ErieGarbage Online

Design Document

Version 1.0

**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 |

1. Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, Acronyms, and Abbreviations

1.4 References

1.5 Overview

2. Architectural Design

2.1 Rationale

2.2 Software Architecture Diagram

2.3 System Topology

3. Software Interface Design

3.1 System Interface Diagrams

3.1.1 User Interface

3.1.2 Software Interface

3.1.3 Hardware Interface

4. Architectural Risk Analysis

4.1 Software Characterization

4.2 Threat Analysis

4.3 Architectural Vulnerability Assessment

4.4 Risk Likelihood Determination

4.5 Risk Impact Determination

4.6 Risk Mitigation Planning

5. Internal Module Design

5.1 Module <Model>

5.1.1 Model Class Diagram

5.1.2 Model Classes

5.2 Module<View>

5.2.1 View Class Diagram

5.2.2 View Classes

5.3 Module<Controller>

5.2.1 Controller Class Diagram

5.2.2 Controller Classes

6. Team Members Log Sheets

6.1 Jake Wheeler

6.2 Nate Christiansen

6.3 Alex Lee

Design Document

# 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.

## 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.

## 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.

## 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.

## References

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

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

## 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.

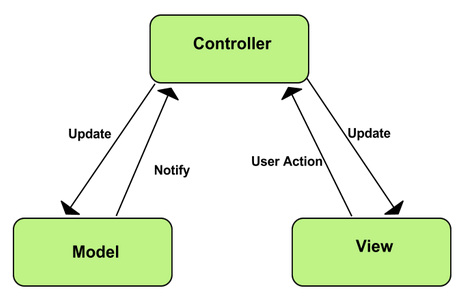
# Architectural Design

## 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.

## 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.



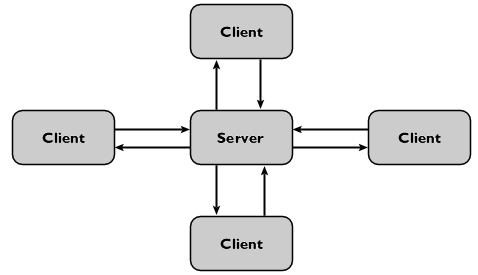
**MVC Model**

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.

## 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. 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.



# Software Interface Design

## 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.

### 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.

### 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.

### Hardware Interface

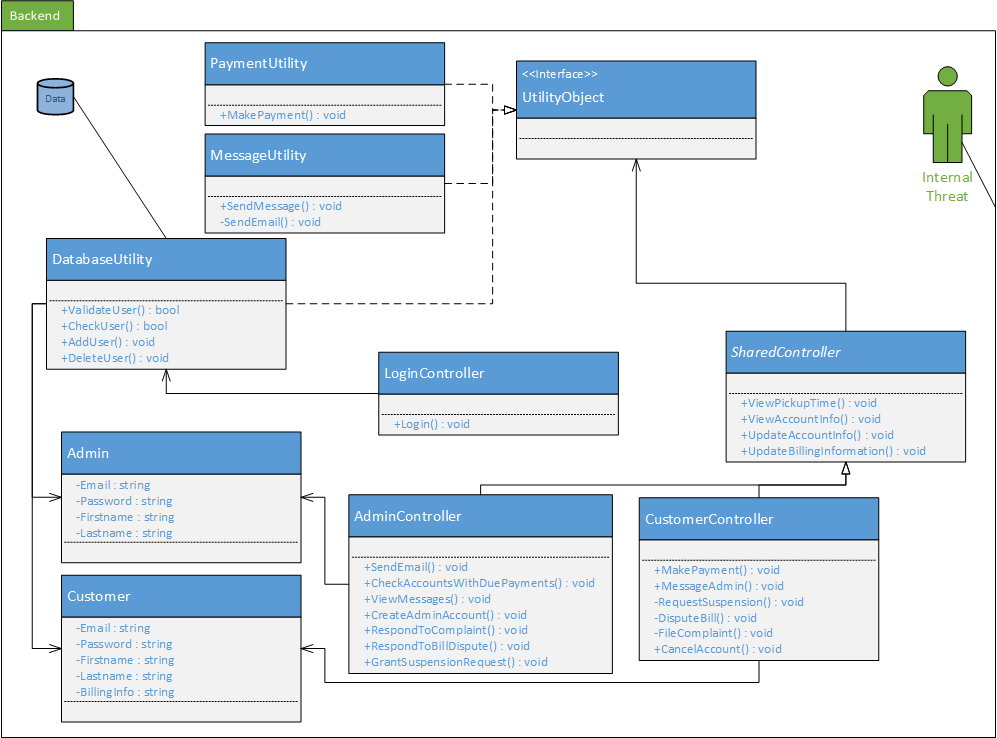
The interface with hardware will consist of networking and the use of servers that will connect any user to the pages of the website.

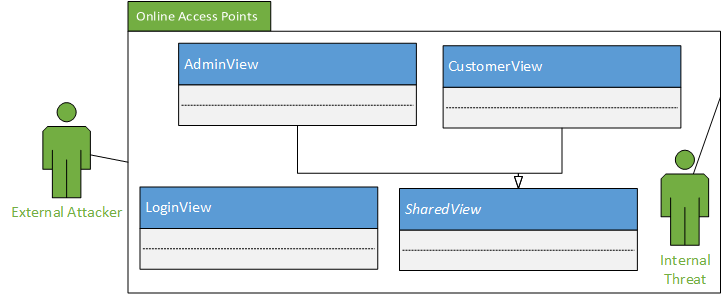
## 

# Architectural Risk Analysis

## Software Characterization

**ErieGarbage Online Architectural Diagram**





The above image represents ErieGarbage Online’s high-level architectural design. This design is divided into two main packages: the backend and online access points. 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 Backend package shown above.

## 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 | * Input of falsified data * Harassing or attacking employees |
| Cracker | * Challenge * Rebellion * Ego | * System profiling * Social engineering * System intrusion * Unauthorized system access |
| Insiders  (poorly trained, disgruntled, malicious, negligent, dishonest, or terminated employees) | * Curiosity * Ego * Intelligence * Monetary gain * Revenge * Non-malicious errors * Lack of training | * Assault on an employee * Blackmail * Computer abuse * Fraud & theft * Input of falsified data * Interception * Malicious code * System bugs * System intrusion * System sabotage * Unauthorized system access |
| Competing Companies | * Learn confidential information * Blackmail * Economic espionage * Destroy information * Study internal tactics * Gain a competitive advantage | * Social engineering * System penetration * Unauthorized access to data and/or systems * Information theft * Economic exploitation * Intrusion on personal privacy |

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

## 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. |

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 |
| 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 |
| Nonrepudiation | Nonrepudiation refers to the ability of a system to counter repudiation threats. For example, a user who purchases an item might have to sign for the item upon receipt. The vendor can then use the signed receipt as evidence that the user did receive the package. | CWE-311 - Missing Encryption of Sensitive Data |
| 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 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. | CWE-400: Uncontrolled Resource Consumption ('Resource 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-288 - Authorization Bypass Using an Alternate Path or Channel  CWE-425 - Direct Request |

Ambiguity analysis:

In this subsection, we attempt to find any ambiguities in the requirements or architecture of the ErieGarbage Online system and clarify them. This will help eliminate misunderstandings between the business requirements of the software and programmer’s implementation of the design in code. Functions, structures, properties, and policies of the software that could 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 functions containing possible ambiguities:

* Checking customer accounts that have payments due
  + Admins should be able to retrieve a list of customers from the EGO system that have not yet paid their bill for the month. This list will display the first name of the customer, the last name of the customer, and the customer’s email address.
* Creating new administrator accounts
  + By default, EGO comes with one admin account pre-installed into the system. It should have a generic login email address (admin@ErieGarbageOnline.com) and a randomized password delivered by the developers to the client either in person or a form of secure communication. The client should be instructed to change this password immediately upon arrival. This account is then used to create other admin accounts using the admin’s company email address and a randomized password. This password will be provided to the new admin, who can then change the password to their choice. Each new admin account provides this same functionality.

Customer functions containing possible ambiguities:

* Make payments for the service
  + Every user will pay for the service on the last business day of the month. This is done by the customer using the payment page to make a payment. On this page, the customer will need to enter their billing information (first name, last name, email address, phone number, and credit card number) to legitimize the transaction and to provide a form of payment. The system will provide to the user the amount that they owe. The system will use the 3rd party payment software (TBD) to process the payment. EGO will make sure the payment is successful or ask the user for another form of payment. There will be no refunds. Payment information may be saved and automated by the system if the customer desires. If the user is not able to pay, they will become suspended until payment is made for all missed payments.
* Cancelling their own account
  + A user has the option to cancel their account. All user data will be deleted from the user database. Transactional records and other logs may be kept by ErieGarbage for data retention purposes.

Software functions that are used by both customer and admin accounts:

* Viewing garbage collection time
  + Admins will be able to request a specific user’s collection time from the system and view that pick up time. Customers will be able to view their own collection pick up time within the system.
* Functions involving account information
  + Account information of a customer:
    - Name (viewable by admins & self only)
    - Phone (viewable by admins & self only)
    - Email address (viewable by admins & self only)
    - Password (viewable only by self)
  + Account information of an admin:
    - Name (viewable by admins, self, and customers)
    - Phone (viewable by admins, self, and customers)
    - Email address (viewable by admins, self, and customers)
    - Password (viewable only by self)
* Functions involving billing information
  + Only customers will have billing information, but some information can be edited by admins
  + Billing information consists of:
    - Name (updatable by self only)
    - Street address (updatable by self or admin)
    - City (updatable by self or admin)
    - Country (updatable by self or admin)
    - State/Province (updatable by self or admin)
    - Postal Code (updatable by self or admin)

Next, we will look at the potential 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.

Possible architectural and design ambiguities:

* The overall architectural diagram shown under **Software Characterization** does not depict the system connected to the internet, however, it is connected to the internet. This should have been assumed since this is an online system.

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
* 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 |
| Microsoft .NET Framework | CWE 200 Information Exposure  CWE Improper Input Validation  CWE Failure to Preserve Web Page Structure (‘Cross-site Scripting’) |

## 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** | **Motivation & Capability** | **Severity** | **Effectiveness of Controls** | **Likelihood** |
| Spoofing identity | Fairly difficult to do and not much reason to do so | Low - not likely to affect many users | Effective- Two-factor authentication | Low |
| Tampering with data | SQL injection or gaining access to a database is a very motivating weakness, there is a lot to gain for an attacker. | High - modifying databases could cause massive damage to the website functionality | Effective - SQL parameterization and checking authentication before any action will help mitigate an attack from happening | Medium |
| Repudiation | Due to the nature of the website, users will probably not attempt illegal actions. | High - difficult to trace users performing illegal actions | Effective- Automatic logging | Low |
| Nonrepudiation | User may not want to pay bill for the month | Medium - possibility of ErieGarbage not able to produce evidence of misuse | Effective- actions and messages are logged in the database | Low |
| Information disclosure | Users have little reason to care about internal system functionality or other user info. | Medium - users would be unhappy about their information being liked, but system functionality being exposed would result in little consequence | Effective - Logs are viewable by admins only | Medium |
| Denial of service | Attacker may be unhappy with the service. Attacker would need scripting knowledge and knowledge of botnets. | Medium - website would be offline temporarily | Effective - DDOS protection | Low |
| Elevation of privilege | An attacker gaining admin rights to the system would gain a lot of control over other users and have a large amount of influence | Low - an admin can perform a lot of duties, however every action can be undone and is logged | Effective - Customer and admin accounts are kept entirely separate, and the privileges are separated between the account types | Low |

## 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 | 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 | Small - Only a single or handful of users will be affected by each successful attack | Low |
| Tampering with data | Any database that EGO uses would be at risk for having data modified | High - The databases are essential to EGO operation | Large - The potential to delete all data from the database would compromise the entire system | High |
| Repudiation | 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 | Large - If the system is not in place for a single user, then every user is affected | Medium |
| Nonrepudiation | User confidence in system, system integrity | High - Not providing a user with confirmation emails would allow an attacker to take potential legal actions | Large - If a single user is affected, then every user is affected | Medium |
| Information disclosure | User information and system functionality exposed | Medium - Could affect EGO if attackers use this information | Medium - Effects would be seen throughout the system | High |
| Denial of service | Functionality of the website | High - Would prevent use of the site for the duration of the attack | Large - Would affect the entire system from being used | High |
| Elevation of privilege | User information, account status | Medium - Admin modifications can be undone but information exposure could hurt the company's image and profit | Large - Admins have a wide degree of influence over the system's users | High |

## 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** | **Likelihood** | **Impact** | **Exposure Level** |
| Spoofing identity | Low | Low | Low |
| Tampering with data | Medium | High | High |
| Repudiation | Low | Medium | Low |
| Nonrepudiation | Low | Medium | Low |
| Information disclosure | Medium | High | High |
| Denial of service | Low | High | Medium |
| Elevation of privilege | Low | High | Medium |

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**

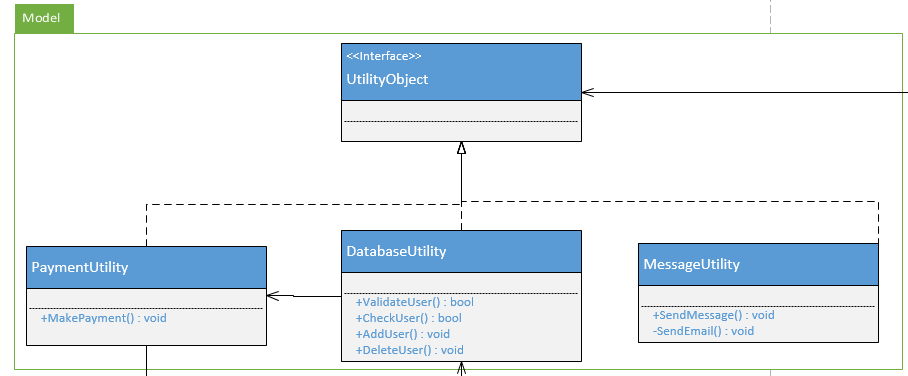
|  |  |  |
| --- | --- | --- |
| **Priority** | **Risk** | **Steps** |
| High | Tampering with data | CWE-89 - Use a framework that doesn’t allow this weakness to occur, enforce separation of data and code |
| Information disclosure | CWE-425 - Apply appropriate access control authorizations for each 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 |
| Nonrepudiation | CWE-311 - Use industry-approved techniques and separate sensitive and non-sensitive data as much as possible |

# Internal Module Design

## 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.

### Model Class Diagram



### Model Classes

Class Descriptions

|  |  |
| --- | --- |
| **Class name** | DatabaseUtility |
| **Description** | Used to access the user database |
| **Attributes** | None |
| **Methods** | * public bool ValidateUser() * public bool CheckUser() * public void AddUser() * public void DeleteUser() |

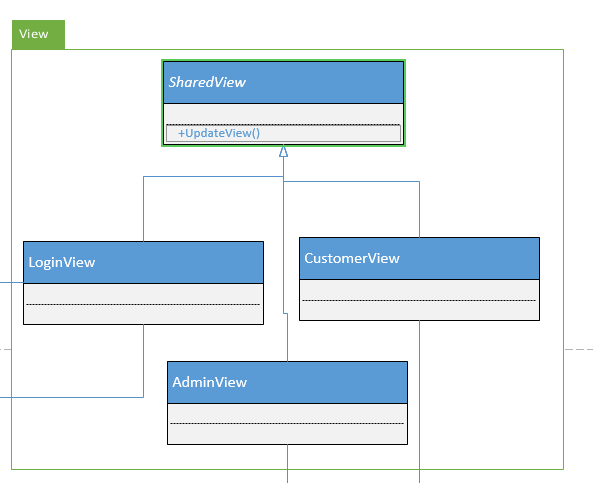
Method Descriptions

|  |  |
| --- | --- |
| **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



Above 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.

### 

### 5.2.2 View Classes

Class Descriptions

|  |  |
| --- | --- |
| **Class name** | LoginView |
| **Description** | The page all users will see when they attempt to login |
| **Attributes** | private LoginController login |
| **Methods** | * public void UpdateView()   + Updates the view as the user performs different actions |

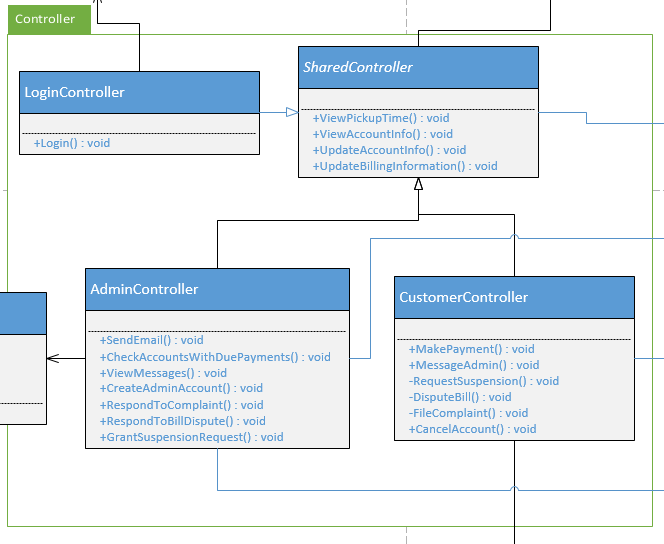
Method Descriptions

|  |  |
| --- | --- |
| **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



### 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 |

# Team Members Log Sheets

## 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. |
|  | **Total :** | 12.5 hrs. |

## 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 |
|  | **Total :** | 9.5 hrs. |

## Alex Lee

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