EricGarbage Online Development and Testing Documentation

Version 1.0

TEAM MEMBERS

Name	Student ID
Jake Wheeler	jlw5970
Nate Christiansen	ncc5136
Alexander Lee	asl5253

Table of Contents

Revision History

Date	Version	Description	Author
11/15/2016	1.0	Set up development project and begin working	Nate
11/20/2016	1.0	continue development of project	Nate
11/27/2016	1.0	Initial document	Jake
11/28/2016	1.0	implementation (admin functionality)	Jake
11/28/2016	1.0	Customer functionality implementation	Nate
11/29/2016	1.0	Updated testing approaches	Jake
11/30/2016	1.0	Continue admin functionality	Jake
11/30/2016	1.0	Continue development	Nate
12/1/2016	1.0	Continue project/document development, set up testing framework	Jake
12/1/2016	1.0	Created fuzz testing tool, added results to document	Jake
12/1/2016	1.0	Continue development	Nate
12/3/2016	1.0	Add to testing section, revisions of previous section, found and corrected bugs	Jake
12/3/2016	1.0	Write unit test cases, perform penetration testing, update document	Nate

1. Introduction

This document is a continuation of the documentation for the ErieGarbage Online system being created for the client ErieGarbage. This document will cover every detail regarding coding and testing (CT) of the ErieGarbage Online software system. This document will first discuss coding ErieGarbage Online. The coding section will feature an updated class diagram of the system, documentation of each class within the system, and documentation of methods that are contained within each class. After the coding section will be the testing section of the document. This section will include the test approaches that were used while testing ErieGabage Online, the bugs that were found and documented in each test, and the way that the development team handled each of these bugs in the implementation of the system.

1.1 Purpose

The purpose of this document is to provide details about the coding and testing (CT) stage of ErieGarbage Online. It will include all information pertaining to the coding and testing of the system. One of the main purposes of this document is record tests and continuously update as needed.

1.2 Scope

The scope of this document covers the ErieGarbage online system coding and testing phase. It will, in conjunction with the requirements documentation and design documentation, detail all of the classes and methods that will be part of the system, as well as the testing that is needed to ensure that the highest quality of security is built into the system. The main intent of this document is to provide a reference as to how each unit of the system (classes, methods, etc.) were designed and implemented, and how each of those components were tested. We will also include the results of those tests in this document.

2. Coding

This section will discuss all details that pertain to the coding of ErieGarbage Online. This section will outline the updated class diagram(s), discuss the documentation of each class in the system, and will discuss the documentation of each method within individual classes. Subsection 2.2 will discuss the findings of the static code analysis tool as the coding of the system takes place.

2.1 Code Documentation

Below, we will outline all documentation relating to the implementation portion of ErieGarbage Online.

2.1.1 Class Diagram

The class diagram of ErieGarbage Online represents the current design of the system. The system has been developed using the MVC design pattern, which consists of the Model, View, and Controller. This subsection will allow a better understanding of how each piece in the design fits together. We will look at each component of the system and explain each in detail.

Models:

The model is called whenever the controller is requested by the user and validates that the user can add, manipulate, and retrieve data from the model. ErieGarbage Online consists of a model package containing models for an admin, customer, the different message types (complaints, bill disputes, and suspensions), and a customer bill. These models store the respective type data. The model package is shown below in **Figure 2.1.1 (1)**.

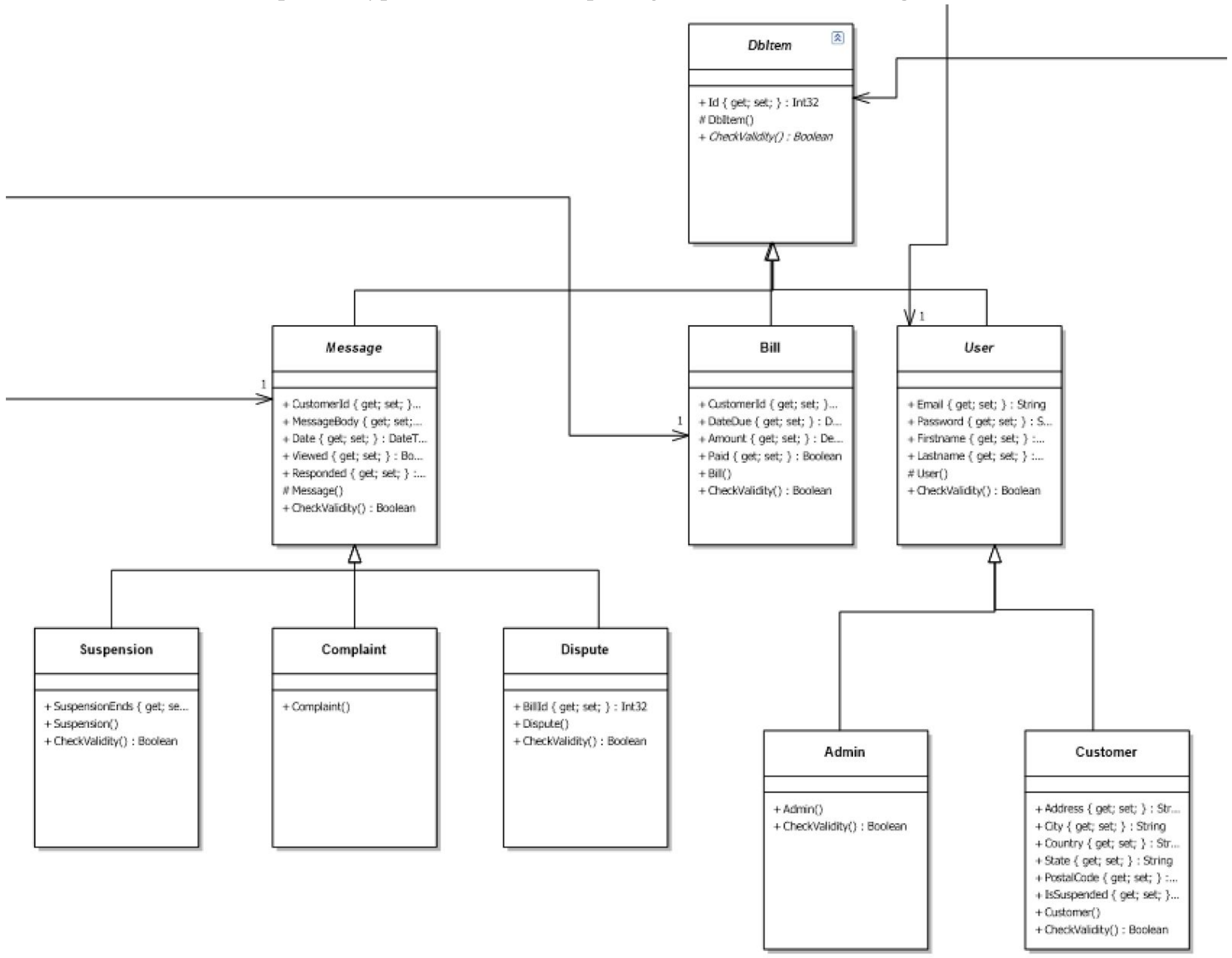

DbItem
+ Id { get; set; } : Int32
# DbItem()
+ CheckValidity() : Boolean
Message
+ CustomerId { get; set; }...
+ MessageBody { get; set;...
+ Date { get; set; } : DateT...
+ Viewed { get; set; } : Bo...
+ Responded { get; set; } :...
# Message()
+ CheckValidity() : Boolean
Bill
+ CustomerId { get; set; }...
+ DateDue { get; set; } : D...
+ Amount { get; set; } : De...
+ Paid { get; set; } : Boolean
+ Bill()
+ CheckValidity() : Boolean
User
+ Email { get; set; } : String
+ Password { get; set; } : S...
+ Firstname { get; set; } :...
+ Lastname { get; set; } :...
# User()
+ CheckValidity() : Boolean
Suspension
+ SuspensionEnds { get; se...
+ Suspension()
+ CheckValidity() : Boolean
Complaint
+ Complaint()
Dispute
+ BillId { get; set; } : Int32
+ Dispute()
+ CheckValidity() : Boolean
Admin
+ Admin()
+ CheckValidity() : Boolean
Customer
+ Address { get; set; } : Str...
+ City { get; set; } : String
+ Country { get; set; } : Str...
+ State { get; set; } : String
+ PostalCode { get; set; } :...
+ IsSuspended { get; set; }...
+ Customer()
+ CheckValidity() : Boolean
1
1
1

Figure 2.1.1 (1) - Models Diagram

Views:

The view is between the model and controller components in MVC. The user uses the view as a guide to create requests for the controller that will add, manipulate, and retrieve data from the model. The view is used to display information about the model to the user. ErieGarbage Online consists of three packages which are an Admin package that holds all views exclusive to an admin, a Customer package holding all views seen only by the customer, and a Login package containing views that can be seen by both admins and customers. The view package is shown below in **Figure 2.1.1 (2)**.

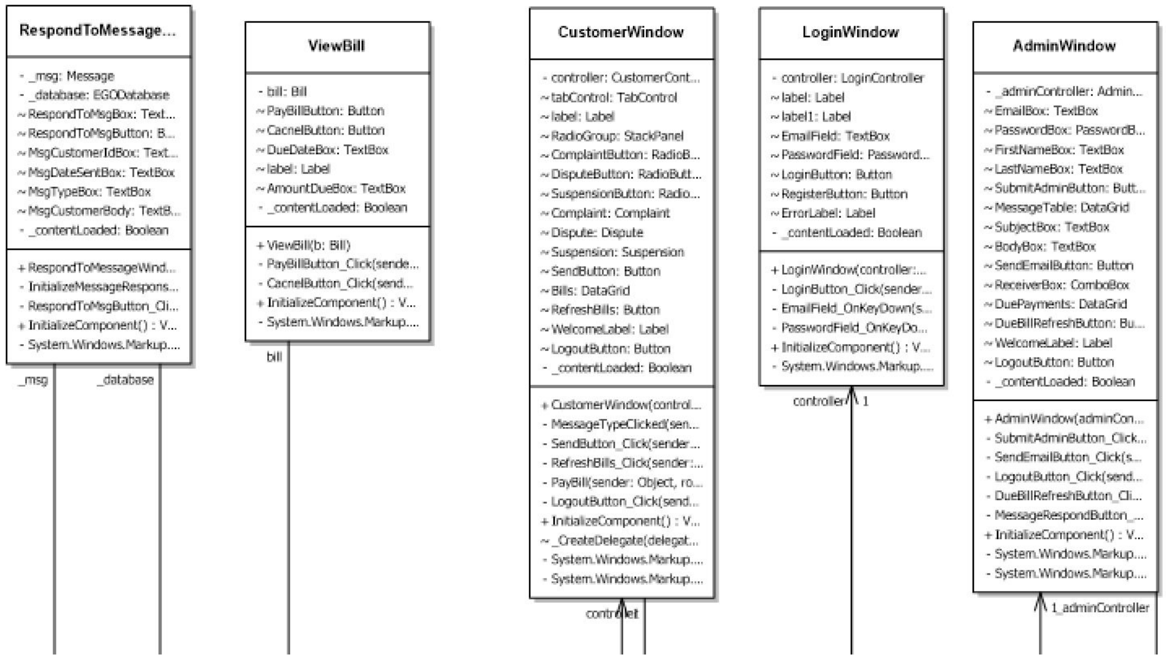

RespondToMessage...
- _msg: Message
- _database: EGODatabase
~ RespondToMsgBox: Text...
~ RespondToMsgButton: B...
~ MsgCustomerIdBox: Text...
~ MsgDateSentBox: TextBox
~ MsgTypeBox: TextBox
~ MsgCustomerBody: TextB...
- _contentLoaded: Boolean
+ RespondToMessageWind...
- InitializeMessageRespons...
- RespondToMsgButton_Cli...
+ InitializeComponent() : V...
- System.Windows.Markup....
_msg
_database
ViewBill
- bill: Bill
~ PayBillButton: Button
~ CacnelButton: Button
~ DueDateBox: TextBox
~ label: Label
~ AmountDueBox: TextBox
- _contentLoaded: Boolean
+ ViewBill(b: Bill)
- PayBillButton_Click(sende...
- CacnelButton_Click(send...
+ InitializeComponent() : V...
- System.Windows.Markup....
bill
CustomerWindow
- controller: CustomerCont...
~ tabControl: TabControl
~ label: Label
~ RadioGroup: StackPanel
~ ComplaintButton: RadioB...
~ DisputeButton: RadioButt...
~ SuspensionButton: Radio...
~ Complaint: Complaint
~ Dispute: Dispute
~ Suspension: Suspension
~ SendButton: Button
~ Bills: DataGrid
~ RefreshBills: Button
~ WelcomeLabel: Label
~ LogoutButton: Button
- _contentLoaded: Boolean
+ CustomerWindow(control...
- MessageTypeClicked(sen...
- SendButton_Click(sender...
- RefreshBills_Click(sender:...
- PayBill(sender: Object, ro...
- LogoutButton_Click(send...
+ InitializeComponent() : V...
~ _CreateDelegate(delegat...
- System.Windows.Markup....
- System.Windows.Markup....
controller
LoginWindow
- controller: LoginController
~ label: Label
~ label1: Label
~ EmailField: TextBox
~ PasswordField: Password...
~ LoginButton: Button
~ RegisterButton: Button
~ ErrorLabel: Label
- _contentLoaded: Boolean
+ LoginWindow(controller:...
- LoginButton_Click(sender...
- EmailField_OnKeyDown(s...
- PasswordField_OnKeyDo...
+ InitializeComponent() : V...
- System.Windows.Markup....
controller 1
AdminWindow
- _adminController: Admin...
~ EmailBox: TextBox
~ PasswordBox: PasswordB...
~ FirstNameBox: TextBox
~ LastNameBox: TextBox
~ SubmitAdminButton: Butt...
~ MessageTable: DataGrid
~ SubjectBox: TextBox
~ BodyBox: TextBox
~ SendEmailButton: Button
~ ReceiverBox: ComboBox
~ DuePayments: DataGrid
~ DueBillRefreshButton: Bu...
~ WelcomeLabel: Label
~ LogoutButton: Button
- _contentLoaded: Boolean
+ AdminWindow(adminCon...
- SubmitAdminButton_Click...
- SendEmailButton_Click(s...
- LogoutButton_Click(send...
- DueBillRefreshButton_Cli...
- MessageRespondButton_...
+ InitializeComponent() : V...
- System.Windows.Markup....
- System.Windows.Markup....
1 _adminController

Figure 2.1.1 (2) - Views Diagram

Controllers:

The controllers are needed to allow the user to obtain information about the models and allow interaction with the models. ErieGarbage Online has three public controller interfaces consisting of the AdminController, CustomerController, and LoginController. The controller package is shown below in **Figure 2.1.1 (3)**.

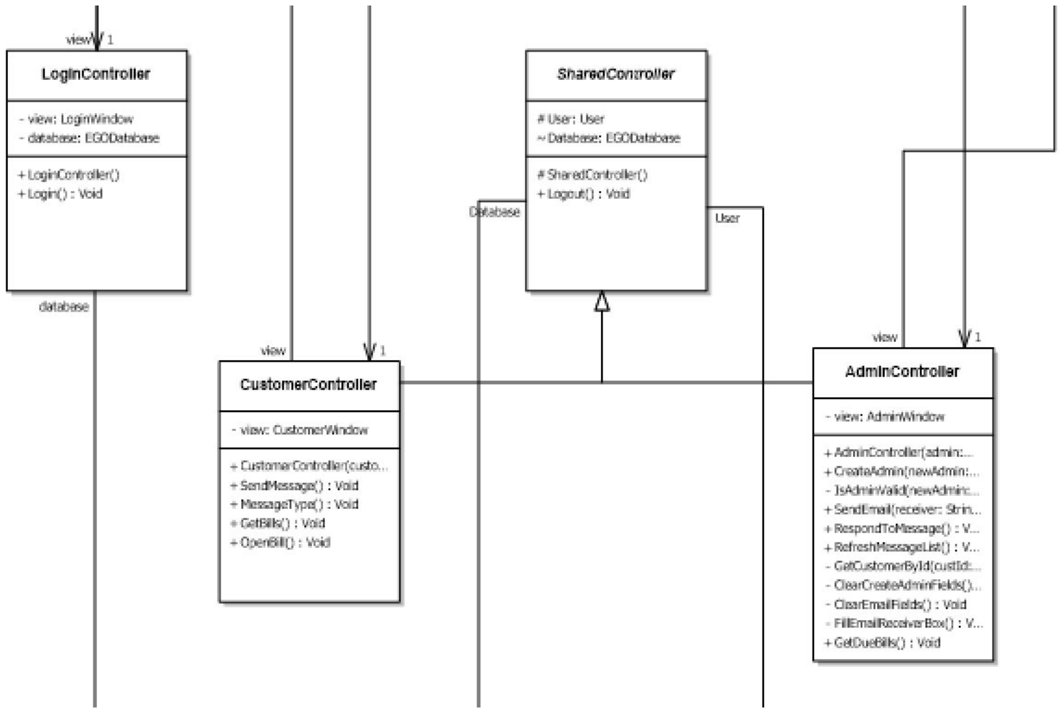


Figure 2.1.1 (3) - Controllers Diagram

Database:

The database currently consists of a serialized object file. This database allows us to store every single type of model as well as the changes that are made to the models during runtime. The database is designed using the singleton design pattern to ensure the same instance of the database is always being accessed by the user. The outline of the database is shown below in **Figure 2.1.1 (4)**.

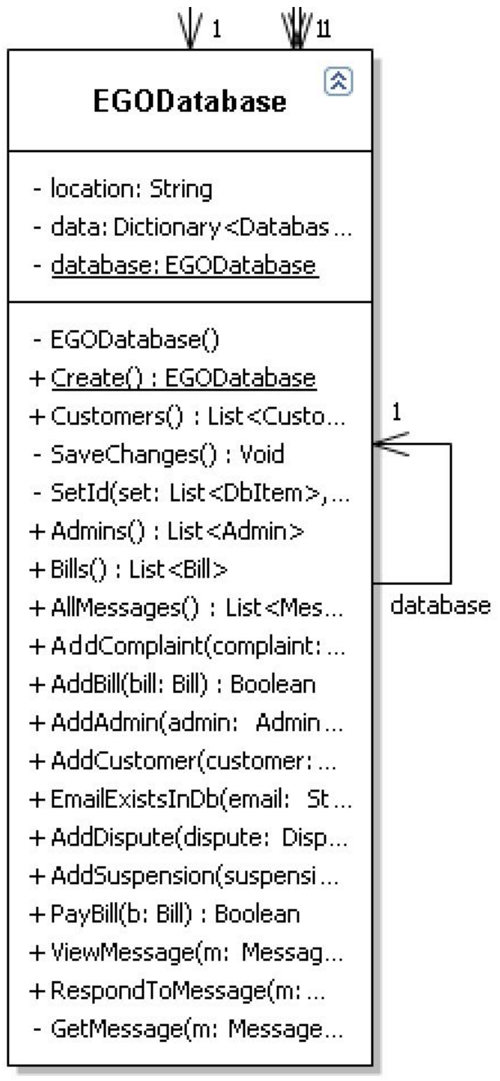

1
11
EGODatabase
- location: String
- data: Dictionary<Databas...
- database: EGODatabase
- EGODatabase()
+ Create() : EGODatabase
+ Customers() : List<Custo...
- SaveChanges() : Void
- SetId(set: List<DbItem>,...
+ Admins() : List<Admin>
+ Bills() : List<Bill>
+ AllMessages() : List<Mes...
+ AddComplaint(complaint: ...
+ AddBill(bill: Bill) : Boolean
+ AddAdmin(admin: Admin...
+ AddCustomer(customer: ...
+ EmailExistsInDb(email: St...
+ AddDispute(dispute: Disp...
+ AddSuspension(suspensi...
+ PayBill(b: Bill) : Boolean
+ ViewMessage(m: Messag...
+ RespondToMessage(m: ...
- GetMessage(m: Message...
1
database

Figure 2.1.1 (4) - Database

2.1.2 Class Documentation

This subsection will outline each class in terms of its security. We will analyze the following in each class:

- Protected data owned by the class
- Authentication procedures used by class before data is revealed
- Methods that change data owned by the class
- Methods that reveal data owned by the class
- Minimum guaranteed security of data owned by class

Class: **LoginController**

Protected data owned by class	**Authentication procedures used by class before data is revealed**	**Methods that change data owned by the class**	**Methods that reveal data owned by the class**	**Minimum guaranteed security of data owned by class**
-view : LoginWindow -database : EGODatabase	Login() : void	No data owned by the class is changed	No data owned by the class is revealed	All secure

Class: **AdminController**

Protected data owned by class	**Authentication procedures used by class before data is revealed**	**Methods that change data owned by the class**	**Methods that reveal data owned by the class**	**Minimum guaranteed security of data owned by class**
#User : User -view : AdminWindow -database : EGODatabase	IsAdminValid() : bool	No data owned by the class is changed	No data owned by the class is revealed	All secure

Class: **Message**

Protected data owned by class	**Authentication procedures used by class before data is revealed**	**Methods that change data owned by the class**	**Methods that reveal data owned by the class**	**Minimum guaranteed security of data owned by class**
-CustomerId : int -MessageBody : string -Date : DateTime -Viewed : bool -Responded : bool	CheckValidity() : bool	All setters can freely change respective data	All getters can freely get respective data	A customer ID must match the correct ID in the database, otherwise the message will not be accepted into the system

2.1.3 Method Documentation

This subsection will outline each method within a specified class in terms of its security. We will analyze the following in each method:

- Minimum expectation of input with optional input allowed
- Output information that results from the method
- Error output
- External resources accessed
- Fail state result

Class: **AdminController**
Method: **CreateAdmin(newAdmin : Admin)**

Min. expectation of input with optional input	**Output information that results from the method**	**Error output**	**External resources accessed**	**Fail state result**
N/A	Message box displaying that the admin was successfully added to the database	Message box displaying that the admin failed to be added to the database	Database.ego file	New admin is not added to the database

Class: **EGODatabase**
Method: **AddBill(bill : Bill)**

Min. expectation of input with optional input	**Output information that results from the method**	**Error output**	**External resources accessed**	**Fail state result**
N/A	Bool true is returned	Bool false is returned	Database.ego file	The bill is not added to the database

Class: **CustomerController**
Method: **OpenBill()**

Min. expectation of input with optional input	**Output information that results from the method**	**Error output**	**External resources accessed**	**Fail state result**
A bill selected on the customer screen	A window opens with the information of the bill and the ability to pay if it is unpaid	If no bill is selected, nothing will happen and no window will appear	Database.ego file	Bill information will not be displayed

2.2 Static Code Analysis

Static code analysis is the automation of searching for flaws in the code that could be potential problems in the future. To perform static code analysis, we will be using the code analysis tool that is built into Microsoft Visual Studio called FxCop. To access this tool, navigate to **Analyze → Run Code Analysis on Solution** within Visual Studio. We are also using Resharper Ultimate, which allows analyzes our code as we type. In theory, this should allow for fewer flaws in our code when it comes time to run the static code analysis tool. All results of the static code analysis will be listed here throughout the development of EricGarbage Online. Static code analysis will be run occasionally throughout the development cycle to ensure quality coding practices.

Analysis ID	**Date of Analysis**	**Results of Analysis**	**Post Analysis Corrections**
0	11/28/16	No code analysis issues were detected.	N/A
1	11/30/2016	CA2200 Rethrow to preserve stack details 'EGODatabase.SaveChanges()' rethrows a caught exception and specifies it explicitly as an argument. Use 'throw' without an argument instead, in order to preserve the stack location where the exception was initially raised. ErieGarbageOnline EGODatabase.cs 77	Removed the argument so that the stack location is preserved
2	12/1/2016	No code analysis issues were detected.	N/A
3	12/3/2016	No code analysis issues were detected.	N/A

3. Testing

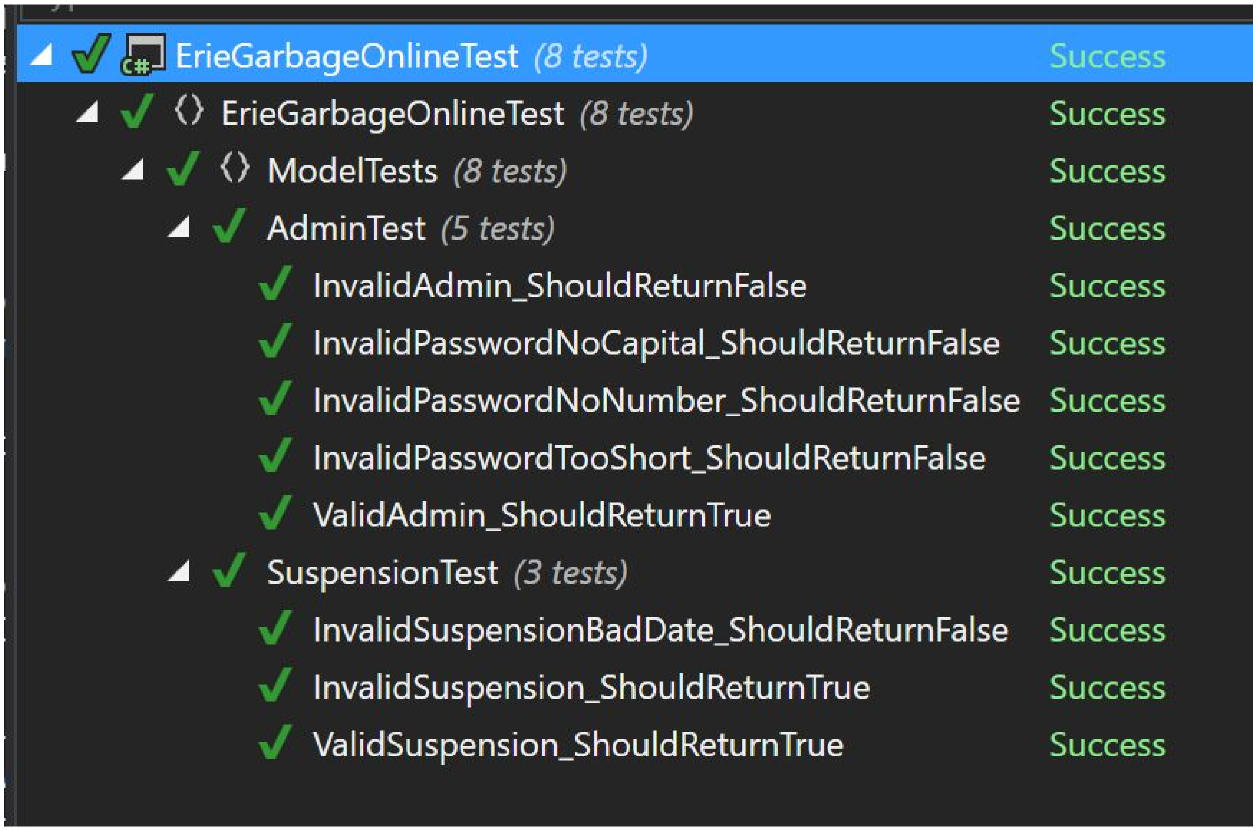
This section documents how testing of ErieGarbage Online has been performed. This section will detail the types of tests that have been performed during and after the development of the system.

3.1 Test Approaches

ErieGarbage Online has been tested using the following approaches: Unit testing, integration testing, system testing, and penetration testing. Each of these test types will be described in detail in the subsections below.

3.1.1 Unit Testing

Unit testing is the testing of one "unit." A unit may refer to a method, class, package, etc. To perform unit testing, we will be utilizing Microsoft Visual Studio's built in testing tools. We will also be using Resharper Ultimate to aid us in completing tests and allow for simpler execution. To perform unit testing, we have created a project within our solution called "ErieGarbageOnlineTests." This project will allow us to create testing classes for each of the classes within the ErieGarbageOnline project. The results of these tests will be shown in **Figure 3.1.1 (1)**.



```
◢ ✓ ErieGarbageOnlineTest (8 tests)                          Success
  ◢ ✓ {} ErieGarbageOnlineTest (8 tests)                     Success
    ◢ ✓ {} ModelTests (8 tests)                              Success
      ◢ ✓ AdminTest (5 tests)                                Success
          ✓ InvalidAdmin_ShouldReturnFalse                   Success
          ✓ InvalidPasswordNoCapital_ShouldReturnFalse       Success
          ✓ InvalidPasswordNoNumber_ShouldReturnFalse        Success
          ✓ InvalidPasswordTooShort_ShouldReturnFalse        Success
          ✓ ValidAdmin_ShouldReturnTrue                      Success
      ◢ ✓ SuspensionTest (3 tests)                           Success
          ✓ InvalidSuspensionBadDate_ShouldReturnFalse       Success
          ✓ InvalidSuspension_ShouldReturnTrue               Success
          ✓ ValidSuspension_ShouldReturnTrue                 Success
```

Figure 3.1.1 (1) - Unit Test Execution

3.1.2 Integration Testing

Integration testing is the phase of testing in which individual components of the software are combined and tested together. This phase of testing allows us to see how different components will work together and check if the expected results are still given by the system.
To perform integration testing, we began with the LoginWindow and the login functionalities. From there, we had a base from which admin and customer functionalities could be added to the system. We were able to distribute functionalities between team members and begin development. Once those components were completed, we could then test them individually. At this point, three components could be combined: LoginWindow, AdminWindow, and CustomerWindow. Debugging was performed until these three combined components had as little errors as possible. From there, we could then add more functionality to both the admin and customers.

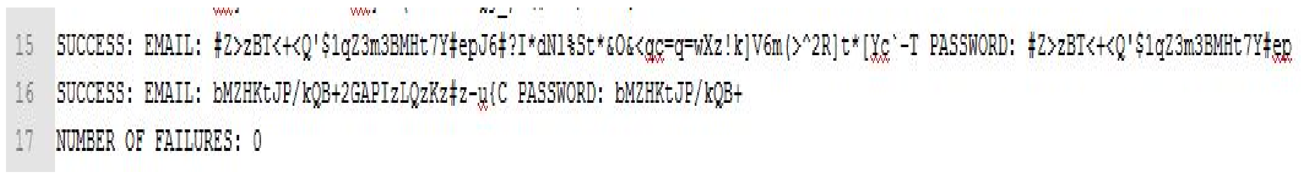
3.1.3 System Testing

System testing is the testing of all components combined as a complete system. This phase of testing takes a "black-box" approach, and we will be checking to see if we are given the expected outputs. In this phase, we aim to discover any inconsistencies between the integrated components that were tested earlier.
To perform system testing, we first combined all components of the system which includes the login, admin, admin functionalities, customer, and customer functionalities. Once combined, we were able to see the system would perform from a black-box point of view. During this phase of testing, many bugs included in **Section 3.2 Bugs** were discovered and corrected.

3.1.4 Penetration Testing

Penetration testing is the testing of a computer network or application to discover vulnerabilities that an attacker could exploit. This subsection will detail which methods of testing were used in the penetration testing phase and what the results of those tests were.

Fuzz Testing:
Fuzz testing is a testing method that involves adding random data into input fields to see if the program can perform correct input checking. Unfortunately, there are not many fuzz testing tools out there for WPF programs. In this case, a custom, automated fuzz testing tool was created. This tool allows for our different views to have ErieGarbage Online's view input fields tested to see if any errors occur. The fuzz testing tool will then generate a simple report to let the developer know if a test failed. **Figure 3.1.4 (1)** below demonstrates a successful fuzz test on the Login view.



```
SUCCESS: EMAIL: #Z>zBT<+<Q'$1qZ3m3BMHt7Y#epJ6#?I*dN1%St*&O&<gc=q=wXz!k]V6m(>^2R]t*[Yc`-T PASSWORD: #Z>zBT<+<Q'$1qZ3m3BMHt7Y#ep
SUCCESS: EMAIL: bMZHKtJP/kQB+2GAPIzLQzKz#z-u{C PASSWORD: bMZHKtJP/kQB+
NUMBER OF FAILURES: 0
```

Figure 3.1.4 (1) - Fuzz Testing

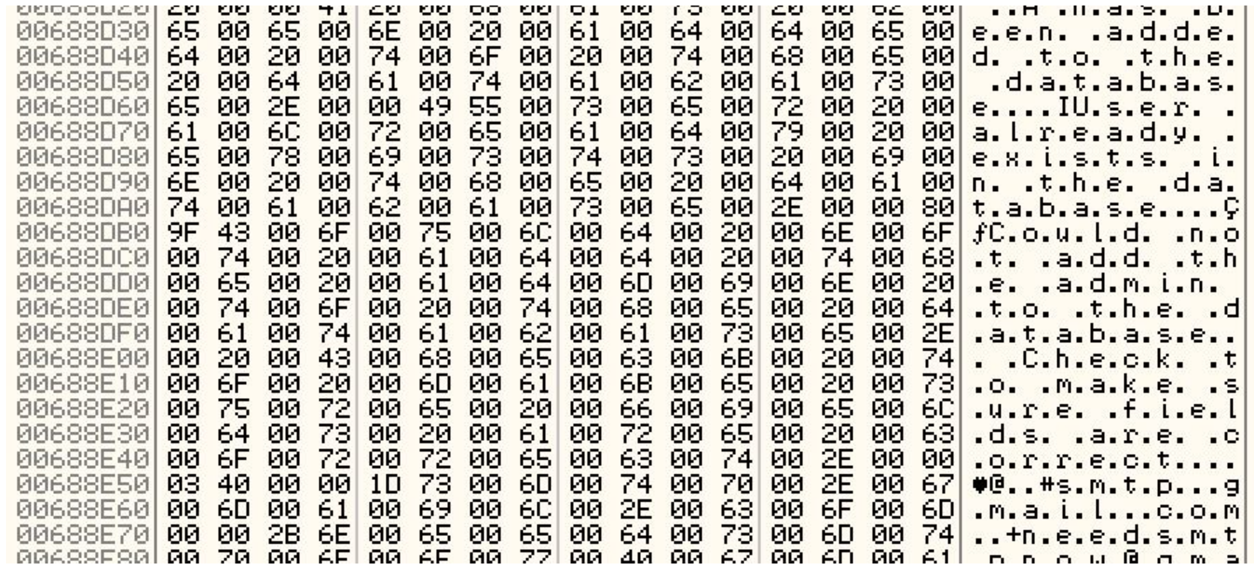
In the figure above, three fields are shown per line: [**Success/Fail], [Email], [Password].** If failures occur, they will be shown similarly to the "SUCCESS" lines above. These failures will also include the exception that was thrown at time of failure. The very end of the log file shows how many failures occurred during the run of the test.

Below are the results of the fuzz testing performed on the login window.

Date of Test	**Form Tested**	**Number of Tests Run**	**Were There Failures?**	**If Failures, What Were They?**
12/3/2016	LoginWindow	200	No	N/A
12/3/2016	LoginWindow	500	No	N/A
12/3/2016	LoginWindow	1000	No	N/A

Table 3.1.4 (1) - Login Window Fuzz Test Results

OllyDbg:
OllyDbg is a tool that allows for analysis of assembly level binaries by loading in processes. It offers live debugging of the process, which can be used for many different tasks. The tool can be used by crackers to extract information from the executables without access to source code. ErieGarbage Online will be using the tool to attempt to extract password information from the memory of the system, as well as database information. The **Figure 3.1.4 (2)** and **Figure 3.1.4 (3)** display some of the interfaces of the tool being used on EGO.



```
00688D30 65 00 65 00 6E 00 20 00 61 00 64 00 64 00 65 00 e.e.n. .a.d.d.e.
00688D40 64 00 20 00 74 00 6F 00 20 00 74 00 68 00 65 00 d. .t.o. .t.h.e.
00688D50 20 00 64 00 61 00 74 00 61 00 62 00 61 00 73 00  .d.a.t.a.b.a.s.
00688D60 65 00 2E 00 00 49 55 00 73 00 65 00 72 00 20 00 e....IU.s.e.r. .
00688D70 61 00 6C 00 72 00 65 00 61 00 64 00 79 00 20 00 a.l.r.e.a.d.y. .
00688D80 65 00 78 00 69 00 73 00 74 00 73 00 20 00 69 00 e.x.i.s.t.s. .i.
00688D90 6E 00 20 00 74 00 68 00 65 00 20 00 64 00 61 00 n. .t.h.e. .d.a.
00688DA0 74 00 61 00 62 00 61 00 73 00 65 00 2E 00 00 80 t.a.b.a.s.e....Ç
00688DB0 9F 43 00 6F 00 75 00 6C 00 64 00 20 00 6E 00 6F ƒC.o.u.l.d. .n.o
00688DC0 00 74 00 20 00 61 00 64 00 64 00 20 00 74 00 68 .t. .a.d.d. .t.h
00688DD0 00 65 00 20 00 61 00 64 00 6D 00 69 00 6E 00 20 .e. .a.d.m.i.n.
00688DE0 00 74 00 6F 00 20 00 74 00 68 00 65 00 20 00 64 .t.o. .t.h.e. .d
00688DF0 00 61 00 74 00 61 00 62 00 61 00 73 00 65 00 2E .a.t.a.b.a.s.e..
00688E00 00 20 00 43 00 68 00 65 00 63 00 6B 00 20 00 74 . .C.h.e.c.k. .t
00688E10 00 6F 00 20 00 6D 00 61 00 6B 00 65 00 20 00 73 .o. .m.a.k.e. .s
00688E20 00 75 00 72 00 65 00 20 00 66 00 69 00 65 00 6C .u.r.e. .f.i.e.l
00688E30 00 64 00 73 00 20 00 61 00 72 00 65 00 20 00 63 .d.s. .a.r.e. .c
00688E40 00 6F 00 72 00 72 00 65 00 63 00 74 00 2E 00 00 .o.r.r.e.c.t....
00688E50 03 40 00 00 1D 73 00 6D 00 74 00 70 00 2E 00 67 ♥@..#s.m.t.p...g
00688E60 00 6D 00 61 00 69 00 6C 00 2E 00 63 00 6F 00 6D .m.a.i.l...c.o.m
00688E70 00 00 2B 6E 00 65 00 65 00 64 00 73 00 6D 00 74 ..+n.e.e.d.s.m.t
```

Figure 3.1.4 (2) - EGO Memory

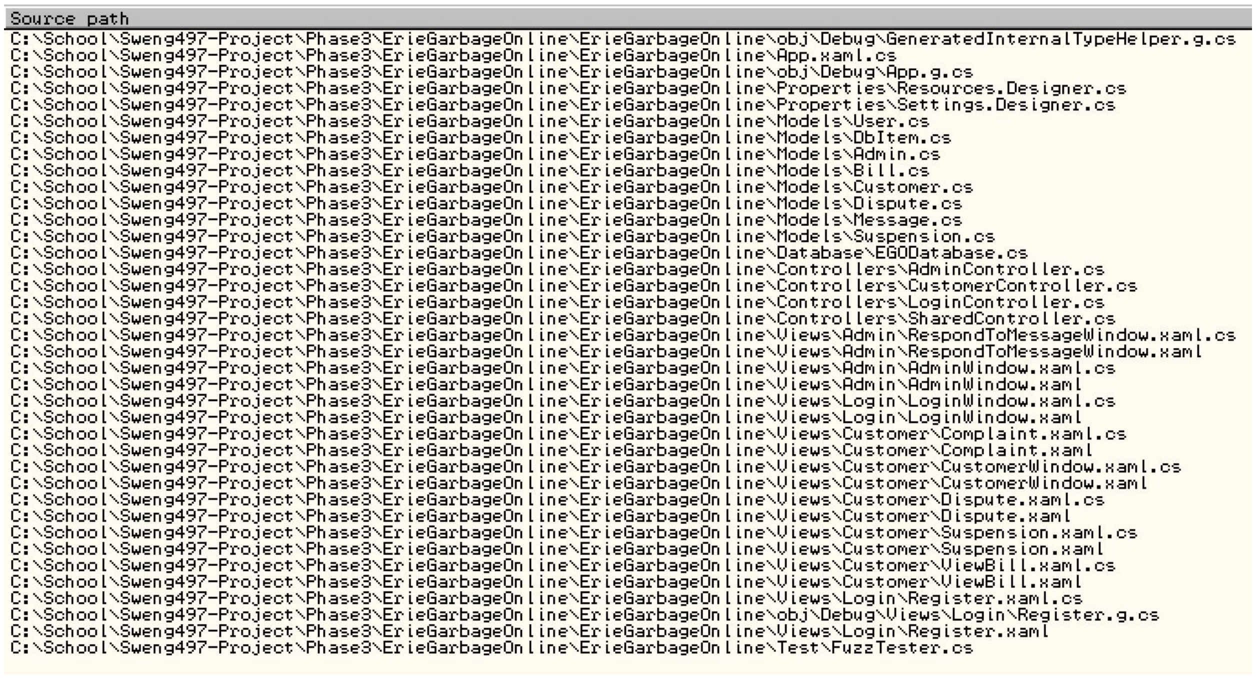

```
Source path
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\obj\Debug\GeneratedInternalTypeHelper.g.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\App.xaml.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\obj\Debug\App.g.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Properties\Resources.Designer.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Properties\Settings.Designer.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Models\User.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Models\DbItem.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Models\Admin.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Models\Bill.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Models\Customer.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Models\Dispute.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Models\Message.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Models\Suspension.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Database\EGODatabase.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Controllers\AdminController.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Controllers\CustomerController.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Controllers\LoginController.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Controllers\SharedController.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Admin\RespondToMessageWindow.xaml.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Admin\RespondToMessageWindow.xaml
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Admin\AdminWindow.xaml.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Admin\AdminWindow.xaml
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Login\LoginWindow.xaml.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Login\LoginWindow.xaml
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Customer\Complaint.xaml.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Customer\Complaint.xaml
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Customer\CustomerWindow.xaml.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Customer\CustomerWindow.xaml
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Customer\Dispute.xaml.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Customer\Dispute.xaml
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Customer\Suspension.xaml.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Customer\Suspension.xaml
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Customer\ViewBill.xaml.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Customer\ViewBill.xaml
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Login\Register.xaml.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\obj\Debug\Views\Login\Register.g.cs
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Views\Login\Register.xaml
C:\School\Sweng497-Project\Phase3\ErieGarbageOnline\ErieGarbageOnline\Test\FuzzTester.cs
```

Figure 3.1.4 (3) - Source File Tree

3.2 Bugs

This section outlines the bugs that were discovered during the implementation and testing of ErieGarbage Online. This outline will describe how the bug was found and what was done to correct the bug (if it was corrected).

Bug ID	**Bug Description**	**Corrected?**
0	When opening a message under the “View Messages” tab in the Admin Control Panel, if no response is entered and the admin exits the response window, a message is shown saying “No message to send.”	Yes - The logic between opening the window and running the business logic has been separated
1	Register button on the LoginWindow does not do anything.	No
2	Double clicking an item in any grid view will crash the program.	Yes - Setting the grids to read only fixes the problem

3	Label displaying “Welcome [customer/admin]” cuts off if the name is too long.	Yes - automatic sizing for labels has been included
4	In the CustomerWindow, the Bill tab does not correctly display the full date.	Yes - The text box size has been increased to show the full size
5	In the CustomerWindow under the “Contact Us” tab, selecting “Bill Dispute” without selecting a Bill ID will crash the program.	Yes - Checking has been added to see if Bill ID is null or empty before attempting to send the message
6	In the CustomerWindow under the “Contact Us” tab, selecting “Request Suspension” without selecting a date will crash the program.	Yes - Checking has been added to make sure the date has been selected by the user before attempting to send the message
7	In the CustomerWindow under the “Contact Us” tab, sending any type of message will not clear the text.	Yes - Text has been set to be cleared after a message has been sent successfully
8	In the CustomerWindow under the “Contact Us” tab, sending any type of message will not clear the Bill ID.	Yes - The Bill ID has been set to clear after a message has been sent successfully
9	In the CustomerWindow under the “Contact Us” tab, sending any type of message will not clear the date selected by the user.	Yes - The selected date is set to null once a message has been set successfully
10	In the CustomerWindow under the “Contact Us” tab, the Bill ID box will clear even if the message was not successful.	Yes - Checking has been implemented to make sure it only clears when a message is successfully sent
11	In the CustomerWindow under the “Contact Us” tab, the selected date box will clear even if the message was not successful.	Yes - Checking has been implemented to make sure it only clears when a message is successfully sent
12	Checking validity of database models with information not assigned throws an exception.	Yes - Check if something is null before checking validity

Table 3.2 (1) - Bugs

4. Team Members Log Sheets

4.1 Jake Wheeler

date	task	duration
11/27/2016	Initial document	1 hr
11/28/2016	Implementation (admin functionality)	6 hrs
11/29/2016	Updated testing approaches,	1 hr
11/30/2016	Continue admin functionality	6 hrs
12/1/2016	Continue project/document development, set up testing framework	3.5 hrs
12/1/2016	Created fuzz testing tool, added results to document	2 hrs
12/3/2016	Add to testing section, revisions of previous section, found and corrected bugs	4 hrs
	Total :	23.5 hrs

4.2 Nate Christiansen

date	task	duration
11/15/2016	Set up development project and begin working	1 hr
11/20/2016	continue development of project	5 hrs
11/28/2016	Customer functionality implementation	6 hrs
11/30/2016	Continue development	6 hrs
12/1/2016	Continue development	3.5 hrs
12/3/2016	Write unit test cases, perform penetration testing, update document	4 hrs
	Total :	25.5 hrs

4.3 Alex Lee

date	task	duration
	Total :	