

Supplier Deliverable S3a

Technical Design Document

for the

ESS LOCKER REGISTRATION SYSTEM

by

Silver One Consultants


Contributors:


Heather Cape	V00795197	
Tanner Zinck	V00801442	
Geoffrey Lorne	V00802043	
Alex Neiman	V00800980	


Submitted March 18th 2016

Executive Summary

LARGE software has identified a need for the UVic Engineering Students Society to update their locker reservation system. They have contracted Silver One consultants to develop a new system to handle reservations for the 300 lockers managed by the ESS. This document outlines the functional specifications, the management plan, testing schedule, system design, and monitoring which, Silver One Consultants, will follow during the development of the Engineering Students Society Locker Management System.

The current system in use by the ESS is very dated.  original developer left the university many years ago, and there has been no maintenance done since their leaving. This is why we plan on implementing a new system from the ground up, with far higher usability, and maintainability than the current system. This will benefit anyone coming in contact with the system, whether it be administrators, future developers, or end users.

Together with LARGE software, Silver One consultants has identified some of the key features desirable for a new system. At the core will be the locker reservation system. This will allow current UVic students, and faculty to reserve a locker, using their @uvic.ca email address as a verification measure. The system will automatically send out reservation expiration notices, as a way to inform users when their reservation will need to be renewed or dropped . The site will be  build a mobile-friendly one. For ease of use a map of locker locations will be available, as well as a list of lockers that are currently open. And finally, the system will provide administrators with the access they need to run the locker system.

Main use cases have been identified; they include locker registration, and renewal, among others. To complement these use cases, we have included user scenarios, which are more focused on the user's actions. This document contains a management plan. This outlines some of the more technical information, such as the technologies planned for use, and the minimum system we can reliably deliver. The technologies planned for use w  be using are MySQL, Express.js, Node.js, and Angular.js.

This document also contains in depth information about each module in the system, how they interact with one another, and how data flows through the system. Pseudo code snippets have been written to provide more detail on how the imagined system will flesh out.

A rigorous testing plan has been put in place, including a testing schedule, testing methodologies, and individual tests. Thorough testing will occur at all stages of the systems lifetime. Testing will occur in phases, starting at the lowest level with Unit testing of individual functions. Then, once a particular module is completed, it will go through integration testing, to ensure that it will function correctly with the rest of the system. Once all the modules are complete and combined, functional testing will commence. This will allow users to work through use cases, and identify any glitches or errors. Finally, acceptance testing will be performed with both potential locker users, and ESS executive members, as a way to ensure that the delivered system is meeting the requirements of the client. This testing plan will allow Silverone Consultants to provide the Engineering Students Society with a high quality locker reservation system.

This document also includes a section on monitoring practices. This includes how the team will be managing themselves and each other. It highlights ways code can be written in an organized, timely manner.

Silver One Consultants have fully prepared for the development of a brand new Locker Reservation system for the UVic ESS, and will follow the functional specifications, and management plan in this document throughout the development process.

Glossary

ESS	Engineering Student Society - The elected body of engineering students who manage the locker system. This group of students changes each semester. The Director of Services manages the locker registration with help from the Director of IT
ELW	The Engineering Lab Wing building. The ESS manages lockers on the first floor of this building only
ECS	Engineering Computer Science building. The lockers in this building are NOT managed by the ESS but rather the Computer Science Union. Management for these lockers are not included in the application.
Renew	Renewing a locker means that the user already has a locker registered to their email. A user may register a locker for one semester, and then renew it for the next semester for a total of two semesters in the same locker.
Re-register	When a user has had a locker for two consecutive semesters, they are not allowed to keep the same locker by the renewal process. They will have to register for a new locker once the system gets reset at the beginning of the semester.
Email Verification	Upon registering for a locker, an email will be sent to the user's account with a link. If they click the link their email will be verified. This ensures that users are using real emails to register lockers.
reCaptcha	A tool to ensure that those registering for lockers are humans and not programmed bots. For security reasons, a user may be asked to answer a question or check a box to prove they are a real user.
TLS	(Transport Layer Security) is the standard security technology for establishing an encrypted link between a web server and a browser. This link ensures that all data passed between the web server and browsers remain private.
mySQL	an open-source relational database management system (RDBMS);

PHP	a server-side scripting language designed for web development but also used as a general-purpose programming language
HTML	HyperText Markup Language , commonly referred to as HTML, is the standard markup language used to create web pages.

Table of Contents

Technical Design Document

Executive Summary

Glossary

Table of Contents

1.0 Functional Specifications

1.1 System Description

1.2 Current System

1.3 User groups

1.4 Important Features

1.5 Hardware

1.6 Non Functional Requirements

1.6.1 Performance

1.6.2 Security

1.6.3 Usability

1.6.4 Maintainability

1.6.5 Reliability

1.7 Ethics

1.7.1 User Privacy

1.7.2 Fair Locker Allocation

2.0 User Interaction

2.1 Wire Frames

2.1.1 Home Page:

2.1.2 Home Page With Locker Availability:

2.1.3 Administrator Contact Page:

2.1.4 Administrator View:

2.1.5 Administrator Ticket View:

2.2 Use Cases

2.2.1 User Registers for a Locker

2.2.2 User Registers for a Group of Lockers

2.2.3 User Renews a Locker Registration

2.2.4 Administration Responds to Support Tickets

2.2.5 Administrator Manually Changes Locker Information

2.3 User Scenarios

2.3.1 Scenario 1:

2.3.2 Scenario 2:

2.3.3 Scenario 3:

3.0 Technical Design

3.1 Features and Modules

3.1.1 Programatically Specific Features

3.1.1.1 Locker Registration

3.1.1.1.1 Pseudo Code for Registering

3.1.1.2 Email Verification

3.1.1.3 Reservation Expiration Notification

3.1.1.3.1 Pseudo code for Email

3.1.2.4 Administrator Tools

3.1.2.4.1 Pseudo Code for Admin Ticketing

3.1.2.5 Overall System Interaction

3.1.2.6 Sequence Diagram

3.1.2 Other Features

3.1.2.1 Show Locker Map And Availability

3.1.2.2 Concurrent Registration

3.1.2.3 Mobile Friendly

3.1.2.4 Login Security and Automation Prevention

3.1.3 Abandoned Features

3.2 Implementation

3.3 Minimal System Goal

3.3.1 Captcha

3.3.2 Ticket System

3.3.3 Available Lockers Interface

4.0 Testing Plan

4.1 Introduction

4.1.1 Success Criteria

4.2 Integration Plan

4.3 Testing and Evaluation

4.3.1 Methodologies

4.3.2 Functional Tests

4.3.2.1 Registration Tests

4.3.2.2 Ticketing Tests

4.3.2.3 Admin Tests

4.3.3 Unit Tests

4.3.4 Integration Testing

4.3.5 Acceptance Testing

4.3.6 Performance Evaluation

4.4 Scheduling

4.4.1 Staying on Schedule

4.4.2 Responsibilities

4.5 Monitoring

4.5.1 Reporting Lack of Compliance

4.5.2 Reporting and Correcting Bugs

4.6 Discussion

5.0 Concluding Summary

1.0 Functional Specifications

The following section outlines the details of the project, including the potential users and possible features.

1.1 System Description

The Engineering Student Society (ESS) manages locker reservation for the 300 lockers in the Engineering Lab Wing (ELW) building. There is an existing system that, because of its rushed development and the confusion it causes users, is in need of an overhaul. With over 1600 students in the Faculty of Engineering, there needs to be a way to handle the lockers that is easy for the ESS executive to manage, and friendly for the students to use. This new project will be known from now on as the “ESS Locker Registration” system.

1.2 Current System

The current system that the ESS uses will be referred to as the legacy system. It runs on the ESS's unix server which will be used to host the new system as well. The legacy system uses the LAMP (Linux operating system, Apache server, MySQL database, PHP programming language) technology stack for its development. More details and shortfalls of the legacy system will be discussed as appropriate regarding the desired features for the new system. In short, the new system is confusing for students and administrators (leading to mistakes in the registrations process), allows for students to hog lockers for long periods of time or use multiple lockers, and has security flaws.

1.3 User groups

The user group of the ESS Locker Registration System is a well defined one; any person wishing to securely store their belongings in an ELW locker. To expand on this, students, professors, and TAs are all welcome to use the lockers. Students do not have to be in engineering (however most

are) and should be taking classes at the time (ie. not on co-op) --although this is hard to police and is left up to the honor system. Each person is allowed only one locker due to the demand. Specialty groups and clubs are allowed to register multiple lockers. These rules have been established by the ESS, not LARGE software or Silver One Consultants.

The new locker system will have more strict guidelines and instructions on who is allowed to register along with for how long. lockers are in high demand, by enforcing the one locker per person rule, and putting in restrictions on how long someone can hold a locker, it will free up more lockers for students to use. More details on these features and constraints can be found in the following sections.

A separate user group is the administrators, which will be elected directors of the ESS (typically director of services and director of IT). They will be able to access the system with a username and password and will have privileges that allow them to see and edit all the lockers and their registrants.

1.4 Important Features

The following is a list of desired features from LARGE software. More detailed information can be found in the Management Plan section of this document.

- Locker Reservation -The primary objective for this project is to allow students to reserve lockers within the Engineering Lab Wing through an online application requiring minimal to no admin assistance. No accounts should be used for registration, but instead @uvic.ca emails will be used. For groups wishing to register more than one locker, instruction will be available on the website to contact the administrator.
- Email Verification- When a user enters their email to verify a locker, they should be sent an email to ensure that their email is real and not misspelled.
- Reservation Expiration Notification -Locker owners should be sent an email with two weeks left in the semester to either renew or vacate their locker. Another reminder should be sent with one week left. After that period a list should be provided to the administrator for locks that need to be cut from lockers that were not renewed. These lockers will appear as available to register online.

- Responsive Design- The system must have a responsive interface and must work well with both desktop and mobile browsers.
- Show Locker Map And Availability -Along with a list of open locker numbers, there should be a map within the system, showing locker locations, and locker availabilities.
- Administrator Tools -An admin should be able to see and edit locker registration information, perform various batch actions, and check and respond to support tickets in a login secured page that is accessible only through knowledge of the URL (ie. no links or buttons available).

1.5 Hardware

The ESS Locker Registration system will run off of the current ESS Unix server. No other specified hardware will be needed.

1.6 Non Functional Requirements

This section of the document outlines all non functional requirements for the ESS locker registration system.

1.6.1 Performance

For time goals, the system should take less than 1 second to load any page to the user (administrator page might take longer to load all locker data) and at most 3 seconds under stress. The emails should arrive to the user within 5 minutes of them registering. It is not anticipated that there will be any times of very high traffic so a conservative estimate is that the loading times will remain low for up to 50 concurrent users.

1.6.2 Security

For security, LARGE software has suggested to use “bot” detection software when registering lockers. Also, all communications which includes sensitive information must be encrypted with

the latest standard of the TLS protocol. LARGE software has suggested the use of “Let’s Encrypt” because they offer free certificate signing.

1.6.3 Usability

The new ESS locker reservation system needs to improve upon the usability of the current system. Users should be able to register for a locker without any major hangups in the process. This can be tested by allowing a group of potential users to register for lockers, while monitoring the process, ensuring there are no usability issues. The admin page should also be more user friendly, and can be tested in a similar manner.

1.6.4 Maintainability

One of the main reasons that the old ESS locker reservation system needs to be replaced is because it is essentially unmaintainable. The original creator left the university a long time ago, and no one knows how the system works. The new ESS locker reservation system must be highly maintainable. Our goal for maintainability of the new system is for any competent software developer to be able to comprehend the inner workings of the system, and be able to make changes, updates, and add new features as necessary. This will be accomplished by maintaining use of coding best practices, avoiding code smells, and maintaining thorough system documentation.

1.6.5 Reliability

The ESS locker reservation system needs to achieve 99 percent uptime over a 24 hour period. Outages for maintenance should be conducted in a way that has minimal effect on users . This will ensure that users are able to reserve a locker when they need to, and administrators will have no problems managing the system.

1.7 Ethics

This section outlines all ethics requirements that have been set out for the ESS locker registration system. These ethics requirements have been defined in the best interest of the public, and user base.

1.7.1 User Privacy

Users of the system will be submitting some personal information, such as their name and UVic email. Because of this, it is very important that we take ethics requirements into consideration. The administrator view contains information for all user and thus should be only accessible by specific ESS executives such as Director of IT, Director of Services, and President. User information should remain private and not be used maliciously in any way.

1.7.2 Fair Locker Allocation

In regards to allocation of the lockers, the system should be as fair as possible. We believe that a first-come first-serve system will be the most fair without becoming overly complicated. Users will all have an equal chance to obtain a locker, if they wish to have one. Another aspect of fair locker allocation is reservation length. We have declared that all users must re-reserve after every semester, and users must re-register after two consecutive semesters in the same locker. This should prevent one user from holding the same locker for an absurd amount of time (this is currently a problem the ESS has). Additionally, administrators must act ethically in their administration of the system. Given that they have elevated access to the inner workings of the system, it is possible that they could allocate any locker they want to to themselves, or a friend. This is highly unethical, and it is the administrator's responsibility to not act unethically. A balance must be struck between giving groups enough lockers while still having enough to give to students. ESS executives will have to allocate lockers to groups in such a way that there is still a fair chance for individual users to reserve a locker. All of these ethical requirements will ensure fair locker allocation within the ESS locker reservation system.

2.0 User Interaction

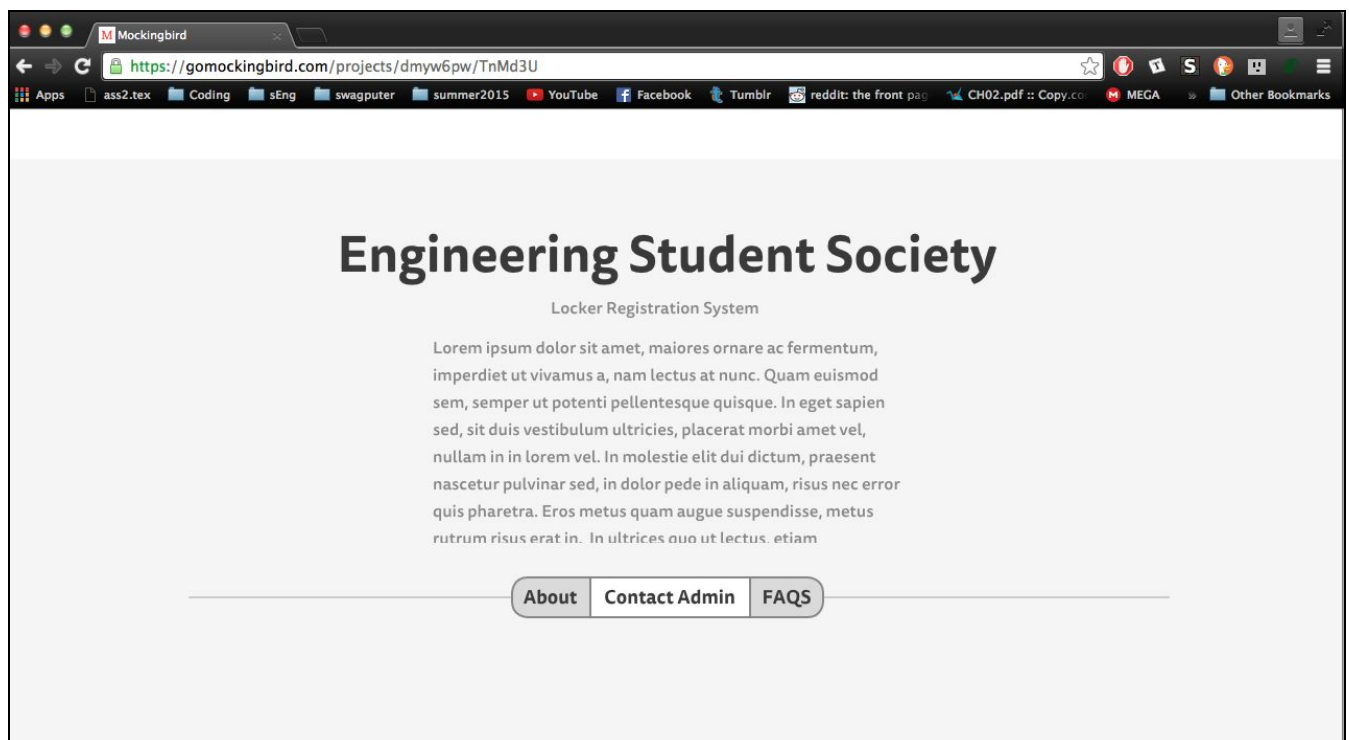
The following section will go in depth into how the user and system will behave in certain situations. Presented are a series of use cases that cover the system's main functionality, dialogues of realistic scenarios, and a glossary for users.

2.1 Wire Frames

This section contains wireframes that have been made up to represent how the finished system could look, from a users perspective. The wireframes chosen are the ones pertaining to the use cases listed in section 2.2 of this document.

2.1.1 Home Page:

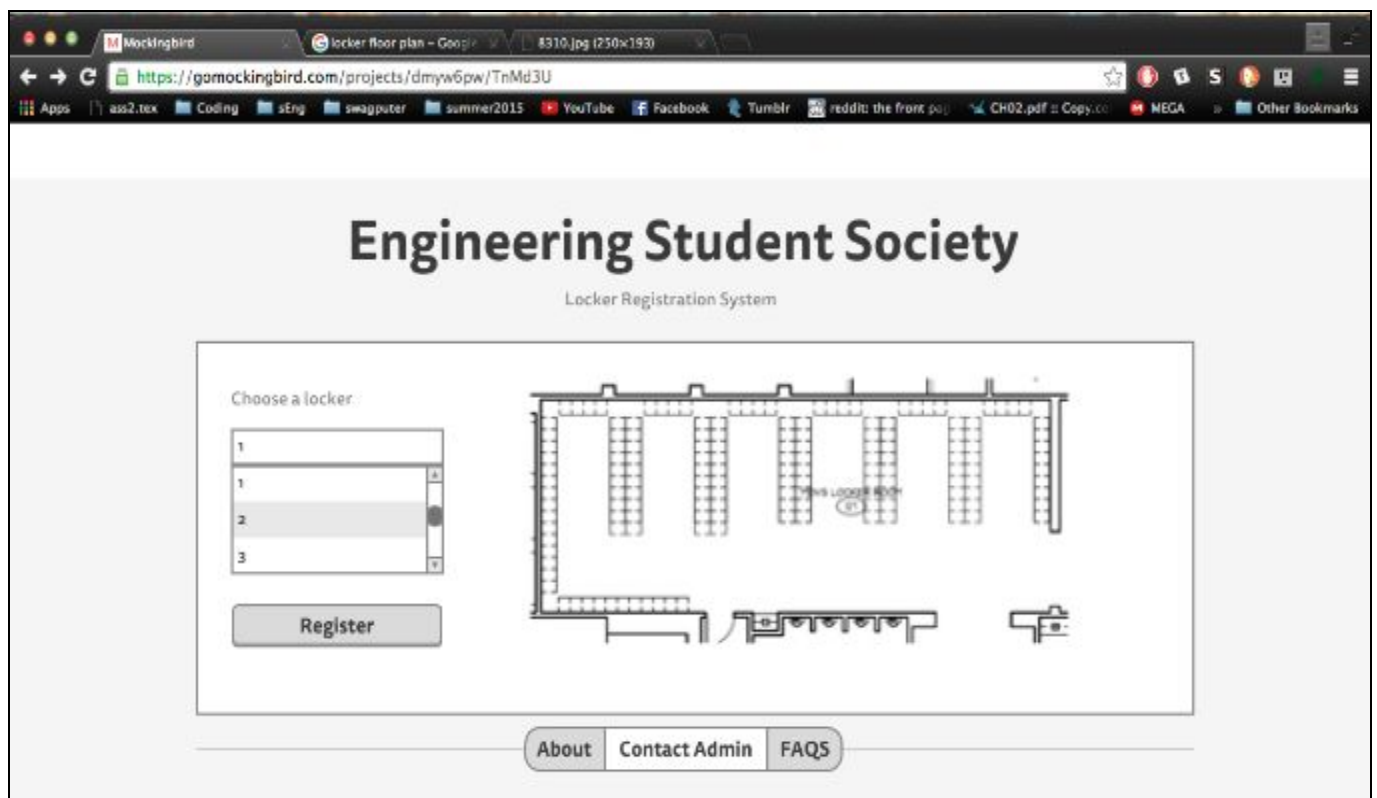
This is a depiction of what the home page of the system would look like if there were no lockers currently available. The ESS logo will be shown, as well as information about the lack of locker availability. Users will be encouraged to check back later, and the date of the next locker expiry date will be listed so that users will have an expected time for new open lockers.



2.1.2 Home Page With Locker Availability:

This shot shows what the main page of the system would look like if there are lockers available for registration. The locker map is available, and the functionality allowing registration is easily accessible. There is a drop down menu for selecting the locker number. As most lockers are taken at any given time, lockers on the map will show up in green if they are available, and not show up otherwise. There is also the possibility of showing out of order lockers in yellow. Sometimes a locker may be unavailable for registration, so there will not be a lock on it. This could confuse users if they cannot register for what they believe to be an open locker. That is why the yellow category will be introduced.

Not pictured is the box to fill in the user's email and name. It is underneath the drop down in the wireframe. These must be filled in for the user to successfully register a locker.



Below is the confirmation popup window that will appear if the user has selected an available locker, and entered a correct email and name. This popup will give the user a brief rundown of the verification processes, along with other important locker registering details. This information will be a shortened version of that available on the about page of the website.


There will also be small windows (not pictured) if the user enters an email that is already used, or enters an email that is not a UVic email.



The image shows a confirmation popup window titled "Locker #1". It features a text input field labeled "Email Address" with a light blue border. Below the input field is a paragraph of placeholder text: "Lorem ipsum dolor sit amet, maiores ornare ac fermentum, imperdiet ut vivamus a, nam lectus at nunc. Quam euismod sem, semper ut potenti". At the bottom of the popup is a rounded rectangular button labeled "Register". The entire popup is centered within a light gray rectangular frame.

2.1.3 Administrator Contact Page:

This is the page that users could use to contact the locker system administrators, given that they need special arrangements (for a club, etc.). This page is accessible from the home page by clicking “Contact Admin”.



The screenshot shows a web browser window with the URL <https://gomockingbird.com/projects/dmyw6pw/vZblI6>. The page title is "Engineering Student Society" with the subtitle "Locker Registration System". The form contains three input fields: "Name", "Email", and "Locker # (if applicable)". Below these is a large text area labeled "Enter message here". A "Send" button is located at the bottom right of the form. At the bottom of the page, there are three navigation buttons: "About", "Contact Admin", and "FAQS".

Engineering Student Society
Locker Registration System

Name Email Locker # (if applicable)

Enter message here

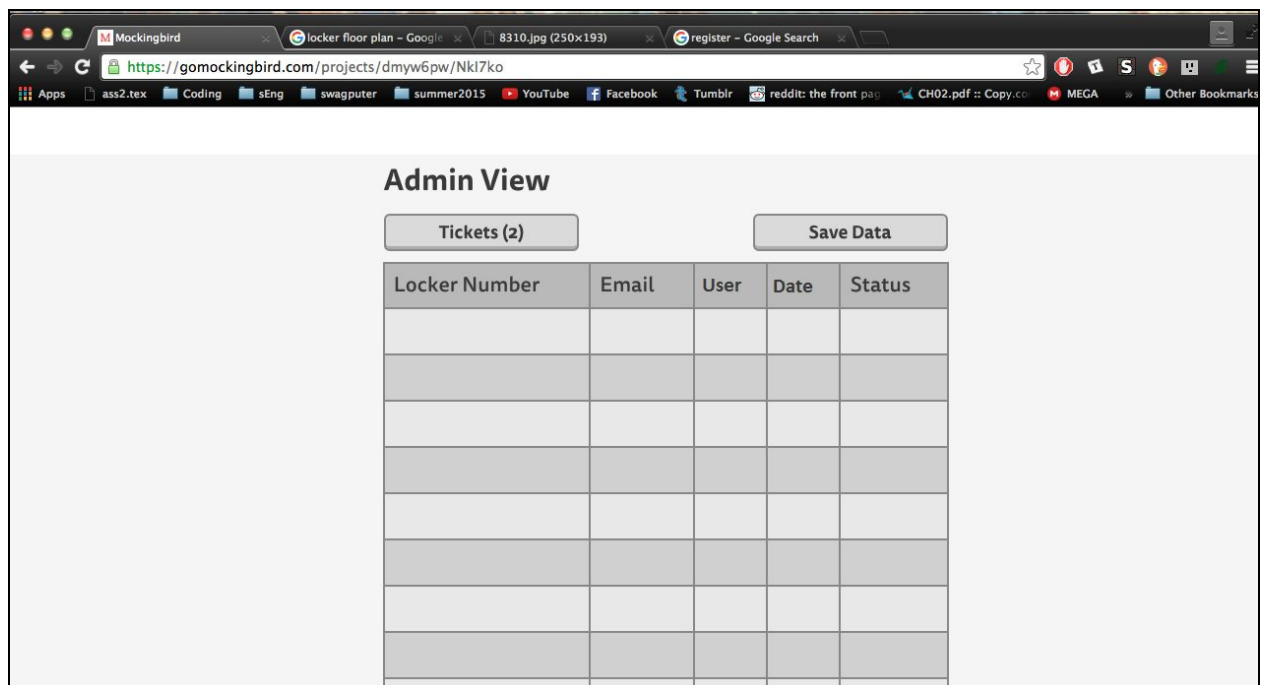
Send

About Contact Admin FAQS

2.1.4 Administrator View:

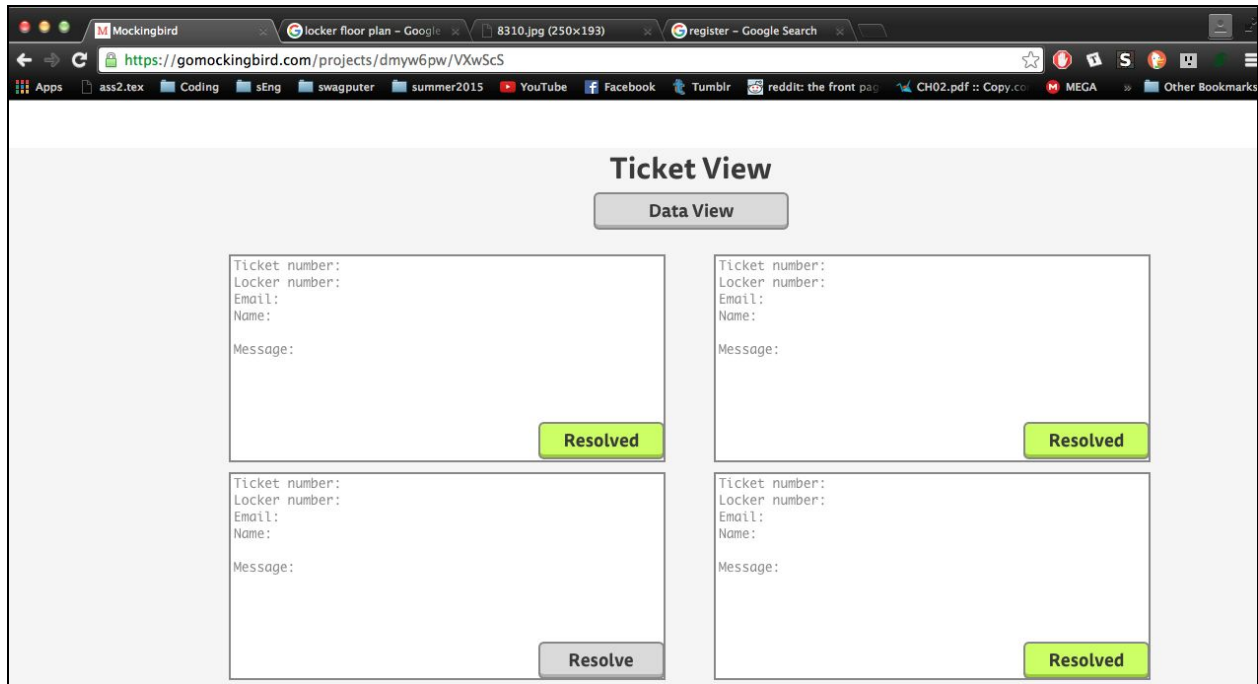
This is the view an administrator would have after logging into the system, using the admin credentials. This view would be essentially a view of the database. The admin will be able to click on and edit any of the cells, and then save the table with the “Save Data” button. Some of the cells will be text fields, and some will be drop down menus. The locker number field cannot be changed, as it is how the database is indexed.

This view is the default view when accessing the admin page of the locker system. The admin page is accessed only by knowing the correct URL (there is no button available to users to access the admin page). When the URL is navigated to in the web browser, the admin will be prompted to enter login credentials before proceeding to the admin tools.



2.1.5 Administrator Ticket View:

The first image shows the administrator's view of all the currently open, and recently resolved problem tickets in the system. The admin can scroll the page to view all previous tickets. The admin can return to the grid of data by clicking “Data View”. When the admin clicks on the “Resolve” it will take them to the individual ticket page.



The second image shows what an administrator would see when viewing a specific ticket. On the left side they can view all the information the user has submitted. As well as a transcript of further emails relating to this ticket. If an admin emails a user in response to this ticket, that email transcript will be available in the left window. In the right email, the admin can draft a response to the ticket. They can add additional information in the bottom right box, this includes things like “I registered lockers 76 and 77 for the Rocket Club”; this will be available in the left hand side for future admins to see. When the admin clicks save, a pop up view will appear. The pop up view will tell the admin “Click to send your email/ save your info/ mark as resolved” depending on the actions they took on the ticket.

Back

Ticket

Ticket number:
Locker number:
Email:
Name:
Message:

Reply to User

To: email@user.com
Message:

Additional Information

Will not be sent to user

☐ Mark as resolved

Save

2.2 Use Cases

This section covers possible use cases for the ESS Locker Registration System.

The use cases discussed are:

1. User registers for a locker
2. User registers a group of lockers
3. User renews a locker registration
4. Admin responds to a support ticket
5. Administrator manually changes locker information

The following templates will be used for the use cases:

ID	Number of the use case
Description	Task/behaviour captured in the use case
Actors	Who is performing the actions
Preconditions	Conditions that must be true before steps are taken to complete the task
Basic Steps	Steps the actor takes to complete the task
Alternate Steps	Possible alternative steps the actor could take to achieve the same task. This also includes steps the actor might have to take if there are alternative situations that end the use case.
Business validations/Rules	Rules about the system that must be followed to complete the task
Postconditions	What happens as a result of the actor completing the task

2.2.1 User Registers for a Locker

ID	1
Description	User registers for a locker
Actors	User
Preconditions	<ul style="list-style-type: none">-The user must be on a device that can access the Locker Registration system page.-There must be at least one open locker to view the registration page.
Basic Steps	<ol style="list-style-type: none">1. The user chooses a locker number to reserve, and their @uvic.ca email address.2. The user completes the reCaptcha to confirm they are human, then submits the form.3. The system sends a confirmation email to the provided @uvic.ca address.4. The user clicks the confirmation link in the email.5. The system allocates that locker to the user for the rest of the term.
Alternate Steps	<ol style="list-style-type: none">1. The email address provided is not an @uvic.ca address:<ul style="list-style-type: none">- The system does not send the confirmation email, and the locker is not registered. An error message is displayed to the user to inform them of the uvic email requirement.2. The user does not click the link in the confirmation email:<ul style="list-style-type: none">- The locker is not registered, and is made available for registration by others.3. There are no available lockers:<ul style="list-style-type: none">- A message is displayed to the user letting them know that all lockers are reserved.
Business validations/Rules	<ul style="list-style-type: none">-The user must be a current UVic student or staff, with a valid @uvic.ca email address.-The user can only reserve one locker at a time. (Exceptions for Science Venture / first year design etc.)

	-Reservation lasts for one semester and then must be renewed. -Reservation for the same locker lasts for two semesters then the user must re-register a locker
Postconditions	The user registered for a locker until the end of the current term.

2.2.2 User Registers for a Group of Lockers

ID	2
Description	User registers for a group of lockers
Actors	User
Preconditions	-The user must be on a device that can access the Locker Registration system page. -There must be at least one open locker to view the registration page
Basic Steps	1. The user clicks the Administrator Contact option and writes up a ticket including group name and desired number of lockers. 2. The user then pushes send, creating the ticket, which administrators can then view, and handle accordingly.
Alternate Steps	N/A
Business validations/Rules	N/A
Postconditions	-The admin page will display the ticket. -The admin will confirm the validity of the group (outside the system) -The admin will manually register the set of lockers for the user

2.2.3 User Renews a Locker Registration

ID	3
Description	User renews a locker registration
Actors	User
Preconditions	User must have a locker currently registered
Basic Steps	<ol style="list-style-type: none">1. The user receives an email from the system notifying them that their current locker registration is coming to an end.2. The user clicks on a renewal link in the email.3. The user is taken to a page on the registration site, where they click a reCaptcha, and confirm the locker renewal.
Alternate Steps	<ol style="list-style-type: none">1. The user does not click the renewal link:<ul style="list-style-type: none">- They do not renew the locker, and the locker can be booked by other users.2. If the reCaptcha fails:<ul style="list-style-type: none">- The locker registration is not renewed.
Business validations/Rules	<ul style="list-style-type: none">- The user must still be a student during the term that they are renewing their locker.-The user can only renew their locker twice before having to register for a new locker.
Postconditions	The user is registered in the same locker for another term.

2.2.4 Administration Responds to Support Tickets

ID	4
Description	Administrator responds to support tickets
Actors	Administrator
Preconditions	Must be logged in as administrator.
Basic Steps	<ol style="list-style-type: none">1. The administrator navigates to the tickets page2. The administrator clicks “Respond”3. The administrator writes a any relevant information about the work they did to address the ticket4. The administrator indicates the ticket is complete by clicking the “Completed” button
Alternate Steps	-The administrator can also reply to user emails, then mark the ticket as complete
Business validations/Rules	N/A
Postconditions	The ticket will be marked as complete and the information will be available to view by other administrators

2.2.5 Administrator Manually Changes Locker Information

ID	5
Description	Administrator manually changes locker information
Actors	Administrator
Preconditions	Must be logged in as administrator.
Basic Steps	<ol style="list-style-type: none">1. The administrator can click the fields to edit them (email, vacant or registered, etc.)2. The administrator clicks “Update”
Alternate Steps	N/A
Business validations/Rules	N/A
Postconditions	<ul style="list-style-type: none">-The database information is updated.-The admin receives visual feedback that the update has taken place.

2.3 User Scenarios

This section includes several sample transcripts of the interaction with the system in the form of a dialogue between the client and system.

2.3.1 Scenario 1:

The client is a student who would like to book a locker in the ELW.

Client: “I would sure like one of these lockers in the ELW!”

Client: Sees sign on the wall indicating locker registration is done via the ESS website.

Client: Navigates to the ESS locker registration page and reads instructions

Client: Selects a locker number to reserve from the available list.

System: Prompts user for their credentials (@uvic.ca email) and a reCaptcha

Client: Enters information and submits form and reCaptcha

System: Confirms the email ends in @uvic.ca and is unused

System: Sends confirmation email to the provided @uvic.ca email and temporarily marks the locker as reserved.

Client: Clicks on confirmation link in email, subsequently booking that locker.

System: Updates database accordingly.

Client: Claims their locker with a lock.

2.3.2 Scenario 2:

The client is a student who has had their locker for one semester and would like to renew their locker booking after the end of a term.

System: Sends email notifying client@uvic.ca that their locker reservation is expiring in 2 weeks.

Client: Does not see the first email

System: Sends another email notifying client that their locker reservation is expiring in 1 week.

Client: Sees the second notification email, and clicks the link provided to renew their booking.

System: Gives client a page confirming the renewal, with a reCaptcha to confirm they are not a robot.

Client: Clicks the reCaptcha, and books their locker for another term (4 months).

2.3.3 Scenario 3:

The client cancels their locker reservation after the end of the term.

System: Sends email notifying client@uvic.ca that their locker reservation is expiring in 2 weeks.

System: Sends another email notifying the client that their locker reservation is expiring in 1 week.

Client: Sees these emails notifying of locker booking expiration, however, since the client is going on a co-op term, they have no need for their locker, and do not click the renewal link in the email.

System: Frees up the locker that was previously allocated to the client, and makes it available for booking by other users.

Administrator: Checks to see if the client left a lock on the locker, removes the lock from the locker if it is left there.

3.0 Technical Design

3.1 Features and Modules

This section lists the features of the ESS Locker Registration System that have already been introduced, in greater detail. There are three types of features, those that can be described with class diagrams, those that are less specific and can be discussed in general terms, and those which are being removed from the possibility of development.

Pseudo code will be written for each module task. Some information about our pseudo code:

- It will be denoted by `this font`
- Comments to explain functionality will be denoted with the `#` symbol

3.1.1 Programatically Specific Features

The first group of features are those that can be described with class diagrams. The classes, methods, variables, relations are described for each picture. It is important to note that the application will be developed with model-view-controller architecture. This will be expanded on in section 3.2.

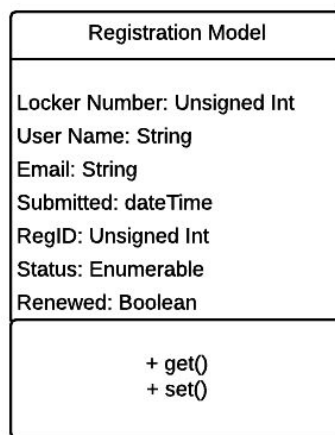
3.1.1.1 Locker Registration

The ESS Locker Registration system will provide the backbone upon which the lockers are managed. The main function of the system will be the ability to register for a locker. Students, staff and faculty can all register an open locker at the start of any term. To do this, they must put a lock onto an open locker, take note of the locker number, and then register that locker online. Users are limited to one locker per person. In place of accounts, users will be verified using their @uvic.ca email address. During the registration process, users will enter their name, the locker number they wish to register, and their UVic email address. Using a confirmation system tied to

that email address, we can confirm identities of users, and ensure no one reserves more than one locker at a time.

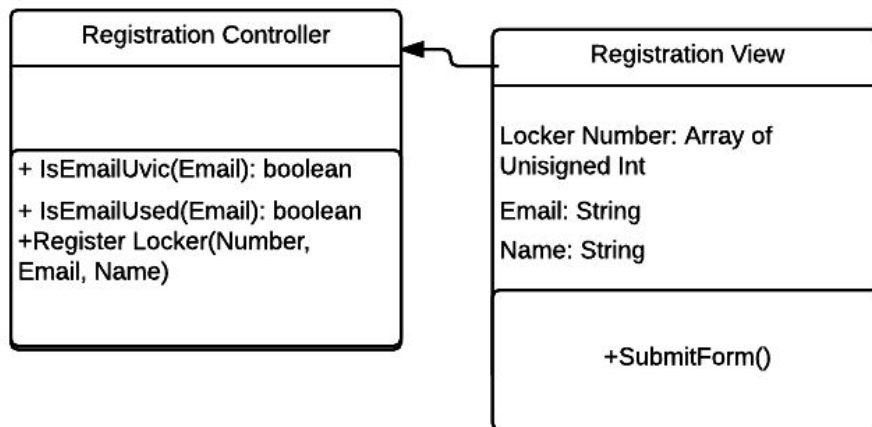
For large groups who may wish to register for more than one locker (ie Science Venture, etc.), there will be a note on the website to contact the administrators, who will then manually sort out lockers for them.

We plan to build on the legacy system database model and may add several new fields. The proposed schema is shown in the figure below.



The values in the model are quite self explanatory. The **Status** element can be unregistered, registered, special reservation, and awaiting renewal. Currently the **Status** is changed to awaiting renewal for all lockers when an admin decides to send out the renewal emails, this will be automatic in the new system. **Renewed** will be a new field added to the database and will take values of 0 - never renewed or 1 - renewed once.

On the registration view (the main page) there will be a list of **lockers**, and the submission form will have room for an **email** and **name**. The controller will check if the email ends in @uvic.ca, it will query the database to check if the email has been used before, and finally it will go through with the registration process. This involves setting the values in the database, and sending the verification email.



3.1.1.1.1 Pseudo Code for Registering

#To get a list of open lockers to list on the main page

```

for all entities in RegistrationModel:
    RegistrationModel.get(LockerNumber)
    if Status of LockerNumber is "open":
        add LockerNumber to list of open LockerNumbers
End.
  
```

#User is registering

When SubmitForm() is called:

#Register button is pressed on homepage

```

    if LockerNumber is not selected or Email is not filled
    or Name is not filled:
        Tell user they are missing information
    End.
  
```

```

    if RegistrationController.IsEmailUsed(Email) is false:
  
```

#Blanks are any characters

```

        Tell user their email must be a uvic email
  
```

End.

```

if RegistrationController.IsEmailUsed(Email) is false:
    Tell user their email is already being used
    #This includes checking for uvic.ca and csc.uvic.ca
    duplicates
    End.
RegisterController.RegisterLocker(LockerNumber, Email, Name)
End.

```

RegisterLocker(LockerNumber, Email, Name) Function:

```

RegistrationModel.set() sets status of LockerNumber =
RegistrationModel.get(LockerNumber) to 'registered'.

```

```

#this prevents another user from registering locker at the
same time

```

```

EmailController.SendEmail('Verification', EmailAddress)

```

When user clicks email verification link:

```

RegistrationModel.set() #all fields

```

If the user does not click link in 30 minutes:

```

RegistrationModel.set() sets status of LockerNumber =
RegistrationModel.get(LockerNumber) to 'unregistered'.

```

```

#this timeout will likely be built into the verification
link (ie it will expire after a certain time). This will
likely be handled by third party frameworks

```

End.

3.1.1.2 Email Verification

During the locker reservation process, users enter their email, for verification purposes. This is our way of ensuring that only valid users are registering for lockers, and that each user reserves at most one locker at a time. The verification process relies on the fact that every one of our potential users has a valid @uvic.ca email address. There will be certain cases where a user may

have an @uvic.ca aswell as an @ece.uvic.ca (or some department other than ece). We will account for this in the system.

3.1.1.3 Reservation Expiration Notification

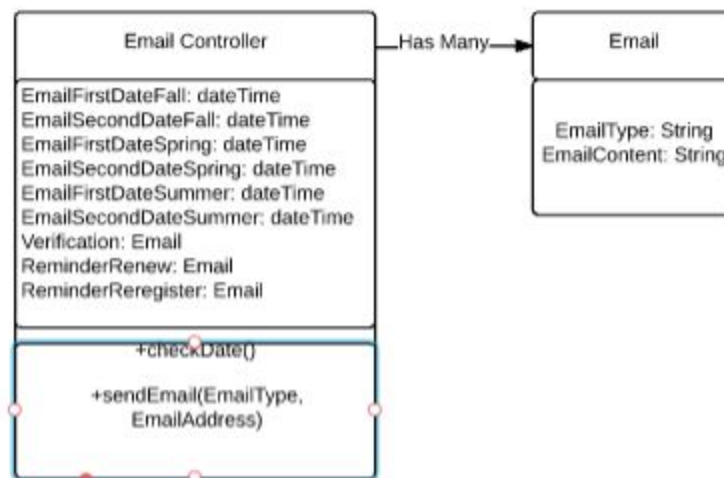
A five times when an email will need to be sent by the system:

- The initial verification email discussed above.
- The first reminder which either tells users:
 - They can renew their locker for another semester, and a renewal link.
 - They must vacate their locker by a certain date.
- The second reminder which either tells users:
 - They can renew their locker for another semester, and a renewal link.
 - They must vacate their locker by a certain date.

The three unique types of emails are:

- Verification
- Reminder - user can renew
- Reminder - user cannot renew

The system is described in the diagram below



There is an email class which contains the **type** and **content** of the email to be sent. Currently there are two types of emails 'Verification' which will only ever be sent to one user at a time, and 'Seasonal' which will always be sent to all users in the system.

There will be three unique email objects created in the Email Controller: **Verification**, **ReminderRenew**, **ReminderReregister**. If the email to be sent is a verification email, **sendEmail** will be called directly with only one email address in the **Emails** array. Otherwise the controller will send out the appropriate timely email depending on the result of **checkDate()** which compares the current date to those of the defined **email send out dates** each semester. Then **sendEmail** will be called with all **emails** of registered users from the database. It will send out a renew or re-register email depending on the value of the **Renew** boolean.

3.1.1.3.1 Pseudo code for Email

Each new day:

```
dateTime today <- EmailController.checkDate()

if today == EmailFirstDateFall:
    EmailController.sendEmail('Seasonal', null);
else if today == EmailSecondDateFall:
    EmailController.sendEmail('Seasonal', null);
else if today == EmailFirstDateSpring:
    EmailController.sendEmail('Seasonal', null);
else if today == EmailSecondDateSpring:
    EEmailController.sendEmail('Seasonal', null);
else if today == EmailFirstDateSummer:
    EmailController.sendEmail('Seasonal', null);
else if today == EmailSecondDateSummer:
    EmailController.sendEmail('Seasonal', null);
#all the Email objects in the function calls above will be
different at their content will be slightly different
End.
```

SendEmail(EmailType, EmailAddress) Function:

If EmailType is 'Seasonal':

```
#this email will be going to all students, so generate a
list of all emails
for all entities in RegistrationModel:
    ListofEmails <- RegistrationModel.get(Email),
```

```

RegistrationModel.get(Renew)

for each Email in ListofEmails:
    CanRenew <- Emails with renew value 0
    CannotRenew <- Emails with renew value 1

Send ReminderRenew Email to CanRenew addresses
Send ReminderReregister Email to CannotRenew addresses
End.

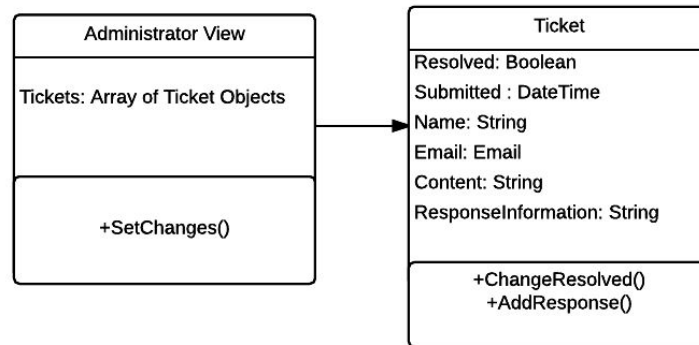
if EmailType is 'Verification':
    Generate Verification Link
    Verification.EmailContent + verification link
    #Add the link to the Verification object
    Send Email to EmailAddress
End.

if EmailType is 'Admin':
    Send Email to EmailAddress

```

3.1.2.4 Administrator Tools

The administrator views are basic, one is a table of all the information in the **registration model** that is retrieved via get calls and changed via set calls. The set function is called when the page is saved by the admin. The other is a view of the **Tickets**. Each ticket has fields for the user to fill out and allows the administrator to set it as **Resolved** and write any **Information** about the ticket.



3.1.2.4.1 Pseudo Code for Admin Ticketing

The admin ticket view get all the ticket data and displays them on a page.

If admin clicks "resolve":

Navigate to individual ticket view

#On the individual ticket view

If admin clicks "save":

Show popup giving the admin information (if they are sending an email, adding info, etc.)

If "Additional Info" is filled in:

Ticket.ResponseInformation <= added info

Ticket.AddResponse

If "Mark as Resolved" is checked:

Ticket.ChangeResolved

If Email to User is filled:

Create new Email

Email.Content <= admin's email

Email.EmailType <= "Admin"

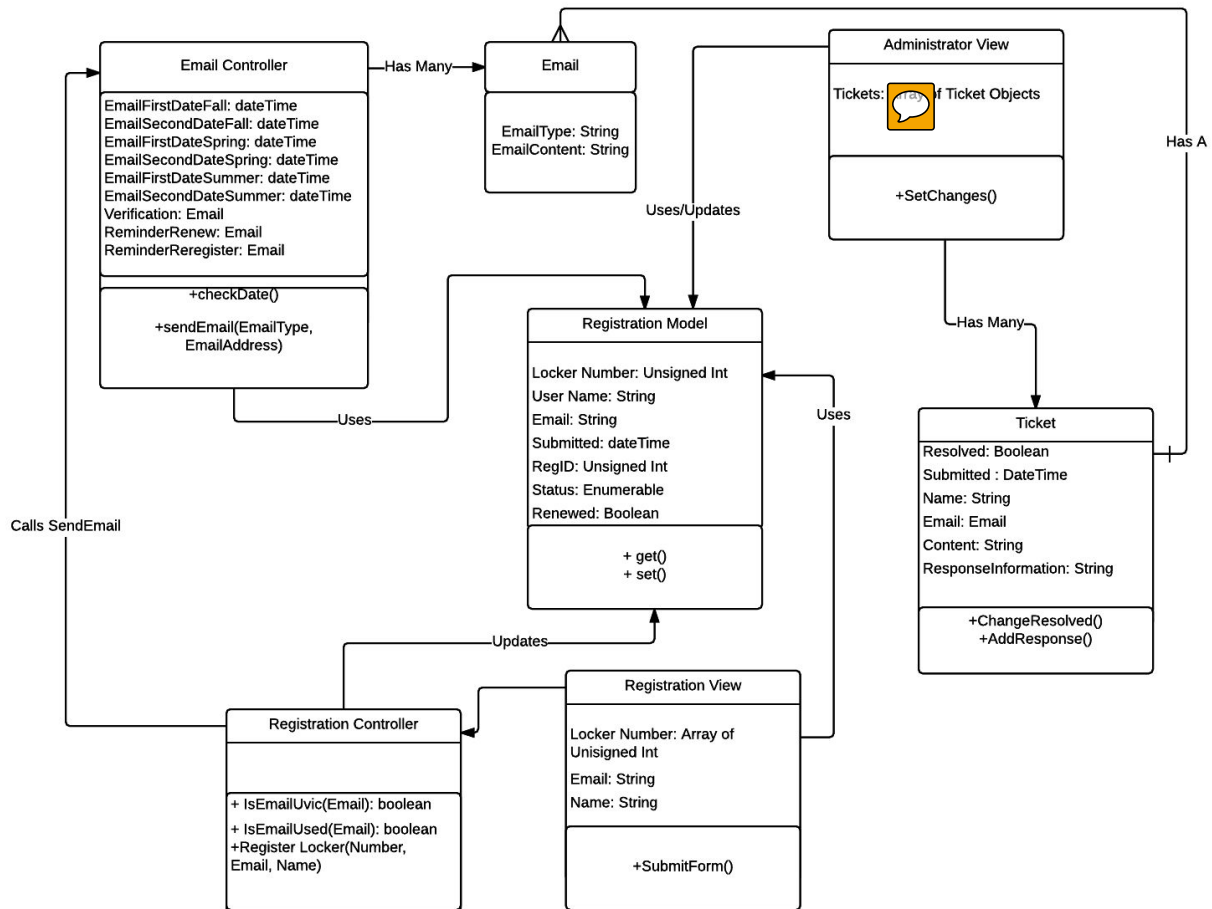
EmailController.SendEmail(new Email, Ticket.Email)

Ticket.Content.append(Email.Content)

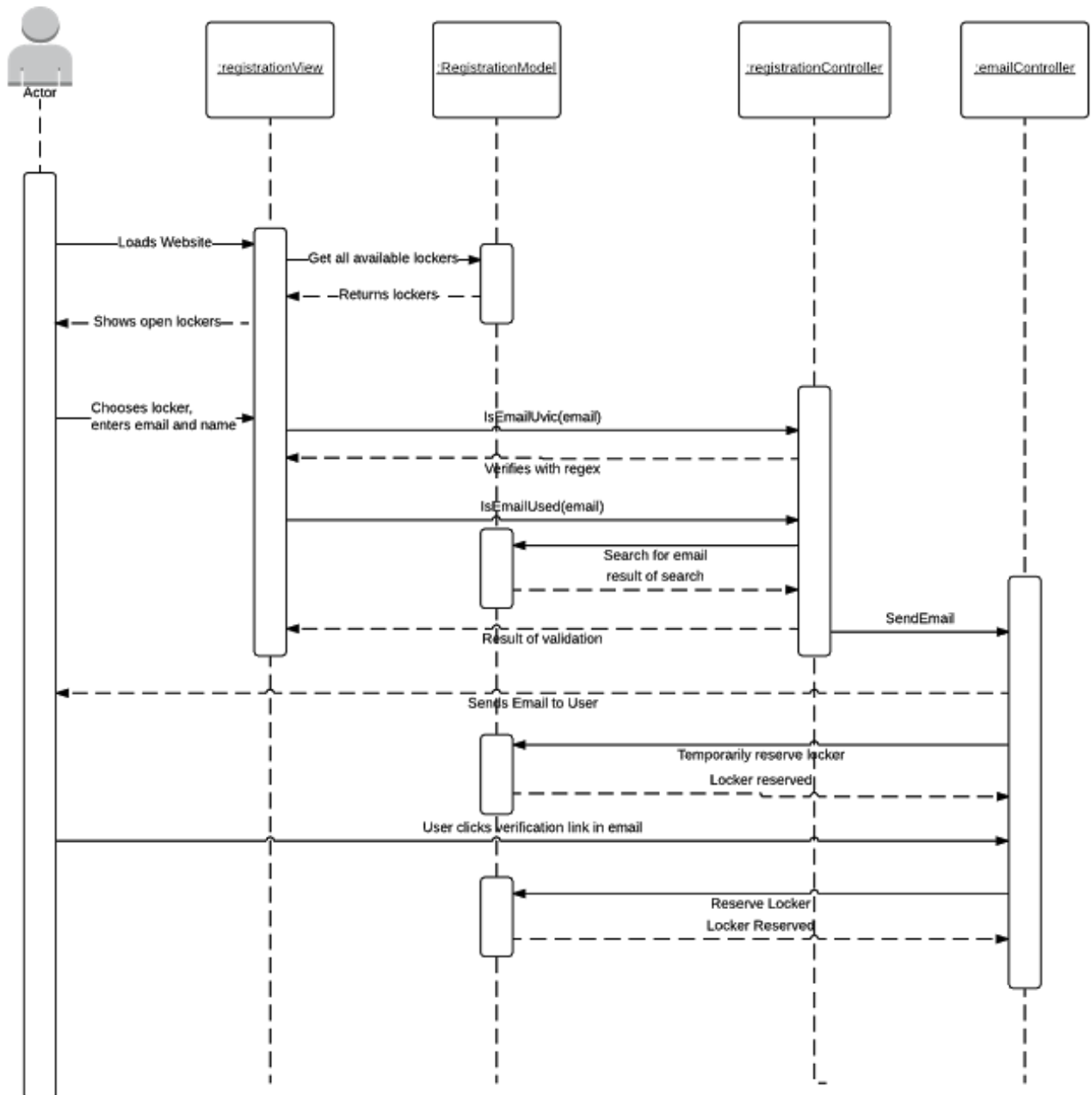
adds the admins email to the content of the ticket
to be viewable by future admins

3.1.2.5 Overall System Interaction

The following image is a pictorial view of the relationships discussed in the previous feature sections.



3.1.2.6 Sequence Diagram



The sequence diagram on the page above serve the purpose of highlighting the data and information flow in the system. This behavioural diagram demonstrates the processes of the system in the use case where a user registers a locker. It does not cover the alternative steps of a user entering a bad email, or not clicking the verification link in their email. This diagram does not show all of our modules interactions, but covers the main use case, and the interactions a regular user will have with the system

3.1.2 Other Features

These sections describe features that are desired, but do not fit well in a class diagram description.

3.1.2.1 Show Locker Map And Availability

LARGE software put forth their desire for a graphical map interface of the availability of lockers. This feature would make it easier to determine where the available lockers are. This features, if included, will likely use third-party software to help map the image to locker values and thus was not included in the class diagram.

3.1.2.2 Concurrent Registration

Due to the nature of the ESS Locker bookings, there could be the potential for concurrency issues. The system must be able to handle multiple user attempt to register for the same locker simultaneously. By checking if the locker is occupied upon submitting the form the backend can verify that the locker is still available and reply with an error message and the opportunity to register for a different locker. By temporarily marking the locker as occupied upon sending the verification email, no two users will be able to be in the process of registering for the same locker.

3.1.2.3 Mobile Friendly

Creating a mobile friendly system is something that is very important. Having a mobile friendly site will allow users to register a locker while they are standing next to it. It also allows more people to access the site if their primary on-the-go device is a phone instead of a laptop or tablet. Frontend frameworks such as Bootstrap can be used for cross platform development of a single site.

3.1.2.4 Login Security and Automation Prevention

In order to keep the system secure for both users and administrators, TLS encryption will be used on all communication containing sensitive data. This will keep administrator passwords safe, and maintain the integrity of the system as a whole.

In order to prevent bots from automatically registering for lockers, some form of automation prevention needs to be put in place. LARGE software has suggested we look into using reCaptcha as a way to ensure our users are in fact humans, and not bots. ReCaptcha is a free service that is used by many web applications similar to the one we will build. We are confident that this will prevent any users from automating locker registration, and giving themselves an unfair advantage regarding locker choice.

3.1.3 Abandoned Features

LARGE software put forth in their Request for Proposal the idea for users to be notified on locker availability. The idea was to have users sign up to be in a queue when there are no lockers available. They would then be notified in order when lockers open up. After discussion it has been decided not to include this feature. One reason is that very rarely do lockers open up in the middle of the semester. When lockers open up, it is in groups at the beginning of each semester. Furthermore, even if lockers did open up, how long should a user be given to respond once they have been sent their email notification? If 24 hours is used (similar to being waitlisted for a class) and there are dozens of people in the queue

3.2 Implementation

Our proposed implementation is to use the MEAN stack with a MySQL database instead of a MongoDB database. The rest of the stack is Express.js server framework, Angular.js for the frontend framework, and Node.js to run the application. We chose to use MySQL instead of MongoDB as reservation data is relatively simple and will easily fit within a predefined data table. The legacy system is already using a MySQL database which means transitioning from the old system to the new system will be easier if we continue to use MySQL. The ticketing system, if implemented, will use a separate MySQL table. We will use Sequelize for our server side scripts and for interacting with the database. The front end will be implemented using HTML, CSS, and Javascript which will be extended with AngularJS where possible. We will also be using Bootstrap, a front end framework, which will help us develop a mobile compatible site. The system will be hosted with heroku for development but will eventually be pushed onto the current server when the system is complete. The for initial prototyping we plan to use simulated data only.

3.3 Minimal System Goal

The bare minimum Silver One Consultants hopes complete this term will be to finish a registration system with the following functionalities:

- Select a locker from an available list and register with UVic email.
- Send a verification email which will update the database once clicking the verification link
- Basic administrator page with the ability to see and make changes to any registration. (The administrator ticket page is not planned to be included.)
- Bot-prevention through recaptcha
- Usable on mobile devices but may contain formatting or front end bugs.


The prototype will likely not be thoroughly tested by the end of the semester.

As we are not associated to the ESS it may be hard to get access to the database and server during prototyping stage. We plan to at first create a local database with a similar schema to the

real database to prototype with. The application will not be hosted on the ESS server, but locally or online in a temporary location.

None of the rest of the features are guaranteed. Some changes can be made to make the task easier to partially implement. These changes can be applied individually or together depending on the extent that the project has to be simplified.

3.3.1 Captcha

The reCaptcha can be removed from the locker registration page because it only prevents users from registering for a locker using an automated system. A user could abuse the system by attempting to register with many fake emails, overloading the system by having to send many verification emails to bad addresses. The system is an important one, yet there are no peak times or vital task that it is needed for; down time would only cause a  inconvenience. However including a reCaptcha on the page is a reasonably simple task, and would not push back the finishing date significantly.

3.3.2 Ticket System

The ticketing system is one of the larger “nice-to-have” sections of our system. Ticketing systems are extremely useful for organizing customer queries; however, with a system as small as this locker system a ticketing system is not required. Due to the small number of users and administrators, it is easy to replace the ticketing system with a simple customer service email. Tickets are beneficial addition because the ESS executive changes every semester and it would be nice to have a paper trail of past interactions. The ticket system is a module that could be added on at a later date.

3.3.3 Available Lockers Interface

There are a few options to add a visualization of the lockers to the site. A static map of the ELW could be added, which would show which lockers are in different sections of the building, and give the users the general location of available lockers. A more complicated option is to have a map which displays the availability of lockers in a certain area, where the different groups of lockers can change to display whether any lockers in that area are available. This would give users an easy way to check if there are lockers available in a certain area of the building though it

lacks the precision of a map which would show individual lockers. The minimum system will not include any graphical representation, but if there is time left one of these simpler approaches could be used.

4.0 Testing Plan

This portion of the document outlines the testing plan for the ESS locker reservation system. It covers our success criteria, integration plan, test methodologies, schedule, and monitoring plans.

4.1 Introduction

The objective for the testing of the ESS locker reservation system is to ensure that the system delivered is functioning according to requirements we have set by the clients and suppliers. This includes unit testing all of our modules in isolation, as well as integration testing for each module and how they interact with one another. Unit testing of each module will occur in isolation, so that we can easily analyze the functionality of only the module being tested. This will reduce the chance of error during unit testing. Integration testing will ensure that these individually developed modules will work together as intended. Functional testing will be implemented to test that the front end is interacting properly with the back end, then performance testing will be run to verify that the system is behaving properly. We are very confident that this rigorous testing structure will allow us to deliver the high quality system we are looking to produce.

Unit tests will be written along side each module. They will be tested as they are developed. When the development of each module begins to wrap up, integration tests will be written to help the modules connect.

Functional testing will be done as the frontend and backend of each section of the system are created. The functional tests will be frontend interactions which will test one or more integration tests. This will allow the team to assess that the frontend and backend are properly interacting before moving on to a different section of the system.

Performance testing is best done when full use cases can be run through. The performance of our site will be tested at the end of the integration tests. The decision behind this is that our

performance tests may look better at the beginning of development, but will not be a true representation of the sites performance. Integrating more modules could reduce the overall performance of the site.

The acceptance test is done last in a demo to the client on a set date. This test will evaluate the overall completion of the product relative to the desired specifications and requirements.

4.1.1 Success Criteria

For the ESS locker reservation system, our success criteria is based on how well we meet the functional requirements, the non-functional requirements, whether the system is delivered on time, and the overall usefulness of the delivered system to the University of Victoria Engineering Students Society. Most of the tests we write and perform will be pass or fail strictly. For example, if a functional black box test returns the wrong output to an input, it fails the test.

Performance and acceptance testing are a bit less black and white in their evaluations. Performance testing will be graded on speed and load tests. The performance tests could fail, but the overall system could still be functioning, it will just be at a slower rate than desired.

The final acceptance test will be an evaluation by our client based on their needs. We will demo the prototype and allow them to experiment as users. From there we will discuss our progress with our clients and determine what future milestones we have, as well as what we need to improve from the previous cycle.

4.2 Integration Plan

Given the size of the ESS locker registration system, both development and testing will be done in modules. These modules are as previously set out in section 3.1, and include the Registration Model, Registration Controller, Registration View, Email Controller, Administrator View, and Ticketing.

Modules will be developed in the following order:

1. Registration Model
2. Registration Controller
3. Registration View
4. Email Controller
5. Administrator View
6. Ticketing


This order allows us to finish the core functionality of the system (the registration modules) first, and then have them act as a backbone while we finish the rest of the system. The testing of modules will be done in the same order as developed. Testing and development will be somewhat concurrent, with testing of completed modules occurring at the same time as the development of the unfinished modules. This will ensure that all of our modules are functioning as they should be before we move on with development.

Integration of modules will occur after they have been tested. This will most likely occur in the same order as the above listed order of development.

4.3 Testing and Evaluation


In order to ensure correctness, and that all requirements are met, testing and validation must occur. This section of the document outlines the specific methodologies, as well as tests that will be used in the testing of the ESS locker reservation system.

4.3.1 Methodologies

For the testing of the ESS Locker Reservation system, we will be employing the use of multiple testing methodologies, to ensure that our system is fully tested, and functioning as desired. Unit tests written with Jasmine will be used to verify that functions are working as they should be. This is the first stage of testing, and serves as the foundation for our entire testing plan. Next, we will perform integration tests. These will ensure that completed modules all work together as they should. System testing will come after integration testing. **Functional (or System) testing** requires that the entire system be completed,  as it is a test performed by a user operating the system. Functional testing will be a series of use cases which the user must work through. These

cases will be well defined, outlining the preconditions, user inputs, and expected outputs, as well as any possible exceptions that could occur. Once these three testing phases are done, we will execute acceptance testing, where we introduce the ESS executives to the system. By walking through some use cases with ESS executives, we will be able to determine whether or not the delivered system is functioning according to requirements or not.

4.3.2 Functional Tests

Test Number	1
Description	Email Verification 
Preconditions	<ul style="list-style-type: none"> • User must have a uvic email address • There must be at least one available locker
Inputs	<ul style="list-style-type: none"> • Uvic email to be associated with the locker reservation • Locker reservation request
Expected Output	<ul style="list-style-type: none"> • Email is sent to corresponding Uvic email address with verification link to finalize locker registration • If the link is not clicked within a timeout period the locker remains available
Exceptions	<ul style="list-style-type: none"> • Users should be alerted if the preconditions are not met

Test Number	2
Description	Temporary Reservation During Email Verification
Preconditions	<ul style="list-style-type: none"> • Users must have two UVic email addresses • There must be at least one available locker

Inputs	<ul style="list-style-type: none"> • Two Uvic emails to be associated with the locker reservation • First user sends a locker reservation request • Second user attempts to reserve the same locker
Expected Output	<ul style="list-style-type: none"> • Second user cannot reserve the same locker for up 10 minutes • If locker is validated through email, second user may not register the locker after 10 minutes
Exceptions	<ul style="list-style-type: none"> • If user has the same locker selected, and their page has not been updated to reflect that the locker is “pending”, then the user is brought to an error page or dialog explaining the error

Test Number	3
Description	Landing Page
Preconditions	<ul style="list-style-type: none"> • There must be at least one available locker
Inputs	<ul style="list-style-type: none"> • Click on an available locker • Click on Register button • Email Address on register window
Expected Output	<ul style="list-style-type: none"> • Landing page shows all available locker • After user clicks register, a new window opens up requesting the user for an email address • Email is sent to corresponding Uvic email address with verification link to finalize locker registration
Exceptions	<ul style="list-style-type: none"> • If the user clicks on a already registered locker, nothing happens

Test Number	4
Description	Ticket View
Preconditions	<ul style="list-style-type: none"> • Users must have created a ticket through contact admin in the locker registration system
Inputs	<ul style="list-style-type: none"> • Email • Name • Locker Number (#If Applicable)
Expected Output	<ul style="list-style-type: none"> • A ticket is generated which is only viewable by an admin. • The ticket contains all relevant information • The generated ticket is in the not-resolved state
Exceptions	<ul style="list-style-type: none"> • If the ticket is not made the user should be alerted and told to retry in a moment

4.3.2.1 Registration Tests

Test Number	5
Description	Basic Registration
Preconditions	<ul style="list-style-type: none"> • None
Inputs	<ul style="list-style-type: none"> • Email Address • Locker Number
Expected Output	<ul style="list-style-type: none"> • If all the following conditions are true send verification email and temporarily reserve locker <ul style="list-style-type: none"> • Email is a valid UVic Email • Email is not used to reserve another locker • Locker is not occupied

	<ul style="list-style-type: none"> • If any conditions are false return error detailing which is false
Exceptions	

Test Number	6
Description	Email Verified
Preconditions	<ul style="list-style-type: none"> • Email has been sent to user for reserving a locker
Inputs	<ul style="list-style-type: none"> • Locker Number • User Email • Verification key
Expected Output	<ul style="list-style-type: none"> • If all the following conditions are true then reserve locker <ul style="list-style-type: none"> • Locker is not occupied by a different user • Verification key correct • Locker is temporarily reserved or registered to that email • If any conditions are false return error detailing which is false
Exceptions	

4.3.2.2 Ticketing Tests

Test Number	7
Description	Create ticket
Preconditions	<ul style="list-style-type: none"> • None

Inputs	<ul style="list-style-type: none"> • Email • Locker Number(optional) • Ticket Message
Expected Output	<ul style="list-style-type: none"> • If all the following conditions are true create ticket <ul style="list-style-type: none"> • Email is a valid UVic Email • Ticket Message is not empty • If any conditions are false return error detailing which is false
Exceptions	Email address is banned from creating tickets (Due to excessive fraudulent or joke tickets being created)

Test Number	8
Description	Reply to Ticket
Preconditions	<ul style="list-style-type: none"> • Ticket available
Inputs	<ul style="list-style-type: none"> • Ticket Object • Ticket Reply
Expected Output	<ul style="list-style-type: none"> • If all the following conditions are true send a reply to the ticket <ul style="list-style-type: none"> • Ticket Reply is not empty • Ticket Object email exists • If any conditions are false return error detailing which is false
Exceptions	<ul style="list-style-type: none"> • Request is not made with proper authorization

Test Number	9
Description	Edit Ticket

Preconditions	<ul style="list-style-type: none"> • Ticket available
Inputs	<ul style="list-style-type: none"> • Ticket Object • Action
Expected Output	<ul style="list-style-type: none"> • If all the following conditions are true resolve the ticket <ul style="list-style-type: none"> • Ticket Object is not resolved • Action is Resolve • If the conditions do not hold, return error detailing what condition was incorrect
Exceptions	<ul style="list-style-type: none"> • Request is not made with proper authorization

4.3.2.3 Admin Tests

Test Number	10
Description	Cancel locker registration
Preconditions	<ul style="list-style-type: none"> • Email has been sent to user for canceling a locker
Inputs	<ul style="list-style-type: none"> • Locker Number • User Email • Verification key
Expected Output	<ul style="list-style-type: none"> • If all the following conditions are true then reserve locker <ul style="list-style-type: none"> • Locker is occupied by the user with provided email • Verification key correct • If any conditions are false return error detailing which is false
Exceptions	User does not respond to cancellation email


	<ul style="list-style-type: none"> • The registration is not canceled
--	--

Test Number	11
Description	Admin Login
Preconditions	<ul style="list-style-type: none"> • Admin account available
Inputs	<ul style="list-style-type: none"> • Admin Email • Admin Password
Expected Output	<ul style="list-style-type: none"> • If all the following conditions are true return a temporary admin verification token <ul style="list-style-type: none"> • Admin Email and Password match an Admin Account • If the conditions do not hold, return error detailing what condition was incorrect
Exceptions	User is not an admin <ul style="list-style-type: none"> • Return error message saying user is not an admin.

Test Number	12
Description	Admin Verification
Preconditions	<ul style="list-style-type: none"> • Admin Account Exists
Inputs	<ul style="list-style-type: none"> • Admin Token
Expected Output	<ul style="list-style-type: none"> • If all the following conditions are true return Admin Verified boolean true/execute admin request <ul style="list-style-type: none"> • Admin Token matches the key account • Admin Token is not expired • If the conditions do not hold, return error detailing what condition was incorrect (Eg. Authentication error: Expired Token)

Exceptions	User is not an admin <ul style="list-style-type: none"> Return error message saying user is not an admin.
------------	--

4.3.3 Unit Tests

Each unit test will be run with each permutation of the possible inputs,  to make sure the system behaves properly no matter the input, by verifying that the expected output matches the input. The expected output will depend on the input, expecting the proper output if all the input conditions hold and expecting an error message if any of the input conditions are not met. In order for the unit tests to pass the expected output has to be correct for each possible input, therefore the tests will verify that error messages are being passed when required.

adminview.SetChanges()

Overview: This method handles any changes to locker registration made by the admin.

Input: Registration changes.

Output: Confirmation message.

Positive Test Cases: The changes are reflected in the database.

Negative Test Cases: The changes are not reflected in the database.

ticket.ChangeResolved()

Overview: This method sets an administrator ticket to be resolved.

Input: Administrator hits the resolve button.

Output: Ticket status changes to resolved.

Positive Test Cases: Databases is updated to reflect changes.

Negative Test Cases: Databases is not updated. Database is erroneously updated.

ticket.AddResponse()

Overview: This method handles responses to tickets made by the admin.

Input: Response to ticket.

Output: Confirmation message. Email sent to user regarding response.

Positive Test Cases: The response is reflected in the database and the UI. The user receives an email.

Negative Test Cases: The user does not receive an email. The user receives an incorrect email. The response is not reflected in the database.

emailcontroller.CheckDate()

Overview: This method checks the current date and determines if the semesterly notification needs to be sent to users.

Input: Current date.

Output: True or false depending on if the input matches a date that an email must be sent on.

Positive Test Cases: True when the input is a reminder day. False when the input is not a reminder day.

Negative Test Cases: True when the input is not a reminder day. False when the input is a reminder day.

emailcontroller.SendEmail(array emails, bool renew)

Overview: This method sends emails to all addresses listed in the emails array.

Input: An array of emails, and a boolean denoting whether it is a renewal email or not.

Output: Emails are sent to addresses in emails array.

Positive Test Cases: All emails are successfully sent.

Negative Test Cases: One or more of the addresses in the emails array receives an erroneous email. One or more emails are sent to addresses not on the list. One or more addresses on the list do not receive emails.

registration.Get(LockerNumber, Email, Name)

Overview: This method return the full registration info for the registration associated with any one of the input variables.

Input: A locker number, a user's name, or an email address.

Output: Full registration information.

Positive Test Cases: Method successfully return all registration information for the

Negative Test Cases: Does not return registration info. Returns incorrect registration info. Returns registration info from another registration.

registration.Set(LockerNumber, Email, Name)

Overview: This method sets registrations made either by the user or an administrator.


Input: locker number, name, and email address.

Output: Registration confirmation message.

Positive Test Cases: Method correctly creates / updates a registration.

Negative Test Cases: The method does not create / update a registration, or sets a registration incorrectly.

registration.IsEmailuvic()

Overview: This method checks whether a provided email is an @uvic address or not. 

Input: An email address.

Output: Boolean, with true indicating it is an @uvic email, and false indicating not an @uvic email address.

Positive Test Cases: True if email is @uvic, false if email is not @uvic.

Negative Test Cases: False if email is @uvic, true if email is not @uvic.

registration.IsEmailUsed()

Overview: This method checks if an email has already been used to register a locker

Input: An email address.

Output: True or false depending on if email is already in use.

Positive Test Cases: False if email is not in use. True if email is in use.

Negative Test Cases: False if email is in use, True if email is not in use.

registration.RegisterLocker(number,email,name)

Overview: This method performs all the necessary tasks that lead up to a user registration.

Input: Locker number, Email and Name.

Output: An email sent to the user and the locker is temporarily registered while waiting for user verification.

Positive Test Cases: The above output is executed successfully.

Negative Test Cases: An email is not sent. The locker is not temporarily registered.

registration.SubmitForm()

Overview: This method performs some basic form validation when the user submits the registration form.

Input: Form is submitted.

Output: Registration process progresses.

Positive Test Cases: Form data is successfully verified.

Negative Test Cases: Form data is not successfully verified.

4.3.4 Integration Testing

Integration testing will occur over time as unit tests are completed. We will use bottom up integration testing for the ESS locker reservation system. This means that as we complete unit testing, we will “bundle” unit tests together into slightly larger, higher level tests, and use these to ensure that our lowest level blocks of code can work together as they should. This process of combining tests will continue with higher and higher abstraction levels, until all modules are complete, and we are ready to do functional testing of the system.

4.3.5 Acceptance Testing

Acceptance testing will be the final testing methodology used in the evaluation of the ESS locker reservation system. Acceptance testing will be performed alongside the UVic ESS. During this test, we will introduce the ESS executive members to the system, and allow them to run through some administrator use cases. Additionally, we will allow a group of potential system users to reserve lockers, and spectate as they do so, to ensure that the system performs adequately to the needs of the user, and the administrators. Interviews could potentially be used to gain additional insight as to the effectiveness of the delivered system.

4.3.6 Performance Evaluation

Performance will be evaluated through the use of both white-box and black-box testing. To test the overall performance of the system there will be black box testing which will measure the response time of various parts of the system. By running multiple tests simultaneously to stress test the system, the effect of different numbers of requests can be properly assessed. If there are tests which are running slower than desired, unit tests will be run with the black box tests to measure how long each part of the request takes, allowing for improvements to be made where they make the largest difference. The stress testing should be run when the system has zero or very few users to maintain the integrity of the test measurements and maintain performance for users. Performance tests should also be run periodically to verify that the system continuously performs at the proper level, and can be automated to run tests periodically and notify the administrators if there are performance issues.

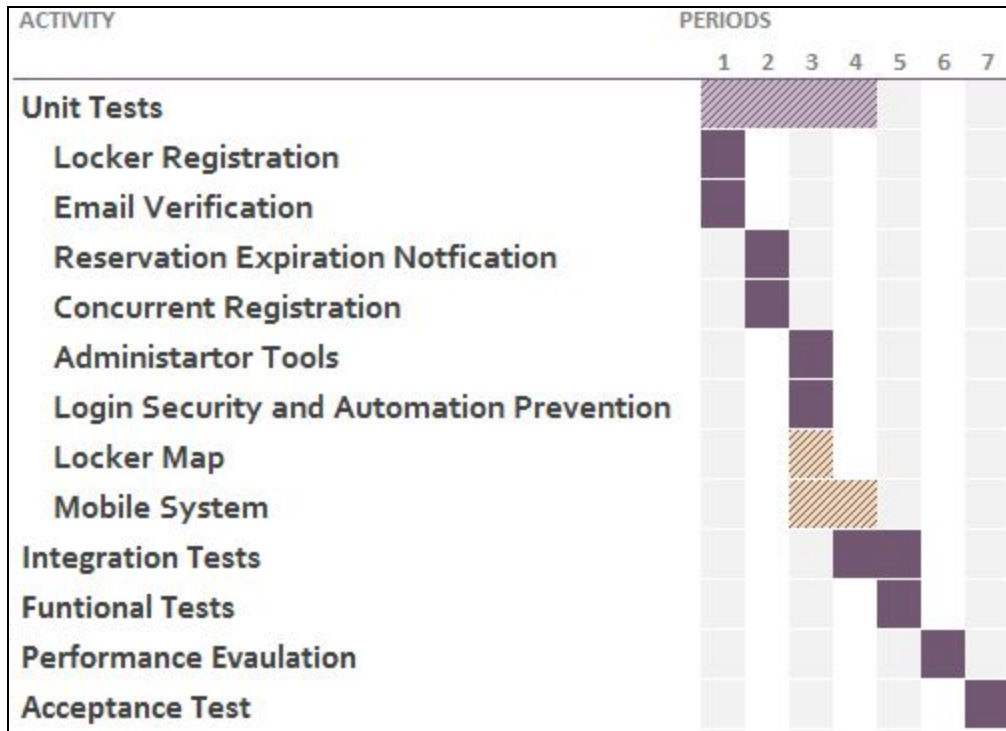
4.4 Scheduling

The testing schedule for the ESS locker reservation system is outlined in the Gantt chart below. Based on the time until the demo, each period should be 2 days. Unit testing will go for 8 days (4 periods), starting with the core functionality of the system (Locker registration) and ending with the Administrator tools, and security features. The locker map, and mobile system will be tested on day three and four, but only if time permits their development to be sufficiently completed. After Unit testing, Integration testing of each component will occur. Two days of testing have been allocated integration. This will allow plenty of time for correction of errors that may arise during testing.

Functional, performance, and acceptance testing have all been scheduled to occur on the last three days of testing. This is because these particular tests rely on the Unit testing and Integration testing to be completed first.

4.4.1 Staying on Schedule

Now that we have a designed the schedule for our test the team will focus on staying on schedule. The acceptance test date is non negotiable because this is in the form of an in class demo. Thus it is crucial that we stay on schedule as much as possible. There will be a meeting held between the developers after each stage of different testing is complete. The only test section that is deemed less important would be performance testing, as it is preferable that we have a prototype that performs the desired features, rather than performs them quickly or can handle large loads. This is preferable because better tested features will give our client a better understanding of what the final product will be like, even if it suffers from poor performance.



Gantt Chart for ESS locker reservation system testing

4.4.2 Responsibilities

Silver One Consultants is lucky to have competent developers that each have unique strengths. There are a number of ways that work could be divided up amongst the team, however what will likely happen is have developers focused on specific modules. Because the team is small, each person should attempt to become a domain expert in one area, rather than have mediocre knowledge of all areas (this is also due to time constraints). The modules (and sub-modules) of work will be:

- Registering
 - Availability map
- Email System
 - Verification link generation
- Admin Tools
 - User view for submitting ticket
 - Ticket view(s)
 - Data grid view

Both of the Registering and Admin Tools modules will also have separate technology layers. Each module will have a front-end view that needs to be developed and consistent with the rest of the webpage in terms of looks. There will also be databases needed for both of these, as well as database interaction.

The email system does not have a distinct database or front-end view associated, but will have interaction with both.

For this reason, the team will try to divide up tasks such that there are two people working for both the Register and Admin Tools modules. The idea is that one developer will handle the front-end and one will handle the database interactions. They will tackle all the middle logic together.

One team member will be assigned the email verification. One member will be assigned the task of getting the website hosted, and then moving it onto the ESS server when that becomes available.

The following chart attempts to divide up the tasks as described above. Each developer has selected the sections they have interest in working on.

	Registration			Admin Tools		Email	Web Hosting
	Front End	Back End	Locker Map	Front End	Back End	_____	_____
Alex				X		X	
Geoff		X			X		
Heather	X		X				
Tanner				X			X

4.5 Monitoring

The following section covers processes in which our development team will follow to ensure a smooth project timeline. It includes how to spot issues that may slow down production time, such as developers not pulling their weight, or bugs in the code. It does not cover keeping on schedule of the testing, which is covered in the testing section.

4.5.1 Reporting Lack of Compliance

The developers will do their best throughout the course of building the software to stay on task, follow professional ethics, and code at the highest quality capable. Developers will have different branches as they were on different features, and time will be taken to review code before it is merged into master. If decent reviews are held, future errors can be prevented, and better code can be written the first time. Reviews will hold our developers accountable to each other and to the client.

When a developer discovers a problem in the code, he or she can use the GitHub blame feature to trace back to who made the original code change. If it is found that problems are always coming back to the same person, or problems are slipping through a code reviewer, more time will be allotted for code reviewing (adding an extra developer to each review).

If a developer is underperforming (see below on assigning issues), based on the number of bugs they tackle, the number of commits they make, etc. the team will have group and one on one meetings to discuss how it is affecting the project's progress. GitHub has comprehensive statistics on the amount each developer is contributing to the project, so it can be seen by all if someone is slacking off.

In addition, members are encouraged to bring forth any ethical concerns. This could be questionable ethics decisions from teammates, or potentially difficult ethical positions that a member has been put in. These will be discussed with the team in length.

4.5.2 Reporting and Correcting Bugs

GitHub will be where the repository of the code source is stored. Because of this, certain features of GitHub can be used such as ‘issues’. Issues can be created in the form of reports for bugs that are encountered. To help with consistency and prioritization, developers should include in their report a summary, suspected files, and a severity number. The severity numbers are described in the chart below. The higher the severity number, the worse the problem is.

#	Severity	Description
9	Critical	Data loss, corruption, or system unavailable to users. No workarounds
8-7	High	Data loss, corruption, or system unavailable to users. Workaround exists.
6-5	Medium High	Important functionality is unavailable. Workaround exists.
4-3	Medium Low	Secondary functionality is unavailable. Workaround exists.
2-1	Low	Cosmetic Issue. Simple workaround.

A feature of issues is the ability to add labels to issues. The labels for this project will be used according to the module they impact. Some examples are “Registration”, “Email”, “Admin”. This allows the developers that worked on specific parts of the problem to see which issues fall under their specialized areas. Developers are encouraged to tackle issues that they may have caused, or work on issues which they know the most about the context.

The assign feature of GitHub issues can also be used. Developers can be assigned to issues. This is a good feature because it ensures that people do not overlap on their work. Moreover, an overview of people’s workloads can be seen from how many issues they have been assigned to.

4.6 Discussion

The above testing will ensure that the final system is correct and meets the specified requirements set by the clients. The testing ranges from automated unit testing of specific methods, to acceptance testing of the entire system which will be conducted with the client. We will also do integration tests for multiple modules as well as functional tests to ensure that each functional requirement is met. Functional testing will also ensure that our front end UI interacts correctly with our back end system and database.

Unit tests will be written for each method / module. The will be fully automated in order to rigorously test each method with a large variety of inputs. Automatic or manual integration tests will be conducted to ensure that each method and module interacts correctly. Performance testing will be conducted after unit, integration, and functional testing but before acceptance testing. Our system will be tested to ensure it can handle a large number of reservation requests in a short amount of time. Ultimately performance testing is not as important as there are a finite number of lockers, but we should ensure that our system can handle a large number of request elegantly and quickly. Once we deem that our system is complete and up to standard, we will conduct acceptance testing with our client to ensure that the system meets their needs and expectations.

5.0 Concluding Summary

This document presents the assessed requirements of the proposed ESS locker registration system. The ESS locker registration system will function as a replacement for the current ESS locker registration system, which is difficult to use. This document contains a basic outline of the planned registration system, the desired features, the user interactions, and plans on the implementation of the system.

The main feature is the locker registration. The registration will allow users to register for an available locker using their UVic email. When registering the user will have to verify their email and enter a captcha. Users will be able to see which lockers are available and the location of the lockers throughout the ELW using a graphical interface making it easier for students to register for lockers. There will also be an admin page which will allow system administrators to add or

remove locker registrations, once they log in to the website. Once the locker has been registered to a user for a semester there will be a email notification alerting them that the locker registration will have to be renewed, if the user wants to use it the following semester. The system may also include a ticketing system, that can be developed if it is deemed necessary later in the planning process. The plan to include a registration queue for the system has been removed in favor of the new available locker page.

A rigorous testing plan has been put in place, including a testing schedule, testing methodologies, and individual tests. Thorough testing will occur at all stages of the systems lifetime. Testing will occur in phases, starting at the lowest level with Unit testing of individual functions. Then, once a particular module is completed, it will go through integration testing, to ensure that it will function correctly with the rest of the system. Once all the modules are complete and combined, functional testing will commence. This will allow users to work through use cases, and identify any glitches or errors. Finally, acceptance testing will be performed with both potential locker users, and ESS executive members, as a way to ensure that the delivered system is meeting the requirements of the client. This testing plan will allow SilverOne Consultants to provide the Engineering Students Society with a high quality locker reservation system.

The main system use cases have been detailed, and include locker registration, renewal, customer support and others. There are also example dialogues of a user's interaction with the system when registering for a locker, renewing their locker registration and cancelling their locker registration.

Overall LARGE Software is impressed with Silver One Consultants' Technical Design Document. Once our remarks have been acknowledged, LARGE Software is prepared to use the Conceptual Design Document and Technical Design Document as a contract for development.

C3 Evaluation of the Technical Design Document

Google Calendar Integration

1. Create valid API key.

Inputs	User signs up with google account
Outputs	User receives API key Calendar functionality is unlocked
Exceptions	If password or username are incorrect they do not receive a key

2. Get events from current day.

Inputs	You have connected with the google API
Outputs	Events are updated within the app, and the app now reads you your upcoming events.
Exceptions	User does not have any events on current day

3. Chronologically order the events.

Inputs	Application requests events
Outputs	Events are ordered in app and read to the user in chronological order
Exceptions	User does not have any events on current day

4. Validate how many times we can request data from the calendar API.

Inputs	Many events exist for current day Application requests events
Outputs	Events are added to applications event database
Exceptions	Too many requests are made and API stops giving responses

Alarm Interface

1. Create an alarm, the alarm should now be enabled.

Inputs	User creates alarm from home screen
Outputs	Alarm is created, and enabled
Exceptions	If alarm is already created on this day, error is displayed.

2. Select different notification settings and times.

Inputs	User opens dropdown on an alarm User selects configuration for an alarm User changes settings
Outputs	New settings should be active immediately
Exceptions	If no notifications are selected, then the alarm should simply be the sound and nothing else (similar to the stock android alarm clock, with the added snooze functionality).

3. Deleting an Alarm

Inputs	User opens dropdown on an alarm User selects delete
Outputs	Prompt is displayed that gives users the option of "Confirm" or "Cancel"
Exceptions	None

4. Undoing Deletion

Inputs	User follows path for test 3 User selects undo after deleting the alarm
Outputs	Alarm returns back to original state, before being deleted
Exceptions	None

5. Snooze

Inputs	Alarm is going off User selects snooze
Outputs	Alarm is snoozed for specified time for given alarm
Exceptions	None

Integration with LIFX Smart Light

1. Configure LIFX light with app.

Inputs	LIFX Smart Light is not Configured Phone and Smart Light are on the same LAN Phone is connected to internet
Outputs	Phone is connected to smart light Smart light is configured
Exceptions	Invalid username or password does not allow configuration

2. LIFX light should not be triggered if user is not connected to local network (ie. at friends house).

Inputs	Phone is not connected to same LAN as Smart Light Smart light is configured Phone is connected to internet
Outputs	Alarm should not trigger light
Exceptions	None

3. Toggle integration on/off

Inputs	LIFX Smart Light is Configured Phone is on same LAN and Smart Light Phone is connected to internet User Toggles smart light integration
Outputs	Integration is toggled correctly Smart light is still configured but not connected

Exceptions	Invalid username or password will not allow configuration
------------	---

4. Select breathe/pulse effects.

Inputs	LIFX Smart Light is Configured Phone is on same LAN and Smart Light Phone is connected to internet User Toggles Breathe or Pulse
Outputs	Breathe or Pulse is correctly selected
Exceptions	None

Implementation Specific Tests

Alarm Integration Tests (Page 31)

Inputs	Android Alarm Library WakefulBroadcastReceiver Alarm Interaction Fragment
Outputs	Alarm Dismissed
Exceptions	None

Alarms should be able to be dismissed or snoozed based on interaction with the default Android alarm library. Mock the library and write tests.

Text to speech (Page 32)

Inputs	Android Text To Speech Android Virtual Assistant Text to be read
Outputs	Text is read out and is understandable Volume is appropriate
Exceptions	On invalid text the text to speech does nothing

Text to speech should not be unit tested, it should be tested using quality assurance personnel. Make sure that text to speech integration is not overlooked.

Database Tests (Page 62)

Test for Injections

Inputs	SQLite database connection Query String
Outputs	None
Exceptions	If user attempts a SQL injection the database should error

SQLite should not allow any code to be executed from user queries. Make sure to test that all user input is sanitized before a query is sent to the database.

Removal of Tables (Page 63)

Inputs	SQLite database connection Query String
Outputs	Removed Table
Exceptions	If a user tries to remove a table that is not initialized it should error

Removing of tables should be handled gracefully. The removal of tables should be a solid and scalable interaction between the application and the database. Should error on removal of a non existent table