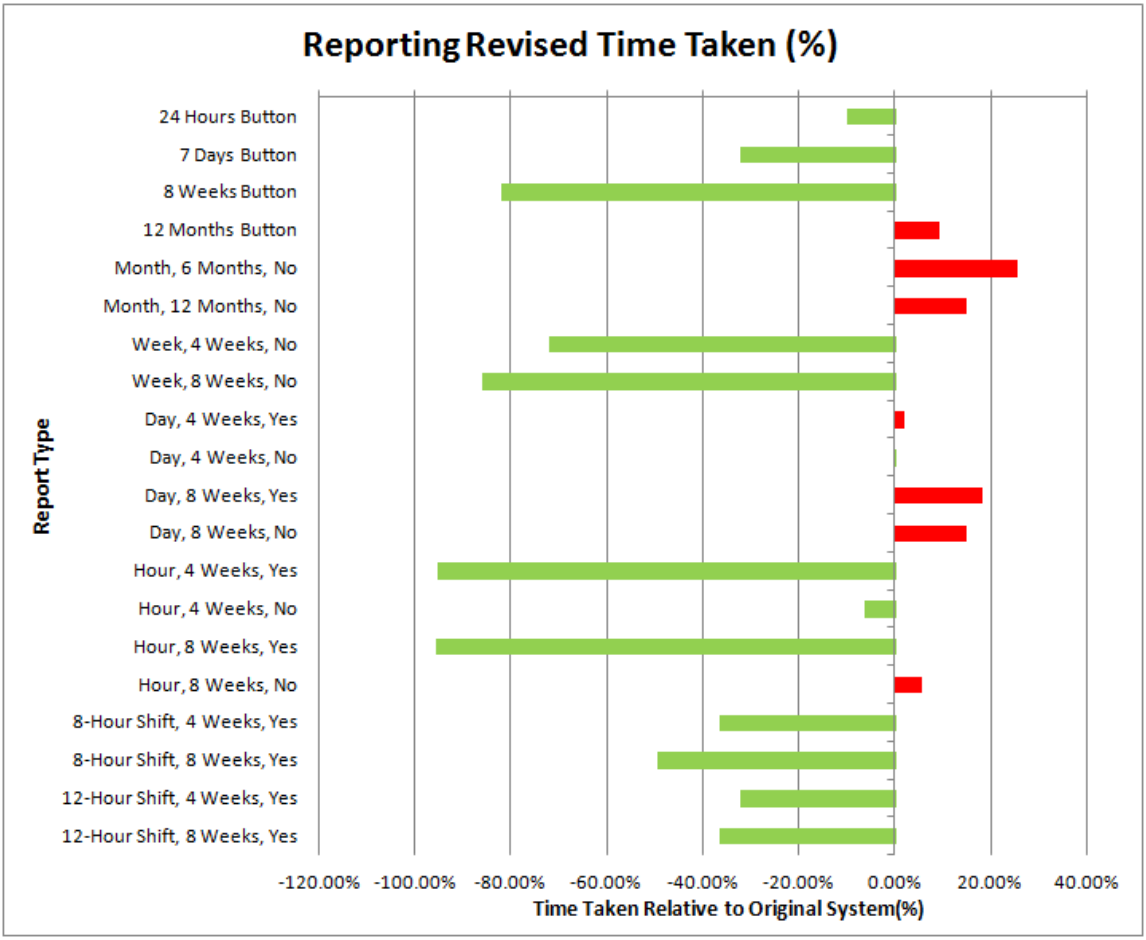


Figure 57: Reporting Revised Time Taken(%) The y-axis shows the reports tested. The x-axis shows the percent difference in time taken from the original. Green bars to the left of 0% had improved performance. Red bars to the right of 0% had reduced performance.

The “Reporting Revised Time Taken, Short (sec)” and “Reporting Revised Time Taken, Long (sec)” charts (Figures and) show the absolute time difference between our system and the original. They are split into two charts to better show the time gains for the reports that originally did not take very long.

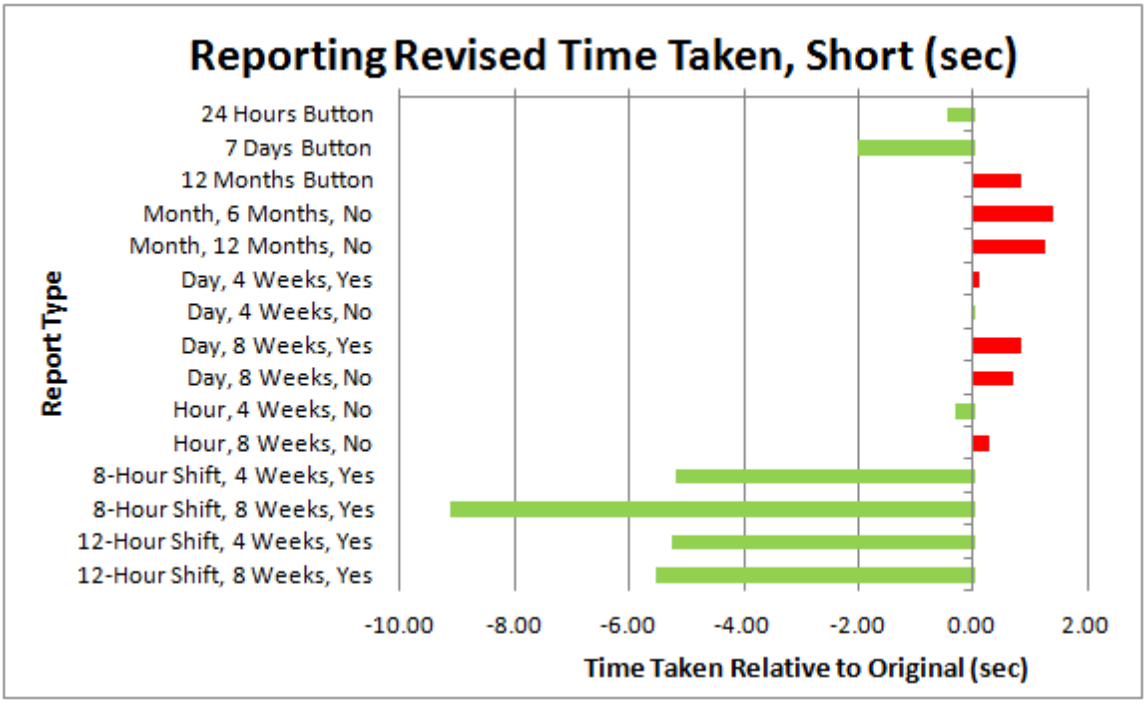


Figure 58: Reporting Revised Time Taken, Short (sec) - times less than 10 sec different. The y-axis shows the reports tested. The x-axis shows the difference in time taken from the original. Green bars to the left of 0% had improved performance. Red bars to the right of 0% had reduced performance.

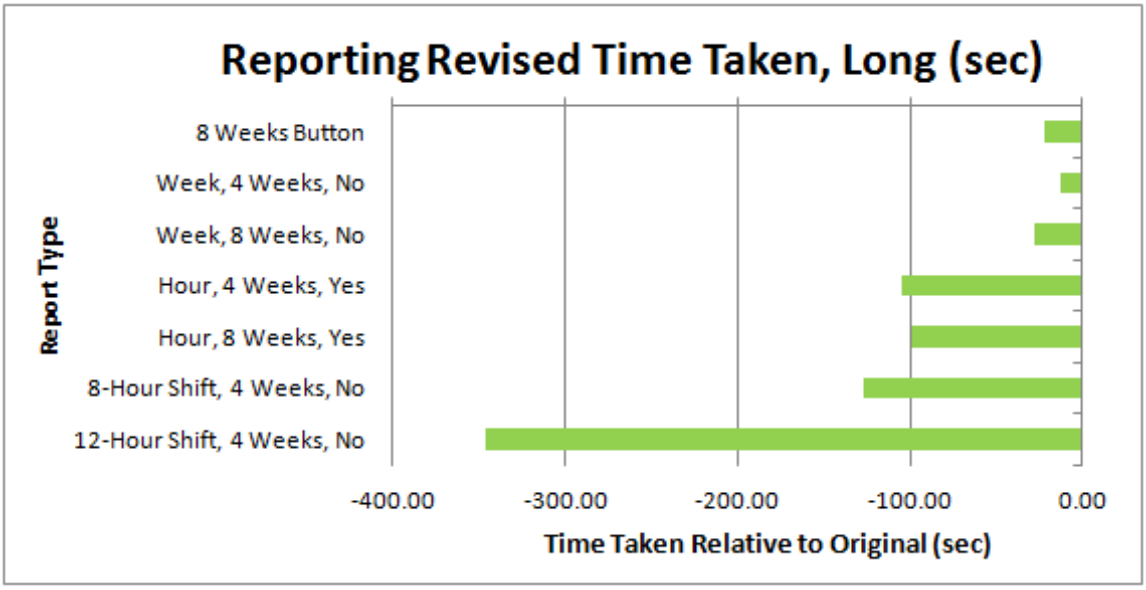


Figure 59: Reporting Revised Time Taken, Long (sec) - times more than 10 sec different. The y-axis shows the reports tested. The x-axis shows the percent difference in time taken from the original. Green bars to the left of 0% had improved performance. Red bars to the right of 0% had reduced performance.

8 Conclusion

The problem of maintaining proper hygiene standards, especially in hospitals, will likely never be completely solved. However, systems like Hyreminder have the potential to go a long way towards reducing the spread of hospital-acquired infection, saving on hospital operational costs, but most importantly: saving more lives.

This project built on and improved the original Hyreminder web application and the original Anomaly web application. Through a framework change, we drastically improved performance, both in database access and report generation, and improved code clarity. In the process of the reengineering, we expanded on both internal and external documentation, making it easier not only for future developers to maintain and expand the system, but also for users to learn how to use the system.

In addition to the reengineering, the team added several features. The JQuery Visualize library that we employed for charting was vastly expanded and can be released as a standalone library for others to use. We added reporting functionality for the ID Based and Multi-ICU versions of Hyreminder. The Anomaly detection web application was overhauled and now supports additional functionality such as maintenance comments, a map showing where the physical sensors are, and sensor maintenance.

The biggest feature we added, mobile bed tracking, enables system technicians to more easily track down missing sensors, especially those attached to moving beds, as well as to make keeping track of an individual bed's location easier. Even though the ID-based system that this bed tracking system relies upon has yet to be implemented by a hospital, plans are in place to install this system, at which time refinements to our work can be added to enhance performance of our mostly theoretical work.

Our reengineering of the system resulted in major performance gains in many areas. While our correctness testing verified that our system performed as expected, the performance testing of charting and report generation demonstrated that our system was faster and more efficient than the original in most respects. Those areas where there was no improvement or a slightly slower performance were judged to be acceptable tradeoffs for making the previously non-functioning charts and reports work, as those areas that saw performance losses suffered less than two additional seconds, whereas the vast majority of improved charts and reports saved much more than that on average.

9 Future Works

While now a number of people have worked on the interface, we are all not normal users and since we understand the system we are biased into thinking it is an easy to use and understand interface. We got basic feedback from a few of our peers, who had good outside opinions on the usability, however as highly technical people, they weren't normal users either. A usability study with a complete and thorough user study, especially including nurses and other hospital staff who will use the system the most, is advised. Getting feedback from people that specialize in user interfaces, such as Professors David Brown and Matt Ward at Worcester Polytechnic Institute, may also provide fruitful.

There are some user interface updates that we already know would improve the user experience, but unfortunately we did not get time to refine. In the Hyreminder web application, there are many pages that make use of drop-down filters to refine the data displayed. Currently on the form submission, these drop-downs return to their default value, but ideally they would remain on the value the user selected. On the Reporting pages with the time unit of hour, it would be nice to be able to pick an hour when choosing a custom time range. In the Anomaly web application, on the Maintenance pages, it would be more user friendly to use JavaScript to enforce required fields before the information is submitted to the server. Additionally, neither allows for deletion of a sensor nor a sensor location. In both web applications where charts are used, it would be more user friendly to have a loading sign instead of leaving the user wondering if it working.

There's more work to be done with the commenting feature. More information should be gathered on what type of preset comments will be most useful to the users. Incorporating these presets will be simple, but we did not have the time to gather the information. Furthermore, adding another layer to the sensor hits chart, at the times these comments are made would be an extremely useful feature. Allowing the user to set a time for the comment would make the comment layover more accurate.

As our system works right now, the are still raw database queries that will need to be executed to add a new ICU to the system, before the admin user can see it on the Options page in the System Maintenance application. Additionally, users and permissions must be set up with raw database queries. To make this a completely functional application, these functions should be added to the System Maintenance application.

Additional testing is never a bad thing. Our stress tests focus on several of the potential weak spots of the system, but there are other areas that can be tested as well. For example, there are many different iterations of report downloading that can be stress tested in addition to the ones we have already done. The stress tests already in place could also be ran at a higher limit to observe the system when handling even more data. One additional stress test was created but never ran due to time constraints. The Sensor Details test should be run to see how the system scales with

the number of sensors.

We created a very basic and highly theoretical bed tracking system. Once the system is in use, we are sure that the tracking algorithm will need to be improved. Additionally, the algorithm will need to be expanded to track beds moving between ICUs.

Works Cited

Apache POI Project, The. (2012). http://poi.apache.org/

Apache POI Project, The. (2012). POI-HSSF and POI XSSF - Java API To Access Microsoft Excel Format Files. http://poi.apache.org/spreadsheet/index.html

Atwood, Jeff. (2009, June). All Abstractions Are Failed Abstractions. http://www.codinghorror.com/blog/2009/06/all-abstractions-are-failed-abstractions.html

Baranovskiy, Dmitry. (2012). Raphaël. http://raphaeljs.com/

Belmonte, Nicolas Garcia. (2013). SenchaLabs. JFreeChart. http://philogb.github.com/jit/

Brazzell, Brena Edwards, et. al. (2011, June). Efficacy of an Electronic Hand Hygiene Surveillance and Feedback Monitoring Device Against Healthcare Associated Infections. *American Journal of Infection Control*. 39, 5, E172-E173. http://www.sciencedirect.com/science/article/pii/S0196655311006523

BSD 2 License. http://opensource.org/licenses/BSD-2-Clause

C, Nick. (2011, January). Why not to use Google Web Toolkit?. http://programmers.stackexchange.com/questions/38441/when-not-to-use-google-web-toolkit

CodePro Analytix. https://developers.google.com/java-dev-tools/codepro/doc/

Duke Medicine News and Communications. (2010, March). Technology-Based Hand Hygiene Monitoring Improves Compliance at Duke Hospital. Duke Medicine. http://www.dukehealth.org/health_library/news/technology_based_hand_hygiene_monitoring_improves_compliance_at_duke_hospital

Google Developers. (2012). Google Web Toolkit. https://developers.google.com/web-toolkit/

Harvest. (2011). Chosen. http://harvesthq.github.com/chosen/

Highsoft Solutions JS. http://www.highcharts.com/

Hospital Hygiene Drive 'Saved 10,000 Lives'. (2012, May). The Guardian. http://www.guardian.co.uk/society/2012/may/04/hospital-hygiene-drive

JDocs. Joda Time API. http://www.jdocs.com/jodatime/1.2.1/overview-summary.html

jQuery Foundation, The. (2013). jQuery UI Datepicker. http://jqueryui.com/datepicker/

jQuery Foundation, The. (2013). jQuery UI Dialog. http://jqueryui.com/dialog/

Jehl, Scott. (2013). JQuery Visualize. https://github.com/filamentgroup/jQuery-Visualize

JUnit. http://junit.sourceforge.net/

Lee, Christopher. (2006, November). Studies: Hospitals Could Do More to Avoid Infections. The Washington Post. http://www.washingtonpost.com/wp-dyn/content/article/2006/11/20/AR2006112001122.html

MedlinePlus. (2013, February). Infection Control. http://www.nlm.nih.gov/medlineplus/infectioncontrol.html

MIT License. http://opensource.org/licenses/MIT

msdn. Model View Controller. http://msdn.microsoft.com/en-us/library/ff649643.aspx

Mularien, Peter. (2010). *Spring Security 3: Secure Your Web Applications against Malicious Intruders with This Easy to Follow Practical Guide*. Birmingham UK: Packt Publishing.

Nielsen, J. (2006). F-Shaped Pattern For Reading Web Content. http://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/

Open Flash Chart for GWT. http://code.google.com/p/ofcgwt/

Oracle. (2013). Introduction to Security in the Java EE Platform. http://docs.oracle.com/javaee/6/tutorial/doc/bnbwj.html

Oracle. (2011). JAAS Reference Guide. http://docs.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html

PMD. (2013). http://pmd.sourceforge.net/

Prüss, A., Giroult, E. & Rushbrook, P. (1999). Hospital Hygiene and Infection Control. In *Safe management of wastes from healthcare activities*. World Health Organization. http://www.who.int/water_sanitation_health/medicalwaste/148to158.pdf

Selenium Browser Automation. http://docs.seleniumhq.org/

Spring Source Community. (2013). http://www.springsource.org/

SpringSource. Spring Security Documentation. http://static.springsource.org/spring-security/site/reference.html

String Function. (2010). SHA1 ONLINE HASH. http://www.stringfunction.com/sha1-hash.html

ThoughtWorks Technology Advisor Board. (2011, July). Technology Radar.

Viklund, Andreas. (2012). Object Refinery Limited. JFreeChart. http://www.jfree.org/jfreechart/

Vogel, Lars. (2009, August). Dependency Injection with the Spring Framework - Tutorial. http://www.vogella.com/articles/SpringDependencyInjection/article.html

W3 Consortium. (2010, November). Display capabilities. http://www.w3.org/International/questions/qa-display-capabilities.en.php

w3schools. MySQL DATE_FORMAT() Function. http://www.w3schools.com/sql/func_date_format.asp

Worcester Polytechnic Institute Database System Research Group. (2011). Hyreminder IDE Setup & User Manual.

A User Guide of Activities on Web Application

This user guide walks you through the Hyreminder Sensor Maintenance application.

Contents:

- Details for each of the screens and the tasks that are accomplished on each.
- Suggested approach for setting up a new ICU.

A.1 Current Errors

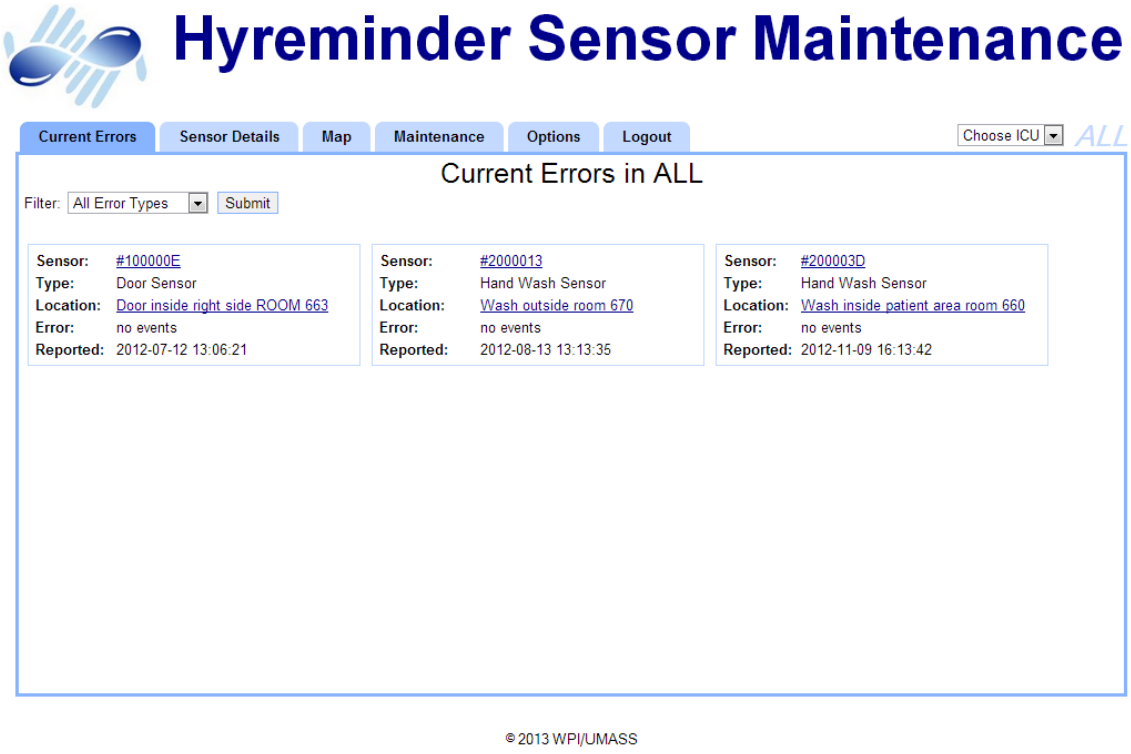


Figure A1: Current Errors Screen

The Current Errors tab is used to see which sensors have errors. As in the general Hyreminder application, you can use the drop-down in the top right corner to switch ICUs. This will filter the sensors to only those in the selected ICU. Additionally, you can filter which types of errors by using the drop-down in the top left corner.

Each box contains the details on one sensor with an error. It contains:

- the physical sensor ID
- the type of sensor
- the location
- the type of error and when it was reported

The physical sensor ID is linked to the appropriate Sensor Details page, while the location is linked to the appropriate Map page. This means that you can click on the link and you will go the the stated page with the selected physical ID already selected.

A.2 Sensor Details

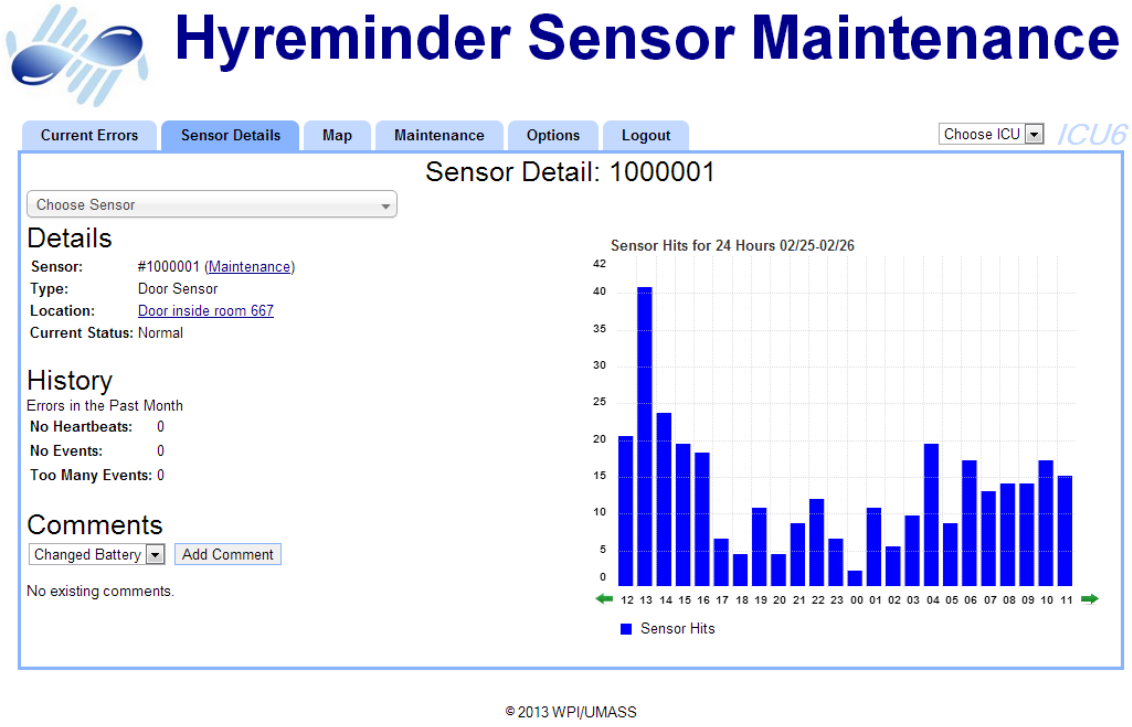


Figure A2: Sensor Detail Screen

The Sensor Details tab is used to view detailed statistics about a particular sensor. If you clicked on a particular sensor link from the Current Errors tab, the details of that selected sensor will already

be loaded. To select a new sensor, use the drop-down at the top left of the screen to pick one. This drop-down is searchable, you can start typing in the ID or part of the location to find a sensor.

Once loaded, on the left side of the screen you will see three sections: details, history and comments. On the right side of the screen is a chart of the hits on the sensor.

Details - contains basic details on the sensor including:

- ID
- type
- location
- status

History - contains how many times the sensor has had the given type of error.
Comments - where you can view, add and remove comments on the sensor.

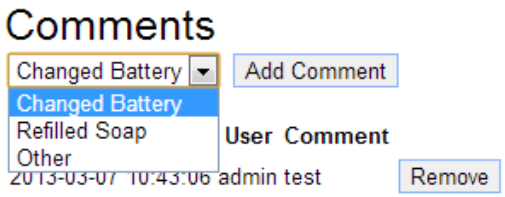


Figure A3: Sensor Comment Section

To add a comment, first choose a message from the drop-down and click the "Add Comment" button. If you choose "Other", you may enter your own message in the text box that appears.

To remove a comment, simply click the "Remove" button next to the appropriate comment.

A.3 Map

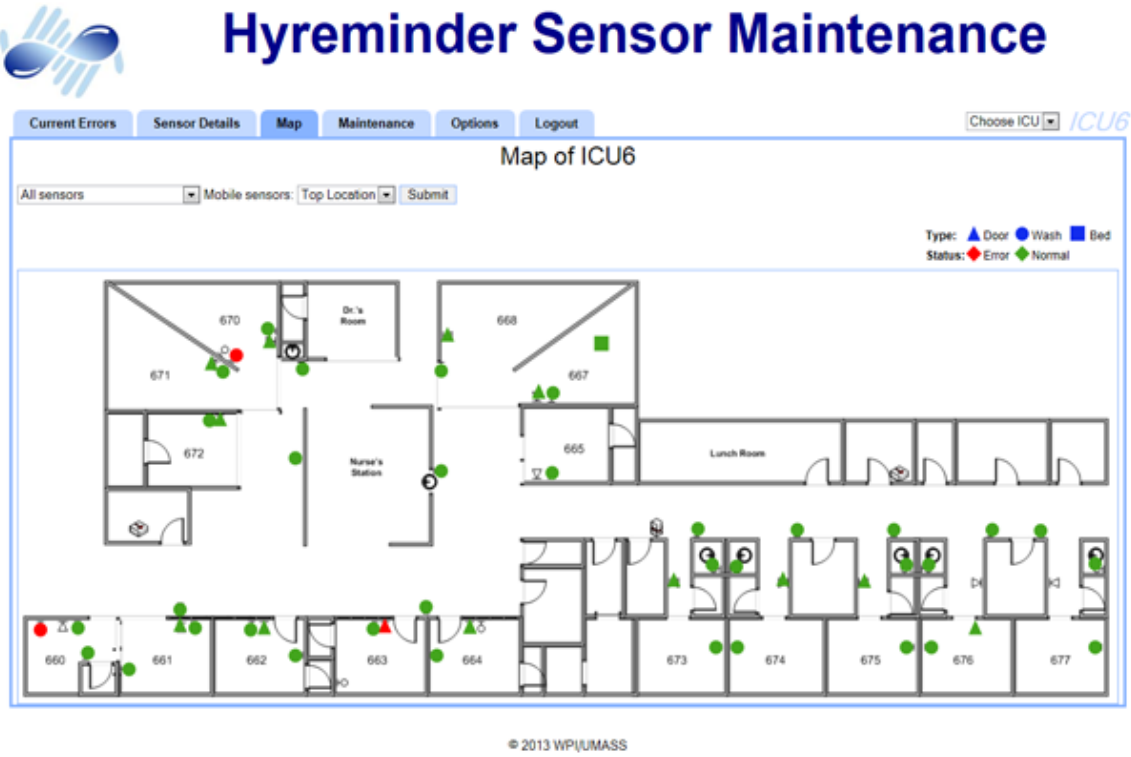


Figure A4: Map Screen

The Map tab is used to view where sensors are located on a map of the ICU. If you clicked on a link from the Current Errors tab or the Sensor Details tab, the appropriate sensor will be the only one on the map. You can change which sensors are displayed on the map, using the filters at the top left of the screen.

- All sensors – displays all sensors in the ICU.
- Sensors with errors – displays only those sensors that were listed on the Current Errors tab, in the selected ICU.
- Selected sensors – choose which sensors to display.
  - Choosing this option will create a multi-select box. Choose all desired sensors.

Since mobile sensors have the potential to be in more than one place, you must choose how you want this displayed.

- Top Location – shows only the location where it is most likely to be.
- All Locations – shows all possible locations, where the confidence is greater than or equal to the confidence entered.

On the map you can hover over a sensor and see details in the upper left corner. Additionally, you can click on the sensor to go to the Sensor Details page.

A.4 Maintenance

The Maintenance tab is used for system maintenance: which includes setting up sensor locations on the map and updating sensor IDs.

A.4.1 Sensor Maintenance

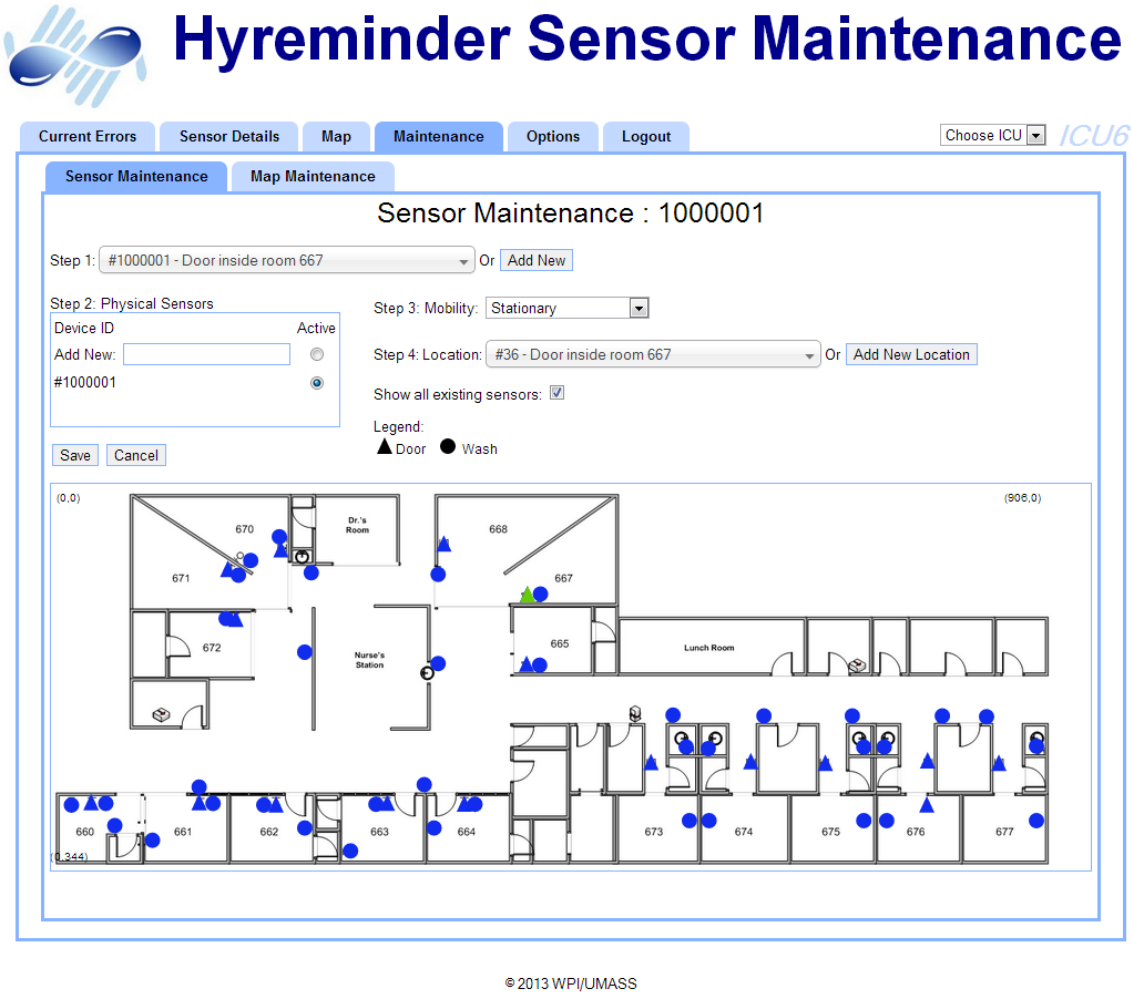


Figure A5: Sensor Maintenance Screen

The Sensor Maintenance sub-tab is used for creating and editing sensors. This includes tasks such as adding new physical sensors, activating and deactivating physical sensors, and updating a location when applicable.

To Add a New Sensor:
Use this when you are setting up the ICU for the first time. After initial setup, adding a new sensor should be a rare task.

1. Click the "Add New" button.
2. Look at the physical sensor and enter the device's ID in the "Add New" box on the right side of the screen.
3. Choose "Mobility" type from drop-down.
   (a) Stationary – indicates that this sensor will not move
   (b) Mobile – indicates that this sensor may move about the ICU or hospital
4. If you choose "Stationary", more fields will appear for entering the location
   (a) Either choose an "Existing Location" or "Add New Location".
   (b) Add New Location will take you to the Map Maintenance Screen (explained later).

To Edit an Existing Sensor:
Follow these steps when you are performing maintenance on the system, for example when you are replacing a physical sensor with a new one.

1. Select a sensor from the "Change Existing" drop-down. This drop-down is searchable.
2. The information about the sensor will be automatically filled in.
3. Edit the details as desired.
   (a) If you are replacing a sensor, enter the new ID into the "Add New" box, and check the radio button under "Active". This will retire the previous sensor.

A.4.2 Map Maintenance

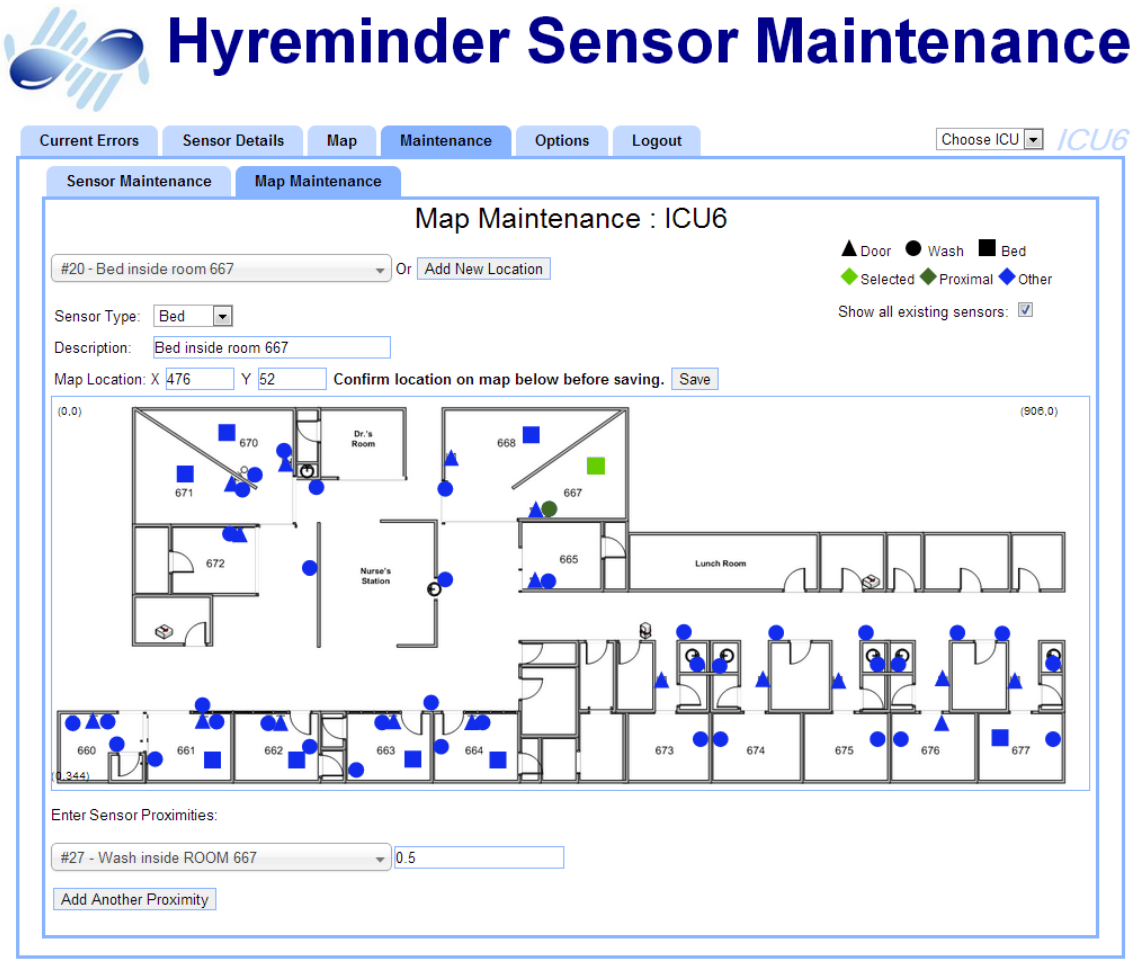


Figure A6: Map Maintenance Screen

The Map Maintenance sub-tab is used for creating new sensor locations and editing existing ones. This page shares the same features with the map interface on the Sensor Maintenance sub-tab.

To Add a Location

1. At the top left, select a sensor Type from the drop-down.
2. Enter a meaningful Description for the location.

3. Map Location: X and Y: the x and y coordinates on the ICU Map. Tip: You can click on the sensor on the map and drag it to a new location.

4. Lastly, enter which sensors are in close proximity.

   (a) Click the "Add Another Proximity" button. A new drop-down and text field will appear.

   (b) Select a sensor from the newly created drop-down. The sensor will be briefly circled on the map.

   (c) Enter the proximity in the newly created text field. A proximity should be entered in seconds. You can edit the proximities later.

   (d) Repeat steps until you've entered proximities for the nearest sensors.

A.5 Options

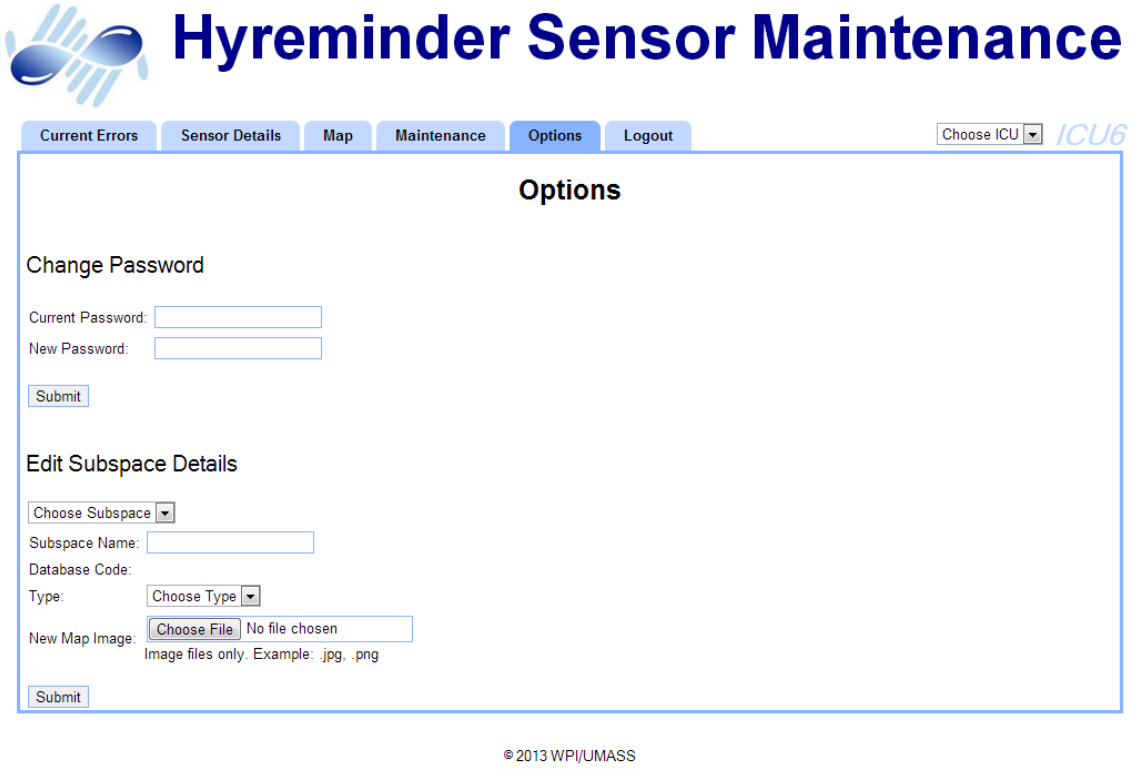


Figure A7: Options Screen

A.5.1 Change Password

As with the Hand Hygiene Performance application you can change your password. Simply enter your Current Password and New Password in the appropriate fields and click the "Submit" button. If you forget your password, please email the Hyreminder group: hyreminder@cs.wpi.edu

A.5.2 Edit Room Details

This section is used for editing the details of ICUs and uploading the floor plans for each.

1. Select the ICU from the drop-down. The ICU's details will be populated.
2. Edit the room's details.
   (a) ICU Name: the display name used in both the Hand Hygiene Performance application and the Hyreminder System Maintenance application.
   (b) Database Code: a non-updateable field, what the ICU is known as in the database
   (c) Type: Group Based or ID Based
3. Upload a new Map image.
   (a) Be sure to upload an image file (for example .jpg or.png).
   (b) Be sure your image includes only what you want - no extra whitespace.
   (c) You want as little detail as possible: walls and doorways and labels for rooms.Too much detail will create confusion.

A.6 Setting Up a New ICU

1. Coordinate with the Hyreminder team.
   (a) Ask them to add a new ICU to the room table in the database.
   (b) Inform them which user should have access to this new ICU.
2. On the Options screen
   (a) Choose the new ICU from the drop-down under "Edit Room Details"
   (b) Confirm ICU name and type are correct and edit as needed.
   (c) Upload the floor plans. To change a floor plan, upload a new image and the old one will be deleted.
3. On the Map Maintenance screen

(a) Enter the sensor locations. These can always be tweaked after installation of the system.

4. On the Sensor Maintenance screen

(a) As you physically install each sensor, enter it as a New sensor.

(b) If the sensor is stationary, choose one of the locations you just set up.

B User Accounts and Security Development Docs

These docs are meant to give a developer a better understanding of the underlying components that control user accounts and security.

Contents:

- Database design for all tables involved with user accounts.
- How to add a new user and user roles
- Login credential verification.
- How to restrict access to parts of Hyreminder based on account permissions.

B.1 User Accounts Database Design

B.1.1 users table

```
mysql> describe users;
+----------+------------------+------+-----+---------+----------------+
| Field    | Type             | Null | Key | Default | Extra          |
+----------+------------------+------+-----+---------+----------------+
| USER_ID  | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| USERNAME | varchar(45)      | NO   |     | NULL    |                |
| PASSWORD | varchar(45)      | NO   |     | NULL    |                |
| ENABLED  | tinyint(1)       | NO   |     | NULL    |                |
+----------+------------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Figure B1: Table Schema: users

Usernames and passwords for each account is stored in the 'users' table as VARCHAR with a max size of 45 characters. Accounts can be enabled or disabled by setting the ENABLED attribute to 1 or 0 respectively.

B.1.2 user_roles table

```
mysql> describe user_roles;
+--------------+------------------+------+-----+---------+----------------+
| Field        | Type             | Null | Key | Default | Extra          |
+--------------+------------------+------+-----+---------+----------------+
| USER_ROLE_ID | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| USER_ID      | int(10) unsigned | NO   | MUL | NULL    |                |
| AUTHORITY    | varchar(45)      | NO   |     | NULL    |                |
+--------------+------------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

Figure B2: Table Schema: user_roles

The 'user_roles' table stores the roles, or permissions, associated with each account. The user_roles.USER_ID attribute is a foreign key that references user.USER_ID. Each account can have multiple roles. The name of the roles are up to the developer but should follow the standard 'ROLE_<Title><ICU Number>' to remain consistent. Some examples of the roles that already exist are 'ROLE_ADMIN', 'ROLE_NURSE5', 'ROLE_NURSE6', etc. These roles will be used to determine which accounts have access to specific parts of the Hyreminder site.

B.2 Inserting New Data

B.2.1 Adding a New User

```
INSERT INTO users (USERNAME, PASSWORD, ENABLED) VALUES ('corey', '123456', 1);
```

Passwords must be encoded used SHA-1 encryption before being placed in the database. If you are doing this manually, there are online tools[1] to perform the encryption. Once again, accounts can be enabled or disabled by setting the ENABLED field to 1 or 0 respectively.

B.2.2 Adding User Roles

```
INSERT INTO user_roles (USER_ID, AUTHORITY) VALUES (1, 'ROLE_ADMIN');
```

The roles should follow the standard ROLE_<Title><ICU Number>as to not interfere with other parts of the code. Remember the names must correspond to the expected role names in the security xml file (discussed later). Each account can have multiple user roles.

[1]Example SHA-1 encryption tool: String Function. (2010). SHA1 ONLINE HASH. http://www.stringfunction.com/sha1-hash.html

B.3 Security

B.3.1 Verifying Login Credentials

Much of the user verification is done automatically by Spring Security[2]. The two key pieces of code that determine this is the Authentication Manager defined in the spring-security.xml file located in src/main/resources directory and the HTML login form found in login.jsp in the src/main/webapp/WEB-INF/views directory.

The authentication manager section of the spring-security.xml config file is located at line 37. Here we specify the data source (aka database connection), the password encoder type, and the SQL queries to retrieve the credentials and user roles. Spring Security will automatically do the decryption and comparisons to verify login.

Spring Security requires specific IDs in the HTML login form found in login.jsp. The action of the login form is 'j_spring_security_check' a built-in action that will request Spring Security automatically do the password verification. The IDs of the username and password input boxes are also built-in. Here is the login form HTML:

```
<form id="login" action="j_spring_security_check" method="post" >
    <table align="center">
        <tr>
            <td><label class="title" for="j_username">Username: </label></td>
            <td><input id="j_username" name="j_username" type="text" /></td>
        </tr>
        <tr>
            <td><label class="title" for="j_password">Password: </label></td>
            <td><input id="j_password" name="j_password" type="password" /></td>
        </tr>
    </table>
    <input type="submit" value="Login"/><br />
</form>
```

B.3.2 Securing URLs

Securing parts of the website is as simple as specifying the URL pattern and the roles allowed to access it in the spring-security.xml file. An example that can be found at line 16 of the xml file is:

```
<security:intercept-url pattern="/ICU5/**" access="hasAnyRole('ROLE_ADMIN', 'ROLE_NURSE5')"/>
<security:intercept-url pattern="/ICU6/**" access="hasAnyRole('ROLE_ADMIN', 'ROLE_NURSE6')"/>
<security:intercept-url pattern="/ICU7/**" access="hasAnyRole('ROLE_ADMIN', 'ROLE_NURSE7')"/>
```

As you can see, any URL with the prefix /ICU5/ will require the account that is logged in to have ROLE_ADMIN or ROLE_NURSE5. Exact URLs can be given or wildcards such as '*' can be used to incorporate many URLs. If the account fails this check, the user will be redirected to the access denied page given at line 11. Once successfully logged in, the user will be redirected to the overall

[2]Spring Security Docs: http://static.springsource.org/spring-security/site/reference.html

compliance page of the lowest number ICU they have access to. It is important that all possible URLs are captured by this series of URL patterns to ensure proper protection. An example of the XML file can be found in Appendix C.

C Excerpt from XML Config File for Spring Security

```
<!-- This is where we configure Spring-Security  -->
<security:http auto-config="true" use-expressions="true" access-denied-page="/denied" >

<!-- List all URLs that Spring Security will intercept and the permitted user access level -->
<security:intercept-url pattern="/login" access="permitAll"/>
<security:intercept-url pattern="/ALL/*" access="hasRole('ROLE_ADMIN')"/>
<security:intercept-url pattern="/ICU5/*" access="hasAnyRole('ROLE_ADMIN', 'ROLE_NURSE5')"/>
<security:intercept-url pattern="/ICU6/*" access="hasAnyRole('ROLE_ADMIN', 'ROLE_NURSE6')"/>
<security:intercept-url pattern="/ICU7/*" access="hasAnyRole('ROLE_ADMIN', 'ROLE_NURSE7')"/>

<security:intercept-url pattern="/options" access="hasAnyRole('ROLE_ADMIN', 'ROLE_NURSE5', 'ROLE_NURSE6', 'ROLE_NURSE7')"/>


<!-- Define which request mapping to go to for login/logout -->
<security:form-login
    login-page="/login"
    authentication-failure-url="/login?error=true"
    default-target-url="/" />

<security:logout
    invalidate-session="true"
    logout-success-url="/login"
    logout-url="/logout"/>

</security:http>

<!-- Declare an authentication-manager to use a JDBC dataSource -->
<security:authentication-manager>
    <security:authentication-provider>
        <security:password-encoder ref="passwordEncoder"/>
        <security:jdbc-user-service data-source-ref="dataSource"

            users-by-username-query="
                select username,password, enabled
                from users where username=?"

            authorities-by-username-query="
                select u.username, ur.authority from users u, user_roles ur
                where u.user_id = ur.user_id and u.username =?  "
        />
    </security:authentication-provider>
</security:authentication-manager>
```

Figure C1: Spring Security XML Config File Except

D Reporting Module Developer Documentation

These docs are meant to give a developer a better understanding of the reporting module.

Contents:

- Reporting module relation diagram
- Reporting module logic design
- POI framework and report template
- How time aggregation works
- Stored procedures used

D.1 Reporting Module Structure (Relation Diagram)

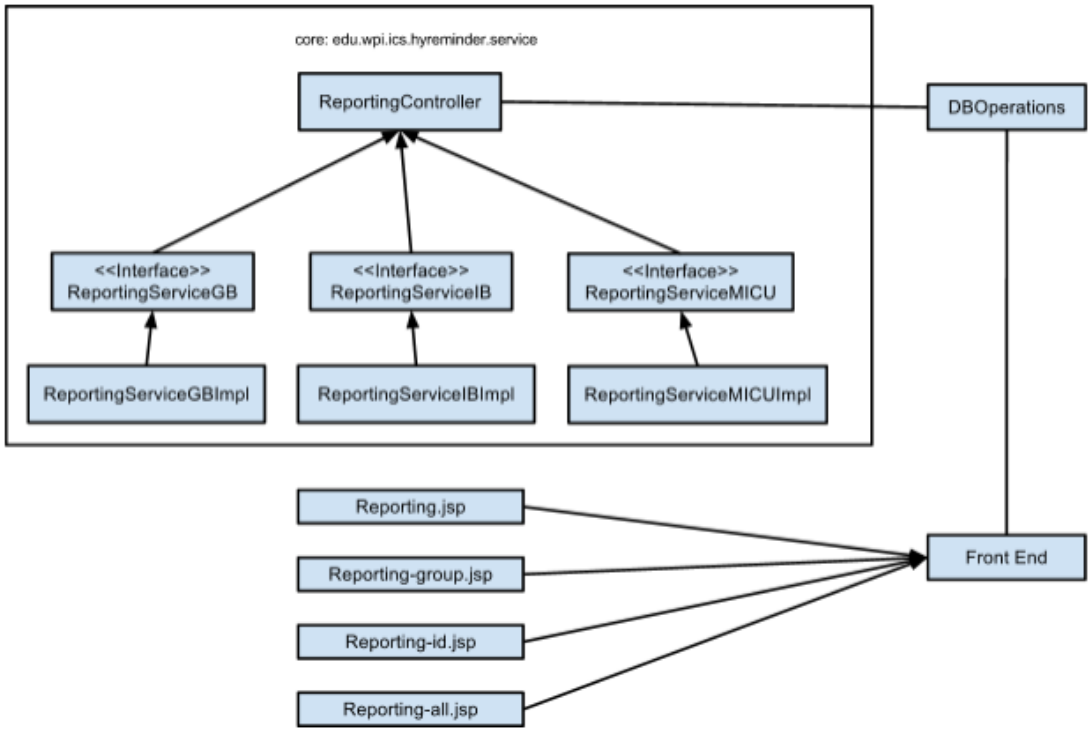


Figure D1: Reporting Module Structure. GB, IB and MICU refer to Group Based, ID Based and Multi-ICU respectively

D.2 POI 3.8 framework & XLS templates

The Apache POI[1] is the Java API for generating Microsoft documents. In Hyreminder, the sub-component "HSSF (Horrible Spreadsheet Format)"[2] is used, a specific interface to read and write Microsoft Excel (.xls) format files.

D.3 Reporting Module Logic

The general query and delivery process for a generated report can be defined as follows:

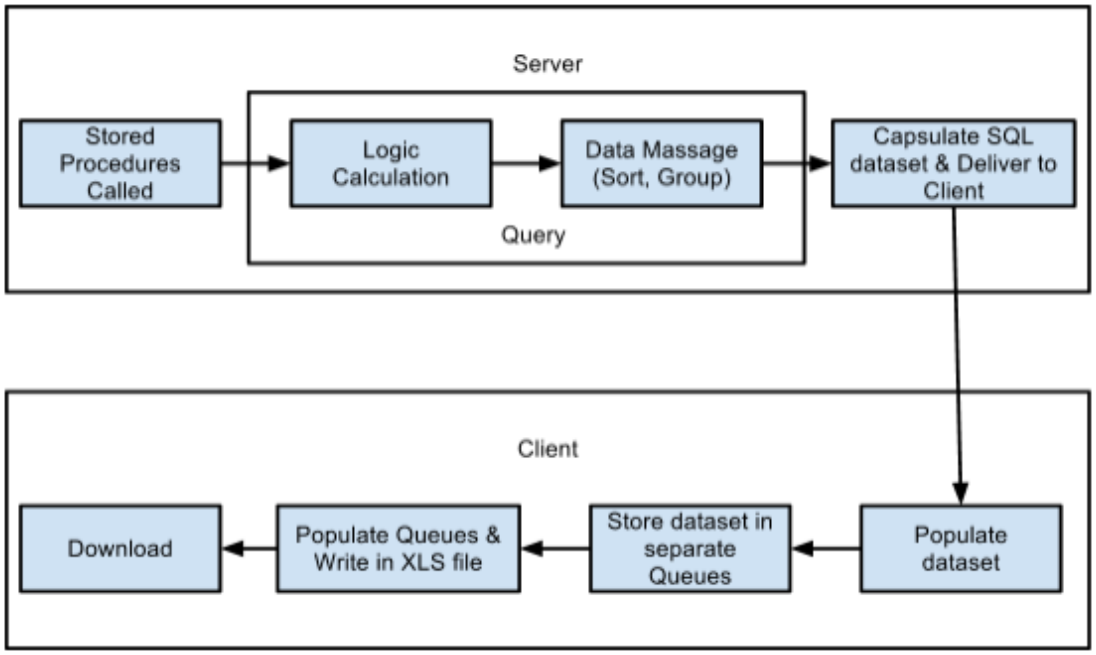


Figure D2: Reporting Module Logic

D.3.1 Controller Mapping

The ReportingController, as seen in Figure D1, defines different requests for Group Based, ID Based and Multi-ICU URLs as follows:

[1] Apache POI Project, The. (2012). http://poi.apache.org/

[2] Apache POI Project, The. (2012). POI-HSSF and POI XSSF - Java API To Access Microsoft Excel Format Files. http://poi.apache.org/spreadsheet/index.html

In ReportingController.java, define URL mapping request:

```
@RequestMapping(value = "/{room}/reporting/GroupbasedPreset", method = RequestMethod.GET)
```

Parse URL path as variable in order to locate the current unit:

```
@PathVariable(value = "room") String roomCode, final Model model)
```

In reporting-group.jsp file, the request is called as below:

```
onClick="location.href='reporting/GroupbasedPreset?rows=24'"
```

So that "rows=24" is passed as a parameter to generate data from the past 24 hours. Because the reports are presets, setting rows to 7,8, or 12 will generate data for the past 7 days, 8 weeks, or 12 months, respectively

D.3.2 XLS Template

There are multiple ways to generate downloadable excel files with POI. One way of doing this would be the client generates the excel file and bar charts from scratch with its own POI framework. However, in order to enhance performance, we used pre-defined templates inside of the following template files:

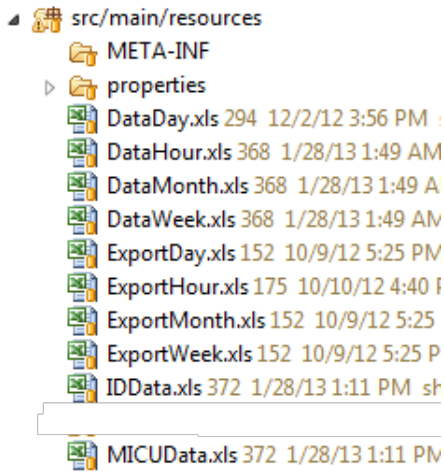


Figure D3: XLS Template Files

In above template files, we created one spreadsheet to store queried data from the server. Another spreadsheet is used for generating bar charts based on the data sheet. In order to pull the

right data, bar chart configuration needs to be defined based on how many columns and rows of data need to be included. Data*.xls are templates used for preset buttons in Group-based reporting page, and the bar chart is contained in these templates. Export*.xls are templates used for generating Group-based customizable data reports, which only contain data. IDData.xls, MICUData.xls are the templates for ID-based and Multi-ICU modules. All template files are defined as input stream in *makeExcel* function as follows:

```
// get data for 24 hours
    inp = new ClassPathResource("ExportHour.xls").getInputStream();
```

D.3.3 Date & Time API

The API we used for querying specific date and time is Joda-Time (org.joda.time.DateTime).[3] Joda-Time provides a quality replacement for the Java date and time classes. The design allows for multiple calendar systems, while still providing a simple API. The "default" calendar is the ISO8601 standard which is used by the XML.

For example, to set a time range from the last 12 month till now, the Java expression would be to define the start date by getting the current date time and the subtracting 12 months from the current date.

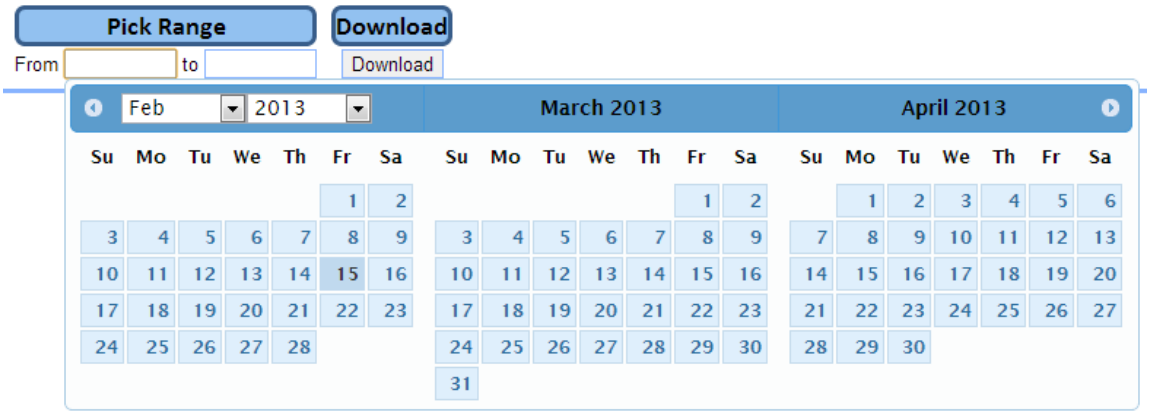


Figure D4: Date selector in the user interface

[3] JDocs. Joda Time API. http://www.jdocs.com/jodatime/1.2.1/overview-summary.html

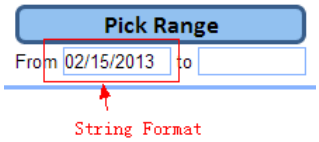


Figure D5: Date Feb. 15, 2013 selected

In customizable time selector, the library will generate a string time format which needs to be converted to DateFormat before being able to do calculation via the *dateConverter* function:

```
private DateTime dateConverter(String inputDate) throws ParseException {
    DateTime outputDate = null;

    DateFormat df = new SimpleDateFormat("MM/dd/yyyy");
    Date date = df.parse(inputDate);
    outputDate = new DateTime(date);

    return outputDate;
}
```

D.3.4 Aggregation

The aggregation data can directly be generated by the stored procedures depends on the MySQL DATE_FORMAT()[4] parameters passed in.

For example, to query data for the past 3 months and aggregate by week unit. The format will be "%u" (Monday is the first day of week).

D.3.5 Parse dataset

All stored procedure return a sql data set and each column of data is stored in a separate queue:

```
// Parse set
    while (set.next()) {
        getHandHygieneEventsResult.add(set.getInt("HandHyCounts"));
        getRoomEntryExitsResult.add(set.getInt("EnExCounts"));
        getRatioResult.add(set.getDouble("Performance") / 100);
        getTimeIndexResult.add(set.getString("groups"));
    }
```

set.next() is necessary to avoid Null Pointer Exception.

D.3.6 POI I/O

Before writing into a template file, POI needs to initialize the input stream by choosing which template file to populate. *wb.getSheetAt(1)* means POI will start to overwrite from sheet 2 (this

[4]w3schools. MySQL DATE_FORMAT() Function. http://www.w3schools.com/sql/func_date_format.asp

situation only occurs when Sheet 1 contains data charts).

```
// Initialize input stream, workbook, data format, cell style, and
// the sheet
wb = WorkbookFactory.create(inp);
format = wb.createDataFormat();
style = wb.createCellStyle();
style.setDataFormat(format.getFormat("0"));
Sheet sheet = wb.getSheetAt(1);
```

The row number of the dataset needs to be known in order to loop through all queues without errors occurring. Since all excel cells are considered as null (not initialized), whenever POI writes to an uninitialized cell, a null pointer exception will occur. In this way, POI requires us to initialize rows and cells before being able to write data to the file as follows.

Initialize rows:

```
// Initialize row
row[i] = sheet.getRow(i + 1);
```

Get cells:

```
// for each cell, initialize it and set the value.
c = row[i].getCell(0);
```

D.4 Stored Procedures Used

Stored procedures[5] are pre-installed queries in the database serve to improve a querys performance between the client and the server.[6] The following report-related procedures are currently supported:

Group-based

- GetGBHygiene
- GetGBHygieneBy8HourShift
- GetGBHygieneBy12HourShift

ID-based

- GetWorkersComparison
- GetShiftReport

[5]Used under *edu.wpi.ics.component.database.DBOperations.java*

[6]Performance is improved by reducing query data sent from client to server. All stored procedures are handled and called through DBOperations class.

- GetIndividualsComparisonNonAggregate
- GetIndividualsComparison

Multi-ICU

- GetUnitsReport

E Selenium Tests and Stress Tests Developer Docs

These docs are meant to give a developer a better understanding of the underlying components of the Selenium Webdriver tests and the stress tests.

Contents:

- What makes up the Selenium tests.
- How to run the Selenium tests.
- What makes up the stress tests.
- How to run the stress tests.

E.1 Selenium

The Selenium framework (org.openqa.selenium.*) uses a series of methods to locate part of a webpage and perform actions that mimic that of a user. In order to create new Selenium tests, the following components must be included.

Components

1. The WebDriver

Located in the /src/test/resources directory is a driver for Chrome and a driver for Internet Explorer. In order to use Firefox, you must install Firefox and point the webdriver path to the firefox.exe.

2. Setting the System Properties

For each webdriver, use the following code to point the webdriver to the correct path.

```
/**
 * Sets system properties to point to the locations of the web drivers for
 * all browsers that will be tested. Calls setup().
 */
@BeforeClass
public static void setProperties() {
    // Firefox must be installed on your system
    System.setProperty("webdriver.firefox.driver", FIREFOX_DRIVER_PATH);
    // Chrome and IE have a webdriver executable that is located in the test resource directory
    System.setProperty("webdriver.chrome.driver", CHROME_DRIVER_PATH);
    System.setProperty("webdriver.ie.driver", IE_DRIVER_PATH);
    setup();
}
```

3. Navigation

Using the method `driver.get();` will make the given browser navigate to the specified URL.

4. Locating Elements on the Page

Webdrivers can locate elements using the `driver.findElement();` method. Three different options can be passed in as a parameter. `By.id("id_name")`, `By.className("class_name")`, or `By.xpath("xpath")`. The findElement method will return a WebElement object that can be manipulated.

5. WebElement Manipulation

The two most common things to do to a WebElement is `web.click()` or `web.sendKeys("keys")`. The names are self explanatory.

```
/**
 * Types in username and password and submits the form.
 *
 * @param driver
 *      WebDriver to send the input to
 * @param username
 *      String representing account username
 * @param password
 *      String representing account password
 */
private static void login(WebDriver driver, String username, String password) {
    WebElement username_field = driver.findElement(By.id("j_username"));
    WebElement password_field = driver.findElement(By.id("j_password"));

    username_field.sendKeys(username);
    password_field.sendKeys(password);
    password_field.submit();
}
```

Running the Test

The Selenium tests are created from a JUnit test file. Include the @Test annotation above each function that represents a test, right click on the JUnit file and select "Run as Junit test". An example of one test is shown below.

```
@Test
public void month_12_yes() {
    for (WebDriver driver : drivers) {
        drop(driver, "Month", "Last 12 Months", "Yes");
        WebElement c = driver.findElement(By.xpath("//button[text()='Download']"));
        DateTime s = DateTime.now();
        c.click();
        DateTime e = DateTime.now();
        float n = (float)(e.getMillisOfDay() - s.getMillisOfDay())/(float)1000;
        System.out.println("Month, Last 12 Months, Yes - " + n + " seconds");
        driver.switchTo().alert().dismiss();
    }
}
```

E.2 Stress Tests

The stress test framework implemented in the component project provides a means of easily producing tests designed to test the limits of a system. Specific tests will be run with various parameters, timed, and produced in a formatted report. The following components are needed to produce a stress test.

Components

1. Stress Test Unit

The stress test unit is the basic component of a stress test. A stress test unit is any class that implements the edu.wpi.ics.component.testing.stress.StressTestUnit interface. This interface provides three methods that must be implemented: setup, teardown and run. Optionally, a stress test unit can implement any number of parameter methods, designed as regular java setter methods, e.g. "setParameter." Parameter methods must take an int as an argument.

A stress test is split up into a number of runs. For each run, the runner will first call the setup method of the unit, followed by the run loop which will repeatedly set the parameters and call run based on how the run is configured. Finally, once the run is complete the teardown method will be called.

2. XML Configuration

The XML configuration is how you specify how the runner should run your test units. An example XML configuration is provided below.

```
<stress:report class="edu.wpi.ics.component.testing.stress.HTMLReportFormatter"/>

<stress:test class="edu.wpi.ics.hyreminder.test.stress.SampleStressTest">


    <stress:runs>
        <stress:run count="2" repeat="10">
            <stress:parameter name="parameter" start="10" step="10"/>
            <stress:parameter name="sample" start="5" step="5"/>
        </stress:run>
        <stress:run count="2">
            <stress:parameter name="parameter" start="5" step="2"/>
            <stress:parameter name="sample" start="5"/>
        </stress:run>
    </stress:runs>
</stress:test>
```

The root tag for a stress xml document is the <stress:tests> tag, and this tag can contain one <stress:report> and any number of <stress:test> tags, both of which are equally important. The <stress:report> tag allows you to specify a fully qualified class name to an implementation of the edu.wpi.ics.component.stress.StressTestReportFormatter interface which will be used to create the

reports for the entire test once it is complete. By default, edu.wpi.ics.component.stress.HTMLReportFormater is provided to allow for basic report formatting into an html document.

The <stress:test> tag is significantly more complicated and it is where you set specifically how the runner will run a StressTestUnit. Like with the <stress:report> tag, you must specify which unit class you want the test to run with the class attribute. Within the <stress:test> tag is exactly one <stress:runs> tag which can contain any number of ¡stress:run¿ tags.

Each <stress:run> tag specifies exactly one unit of setup, run, repeat, teardown that the runner will execute. The <stress:run> tag has three attributes and can have any number of <stress:parameter> tags inside of it. The attributes on the <stress:run> tag are count, repeat and timeout. The count attribute specifies exactly how many times the test will be run, incrementing the parameters each time. The repeat parameter specified how many times you want each step of the test to be rerun. The results will be averaged for the report. And finally, the timeout parameter allows you to specify a timeout in milliseconds, if a run method takes longer than this the test will be interrupted and the entire run will be stopped prematurely. The timeout attribute is optionally, and its absence means that no timeout will be used.

The <stress:parameter> tag allows you to specify how you want the runner to increment the parameter for each step of the run. A <stress:parameter> tag has three attributes: name, start and step. The name of the parameter refers to the specific parameter method within the stress test unit. The first character will be capitalized and the string set will be prepended to determine the method that the tag is referring to, e.g. parameter becomes setParameter. The start and step attributes specify how the runner should change the parameter for each step. The first step, the runner will set each parameter to the value in the start attribute for the parameter, and each subsequent step after that the value in step will be added to each parameter. Step is an attribute, and its absence means that the step value is 0.

Multiple runs can be used to vary each attribute in various ways, up to as many runs as needed by your tests.

Running the Test

Running a stress test is simple assuming you have a completed stress test unit and xml configuration file. Simply make a new instance of edu.wpi.ics.component.testing.stress.StressTestRunner and provide it the path to the xml configuration file in the constructor. Then call the runTests() method on the runner and it will run all your tests as specified in the xml configuration and produce the report.

```
public static void main(String[] args) {
    StressTestRunner runner = new StressTestRunner("stress/StressComplianceRows.xml");
    runner.runTests();
}
```