**Secure Web Application Development**

**M2M Connect; SMS => PHP Processing**

**18-3110-AN**

Contents

[Introduction 2](#_Toc535188830)

[Security Techniques 2](#_Toc535188831)

[Specification 4](#_Toc535188832)

[General Usage Specification 4](#_Toc535188833)

[Security/Authentication Specification 4](#_Toc535188834)

[Implementation 5](#_Toc535188835)

[Use Case Diagram 5](#_Toc535188836)

[Testing 6](#_Toc535188837)

[Testing Strategy 6](#_Toc535188838)

[Testing Plan 7](#_Toc535188839)

[Testing Results 8](#_Toc535188840)

[Analysis and Recommendations 9](#_Toc535188841)

[User Guide 10](#_Toc535188842)

[Individual Contributions 12](#_Toc535188843)

[Vinay Tailor 12](#_Toc535188844)

[Mykolas Kaminskas 12](#_Toc535188845)

[Matthew Baber 12](#_Toc535188846)

[Bibliography 13](#_Toc535188847)

# Introduction

For this assignment with have been tasked with creating a web application in PHP that will receive and securely download SMS messages, more commonly known as text messages, from a M2M server. Once the message has been downloaded, it will need to be parsed by the web application, and any content contains validated. Finally, users should be able to log into the web application, or register a new account if they do not have one, and view the message’s metadata, which will contain info about its source, and the message’s content, which will contain information from a circuit board being used to send these messages.

# Security Techniques

When building a web application such as this one it is imperative to consider the security techniques and aspects that will be used to deter any users for using it for their own malicious purposes, as well as to preclude any attempts to do so. To this end, it has been deemed necessary to investigate what is considered the 3 most potent threats to the security of a web application that has been created in PHP, as well as techniques that can be implemented to reduce the chance of these threats being utilised and even remove them entirely.

The first, and debatably the most potent, of these potential threats is something called SQL Injection (David Shirey, 2012), which has been used to gain access to the servers of multiple major organisations such as the Wall Street Journal in 2014 (Jeremy Kirk, 2014) and even the World Trade Organisation in 2015 (Pierluigi Paganini, 2015). An SQL Injection attack occurs when the user enters SQL commands, such as the “SELECT” command, into an input box on the web application in an attempt to manipulate the database that is associated with the web application to give themselves access to it, bypassing any security and authentication that may have been put in place on the database. Naturally, the user gaining unauthorised access to the web application’s associated database via SQL injection is exceptionally bad and is definitely something that will want to be avoided.

There are a number of security techniques that can be utilised in the prevention of SQL injection attacks, each with varying degrees of difficulty and effectiveness. However, it is widely agreed that the most effective countermeasure to these attacks is the usage of a “prepare” statement. This is due to the fact that using a prepare statement will separate out the query that is being sent to the SQL server, and the data that the user is attempting to enter into the server. This will ensure that, if the user starts entering SQL commands into the input box, these commands won’t be sent to the server as a part of the query and thus executed. Instead, they will be sent to the server as the data that will be used by the query which will have already been sent to the server, meaning any commands the user has entered will not be carried out.

The second of the potential threats that could be utilised against the web application for malicious purposes is known as Cross Site Scripting (XXS) (WordFence, 2018), a famous, albeit harmless example of this being the Samy worm that was used on Myspace in 2005 (Samy, 2005). Cross site scripting works in a similarly to the aforementioned SQL Injection where the user injects their own malicious code into the web application, usually in the form of JavaScript. However, unlike SQL injection attacks, these attacks affect the web application’s users rather than the web application itself as the malicious script will be run every time they open the web page. How this occurs depends on the type of Cross Site Scripting. A reflected XSS attack occurs when the malicious code is “reflected” off the server being used to host the application and is sent to the user as part of a request, for example, the user opening a malicious link in a phishing email (OWASP, 2018). Whereas, a stored XSS attack occurs when the malicious code is stored in the web server itself, usually in a database, to be sent out to any users that attempt to access the web application (Margaret Rouse, 2018).

There are 3 main techniques and methods that can be employed when attempting to prevent cross site scripting: Escaping or Validating the User Input and Sanitization (Sarah Vonnegut, 2017). The first of these techniques, escaping, involves replacing the potentially harmful characters, for example <, with their equivalent in ASCII characters to prevent them being inserted into the HTML or JavaScript of the application for malicious purposes (Richard Ishida, 2010). The second technique, validating user input, is essentially filtering out any characters that could potentially be used for malicious purposes from a user’s input. For example, filtering out the characters < and > from a text box used to submit a user’s name as those characters are not used within first names or last names and could potentially be used in malicious script that the user may be attempting to inject into the web application. Naturally, these filters will need to be context sensitive and allow the usage of certain characters in certain scenarios, eg the @ character for an email address. The final technique, sanitization, is rather similar to the second technique but with one key difference. Instead of simply filtering out the potentially malicious characters, sanitization “corrects” the user’s input by automatically removing or changing any of these characters (oracle, No Date Given).

The final threat of what is considered the 3 most severe threats to a web application’s security is called session hijacking (David Shirey, 2012). As the threat’s name implies, session hijacking involves taking control of a user’s session through their session ID and then pretending to be said user, allowing all sorts of malicious things to be done to the user’s account or the web application itself if they turn out to be an administrator (Margaret Rouse, 2006). There are 2 main ways in which the session ID of a user can be obtained for a session hijack: session sniffing and cross-site scripting. Cross site scripting is simply using the previous security threat to access the cookie that contains the session ID of the user, thus allowing the session to be hijacked. Naturally this is particularly worrisome if the session ID is being obtained through stored XSS as that will enable the hijacking of every session that is using the server. The other way the ID can be obtained is by using something called a sniffer to search for sessions that are connected to the web server, then attempting to “capture” the session ID before it can be sent to the web server (OWASP, 2014).

The most effective way to prevent session hacking is to prevent it before it can occur, that is, prevent the hacker from accessing the session ids of the web application’s users. Naturally, by protecting the web application from cross site scripting using the previously mentioned methods, you will also be protecting its users from one of the methods used to obtain their session ids. As such, the only other way the user’s session ID could potentially be obtained is through the usage of a sniffer. However, this can also be quite easily prevented by simply encrypting the cookie and the session ID that it contains. This means that, even if the cookie was captured by someone before it reached the web server, said person would not be able to use it as they would not have the decryption key required to decrypt the encryption on the cookie and thus view the session id (Mike Chapple, 2009).

While each of these techniques are, by themselves, not particularly effective at preventing all of the threats to the web application’s security, when used in conjunction with one another, they prove to be an effective prevention method and deterrent for anyone that would potentially make and malicious changes to the web application.

# Specification

Before designing and implementing a system or application, it is a good idea to produce a specification for it as this will help give us, the developers, and idea of what is required by the system and thus how we should design it. Below is a general specification for the web application we will be producing using PHP.

## General Usage Specification

* The web application must be able to connect to the M2M server
* The web application must be able to download the SMS messages from the M2M server
* The web application must parse (breakdown) these messages
* The web application must store the message’s metadata (message info eg the time sent) in its database
* The web application must store the message’s content in its database
* The user must be able to create and account on the web application
* The user must be able to log into their account on the web application
* The user must be able to view the stored message metadata
* The user must be able to view the stored message contents

## Security/Authentication Specification

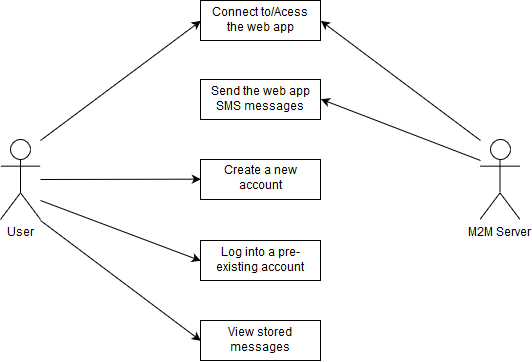
* The session ID of the web application’s user must be protected using encryption, as must the connection to the server the messages are being downloaded from
* The web application must be protected from SQL Injection attacks through the usage of techniques such as prepare statements
* The web application must be protected from XSS through the use of techniques such as sanitization
* The web application must be validated by W3C validation

# Implementation

As specified in the assignment’s specification, the web application will be built using the PHP language with the utilization of the “SLIM framework” to assist in the web application’s creation. The web application will also utilise the “Twig template engine” to help build the HTML web pages that its users will be viewing by, as the name suggest, creating premade HTML templates for the web app to use when it creates the web pages for the user to view. Both this framework and the template engine will be used in a MVC (Model, View and Controller) format with the SLIM framework acting as the controller for the web application.

The MVC approach to build a web application is a rather popular one and can be applied to many different languages, not just PHP. As the name suggest, the approach splits the web application into 3 different parts: the Model, the View and the Controller. The model part of this approach is used to describe the data that will be used within the application, meaning it is not dependent on the on the other 2 parts of the approach, the view and the controller. Speaking of which, the second part of the approach, the view, is what displays the data (model) to the user, and sends any inputs to the third part of the approach, the controller. The view can either be independent of the model and the controller, or it can actually be a part of the controller itself, making it dependent on the model. Finally, the controller is “the brain” of the approach. It sends the data to the view and processes the user’s inputs.

During the implementation of the web application will we be utilising some of the models that Clinton Ingrams created for the lab sessions when we were first learning how to use PHP with SLIM. This was simply done help have time during the application’s creation and allow us to implement more functionality into it.

Use Case Diagram

# Testing

## Testing Strategy

As the testing of the system is a key part of the project, to ensure that the web application is functioning and secure, it is important to organise it through the use of a testing scope to define key aspects of said testing. As agreed by the team during our first meeting after being given the project, Matthew will be responsible for the testing of the application and thus will be creating the plans for the testing, as well as doing the actual testing itself. The testing itself will mostly be white-box testing, which is the testing of the code used within the application rather than the application as a whole, as is required by the project’s specification, and will particularly focus on the security aspects of the web application and the prevention of the threats that were mentioned in the security techniques portion of the document. The testing itself will not take long, a day at the most, as most of the tests will be relatively simple and should be very short provided that nothing goes wrong or there are no errors with the coding, which is what the will be tested anyway.

The approach that will be used in the testing of the application will be a relatively simplistic one, as not to overcomplicate the process and make it easier to reproduce should future testing be required. The approach itself will simply involve the creation of a testing plan with written instructions on how to carry out each of the tests on the application, for future ease of use. Once a test has been conducted, its results will be recorded in a table and conclusions drawn once all the tests have been completed. The testing of the application will mainly focus on its security, looking at how it prevents some of the aforementioned threats to its security, but will also look at the application’s functionality and investigate whether it has met the requirements that were specified for it there.

The environment of the web application’s testing is an important thing to take into account as different environments can have different effects on the application, due to how the different programs being used to host the application are coded. That said, during the its testing, the application will most likely be hosted using a program called UwAmp, which acts as a WAMP server and thus allows the tester (Matthew) to host the application wherever he is. The application itself will be viewed using a web browser such a Mozilla Firefox.

As the testing approach is so simple, not many tools will be required during the testing of the application, and the tools that are required are relatively simple to use, making the testing easy to replicate. As previously mentioned, UwAmp will be utilised as a WAMP server to host the web application, and Mozilla Firefox will be used to view it. The testing plans and results of said test will be record in a word document, most likely this one.

As the testing will be occurring in an enclosed environment, there is not really any notable risk to the testing procedure. The only potential risk would have been corruption of the files used by the application, however this is complete nullified by the fact that the group is using GitHub to work on the project and thus a backup version is always available in case this does occur.

## Testing Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Test Name | Test Purpose | Test Instructions | Predicted Result |
| Test 1: M2M Connection Test | Testing whether the web application can connect to the M2M server being used to send the messages | Log into the application as a user | The application should automatically connect to the server when the user logs in |
| Test 2: SMS Download Test | Testing whether the web application can download SMS messages from the sever, parse them and then store them in its database | Log into the application as a user | The application should automatically download, parse and store the messages when the user logs in |
| Test 3: Account Creation Test | Testing whether the user is able to create a new account/profile using the account creator | Create an account using the application’s account creator | The application should create a new account using the given details |
| Test 3b: Account Creation Without Details Test | Testing whether the user is able to create a new account/profile using the account creator without entering any details into the boxes | Create an account using the application’s account creator without entering any details | The application should reject the request as there have been no details given |
| Test 4: Login Test | Testing whether the user can successfully log into the application | Log into the application using the right details | The application should accept the user’s details and log them in |
| Test 4b: Login With Wrong Details Test | Testing whether the user can successfully log into the application with incorrect login details | Log into the application using the incorrect details | The application should reject the request to login with the incorrect details |
| Test 4c: Login Without Details Test | Testing whether the user can successfully log into the application without entering any login details | Log into the application without any details | The application should reject the request to login with no details |
| Test 5: Data Retrieval Test | Testing whether the web application can retrieve information is has stored in its database | Successfully log into the application and view the stored messages | The application should retrieve the stored messages and display them to the user |
| Test 6: Session Hijack Test | Testing whether someone will be able to view the session ID of a user and thus whether the web app is vulnerable to session hijacks | Log into the application and view its temp/session folder to try and view the session ID | The session ID should be encrypted, making it impossible to read and use to hijack the session |
| Test 7: Cross Site Scripting Test | Testing whether the application is vulnerable to cross site scripting | Enter a piece of alert script into the login details | The application should reject the malicious JavaScript code |
| Test 8: SQL Injection Test | Testing whether the web application is vulnerable to SQL injection attacks | Enter a SELECT statement into the login details | The application should reject the malicious SQL code |
| Test 9: W3C Validation Test | Testing whether the web application passes W3C validation | Run the application through the W3C validator | The application should pass W3C validation |

## Testing Results

|  |  |  |
| --- | --- | --- |
| Test Number | Predicted Result | Actual Result |
| Test 1: M2M Connection Test | The application should automatically connect to the server when the user logs in | Pass: The application automatically connected to the server on login |
| Test 2: SMS Download Test | The application should automatically download, parse and store the messages when the user logs in | Pass: the application automatically downloaded, parsed and stores the messages on login |
| Test 3: Account Creation Test | The application should create a new account using the given details | Pass: the application created a new account using the given details |
| Test 3b: Account Creation Without Details Test | The application should reject the request as there have been no details given | Pass: the application rejected the request |
| Test 4: Login Test | The application should accept the user’s details and log them in | Pass: the application accepted the login details and logged the user in |
| Test 4b: Login With Wrong Details Test | The application should reject the request to login with the incorrect details | Pass: the application rejected the login request |
| Test 4c: Login Without Details Test | The application should reject the request to login with no details | Pass: the application rejected the login request |
| Test 5: Data Retrieval Test | The application should retrieve the stored messages and display them to the user | Pass: the application retrieved the stores messages and displayed them |
| Test 6: Session Hijack Test | The session ID should be encrypted, making it impossible to read and use to hijack the session | Pass: the session ID was encrypted, making it impossible to read |
| Test 7: Cross Site Scripting Test | The application should reject the malicious JavaScript code | Pass: the application rejected the malicious code |
| Test 8: SQL Injection Test | The application should reject the malicious SQL code | Pass: the application rejected the malicious code |
| Test 9: W3C Validation Test | The application should pass W3C validation | Pass: the application passed the WC3 validation |

# Analysis and Recommendations

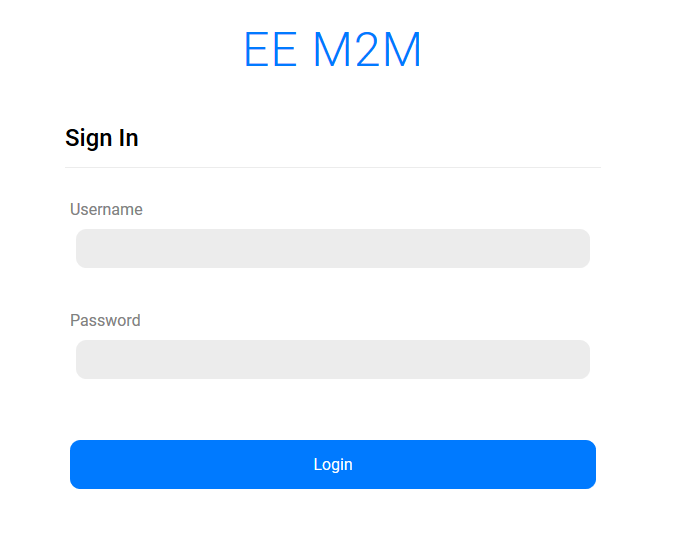
As shown by the above testing results, the web application passed the tests that were laid out before it and thus meets the requirements that were laid out in the earlier specification. The application functioned as intended, allowing the user to log in and view the messages being retrieved from the M2M server in an easy to read format, and allowed them to create new profiles to log into the application with, and was also properly secured from the threats that were discussed in the security techniques part of this document using the techniques that were discussed in this same section. The testing of the application did pull of a few errors here and there, but these errors were resolved as soon as they occurred, meaning that they had minimal impact on the testing itself and are very unlikely to occur again.

While the application is functioning correctly and as specified in the specification of this document, there are several improvements and recommendations that could be made to it, based on the extensions that could be done for the implementation. One such improvement could be displaying of some of the data being included in the messages, such as the temperature of the circuit, in a graph format such as a line graph. This would result in the data being conveyed to the user much faster and easier to understand format, rather than displaying it to them through text which they would have to read.

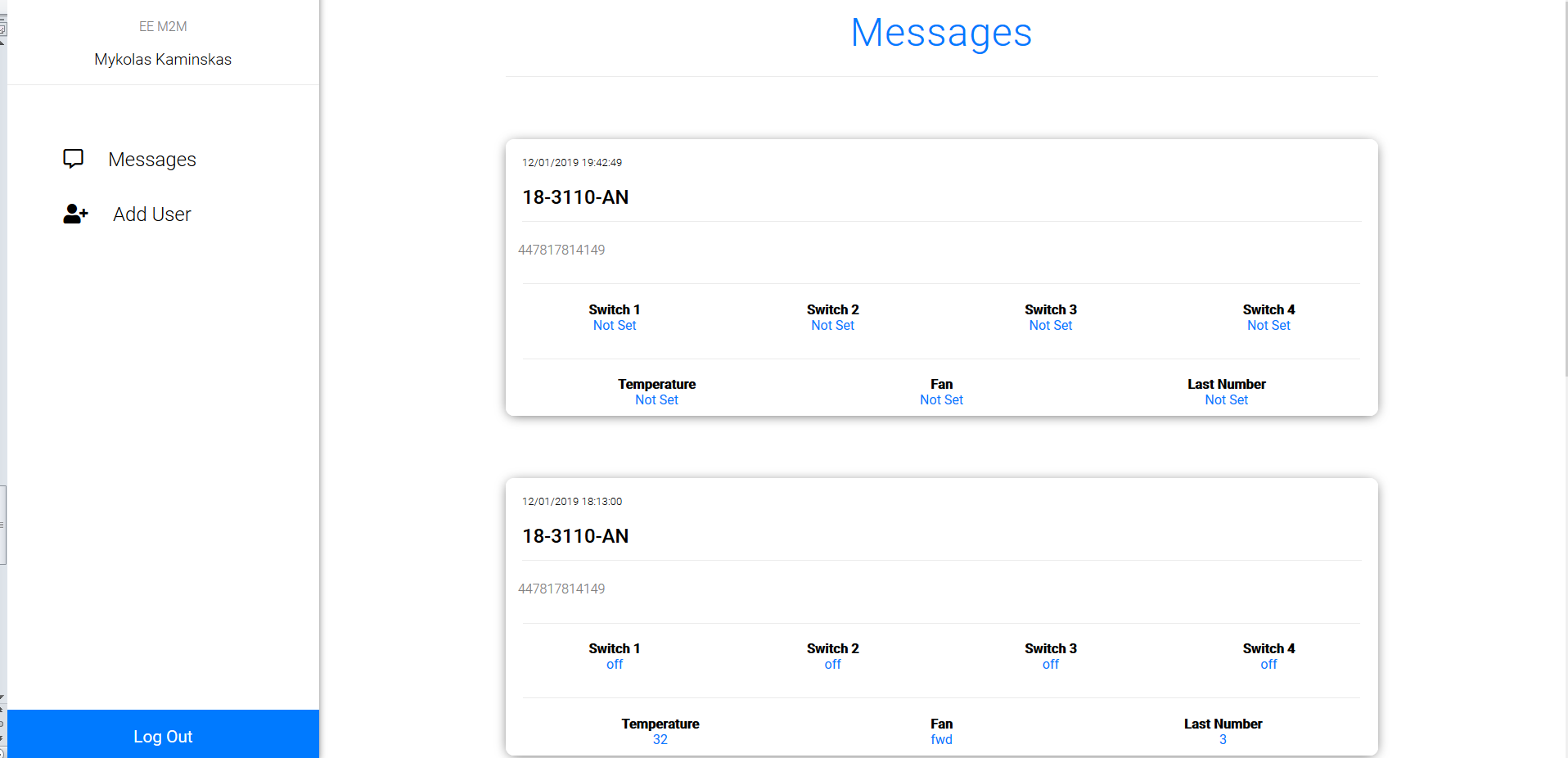
Another recommendation would be moving the option to ass a user to the application to the login page, rather than on the homepage. This is simply due to the fact that it would allow for the creation of new accounts without the user having to log into the application to create it. Meaning that if they do not already have an account registered on the system they will not have to ask another user to login and create a new account for them.

Finally, a third improvement that could be made to the web application would be using Java Script variants such as AJAX and JSON to allow the application to automatically update the page with the new messages from the M2M server as they are downloaded and parsed.

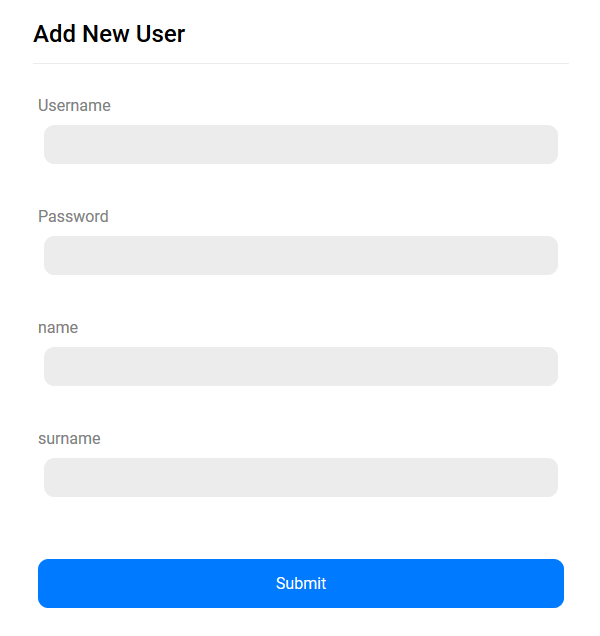
# User Guide

The application is very simple and self-explanatory. When you first access it you will be greeted by the application’s login page, which is picture below.

To access the application, simple enter your username and password into the retrospective boxes and click on the “Login” button.

Once you have logged into the application using the above method, you will reach the application’s hompages. Here you can view the messages that have been retrieved from the M2M server and stored in the application’s database, which will be displayed to you in an easy to understand format, as shown in the below picture. 

As is evidenced by the picture, you will also be able to add and new users to the application and log out of the account you have signed into. To log out of the account, simple click the “Log Out” button in the bottom left of the page to be taken back to the login page

Clicking “Add User” will cause you to be taken to the page shown below, where you will be able to create a new user account on the application.

To create a new account simply enter the details into the appropriate boxes and click the “Submit” button, which will create the account using the details that were entered and will take you back to the home page.

# Individual Contributions

Just after the teams for this assignment were announced and thus we knew we would be working together, we organised to meet up during one of our Friday lab sessions to organise ourselves and decide who was doing what for the project. We used the recommended roles as a template for our task assignment, but we made a few adjustments to make it better suit the skillset of each of the members of the team. Eventually, we decided upon the below allocation of tasks:

## Vinay Tailor

* SOAP Server
  + Creating the connection between the SOAP server and the web application
  + Getting the web application to download the messages from the server and stored in the database
  + Parsing the message’s XML content
  + Getting messages to display in the web app
* Logging user activity
* PHP Extensions
* XDebug
* Preventing duplicates in SQL table
* Code comments

## Mykolas Kaminskas

* SOAP Server
  + Getting the web application to download the messages from the server and stored in the database
* Code comments
* Sunnitisation
* Validation
* Login and register system
* SQL queries and SQL wrapper
* Sessions

## Matthew Baber

* Documentation
  + Researching and writing about Security Techniques
  + Creating a Specification
  + Creating a Use Case Diagram
  + Creating a User Guide
* Testing
  + Creating Testing Plans and Strategy
  + Conducting Tests and Analysing the results

# Bibliography

David Shirey (2012) Top 10 PHP Security Vulnerabilities Available from: <https://www.sitepoint.com/top-10-php-security-vulnerabilities/> Accessed: Wednesday 2nd January

Jeremy Kirk (2014) SQL injection flaw in Wall Street Journal database led to breach Available from: <https://www.pcworld.com/article/2457240/sql-injection-flaw-in-wall-street-journal-database-led-to-breach.html> Accessed: Wednesday 2nd January

Pierluigi Paganini (2015) Anonymous Hacker breached WTO database and Leaked data of internal staff Available from: <http://securityaffairs.co/wordpress/36528/hacking/anonymous-breached-wto-db.html> Accessed: Wednesday 2nd January

WordFence (2018) Understanding PHP Vulnerabilities & How They Originate Available from: <https://www.wordfence.com/learn/understanding-php-vulnerabilities/> Accessed: Wednesday 2nd January

Samy (2005) The MySpace Worm Available from: <https://samy.pl/myspace/> Accessed: Wednesday 2nd January

OWASP (2018) Cross-site Scripting (XSS) Available from: <https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)> Accessed: Wednesday 2nd January

Margaret Rouse (2018) cross-site scripting (XSS) Available from: <https://searchsecurity.techtarget.com/definition/cross-site-scripting> Accessed: Wednesday 2nd January

Sarah Vonnegut (2017) 3 Ways to Prevent XSS Available from: <https://www.checkmarx.com/2017/10/09/3-ways-prevent-xss/> Accessed: Thursday 3rd January

Richard Ishida (2010) Using character escapes in markup and CSS Available from: <https://www.w3.org/International/questions/qa-escapes> Accessed: Thursday 3rd January

Oracle (No Date Given) What Is Input Validation and Sanitization? Available from: <https://download.oracle.com/oll/tutorials/SQLInjection/html/lesson1/les01tmovw3.htm> Accessed: Thursday 3rd January

Margaret Rouse (2006) session hijacking (TCP session hijacking) Available from: <https://searchsoftwarequality.techtarget.com/definition/session-hijacking> Accessed: Friday 4th January

OWASP (2014) Session hijacking attack Available from: <https://www.owasp.org/index.php/Session_hijacking_attack> Accessed: Friday 4th January

Mike Chapple (2009) How to prevent network sniffing and eavesdropping Available from: <https://searchsecurity.techtarget.com/answer/How-to-prevent-network-sniffing-and-eavesdropping> Accessed: Friday 4th January

Clinton Ingrams (2015) <http://www.tech.dmu.ac.uk/~cfi/>