ErieGarbage Online
Design Document
Version 1.1

TEAM MEMBERS

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

Revision History

Date	Version	Description	Author
10/24/2016	1.0	Created document, did some editing	Jake
10/26/2016	1.0	Added info about architecture	Jake
10/30/2016	1.0	Added info to introduction and architecture	Nate
10/30/16	1.0	Implement company logo	Alex
10/30/16	1.0	Rough draft of Introduction of Design Document	Alex
10/31/2016	1.0	Added more to the introductory information	Jake
10/31/2016	1.0	Added more to document and created initial architectural design	Nate
11/1/2016	1.0	Created architectural risk analysis section and added to it	Jake
11/1/2016	1.0	Began ambiguity analysis	Jake
11/1/2016	1.0	Added threats to architecture diagram	Nate
11/1/2016	1.0	Update some of risk likelihood and architectural design	Nate
11/2/2016	1.0	Added to ambiguity analysis and risk impact	Jake
11/2/2016	1.0	Updates some more risk likelihood, started impact analysis	Nate
11/2/2016		Fixed and updated weaknesses table, accidentally deleted necessary stride table	Nate
11/3/2016	1.0	Began mapping threats to STRIDE, formatted document to make developing the internal module section easier, fixed table of contents, added a few more CWEs	Jake
11/5/2016	1.0	Added to risk likelihood table, added to risk impact table, worked on internal module section, added software interface description	Jake
11/5/2016	1.0	Added to risk likelihood table, risk impact table, risk mitigation planning	Nate
11/6/2016	1.0	Wrap-up loose ends and print version 1.0	Jake

11/10/2016	1.1	Updated section 2 - rationale and software architecture, worked on changing class diagram, added section 3.2 and section 3.3	Jake
11/10/2016	1.1	Created mock-up images and added text descriptions to them in section 3.1.1, labeled every table and image	Jake
11/10/2016	1.1	Fix all vulnerability assessment stuff	Nate
11/11/2016	1.1	Updated class diagram, fixed section 5, added a javascript vulnerability	Jake
11/11/2016	1.1	Created a better architectural diagram, fixed most of ambiguity analysis	Nate
11/13/2016	1.1	Recreated architectural diagram because I don't know how to save things. Finished ambiguity analysis	Nate
11/13/2016	1.1	Final revision & submission	Jake

Design Document

1. Introduction

This document is a continuation of the Software Requirements Specification (SRS) for the ErieGarbage Online system being created for the client ErieGarbage. This document will cover every detail regarding the architecture of the ErieGarbage Online software system. This document will discuss its purpose, scope, definitions, acronyms, abbreviations, references, and overall software architecture and software interface design. It will also discuss how security will be included in each step of the design phase.

1.1 Purpose

The purpose of this document is to provide details of the architecture of ErieGarbage Online. It will include architectural and structural diagrams of the system, as well as provide risk analysis and information about the security of the architecture.

1.2 Scope

The scope of this document covers the ErieGarbage online system architecture description and diagrams. It will, in conjunction with the requirements documentation, detail all of the classes that will be part of the system, as well as the functionalities that they provide. The main intent of this document is to provide a framework for when it comes time to implement the system. This document should be used by the programmers of ErieGarbage Online as a reference as they work through the system.

1.3 Definitions, Acronyms, and Abbreviations

SDS – Software Design Specification

EGO – ErieGarbage Online

GUI – Graphical User Interface

Customer – Person using the system that cannot use admin functionality

Admin – Person using the system with access to all admin functionality, and some customer functionality

Structured external threat - State-sponsored entities that pose a risk to EGO, such as a government organization.

Transnational threat - Organized nonstate-sponsored entities that pose a risk to EGO, such as terrorist groups.

Unstructured external threat - Any entity lacking resources and organization, such as crackers.

1.4 References

Secure Software Design – Author(s): Theodor Richardson & Charles Thies – 2013

Software Quality Assurance – Authors(s): Daniel Galin - 2004

1.5 Overview

This section provided an introduction to the purpose and goals of this document. The following sections will go into details about this purpose and provide information about how the goals will be reached. Section 2 will detail the architectural design, which consists of a high level view of the system as a whole. Section 3 will provide the system interface designs, which consist of the defined interfaces that the system will use. Section 4 will give the details of these interfaces, listing the scope and functionality that each one will provide.

2. Architectural Design

2.1 Rationale

ErieGarbage Online will take advantage of the Model-View-Controller (MVC) architecture. This project is the perfect candidate for this architecture as EGO's main purpose is for data entry, manipulation, and viewing. This architecture is also popular in web development modern web development. The reason we have chosen the MVC architectural design pattern is because of its flexibility and the support around it. There is a large community around MVC, meaning it is much simpler to solve problems we run into. MVC has been studied and well documented whether it be by Wikipedia, CodeProject, or Google. In the chance that ErieGarbage chooses another company to perform updates or the maintenance of EGO, it will be much easier for that development team to adapt to the project since there is a good chance either one person or multiple have worked with MVC projects in the past. It is likely that someone on a software development team is experienced in the design philosophy of MVC or something very similar to it.

2.2 Software Architecture Diagram

ErieGarbage Online can be split into three components as shown below. The user will be able to see the View component, which will allow the user to interact directly with the Controller through a layer of abstraction, and those inputs will manipulate the model. As the Model updates, the View will update depending on the user's inputs and the Controller's actions. Please refer to **Figure 2.2** below shows each component and the way in which they interact with one another.

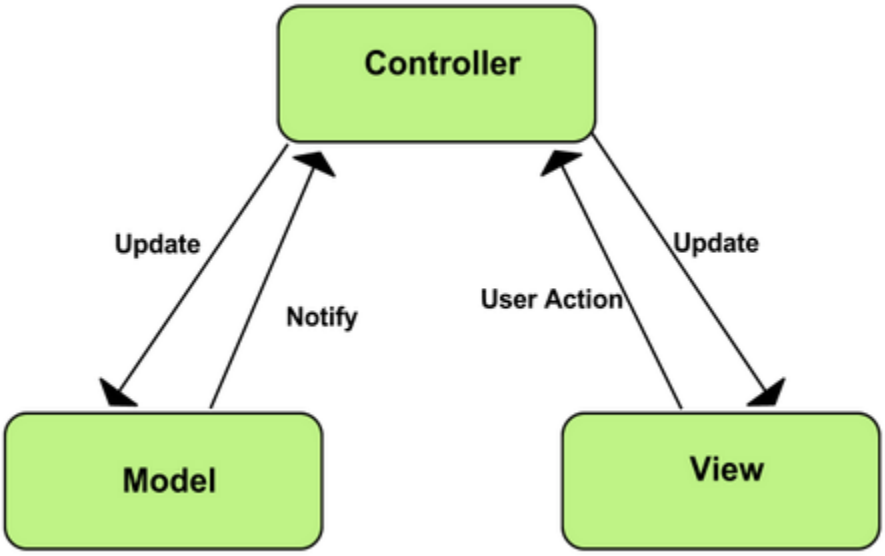

Figure 2.2 - MVC Model

(taken from https://developer.chrome.com/static/images/mvc.png)

This architecture is able to be easily utilized when using Microsoft's ASP.NET MVC framework. This framework enables a clear separation of components. This will allow us to easily modify the frontend components of the site if

the client wants to make changes. This architecture is widely used in web development, which will help to aid us in the creation and design of ErieGarbage Online. Resources on the MVC architecture and more specifically, Microsoft's ASP.NET MVC framework, are easily found online through a simple search. ErieGarbage Online will use the view component to display the user interface and model to the user. This component is made up of subcomponents, which allows the user to have a variety of views depending on their requests to the controller and the model that is being manipulated. The user will seemingly interact with these views, however, all interaction will be happening through the controller component which, like the view, is made up of multiple subcomponents. Similar to the other two components, the model component will be made up of subcomponents and those subcomponents will be used by the controllers and views.

2.3 System Topology

The ErieGarbage Online system is a web-based application that will be installed on a server and accessed through clients' computers through a web browser. The clients will be the customers running the frontend website through their browsers and the server will be the backend portion of the system running from ErieGarbage's hosting provider (SecQuality Development). Each client will connect to the system and interact with the server to use the system. There can be many clients connected at once, and should be built to scale with a large number of users. Please refer to **Figure 2.3** below.

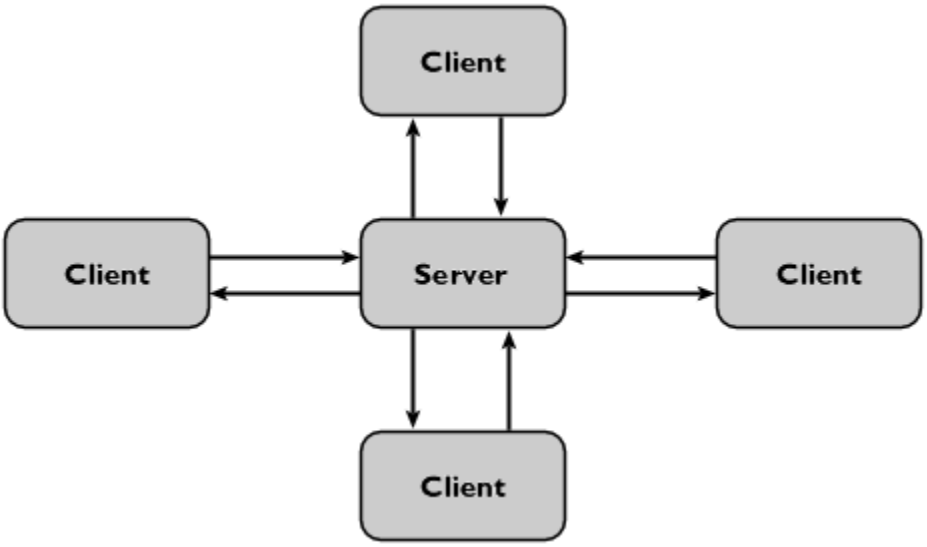

Figure 2.3 - Client-Server

3. Software Interface Design

3.1 System Interface Diagrams

System interfaces provide information about the users and systems that interact with the ErieGarbage Online system. The user interface will handle all interactions with the users, the software interfaces will allow ErieGarbage to interact with other software applications, and the hardware interface will consist of networking and the use of servers that will connect a user to the pages of the website.

3.1.1 User Interface

ErieGarbage Online will consist of a customer view and an admin view; customers and admins will only see the functions available to each group. Every user will see the same login screen, requesting the user's email and password. Once logged in, the customer, for example, will see that they can view/update their personal information, make an online payment, contact an admin to file a complaint, request an account suspension, or dispute a bill, and they can cancel their own account. Admins will be able to send an email, retreive accounts that have monthly payments due, respond to any type of customer message, and create other admin accounts. Both views are minimalistic to give the user a simple and streamlined experience.

The figure below demonstrates how logging in will look to users of ErieGarbage Online. The main page will display information about the site, its functionality and capabilities, and any additional information the ErieGarbage would like included. To the right of the information section, the user will be able to login using their email and password for EGO. Clicking the login button will take them to either the customer or the admin view, depending on their type of account.

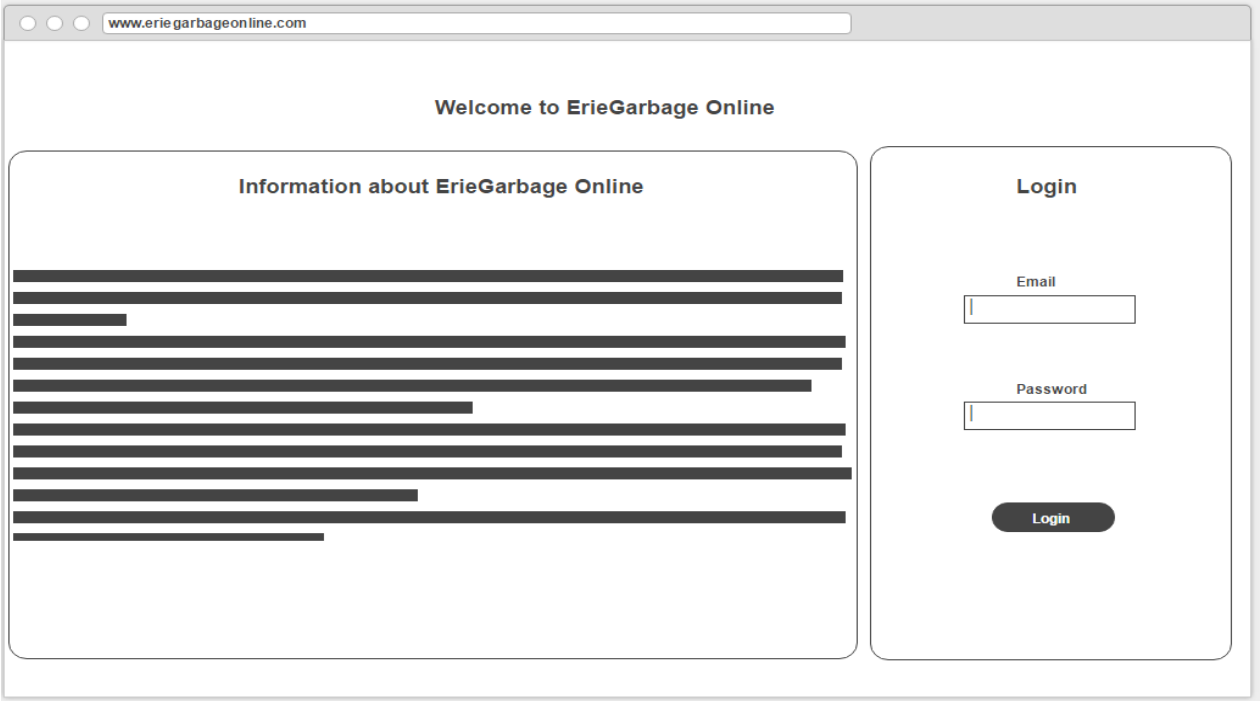

Figure 3.1.1 (1) - Login Interface

The figure below demonstrates what the customer will see once logged in to EGO. The customer will see their main options on the left. When an option is clicked on, the information inside the box under “Customer Control Panel” will change. This allows the user to stay in the same familiar view while having the ability to use all of their required functionality. Note that the tab on the left, “Account Information”, will allow the user to use multiple functionalities at once, such as editing/viewing their account information, viewing their garbage pickup time, and updating their billing information. They will also have the option to logout, which is not shown in the figure.

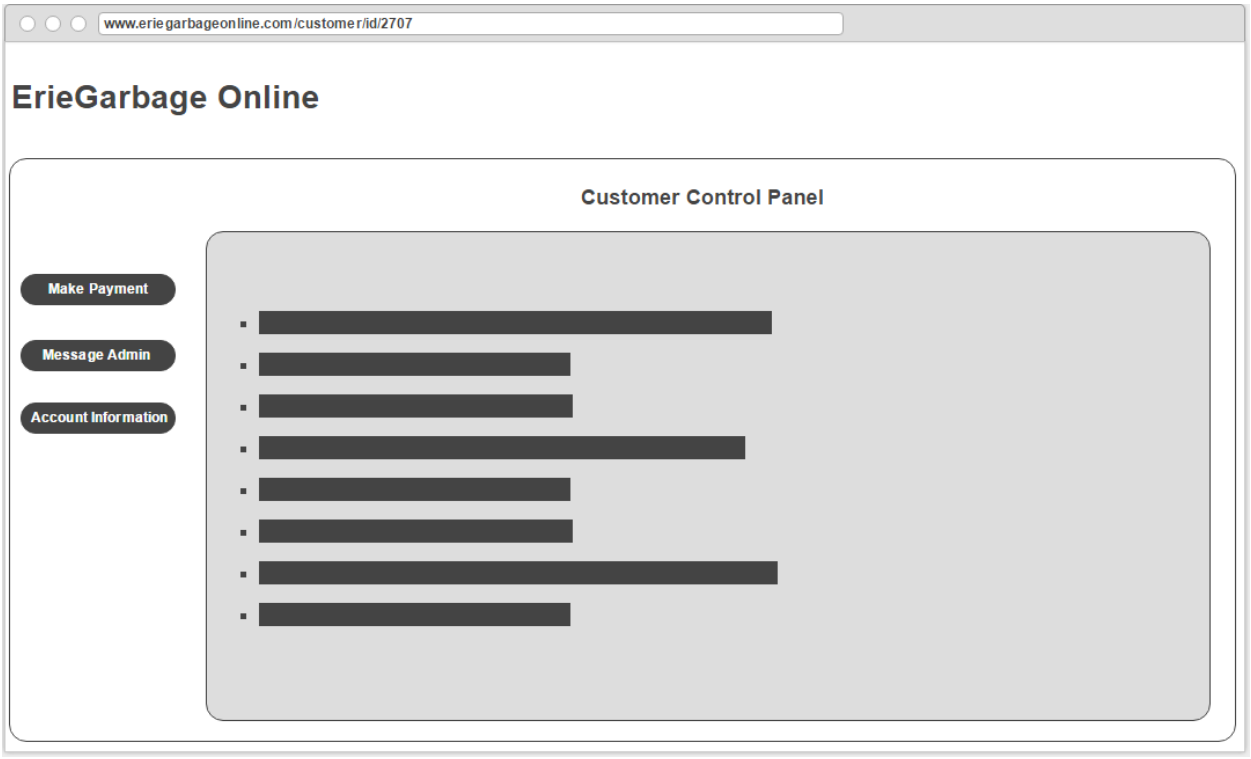

Figure 3.2.1 (2) - Customer View

The figure below shows what the administrator will see when logged into EGO. Similar to customers, all admin functionality will be completed in one familiar view. They will have the option to compose emails to send to customers, check the current due payments, view messages (complaints, billing disputes, suspension requests) from customers, and create other admin accounts. Like customer, some functionality may be grouped into a single tab and there will be a logout option present.

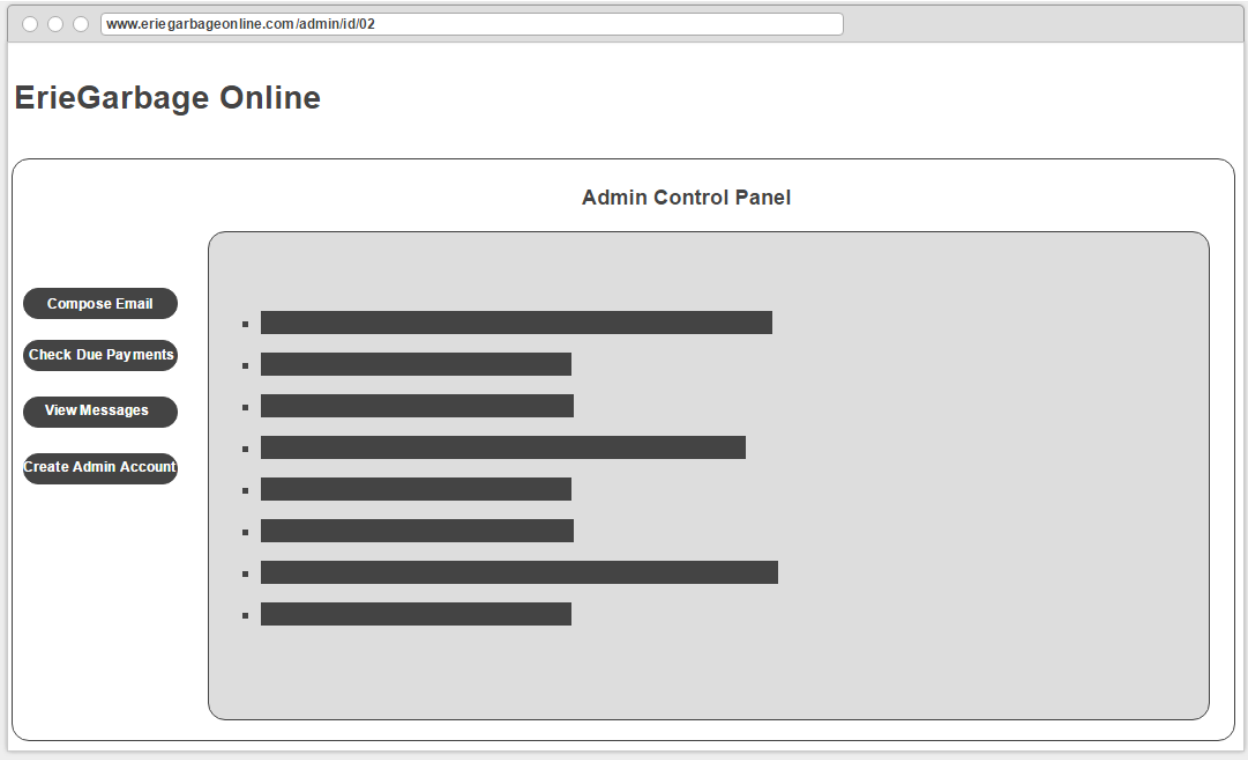

Figure 3.1.1 (3) - Admin View

3.1.2 Software Interface

The ErieGarbage system will be designed for a large number of users, and as such will need access to a database to store a large amount of information. ErieGarbage Online will communicate with Microsoft SQL Server in order to store the information. ErieGarbage is a paid service, so it will interface with payment systems such as banks or PayPal. ErieGarbage will also display information through web browsers for the user to access the system.

3.1.3 Hardware Interface

The interface with hardware will consist of networking and the use of servers that will connect any user to the pages of the website.

3.2 Module Interface Diagrams

The MVC architectural pattern consists of three components: Model, View, and Controller. Each component performs its own job while interfacing with the other two components. To do this, each component will allow the other components to access its public methods. These public methods represent the interface between components and how the data will flow between them. Each component of the MVC architectural pattern will be described in more detail below.

3.2.1 Model Interface

The model interface is called whenever the controller is requested by the user and validates that the user can add, manipulate, and retrieve data from the model. The model's main interface is the *Model* class in EGO's structural diagram.

3.2.2 View Interface

The view interface 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. Every user of EGO will see multiple views within the system. The main interface of this component is the *View* class in EGO's structural diagram.

3.2.3 Controller Interface

The controller interface is needed to allow the user to obtain information about the model. The main interface of the controller is a class called *Controller* in EGO's structural diagram. This class is the parent class to three children classes, which are AdminController, LoginController, and CustomerController.

3.3 Dynamic Models of System Interface

The following sequence diagrams are used to represent interaction between the user and the *View* of ErieGarabage Online. Please keep in mind, there are multiple views that the user can potentially access. Each sequence diagram should make it clear which view is being used by the name indicated under the "<<boundary>>" stereotype.

3.3.1 User Logs in

In the following sequence diagram, the interaction between the user and the boundary (or view) is shown. The user will initially enter his username and password and then attempt to login to EGO. From there, the view will submit the info (with an encrypted password) on to the login controller. The login controller will verify the login status of the user to ensure he is not logged in elsewhere, validate the structure of the username and password to help prevent malicious input, and will send a request to the database to retrieve the user's data as long as everything checks out to be correct and valid. The view updated accordingly and the user's status will be changed to logged in.

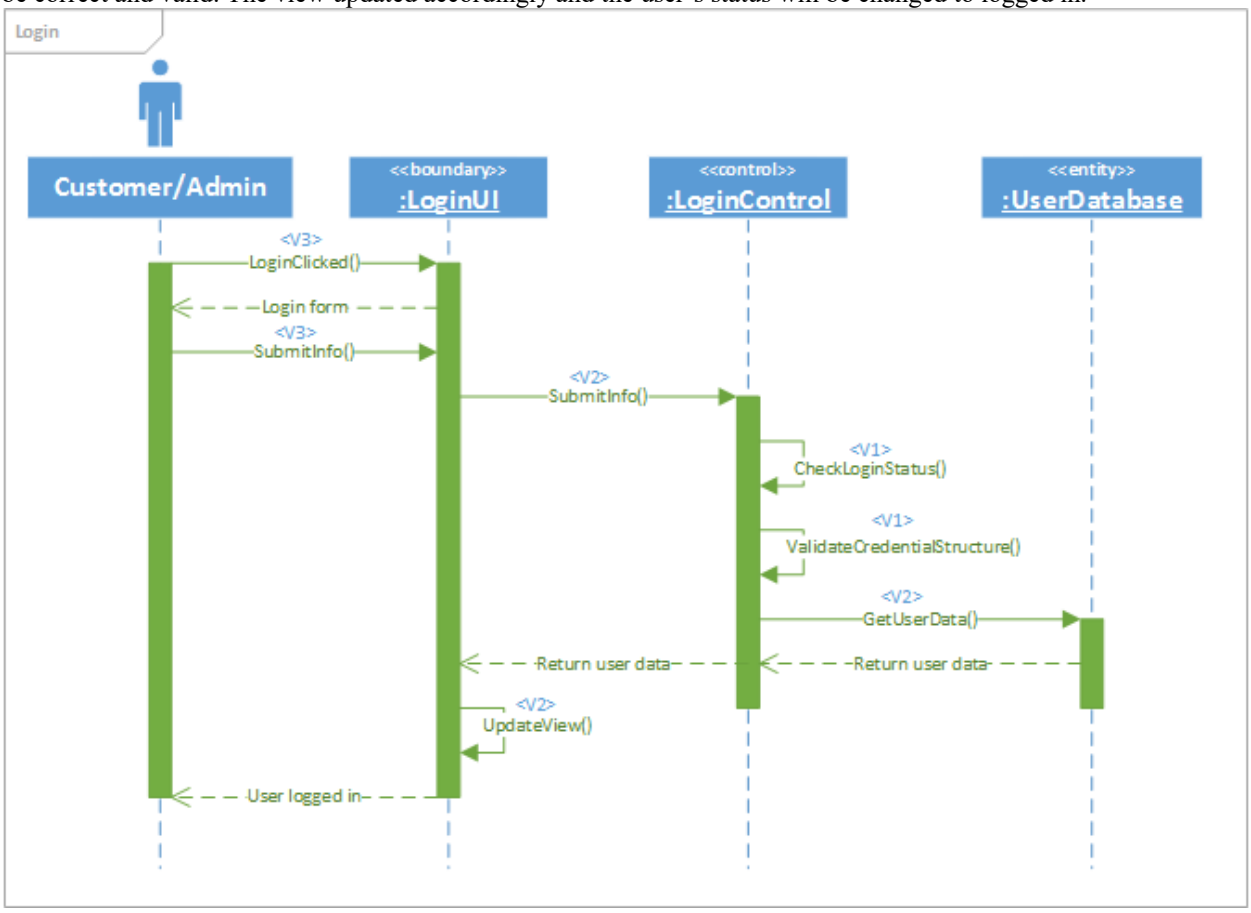

Figure 3.3.1 - User Logs in

3.3.2 Admin Emails a Customer

In the following sequence diagram, the administrator creates and sends an email to a customer's email account. The admin interacts with the view to create an email. The administrator will need to fill out the message and submit it to the system. Once submitted, the view will pass along the email to the email controller. Within the controller, the admin's status will be checked to see if he is logged in or not. If so, the email will be sent to the customer. The controller will also interact with ErieGarbage's email database to store the message. The admin will receive a notification that the email has been successfully sent out.

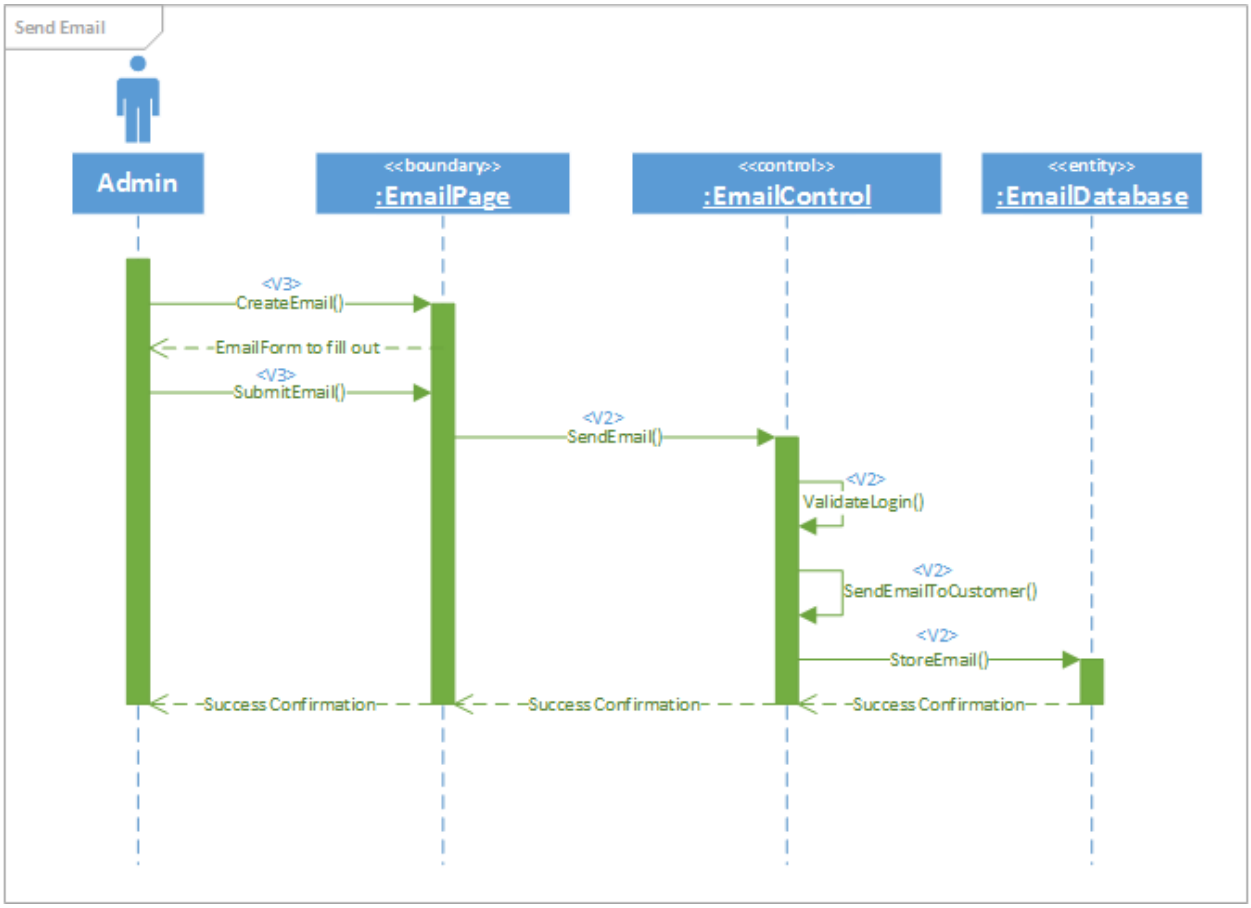

Figure 3.3.2 - Admin Emails a Customer

3.3.3 Admin Views Messages From Customers

In the following sequence diagram, an admin interacts with the view to see a list of messages sent from customers. The admin requests to view a list of messages from customers by using the system's view component. The view will pass along that request to the message controller, which will then validate that the admin is logged in before proceeding. Once validated, a list of user messages will be retrieved from the database by the controller. The view will be updated and the admin will be able to view the list of messages.

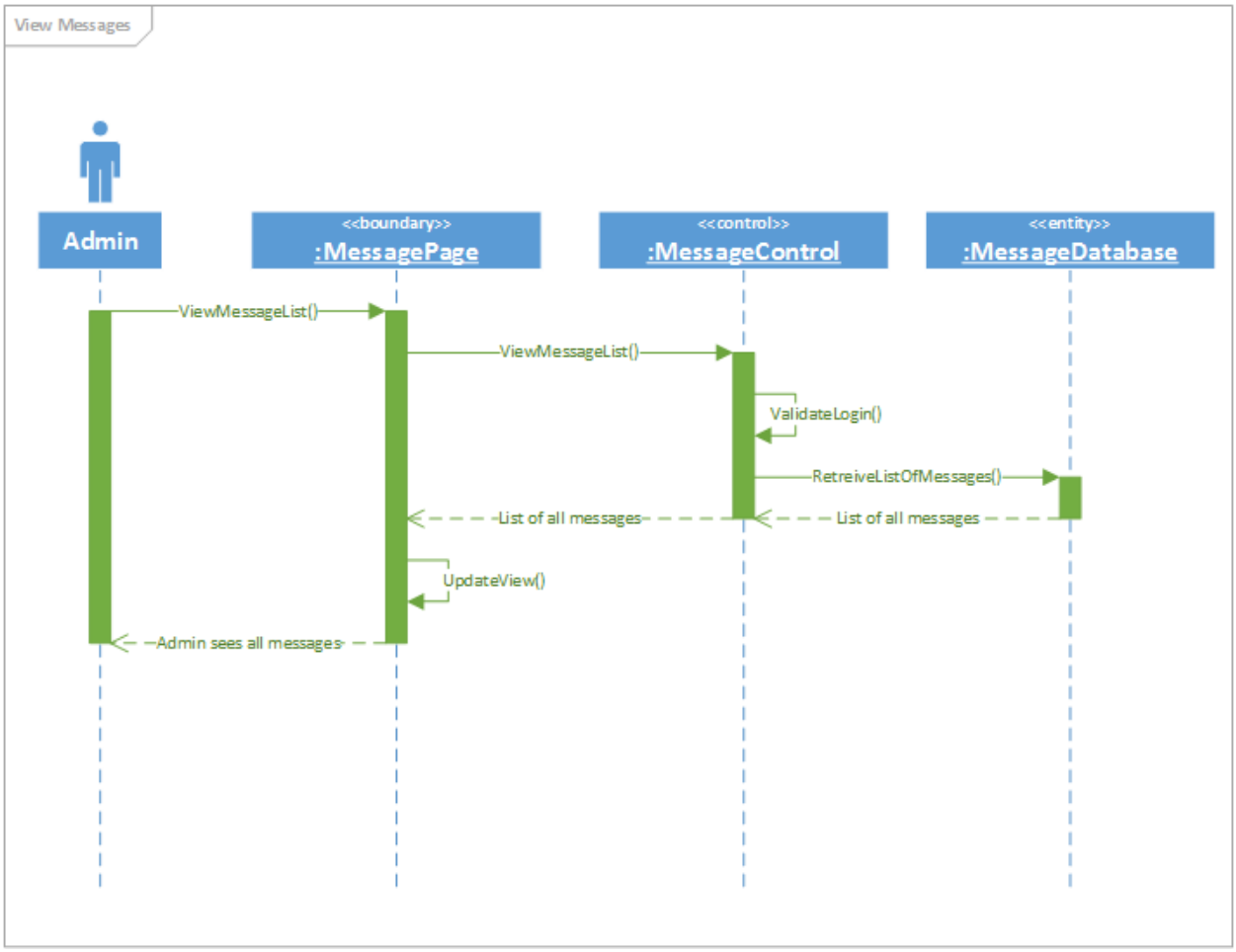

Figure 3.3.3 - Admin Views Messages From Customers

4. Architectural Risk Analysis

4.1 Software Characterization

ErieGarbage Online Architectural Diagram

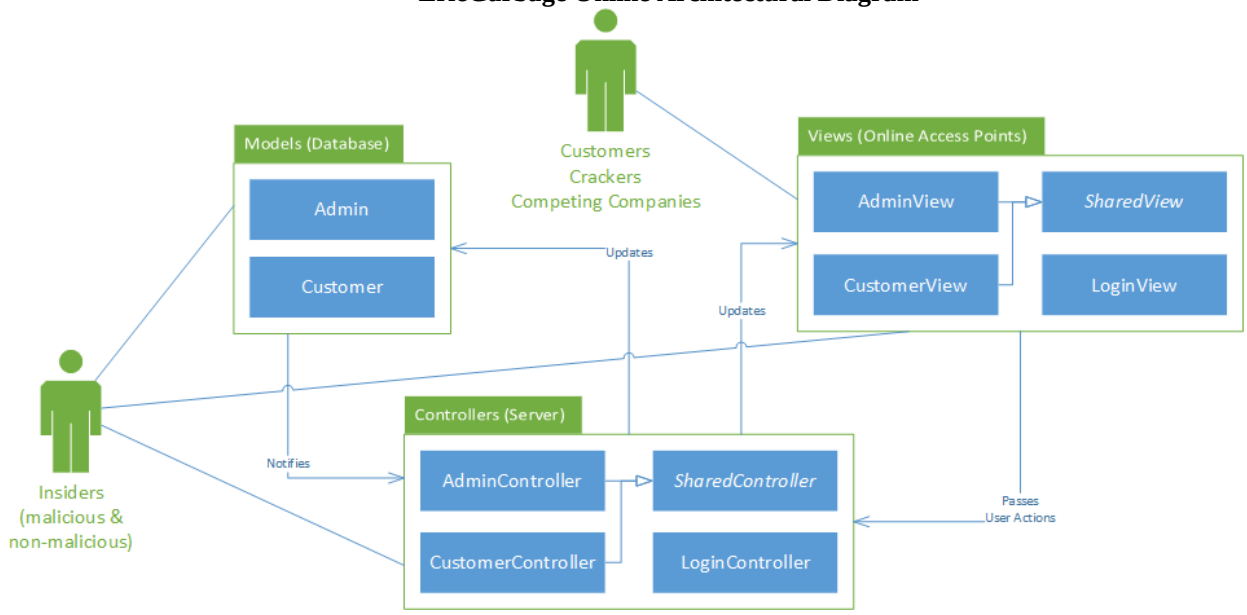

Figure 4.1 - Architectural Diagram

The above images represents ErieGarbage Online's high-level architectural design. Since EGO makes use of the MVC architectural design pattern, users of the system will view and manipulate data through the frontend, which is shown in the "Online Access Points" package above. This View is accessed through the internet by a user's browser. This is likely to be the most common area of potential attacks due to all users of EGO having access to this part of the system. Attacks are most likely to come from external attackers; however, internal attackers have the ability to compromise every component in the system shown above.

4.2 Threat Analysis

The threat analysis section of the Architectural Risk Analysis methodology is concerned with identifying the relevant threats to the ErieGarbage Online system. During this section, a given level of access and skill level of attackers may be assumed. These potential threats of attack will be mapped to vulnerabilities to further explore how the software may be exploited by attackers.

Threat Identification & Characterization

Threat Source	**Motivation**	**Threat Actions**
Customer	● Angry about company policy or customer support ● Misuse of software, accidents	● Spoofing identity ● Repudiation
Cracker	● Challenge ● Rebellion ● Ego ● Monetary gain ● Vigilantism	● Spoofing identity ● Tampering with data ● Repudiation ● Denial of service
Insiders (malicious)	● Ego ● Intelligence ● Monetary gain ● Revenge ●	● Tampering with data ● Information disclosure ● Elevation of privilege ● Repudiation
Insiders (non-malicious)	● Non-malicious errors ● Lack of training	● Tampering with data ● Information disclosure ● Elevation of privilege ● Repudiation
Competing Companies	● Learn confidential information ● Blackmail ● Economic espionage ● Destroy information ● Study internal tactics ● Gain a competitive advantage	● Spoofing identity ● Tampering with data ● Denial of service

Table 4.2 - Threat Identification & Characterization

The **Threat Identification & Characterization** table above shows the five main groups of attackers that ErieGarbage Online faces once deployed.

4.3 Architectural Vulnerability Assessment

The architectural vulnerability assessment section of the architectural risk analysis methodology is concerned with the preconditions that must be present for vulnerabilities of ErieGarbage to be exploited. This section will also assess the states that the software may enter upon successful system exploitation. This vulnerability assessment section will be composed of three main activities: attack resistance analysis, ambiguity analysis, and dependency analysis. Applying these activities to ErieGarbage Online will help identify vulnerabilities and threats of the system, whether they are malicious or non-malicious in nature.

Attack Resistance:

In this subsection, we will apply a list of known attacks and calculate the risk-based impact that these threats pose to ErieGarbage Online. Below is a list of weaknesses that EGO currently is susceptible to.

List of Applicable Weaknesses

Type	**Definition**
CWE-396 - Declaration of Catch for Generic Exception	Catching overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities
CWE-288 - Authorization Bypass Using an Alternate Path or Channel	A product requires authentication, but the product has an alternate path or channel that does not require authentication
CWE-425 - Direct Request	The web application does not adequately enforce appropriate authorization on all restricted URLs, scripts, or files
CWE-285 - Improper Authorization	The software does not perform or incorrectly performs an authorization check when an actor attempts to access a resource or perform an action
CWE-321 - Use of Hard-coded Cryptographic Key	The use of a hard-coded cryptographic key significantly increases the possibility that encrypted data may be recovered
CWE-311 - Missing Encryption of Sensitive Data	The software does not encrypt sensitive or critical information before storage or transmission
CWE-287 - Improper Authentication	When and actor claims to have a given identity, the software does not prove or insufficiently proves that the claim is correct
CWE-209 - Information Exposure Through an Error Message	The software generates an error message that includes sensitive information about its environment, users, or associated data

CWE-89 - Improper Neutralization of Special Elements Used in a SQL Command	The software constructs all or part of a SQL command using externally influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component
CWE-522 - Insufficiently Protected Credentials	This weakness occurs when the application transmits or stores authentication credentials and uses an insecure method that is susceptible to unauthorized interception and/or retrieval
CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion')	The software does not properly restrict the size or amount of resources that are requested or influenced by an actor, which can be used to consume more resources than intended.

Table 4.3 (1) - List of Applicable Weaknesses

As a web-based application, ErieGarbage Online is susceptible to common internet attacks. By mapping the different threats to weaknesses in the STRIDE model below, a visualization of the overall vulnerability of the system can be created.

STRIDE Model

Type	**Definition**	**Mapped Weakness**
Spoofing identity	An example of identity spoofing is illegally accessing and then using another user's authentication information, such as username and password.	CWE-288 - Authorization Bypass Using an Alternate Path or Channel CWE-287 - Improper Authentication CWE-522 - Insufficiently Protected Credentials
Tampering with data	Data tampering involves the malicious modification of data. Examples include unauthorized changes made to persistent data, such as that held in a database, and the alteration of data as it flows between two computers over an open network, such as the Internet.	CWE-89 - Improper Neutralization of Special Elements Used in a SQL Command CWE-287 - Improper Authentication CWE-311 - Missing Encryption of Sensitive Data
Repudiation	Repudiation threats are associated with users who deny performing an action without other parties having any way to prove otherwise—for example, a user performs an illegal operation in a system that lacks the ability to trace the prohibited operations.	CWE-425 - Direct Request
Information disclosure	Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it—for example, the ability of users to read a file that they were not granted access to, or the ability of an intruder to read data in transit between two computers.	CWE-209 - Information Exposure Through an Error Message CWE-425 - Direct Request CWE-285 - Improper Authorization
Denial of service	Denial of service (DoS) attacks deny service to valid users—for	CWE-400: Uncontrolled Resource Consumption ('Resource

	example, by making a Web server temporarily unavailable or unusable. You must protect against certain types of DoS threats simply to improve system availability and reliability.	Exhaustion')
Elevation of privilege	In this type of threat, an unprivileged user gains privileged access and thereby has sufficient access to compromise or destroy the entire system. Elevation of privilege threats include those situations in which an attacker has effectively penetrated all system defenses and become part of the trusted system itself, a dangerous situation indeed.	CWE-285 - Improper Authorization CWE-425 - Direct Request

Table 4.3 (2) - STRIDE Model

Ambiguity analysis:

In this subsection, we attempt to find any ambiguities in the requirements of the ErieGarbage Online system and clarify them. This will help to eliminate misunderstandings between the business requirements of the software and the programmer's implementation of the design in code. Functions, structures, properties, and policies of the software that may lead to misunderstandings or a variety of interpretations will be cleared up in this section.

Firstly, we will examine the functions of the program specified by the ErieGarbage company. ErieGarbage has provided many business requirements and some may have the potential to be misunderstood by the developers of the EGO system.

Admin requirement ambiguities:

- Checking customer accounts that have payments due
    - Admins will be able to view all of the accounts that have a negative balance on their account. A negative balance indicates that the customer has a payment that has not been paid for. The system will pull all of the customers from the database which have a negative balance and return them to the admin's view.
- Viewing and responding to customer complaints, bill disputes, and suspension requests
    - Customers are able to contact admins for a variety of reasons. When an admin attempts to view messages, the system will retrieve a list of messages that have not been responded to and provide them to the admin.
    - When the admin selects a message to view, the system will provide the admin a way to respond which is determined based on the message type.
    - When responding to a complaint, the admin will simply send an email back to the customer, which will mark the complaint as complete. The customer can reply to the admin if they are unsatisfied with the response.
    - When responding to bill disputes, the admin is provided with a copy of the bill that the customer is taking issue with. According to ErieGarbage guidelines, the admin can choose to cancel the bill or not. The response is sent as an email to the customer.

    - When viewing a suspension request, the admin will determine whether a customer's account is eligible for suspension. If the admin chooses to suspend the account, the customer will receive a notification email stating that their request has been granted.

Customer requirement ambiguities:

- Make payments for the service
    - Customers have an overall balance for their account. If this balance is negative, they have an unpaid expense or fee. When making a payment, the customer can choose to pay more than the negative balance in order to prepay future bills. If a charge is made to the customer's account, the amount is deducted from their balance automatically.
    - Customers are notified when charges are made and their current balance after the charge.
    - In order to make a payment, the customer must provide payment information which will be verified before a payment is made. The customer can choose to store the information for future payments.
- Messaging an admin with a complaint, suspension, or bill disputes
    - While using the service, if the user decides to contact an admin, they will select one of the types. The system will return the required information for the customer to provide.
    - When filing a complaint, the customer will add what they have a complaint about and provide details. This will be added to a database that the admin will be able to view from when they view messages.
    - When requesting a suspension, the user will provide the reason for the suspension as well as the length of time. The system will validate that the length of time is valid before sending to admins.
    - When disputing a bill, the customer will select the bill they wish to dispute as well as provide the issue they have with the bill.
- Cancelling their own account
    - When cancelling an account, a permanent suspension will be placed on the account. In case they ever decide to rejoin, the information will be kept to provide easier signup.
    - Customers must provide a piece of personal information to cancel the account, in order to prevent accidental cancels.

Shared functionality requirement ambiguities:

- Viewing garbage collection pick up time
    - Customers have the ability to view their weekly pickup times. The system will use the customer's location to determine which route they belong to and display the time at which the route will be at their location
    - Admins have the ability to view all customers' pickup times. They have the ability to view as a table, or they can choose a route and view all customers along the route. When selecting a customer on the route they can view specific times of pickup.
- Updating account information
    - Not all information can be updated for a customer. When attempting to update, they will only be able to view the items they are able to update.
- Updating billing information
    - When updating billing information, the credit card details are not accessible by the user attempting to view it. This information will not be sent to the front end; to update it, it must be resubmitted to the server.

Next, we will look at the ambiguities in the architecture and design phase of ErieGarbage Online. Ambiguities in the design portion of the project are important to clear up with the developers so that long term effects of these ambiguities are minimized as much as possible.

Architectural and design ambiguities:
The architecture shown in **Figure 4.1** depicts the controller running all of the code execution, however, JavaScript will need to provide functionality to the view in order to pass actions to the controller. Interactions between the components must be validated before and after sending through the internet, in order to mitigate any possible errors. For the models, they represent both an object representing data, as well as the database itself.

Dependency Analysis:

In this subsection, we will analyze the potential risks caused by dependencies of the ErieGarbage Online system. EGO's dependencies are as follows:

- Server operating system - Windows 10
- Storage of users and other system data - Microsoft Access
- Functions executing in the front end - JavaScript
- Microsoft .NET framework

This table maps vulnerabilities to the each 3rd party software that EGO will take advantage of. These vulnerabilities should be accounted for in this documentation and during the implementation of the project.

Software	**Vulnerabilities**
Microsoft Windows 10	CWE 20 Improper Input Validation CWE 284 Access Control (Authorization) Issues CWE 119 Failure to Constrain Operations within the Bounds of a Memory Buffer
Microsoft Access	CWE 94 Failure to Control Generation of Code ('Code Injection') CWE 119 Failure to Constrain Operations within the Bounds of a Memory Buffer
JavaScript	CWE 79 Failure to Preserve Web Page Structure ('Cross-site Scripting')
Microsoft .NET Framework	CWE 200 Information Exposure CWE 20 Improper Input Validation CWE 79 Failure to Preserve Web Page Structure ('Cross-site Scripting')

Table 4.3 (3) - Vulnerabilities of 3rd Party Software

4.4 Risk Likelihood Determination

In the following table **Risk Likelihood Table**, each risk is ranked on the motivation of attacks, impact if an attack were to happen, and the effectiveness of controls in place.

Risk Likelihood Table

Risk	Threat	Motivation & Capability	Severity of Weaknesses	Likelihood of Control Breach	Likelihood
Spoofing identity	Customer	Low - They will most likely use their own information	High - Many vulnerabilities which have high likelihood of attack	Low - Two-factor authentication requiring users to use a device they keep on them to log in	Low
	Cracker	High - They would like to break into an admin account to steal information			Medium
	Competing Companies	High - Gaining admin information would allow them to see complaints users have, which they could target as customers			Medium
Tampering with data	Cracker	SQL injection or gaining access to a database is a very motivating weakness, there is a lot to gain for an attacker.	High - A couple weaknesses with high likelihood of attack	Low - SQL parameterization and checking authentication before any action will help mitigate an attack from happening	Medium
	Insiders (malicious)	High - They have direct access to the in house systems with the intention of doing harm			Medium

	Insiders (non-malicious)	Low - Any mistake would be accidental and there would be no motivation			Low
	Competing Companies	High - Sabotaging the system's data gives them a major competitive edge			Medium
Repudiation	Customers	High - If a customer is angry with the service, they may claim that the website did something that was incorrect	Low - Small number of weaknesses	Low - Automatic logging	Low
	Insiders (non-malicious)	Low - Most employees will not attempt to abuse the system			Low
	Insiders (malicious)	High - Malicious insiders will want to destroy systems without their hand in it being recorded			Low
Information disclosure	Insiders (non-malicious)	Low - Disclosed information will be an accident and oversight in development/design	High - Many vulnerabilities	Low - Logs are viewable by admins only, code reviews before builds	Low
	Insiders (malicious)	High - Release too much information will damage the			Medium

		service and company's image			
Denial of service	Crackers	High - DDOS attacks are very common for crackers to do just for fun	Low - Small weaknesses	Low - DDOS protection, resource controls	Low
	Competing Companies	High - Bringing a competitor's service down will likely boost use of the attackers			Low
Elevation of privilege	Insiders (non-malicious)	Low - An insider would have little reason to gain extra privilege without malicious intent	Low - medium number of weakness with low likelihood	Low - Customer and admin accounts are kept entirely separate, and the privileges are separated between the account types	Low
	Insiders (malicious)	High - A malicious insider granting himself higher privilege would be able to have more control over the system			Low

Table 4.4 - Risk Likelihood Table

4.5 Risk Impact Determination

In the following **Risk Impact Table** the risks are measured based on the assets that they threaten, the business impact if an attack is successful, and the locality of the risk in order to determine the overall impact of the risk.

Risk Impact Table

Risk	Threatened Assets	Business Impact	Locality	Impact
Spoofing identity	Low - The user whose account is spoofed will have some of their information compromised, however billing information would remain undisclosed	Low - Every action performed is logged and can be undone, so impact is low	Low - Only a single or handful of users will be affected by each successful attack	Low
Tampering with data	High - Any database that EGO uses would be at risk for having data modified	High - The databases are essential to EGO operation	High - The potential to delete all data from the database would compromise the entire system	High
Repudiation	Low - User confidence in the system, system integrity	High - Not keeping track of user actions would allow an attacker to take advantage of the systems in place	High - If the system is not in place for a single user, then every user is affected	Medium
Information disclosure	High - User information and system functionality exposed	High - Could affect EGO if attackers use this information	High - Effects would be seen throughout the system, would increase knowledge of system to attacker	High
Denial of service	High - Functionality of the website	High - Would prevent use of the site for the duration of the attack	High - Would affect the entire system from being used	High
Elevation of privilege	High - User information,	Low - Admin modifications can	High - Admins have a wide degree of	Medium

	account status	be undone	influence over the system's users	

Table 4.5 - Risk Impact Table

4.6 Risk Mitigation Planning

The following **Risk Exposure Table** allows rank the risks based on a combination of their likelihood and impact. Depending on the exposure level, more priority should be given to the risk during development.

Risk Exposure Table

Risk	Threat	Likelihood	Impact	Exposure Level
Spoofing identity	Customer	Low	Low	Low
	Cracker	Medium		Low
	Competing Companies	Medium		Low
Tampering with data	Cracker	Medium	High	High
	Insiders (malicious)	Medium		High
	Insiders (non-malicious)	Low		Medium
	Competing Companies	Medium		High
Repudiation	Customers	Low	Medium	Low
	Insiders (non-malicious)	Low		Low
	Insiders (malicious)	Low		Low
Information disclosure	Insiders (non-malicious)	Low	High	Medium
	Insiders (malicious)	Medium		High
Denial of service	Crackers	Low	High	Medium
	Competing Companies	Low		Medium

Elevation of privilege	Insiders (non-malicious)	Low	High	Medium
	Insiders (malicious)	Low		Medium

Table 4.6 (1) - Risk Mitigation Planning

Below is our **Risk Mitigation Plan** which lays out the priority of each risks vulnerabilities, and steps to take which will mitigate attacks that go through the weakness.

Risk Mitigation Plan by Priority

Priority	Risk	Steps
High	Tampering with data	CWE-89 - Use a framework that doesn't allow this weakness to occur, enforce separation of data and code CWE-311 - Use industry-approved techniques and separate sensitive and non-sensitive data as much as possible
	Information disclosure	CWE-425 - Apply appropriate access control authorizations for each access CWE-285 - Use a framework that doesn't allow the weakness to occur, and make sure that access control is enforced on server side for every page CWE-209 - Handle exceptions internally and display minimal details to the user
Medium	Denial of service	CWE-400: Add throttling mechanisms that limit the effect a user can have on resource consumption
	Elevation of privilege	CWE-288 - Funnel all access through a single choke point to simplify how users can access a resource, and keep track of all activity CWE-425 - Apply appropriate access control authorizations for each access, have a user provide confirmation for each access CWE-285 - Use a framework that doesn't allow the weakness to occur, and make sure that access control is enforced on server side for every page
Low	Spoofing identity	CWE-288 - Funnel all access through a single choke point to simplify how users can access a resource, and check if user has permissions to the resource every time they access it CWE-287 - Use an authentication framework or library during implementation

		CWE-522 - Make appropriate use of cryptography
	Repudiation	CWE-425 - Apply appropriate access control authorizations for each access, keep track of all activity for each access

Table 4.6 (2) - Risk Mitigation Plan by Priority

5. Internal Module Design

Below is the overall class diagram of EGO, represented by **Figure 5**. This class diagram shows all three components of the MVC design pattern (Model, View, and Controller) as well as the Utility package that will be used to manage external systems (making payments, providing a messaging service, and encryption). This is our detailed design view, which will be split up by component in the later sections of this document. Those later sections will provide more detail on each component and its purpose within the system.

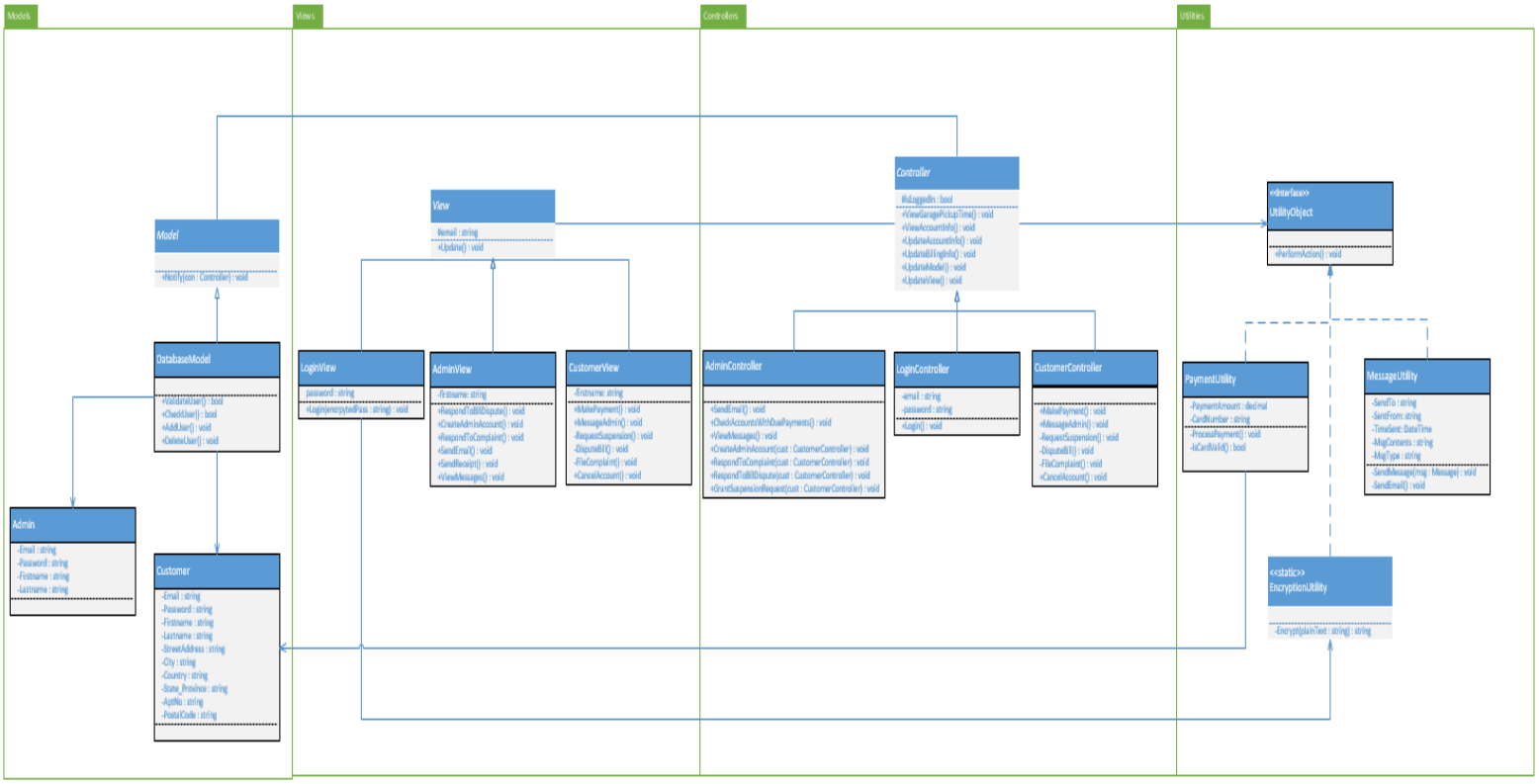

Figure 5 - ErieGarbage Online Class Diagram

5.1 Module <Model>

The model is the part of the system that the user modifies and views. The user takes advantage of the controller component to interact with the model, which consists of data.

5.1.1 Model Class Diagram

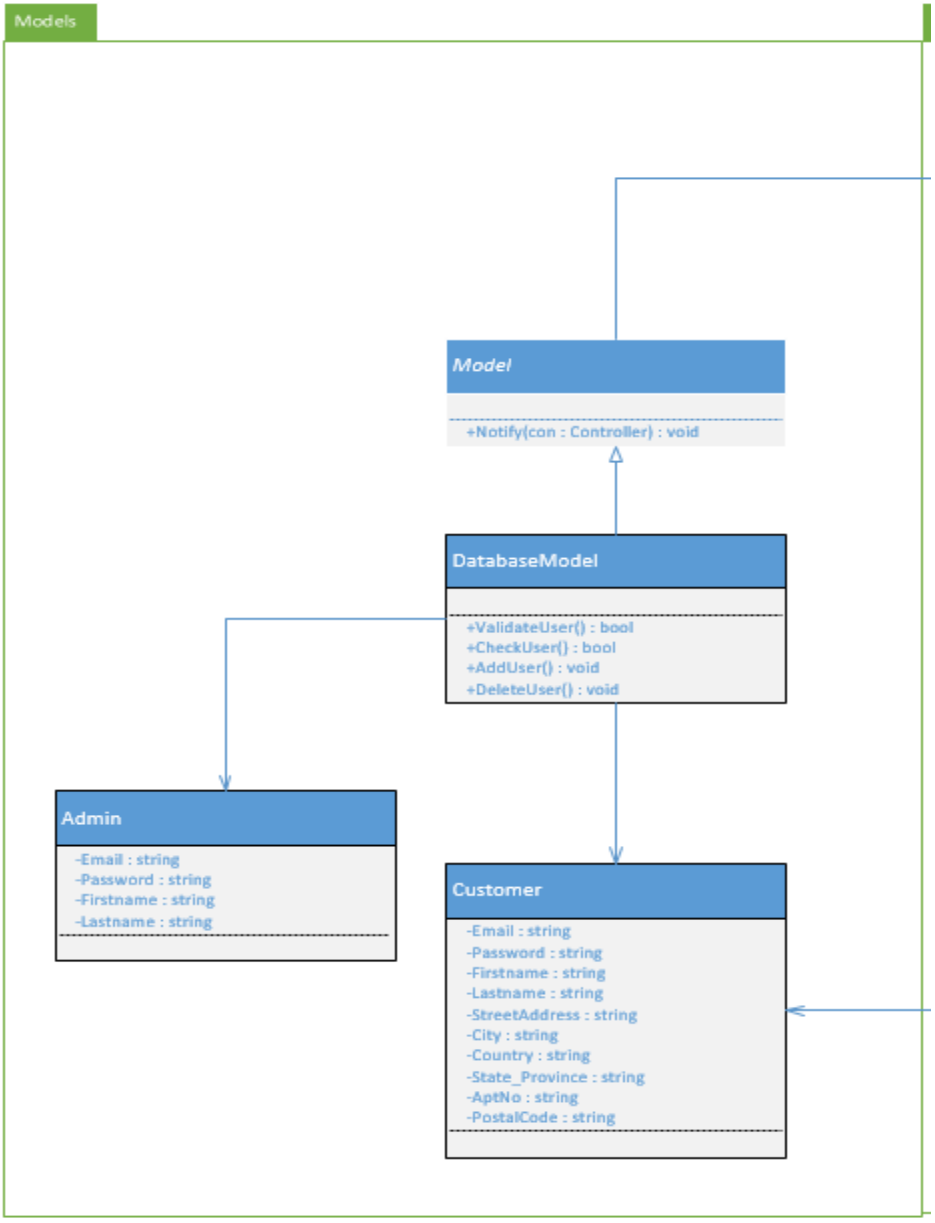

Figure 5.1 - Model

5.1.2 Model Classes

Class Descriptions

Class name	DatabaseModel
Description	Used to access the user database
Attributes	None
Methods	+public bool ValidateUser() +public bool CheckUser() +public void AddUser() +public void DeleteUser()

Method Descriptions

Method name	CheckUser()
Class name	DatabaseUtility
Returns	True if the user exists in the database, false is the user does not exist
Input	User email address
Output	True or false

Pseudo Code	BEGIN
	
	for each user in the database
	  if (currentIndex.Email == specifiedUserEmail)
	  then return true
	
	return false
	
	END

5.2 Module<View>

The View module is what the user will see, whether the user is a customer or an admin. The view is used to give a level of abstraction to the user. The user can modify data by working through the controller, however, to the user, it appears that they are working with the image displayed to them. This makes it much easier for the user to interact with the system at a high-level.

5.2.1 View Class Diagram

Below is the class diagram for the View module. This class diagram does not represent all functionality but allows us to understand the concept of the view. Below, we will outline the LoginView class.

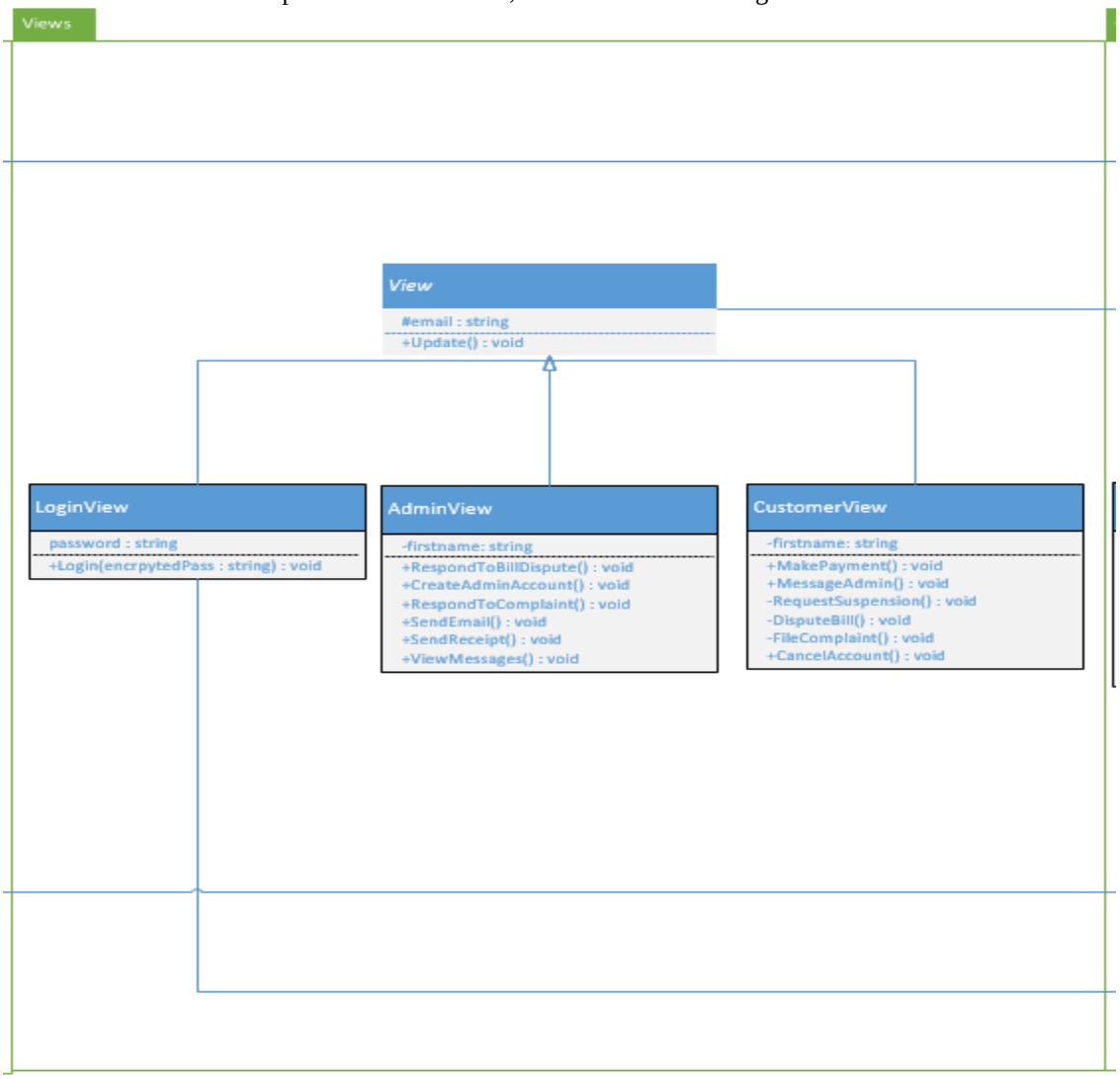

Figure 5.2 - View

5.2.2 View Classes

Class Descriptions

Class name	LoginView
Description	The page all users will see when they attempt to login
Attributes	-password : string
Methods	+Login(encryptedPass : string) : void

Method Descriptions

Method name	UpdateView()
Class name	LoginView()
Returns	Nothing - updates the user's view
Input	Nothing
Output	New view

Pseudo Code	BEGIN
	
	if (the user is logged in)
	  if (user is an admin)
	  go to the admin page
	  else
	  go to the customer page
	
	END

5.3 Module<Controller>

The controller is the main interaction component in the MVC architectural pattern. This is what the user will be using to make choices and interact directly with the system. The user will be able to retrieve and modify data using the controller.

5.2.1 Controller Class Diagram

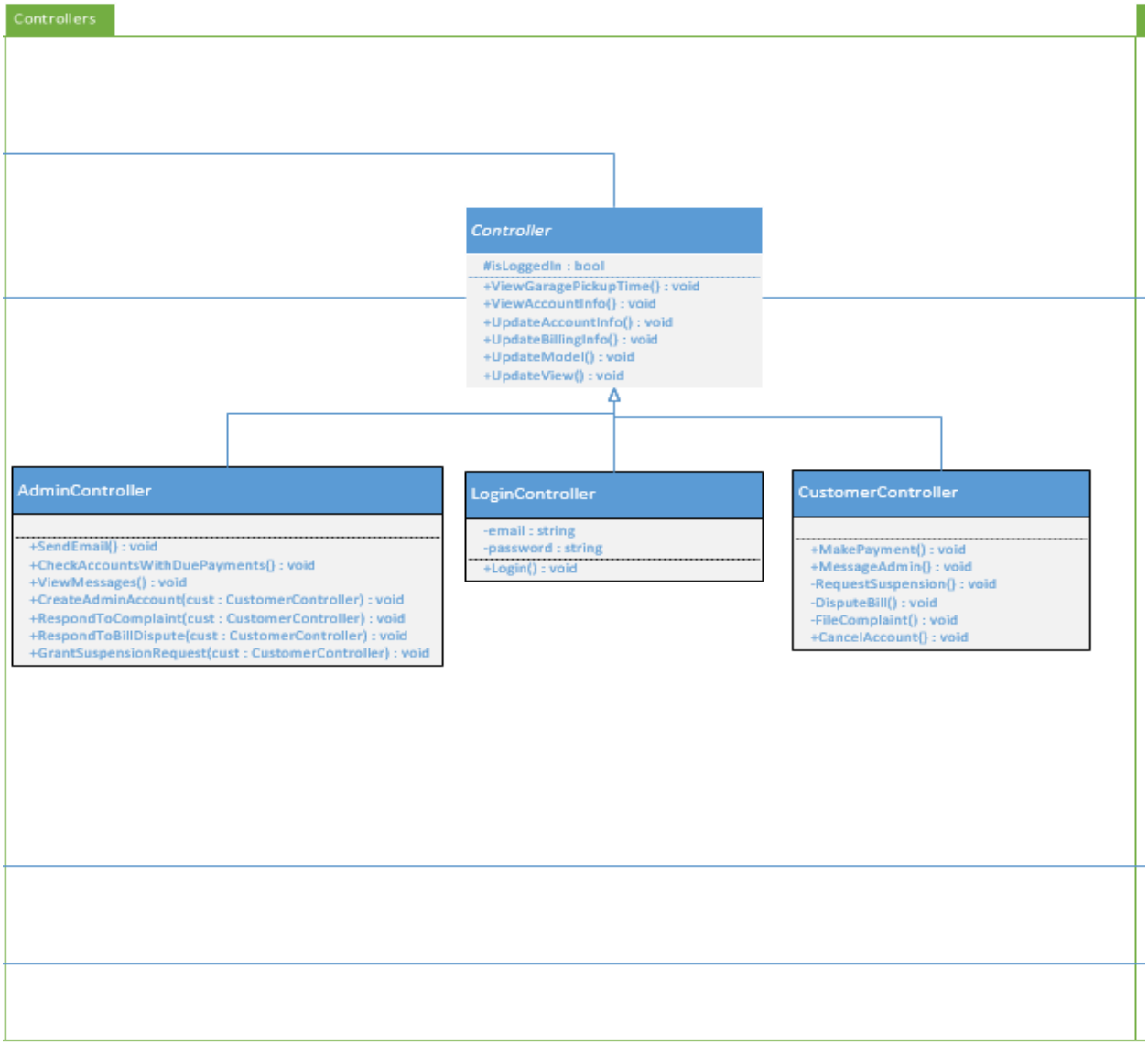

Figure 5.3 - Controller

5.3.2 Controller Classes

Class Descriptions

Class name	AdminController
Description	Class that an admin user will use to interact with the system
Attributes	private AdminView adminView
Methods	• public void SendEmail() • public void CheckAccountsWithDuePayments() • public void ViewMessages() • public void CreateAdminAccount() • public void RespondToComplaint() • public void RespondToBillDispute() • public void GrantSuspensionRequest()

Method Descriptions

Method name	SendEmail()
Class name	AdminController
Returns	Nothing - Sends an email to a user
Input	Subject and message body
Output	Email message to a user

Pseudo Code	BEGIN
	
	if (subject.length is acceptable && message.length is acceptable)
	then
	sendMessage(recipient, subject, body);
	else
	print “message could not be sent”
	
	END

5.4 Module<Utility>

5.4.1 Utility Class Diagram

In the image below (**Figure 5.4)**, the Utility package is shown. The purpose of the Utility package is to provide services that are outside of the scope of the system. For example, ErieGarbage Online does not aim to create an encryption algorithm or a library to handle the processing of credit cards. Instead, we will utilize outside services through these classes.

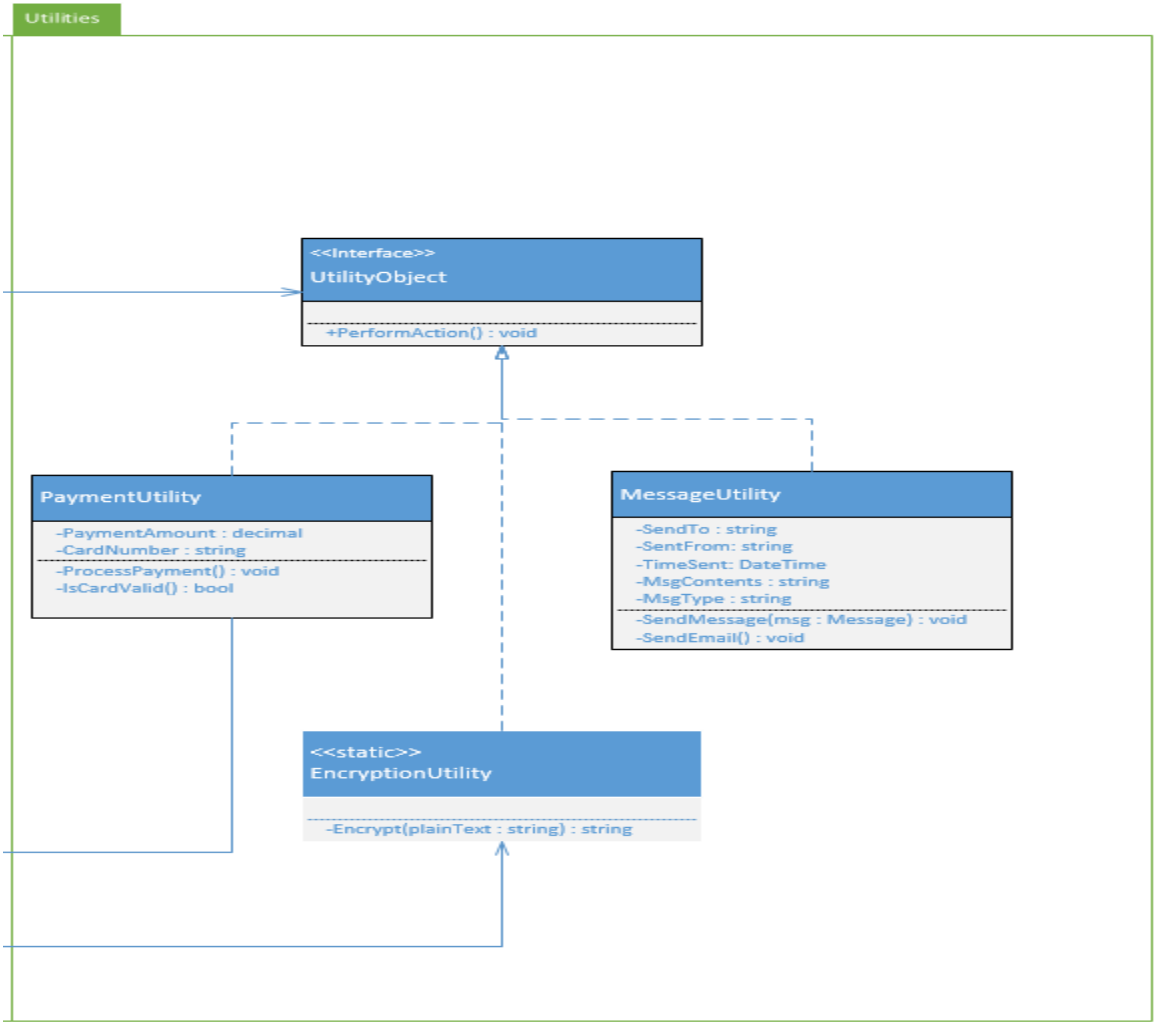

Figure 5.4 - Utility

5.4.2 Utility Classes

Class name	PaymentUtility
Description	Class that an admin user will use to interact with the system
Attributes	-PaymentAmount : decimal -CardNumber : string
Methods	+ProcessPayment() : void +IsCardValid() : bool

Method Descriptions

Method name	IsCardValid()
Class name	PaymentUtility
Returns	A boolean value specifying whether the card number is valid or not
Input	A credit or debit card number
Output	A true or false value indicating the validity of the card

Pseudo Code	BEGIN
	bool result = false;
	
	if (processor.isValid(CardNumber.length) && processor.ExistsInDb(CardNumber))
	then
	result = true;
	
	return result;
	
	END

6. Team Members Log Sheets

6.1 Jake Wheeler

Date	task	duration
10/24/2016	Created document, did some editing	.5 hrs.
10/26/2016	Added info about architecture	.5 hrs.
10/31/2016	Added more to the introductory information	1 hrs.
11/1/2016	Created architectural risk analysis section and added to it	3 hrs.
11/1/2016	Began ambiguity analysis	2 hrs.
11/2/2016	Added to ambiguity analysis and risk impact	.5 hrs.
11/3/2016	Began mapping threats to STRIDE, formatted document to make developing the internal module section easier, fixed table of contents, added a few more CWEs	1 hrs.
11/5/2016	Added to risk likelihood table, added to risk impact table, worked on internal module section, added software interface description	3 hrs.
11/6/2016	Wrap-up loose ends and print version 1.0	1 hrs.
11/10/2016	Updated section 2 - rationale and software architecture, worked on changing class diagram, added section 3.2 and section 3.3	2.5 hrs.
11/10/2016	Created mock-up images and added text descriptions to them in section 3.1.1, labeled every table and image	1.5 hrs.
11/11/2016	Updated class diagram, fixed section 5, added a javascript vulnerability	1 hrs.
11/13/2016	Final revision & submission	.5 hrs.
	Total :	18 hrs.

6.2 Nate Christiansen

date	task	duration
10/30/2016	Added info to introduction and architecture	1 hr
10/31/2016	Added more to document and created initial architectural design	1 hr
11/01/2016	Added threats to architecture diagram	.5 hrs
11/01/2016	Update some of risk likelihood and architectural design	2 hrs
11/02/2016	Updates some more risk likelihood, started impact analysis	1 hr
11/02/2016	Fixed and updated weaknesses table, accidentally deleted necessary stride table	1 hr
11/5/2016	Added to risk likelihood table, risk impact table, risk mitigation planning	3 hrs
11/10/2016	Fix all vulnerability assessment stuff	3 hrs
11/11/2016	Created a better architectural diagram, fixed most of ambiguity analysis	1.5 hrs
11/13/2016	Recreated architectural diagram because I don't know how to save things. Finished ambiguity analysis	1.5 hrs
	Total :	15.5 hrs

6.3 Alex Lee

date	task	duration
10/30/16	Implement company logo	.5 hrs
10/30/16	Rough draft of Introduction of Design Document	1 hr
	Total :	1.5 hrs