# PennStateSoft

# Meeting Scheduling System Design Document

**Version <1.0>**

**TEAM MEMBERS**

| Name | Student ID |
|---|---|
| AbdAllah Mahassen | akm6518 |
| Chin Shiang Jin | Sjc6393 |
| Caralyn Harben | clh6036 |
| | |
| | |

## Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 25/7/2021 | <1.0> | Submission | Chin Shiang Jin, Caralyn Harben, AbdAllah Mahassen |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

*Table of Contents*

# Design Document

## 1. Introduction

This software design document outlines all the key details related to the development of the Meeting Scheduling System. These key details consist of Architectural Design, Software Interface Design, and Internal Module Design. The upcoming sections describe the purpose of the document, the scope of the document, the abbreviations used in the document, the references, and an overview of the document.

### 1.1 Purpose

This Software Design Document (SDD) serves to provide all the key details of the Architectural Design (AD), the Module Interface Design (MID), and Internal Module Design (IMD) for the Meeting Scheduling System (MSS). The AD's purpose is to present a high level of understanding of how the system works. The purpose of the MID is to describe the relationships between the system's modules. Finally, the IMD's purpose is to go into depth about the key details in each of the system's modules.

### 1.2 Scope

This Software Design Document (SDD) discusses, in detail, the design as it pertains to the PennStateSoft Meeting Scheduling System (MSS). The SDD shall cover the design of the system by modeling class relationships between various internal modules, as well as the user interface with which both types of users, administrators, and general clients, will interact with the system. Further, the SDD shall discuss risks posed by external entities, internal factors, and design decisions.

### 1.3 Definitions, Acronyms, and Abbreviations

| **Abbreviation** | **What it stands for** |
|---|---|
| AD | Architectural Design |
| CAPEC | Common Attack Pattern Enumeration and Classification |
| CRUD | Create, Read, Update and Delete |
| CWE | Common Weakness Enumeration |
| ID | Identification |
| IMD | Internal Module Design |
| ISP | Internet Service Provider |
| JSON | JavaScript Object Notation |
| MID | Module Interface Design |

| | |
|---|---|
| MSS | Meeting Scheduling System |
| MVC | Model-View-Controller |
| SDD | Software Design Document |
| TLS | Transport Layer Security |
| UI | User Interface |

## 1.4 References

Project Software Requirement Specification document from phase I project submission, titled SWENG455_Team4_MSS-SRS.

*Common Attack Pattern Enumeration and Classification*. CAPEC. (n.d.). https://capec.mitre.org/

*Common Weakness Enumeration*. CWE. (n.d.). https://cwe.mitre.org/

Morgenroth, S. (2020, May 5). *What is the SQL Injection Vulnerability & How to Prevent it?* Netsparker. https://www.netsparker.com/blog/web-security/sql-injection-vulnerability/.

## 1.5 Overview

The document is organized as follows:

### *1.1.1 Section 1: Introduction*

The introduction provides a general overview of the SDD, including the purpose of the document, the scope of what the document covers, all definitions, acronyms, and abbreviations, a list of documents referenced within the SDD, and an overview of the document's structure.

### *1.1.2 Section 2: Architectural Design*

Section 2 discusses the overall architecture of the MSS and why the development team chose to model the MSS in this way. Further, a risk analysis is conducted and logged in section 2, which covers internal and external risk factors, impacts on the system, and mitigation methods. The analysis discusses the applications of the Principles of Software Security used in the MSS.

### *1.1.3 Section 3: Software Interface Design*

The Software Interface Design section consists of a detailed description of the interactions between the user and the MSS, the interactions between the internal modules contained within the MSS, and any interactions between the MSS and external systems. A general overview of the system states that can be toggled through direct user interaction is shown in the section. In addition, mock-ups of the UI are provided to show how each user might interact with the system.

### *1.1.4 Section 4: Internal Module Design*

Section 4 provides a more detailed description of each internal module within the MSS. An overall class diagram of the system, as well as a description of each class, has been provided to describe the functions and attributes within the classes shown. Relationships between each class are also modeled in the provided diagrams and tables.

*1.1.5 Section 5: Team Member Log Sheets*

The log sheet's purpose is to show the amount of time spent by each member of the development team to calculate the total time cost to design and develop the MSS. Provided is a description of each task completed by a given team member and the amount of time it took to complete the listed task(s).

## 2. Architectural Design

### 2.1 Rationale

We model our project based on the Model-View-Controller(MVC) architecture as it is most appropriate to model the website system. MVC architecture allows us to separate the presentation and interaction from the system data objects. This enables us to change the data objects independently of their representation and vice versa, enabling us to work simultaneously on different components of our projects without impacting one another. This will help us develop our software application faster and more efficiently. Besides, the services and tools that we have chosen, which include Html and JavaScript, PHP scripts, and MySQL database lend themselves to an MVC architecture as well.

We didn't select the client-server architecture as the web services for our project cannot be separated into different client-services interaction pairs. Instead, all the web services are grouped and hosted in a centralized server to work on the same database. Due to limited budget constraints as well as the limited number of users, it will be impractical to host the website and the database across multiple servers as well.

We also didn't select the layered architecture as our project is completely new instead of building on top of existing systems. It is also not possible to provide a clean separation between the layers as the services provided are interrelated to each other.

We didn't select the repository architecture because this architecture does not model the separation of view and controller in a website application. Besides, although our data objects are stored in a centralized location, they are not communicated via a single repository (like a single PHP script). Separating the data objects and their interaction into separate UI pages, controller class, PHP script and MySQL tables allow us to develop the system faster, allow easier troubleshooting of bugs, as well as implementing and testing of the security features added.

## 2.2 Software Architecture Diagram

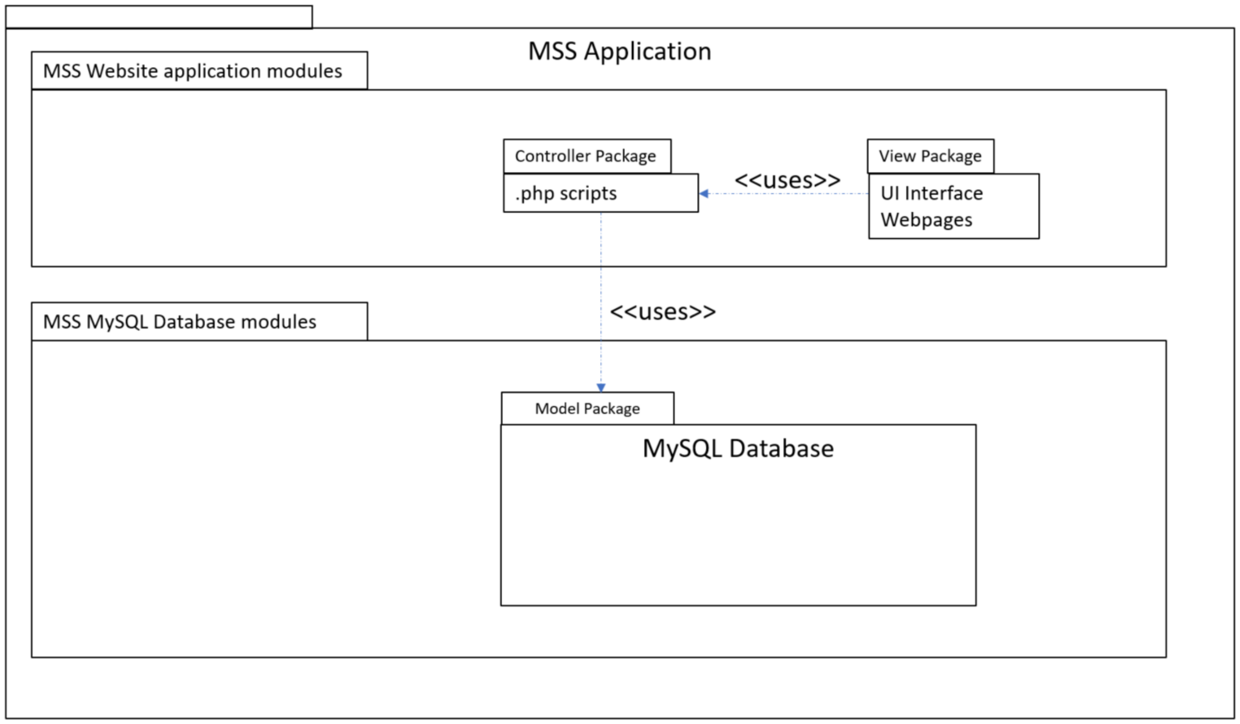


*Figure 1: System Architecture Diagram showing the modules of the system*

Figure 1 shows the very high-level summary of the software architecture diagram, a more detailed version of the module interface diagrams is included in section 3.2. In summary for each of the functional features, the implementation is divided into three classes with one class for each of the view, controller, and model package. The view package will realize the UI interface on the client's terminal (monitor), which mainly consists of webpages viewable on the web browsers. The controller package will act as the event handler, handling the interaction request from the users, and interacting with the model package for the common data Create, Read, Update and Delete (CRUD) operation. The model package is representing the MySQL database tables, where each of the tables will store the entity and data required to implement the function.

## 2.3 System Topology

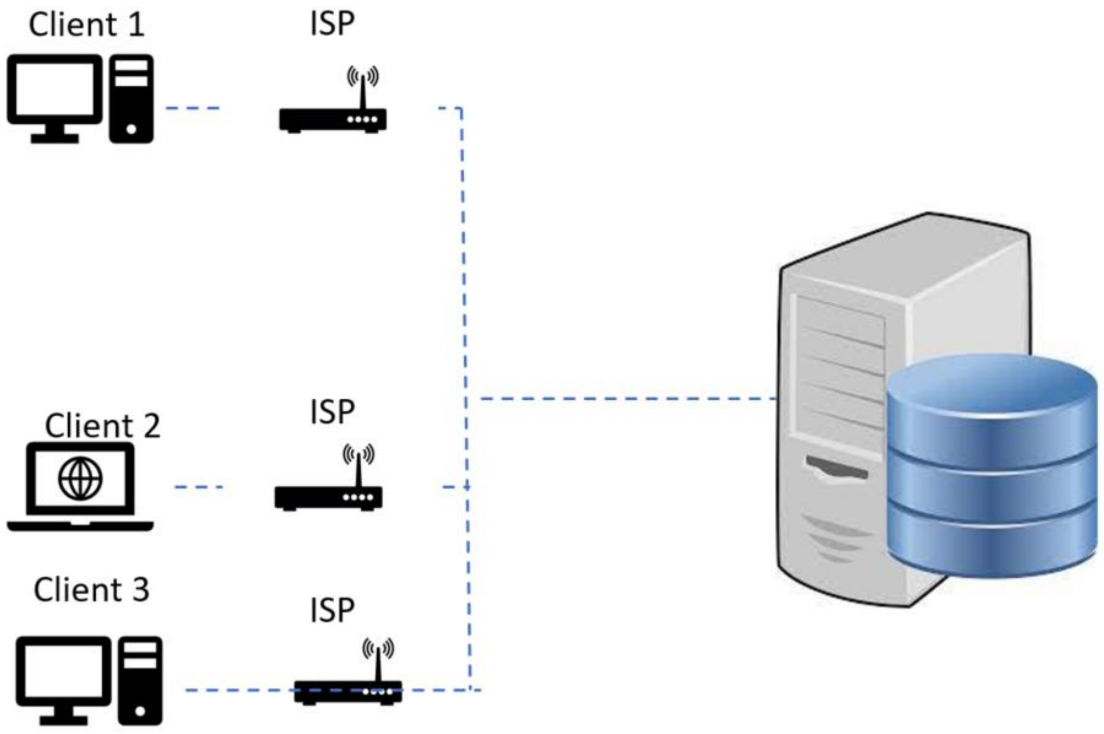


*Figure 2: System Topology Diagram*

A very high-level summary of the system topology is shown in figure 2. In summary, the users (either the administrator or the client) will start the connection from the browser on their computer or devices. Going through the same or different Internet Service Provider (ISP), the client browser will connect to the centralized server hosting the controller package (the PHP scripts) and also the database model.

## 2.4 Architectural Risk Analysis

### 2.4.1 Software Characterization

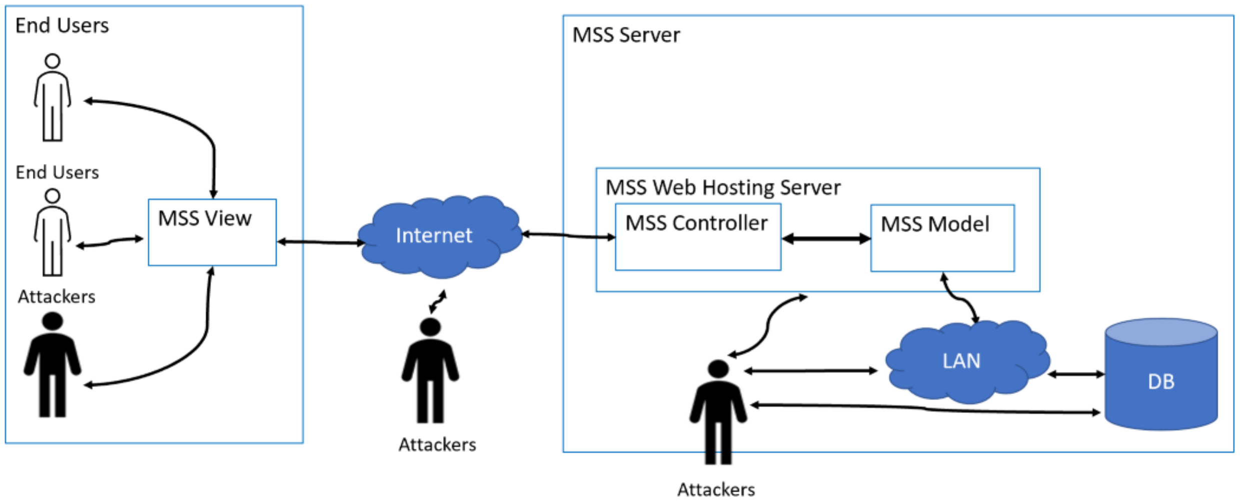


*Figure 1: Software Characterization Diagram*

The figure above shows the software characterization diagram for the system. The view content will be delivered to the browsers of the end-users over the internet. The view modules will be interacting with the controller modules hosted on the webserver. The controller modules used the Model objects to interact with the database, most likely hosted on the same webservers but at different hard disk locations. The potential point of attack includes the attack on the end browser, attack on the connection between the view and the controller, attack on the web hosting server, attack on the connection between the web hosting server and the database, and the attack on the database directly.

### 2.4.2 Threat analysis

The following table lists the potential threat source and threat actions. The threat highlighted in bold is a highly likely threat that will be discussed further in the architectural vulnerability assessment, risk likelihood, risk impact, and risk mitigation planning. From the threat identified, the credit card payment information stored is the most likely target for monetary gain.

| Threat Source | Motivation | Threat Actions |
|---|---|---|
| Cracker | 1. Challenge<br>2. Ego<br>3. Rebellion<br>4. Monetary gain | - System Profiling<br>- Social engineering<br>- System intrusion, break-ins<br>- **Unauthorized system access**<br>- **Computer crime ( Browsing of credit card information, Fraud and theft)**<br>- Spoofing<br>- Spam |

| | | |
|---|---|---|
| Terrorist | 1. Blackmail<br>2. Destruction<br>3. Revenge<br>4. Monetary gain | · System attack (like denial of service)<br>· System tampering |
| Industrial Espionage/Competing company | 1. Competitive advantage<br>2. Economic espionage | · **Information theft**<br>· **Unauthorized system access**<br>· Social engineering to turn insider into the attacker |
| Insider | 1. Monetary gain<br>2. Unintentional errors and omissions<br>3. Victims of social engineering<br>4. Lacks of procedures or training | · **Browsing of credit card information**<br>· **Fraud and theft**<br>· System bugs<br>· System sabotage<br>· **Unauthorized system access** |

*Table 1: Threat Analysis for potential threat sources and threat actions.*

### *2.4.3 Architectural vulnerability assessment*

#### 2.4.3.1 Assessment against CWE and CAPEC

Examining the software architecture and design for common weaknesses and comparing against known bad practices, we found the following Common Weaknesses Enumeration (CWE) that applies to our software architecture. The potential Common Attack Pattern Enumeration Classification (CAPEC) exploiting the CWEs are also listed in the following table.

| CWE ID | CWE Title | Description |
|---|---|---|
| 89 | Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') | The software constructs all or part of an 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) |
| 94 | Improper Control of Generation of Code ('Code Injection') | The software constructs all or part of a code segment using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the syntax or behavior of the intended code segment (CWE). |
| 261 | Weak Cryptography for Passwords | Obscuring a password with a trivial encoding does not protect the password (CWE) |
| 308 | Use of Single Factor Authentication | The use of single-factor authentication can lead to unnecessary risk of compromise when compared with the benefits of a dual-factor authentication scheme. |

| | | |
|---|---|---|
| 311 | Missing Encryption of sensitive data | The software does not encrypt sensitive or critical information before storage or transmission (CWE) |
| 521 | Weak Password Requirements | The product does not require that users should have strong passwords, which makes it easier for attackers to compromise user accounts (CWE) |

*Table 2: Common Weakness Enumeration related to the threats highlighted*

| CWE ID | Related CAPEC Type | Description |
|---|---|---|
| 89 | CAPEC-7: Blind SQL Injection | Blind SQL Injection results from an insufficient mitigation for SQL Injection. Blind SQL Injection is a form of SQL Injection that overcomes the lack of error messages. Without the error messages that facilitate SQL Injection, the adversary constructs input strings that probe the target through simple Boolean SQL expressions. The adversary can determine if the syntax and structure of the injection were successful based on whether the query was executed or not. Applied iteratively, the adversary determines how and where the target is vulnerable to SQL Injection (CAPEC). |
| | CAPEC-66: SQL Injection | This attack exploits target software that constructs SQL statements based on user input. An attacker crafts input strings so that when the target software constructs SQL statements based on the input, the resulting SQL statement performs actions other than those the application intended. SQL Injection results from the failure of the application to appropriately validate input. When specially crafted user-controlled input consisting of SQL syntax is used without proper validation as part of SQL queries, it is possible to glean information from the database in ways not envisaged during application design. Depending upon the database and the design of the application, it may also be possible to leverage injection to have the database execute system-related commands of the attackers' choice. SQL Injection enables an attacker to interact directly with the database, thus bypassing the application completely. Successful injection can cause information disclosure as well as ability to add or modify data in the database. (CAPEC) |
| | CAPEC-108: Command Line Execution through SQL Injection | An attacker uses standard SQL injection methods to inject data into the command line for execution. This could be done directly through misuse of directives such as MSSQL_xp_cmdshell or indirectly through injection of data into the database that would be interpreted as shell commands. Sometime later, an unscrupulous backend application (or could be part of the functionality of the same application) fetches the injected data stored in the database and uses this data as command line arguments without performing proper validation. The malicious data escapes that data plane by spawning new commands to be executed on the host (CAPEC). |
| 94 | CAPEC-242: Code Injection | An adversary exploits a weakness in input validation on the target to inject new code into that which is currently executing. This differs from code inclusion in that code inclusion involves the addition or replacement of a reference to a code file, which is subsequently loaded by the target and |

| | | |
|---|---|---|
| | | used as part of the code of some application (CAPEC). |
| 261 | CAPEC-55: Rainbow Table Password Cracking | An attacker gets access to the database table where hashes of passwords are stored. They then use a rainbow table of pre-computed hash chains to attempt to look up the original password. Once the original password corresponding to the hash is obtained, the attacker uses the original password to gain access to the system. A password rainbow table stores hash chains for various passwords. A password chain is computed, starting from the original password, P, via a reduce(compression) function R and a hash function H. A recurrence relation exists where Xi+1 = R(H(Xi)), X0 = P. Then the hash chain of length n for the original password P can be formed: X1, X2, X3, ... , Xn-2, Xn-1, Xn, H(Xn). P and H(Xn) are then stored together in the rainbow table. Once a rainbow table is computed, it can be very effective in cracking the passwords that have been hashed without the use of salt (CAPEC). |
| 308 | CAPEC-49: Password Brute Forcing | In this attack, the adversary tries every possible value for a password until they succeed. A brute force attack, if feasible computationally, will always be successful because it will essentially go through all possible passwords given the alphabet used (lower case letters, upper case letters, numbers, symbols, etc.) and the maximum length of the password. A system will be particularly vulnerable to this type of an attack if it does not have a proper enforcement mechanism in place to ensure that passwords selected by users are strong passwords that comply with an adequate password policy (CAPEC). |
| | CAPEC-70: Try Common or Default Usernames and Passwords | An adversary may try certain common or default usernames and passwords to gain access into the system and perform unauthorized actions. An adversary may try an intelligent brute force using empty passwords, known vendor default credentials, as well as a dictionary of common usernames and passwords. Many vendor products come preconfigured with default (and thus well-known) usernames and passwords that should be deleted prior to usage in a production environment. It is a common mistake to forget to remove these default login credentials. Another problem is that users would pick very simple (common) passwords (e.g. "secret" or "password") that make it easier for the attacker to gain access to the system compared to using a brute force attack or even a dictionary attack using a full dictionary (CAPEC). |
| 311 | CAPEC-158: Sniffing Network Traffic | In this attack pattern, the adversary monitors network traffic between nodes of a public or multicast network in an attempt to capture sensitive information at the protocol level. Network sniffing applications can reveal TCP/IP, DNS, Ethernet, and other low-level network communication information. The adversary takes a passive role in this attack pattern and simply observes and analyzes the traffic. The adversary may precipitate or indirectly influence the content of the observed transaction, but is never the intended recipient of the target information (CAPEC). |
| 521 | Same as CAPEC for CWE308 | |

*Table 3: Common Attack Pattern Enumeration Classification (CAPEC) related to the CWE*

#### 2.4.3.2 Assessment against Good Security Design Principles

The first security principle to consider during the software architecture design is the principle of least privilege. This principle states that only the minimum necessary rights should be assigned to a subject that requests access to a resource and should be in effect for the shortest duration necessary. In our software architecture design, this principle is implemented by segregating the different type of normal client account and administrator account. By granting the normal client account the necessary right only to perform their required functions such as create a meeting, reserve room, pay for a special room, add or remove participants, and review meeting they are participates in, we can ensure the client account won't be able to perform functions allowed for administrator account only. For example, the system should allow the client to be able to view meetings created by themselves or meetings participated by them only. While they will require administrative rights to view all the meetings created by others or joined by others.

The second security principle to consider is the principle of failing securely. This includes implementing the element of secure defaults which is to deny access during authentication failure, reject making changes to the system on input validation failure, or authentication failure. During failure, the system shouldn't reveal any sensitive or confidential information about the failure, instead, it should just reveal the high-level cause of failure.

The third security principle to consider is the principle of securing the Weakest Link. As the attackers are more than likely to attack a weak spot in a weak spot in the software system than penetrating a heavily fortified component, the strategy is to fortify the back-end system consisting of the web-hosting server which hosts the website and the database.

The fourth security principle to employ is the principle of defense in Depth. Redundant security mechanisms are necessary when handling sensitive information such as credit card payment details. Hence two-factor authentication will be required to build the layering security defenses to prevent the attackers from gaining access or stealing the credit card information.

The fifth security principle to assess is the principle of separation of privilege. For the system to ensure multiple conditions are met before granting permissions to an object, it may be best to avoid granting the administrator account the same functional requirement as the normal client account, as doing so may make it difficult to implement the scheme of multiple checks to determine if a normal client can also access administrative features. By separating the functions for the normal client and administrator into two different compartments, the system can help ensure the attackers who granted access to the client account will not be able to access the administrative features that can do more damage to the system if misused. Besides, enhance security and authentication control can be implemented on the administrator account, making it harder for the attacker/terrorist to gain access to one.

The sixth security principle we need to bear in mind is the principle of Economy of Mechanism. The principle states that we should always strive for implementing simpler system designs whenever possible, as more complex designs increase the likelihood of security vulnerabilities exists and being exploited. Its effect on the system architecture designs is that we will adhere to standardize MVC model, where each of the functions is implemented via a single user interface controlled by a single PHP controlling script, and each PHP script access as few database tables as possible.

The seventh security principle to consider is to implement the principle of a least common mechanism. However, this should be considered together with the principle of Psychological acceptability. We need to try to avoid having multiple subjects share the mechanisms that grant access to a resource, an example being caching the user login status, which will grant the user access to several features of the systems as long as their browser remain login to the system even though they have left the desk. This will allow other attackers to gain access to the system immediately using the same browser. The system architecture design will need to implement a time-out feature, where the users will need to be reauthenticated after a certain time lapse. These time-lapses shouldn't be too short that the user will need to authenticate themselves now and then, thus potentially turning them away from the system.

The next security principle to assess is the principle of reluctance to trust. In this case, the webpage UI as the boundary class should be considered untrusted because they are the direct input location for the users including the attackers. The controller classes should always authenticate the boundary class and secured the interface between the two systems.

Another security principle to assess is the principle of complete mediation. For security-critical objects such as the storing of credit card information, the system should require access checks each time the client/administrator account requests access. For example, between each successive payment attempt, the system should re-authenticate the users and do not cache the permissions granted. Even if permission is granted, the concept of multiple-factor authentication should be used such as asking the user to reenter the security pin of their credit card.

The last security principle to be considered is the principle of psychological acceptability. Access to common resources which doesn't reveal sensitive information (such as credit card payment) should not be hindered by the security mechanism deployed. Thus, retaining log-in status when the user navigates through different user interfaces needs to be considered to introduce minimal obstruction to the users in using the majority of the system features. Additional authentication should be employed only to control access to critical data.

#### 2.4.3.3 Ambiguity Analysis

The first ambiguity being examined is whether the administrator account can perform the functions available to the normal client account. This could potentially lead to the crackers being able to upgrade a normal client account into an administrator account through code injection or SQL injection. After further clarification with the client, the requirements have been refined such that the administrator account can only access the admin account function, and not allowing the client account to be upgraded to the administrator account.

The second ambiguity being examined is whether the client account can make the payment to the special room using payment information (credit card) stored in the database. Although this brings convenience to the client, the protection against stored credit card information is weak against code injection or SQL injection. To enhance the security protection, the requirements have been refined to store only the partial information of the credit card (the number and the name), and require the client to re-authenticate themselves to proceed with the payment.

#### 2.4.3.4 Dependency Analysis

The User Interface (UI) module of the software will be executed on the client's browser, which can be any common browser such as Google Chrome, Mozilla Firefox, Apple Safari, or Microsoft Edges that resides on Windows, Androids, and IOS operating systems. Hence this UI module will be subject to the same operating system vulnerabilities as the client's terminal browser and operating system.

As the UI module will communicate with the Controller module over the internet infrastructure, it will be subject to the common network vulnerabilities associated with the internet as well. The example includes CWE-311: Missing Encryption of sensitive data and CAPEC Sniffing Network Traffic.

The Controller and Model modules will be built based on PHP, HTML, JavaScript languages, hence it will be subject to the common platform vulnerabilities related to the languages above. One example being CWE-94 Improper Control of Generation of Code ('Code Injection') which can lead to CAPEC-242: Code Injection.

Last but not least, as the Controller and Model modules will be interacting with the MySQL database, it will have to protect against interaction vulnerabilities related to SQL database such as prevention against SQL injections. Example vulnerabilities are similar to CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') which can lead to the following three CAPECs: CAPEC-7: Blind SQL Injection, CAPEC-66: SQL Injection, CAPEC-108: Command Line Execution through SQL Injection.

#### 2.4.3.5 Threats and Vulnerabilities Mapping

The following tables map the threats (separated by sources and actions) with vulnerability (CWE and CAPEC) discussed in previous sections:

| Threat | Related Vulnerability | |
|---|---|---|
| | CWE | CAPEC |
| Unauthorized system access by Cracker | CWE-261 Weak Cryptography for Passwords<br>CWE-308 Use of Single Factor Authentication<br>CWE-521 Weak Password Requirements | CAPEC-55: Rainbow Table Password Cracking<br>CAPEC-49: Password Brute Forcing<br>CAPEC-70: Try Common or Default Usernames and Passwords |
| Computer crime ( Browsing of credit card information, Fraud, and theft) by Cracker | CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')<br>CWE-94: Improper Control of Generation of Code ('Code Injection')<br>CWE-311: Missing Encryption of sensitive data | CAPEC-7: Blind SQL Injection<br>CAPEC-66: SQL Injection<br>CAPEC-108: Command Line Execution through SQL Injection<br>CAPEC-242: Code Injection<br>CAPEC-158: Sniffing Network Traffic |
| Information theft by Industry espionage / competing company | CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')<br>CWE-94: Improper Control of Generation of Code ('Code Injection')<br>CWE-311: Missing Encryption of sensitive data | CAPEC-7: Blind SQL Injection<br>CAPEC-66: SQL Injection<br>CAPEC-108: Command Line Execution through SQL Injection<br>CAPEC-242: Code Injection<br>CAPEC-158: Sniffing Network Traffic |
| Unauthorized system access by Industry espionage / competing company | CWE-261 Weak Cryptography for Passwords<br>CWE-308 Use of Single Factor Authentication<br>CWE-521 Weak Password Requirements | CAPEC-55: Rainbow Table Password Cracking<br>CAPEC-49: Password Brute Forcing<br>CAPEC-70: Try Common or Default Usernames and Passwords |
| Browsing of credit card information by an insider | CWE-311: Missing Encryption of sensitive data | CAPEC-158: Sniffing Network Traffic |
| Fraud and theft by an insider | CWE-311: Missing Encryption of sensitive data | CAPEC-158: Sniffing Network Traffic |
| Unauthorized system access by insider | CWE-261 Weak Cryptography for Passwords<br>CWE-308 Use of Single Factor Authentication<br>CWE-521 Weak Password Requirements | CAPEC-55: Rainbow Table Password Cracking<br>CAPEC-49: Password Brute Forcing<br>CAPEC-70: Try Common or Default Usernames and Passwords |

*Table 1: Threats and Vulnerabilities mapping results*

Note: Code injection and SQL injection is deemed unlikely to be performed by an insider as they are considered to have the low capability to carry out such an attack.

### 2.4.4 Risk likelihood determination

The risk likelihood determination for the key threats selected are described below

**Unauthorized system access by Cracker**
The threat's motivation and capability:

- The attacker will have high motivation in exploitation for monetary gain and most likely having the high capability.

The vulnerability's impact (and therefore attractiveness to an attacker):

- Gaining access to an administrator account or a client account allow the attacker to view the sensitive information such as credit card. Hence the vulnerability's impact will be high.

The effectiveness of current controls:

- The current system design will request for a strong password to be selected during account creation. However additional measures are required to encrypt the password and protect the password stored in the database. Hence without further mitigation, the current control is weak.

**Resulting likelihood: High**

**Computer crime ( Browsing of credit card information, Fraud, and theft) by Cracker**
The threat's motivation and capability:

- The attacker will have high motivation in exploitation for monetary gain and most likely having the high capability.

The vulnerability's impact (and therefore attractiveness to an attacker):

- Browsing of credit card information allows the cracker to perform fraud related to the credit card, for example like using the stolen credit card information to buy things online. Hence the vulnerability's impact will be high.

The effectiveness of current controls:

- The credit card information will be store in the database which can be viewed by the administrator user without additional control. There is currently no control over the transmission of credit card information over the internet. Hence without further mitigation, the current control is considered weak.

**Resulting likelihood: High**

**Information theft by Industry espionage / competing company**
The threat's motivation and capability:

- The attacker will have high motivation through misuse of the information obtained, or using the information to disrupt the business of the company. They are most likely having the high capability.

The vulnerability's impact (and therefore attractiveness to an attacker):

- As most of the information stored in the database is of meeting information instead of meeting minutes, this information is unlikely to have commercial value. Therefore information stolen at the hand of Industry espionage / competing companies will have little usefulness. Hence the vulnerability's impact is low.

The effectiveness of current controls:

- The meeting information stored in the database won't be encrypted. Hence any intruders gaining access to the database can easily extract the information out. Without further mitigation, the current control is considered weak.

**Resulting likelihood: Medium**

**Unauthorized system access by Industry espionage / competing company**

The threat's motivation and capability:

- The attacker will have high motivation through misuse of the information obtained, or using the information to disrupt the business of the company. They are most likely having the high capability.

The vulnerability's impact (and therefore attractiveness to an attacker):

- Industry espionage / Competing companies gaining unauthorized access (especially administrator access privilege) can perform a lot of sabotage action. Examples include deleting all the users from the system and thus causing business disruption to the company. Thus the vulnerability's impact is high.

The effectiveness of current controls:

- The current system design will request for a strong password to be selected during account creation. However additional measures are required to encrypt the password and protect the password stored in the database. Hence without further mitigation, the current control is weak.

**Resulting likelihood: High**

**Browsing of credit card information by an insider**

The threat's motivation and capability:

- The attacker will have medium motivation if they are aiming for financial gain as the reward (spending limit of the credit card) is usually low compared to the job income they will lose. They will likely have a low capability as well.

The vulnerability's impact (and therefore attractiveness to an attacker):

- Browsing of credit card information allows the cracker to perform fraud related to the credit card, for example like using the stolen credit card information to buy things online. Hence the vulnerability's impact will be high.

The effectiveness of current controls:

- The credit card information will be store in the database which can be viewed by the administrator user without additional control. There is currently no control over the transmission of credit card information over the internet. Hence without further mitigation, the current control is considered weak.

**Resulting likelihood: Medium**

**Fraud and theft by an insider**

The threat's motivation and capability:

- The attacker will have medium motivation if they are aiming for financial gain as the reward (theft of credit card information) is usually low compared to the job income they will lose. They will likely have a low capability as well.

The vulnerability's impact (and therefore attractiveness to an attacker):

- Fraud on meeting information will have relatively low impacts as the business disruption caused is unlikely to be costly. Theft on credit card information will have a higher impact as the attacker can then spend at will up to the credit card's limit. Hence the vulnerability's impact will be high.

The effectiveness of current controls:

- The credit card information will be store in the database which can be viewed by the administrator user without additional control. There is currently no control over the transmission of credit card information over the internet. Hence without further mitigation, the current control is considered weak.

**Resulting likelihood: Medium**

**Unauthorized system access by insider**

The threat’s motivation and capability:

- The attacker will have medium motivation if they are aiming for financial gain as the reward (theft of credit card information) is usually low compared to the job income they will lose. They will likely have a low capability as well.

The vulnerability’s impact (and therefore attractiveness to an attacker):

- Fraud on meeting information will have relatively low impacts as the business disruption caused is unlikely to be costly. Theft on credit card information will have a higher impact as the attacker can then spend at will up to the credit card’s limit. Hence the vulnerability’s impact will be high.

The effectiveness of current controls:

- In the current system design, the administrator account can only be created by another administrator. There is no option to upgrade the client’s account into the administrator account.
- The current system design will request for a strong password to be selected during admin account creation. However additional measures are required to encrypt the password and protect the password stored in the database. Hence without further mitigation, the current control is weak.

**Resulting likelihood: Medium**

### *2.4.5 Risk Impact determination*

This section describes the risk impact determination for the key threats identified. Note for threats that have the same threatened assets, business impact, and locality, their risk impact discussion are combined.

**Unauthorized system access by Cracker/Industry espionage / Competing Company/Insider**

- Threatened assets
  - The threatened assets will be the meeting information stored in the database, as well as the payment information (credit card information stored)
- Business Impact
  - As the meeting information stored will be of one-week duration only, the business impact will be minimal should the system become unavailable. The employees can still use alternative means such as verbal or instant messaging application communications to schedule the meetings.
- Locality
  - The impact will be local to the employees of the company.

**Resulting impact: Low**

**Computer crime ( Browsing of credit card information, Fraud, and theft) by Cracker**

- Threatened assets
  - The threatened assets will likely be the payment information (credit card information stored)
- Business Impact
  - Given the average credit card limit of $30,000. If the credit card information of 100 employees were stolen, the total damage will be around $3 million.
- Locality
  - The impact will be local to the credit cardholder.

**Resulting impact: High**

**Information theft by Industry espionage / competing company**

- Threatened assets
  - The threatened assets will be the meeting information stored in the database, as well as the payment information (credit card information stored)
- Business Impact
  - As the meeting information stored will be of one-week duration only, the business impact will be minimal as the meeting information is unlikely to carry much commercial value to the Industry espionage or competing company.
- Locality
  - The impact will be local to the employees of the company.

**Resulting impact: Low**

**Browsing of credit card information by an insider**

- Threatened assets
    - The threatened assets will likely be the payment information (credit card information stored)
- Business Impact
    - Given the average credit card limit of $30,000. If the credit card information of 100 employees were stolen, the total damage will be around $3 million.
- Locality
    - The impact will be local to the credit cardholder.

**Resulting impact: High**

**Fraud and theft by an insider**

- Threatened assets
    - The threatened assets will likely be the payment information (credit card information stored)
- Business Impact
    - Given the average credit card limit of $30,000. If the credit card information of 100 employees were stolen, the total damage will be around $3 million.
- Locality
    - The impact will be local to the credit cardholder.

**Resulting impact: High**

### 2.4.6 Overall Risk Exposure

The following table shows the overall risk exposure based on the analysis from the risk likelihood determination and risk impact determination.

| Main Risk | Risk Likelihood | Risk Impact | Risk Exposure |
|---|---|---|---|
| Unauthorized system access by Cracker | High | Low | Medium |
| Computer crime ( Browsing of credit card information, Fraud, and theft) by Cracker | High | High | High |
| Information theft by Industry espionage / competing company | Medium | Low | Low |
| Unauthorized system access by Industry espionage / competing company | High | Medium | High |
| Browsing of credit card information by an insider | Medium | High | Medium |
| Fraud and theft by an insider | Medium | High | High |
| Unauthorized system access by insider | Medium | Low | Low |

*Table 2: Overall Risk Exposure table*

### 2.4.7 Risk mitigation planning

To mitigate the risks outlined in section 2.4.6 caused by the common CWE and CAPEC described in section 2.4.3.1, the following risk mitigation actions will be employed.

To protect against computer crime (browsing of credit card information), Transport Layer Security (TLS) encryption protocol should be used to encrypt the transmission of the credit card information between the client browser and the web hosting server. Besides, Two-Factor Authentication (2FA) technique should be used to authenticate the client or the administrator before revealing the stored credit card information.

The two-factor authentication (2FA) technique can also help to strengthen the system's defense against unauthorized access. Client or Administrator may need to verify themselves through email confirmation or key in the pin received via SMS aside from using their username and password to log into the system. Other mitigation to reduce the likelihood of unauthorized access includes the use of a strong password format and encryption of passwords stored in the database. To further prevent attacks through the browsers, the system should automatically log out the users after a pre-set time -interval, to prevent others from using the login session of the previous user.

Input validation will be used to mitigate against code injection and SQL injection, to reduce the likelihood of information thefts. For SQL injection prevention, all PHP scripts interacting with the MySQL database objects will need to be written in the prepared and execute statements (Morgenroth, 2020).

## 3. Software Interface Design

### 3.1 System Interface Diagrams

#### *3.1.1 User Interface - Figure 3.1.1*

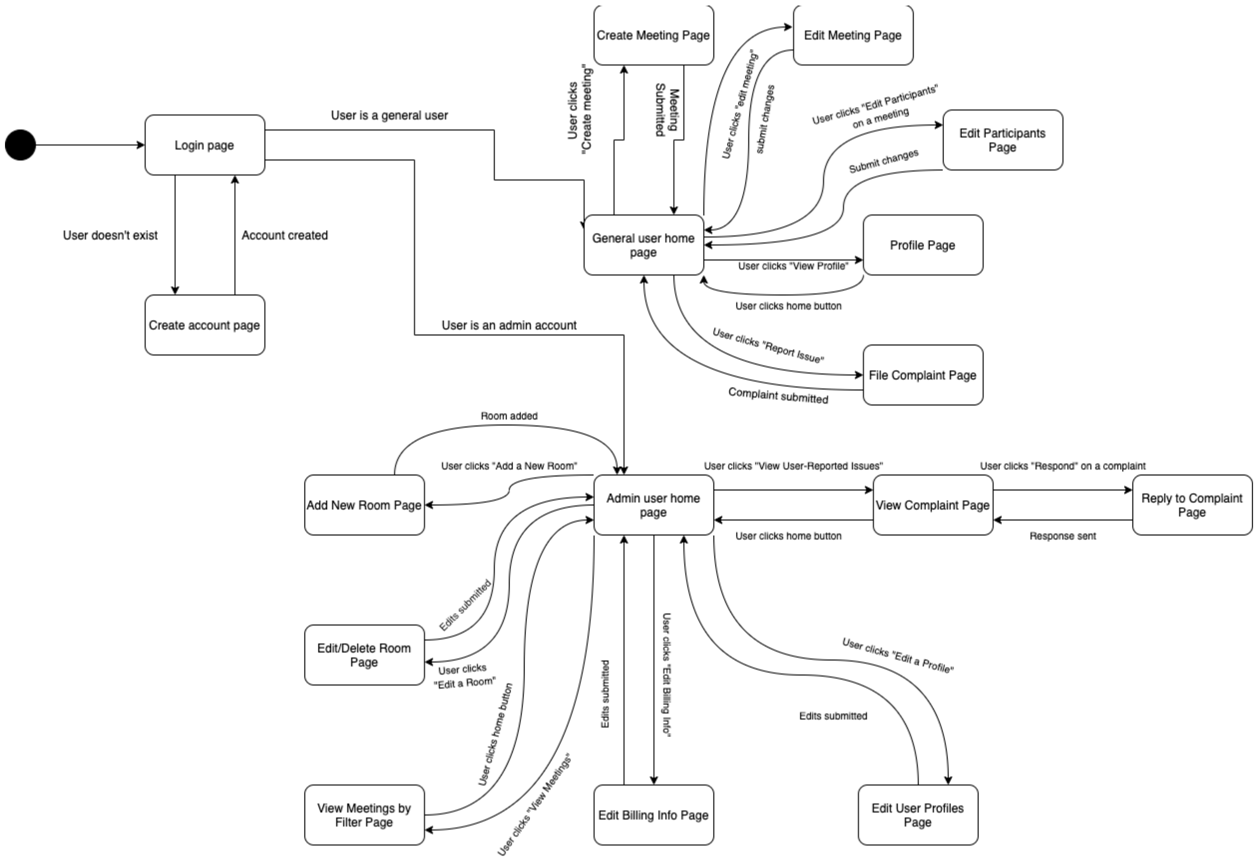
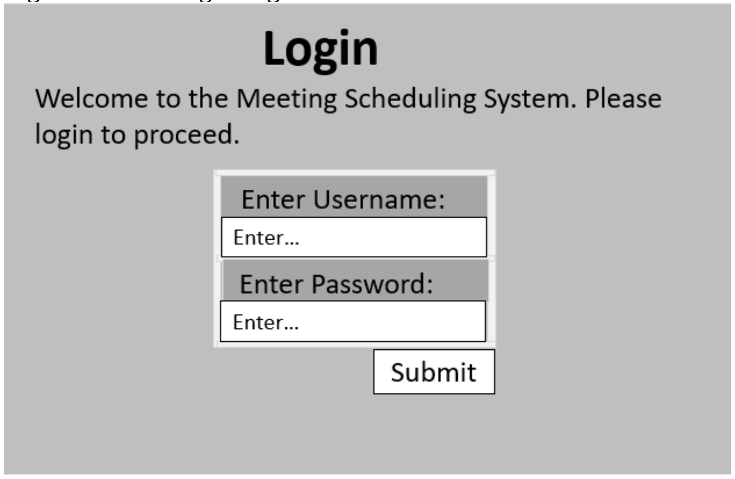


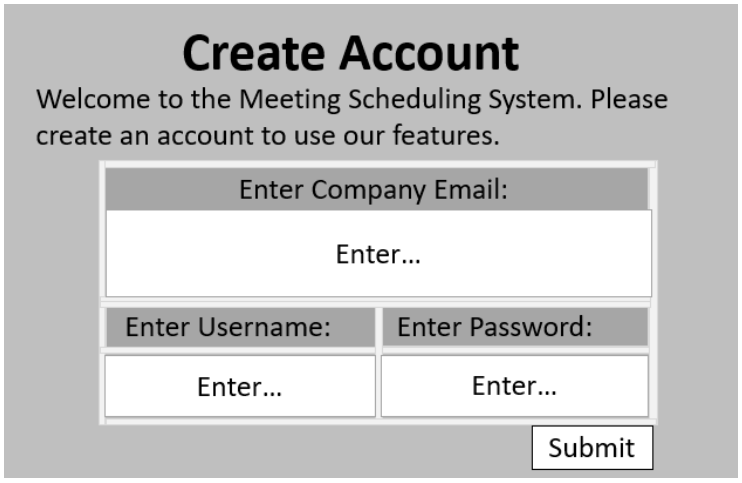
Figure 3.1.1 models all user interactions possible on the website. After logging in, the user is sent to a specific homepage based on whether or not they have administrative privileges (unless their account does not exist; then, they are prompted to create an account and sent to the Create Account page). All of the major functionalities of the website shall be accessible via the user's home page. The screenshots shown below model how the MSS's actual UI would look to a user.

Figure 3.1.1.1: Login Page


Login
Welcome to the Meeting Scheduling System. Please login to proceed.
Enter Username:
Enter...
Enter Password:
Enter...
Submit

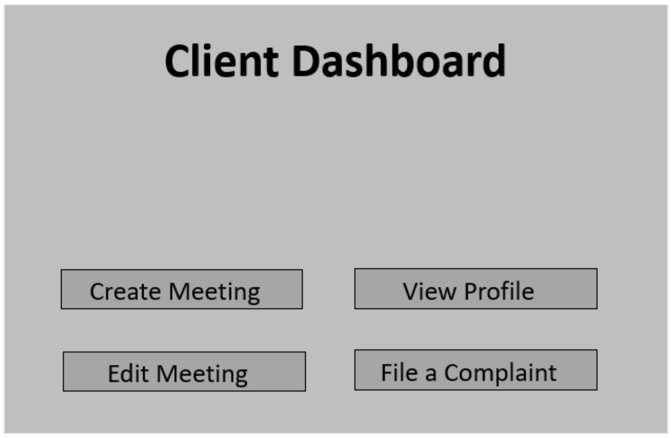
The user is greeted by a familiar interface on the login page. Here, they are prompted to enter their already existing username and password.

Figure 3.1.1.2: Create an Account page



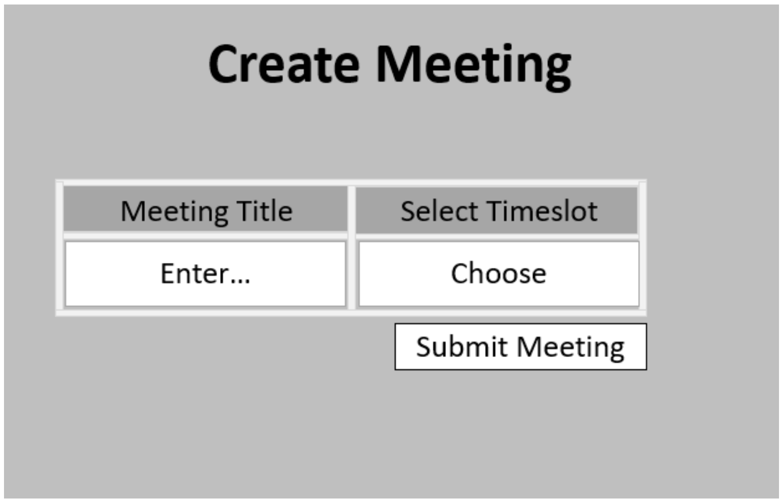
If the user's account does not already exist, the user will be prompted to create a new account. They will use their company-provided email to create their account. Further, they will need to enter a unique username and password to create their account.

Figure 3.1.1.3: Client Dashboard



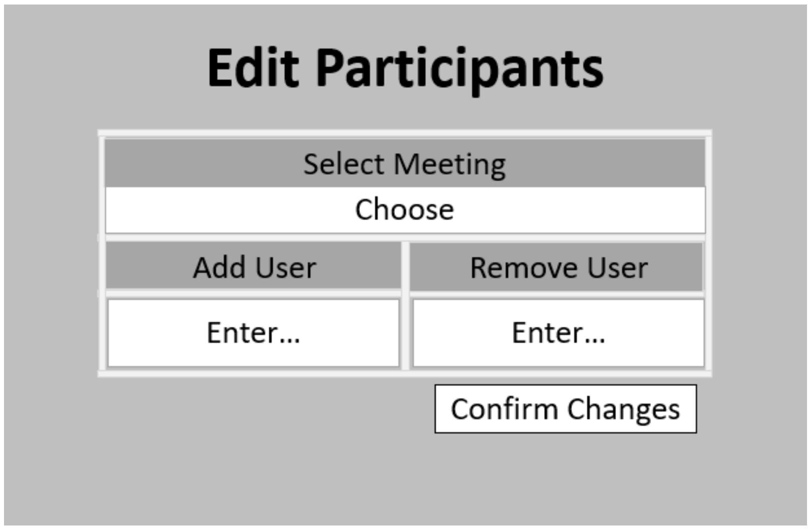
On the homepage, the user will be able to create a meeting, edit an existing meeting, view their custom profile, or file a complaint if they come across an issue with the website. Further, they will be able to view their upcoming meetings for the week on the homepage (not shown currently).

Figure 3.1.1.4: Create Meeting



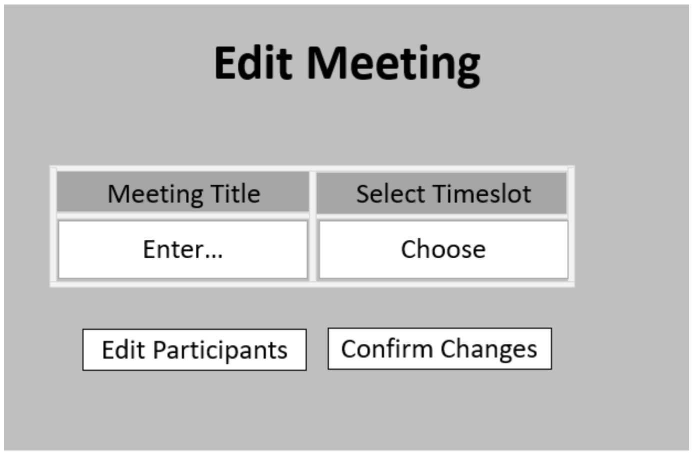
When the user selects "Create Meeting," they will be required to enter a name for the meeting, select the time the meeting will occur, add participants to the meeting, and choose a room for the room to take place in. If the room is already occupied, the user will be prompted to change the meeting location.

Figure 3.1.1.5: Edit Participants



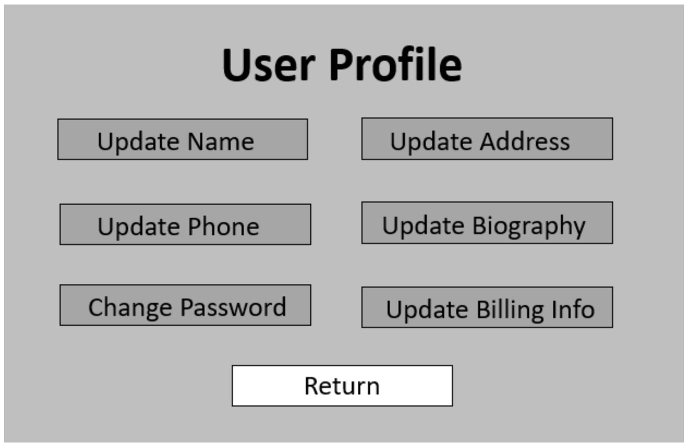
The “Edit Participants” option allows users to add or remove users from a meeting. The user simply needs to choose which meeting to add or remove participants from and search for the user by username.

Figure 3.1.1.6: Edit Meeting



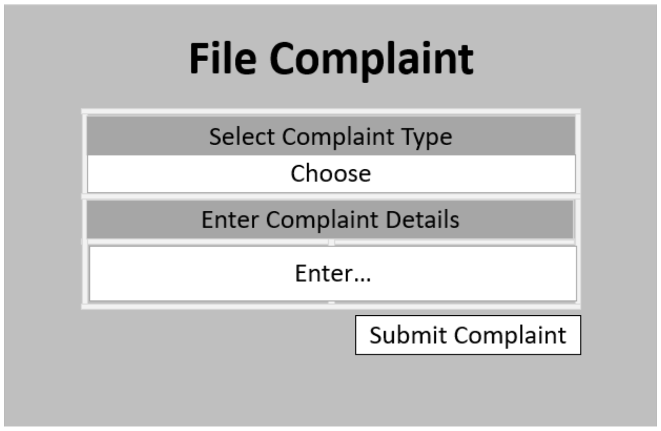
If a meeting needs to be changed, whether it be time, name, or location, the user can simply update the meeting data on the “Edit Meeting” page.

Figure 3.1.1.7: View Profile



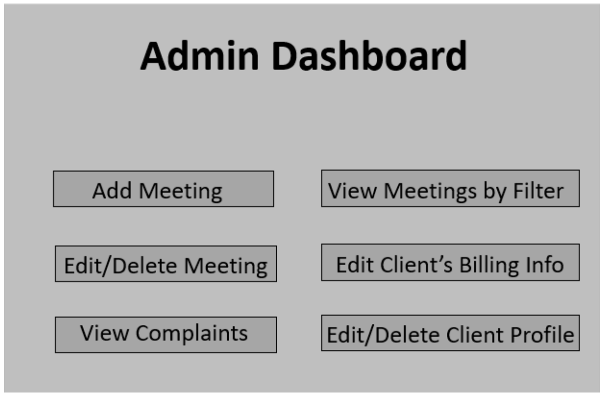
The user profile page allows the user to make basic changes to their profile. The user can change their display name, add their phone number, update their password, show their location, add a personal biography, or update their payment information used to book special rooms.

Figure 3.1.1.8: File Complaint



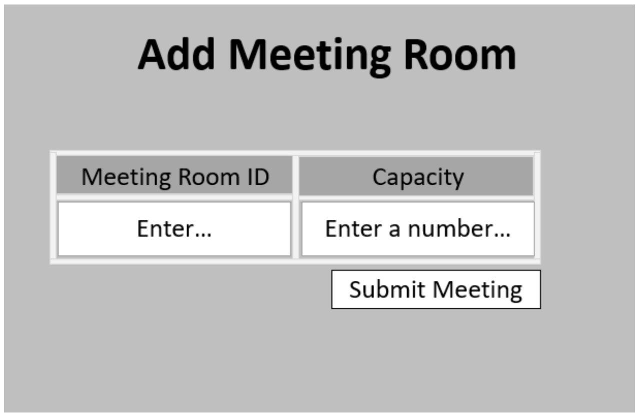
If the user needs to report a website issue, the user can choose "File Complaint" on the dashboard. The complaint can have a specific type (bug, typo, other, etc.) to briefly describe the issue. Further, the user can enter a more detailed description of the complaint before submitting it. Complaints are submitted to all system administrators.

Figure 3.1.1.9: Admin Dashboard



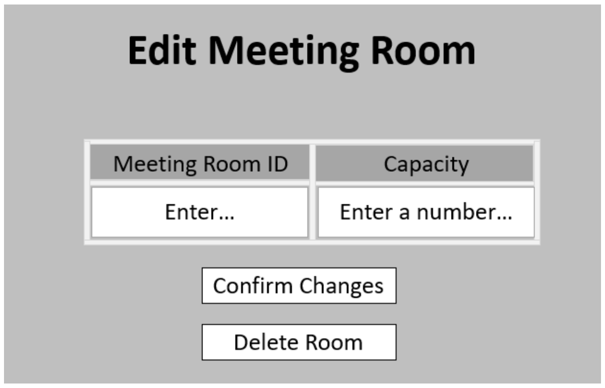
On the admin dashboard, administrators can add, edit, or delete rooms, view a list of complaints submitted by users, view scheduled meetings, edit a user's billing info, or edit or delete a general user's profile.

Figure 3.1.1.10: Add a New Room



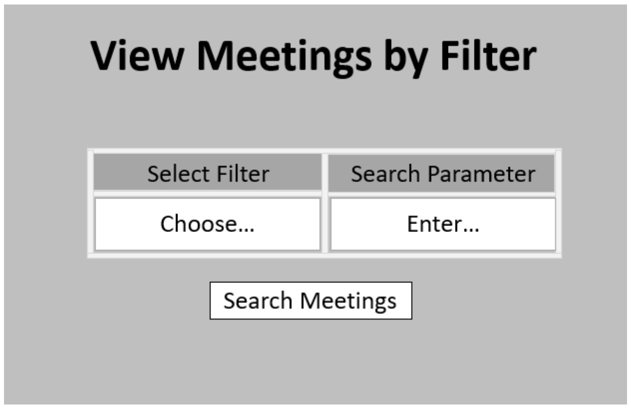
To add a room, the administrator needs to provide a room ID (which pertains to the room's location) and the capacity of the room. Once the new room is added, it will be made available for general users to schedule meetings.

Figure 3.1.1.11: Edit/Delete a Room



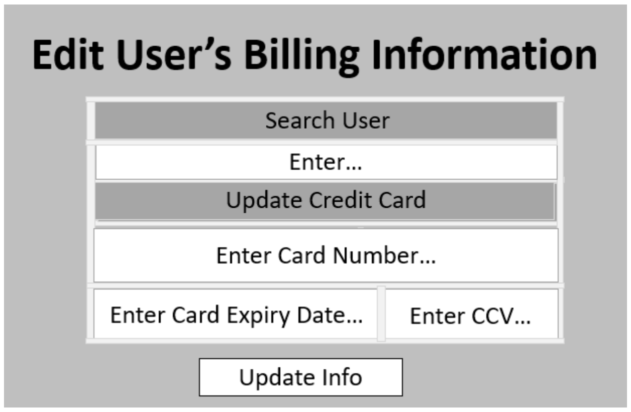
If the administrator needs to update a room's details or delete the room from the system, they can do so on the "Edit Meeting Room" page. The administrator can update the room's ID and capacity, as well as remove it from the database altogether.

Figure 3.1.1.12 View Meetings by Filter



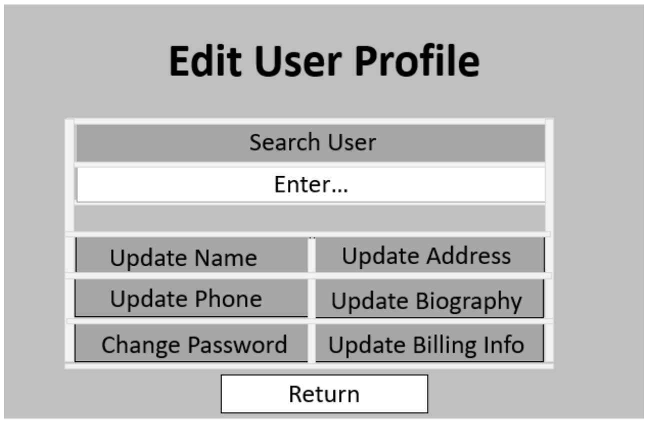
The administrator can view all meetings scheduled by any user on the "View Meetings by Filter" page. Here, they can filter the search results by location, meeting creator, and attendees.

Figure 3.1.1.13: Edit User's Billing Info



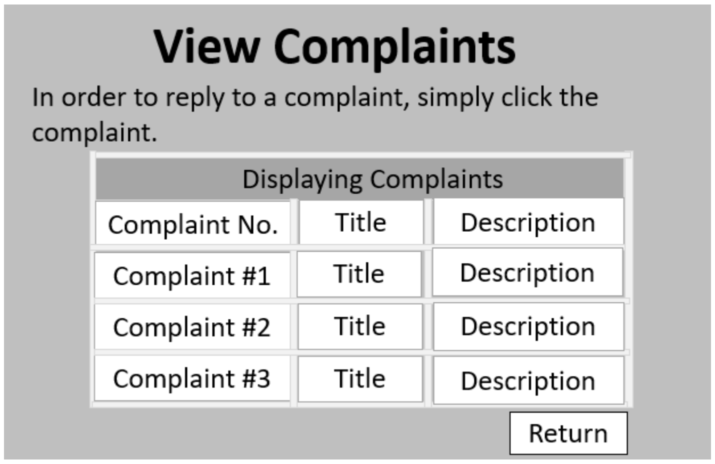
The admin can update billing information for a general user. Here, the administrator can add a company credit card, along with its expiry date and CVV. The admin can update billing information at any time and will be given a confirmation message on the successful completion of the update.

Figure 3.1.1.14 Edit User Profile



If an administrator needs to update a user profile on the user's behalf, all necessary changes can be made on the "Edit User Profile" page. It features all the same options to update as the general users see on their profile page, with the addition of a search bar to find a user by the username to update.

Figure 3.1.1.15: View Complaints

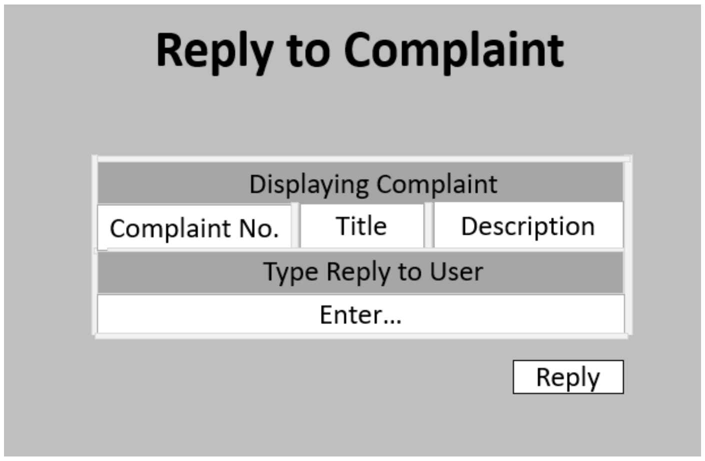


In order to reply to a complaint, simply click the complaint.

| Displaying Complaints | | |
|---|---|---|
| Complaint No. | Title | Description |
| Complaint #1 | Title | Description |
| Complaint #2 | Title | Description |
| Complaint #3 | Title | Description |

As part of an administrator's duty, they must address complaints submitted by general users. The "View Complaints" page allows the administrator to view and respond to user complaints.

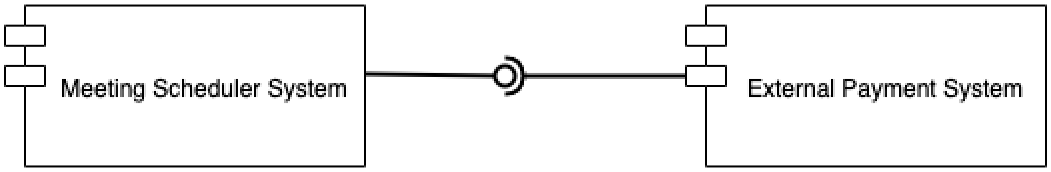
Figure 3.1.1.16: Reply to Complaint



| Displaying Complaint | | |
|---|---|---|
| Complaint No. | Title | Description |
| Type Reply to User | | |
| Enter... | | |

When a reply is submitted by an administrator, the user who submitted the original complaint is notified via the email attached to their user profile. The administrator would use this interface to acknowledge the issue's existence and inform the user of steps to take to rectify the problem.

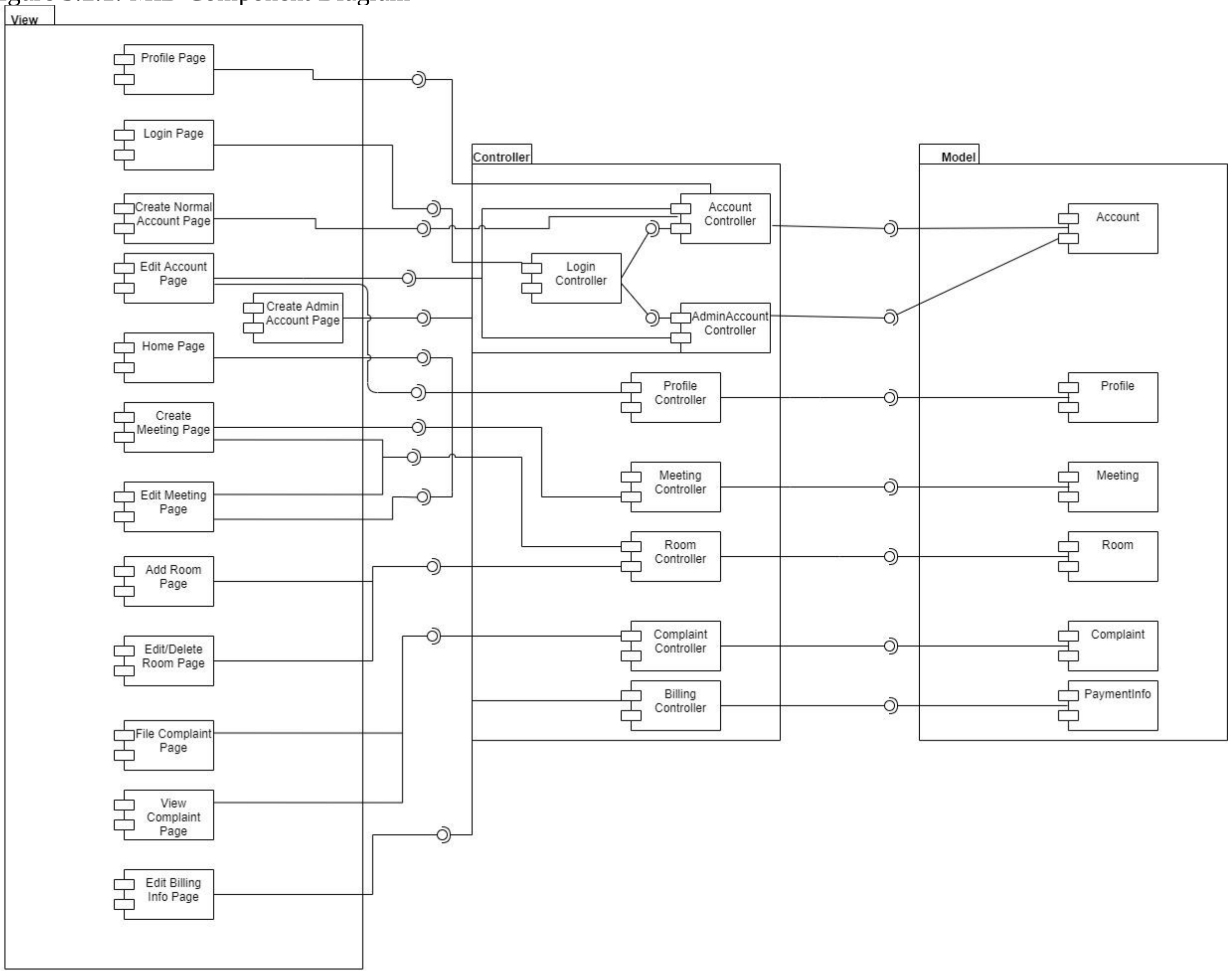
### *3.1.2 Software Interface - Figure 3.1.2 (external software systems, like payment, APIs, etc.)*



Currently, the system only requires the use of an external payment system to securely complete payments. The external system has not been chosen or implemented at the time of the writing of this document. Figure 3.1.2 models the relationship between the MSS and an external payment system.

## 3.2 Module Interface Diagrams - Figure 3.2 (internal)

Figure 3.2.1: MID Component Diagram



The MSS's main functionalities have been broken up into three packages based on the system's MVC architecture: models, views, and controllers. The view modules shown above represent each page and its associated classes and files that are required to display it for the user. The controller modules act as a middle man for their associated views and models, taking requests from the view and retrieving data from the model(s). The models can be classes or objects that have data stored for a particular item, like a meeting or a user. The figure shown below models the communications that happen among each module and between packages.

## 3.3 Dynamic Models of System Interface (Top 3 major functionalities for both general and admin)

### *3.3.1 Login Account Interface*

When a user visits the website page, the login account interface is shown and the user is prompted to log in with their username and password through text entry fields. Once the user clicks the "Login" button, the login account controller will take in the username and password and verify the credentials in the user account credentials database. If the login is successful, the website saves the user's login status and displays the user's dashboard (either a client or administrator dashboard, depending on the account details).

Figure 3.3.1.1: Login Sequence Diagram

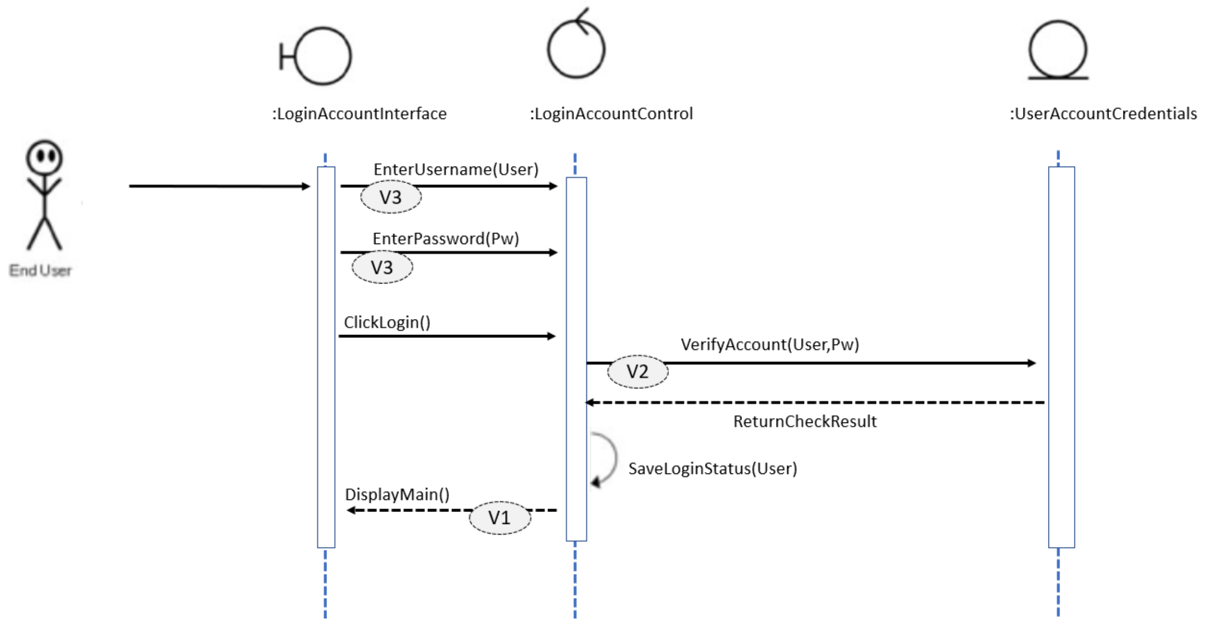


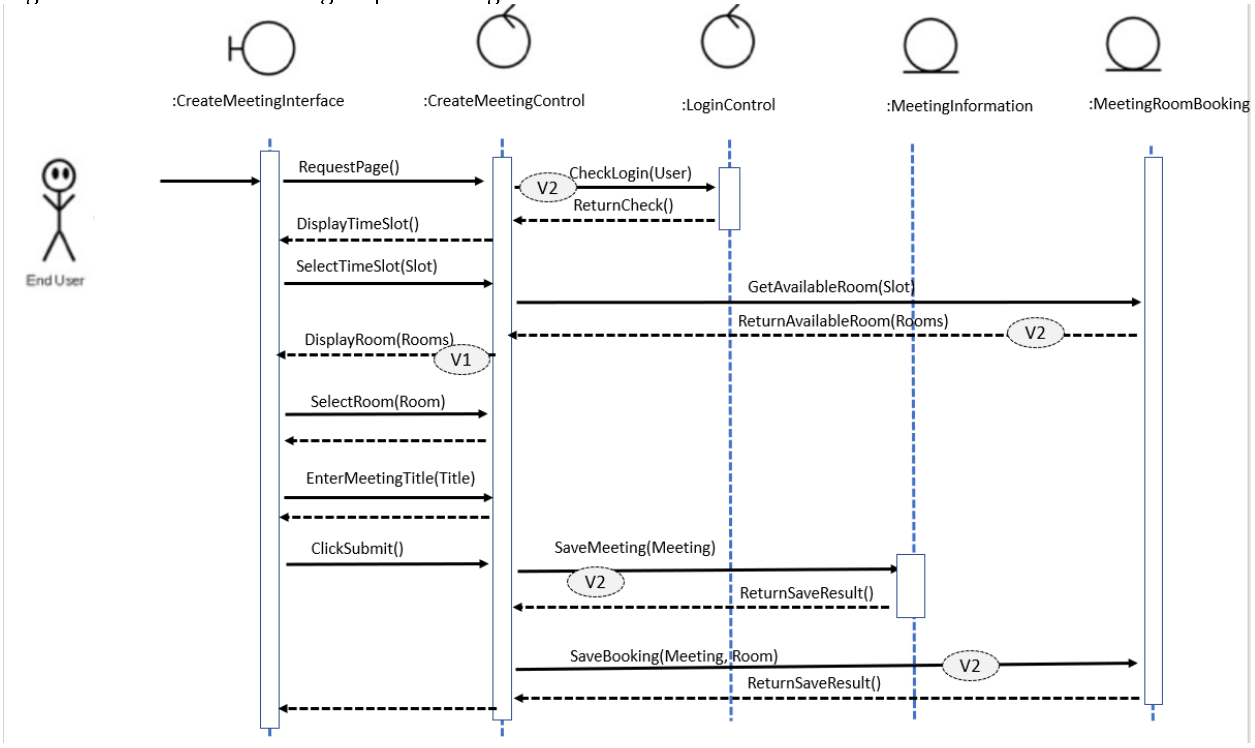
Figure 3.3.1.2: Login Vulnerability Mapping

| Action [Vulnerability] | Mitigation Plan and Control |
|---|---|
| EnterUsername [V3] | Encryption for data transmitted. |
| EnterPassword [V3] | Encryption for data transmitted. |
| VerifyAccount [V2] | Encryption for data transmitted. |

### *3.3.2 Create Meeting Interface*

When a client user clicks the “Create Meeting” button on their dashboard, the interface requests the page based on the user’s login status through the Login Controller. If the user is logged in, the user is shown the available timeslots. Once the user selects a timeslot, the create meeting controller checks whether any rooms have availability at the timeslot. The controller then sends the results to the create meeting interface, so that they can be displayed to the user. Once the user selects the room they wish to reserve their meeting in, the interface allows the user to enter the meeting title. Once the user enters the meeting title of their choice, the submit button will allow them to submit their request to reserve the meeting room. The controller saves the information to the meeting information database, and if successful, the booking is saved in the meeting room bookings database. If successful, the user will be informed that their meeting was successfully booked.

Figure 3.3.2.1: Create Meeting Sequence Diagram

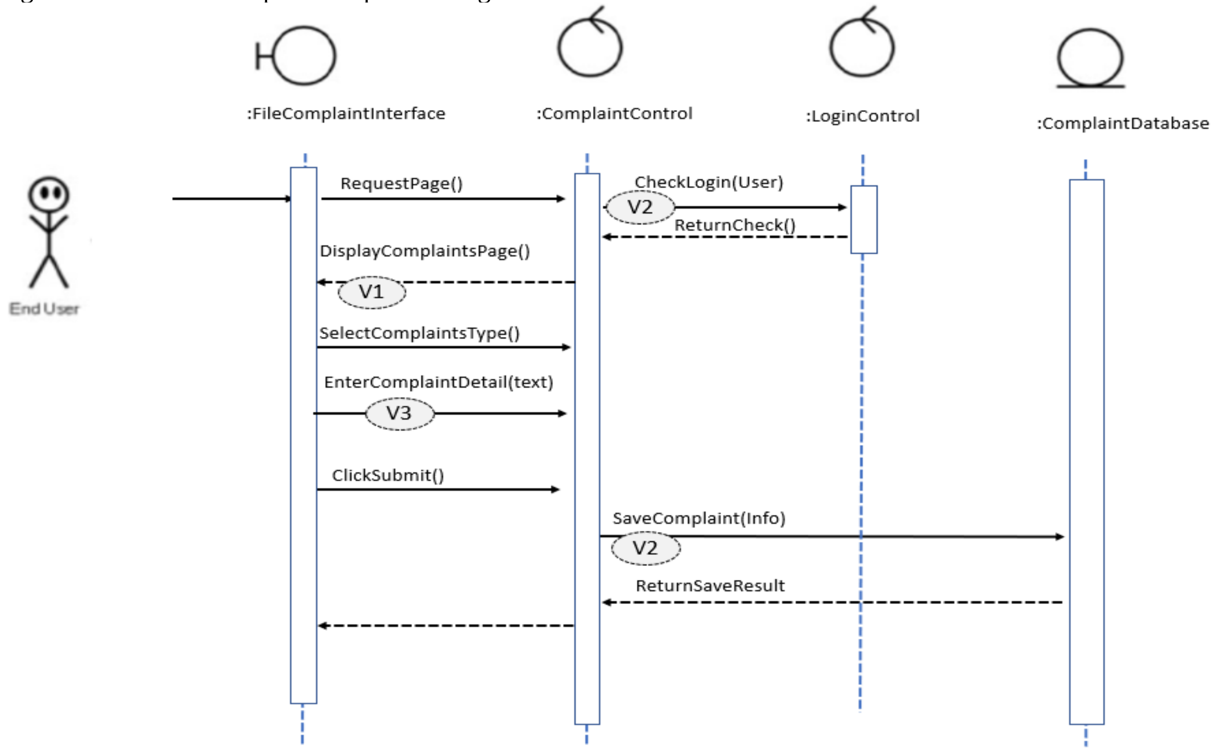


| Action [Vulnerability] | Mitigation Plan and Control |
|---|---|
| CheckLogin [V2] | Encryption for data transmitted. |
| ReturnAvailableRoom[V2] | Require authentication via CheckLogin. On authentication failure, the operation to retrieve the available rooms should stop.<br>Encryption for data transmitted. |
| SaveMeeting [V2] | Require authentication via CheckLogin. On authentication failure, the operation to save a new meeting should stop. Note the authentication status can be maintained if the user stays on the same page.<br>Encryption for data transmitted. |
| SaveBooking[V2] | Require authentication via CheckLogin. On authentication failure, the operation to save the room booking information should stop. Note the authentication status can be maintained if the user stays on the same page.<br>Encryption for data transmitted. |

### *3.3.3 File a Complaint Interface*

When the user clicks the “File Complaint” button on the dashboard, the file complaint controller first verifies whether the user is logged in or not through the login controller. If the user is logged in, the complaints page is displayed. The user is prompted to select a type of complaint in the dropdown menu. Once they do so, the user can enter their complaint details in a text box. Once they click “Submit”, the complaint controller sends the data to the complaint database, and if the data is successfully saved, the user is informed that their complaint was successfully sent to the administrators.

Figure 3.3.3.1: File Complaint Sequence Diagram



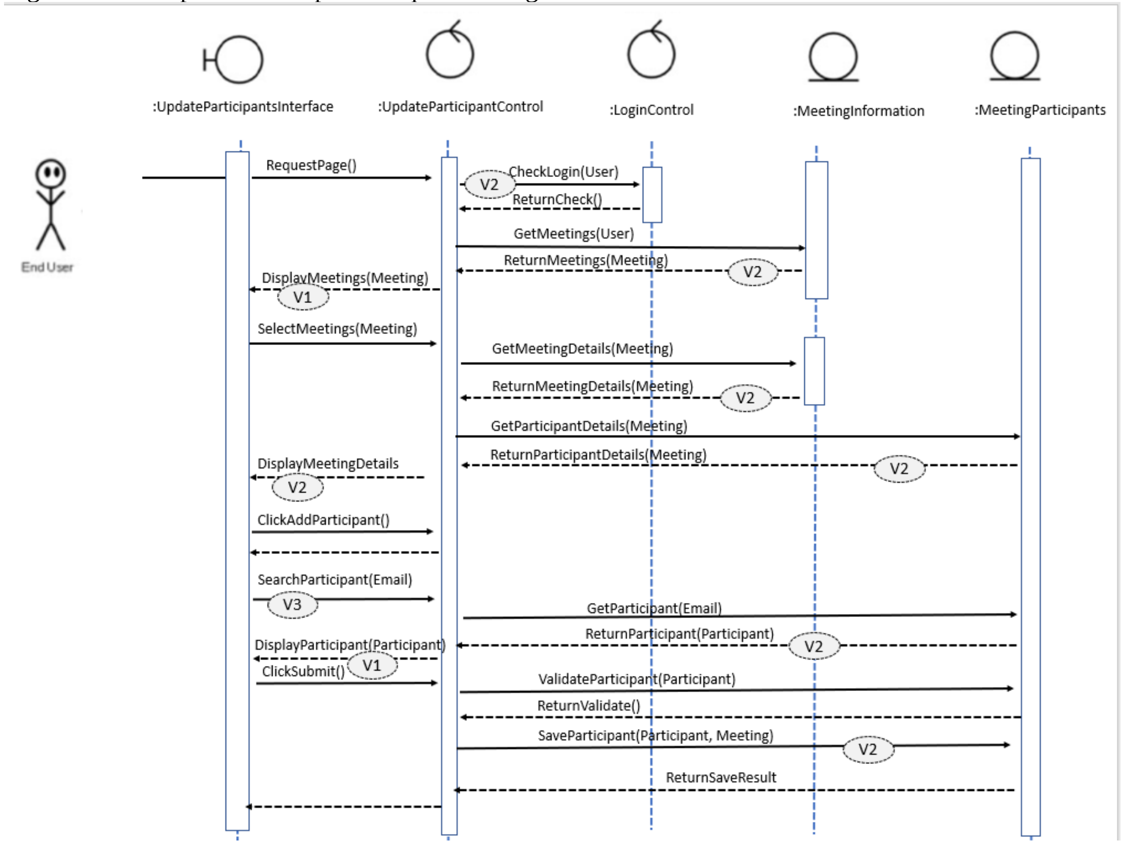
Mitigation plan and Control for vulnerability identified in the extended sequence diagram.

| Action [Vulnerability] | Mitigation Plan and Control |
|---|---|
| CheckLogin[V2] | Encryption for data transmitted. Note if the client navigates to this page after login their login status should be maintained across the pages. |
| EnterComplaintDetail[V3] | Encryption for data transmitted. |
| SaveComplaint[V2] | Encryption for data transmitted. |

### *3.3.4 Update Participants Interface.*

When the user clicks the "Edit Participants" button on the client dashboard, the update participants controller first needs to verify whether the user is logged in or not through the login controller. If the user was logged in, the meetings that they will host will be retrieved by the update participants controller from the meeting information database. Then, they will be displayed by the update participants interface for the user to see. Once the user clicks on a meeting that they would like to update the participants for, the update participants controller fetches the meeting participants from the meeting participants database. Then the results are displayed on the update participants interface for the user to see. If the user clicks the "Add Participant" button, the user is prompted to type in the participant's email address in a text field. Then, the update participants controller fetches the user given their email from the meeting participants database. If the fetch was successful, the participant is returned to the update participants interface. When the user clicks "Submit", the update participants controller verifies that the participant is not in another meeting during the meeting's timeslot through the meeting participants database. If the participant is not in another meeting, the update participants controller saves the participant to the meeting in the meeting participants database.

Figure 3.3.4.1 Update Participants Sequence Diagram



| Action [Vulnerability] | Mitigation Plan and Control |
|---|---|
| CheckLogin [V2] | Encryption for data transmitted. |
| ReturnMeetings[V2] | Require authentication via CheckLogin. On authentication failure, the operation to retrieve the meeting should stop.<br>Encryption for data transmitted. |
| ReturnMeetingDetails [V2] | Require authentication via CheckLogin. On authentication failure, the operation to retrieve the meeting details should stop. Note the authentication status can be maintained if the user stays on the same page.<br>Encryption for data transmitted. |

| | |
|---|---|
| DisplayMeetingDetails [V2] | Require authentication via CheckLogin. On authentication failure, the operation to display the meeting details should stop. Note the authentication status can be maintained if the user stays on the same page. Encryption for data transmitted. |
| SearchParticipant[V3] | Encryption for data transmitted. |
| ReturnParticipant[V2] | Require authentication via CheckLogin. On authentication failure, the operation to retrieve the participant's name should stop. Note the authentication status can be maintained if the user stays on the same page. Encryption for data transmitted. |
| SaveParticipant[V2] | Require authentication via CheckLogin. On authentication failure, the operation to save the participant in the meeting should stop. Note the authentication status can be maintained if the user stays on the same page. Encryption for data transmitted. |

### *3.3.5 Add New Meeting Room*

An administrator user is able to add a new meeting room to the system via the Add Room page. First, the user inputs the new room data (RoomID, capacity, etc.) to the page. Upon hitting submit, the MSS will validate the user input to protect against possible attacks (see figure 3.3.5.2 for the mitigation plan). Once validated, the RoomControl creates a new room in the MSS database with the room data. If no exceptions are thrown during execution, the user will be given a confirmation message containing the new room's data.

Figure 3.3.5.1: Add New Meeting Room Sequence Diagram

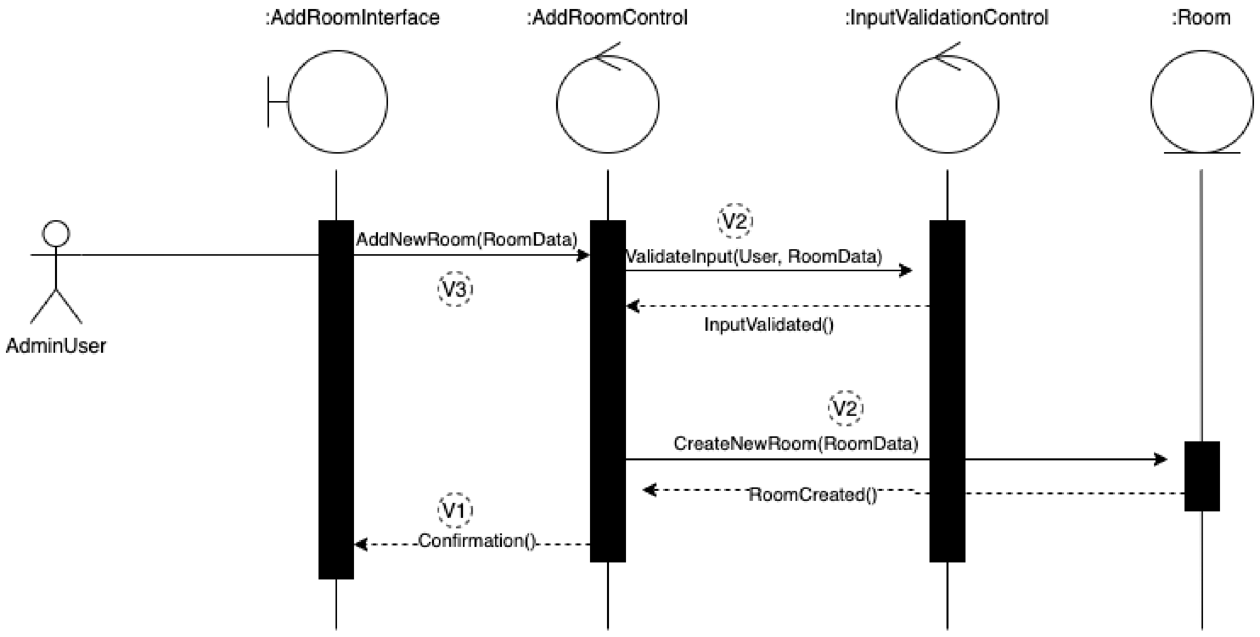


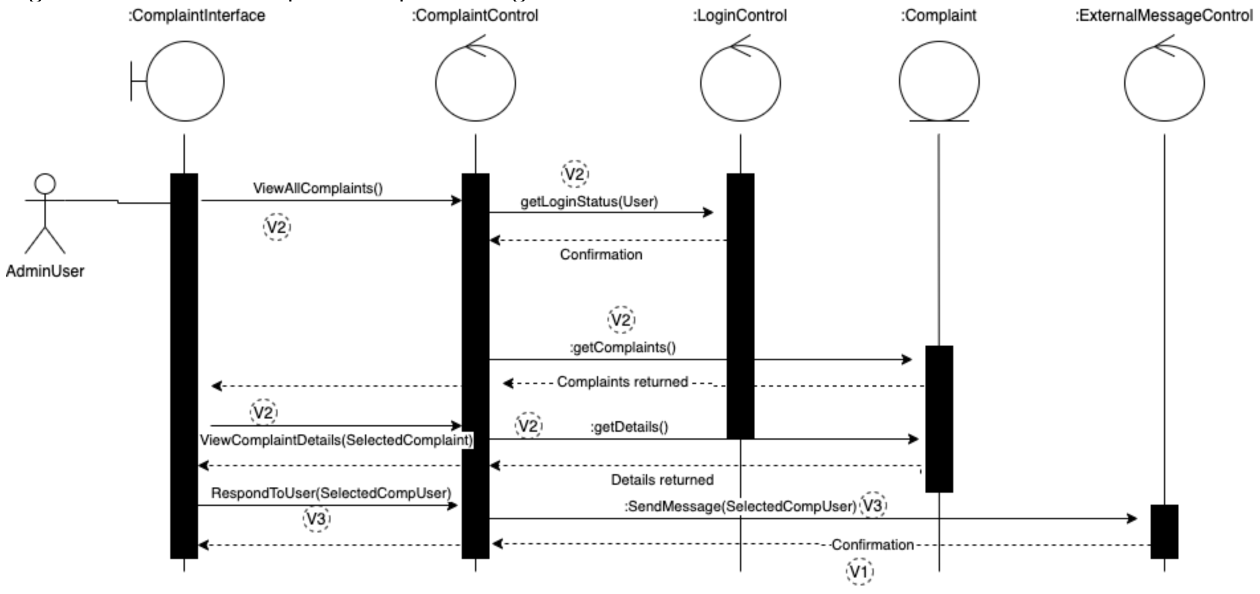
Figure 3.3.5.2: Add New Room Vulnerability Mapping

| Action [Vulnerability] | Mitigation Plan and Control |
|---|---|
| AddMeetingRoom[V3] | The administrator's login information should be re-verified prior to allowing a new room to be added. Further, all input should be validated prior to entering into a database to prevent malicious data from being entered (ValidateInput serves this purpose). |
| ValidateInput[V2] | All sensitive data being transmitted should be encrypted. Avoid excessive details from being visible. |
| CreateNewRoom[V2] | Any and all user data being transmitted should be encrypted. |
| Confirmation[V1] | There is potential for the page to not load properly. |

### *3.3.6 View Complaints*

Administrators have access to the list of complaints submitted by users of the website. These complaints are related to either user confusion or bugs on the site. First, the Administrator will view the View Complaints page. Before requesting the complaints, the page will verify that the current user is logged in. Then, it will retrieve the list of complaints in the database and sort them by most recent. The administrator can then select a complaint to view a more detailed description of the complaint. Finally, they would click the "Respond" button to send the user that submitted the complaint a message. The system will display a confirmation message once the message has been sent successfully.

Figure 3.3.6.1: View Complaints Sequence Diagram



| Action [Vulnerability] | Mitigation Plan and Control |
|---|---|
| ViewAllComplaints[V2] | Ensure that returned user data is encrypted. Further, the administrator’s login information should be re-verified prior to allowing access to the list of complaints. Necessary details should not be viewable until the administrator is verified. |
| getLoginStatus[V2] | Admin user data should be encrypted to prevent unauthorized access to the system. |
| getComplaints[V2] | As previously stated, any administrator should be verified prior to returning complaints (VerifyLogin performs this check). In addition, details pertaining to users that submitted complaints should be hidden during transmission. Only necessary data should be returned. |
| RespondToUser[V3] | Input should be validated to prevent SQL and script injection attacks. Further, messages should not be able to be sent unless the user is an administrator. Finally, user data must be left encrypted. |
| SendMessage[V3] | Encrypt sensitive data while sending messages. Again, further check the data being sent to avoid sending any malicious messages. |
| Confirmation[V1] | There is potential for the page to not load properly. |

#### *3.3.7 Change User Billing Info*

In the event an administrator needs to change a general user’s billing information (for example, if they needed to add a company card to their account to pay for a special room), the admin would visit the Update Billing Info page. First, the administrator can select a user to apply a new card to and enter the card information. Upon hitting submit, the system will first verify that the administrator is currently logged in. Then, the system will validate the administrator’s input to ensure no malicious input has been submitted. Finally, the system will submit the changes to the database and display a confirmation message to the user.

Figure 3.3.7.1: Change User Billing Info Sequence Diagram

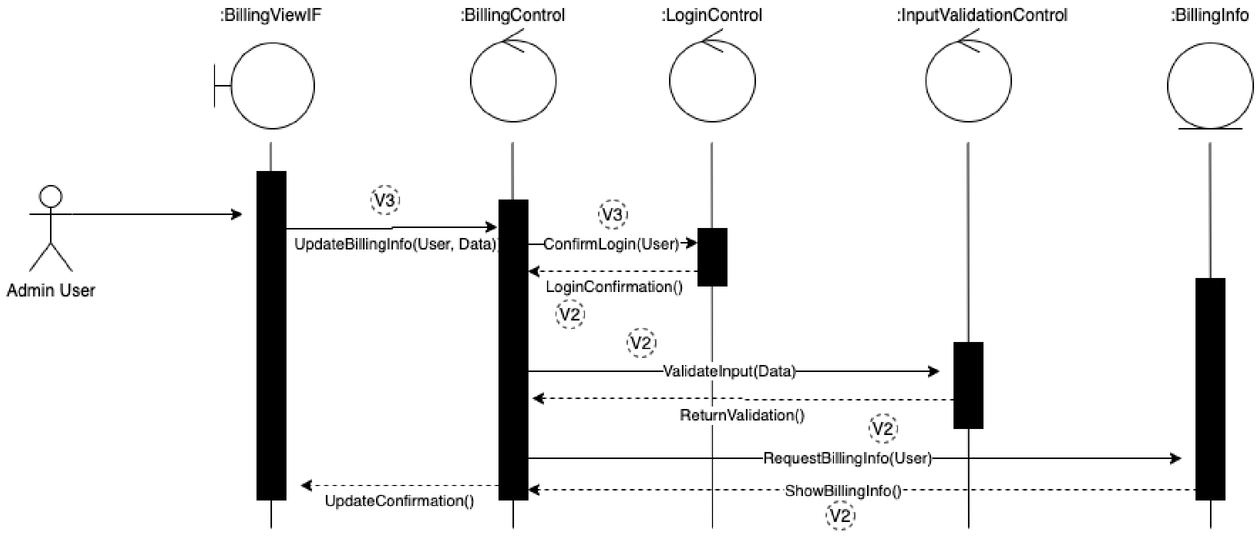
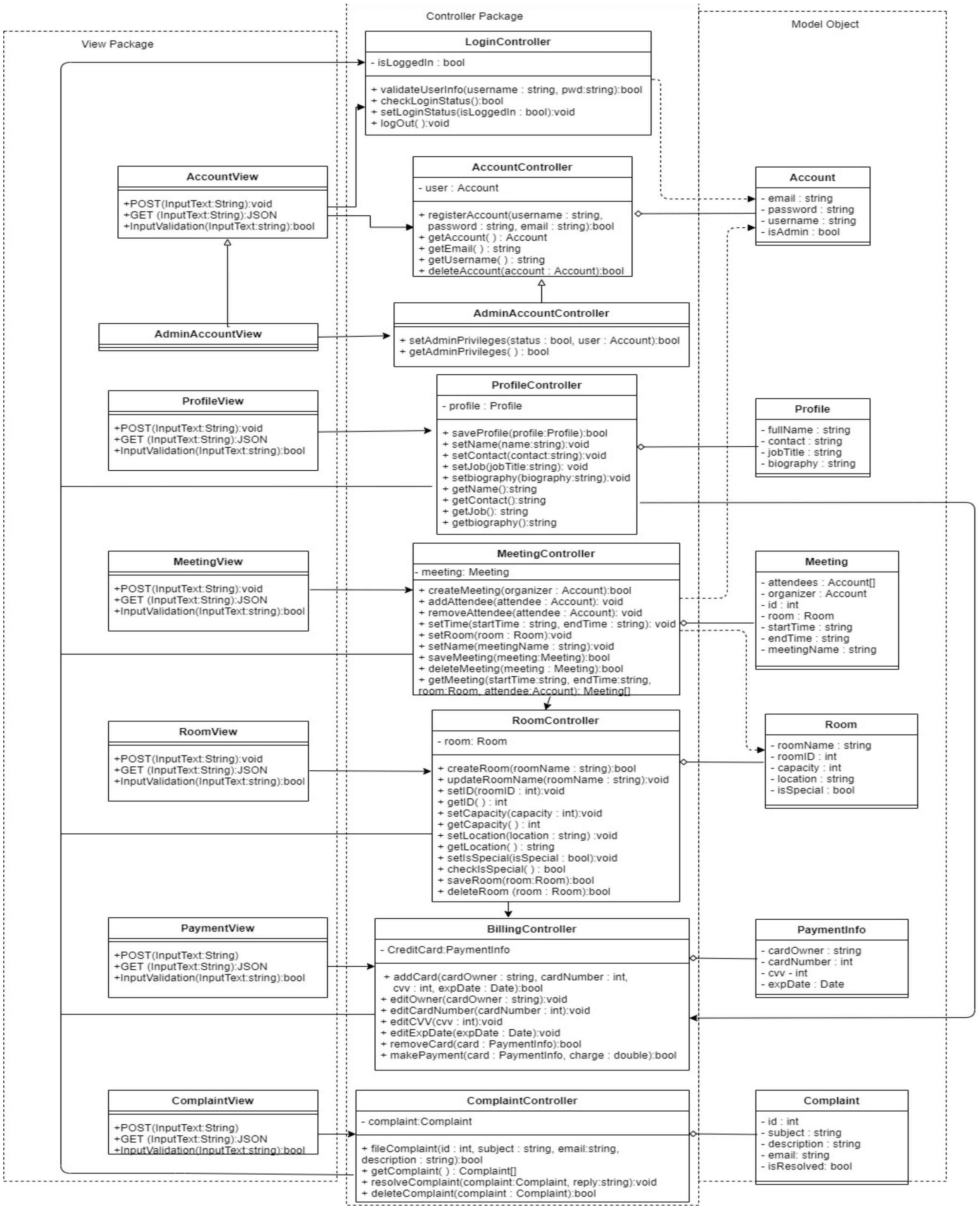


Figure 3.3.7.2: Change User Billing Info Vulnerability Mapping

| Action [Vulnerability] | Mitigation Plan and Control |
|---|---|
| UpdateBillingInfo[V3] | Encryption for data transmitted.<br>Required authentication via the ConfirmLogin(). On authentication failure, the operation to update BillingInfo should stops. |
| ConfirmLogin[V3] | Encryption for data transmitted. |
| ValidateInput[V2] | Encryption for data transmitted.<br>Required authentication, authentication status from previous step should be maintained. |
| RequestBillingInfo[V2] | Encryption for data transmitted.<br>Required authentication, authentication status from previous step should be maintained. |
| ShowBillingInfo[V2] | Encryption for data transmitted.<br>Required authentication, authentication status from previous step should be maintained. |

## 4. Internal Module Design

### 4.1 Overall Module Class Diagram


View Package
Controller Package
Model Object
LoginController
- isLoggedIn : bool
+ validateUserInfo(username : string, pwd:string):bool
+ checkLoginStatus():bool
+ setLoginStatus(isLoggedIn : bool):void
+ logOut( ):void
AccountView
+POST(InputText:String):void
+GET (InputText:String):JSON
+InputValidation(InputText:string):bool
AccountController
- user : Account
+ registerAccount(username : string, password : string, email : string):bool
+ getAccount( ) : Account
+ getEmail( ) : string
+ getUsername( ) : string
+ deleteAccount(account : Account):bool
Account
- email : string
- password : string
- username : string
- isAdmin : bool
AdminAccountView
AdminAccountController
+ setAdminPrivileges(status : bool, user : Account):bool
+ getAdminPrivileges( ) : bool
ProfileView
+POST(InputText:String):void
+GET (InputText:String):JSON
+InputValidation(InputText:string):bool
ProfileController
- profile : Profile
+ saveProfile(profile:Profile):bool
+ setName(name:string):void
+ setContact(contact:string):void
+ setJob(jobTitle:string): void
+ setbiography(biography:string):void
+ getName():string
+ getContact():string
+ getJob(): string
+ getbiography():string
Profile
- fullName : string
- contact : string
- jobTitle : string
- biography : string
MeetingView
+POST(InputText:String):void
+GET (InputText:String):JSON
+InputValidation(InputText:string):bool
MeetingController
- meeting: Meeting
+ createMeeting(organizer : Account):bool
+ addAttendee(attendee : Account): void
+ removeAttendee(attendee : Account): void
+ setTime(startTime : string, endTime : string): void
+ setRoom(room : Room):void
+ setName(meetingName : string):void
+ saveMeeting(meeting:Meeting):bool
+ deleteMeeting(meeting : Meeting):bool
+ getMeeting(startTime:string, endTime:string, room:Room, attendee:Account): Meeting[]
Meeting
- attendees : Account[]
- organizer : Account
- id : int
- room : Room
- startTime : string
- endTime : string
- meetingName : string
RoomView
+POST(InputText:String):void
+GET (InputText:String):JSON
+InputValidation(InputText:string):bool
RoomController
- room: Room
+ createRoom(roomName : string):bool
+ updateRoomName(roomName : string):void
+ setID(roomID : int):void
+ getID( ) : int
+ setCapacity(capacity : int):void
+ getCapacity( ) : int
+ setLocation(location : string) :void
+ getLocation( ) : string
+ setIsSpecial(isSpecial : bool):void
+ checkIsSpecial( ) : bool
+ saveRoom(room:Room):bool
+ deleteRoom (room : Room):bool
Room
- roomName : string
- roomID : int
- capacity : int
- location : string
- isSpecial : bool
PaymentView
+POST(InputText:String)
+GET (InputText:String):JSON
+InputValidation(InputText:string):bool
BillingController
- CreditCard:PaymentInfo
+ addCard(cardOwner : string, cardNumber : int, cvv : int, expDate : Date):bool
+ editOwner(cardOwner : string):void
+ editCardNumber(cardNumber : int):void
+ editCVV(cvv : int):void
+ editExpDate(expDate : Date):void
+ removeCard(card : PaymentInfo):bool
+ makePayment(card : PaymentInfo, charge : double):bool
PaymentInfo
- cardOwner : string
- cardNumber : int
- cvv - int
- expDate : Date
ComplaintView
+POST(InputText:String)
+GET (InputText:String):JSON
+InputValidation(InputText:string):bool
ComplaintController
- complaint:Complaint
+ fileComplaint(id : int, subject : string, email:string, description : string):bool
+ getComplaint( ) : Complaint[]
+ resolveComplaint(complaint:Complaint, reply:string):void
+ deleteComplaint(complaint : Complaint):bool
Complaint
- id : int
- subject : string
- description : string
- email: string
- isResolved: bool

### 4.2 View Module (Generic Class)

The following class is generic for all the UIs under the view module. In summary, the view class will have three generic methods, one to POST the input text to the controller, intended to save the input text into the database, and the second method is to GET the data from the database through the controller, returned as a JSON object. The third method is to validate the input to prevent code injection or SQL injection (see section 2 for the discussion). The method to encrypt and decrypt the data over the internet is not depicted here as these will be covered under the Transport Layer Security (TLS) protocol requirement. These generic classes are not depicted to have an inheritance relationship as each of them will be delivered independently to the client's terminal using HTML and JavaScript to realize the User Interface design.

#### *4.2.1 Generic View Class*

| | |
|---|---|
| **Class Name** | AccountView/AdminAccountView/ProfileView/MeetingView/RoomView/PaymentView/ComplaintView |
| **Inherits From** | N/A |
| **Description** | Display the UI page for a particular function, allow posting of data, and requesting data to the respective controller class. Provide input validation to eliminate code injection and SQL injection. |
| **Attributes** | |
| **Methods** | |

| *Visibility* | *Name* | *Description* |
|---|---|---|
| public | POST(inputText:string):void | POST the inputText entry into the controller class resides on the server PHP scripts. |
| public | GET(inputText:string):JSON | GET the requested data based on inputText from the controller class, returned as a JSON object. |
| public | inputValidation(inputText:string):bool | Provide input validation to filter out expression and symbol used for code injection or SQL injection, return a boolean value to indicate if the validation pass (true) or fail (false). |

| **Method Name** | POST(inputText:string):void |
|---|---|
| **Class Name** | AccountView/AdminAccountView/ProfileView/MeetingView/RoomView/PaymentView/ComplaintView |
| **Functionality** | POST the inputText entry into the controller class resides on the server PHP scripts. |
| **Input** | string inputText: a generic reference to all the form text input. It can be for the username, password, meeting title, and so on. |
| **Output** | N/A. Data passed to the respective controller class for further action. |

| **Method Name** | GET(inputText:string):JSON |
|---|---|
| **Class Name** | AccountView/AdminAccountView/ProfileView/MeetingView/RoomView/PaymentView/ComplaintView |
| **Functionality** | GET the requested data based on inputText from the controller class, returned as a JSON object. |
| **Input** | string inputText: a generic reference to the data object request. It can be for the user account, meetings, rooms, etc. |
| **Output** | JSON object containing the attribute values of the objects requested. |

| **Method Name** | POST(inputText:string):void |
|---|---|
| **Class Name** | AccountView/AdminAccountView/ProfileView/MeetingView/RoomView/PaymentView/ComplaintView |
| **Functionality** | POST the inputText entry into the controller class resides on the server PHP scripts. |
| **Input** | string inputText: a generic reference to all the form text input. It can be for the username, password, meeting title, and so on. |
| **Output** | N/A. Data passed to the respective controller class for further action. |

### 4.3 Module <Account>

#### *4.3.1 Class <AccountController>*

| | |
|---|---|
| **Class Name** | AccountController |
| **Inherits From** | N/A |
| **Description** | Creates, edits, and retrieves data for user accounts |

**Attributes**

| *Visibility* | *Data Type* | *Name* | *Description* |
|---|---|---|---|
| private | Account | user | A user account that the controller operates on |

**Methods**

| *Visibility* | *Name* | *Description* |
|---|---|---|
| public | registerAccount(username : string, password : string, email : string) : bool | Creates an Account object using the input username, password, and email as the account credentials. Save the account information into the database. |
| public | getAccount( ) : Account | Retrieves the account object stored in the “user” attribute |
| public | getEmail( ) : string | Retrieves the email associated with the account stored in the “user” attribute |
| public | getUsername( ) : string | Retrieves the username associated with the account stored in the “user” attribute |
| public | deleteAccount(account : Account): bool | Removes the input account object from the MSS database |

| | |
|---|---|
| **Method Name** | registerAccount() |
| **Class Name** | AccountController |
| **Functionality** | Creates an Account object using the input username, password, and email as the account credentials. Save the account information into the database. |
| **Input** | string username - the desired username for the newly created account<br>string password - the desired password for the newly created account<br>string email - the user's email to be associated with the newly created account |
| **Output** | A boolean value indicates if the account registration is a success (true) or failure (false). |

| | |
|---|---|
| **Method Name** | getAccount() |
| **Class Name** | AccountController |
| **Functionality** | Retrieves the account object stored in the "user" attribute of the AccountController class |
| **Input** | N/A |
| **Output** | Account - the Account object stored in the "user" attribute of the AccountController class |

| | |
|---|---|
| **Method Name** | getEmail() |
| **Class Name** | AccountController |
| **Functionality** | Retrieves the email associated with the account stored in the "user" attribute of the AccountController class |
| **Input** | N/A |
| **Output** | string - the email associated with the account stored in the "user" attribute of the AccountController class |

| | |
|---|---|
| **Method Name** | getUsername() |
| **Class Name** | AccountController |
| **Functionality** | Retrieves the username associated with the account stored in the "user" attribute of the AccountController class |
| **Input** | N/A |
| **Output** | string - the username associated with the account stored in the "user" attribute of the AccountController class |

| | |
|---|---|
| **Method Name** | deleteAccount() |
| **Class Name** | AccountController |
| **Functionality** | Removes the input account object from the MSS database |
| **Input** | Account object - the account to be permanently removed |
| **Output** | A boolean value indicates if the account deletion is a success (true) or failure (false). |

### *4.3.2 Class <Account>*

| | |
|---|---|
| **Class Name** | Account |
| **Inherits From** | N/A |
| **Description** | Account Object class holding the attributes such as the associated email, password, username, and whether the user is an admin. |
| **Attributes** | |
| **Methods** | |

| Visibility | Data Type | Name | Description |
|---|---|---|---|
| Private | string | email | Hold the user company email |
| Private | string | password | Hold the user password |
| Private | string | username | Hold the user username |
| Private | bool | isAdmin | Hold the boolean value whether the user is the admin(true) or not (false) |

## 4.4 Module <AdminAccount>

### *4.4.1 Class <AdminAccountController>*

| | |
|---|---|
| **Class Name** | AdminAccountController |
| **Inherits From** | Account Controller |
| **Description** | Checks and sets administrative privileges for user accounts |
| **Attributes** | |
| **Methods** | |

| *Visibility* | *Name* | *Description* |
|---|---|---|
| public | setAdminPrivileges(status : bool, user : Account) : bool | Changes the privilege level of the account specified to the level specified (0 for general user, 1 for administrative privileges) |
| public | getAdminPrivileges(user : Account) : bool | Retrieves the privilege level of the user-specified |

| | |
|---|---|
| **Method Name** | setAdminPrivilege() |
| **Class Name** | AdminAccountController |
| **Functionality** | Changes the privilege level of the account specified to the level specified (0 for general user, 1 for administrative privileges) |
| **Input** | bool status - the level of privilege to give the specified account<br>Account user - the user for which the status will be changed |
| **Output** | A boolean value indicating the change is a success (true) or fail (false) |

| | |
|---|---|
| **Method Name** | getAdminPrivilege() |
| **Class Name** | AdminAccountController |
| **Functionality** | Retrieves the privilege level of the user-specified (0 for general user, 1 for administrative privileges) |
| **Input** | user Account - the user whose privilege level is being checked |
| **Output** | bool - the privilege level of the user-specified, true for admin, false for the normal client. |

## 4.5 Module <Login>

### *4.5.1 Class <LoginController>*

| | |
|---|---|
| **Class Name** | LoginController |
| **Inherits From** | N/A |
| **Description** | Verifies the user's credentials and ensures the user is logged in before allowing them to use any functionalities of the MSS |
| **Attributes** | |

| *Visibility* | *Data Type* | *Name* | *Description* |
|---|---|---|---|
| private | bool | isLoggedIn | The current user's login status (0 for not logged in, 1 for logged in) |

**Methods**

| *Visibility* | *Name* | *Description* |
|---|---|---|
| public | validateUserInfo(username : string, password : string):bool | Verifies whether a user with the specified username/password combo exists in the database |
| public | checkLoginStatus( ) : bool | Retrieves the login status of the current user |
| public | setLoginStatus(status : bool):void | Sets the login status (0 for logged out, 1 for logged in) for the current user, stored in the "isLoggedIn" attribute |
| public | logOut():void | Sets "isLoggedIn" to false |

| | |
|---|---|
| **Method Name** | validateUserInfo() |
| **Class Name** | LoginController |
| **Functionality** | Verifies whether a user with the specified username/password combo exists in the database |
| **Input** | string username - the username input by the user logging in<br>string password - the password input by the user logging in |
| **Output** | A boolean value indicates the username and password combination exist (true) or does not exist (false) |

| | |
|---|---|
| **Method Name** | checkLoginStatus() |
| **Class Name** | LoginController |
| **Functionality** | Retrieves the login status of the current user |
| **Input** | N/A |
| **Output** | bool - 0 for logged out, 1 for logged in |

| | |
|---|---|
| **Method Name** | setLoginStatus() |
| **Class Name** | LoginController |
| **Functionality** | Changes the login status for the current user (represented by the “isLoggedIn” attribute of the LoginController class) to the input status |
| **Input** | bool status - 0 for logged out, 1 for logged in |
| **Output** | N/A |

| | |
|---|---|
| **Method Name** | logOut () |
| **Class Name** | LoginController |
| **Functionality** | Changes the login status for the current user (represented by the “isLoggedIn” attribute of the LoginController class) to false to show that the user has been logged out |
| **Input** | N/A |
| **Output** | N/A |

## 4.6 Module <Profile>

### *4.6.1 Class < ProfileController>*

| | |
|---|---|
| **Class Name** | ProfileController |
| **Inherits From** | N/A |
| **Description** | Provide functionality to view and edit the user profile information |
| **Attributes** | |

| *Visibility* | *Data Type* | *Name* | *Description* |
|---|---|---|---|
| private | Profile | profile | A Profile object holding the user’s profile information. |

**Methods**

| Visibility | Method Name | Description |
|---|---|---|
| public | saveProfile(profile:Profile):bool | Save the profile object into the database table |
| public | setName(name:string):void | Set the full name of the user |
| public | setContact(contact:string):void | Set the contact number of the user |
| public | setJob(jobTitle:string):void | Set the job title of the user |
| public | setBiography(biography:string): void | Set the biography of the user |
| public | getName():string | Get the full name of the user |
| public | getContact():string | Get the contact number of the user |
| public | getJob():string | Get the job title of the user |
| public | getBiography():string | Get the biography of the user |

| | |
|---|---|
| Method Name | saveProfile() |
| Class Name | ProfileController |
| Functionality | Save the profile object into the database table |
| Input | Profile object |
| Output | Boolean value indicating if the save is successful (true) or unsuccessful(false) |

| | |
|---|---|
| Method Name | setName() |
| Class Name | ProfileController |
| Functionality | Set the full name of the user |
| Input | Full name in string |
| Output | N/A |

| | |
|---|---|
| Method Name | setContact() |
| Class Name | ProfileController |
| Functionality | Set the contact number of the user |
| Input | Contact number as a string value |
| Output | N/A |

| | |
|---|---|
| Method Name | setJob() |
| Class Name | ProfileController |
| Functionality | Set the job title of the user |
| Input | Job title in string |
| Output | N/A |

| | |
|---|---|
| Method Name | setBiography() |
| Class Name | ProfileController |
| Functionality | Set the biography of the user |
| Input | biography in string |
| Output | N/A |

| | |
|---|---|
| Method Name | getName() |
| Class Name | ProfileController |
| Functionality | Get the full name of the user |
| Input | N/A |
| Output | Full name in string |

| | |
|---|---|
| Method Name | getContact() |
| Class Name | ProfileController |
| Functionality | Get the contact number of the user |
| Input | N/A |
| Output | Contact number as string value |

| | |
|---|---|
| Method Name | getJob() |
| Class Name | ProfileController |
| Functionality | Get the job title of the user |
| Input | N/A |
| Output | Job title in string |

| | |
|---|---|
| Method Name | getBiography() |
| Class Name | ProfileController |
| Functionality | Get the biography of the user |
| Input | N/A |
| Output | biography in string |

### *4.6.2 Class <Profile>*

| | |
|---|---|
| **Class Name** | Profile |
| **Inherits From** | N/A |
| **Description** | Profile Object class holding the attributes such as the full name, contact number, job title, and biography. |
| **Attributes** | |

| Visibility | Data Type | Name | Description |
|---|---|---|---|
| Private | string | fullName | Hold the user full name |
| Private | string | contact | Hold the user contact number |
| Private | string | jobTitle | Hold the user job title |
| Private | string | biography | Hold the user biography |

| | |
|---|---|
| **Methods** | |

## 4.7 Module <Meeting>

### *4.7.1 Class <MeetingController>*

| | |
|---|---|
| **Class Name** | MeetingController |
| **Inherits From** | N/A |
| **Description** | Create meetings and perform associated functions such as add attendee, remove attendee, setTime, setRoom, setName, and deleteMeeting. |

**Attributes**

| *Visibility* | *Data Type* | *Name* | *Description* |
|---|---|---|---|
| private | Meeting | meeting | A Meeting object holding the meeting information. |

**Methods**

| Visibility | Method Name | Description |
|---|---|---|
| public | createMeeting(organizer : Account):bool | Creates a meeting object with the organizer account username |
| public | addAttendee(attendee : Account):void | Add attendee to the meeting |
| public | removeAttendee(attendee : Account):void | Remove attendee from the meeting |
| public | setTime(startTime : string, endTime : string):void | Set the starting time and end time for the meeting |
| public | setRoom(room : Room):void | Book the room for the meeting |
| public | setName(meetingName : string):void | Set the title name for the meeting |
| public | saveMeeting(meeting:Meeting) :bool | Save the meeting object's information into the database. |
| public | getMeeting(startTime:string, endTime:string, room:Room, attendee:Account):Meeting[] | Get the list of meetings based on the arguments, startTime and endTime are used to extract list based on timeframe or time slots, the room is used to extract meeting list based on a specific room, the attendee is used to extract meeting list based on the attendee. Note the other arguments can be set to NULL, for example when want to extract a meeting list based on room, the rest of |

| | | | |
|---|---|---|---|
| | | | the three arguments can be set to NULL |
| | public | deleteMeeting(meeting : Meeting):bool | Delete this meeting |

| | |
|---|---|
| Method Name | createMeeting() |
| Class Name | MeetingController |
| Functionality | Creates a meeting object with the organizer account username |
| Input | Organizer account object |
| Output | A boolean value indicating the meeting creation is a success (true) or fail (false) |

| | |
|---|---|
| Method Name | addAttendee() |
| Class Name | MeetingController |
| Functionality | Add attendee to the meeting |
| Input | Attendee account object |
| Output | N/A |

| | |
|---|---|
| Method Name | removeAttendee() |
| Class Name | MeetingController |
| Functionality | Remove attendee from the meeting |
| Input | Attendee account object |
| Output | N/A |

| | |
|---|---|
| Method Name | setTime() |
| Class Name | MeetingController |
| Functionality | Set the starting time and end time for the meeting |
| Input | StartTime (String) and EndTime(String) |
| Output | N/A |

| | |
|---|---|
| Method Name | setRoom() |
| Class Name | MeetingController |
| Functionality | Book the room for the meeting |
| Input | Room object representing room to be booked |
| Output | N/A |

| | |
|---|---|
| Method Name | setName(meetingName : string) |
| Class Name | MeetingController |
| Functionality | Set the title name for the meeting |
| Input | meeting name in String |
| Output | N/A |

| | |
|---|---|
| Method Name | saveMeeting() |
| Class Name | MeetingController |
| Functionality | Save the meeting object's information into the database. |
| Input | Meeting object |
| Output | A boolean value indicates if the save is a success (true) or fail (false). |

| | |
|---|---|
| Method Name | deleteMeeting() |
| Class Name | MeetingController |
| Functionality | Delete this meeting |
| Input | Meeting object |
| Output | A boolean value indicates if the delete is a success (true) or fail (false). |

| | |
|---|---|
| Method Name | getMeeting() |
| Class Name | MeetingController |
| Functionality | Get the list of meetings based on the arguments, startTime, and endTime are used to extract the list based on timeframe or time slots, the room is used to extract meeting list based on a specific room, the attendee is used to extracting meeting list based on the attendee. Note the other arguments can be set to NULL, for example when want to extract a meeting list based on room, the rest of the three arguments can be set to NULL |
| Input | startTime and endTime in string, room in Room object, attendee in the Account object |
| Output | Array list of Meeting object: Meeting[] |

### *4.7.2 Class <Meeting>*

| | |
|---|---|
| **Class Name** | Meeting |
| **Inherits From** | N/A |
| **Description** | Meeting Object class holding the attribute name such as attendee, organizer, id, room, start time and end time, and meetingName |
| **Attributes** | |
| **Methods** | |

| Visibility | Data Type | Name | Description |
|---|---|---|---|
| Private | An array of Account Object | attendees | Hold the attendees' name |
| Private | Account object | organizer | Hold the organizer name |
| Private | Int | Id | Autogenerated id for the meeting |
| Private | Room object | Room | Hold the meeting room information |
| Private | String | startTime | Hold the startTime information |
| Private | String | endTime | Hold the endTime information |
| Private | String | meetingName | Hold the meetingName (title) information |

## 4.8 Module < Room>

### 4.8.1 Class <RoomController>

| | |
|---|---|
| Class Name | RoomController |
| Inherits From | N/A |
| Description | Create meeting room, update meeting room information, and set if the room is special, as well as provide a method to delete the room |
| Attributes | |

| *Visibility* | *Data Type* | *Name* | *Description* |
|---|---|---|---|
| private | Room | room | A Room object holding the room information. |

Methods

| Visibility | Method Name | Description |
|---|---|---|
| public | createRoom(roomName : string):bool | Creates a room object using roomName |
| public | updateRoomName(roomName : string):void | Update the roomName |
| public | setID(roomID : int):void | Set the roomID |
| public | getID( ): int | Get the roomID |
| public | setCapacity(capacity : int):void | Set the room capacity |
| public | getCapacity( ) : int | Get the room capacity |
| public | setLocation(location : string):void | Set the room location |
| public | getLocation( ) : string | Get the room location |
| public | setIsSpecial(isSpecial : bool):void | Set special for the room |
| public | checkIsSpecial( ) : bool | check if the room is special |
| public | saveRoom(room:Room):bool | save the room object information into the database |
| public | deleteRoom (room : Room):bool | Delete the room |

| | |
|---|---|
| Method Name | createRoom() |
| Class Name | RoomController |
| Functionality | Creates a room object using roomName supplied |
| Input | String (roomName) |
| Output | A boolean value indicating if the creation is a success(true) or fail(false) |

| | |
|---|---|
| Method Name | updateRoomName() |
| Class Name | RoomController |
| Functionality | Update the roomName |
| Input | String (roomName) |
| Output | N/A |

| | |
|---|---|
| Method Name | setID() |
| Class Name | RoomController |
| Functionality | Set the roomID |
| Input | integer (roomID) |
| Output | N/A |

| | |
|---|---|
| Method Name | getID() |
| Class Name | RoomController |
| Functionality | Get the roomID |
| Input | N/A |
| Output | integer (roomID) |

| | |
|---|---|
| Method Name | setCapacity() |
| Class Name | RoomController |
| Functionality | Set the room capacity |
| Input | integer (room capacity) |
| Output | N/A |

| | |
|---|---|
| Method Name | getCapacity() |
| Class Name | RoomController |
| Functionality | Get the room capacity |
| Input | N/A |
| Output | integer (room capacity) |

| | |
|---|---|
| Method Name | setLocation() |
| Class Name | RoomController |
| Functionality | Set the room location |
| Input | String (room location) |
| Output | N/A |

| | |
|---|---|
| Method Name | getLocation() |
| Class Name | RoomController |
| Functionality | Get the room location |
| Input | N/A |
| Output | String (room location) |

| | |
|---|---|
| Method Name | setIsSpecial() |
| Class Name | RoomController |
| Functionality | Set special for the room |
| Input | boolean value for attribute isSpecial (true for special, false for not) |
| Output | N/A |

| | |
|---|---|
| Method Name | checkIsSpecial( ) |
| Class Name | RoomController |
| Functionality | check if the room is special |
| Input | N/A |
| Output | boolean value for attribute isSpecial (true for special, false for not) |

| | |
|---|---|
| Method Name | saveRoom () |
| Class Name | RoomController |
| Functionality | save the room object information into the database |
| Input | Room object (selected by the administrator) |
| Output | A boolean value indicating if the save is a success(true) or fail(false) |

| | |
|---|---|
| Method Name | deleteRoom () |
| Class Name | RoomController |
| Functionality | Delete the room |
| Input | Room object (selected by the administrator) |
| Output | A boolean value indicating if the delete is a success(true) or fail(false) |

### *4.8.2 Class <Room>*

| | |
|---|---|
| Class Name | Room |
| Inherits From | N/A |
| Description | Room Object class holding the attribute name such as roomName, roomID, capacity, location and isSpecial |
| Attributes | |
| Methods | |

| Visibility | Data Type | Name | Description |
|---|---|---|---|
| Private | string | roomName | Name of the room |
| Private | int | roomID | The ID of the room |
| Private | int | capacity | Capacity (number of people can be accommodated) by the room |
| Private | string | Location | Location of the room |
| Private | bool | isSpecial | Boolean value indicating if the room is special |

## 4.9 Module <Billing>

### *4.9.1 Class <BillingController>*

| **Class Name** | BillingController |
|---|---|
| **Inherits From** | N/A |
| **Description** | This class controls all the operations that an administrator does to update a client's billing information |

**Attributes**

| *Visibility* | *Data Type* | *Name* | *Description* |
|---|---|---|---|
| private | PaymentInfo | CreditCard | PaymentInfo object contains the user credit card information |

**Methods**

| *Visibility* | *Name* | *Description* |
|---|---|---|
| public | addCard(cardOwner:string, cardNumber:int, cvv:int; expDate:Date):bool | Add the card with the information supplied to the database |
| public | editOwner(cardOwner:strong):void | Edit the cardOwner |
| public | editCardNumber(cardNumber:int):void | Edit the card number |
| public | editCVV(cvv:int):void | Edit the CVV number |
| public | editExpDate(expDate:Date):void | Edit the expiry date |
| public | removeCard(card:PaymentInfo) | Remove the selected credit card from the database |
| public | makePayment(card:PaymentInfo, charge:double):bool | Use the selected credit card to make payment |

| | |
|---|---|
| **Method Name** | addCard() |
| **Class Name** | BillingController |
| **Functionality** | Add the card with the information supplied to the database |
| **Input** | String – cardOwner: name of the card owner<br>Int – cardNumber: credit card number<br>Int - CVV: cvv number<br>Date-expDate: expiry date for the card |
| **Output** | A boolean value indicating if the addition is a success(true) or fail(false) |

| | |
|---|---|
| **Method Name** | editOwner() |
| **Class Name** | BillingController |
| **Functionality** | Edit the card owner name |
| **Input** | String cardOwner: card owner name |
| **Output** | N/A |

| | |
|---|---|
| **Method Name** | editCardNumber() |
| **Class Name** | BillingController |
| **Functionality** | Edit the card number |
| **Input** | Int – cardNumber: credit card number |
| **Output** | N/A |

| | |
|---|---|
| **Method Name** | editCVV |
| **Class Name** | BillingController |
| **Functionality** | Edit the CVV number |
| **Input** | Int - CVV: cvv number |
| **Output** | N/A |

| | |
|---|---|
| **Method Name** | editExpDate(expDate:Date):void |
| **Class Name** | BillingController |
| **Functionality** | Edit the expiry date |
| **Input** | Date-expDate: expiry date for the card |
| **Output** | N/A |

| | |
|---|---|
| **Method Name** | removeCard() |
| **Class Name** | BillingController |
| **Functionality** | Remove the selected credit card from the database |
| **Input** | PaymentInfo card: PaymentInfo object of the selected card |
| **Output** | A boolean value indicating if the deletion is a success(true) or fail(false) |

| | |
|---|---|
| **Method Name** | makePayment(card:PaymentInfo, charge:double):bool |
| **Class Name** | BillingController |
| **Functionality** | Use the selected credit card to make payment |
| **Input** | PaymentInfo card: PaymentInfo object of the selected card<br>Double charge: the total amount of charge in double data type. |
| **Output** | A boolean value indicating if the payment is a success(true) or fail(false) |

### *4.9.2 Class <PaymentInfo>*

| **Class Name** | PaymentInfo |
|---|---|
| **Inherits From** | N/A |
| **Description** | This Object class holds its parameters in the PaymentInfo database such as the credit card owner's name, credit card number, expiry date, and the CCV. |
| **Attributes** | |

| *Visibility* | *Data Type* | *Name* | *Description* |
|---|---|---|---|
| private | string | cardOwner | Name of the card owner |
| private | int | cardNumber | Credit card number |
| private | int | CCV | The CCV of the credit card in the database |
| private | Date | expDate | The expiry date of the credit card |

## 4.10 Module < Complaint>

### *4.10.1 Class <ComplaintController>*

| | |
|---|---|
| **Class Name** | ComplaintController |
| **Inherits From** | N/A |
| **Description** | This class handles all the operations that either an admin or client do regarding any complaints. |
| **Attributes** | |

| *Visibility* | *Data Type* | *Name* | *Description* |
|---|---|---|---|
| private | Complaint | complaint | The complaint a user is filing or the complaint being viewed by an admin. |

**Methods**

| *Visibility* | *Name* | *Description* |
|---|---|---|
| public | fileComplaint(id:int, subject:string, email:string, description:string):bool | Save the complaint with the supplied information into the database. |
| public | getComplaint():complaint[] | Retrieve the list of complaints from the database. |
| public | resolveComplaint(complaint:Complaint, reply:string):void | Allow the administrator to resolve the complaint by typing the reply and sending an email to the person. |
| public | deleteComplaint(complaint:Complaint):bool | Delete the selected complaint from the database |

| | |
|---|---|
| **Method Name** | fileComplaint() |
| **Class Name** | ComplaintController |
| **Functionality** | Save the complaint with the supplied information into the database. |
| **Input** | Int – id: complaint's id,<br>String – subject: title of the complaint,<br>String – email: email address of the person who complains<br>String – description: description text for the complaint |
| **Output** | A boolean value indicating if the saving is a success(true) or fail(false) |

| | |
|---|---|
| **Method Name** | getComplaint() |
| **Class Name** | ComplaintController |
| **Functionality** | Returns the list of complaints for the administrator to view. |
| **Input** | N/A |
| **Output** | An array list of Complaint object. Complaint[] |

| | |
|---|---|
| **Method Name** | resolveComplaint(complaint:Complaint, reply:string):void |
| **Class Name** | ComplaintController |
| **Functionality** | Allow the administrator to resolve the complaint by typing the reply and sending an email to the person. |
| **Input** | Complaint - complaint : the selected complaint object<br>string : reply : Reply text entered by the administrator |
| **Output** | N/A |

| | |
|---|---|
| **Method Name** | deleteComplaint() |
| **Class Name** | ComplaintController |
| **Functionality** | Delete the selected complaint from the database |
| **Input** | Complaint - complaint : the selected complaint object |
| **Output** | A boolean value indicating if the deletion is a success(true) or fail(false) |

### *4.10.2 Class <Complaint>*

| **Class Name** | Complaint |
|---|---|
| **Inherits From** | N/A |
| **Description** | This Object class holds its parameters in the Complaint database such as the complaint id, complaint's subject, complaint person's email, and complaint description. |
| **Attributes** | |

| *Visibility* | *Data Type* | *Name* | *Description* |
|---|---|---|---|
| private | Int | id | The id of the complaint |
| private | string | subject | The subject of the complaint |
| private | string | email | Email address of the person who makes the complaint |
| private | string | description | description of the complaint stored in the database. |
| private | Boolean | isResolved | A boolean value indicates if the complaint has been resolved (true) or not (false) |