PennStateSoft

Meeting
Scheduling System
Coding & Testing 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
13/8/2021	<1.0>	Submission	Chin Shiang Jin, Caralyn Harben, AbdAllah Mahassen

Table of Contents

Coding and Testing Document

1. Introduction

This Coding and Testing document outlines all the key details related to the coding of the final developed product and the associated testing results.

1.1 Purpose

The purpose of this Coding and Testing document is to serve as the guide for the software release and maintenance purpose. Result from the testing can be used as the input for the Incident Response Plan to be developed later.

1.2 Scope

The scope of this document includes the result of the static analysis performed on the codes, a complete documentation of the code which include the updated class diagram and documentation for each class and method, the testing result covered the test approaches used, bugs found and dealing with it, and the result of the functional and security testing. The complete codes and the deployment instructions will be submitted separately in Project Demo submission section.

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
CSRF	Cross Site Request Forgery
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.
Project Software Design Document from phase II project submission, titled SWENG455_Team4_MSS_ProjectPhase2

1.5 Overview

The document is organized as follows: section 2 shows the result of the static code analysis and potential future improvement method. Section 3 lists the updated documentation of the code which includes the updated class diagram, documentation for each class and method, and also the Entity Relation diagram for the database as well as the database schema used. Section 4 lists the testing results which includes the testing of the key functional and security features, notable bugs found and the way we dealing with the bugs.

2. Result of Static Code Analysis

The static analysis tool chosen is Embold (Website link: https://embold.io/sign-in/). This online tool can direct scan the GitHub repository to perform the static code analysis.

The analysis report generated by the static analysis tool is summarized in the figure 1. The overall rating is 2.11 for a 2.2K executable line of code. As shown in figure 2, there is no vulnerability or anti-pattern issues. The duplication density is high, the code issue density is critically high. Details of the code issues are discussed in section 2.1. The team will try to rectify the code issue and duplication whenever the time allows.

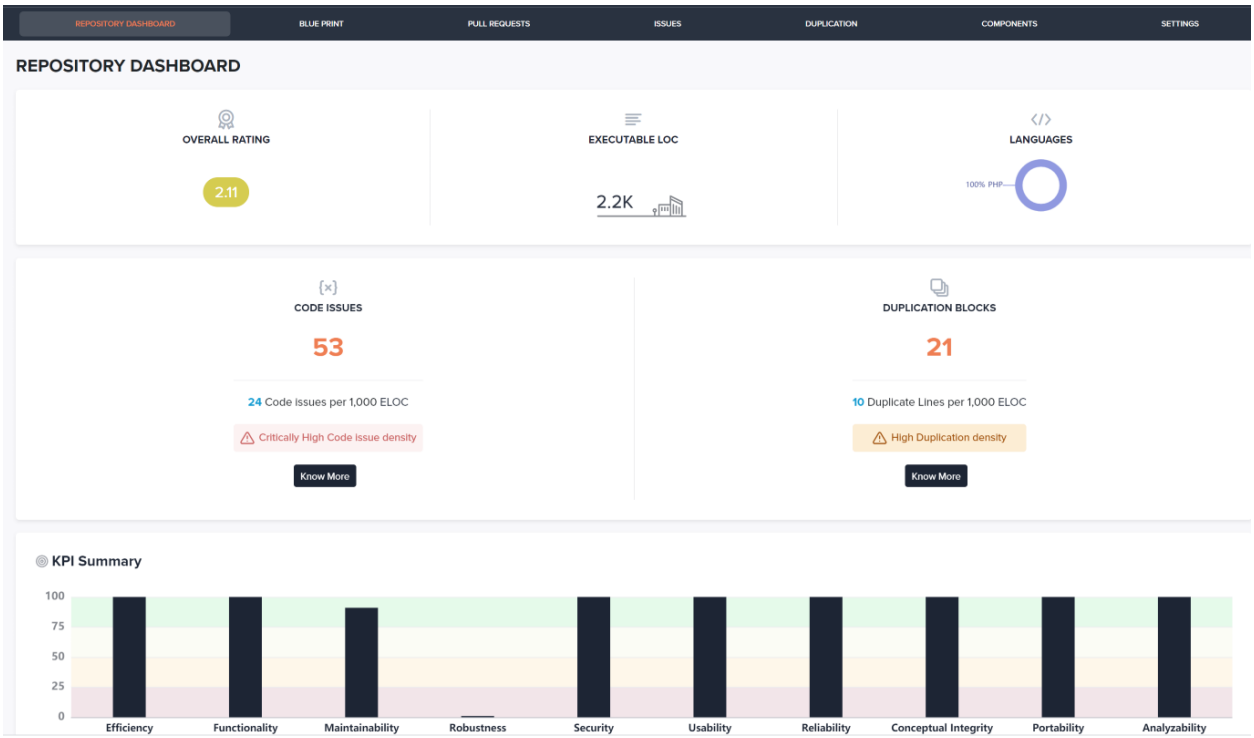


Figure 1: Embold Static Analysis Result

There is no vulnerabilities issue and anti-patterns issue.

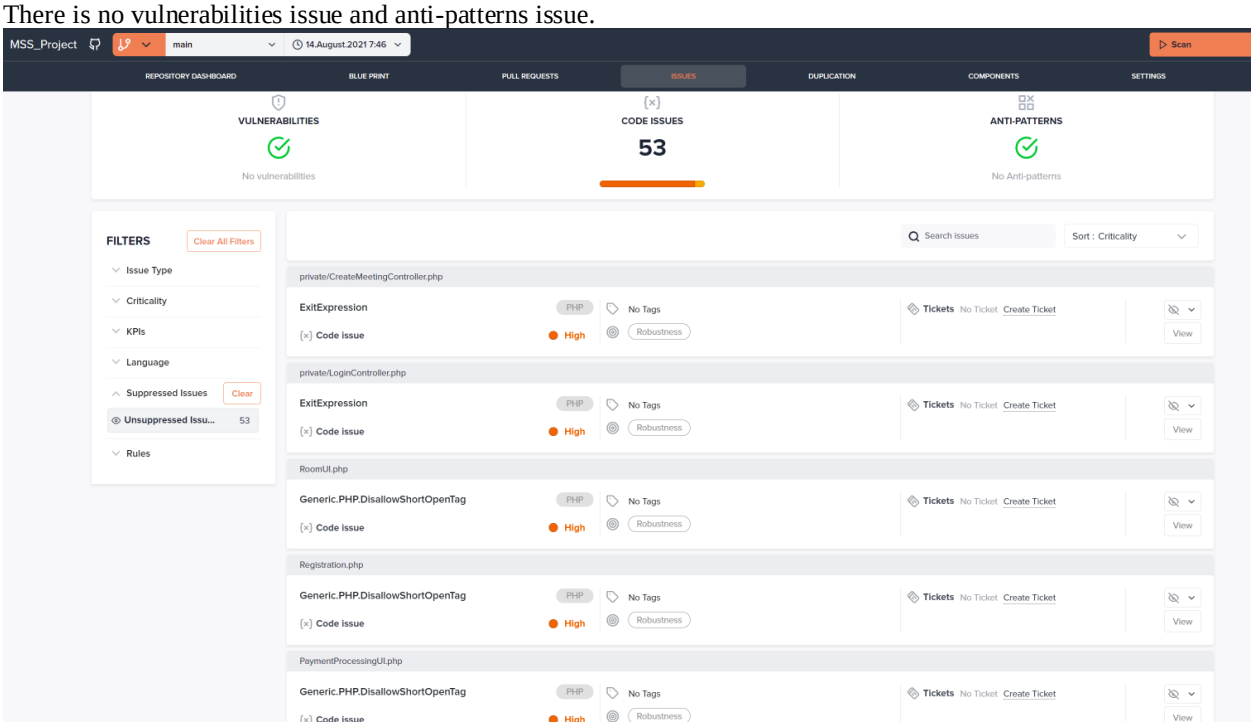


Figure 2: Vulnerabilities highlighted by the static analysis tool

2.1 Code Issues

A total of 53 code issues were highlighted, these can be grouped into following category:

- ExitExpression
- Generic.PHP DisallowShortOpenTag
- UnusedFormalParameter/LocalVariable/PrivateField

To mitigate the issues discovered by the static analysis tool, we will click on the details of the issue to understand how to improve the code. Looking at the first code issue related to ExitExpression, it was due to the use of ExitExpression in regular code. The recommendation is to avoid using it.

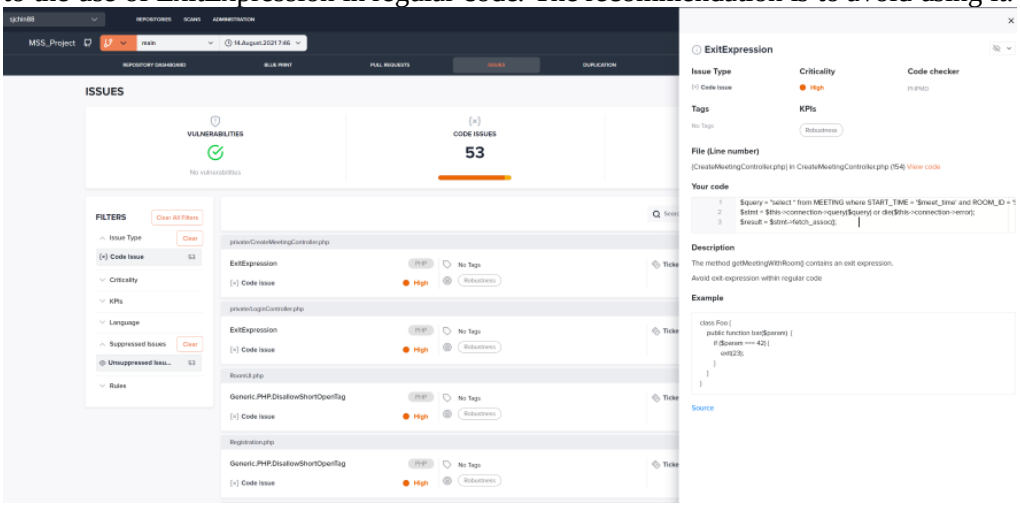


Figure 3: Drill down to the code issue

Refer to figure 4, the second group of Code issues is related to the use of Short PHP opening tag such as <?=?>. The recommendation is to avoid using shorthand PHP open tags altogether.

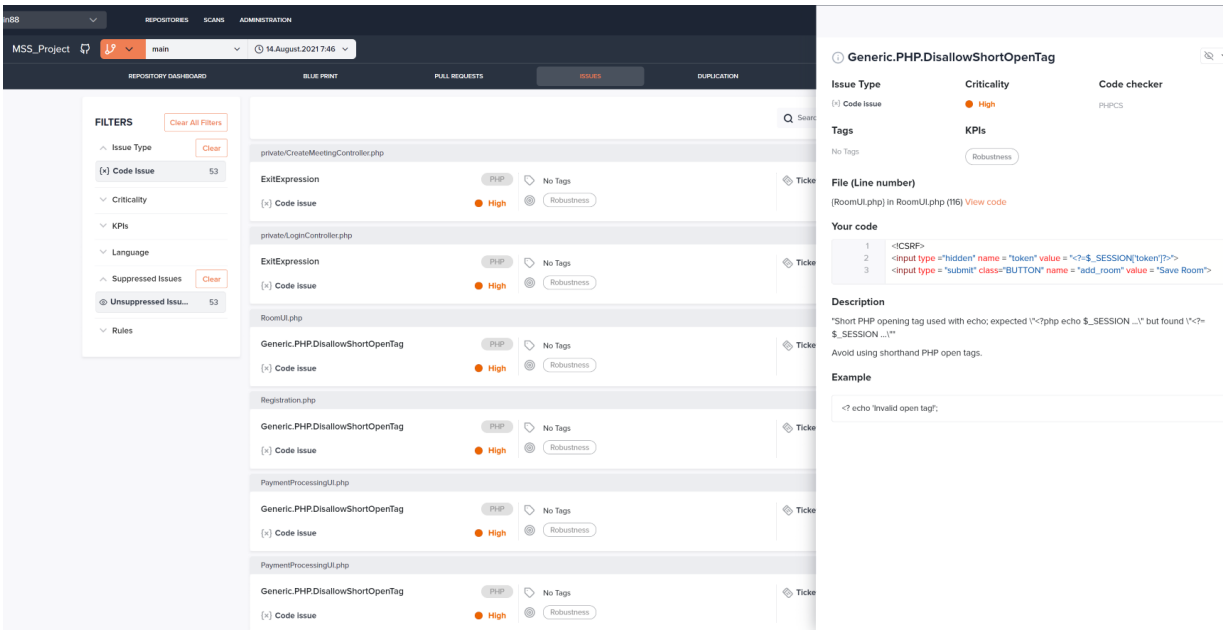


Figure 4: Code Issue Group 2 – DisallowShortOpenTag

Refer to figure 5, the third group of issues is related to unused private file, recommendation is to avoid declaring an unused private field.

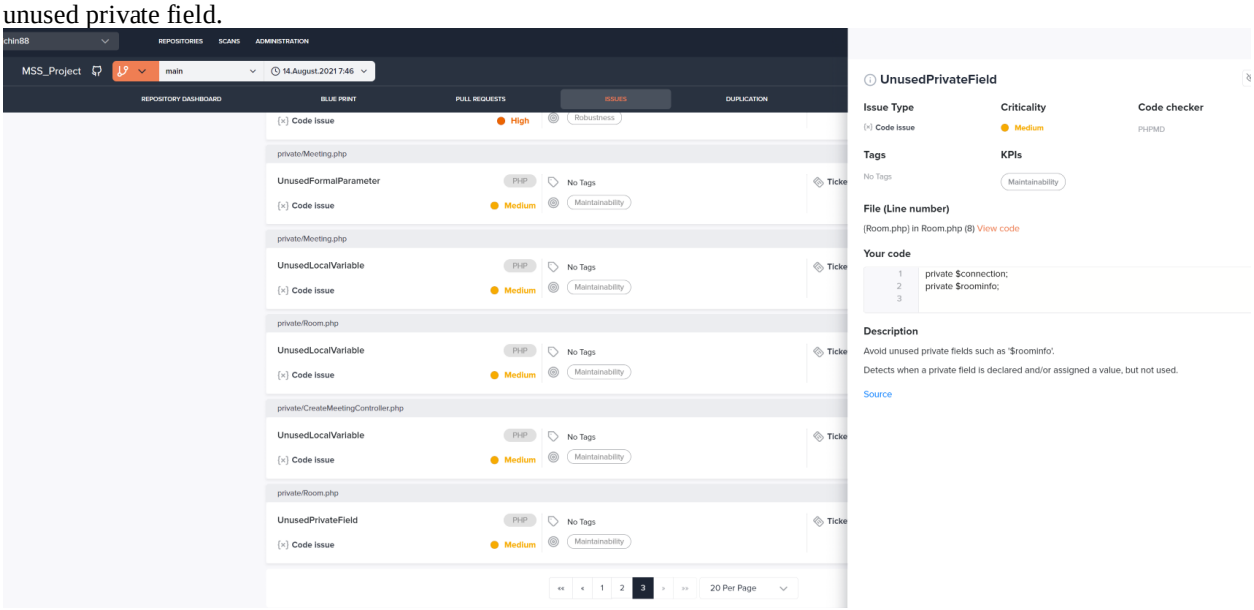


Figure 5: Code Issue Group 3: UnusedPrivateField

3. Complete Documentation of the Code

3.1 Updated Overall Module Class Diagram

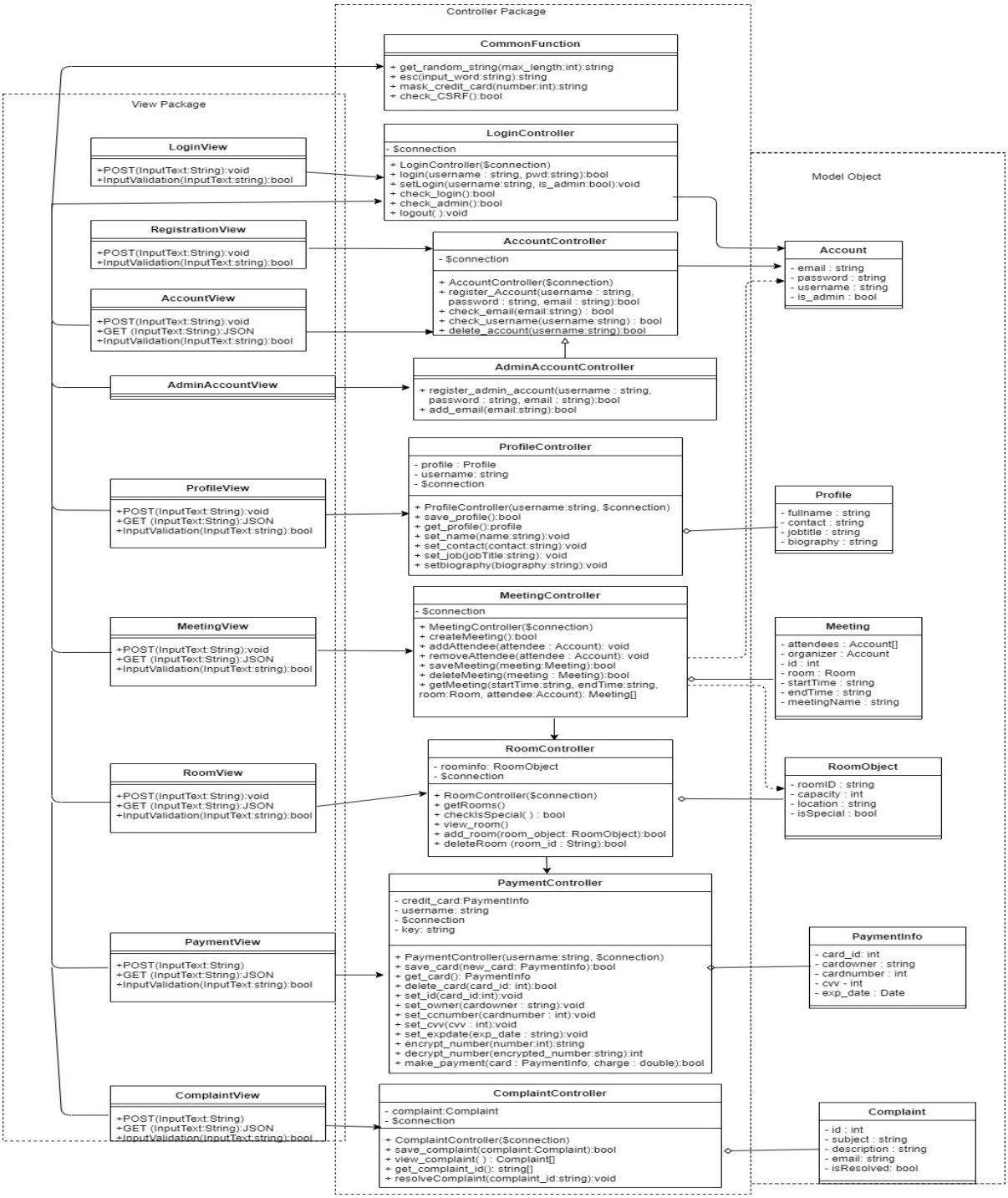


Figure 1: Updated Overall Module Class Diagram

Figure 1 shows the updated overall module class diagram for this software application project. In summary, the view package will consist of generic class similar to all the UIs. For the controller package, there is a collection of common functions usable by each of the controller class. There will be one controller class corresponding to each of the main functional modules, with the associated object class if necessary. In addition, all the view page will use the same LoginController class to manage the login status and session. Further details of the classes are available in the next sections.

3.2 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.

3.2.1 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.

3.3 Common Functions

Method Name	Get_random_string()
Class Name	N/A
Functionality	This function generates a random string with length between 5 to maximum length equal to the number given.
Input	Int - $max_length: An int number indicate the maximum length of string to be generated
Output	String - $text: A random string generated

Method Name	Esc()
Class Name	N/A
Functionality	This function sanitizes the input text by adding slashes to every characters of the text. This is to prevent SQL injection.
Input	String – input_word: text to be sanitized
Output	String – text sanitized

Method Name	Check_CSRF()
Class Name	N/A
Functionality	This function verifies the CSRF token of the session and the CSRF token submitted together with the form, both token should be set and match each other.
Input	N/A
Output	A Boolean value indicate if the CSRF tokens are valid (1) or not valid (0)

3.4 Module <Account>

3.4.1 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	Sql_connection	$connection	A variable to hold the database connection

Methods

Visibility	*Name*	*Description*
public	AccountController($connection)	Constructor, initialize the private variable $connection with the $connection passed on from global
public	register_account(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	check_email(email: string) : bool	Check if the email exists in the company email table and the email not registered before
public	check_username(username:string) : bool	Check if the username registered before
public	delete_account(username:string): bool	Removes the account associated with the username from the MSS database

Method Name	AccountController()
Class Name	AccountController
Functionality	This is the constructor for AccountController class. It will initialize the object with a database connection passed on
Input	$connection object – a database connection for MySQL.
Output	N/A

Method Name	register_account
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 string password - the desired password for the newly created account 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	Check_Email()
Class Name	AccountController
Functionality	Check if the email exists in the company email table and the email not registered before
Input	String email: the email to verify
Output	A boolean value indicates if the email can be used (true) or cannot (false).

Method Name	Check_username()
Class Name	AccountController
Functionality	Check if the username registered before
Input	String username: the username to verify
Output	A boolean value indicates if the username can be used (true) or cannot (false).

Method Name	delete_account()
Class Name	AccountController
Functionality	Removes the account associated with the username from the MSS database
Input	String username: the username whos account to be removed from the database
Output	A boolean value indicates if the account deletion is a success (true) or failure (false).

3.4.2 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	

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	Is_admin	Hold the boolean value whether the user is the admin(true) or not (false)

Methods	

3.5 Module <AdminAccount>

3.5.1 Class <AdminAccountController>

Class Name	AdminAccountController
Inherits From	AccountController
Description	Checks and sets administrative privileges for user accounts
Attributes	
Methods	

Visibility	*Name*	*Description*
public	register_admin_account(user name : string, password : string, email : string) : bool	Creates an admin account using the input username, password, and email as the account credentials. Save the account information into the database.
public	Add_email(email:string):bool	Add the email into the company email table record.

Method Name	register_admin_account
Class Name	AdminAccountController
Functionality	Creates an admin account 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 string password - the desired password for the newly created account 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	Add_email()
Class Name	AdminAccountController
Functionality	Add the email into the company email table record.
Input	String email: the email to be add into the database
Output	A boolean value indicates if the email addition is a success (true) or failure (false).

3.6 Module <Login>

3.6.1 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	Sql_connection	$connection	A variable to hold the database connection

Methods

Visibility	*Name*	*Description*
public	LoginController($connection)	Constructor, initialize the private variable $connection with the $connection passed on from global
public	Login(username:string, pwd:string):bool	Verifies whether a user with the specified username/password combo exists in the database.
public	SetLogin(username:string, is_admin:bool):void	Sets the global session variable $_SESSION['username'] & $_SESSION['is_admin'] into the value of username and is_admin
public	checklogin( ) : bool	Retrieves the login status of the current user
public	checkadmin( ) : bool	Retrieves the admin status of the current user
public	logout():void	Logout the user from the session.

Method Name	LoginController()
Class Name	LoginController
Functionality	This is the constructor for LoginController class. It will initialize the object with a database connection passed on
Input	$connection object – a database connection for MySQL.
Output	N/A

Method Name	Login()
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 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	setLogin()
Class Name	LoginController
Functionality	Sets the global session variable $_SESSION[‘username’] & $_SESSION[‘is_admin’] into the value of username and is_admin
Input	String username: the username Boolean is_admin: indicate whether the user is an admin
Output	N/A

Method Name	Check_login()
Class Name	LoginController
Functionality	Retrieves the login status of the current user
Input	N/A
Output	A boolean value indicate the user is logged in (1) and not logged in (0)

Method Name	Check_admin()
Class Name	LoginController
Functionality	Retrieves the admin status of the current user
Input	N/A
Output	A boolean value indicate the user is admin (1) or not admin (0)

Method Name	logout ()
Class Name	LoginController
Functionality	Logout the user from the session.
Input	N/A
Output	N/A

3.7 Module <Profile>

3.7.1 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.
private	String	Username	Hold the current username for the user
private	Sql_connection	$connection	A variable to hold the database connection

Methods

Visibility	Method Name	Description
public	ProfileController(username:string; $connection)	The constructor, initialize an Profile Controller object with the username and $connection variable passed on
public	save_profile():bool	Save the profile object into the database table
public	get_profile():bool	Return the profile object
public	set_name(name:string):void	Set the fullname variable of the profile object
public	set_contact(contact:string):void	Set the contact number variable of the profile object
public	set_job(jobTitle:string):void	Set the jobtitle variable of the profile object
public	set_biography(biography:string):void	Set the variable of the profile object

Method Name	ProfileController()
Class Name	ProfileController
Functionality	This is the constructor for ProfileController class. It will initialize the object with the username and the database connection passed on
Input	String username – username get from the global session $_SESSION['username'] variable $connection object – a database connection for MySQL.
Output	N/A

Method Name	save_profile()
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	get_profile()
Class Name	ProfileController
Functionality	return the profile object of the profile controller.
Input	N/A
Output	Profile object

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

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

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

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

3.7.2 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	

3.8 Module <Meeting>

3.8.1 Class <MeetingController>

Class Name	MeetingController
Inherits From	N/A
Description	Create meetings and perform associated functions such as add attendee, remove attendee, getMeeting, and deleteMeeting.
Attributes	

Visibility	*Data Type*	*Name*	*Description*
private	mysqli	connection	An object to hold the current database connection

Methods

Visibility	Method Name	Description
public	createMeeting( ):void	Adds a meeting to the database using the information posted from a form
public	addAttendee(username: string, meeting_id:int):void	Add attendee to the meeting using the meeting ID given
public	removeAttendee(username:string, meeting_id:int):void	Remove attendee from the meeting using the meeting ID given
public	getMeeting(date:string, time:string):void	Generates an HTML button in a given time slot when a meeting exists at the input date and time
public	getMeetingDetails(meet_time: datetime) : void	Generates an HTML section for each meeting that takes place at the specified date and time
public	getMeetingWithRoom(date:string, time:string, room:string):array	Returns a list of meetings that take place at the specified time and in the specified room
public	retrieveRooms( ) : array	Returns all rooms currently stored in the database

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 with the specified ID
Input	Attendee username, Meeting ID
Output	N/A

Method Name	removeAttendee()
Class Name	MeetingController
Functionality	Remove attendee from the meeting with the specified ID
Input	Attendee username, Meeting ID
Output	N/A

Method Name	getMeeting()
Class Name	MeetingController
Functionality	Generates an HTML button in a given time slot when a meeting exists at the input date and time
Input	A date (string) and a starting time (string)
Output	An HTML button that links to the associated meeting details page

Method Name	getMeetingDetails()
Class Name	MeetingController
Functionality	Generates an HTML section for each meeting that takes place at the specified date and time
Input	A datetime object representing the starting time
Output	An HTML section detailing the meeting's title, ID, start time, end time, organizer, and location

Method Name	getMeetingWithRoom()
Class Name	MeetingController
Functionality	Returns a list of meetings that take place at the specified time and in the specified room
Input	A date (string), a starting time (string), and the associated room ID (string)
Output	The Meeting rows retrieved from the database associated with the meeting taking place at the specified time and day in the specified room

Method Name	retrieveRooms()
Class Name	MeetingController
Functionality	Returns all rooms currently stored in the database
Input	N/A
Output	An array of rows containing the desired room data

3.8.2 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	

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

Methods	

3.9 Module < Room>

3.9.1 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	RoomController($connection)	A constructor
public	add_room(room_object : Room_Object):bool	add the room object to the database
public	getRooms( ): string[]	Get the roomIDs
public	checkIsSpecial( ) : bool	check if the room is special
public	view_room( ): string[]	Get the rooms' details
public	deleteRoom (room_id:String):bool	Delete the room

Method Name	RoomController()
Class Name	RoomController
Functionality	This is the constructor for RoomController class. It will initialize the object with the database connection passed on
Input	$connection object – a database connection for MySQL.
Output	N/A

Method Name	add_room ()
Class Name	RoomController
Functionality	save the room object information into the database
Input	RoomObject: room_object - with value passed from the POST method
Output	A boolean value indicating if the save is a success(true) or fail(false)

Method Name	getRooms()
Class Name	RoomController
Functionality	Get the roomID
Input	N/A
Output	Return a list of available room_id, output as option for selection.

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	view_room()
Class Name	RoomController
Functionality	Get full list of the rooms with details
Input	N/A
Output	Return a list of available rooms with details to be displayed in a table view

Method Name	deleteRoom ()
Class Name	RoomController
Functionality	Delete the room
Input	String: room_id
Output	A boolean value indicating if the delete is a success(true) or fail(false)

3.9.2 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	

Visibility	Data Type	Name	Description
Private	string	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

Methods	

3.10 Module <Payment>

3.10.1 Class <PaymentController>

Class Name	PaymentController
Inherits From	N/A
Description	This class controls all the operations that an administrator does to update a client's billing information and processing the payment
Attributes	

Visibility	*Data Type*	*Name*	*Description*
private	PaymentInfo	CreditCard	PaymentInfo object contains the user credit card information
private	String	Username	Hold the current username for the user
private	Sql_connection	$connection	A variable to hold the database connection
private	String	key	Hold the encryption key

Methods

Visibility	*Name*	*Description*
public	PaymentController(username: string; $connection)	The constructor, initialize the Payment Controller object with the username and $connection variable passed on
public	save_card(new_card:PaymentInfo):bool	Add the card with the information supplied to the database
public	get_card():PaymentInfo	Retrieve the card from the database for the user
public	delete_card(card_id:int):bool	Delete the selected card from the database
public	set_id(card_id:int):void	Set the card_id of the credit_card object
public	set_owner(cardowner:string):void	Set the card_owner of the credit_card object
public	set_ccnumber(cardnumber:int):void	Set the cardnumber of the credit_card object
public	set_cvv (cvv:int):void	Set the ccv of the credit_card object

	public	set_expdate(expdate:string):void	Set the expiry date of the credit_card object
	public	encrypt_number(number:int):string	Encrypt the number using the open_ssl encryption
	public	decrypt_number(encrypted_number:string):int	Decrypt the number using the open_ssl decryption
	public	makePayment(card:PaymentInfo, charge:double):bool	Use the selected credit card to make payment

Method Name	PaymentController ()
Class Name	PaymentController
Functionality	This is the constructor for PaymentController class. It will initialize the object with the username and the database connection passed on
Input	String username – username get from the global session $_SESSION['username'] variable $connection object – a database connection for MySQL.
Output	N/A

Method Name	Save_card()
Class Name	PaymentController
Functionality	Add the card with the information supplied to the database
Input	PaymentInfo object – new_card: A PaymentInfo object passed on from the view page.
Output	A boolean value indicating if the addition is a success(true) or fail(false)

Method Name	get_card()
Class Name	PaymentController
Functionality	Retrieve the card for the user from the database
Input	N/A
Output	A PaymentInfo object

Method Name	delete_card()
Class Name	PaymentController
Functionality	Remove the selected credit card from the database
Input	Int-card_id: Card_id of the card to be deleted
Output	A Boolean value indicating if the deletion is a success(true) or fail(false)

Method Name	set_id()
Class Name	PaymentController
Functionality	Set the card_id of the credit card object
Input	Int-card_id: the card_id from the database
Output	N/A

Method Name	set_owner()
Class Name	PaymentController
Functionality	Edit the card owner name
Input	String cardowner: card owner name
Output	N/A

Method Name	set_ccnumber()
Class Name	PaymentController
Functionality	Edit the card number
Input	Int – cardnumber: credit card number
Output	N/A

Method Name	set_cvv()
Class Name	PaymentController
Functionality	Edit the CVV number
Input	Int - cvv: cvv number
Output	N/A

Method Name	set_expdate()
Class Name	PaymentController
Functionality	Edit the expiry date
Input	String-exp_date: expiry date for the card
Output	N/A

Method Name	encrypt_number()
Class Name	PaymentController
Functionality	Encrypt the number using the open_ssl encryption, using the key variable stored in the PaymentController class
Input	Int-number: A number (either the credit card number or ccv) to be encrypted
Output	String-encrypted data

Method Name	decrypt_number()
Class Name	PaymentController
Functionality	decrypt the number using the open_ssl decryption, using the key variable stored in the PaymentController class
Input	String-encrypted data
Output	Int-number: A number (either the credit card number or ccv) decrypted

Method Name	makePayment(card:PaymentInfo, charge:double):bool
Class Name	PaymentController
Functionality	Use the selected credit card to make payment
Input	PaymentInfo card: PaymentInfo object of the selected card 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)

3.10.2 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*
Public	int	card_id	Hold the card id in the database
Public	string	cardowner	Name of the card owner
Public	int	cardnumber	Credit card number
Public	int	cvv	The CVV of the credit card in the database
Public	Date	exp_date	The expiry date of the credit card

3.11 Module < Complaint>

3.11.1 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.
private	Sql_connection	$connection	A variable to hold the database connection

Methods

Visibility	*Name*	*Description*
public	ComplaintController($connection)	The constructor, initialize the ComplaintController object with the $connection variable passed on
public	save_complaint(complaint):bool	Save the complaint with the supplied information into the database.
public	view_complaint():Complaint[]	Retrieve the list of complaints from the database and display it
public	get_complaint_id():String[]	Retrieve the list of complain_ids
public	resolveComplaint(complaint_id:string):void	Allow the administrator to resolve the complaint.

Method Name	ComplaintController ()
Class Name	ComplaintController
Functionality	This is the constructor for the ComplaintController class. It will initialize the object with the database connection passed on
Input	$connection object – a database connection for MySQL.
Output	N/A

Method Name	save_complaint()
Class Name	ComplaintController
Functionality	Save the complaint with the supplied information into the database.
Input	Complaint : complaint - A Complaint object storing the complaint information
Output	A boolean value indicating if the saving is a success(true) or fail(false)

Method Name	view_complaint()
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[] display into a table

Method Name	get_complaint_id()
Class Name	ComplaintController
Functionality	Returns the list of complaint's id for the administrator to select
Input	N/A
Output	An array list of Complaint's id displayed as the option of a select dropdown menu

Method Name	resolveComplaint()
Class Name	ComplaintController
Functionality	Allow the administrator to resolve the complaint
Input	String: complaint_id - the complaint to resolve
Output	N/A

3.11.2 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)

3.12 ER Diagram for the Database

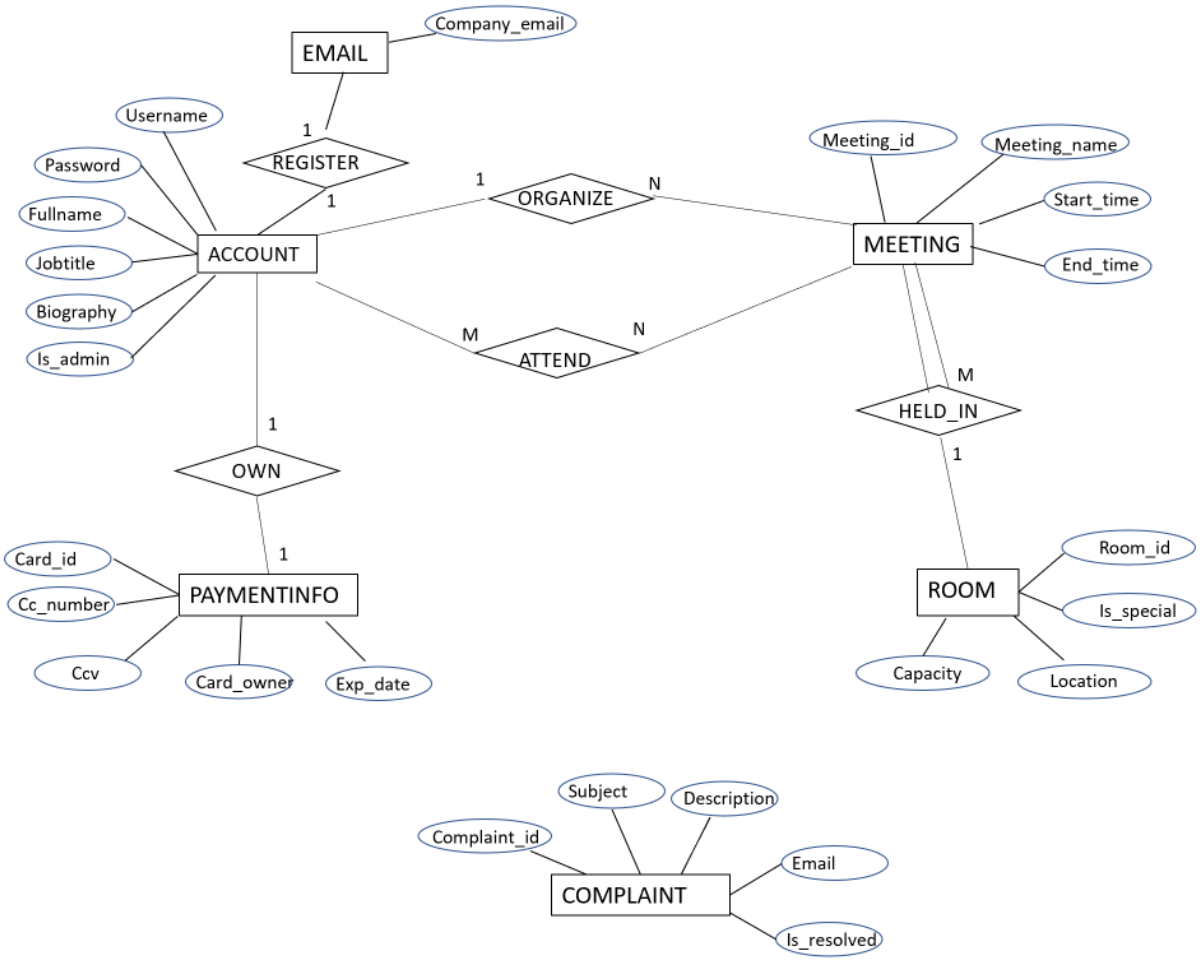


Figure 2: ER Diagram

Figure 2 shows the Entity Relation (ER) diagram for the back-end MYSQL database to be used in the software application. There are a total of 6 entities identified, with additional one relationship that need to be recorded in a separate table. The resulting database schema is shown in section 3.13.

3.13 Database Schema

Table EMAIL

Company_email

Table ACCOUNT

Username	Password	Fullname	Contact	Jobtitle	Biography	Is_admin	Company_email

Table MEETING

Meeting_id	Meeting_name	Start_time	End_time	Organizer(Username)	Room_Id

Table ATTENDANCE

Meeting_id	Attendee

Table PAYMENTINFO

Card_id	Cc_number	Cvv	Card_owner	Exp_date	Username

Table ROOM

Room_id	Is_special	Capacity	Location

Table COMPLAINT

Complaint_ID	Subject	Description	Email	Is_resolved

Figure 3: The database schema

Figure 3 shows the database schema used to build the database for the software application. The first table EMAIL is needed to store all valid company email address to be used by the users to create their login account. The second table ACCOUNT stores the account information such as username, password, Is_admin and company email as well as the user profile information. The next table MEETING captures the key meeting information such as meeting_id, meeting_name, start and end time, as well as the organize name and room id. The attendance list of the meeting will be store in the separate table ATTENDANCE to capture all meeting_id & attendee pairs information.

The table PAYMENTINFO will store the credit card information such as number, cvv, card owner and expiry date. The card_id is required to identify the credit card within the software system. The next table ROOM will store the information for the room such as its room_id, whether the room is_special, the capacity and location of the room. The last table COMPLAINT will store the complaints information such as the complaint_id, subject, description, email and whether the complaint has been resolved.

4. Testing Results

4.1 Overview of Testing Approaches

The testing activities can be divided into two group, the first group focuses on testing the software application against the key functional requirements outlined in the SRS document section 2. The second group of testing activities focuses on testing the software application against the potential threats and vulnerabilities identified in the Architectural Risk Analysis section of the SDS document.

For the software security testing activities, the main method used is the functional security testing which translate the risk mitigations requirement into positive functional requirements. This is supplemented by the risk-based testing activities which outline the negative testing requirements. Table 1 lists the security features and testing requirements used to mitigate the risk identified in the Architectural risk analysis section.

Security Features	Testing Requirements	Protection against risk/vulnerabilities
Private Folder	**Negative testing requirement:** The system shall not allow the attackers to gain access to the system illegally by looking through the public files **Positive testing requirement:** The system shall hide all the controller files in a private folder, which is protected against index.php query.	Threat: Unauthorized system access by Cracker/ Industry espionage / competing company
Input validation	**Negative testing requirement:** The system shall not allow text input not in accepted format to be passed through. **Positive testing requirement:** The system validates that the text input is of accepted format, and only allows text input in accepted whitelist to pass through.	Threat: Computer Crime/Information theft by Industry espionage / competing company. ● CWE-94: Improper Control of Generation of Code ('Code Injection') ● CAPEC-242: Code Injection
Strong password requirement	**Negative testing requirement:** The system shall not allow the attackers to gain access to the system illegally through username and password cracking. **Positive testing requirement:** The system shall validate that the password used for registration is strong enough and consists of required characters	Threat: Unauthorized system access by Cracker/ Industry espionage / competing company ● CWE-521 Weak Password Requirements ● CAPEC-49: Password Brute Forcing ● CAPEC-70: Try Common or Default Usernames and Passwords
Input verification	**Negative testing requirement:** The system shall not allow the attackers to gain access to the system illegally. **Positive testing requirement:** The system shall validate that the company email used for	Threat: Unauthorized system access by Cracker / Industry espionage / competing company ● CWE-308 Use of Single Factor Authentication

	registration exists and the company email and username not registered before	
Input sanitization	**Negative testing requirement:** The system shall not allow the attackers to perform SQL injection **Positive testing requirement:** The system shall sanitize the input text to process the quotation properly.	Threat: Computer Crime/Information theft by Industry espionage / competing company. ● CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') ● CAPEC-7: Blind SQL Injection ● CAPEC-66: SQL Injection ● CAPEC-108: Command Line Execution through SQL Injection
Html escaping	**Negative testing requirement:** The system shall not allow the attackers to perform code injection using JavaScript. **Positive testing requirement:** The system shall process the text from database to be displayed on the website into html text.	Threat: Computer Crime/Information theft by Industry espionage / competing company. ● CWE-94: Improper Control of Generation of Code ('Code Injection') ● CAPEC-242: Code Injection
CSRF Token	**Negative testing requirement:** The system shall prevent against Cross Site Request Forgery (CSRF) attack. **Positive testing requirement:** The system shall use a CSRF session token to verify against form input token to ensure the form action came from its own sources.	Threat: Computer Crime/Information theft by Industry espionage / competing company. ● CWE-352: Cross-Site Request Forgery (CSRF) ● CAPEC-62: Cross Site Request Forgery
Session variable	**Negative testing requirement:** The system shall prevent unauthorized access from the attackers. **Positive testing requirement:** The system shall use the php $_SESSION variables to store the login status, and verify these session variables every time a new page is load.	Threat: Unauthorized system access by Cracker / Industry espionage / competing company ● CWE-308 Use of Single Factor Authentication
Session timeout	**Negative testing requirement:** The system shall prevent unauthorized access from the attackers through the accidentally left-open session page by the user. **Positive testing requirement:** The system shall automatically log out the users from the system after a fixed duration.	Threat: Unauthorized system access by Cracker / Industry espionage / competing company ● CWE-308 Use of Single Factor Authentication
Display masking	**Negative testing requirement:** The system shall prevent the attackers from gaining sensitive	Threat: Browsing of credit card information

	information through recording of web browser activities. **Positive testing requirement:** The system shall mask part of the sensitive information (such as credit card number) displayed.	
Data encryption	**Negative testing requirement:** The system shall prevent the attackers from gaining sensitive information when they gained access into the database. **Positive testing requirement:** The system shall encrypt sensitive information stored in the database.	Threat: Browsing of credit card information • CWE-311: Missing Encryption of sensitive data
Vague Error message	**Negative testing requirement:** The system shall prevent the unauthorized access from attacker through trial combination of email and password. **Positive testing requirement:** The system shall give vague error message to prevent revealing too much details of the system.	Threat: Unauthorized system access, computer crime.
Disabled Error message	**Negative testing requirement:** The system shall prevent the unauthorized access from attacker through trial combination of email and password. **Positive testing requirement:** The system shall disabled error message at the server side to prevent revealing too much details of the system.	Threat: Unauthorized system access, computer crime.

4.2 Functional Feature Testing

4.2.1 Registration (Normal Client)

Figure 4 shows the company email table containing valid email that can be used for registration. After supplying the valid email address, username and password as per figure 5, the new user account is registered in the database as shown in figure 6. Note that this registration method can only create the normal client account.

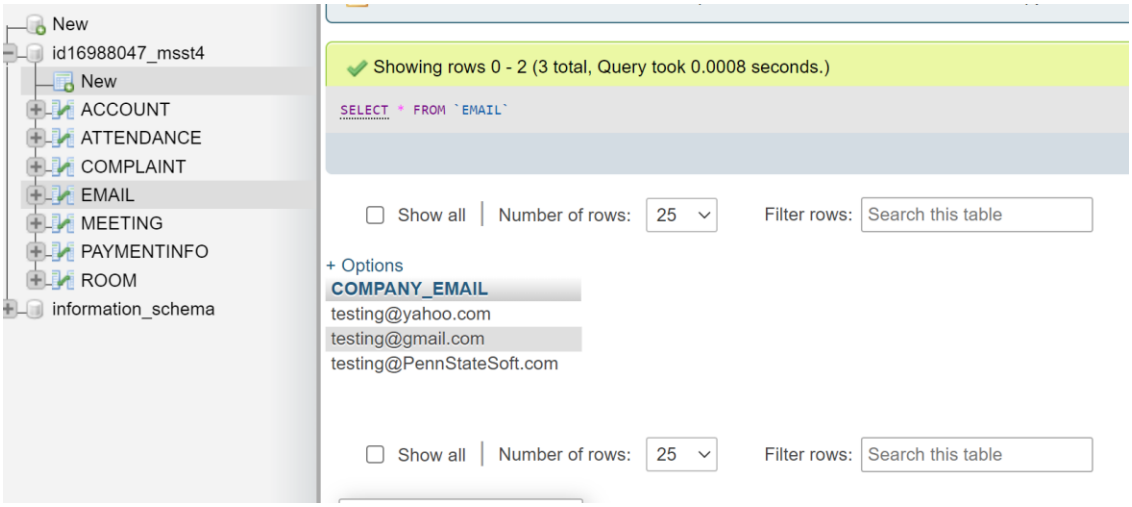


Figure 4: Company Email Database

Figure 5: Registration Page with testing details

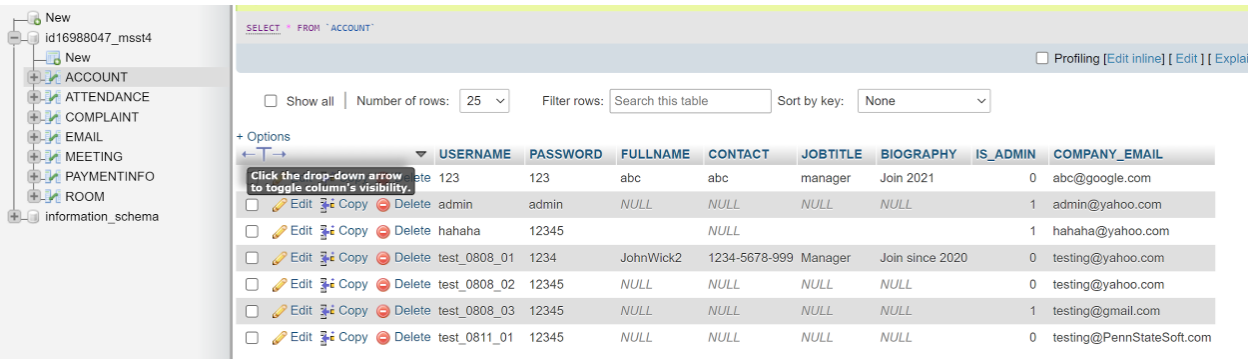


Figure 6: Account Database shown the registered details

4.2.2 Login

Figure 7 shows the login details after registration in section 4.2.1, figure 8 shows the success message after successful login. Note the UI page will be further improved in the final product.

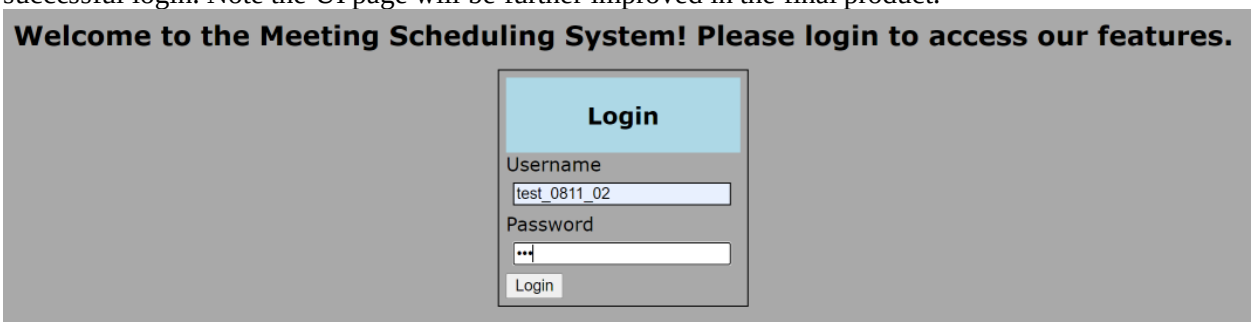


Figure 7: Login Details

Figure 8: Confirmation of successful login.

4.2.3 Profile Management

Figure 9 shows the General Profile UI page, the user can update their profile information which includes the full name, contact number, job title and biography here. After successfully saving the changes, the new information will be displayed immediately on the top left corner as shown in figure 10.

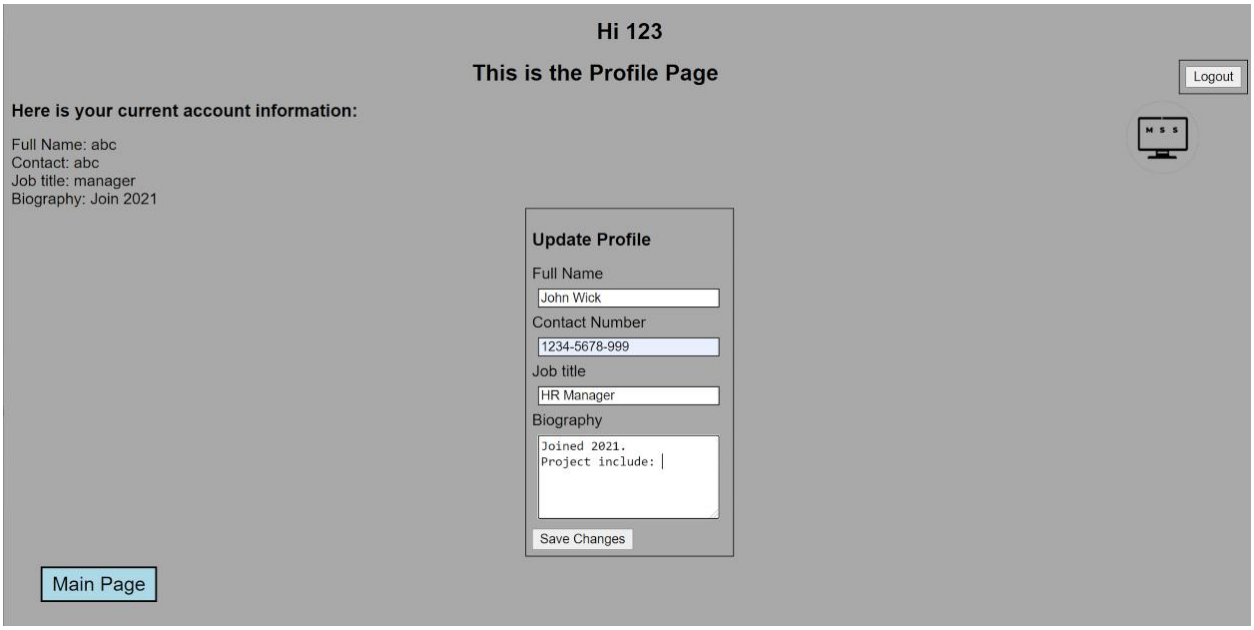


Figure 9: General Profile Page with Profile Change Details

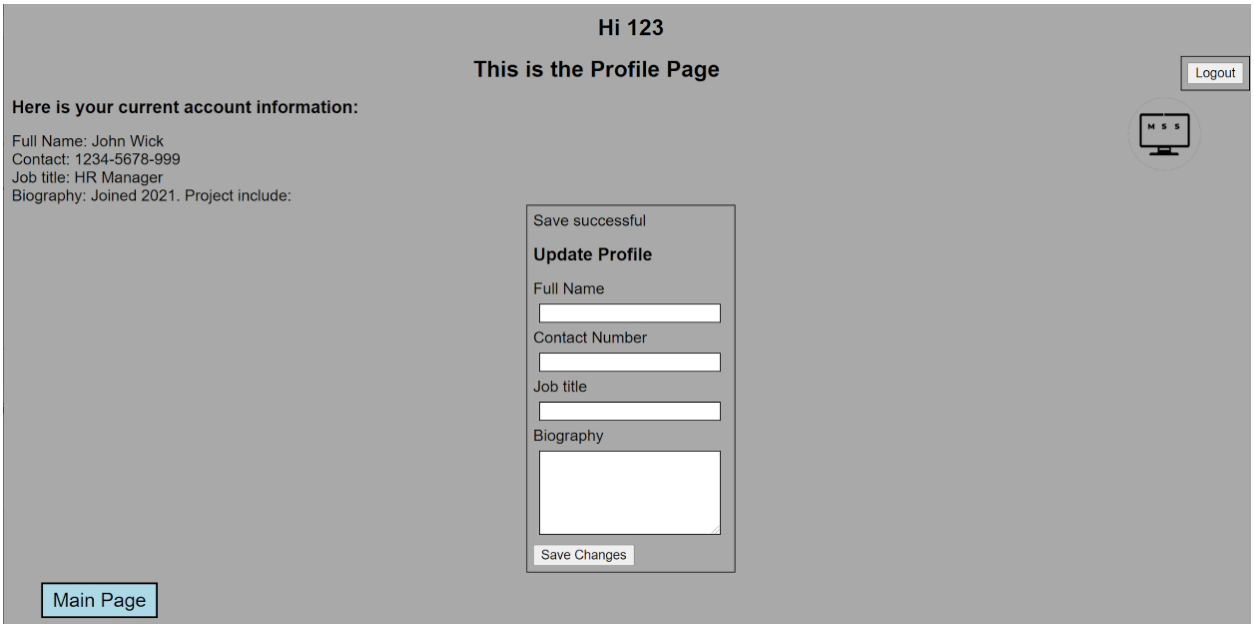


Figure 10: Profile Page After the changes have been updated.

4.2.4 Payment Info Management

Figure 11 shows the manage payment UI interface. The user can display the credit card information they have in the database, add a new card, or delete the existing card. Figure 12 shows how the credit card information looks like at the back-end MySQL database.

4.2.4.1 Add New Card

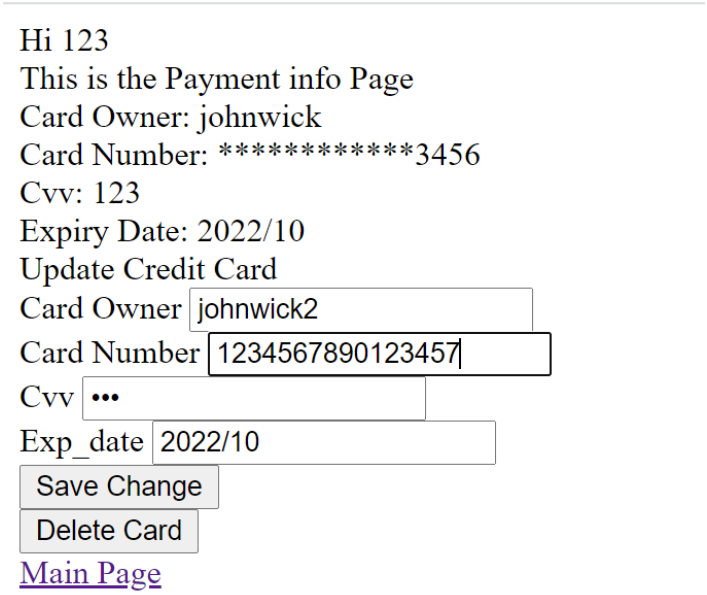


Figure 11The manage payment UI interface

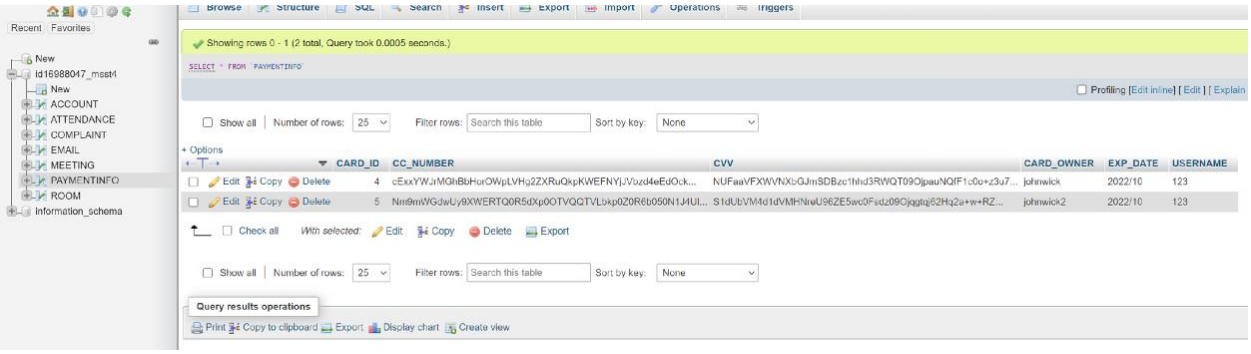


Figure 12: The PAYMENTINFO table to store the credit card information

4.2.5 Meeting Creation

The following figure 13 to 15 is a demonstration of how a general user would create a new meeting. The system will dynamically list all available rooms in the room selection box. Upon creation, a unique meeting ID number is automatically generated by the system, and the current user is saved as the organizer of the meeting.

4.2.5.1 Create a New Meeting

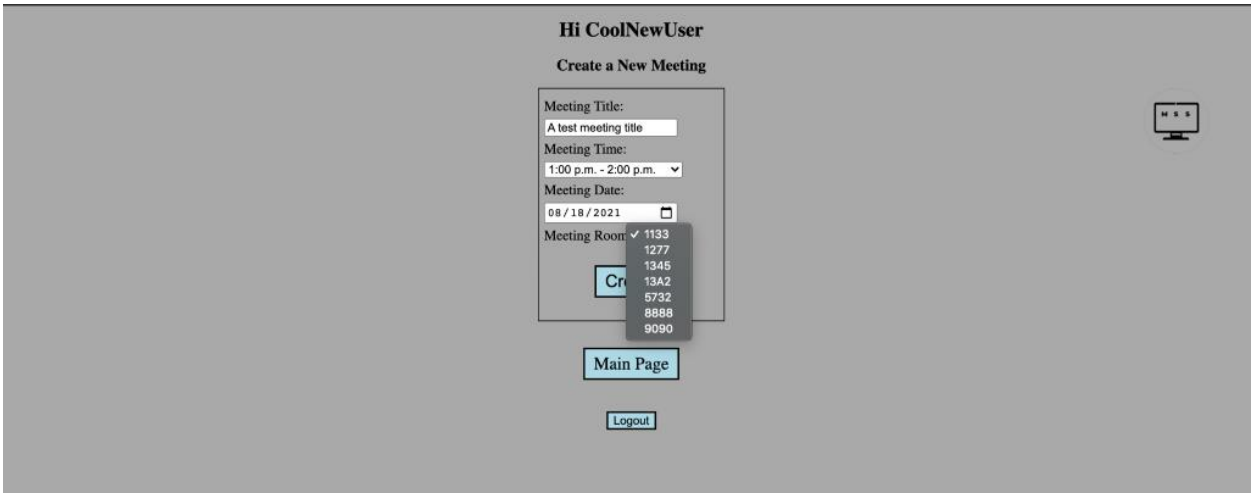


Figure 13: Selecting the room id for the meeting

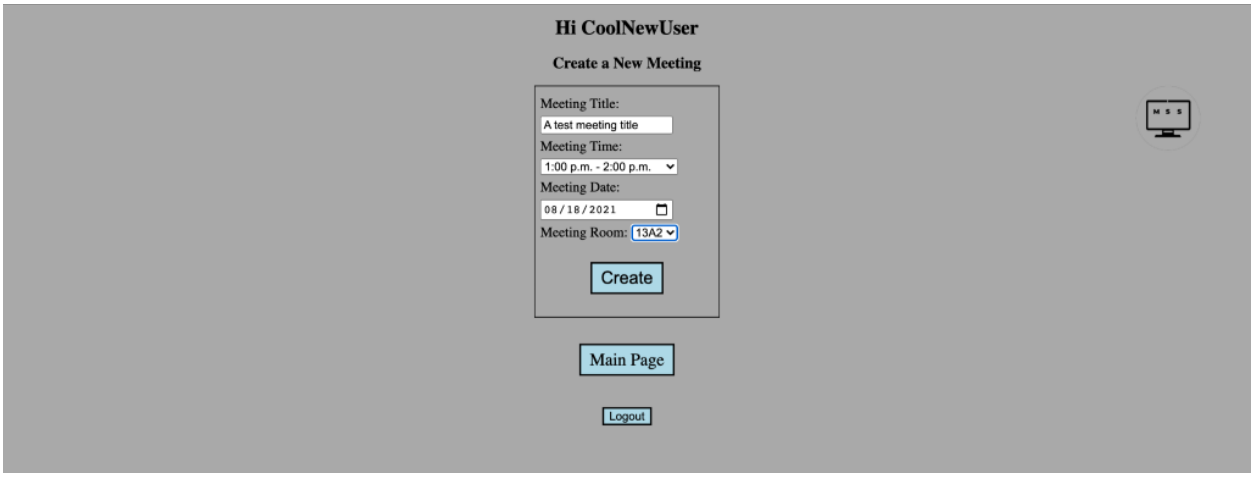


Figure 14: Clicking the create meeting button

☐	Edit	Copy	Delete	182	please	2021-08-17 11:00:00	2021-08-17 12:00:00	CoolNewUser	13A2
☐	Edit	Copy	Delete	183	Cool rad new meeting	2021-08-26 13:00:00	2021-08-26 14:00:00	CoolNewUser	13A2
☐	Edit	Copy	Delete	184	A test meeting title	2021-08-18 13:00:00	2021-08-18 14:00:00	CoolNewUser	13A2

Figure 15: Meeting information stored in database.

4.2.6 View Meetings

The View Meetings page is formatted as a weekly calendar as shown in figure 16. It displays 5 columns representing Monday through Friday and 9 rows, each representing an hour period from 8 am to 9 pm. When a meeting is scheduled for a timeslot within the current week, a button will be displayed that can be clicked to show the details for every meeting scheduled in that time slot. For a general user, only meetings for which they are an attendee (including cases in which they created the meeting) are shown. For an administrator user, all meetings scheduled by anyone with any attendees is shown in figure 17.

4.2.6.1 View as a General User

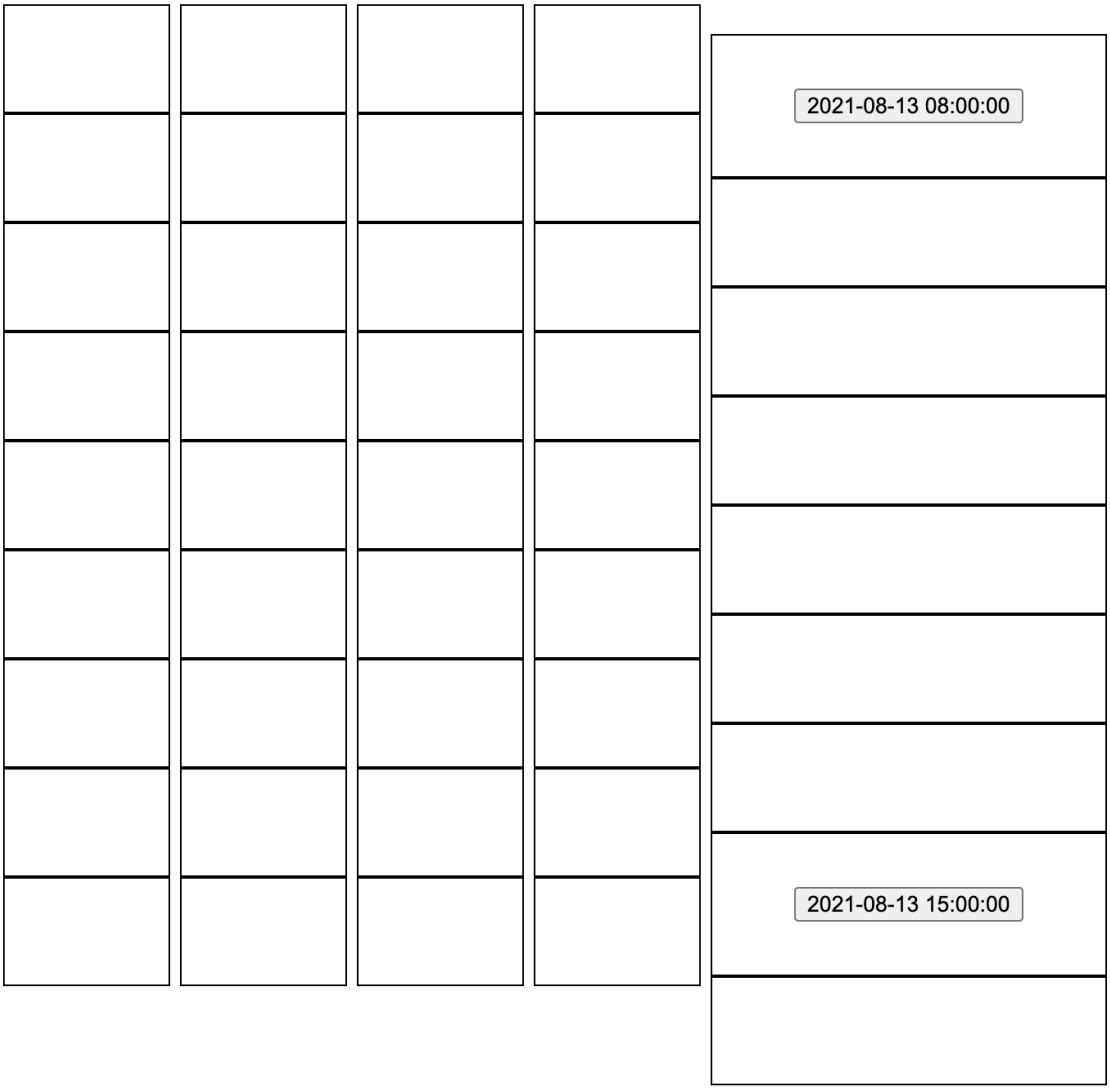


Figure 16: Draft meeting view for general user

4.2.6.2 View as an Admin User

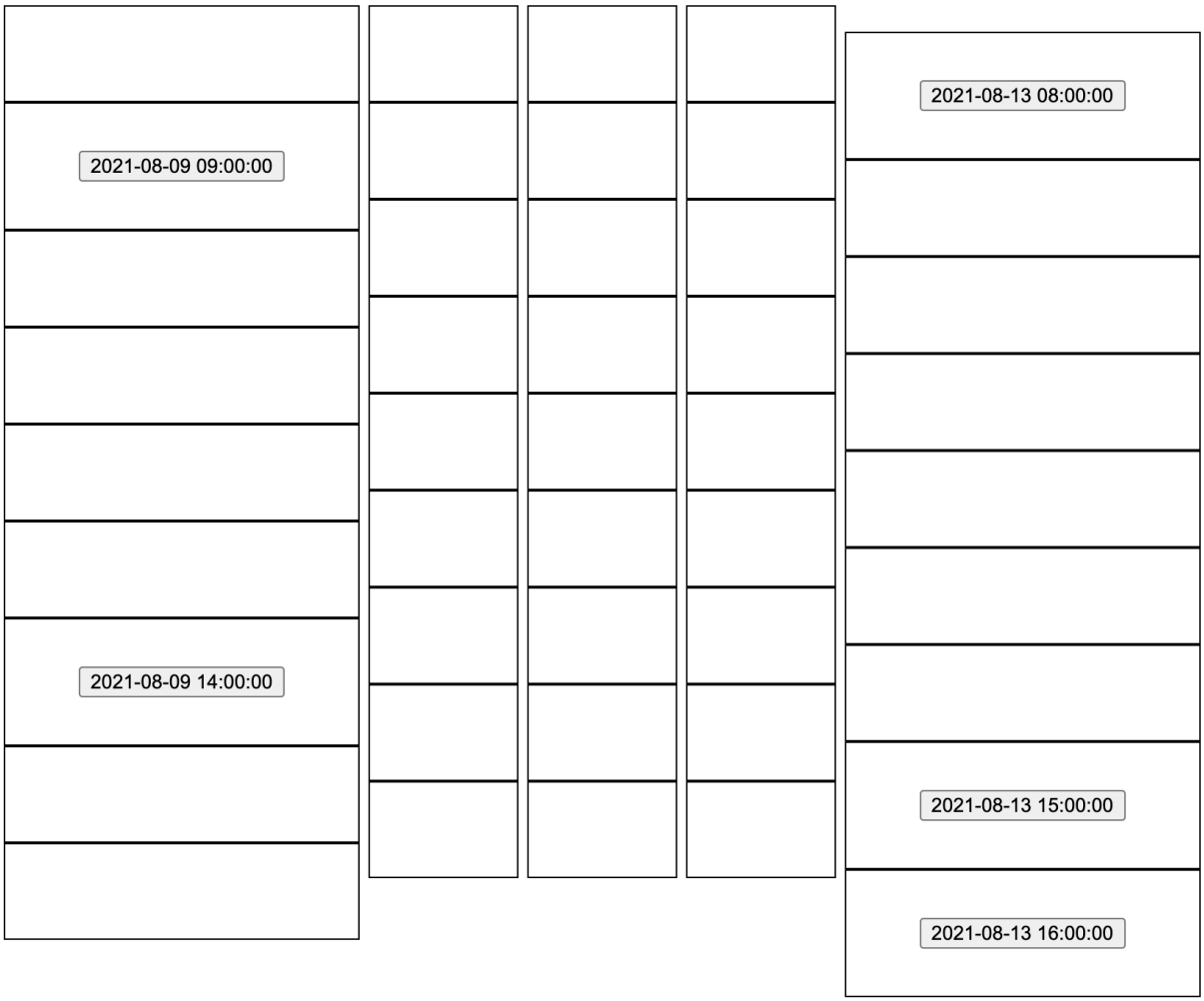


Figure 17: Draft meeting view for admin

4.2.7 View Meeting Details

Upon clicking a button in one of the time slots shown in the above figures in section 4.2.6, the user is redirected to the View Meeting Details page, where they can see the details for each meeting scheduled within that time slot. For a general user, they can see any meeting in that time slot that they have either created or that they are added to as per figure 18. An admin user, however, can see the details for every meeting scheduled in that time slot without having to be added to those meetings as per figure 19.

4.2.7.1 View Meeting Details as a General User

Figure 18: Draft detail meeting view for general user

4.2.7.2 View Meeting Details as an Admin User

Figure 19: Draft detail meeting view for admin

4.2.8 Add/Remove User From a Meeting

The Edit Participants page allows a general user to add or remove another user to or from a meeting by entering the username of the desired participant and the ID of the meeting that they are to be added to or removed from (Note: the Remove feature is currently not implemented).

4.2.8.1 Adding a User to a Meeting

Figure 20-22 showing the add user feature demonstration.

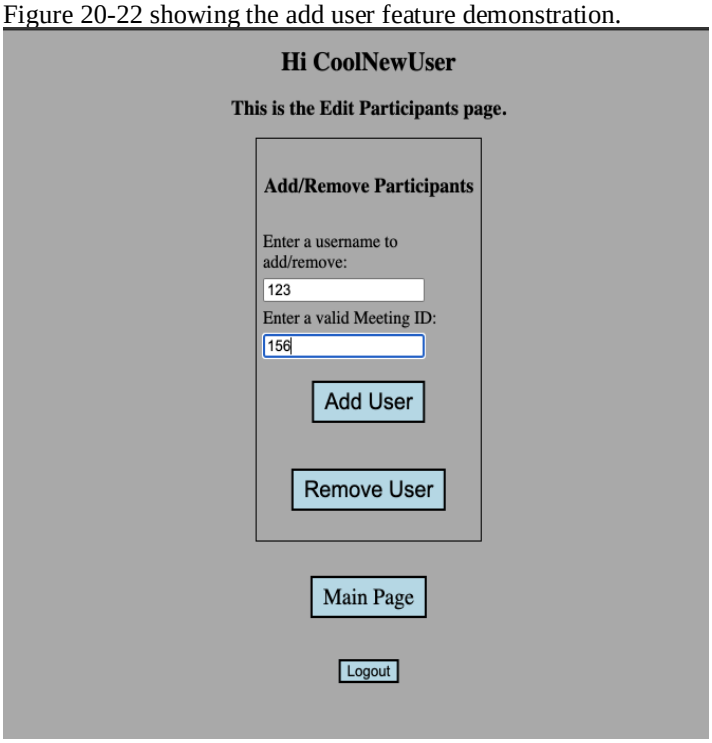


Figure 20: Adding user to the meeting

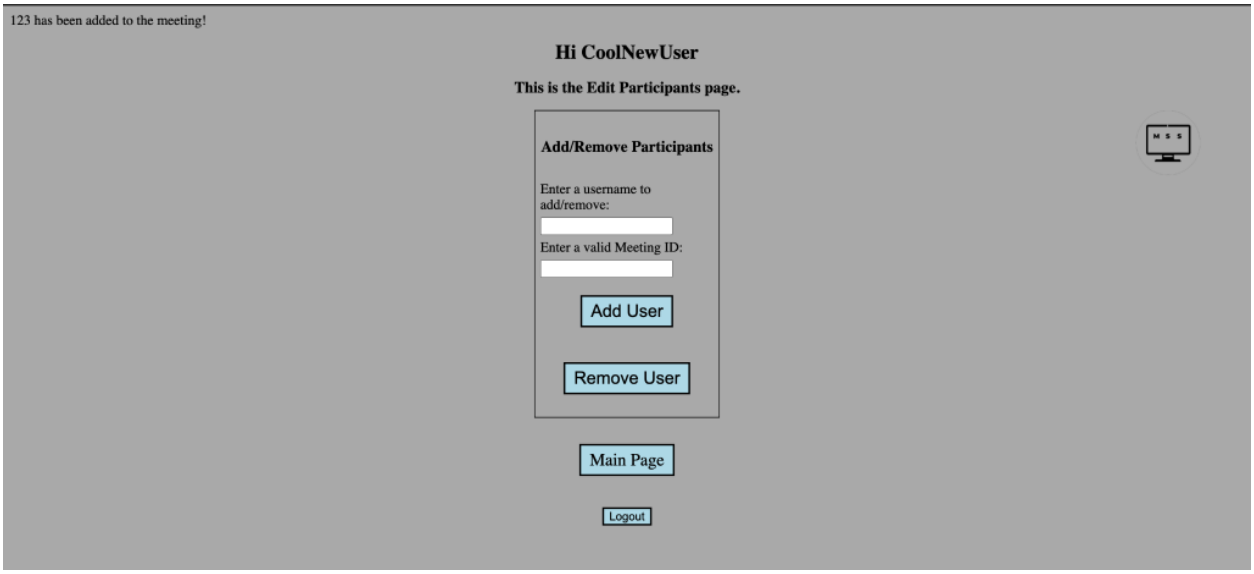


Figure 21: Messages showing the addition is successful

+ Options

MEETING_ID	ATTENDEE
172	CoolNewUser
137	CoolNewUser
175	CoolNewUser
172	CoolNewUser
132	CoolNewUser
179	CoolNewUser
180	CoolNewUser
181	CoolNewUser
183	CoolNewUser
174	CoolNewUser
184	CoolNewUser
156	123

Figure 22: Meeting ID attended by selected users

4.2.9 Complaint Management

Figure 23 shows the functional feature for the complaint management page. For the normal client, they can create a new complaint and save it to database as shown in figure 24. The admin can then view all available complaint as per figure 25, and click to resolve it. Note the complaint resolving feature is currently incomplete and pending further development.

Figure 23: Create complaint view page

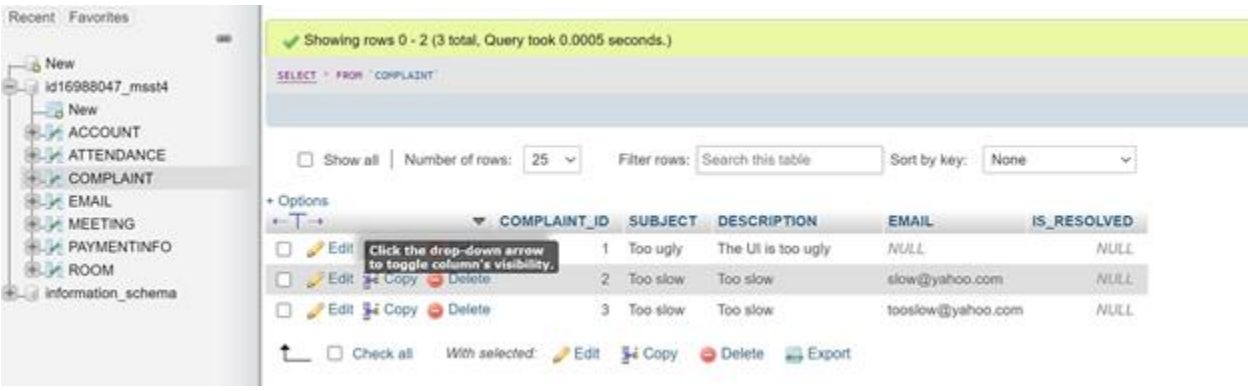


Figure 24: Database showing the complaint is created successfully.

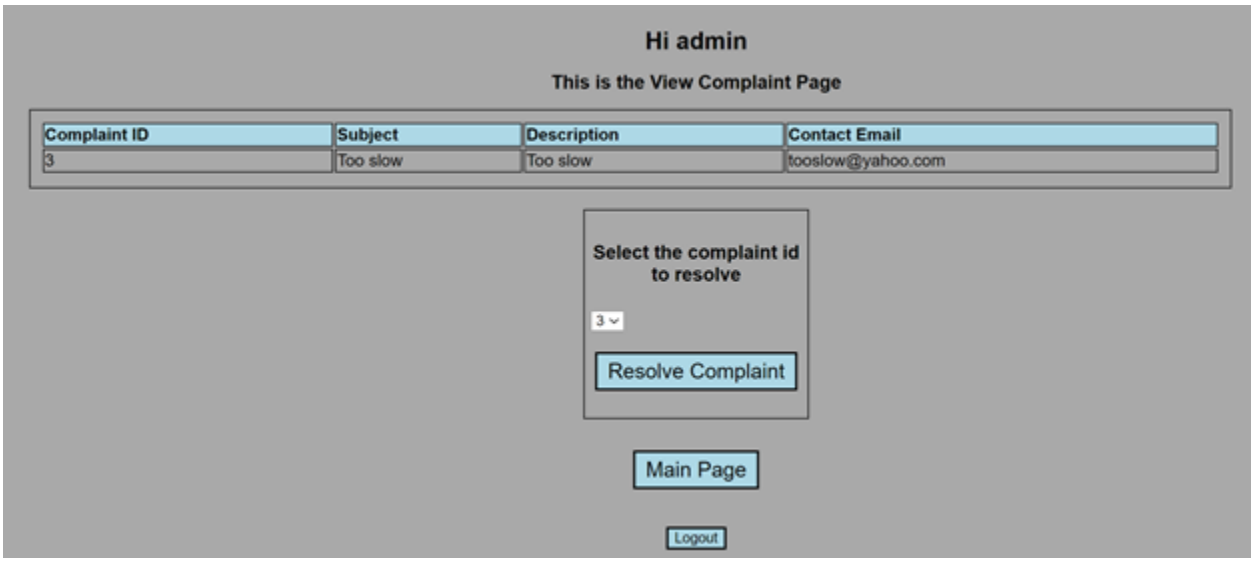


Figure 25: View Complaint page available to admin

4.2.9 Room Management

Figure 26 shows how the room management page will look like for the Administrator. The Administrator can view all the room available, create a new meeting room or delete existing meeting room.

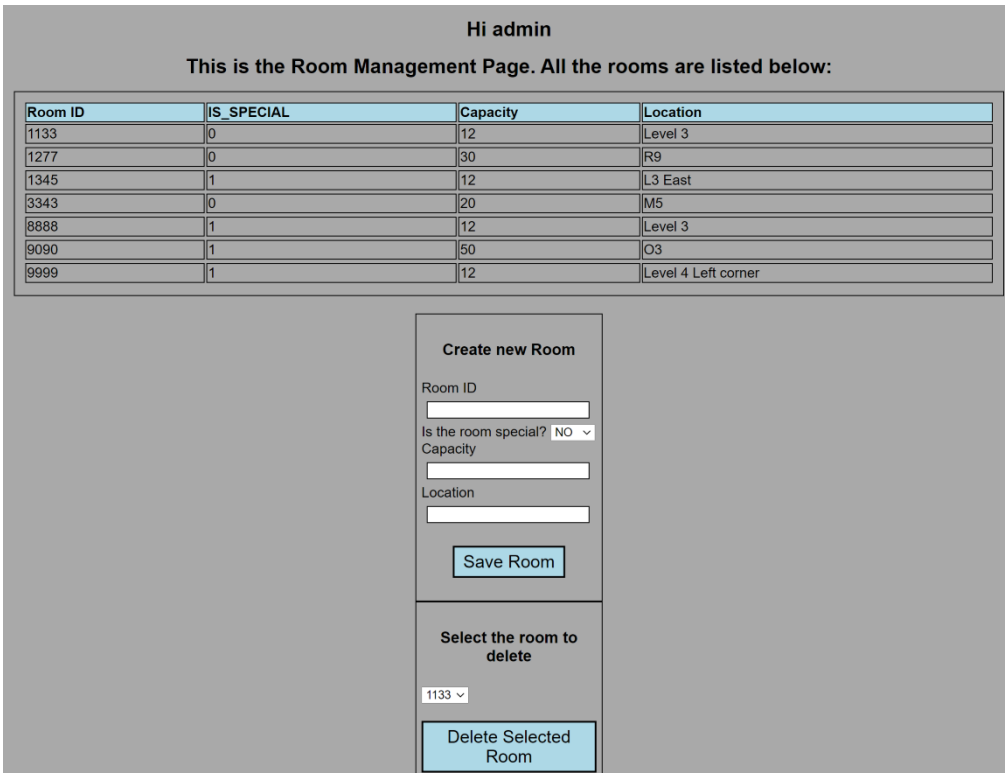


Figure 26: View meeting page for admin

4.3 Security Feature Testing

4.3.1 Folder structure

Figure 27 shows the folder structure for the project files. All the controller files are located within the private folder inaccessible to the public. Figure 28 shows what will be displayed on the browser if the attacker attempts to view the index.php file for the private folder.

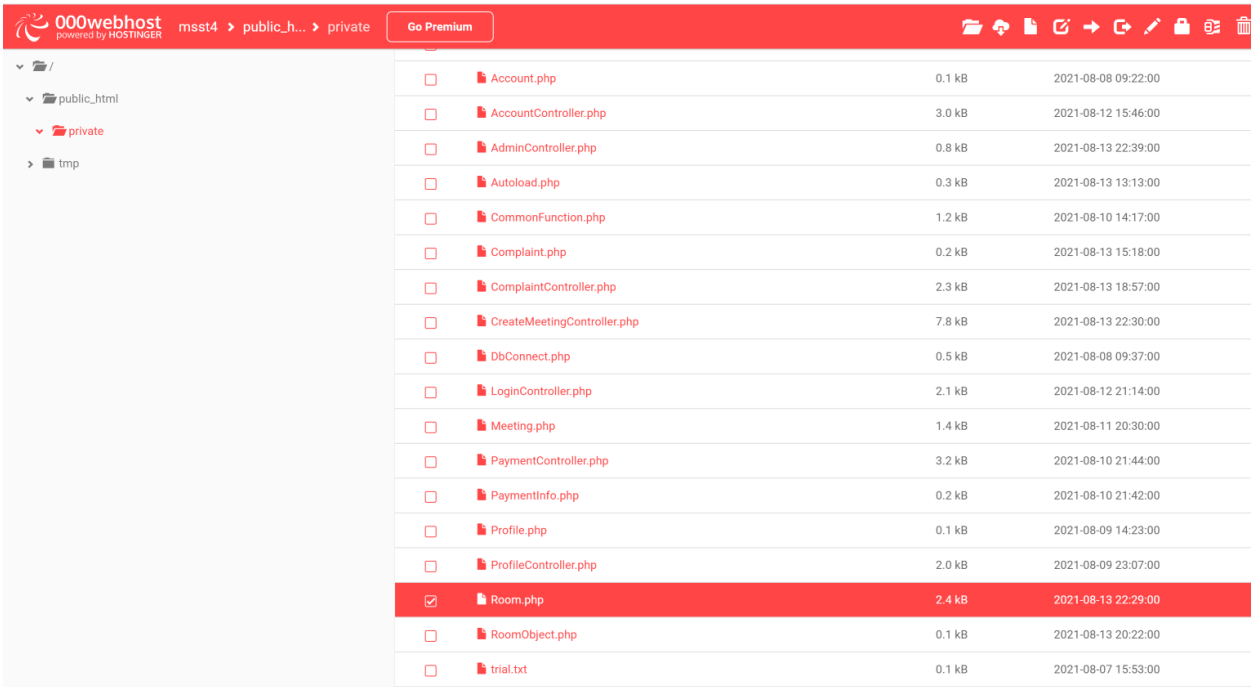


Figure 27: Folder structure

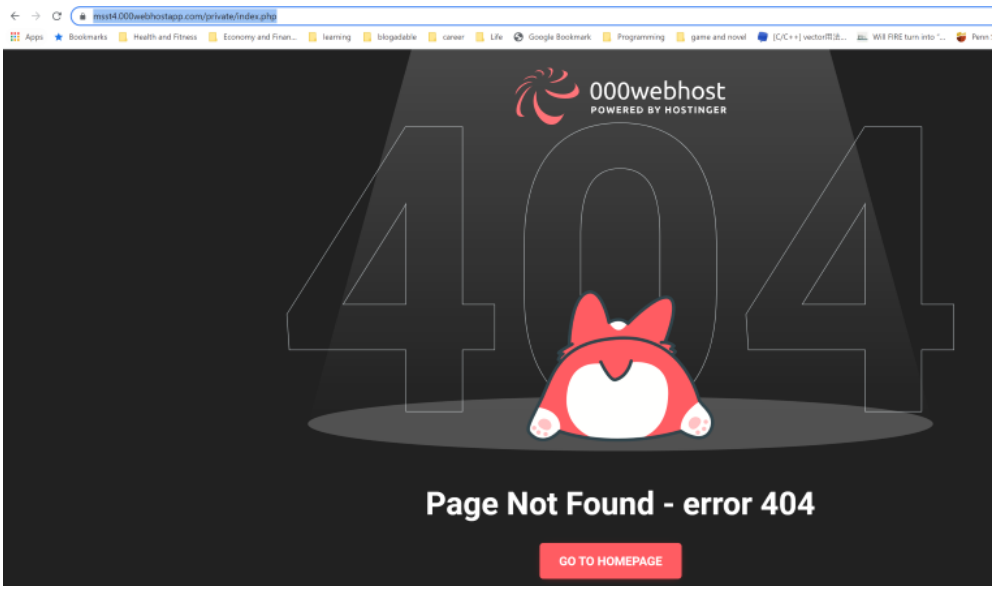


Figure 28: Error when attempting to view index.php file in private folder

4.3.2 Input Validation

Figure 29 shows the implementation of input validation for the email address used for registration. In this case the email address entered must comply to specific whitelisted format (containing @ and '.' characters in the correct order).

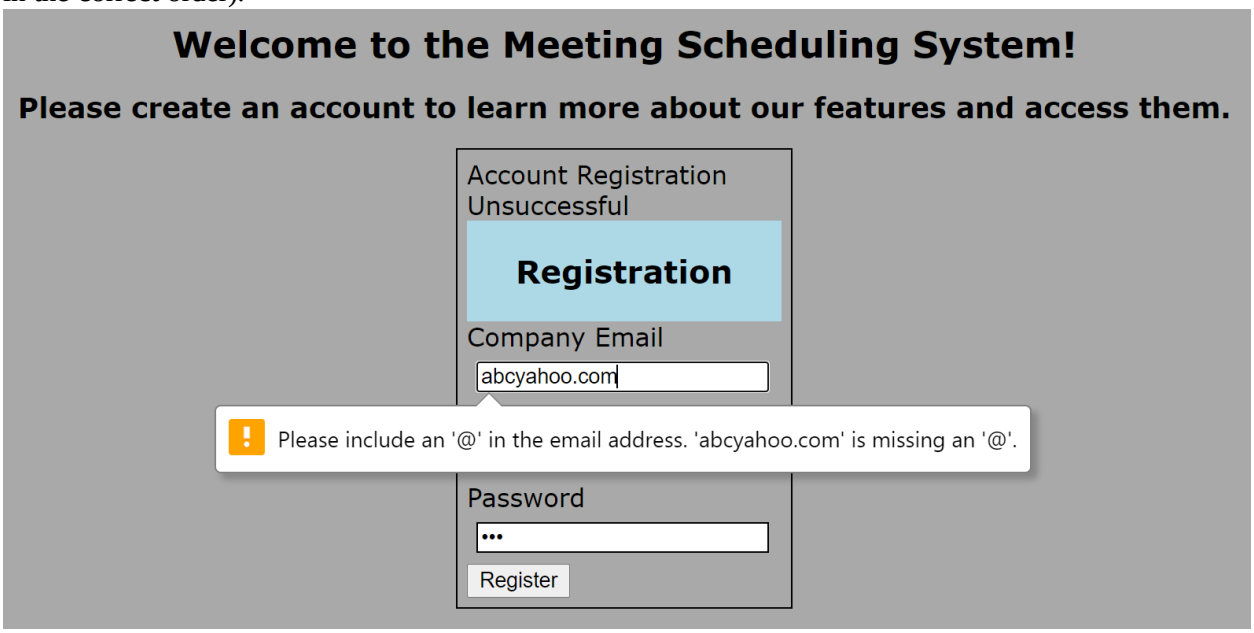


Figure 29: Input Validation for the email for registration

4.3.3 Strong Password Requirement

For strong password requirement, current requirement set is for the password to has at least 1 uppercase letter, 1 lowercase letter and 1 numeric character with the total length of password no less than 8. If the user enter an invalid password, the warning message as per figure 30 will be shown.

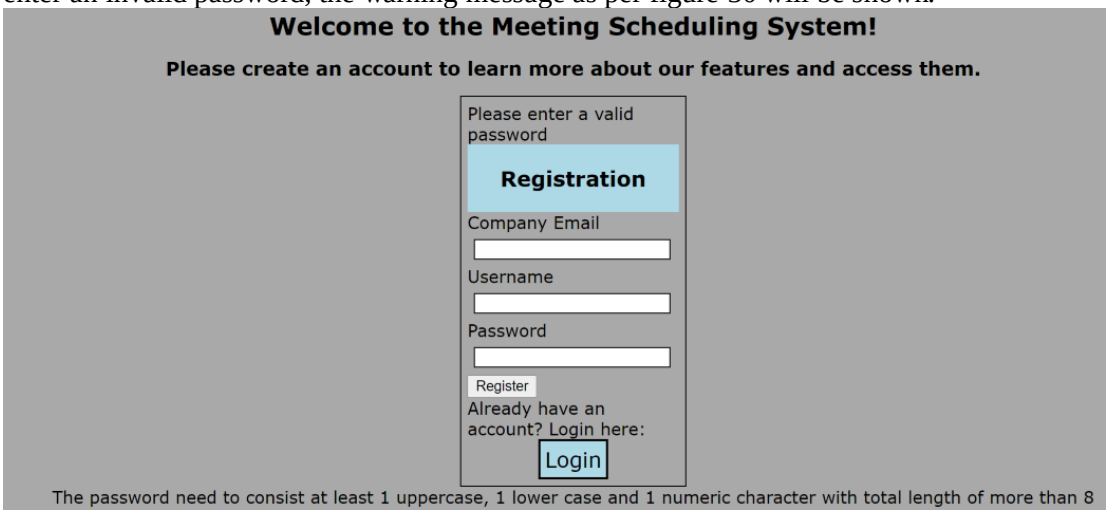


Figure 30: Strong Password Requirement verification

4.3.4 Input Verification

Example of input verification include: checking if the email exists in the database, checking if the email has been registered before, checking if the username has been registered before. In all cases if the email exist, registered before or the username registered before, an error message stating account registration unsuccessful will be shown.

Figure 31-33 show the attempt in registering using non-exist email or email/username registered before, figure 34 shows the expected response from the UI pages.

Figure 31: Attempting to register using email not exist in the database

Figure 32: Attempting to register using email registered before.

Figure 33: Attempting to register using username registered before

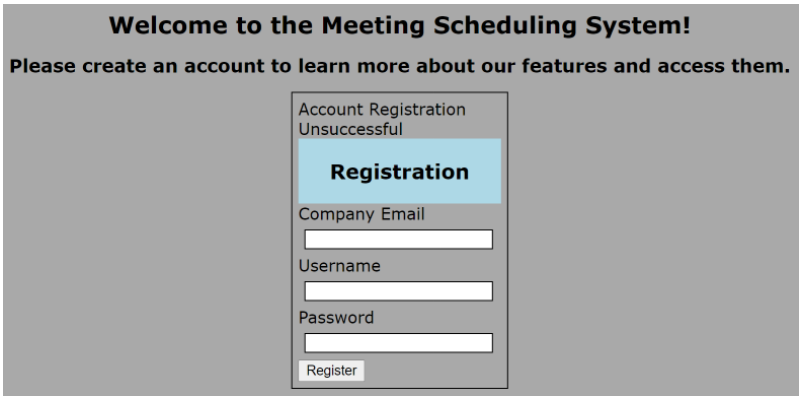


Figure 34: Error message when the input verification failed.

4.3.5 Input Sanitization

Using php MySQL prepared and execute() statement will perform the input sanitization to protect against the SQL injections. The prepared statement will automatically process the quote, so that it will be store correctly in the database. Figure 35 shows the test attempts to store a quote into the database and check if the database can store this code correctly, figure 36 shows value “John Wick’s son” has been processed correctly and store in the database without issues.

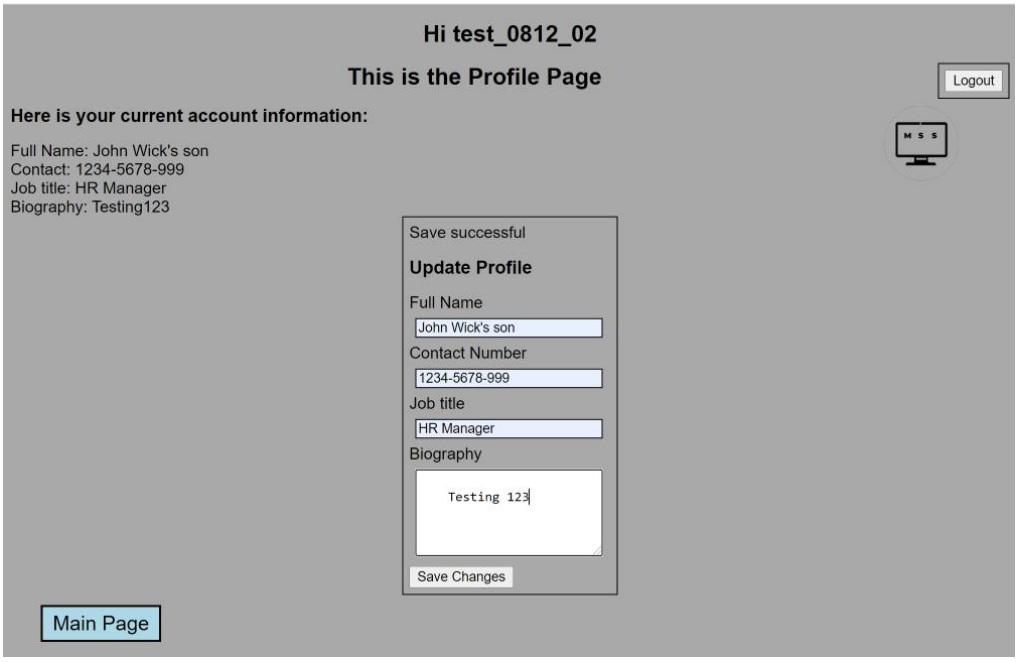
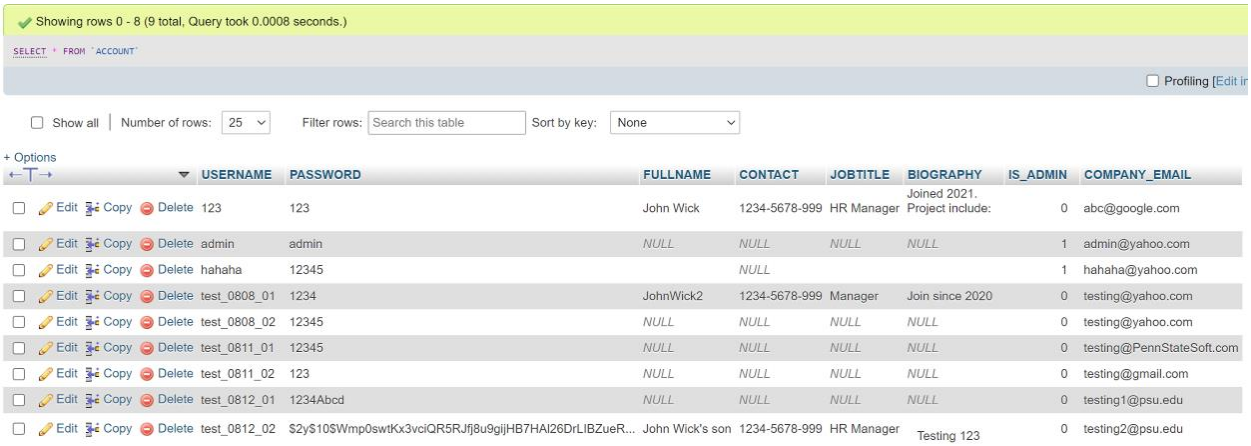


Figure 35: Testing with quote, a necessarily for SQL injections.



Showing rows 0 - 8 (9 total, Query took 0.0008 seconds.)

```
SELECT * FROM `ACCOUNT`
```

Profiling [Edit in

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

+ Options

	USERNAME	PASSWORD	FULLNAME	CONTACT	JOBTITLE	BIOGRAPHY	IS_ADMIN	COMPANY_EMAIL
Edit Copy Delete	123	123	John Wick	1234-5678-999	HR Manager	Joined 2021. Project include:	0	abc@google.com
Edit Copy Delete	admin	admin	NULL	NULL	NULL	NULL	1	admin@yahoo.com
Edit Copy Delete	hahaha	12345		NULL			1	hahaha@yahoo.com
Edit Copy Delete	test_0808_01	1234	JohnWick2	1234-5678-999	Manager	Join since 2020	0	testing@yahoo.com
Edit Copy Delete	test_0808_02	12345	NULL	NULL	NULL	NULL	0	testing@yahoo.com
Edit Copy Delete	test_0811_01	12345	NULL	NULL	NULL	NULL	0	testing@PennStateSoft.com
Edit Copy Delete	test_0811_02	123	NULL	NULL	NULL	NULL	0	testing@gmail.com
Edit Copy Delete	test_0812_01	1234Abcd	NULL	NULL	NULL	NULL	0	testing1@psu.edu
Edit Copy Delete	test_0812_02	$2y$10$Wmp0swtKx3vciQR5RJfj8u9gijHB7HAl26DrLIBZueR...	John Wick's son	1234-5678-999	HR Manager	Testing 123	0	testing2@psu.edu

Figure 36: Back-end database showing the quote correctly stored for the FullName for user test_0812_02

4.3.6 HTML Escaping

Whenever we want to display the data retrieved from the database, we need to perform html escaping by treating the text as htmlspecialchars. Figure 37 shows such attempt to insert the java script code into the database. If no HTML escaping performed, the java script will run as shown in figure 38. After performing HTML escaping, the script will not run and only display as text as shown in figure 39.

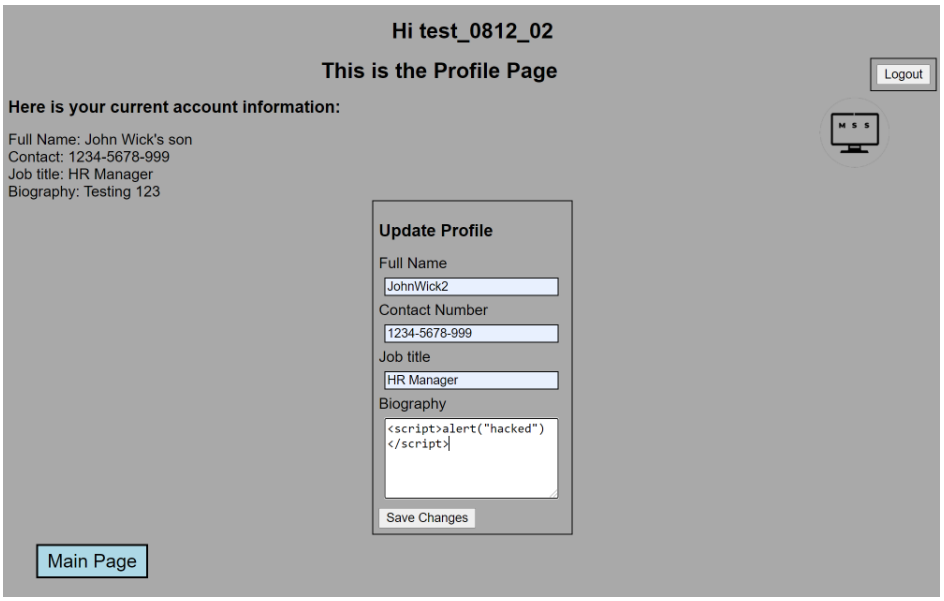


Figure 37: Attempt to insert java script code into the database

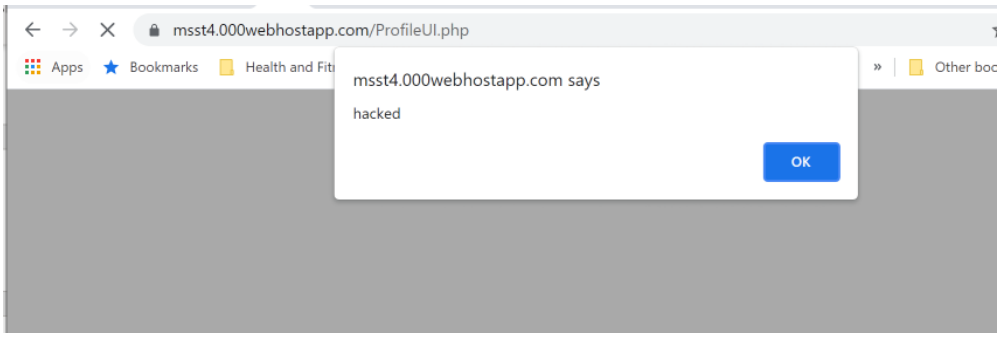


Figure 38: If no html escaping, the script will run and potentially redirected to malicious website.

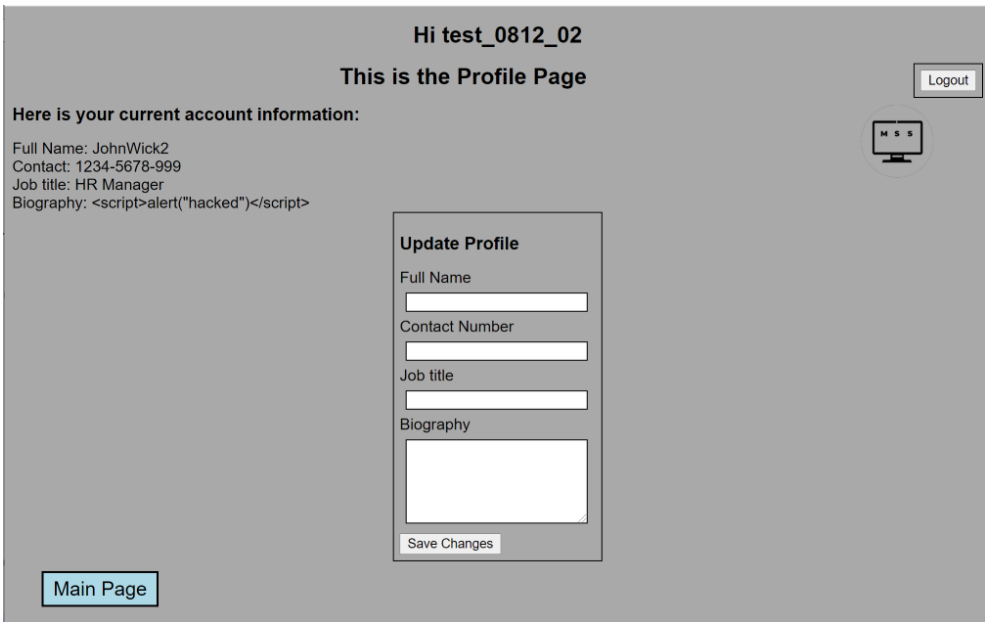


Figure 39: With html escaping, the text will be displayed without running the script.

4.3.7 CSRF Token

A Cross Site Request Forgery token is used to ensure that any form entry calling the controller class to the database came from authenticated source, to prevent the hacker using their own form to try to communicate with the database. When loading any of the UI page, a CSRF token of a random string will be generated and stored as the session variable $_SESSION['token']. This session token will be passed on as a hidden input to the form action. For every POST method called by the UI page, the system will check if the CSRF token submitted by the form is equal to the CSRF token stored in the session. A failure to match means the form came from unauthenticated source. The CSRF token will be refreshed every time the user reload the page, hence preventing the attacker from forging it.

Figure 40 shows the CSRF token generation code inside the php header section. The function get_random_string(60) will generate a random string with length up to 60 characters and passed to one of the hidden form input value, as shown in figure 41.

```php
<?php
    require "private/ProfileController.php";
    $login_controller = new LoginController($connection);

    $profile = new Profile();
    $username = "";
    $Error ="";

    //Check the login status
    if(!$login_controller->check_login()){ ...
    } else if(isset($_SESSION['username'])){ ...
    }

    //Trigger the logout function
    if(($_SERVER['REQUEST_METHOD']) == "POST" && isset($_POST['logout'])){ ...
    }

    //POST Method, check also if the CSRF token from the form is same as session token
    if($_SERVER['REQUEST_METHOD']=="POST" &&isset($_POST['save_profile']) && check_CSRF()){ ...
    }

    //CSRF Token Generation
    $_SESSION['token'] = get_random_string(60);
?>
<!DOCTYPE html>
```

Figure 40: CSRF Token Generation Code for the session

```
view-source:https://msst4.000webhostapp.com/Login.php
Apps  Bookmarks  Health and Fitness  Economy and Finan...  learning  blogadable  career  Life  Google Bookmark  Pro
Line wrap

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title> Login Page</title>
    <link rel="stylesheet" type="text/css" href="LoginStyle.css">
</head>
<body style="font-family: verdana;">
    <h1>Welcome to the Meeting Scheduling System! Please login to access our features.</h1>
    <img src="https://msst4.000webhostapp.com/logo_mss.png" alt="MSS Logo" title="MSS Logo"/>
    <form method="post">
        <div>
        </div>
        <div id="title"> Login </div>
        <! type can be email (validation), password (hide text) or normal text, required means the field is required>
        <div>
            <label>Username</label>
            <input id="textbox" type ="text" name = "username" required>
        </div>
        <div>
            <label>Password</label>
            <input id="textbox" type ="password" name = "password" required>
        </div>
        <!CSRF>
        <input type ="hidden" name = "token" value = "YODTaiM5bOdHQIRwXJpjEWl9aqAacmYipY1BlF">
        <input type = "submit" value = "Login">
        <div>
            <label>Don't have an account? Register here:
        </div>
        <a href="https://msst4.000webhostapp.com/Registration.php" class="BUTTON">Register</a>

    </form>

<div style="text-align: right;position: fixed;z-index:9999999;bottom: 0;width: auto;right: 1%;cursor: pointer;line-height: 0;displ
</html>
```

Figure 41: CSRF token value passed on to the form input

4.3.8 Session Variables

The system will keep the login status and whether the user is admin using the PHP $_SESSION variable. This session is called by the session_start(); command. The session variables will be set during login. Upon successful login application, the system will set the three global session variables, $_SESSION['username'] keeps the username of the user, $_SESSION['admin'] keeps the boolean value indicate whether the user is the admin, $_SESSION['expire'] will keep the time out value for the session.

```
function setLogin($username, $is_admin){
    $_SESSION ['username'] = $username;
    $_SESSION ['admin'] = $is_admin;
    //Set the expire session timing
    $_SESSION ['expire'] = time() + (1800);
}
```

Figure 42: The function to set the $_SESSION variables upon successful login

4.3.9 Session Timeout

The time out function in figure 43 is attached at the php section of every UI page. Thus every time the user trying to load the particular page (whether through manual selection or redirect from other pages), the system will check the current time against the session expiry time, and log out the user if the current time exceeds. Note the user can still stay on in the current page when the time-out expires, as long as they don't move to other UI pages.

```
    if((time()>$_SESSION['expire'])){
        $login_controller->logout();
    }
```

Figure 43: Time out function

4.3.10 Display Masking

Display masking simply mask part of the data retrieved from the database when displaying it out to the UI page. For example, one of the current system design is to mask the first 12 digits of the credit card information out when displaying it to the user, as shown in figure 44.

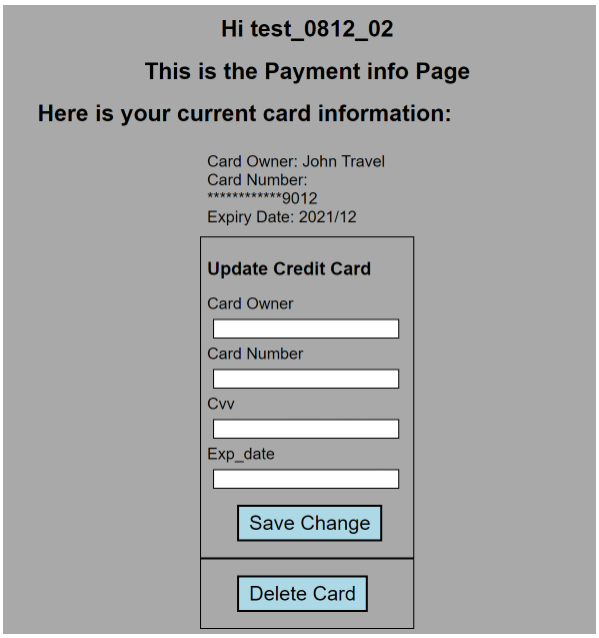


Figure 44: Masking of credit card number

4.3.11 Data Encryption and Decryption

There are three items identified as the sensitive item which required data encryption before storing it into the database. The first two are the credit card number and credit card cvv, these will be encrypted using open_ssl_encryption method and retrieved later using open_ssl_decryption. The last item to be encrypt is the password for the user account, this will be encrypted using password_hash and verify later using the password_verify function of the php. Figure 45 shows the encrypted credit card number and cvv stored in the database, and figure 46 shows the encrypted password stored in the database.

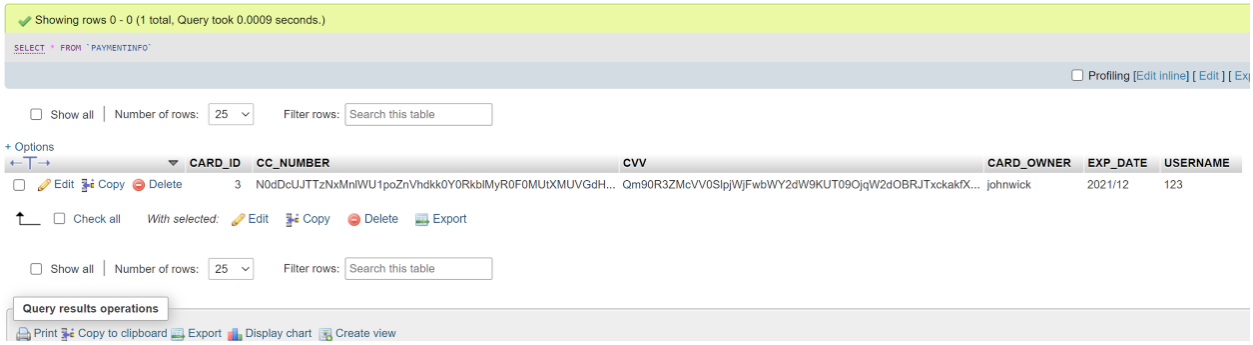
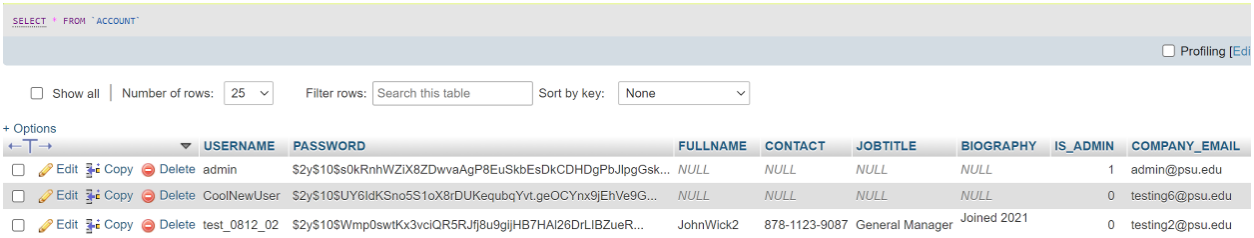


Figure 45: Encrypted Credit card number and CVV stored in the database



SELECT * FROM `ACCOUNT`

Profiling [Edi

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

+ Options

	USERNAME	PASSWORD	FULLNAME	CONTACT	JOBTITLE	BIOGRAPHY	IS_ADMIN	COMPANY_EMAIL
Edit Copy Delete	admin	$2y$10$s0kRnhWZiX8ZDwvaAgP8EuSkbEsDkCDHDgPbJlpgGsk...	NULL	NULL	NULL	NULL	1	admin@psu.edu
Edit Copy Delete	CoolNewUser	$2y$10$UY6IdKSno5S1oX8rDUKequbqYvt.geOCYnx9jEhVe9G...	NULL	NULL	NULL	NULL	0	testing6@psu.edu
Edit Copy Delete	test_0812_02	$2y$10$Wmp0swtKx3vciQR5RJfj8u9gljHB7HAl26DrLIBZueR...	JohnWick2	878-1123-9087	General Manager	Joined 2021	0	testing2@psu.edu

Figure 46: Encrypted password stored in the database

4.3.12 Vague Error Message

To prevent the attackers gaining too much information about the system, the error message should display as few details as possible. For example, during registration page, if the error message specifically tells the user the email or username is already registered in the database, the attacker can then use the known email or username to try to gain unauthorized access to the system. Hence, a vague error message as shown in figure 47 is required.

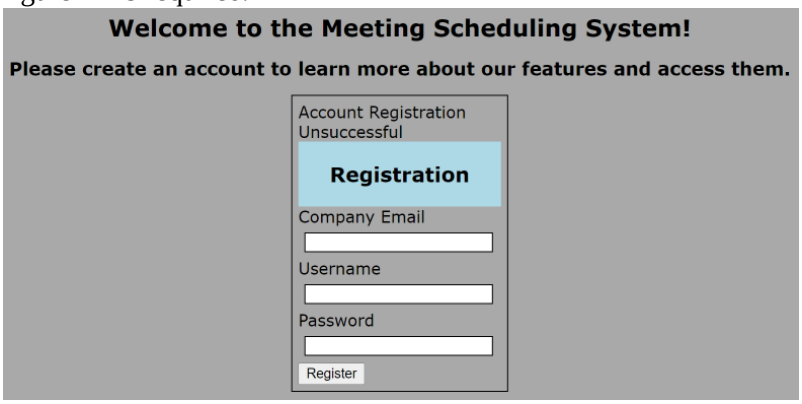


Figure 47: Example of vague error message.

4.3.13 Disabled Error Message

The error messages generated by running the php script at the server side is helpful to the developer in debugging the system. However it will reveal too much information about the system and hence should be disabled during the product deployment. The following line of code is required to hide all the error message, by setting the ini_set("display_errors", 0) value to 0 during loading of the php script. Figure 48 shows the result if the display_error value is set to 1, figure 49 shows the result if the display_error is disabled.

```
//Set display_errors = 0 to hide all error message, 1 to show the error message.
ini_set("display_errors", 1);
```

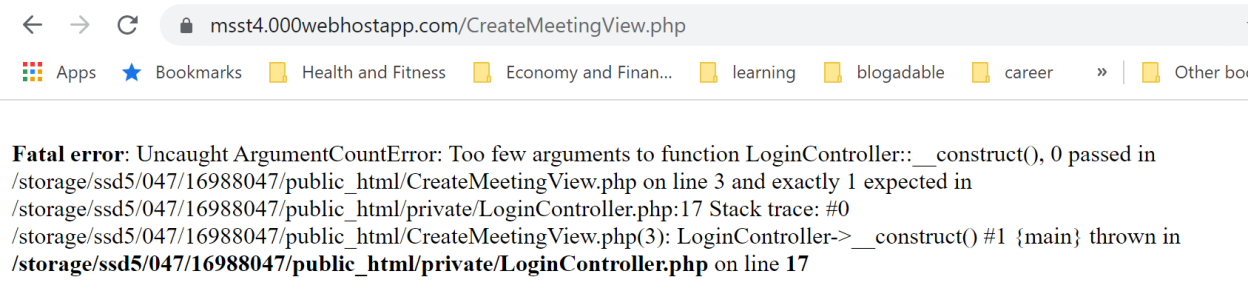


Figure 48: Error message displayed

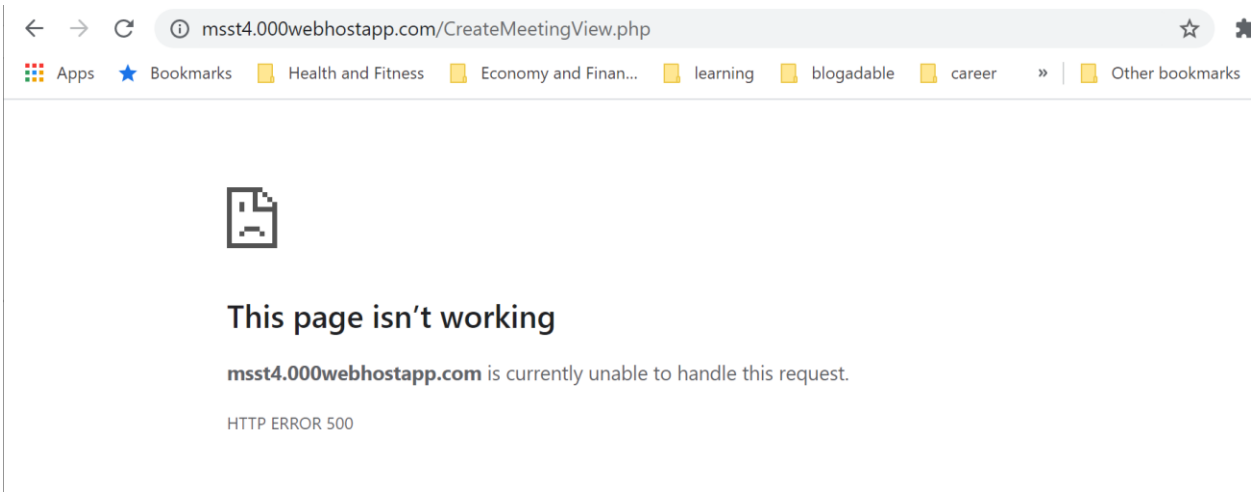


Figure 49: Error message turned off