CTEC3110 Assignment Documentation

Grant Rigby – P15223403

Justin Shanks – P15225881

Sohail Raza – P1523890X

Table of Contents

[Entity Relationship Diagram 2](#_Toc502828394)

[Use Case Diagram 2](#_Toc502828395)

[Interactive Sequence 2](#_Toc502828396)

[Specification 2](#_Toc502828397)

[Implementation 2](#_Toc502828398)

[Test Plan and Logs 2](#_Toc502828399)

[Group Member Roles 2](#_Toc502828400)

[Summary and Conclusion 2](#_Toc502828401)

[Bibliography 2](#_Toc502828402)

# Entity Relationship Diagram

points to

bootstrap.php

index.php

login\_page.html.twig

settings.php

requires path to

displays

dependencies.php

autoload.php

displays

login\_page.php

routes.php

commands\_page.html.twig

mainpage.php

displays

send.php

download.php

display.php

change.php

requires

alert\_layout.html. twig

displays

table\_layout.html.twig

displays

formlet\_layout.html.twig

saved.php

displays

displays

# 

# Use Case / Iterative Sequence Diagrams

**Signing up, Logging In and Viewing the commands main page**

When a user first enters the app, they access the index.php fie in the public PHP folder. The index.php file calls the bootstrap file and proceeds to present the login / signup page to the user. When a user signs up (*login\_page.html.twig – Figure 1)* for the first time the users details including first name, surname, username and password are stored in an SQL table in a database. However if a user already has an account then they simply login (*login\_page.html.twig – Figure 2)*, and this retrieves the existing users details from the database table to ensure the user is a valid on the app. Both signing up for the first time and logging in again both lead to the user to be directed to the main commands page (*commands\_page.html.twig – Figure 3)*, which gives the user a list of operations that they can choose to interact with the app.

login\_page.html. twig

bootstrap.php

index.php

User

login\_page.php

Details are retrieved from the database

Logging In

Signing up

commands\_page.html.twig

mainpage.php

Details are stored in the database

Note that this only shows the paths up to when a user logs in or signs up and views the main page of the app. The other routes will be presented in separate diagrams following on from the end of this diagram. Which is the commands\_page.html.twig file.

**Downloading, Displaying and Saving messages**

Once the user has signed up, or logged in the first option they probably want to choose is to download any pending messages from the M2M server. This is achieved by simply clicking the download messages button. Note that this does not take the user to another page but rather displays a prompt at the bottom of the commands main page that confirms that the messages were successfully downloaded from the M2M server with a timestamp and a date (*commands\_page.html.twig – Figure 4)*. This is indicated by the arrow returning from downloads.php back to the commands\_page.html.twig file.

Once a user has successfully downloaded any pending messages then they can display these messages. These can be seen by clicking the ‘display messages’ button on the commands main page. This will then direct the user to a new page that shows a table of messages (*table\_layout.html.twig – Figure 5)*. Message details can be toggled on or off to expand or reduce the amount of information shown for the messages. Messages can also be saved to a users profile by clicking the ‘save messages’ icon. This will store the saved messages in the users unique session file as well as inserting them into the database table. Doing this will bring up a page that notifies the user that the downloaded messages were successfully saved *(alert\_layout.html. twig – Figure 6).* A user can also return to the commands page by pressing the ‘go back button’ this is represented by the arrow returning from the table\_layout.html.twig file back to the commands\_page.html.twig file.

Messages are downloaded from the M2M server and saved to the user’s session file

download.php

commands\_page.html.twig

alert\_layout.html. twig

table\_layout.html.twig

User can view the downloaded messages, and save the downloaded messages.

# 

display.php

**Altering the Saved messages and Sending a new message**

Once a user has saved messages in on their profile they can delete messages from their saved list. This can be done by clicking the ‘saved messages’ button on the commands main page. This will direct the user to another table layout page similar to the one mentioned before with the display, but will instead display all of the saved messages (*table\_layout.html.twig – Figure 7)*. This page will also allow a user to delete individual messages from their saved list of messages. Once again the user can return back to the commands page from the saved messages page by pressing the ‘go back’ button. This is represented by the arrow returning from the table\_layout.html.twig file back to the commands\_page.html.twig file.

A user may also want to send a message of their own to the M2M server. This can be achieved by pressing the ‘send message’ button on the commands main page. This will then direct the user to a new page that allows the user to send a message (*formlet\_layout.html.twig – Figure 8)*. A username and password will be required before a user can enter a message. Once the user has authenticated themselves they can select the options such as selecting the switch numbers, raising or lowering the temperature, specifying the fan direction and selecting a number on the keyboard. Finally, the user can send the message by pressing the ‘send message’ button. Like with all of the other pages the send message page allows a user to go back to the commands main page by pressing the ‘go back button’. Once again this is represented by the arrow returning from the formlet\_layout.html.twig file back to the commands\_page.html.twig file.

saved.php

table\_layout.html.twig

commands\_page.html.twig

The user can delete specific saved messages

formlet\_layout.html.twig

send.php

Allows the user to send a new message to the M2M server

**Changing the user’s password and Logging out**

The final command a user may choose is to change their password. This can be done by pressing the ‘change password’ button from the commands main page. Once this has been pressed it directs the user to a new page that allows a user to change their password (*formlet\_layout.html.twig – Figure 9)*. Like with sending a message authentication is required in the form of a username and a password. Once the user is authenticated they can enter their new desired password, but they also need to type it in again in the new password conformation box. This ensures that the user will not be locked out of their account in case they forget what they typed as their new password. A users new password will also updated in the users session file as well as being updated in the table of the database. As usual pressing the ‘go back’ button will take the user back to the commands main page. Once again this is represented by the arrow returning from the formlet\_layout.html.twig file back to the commands\_page.html.twig file.

The final action a user can do is to logout of the app. This can be done by pressing the ‘logout’ button from the commands main page. Doing this will take the user to a page that confirms that the user logged out, (*alert\_layout.html.twig – Figure 10)*. when the user presses the understood button the login page is presented back to the user (*login\_page.html.twig – Figure 1)*.

alert\_layout.html. twig

User presses the logout button

commands\_page.html.twig

login\_page.html. twig

User is logged out

formlet\_layout.html.twig

Allows the user to change their password. The session file and the database will be updated with the new user password.

change.php

# Specification

* Allow a user to create an account on the App, and for that account to be remembered so that the user can log back in with the same details they signed up with. This will use a session file for each user that signs up to the app.
* A first name, surname, user name and password will be required to sign up, and to login only the username and password will be required. It should also be noted that these entries will be fully sanitised and validated.
* When the user logs in or signs up for the first time, they will be directed to a main page that gives a list of further options. These options will include the ability to download any pending messages from the M2M server, to be able to display all of the downloaded messages, to save the downloaded messages, to alter the saved messages, to send a new message, to change a users password and to logout of the app.
* Each of these options will also redirect back to the main page with a “go back” button.
* The download messages option will simply prompt the user of the time and date that they downloaded the pending M2M messages, it will not direct the user to another page. It should also be noted that only messages with our group name should be downloaded from the M2M server.
* The display option will in fact direct the user to a new page that will give a table of all of the downloaded messages, there is also an option to filter the message information by metadata. This is changed by ticking the boxes at the top of the page to either reduce or expand the amount of information to display for the downloaded messages. This page will also allow a user to save the downloaded messages, to their profile.
* The saved messages option will direct the user to a new page that will display all of the saved messages in a table, and allow the user to delete individual saved messages.
* The send message will again send the user to a new page. Some authorisation will be required before a user can send a message in the form of a username and password. The messages a user can send will be to control things on a circuit board, including activating switches, the fan direction, the temperature, and choosing a digit on the keypad.
* Changing a user password will also direct the user to a new page that allows them to change their password. Doing so once again requires authentication in the form of the user’s username and current password. The new password should also be entered, but also a new password conformation to ensure that the user typed their new password correctly.
* Finally, the logout page simply logs the user out and gives a prompt to the user telling them that they successfully logged out.
* There will also be a SQL database with a table in that stores a users username, password, first name, surname and the saved messages of a user.

# Implementation

# 

# Test Plan and Logs

For the testing of the app, we decided to use the PHPUnit Framework. In particular, we decided to test if the Base64 encoder worked properly, we also ensured that the hashing class worked for passwords. Related to this we also tested to see if the encryption class worked when storing details in both the session file and the details inside the database. Aside from security and encoding we also tested to see if the Profile Model class worked to allow user details to be added and retrieved. The class that that allows user input to be sanitised and validated was also tested. The session model was also tested to ensure that a session can store details as well as being able to retrieve details from it, as well as the model the session wrapper file was also tested. Finally, we tested the connection the soap server, including both peeking and sending messages, as well as testing to ensure that downloaded messages are properly parsed into XML.

To run the tests follow the instructions below:

1. Change requires for the PhpUnit Tests

At the top of every test there will be a series of require\_once statements that link the test to the relevant files.

**require\_once '../../coursework/smsSeeker/app/src/SanitisedValidator.php';**

**require\_once '../../coursework/smsSeeker/app/settings.php';**

2. Run the PhpUnit Test

Change the dir to where the phpunit file is such as:

**C:> cd ./xampp/htdocs/vendor/bin/**

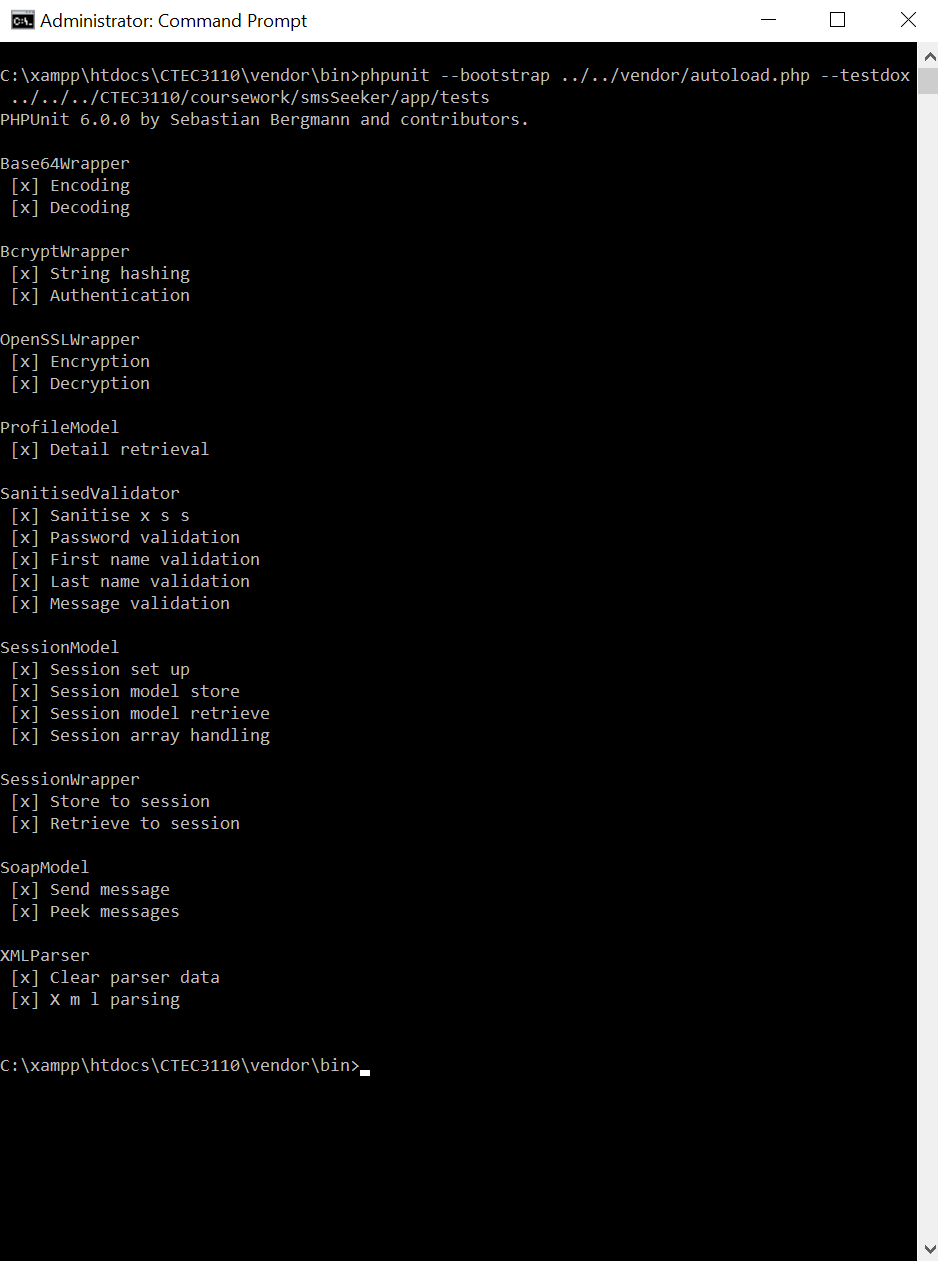
Then run to run all the tests:

**C:\xampp\htdocs\CTEC3110\vendor\bin>phpunit --bootstrap ../../vendor/autoload.php ../../../CTEC3110/coursework/smsSeeker/app/tests**

To specify a single test merely extend the above command with something along the lines of:

{previous\_command}/SanitisedValidatorTest

Below is an image that shows all the unit tests being passed, note each of the specific tests carried out for each category. Also note that passed test are denoted by a cross on the left side :



# Group Member Roles

**Justin Shanks – P15225881:**

* Set up and wrote the testing regiment using PhpUnit, the testing regiment included the source files except for the mysql parts.
* Designed and implemented the front-end of the application including the css, javascript and html components. The html was implemented using twig.
* Set up the functionality to store, retrieve, and clear information from session files.
* Set up the ability to connect, read from, and alter the mysql database.
* Set up the functionality to connect to the m2m server and, using the server’s soap commands, send and retrieve information.
* Set up the functionality to display both downloaded and saved messages from the session file to the html so that the user may see the messages easily.
  + Included javascript to help remove columns that the user did not need to view.
  + Included the ability to save specific messages for separate viewing.
* Implemented the functionality to encrypt, encode, and hash different pieces of information for the sake of security.
* Implemented the ability to sanitise and validate the username and password.
* Implemented the ability to change the user’s password and update relevant information in the related mysql database entries.

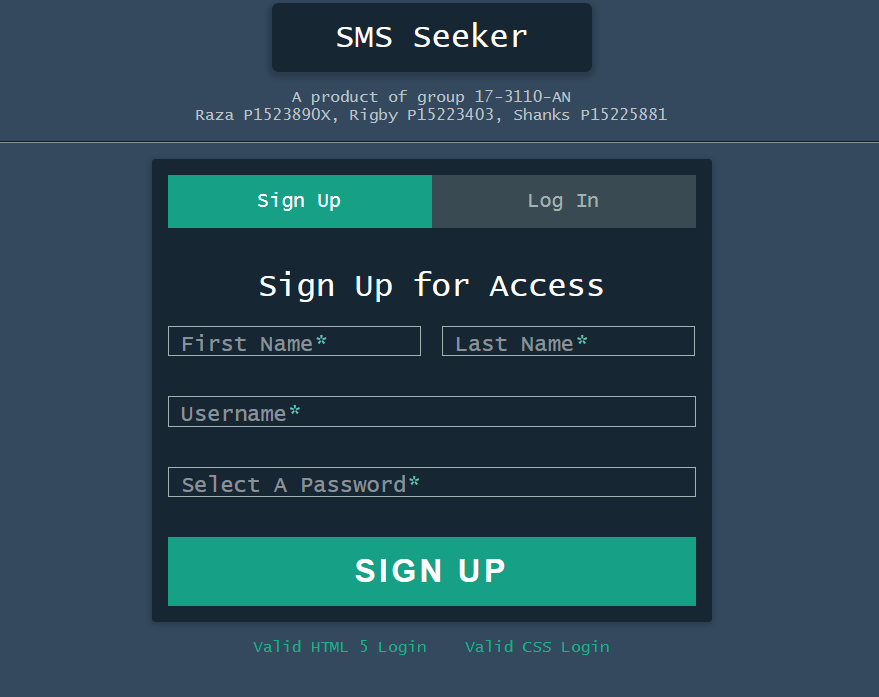
**Grant Rigby – P15223403:**

* Set up the template structure of the app, including creating all the different folders required and implementing some of the required files with template code to start with
* Helped plan the initial first draft design, mostly concerning the different classes and routes that were needed and their role in the app.
* Ensured the HTML and CSS was fully validated and added a link to the HTML and css validator for all of the html pages in the app.
* Created functions in the SanitisedValidator class to ensure a users first and last name are fully sanitised and validated when signing up and logging in.
* Assisted the other group members with the code where I could, such as help debugging issues, or discussing plans as to how implement features of the app.
* Removed a redundant html twig file and generally cleaned up the code.
* Commented the routes and class files with PHP DocBlock, making use of the @var, @param and @return where appropriate.
* Wrote the assignment documentation including the diagrams, the specification, the implementation and the test plan
* Co-ordinated communication with the other group members by setting a discord chat server, and organising meet ups at DMU.
* Provided the CD’s used to submit the work on
* Pushed regular commits to the group project GitHub page, under the alias GR412. The GitHub page can be found here: <https://github.com/theShanks96/RazaRigbyShanks>

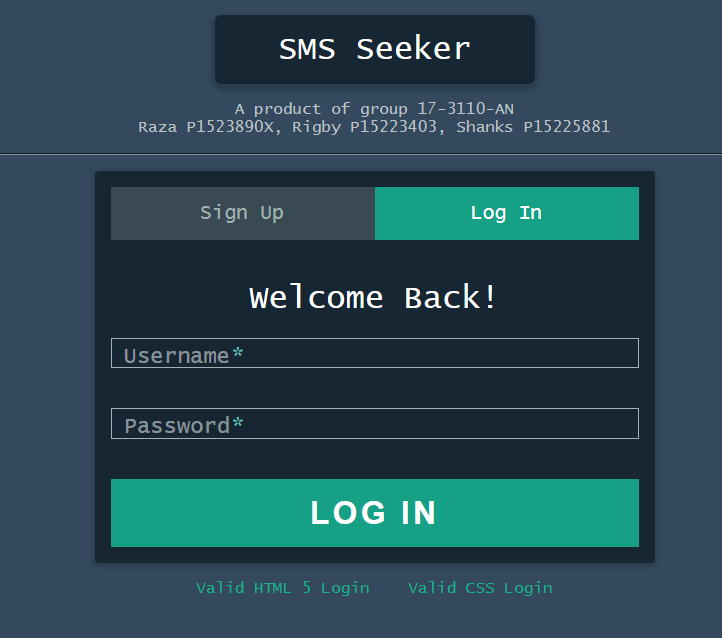
**Sohail Raza – P1523890X:**

* Completion of code in the src folder commenting alongside Grant Rigby
* Undertook the commenting within the testing folder (found exclusively on the GitHub server due to an error).
* Coordinated with group members to attend coding sessions and group meetings
* Set up MySQL features and commands which were later implemented into a larger system
* Pushed regular commits to the group Github server: <https://github.com/theShanks96/RazaRigbyShanks>
* Assisted Grant Rigby (P15223403) in the completion of assignment documentation including this current file

# Appendices



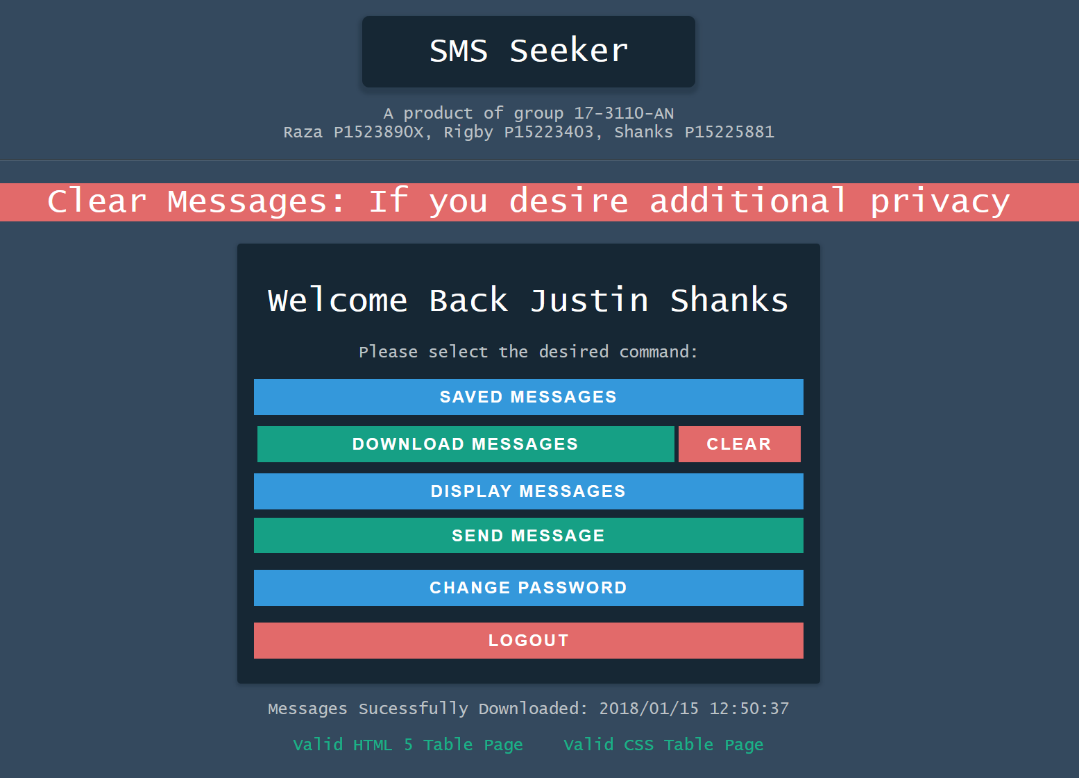
*Figure 1 – Depicting the login html page that has the signup tab open. This is the first page a user see’s when they enter the app. Note that the login\_page.php route file is used to control the signup process*

**

*Figure 2 - Depicting the login html page that has the login tab open. This is the first page a user see’s when they enter the app. Note that the login\_page.php route file is used to control the login process.*



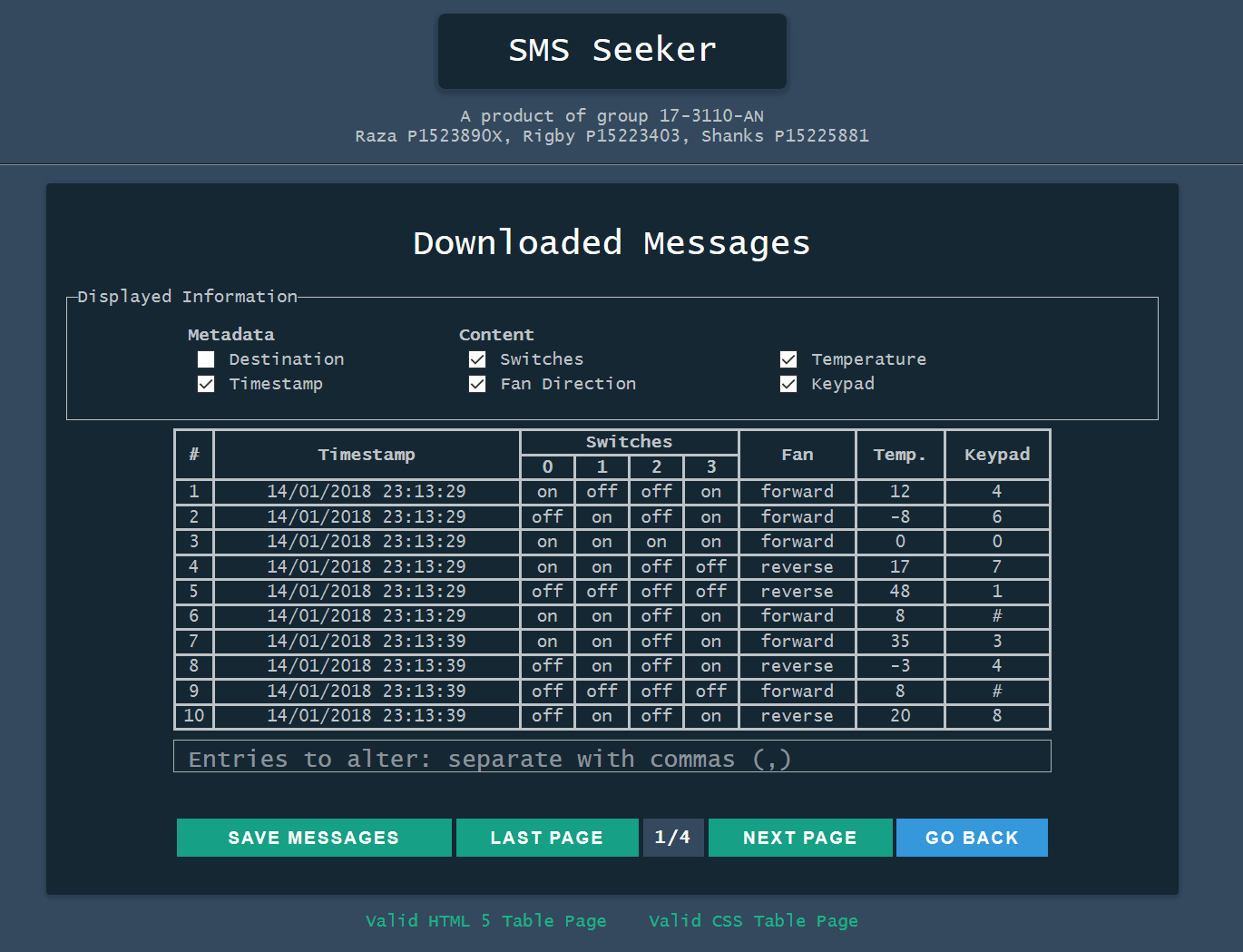
*Figure 3 – Depicting the commands\_page.html page that is the main page when a user logs in or signs up. It shows all the available operations a user can perform on the app. Note that the mainpage.php routes file is used to control the this.*



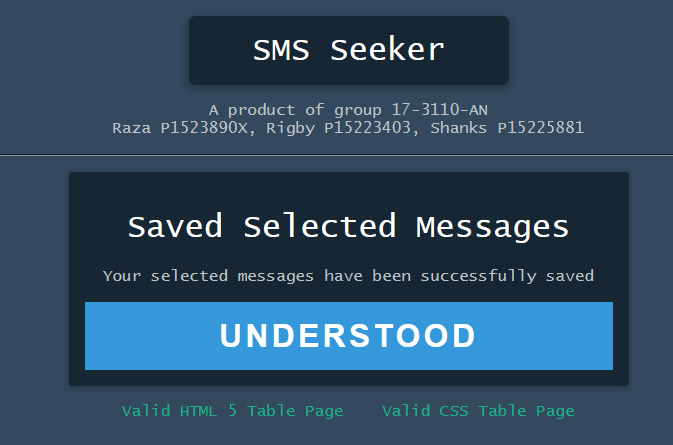
*Figure 4 – Depicting the commands\_page.html page but with the downloaded messages promt . Note that the mainpage.php routes file is used to control the this*



*Figure 4.5 – Depicting the commands\_page.html page but with the cleared messages prompt. Note that the mainpage.php routes file is used to control the this*

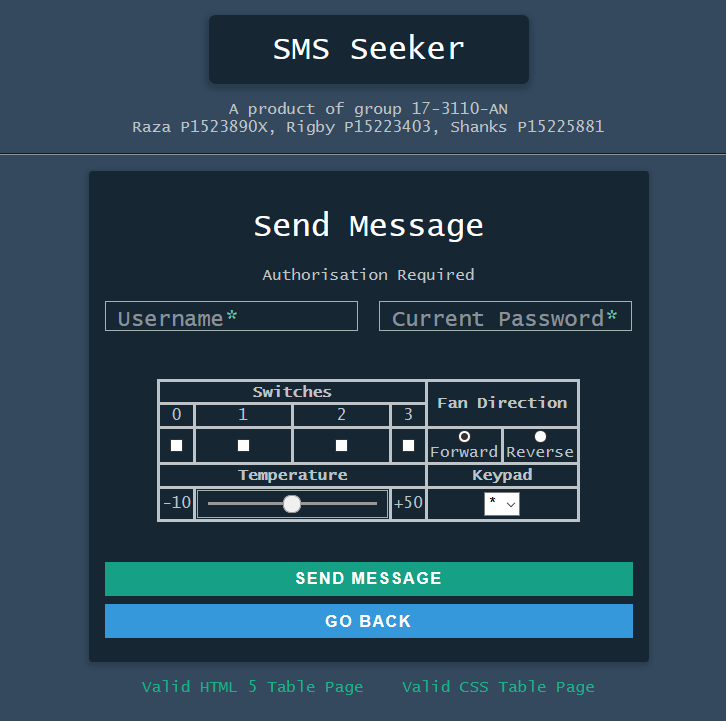


*Figure 5 – Depicting the* table\_layout.html.twig for the downloaded messages



*Figure 6 -*

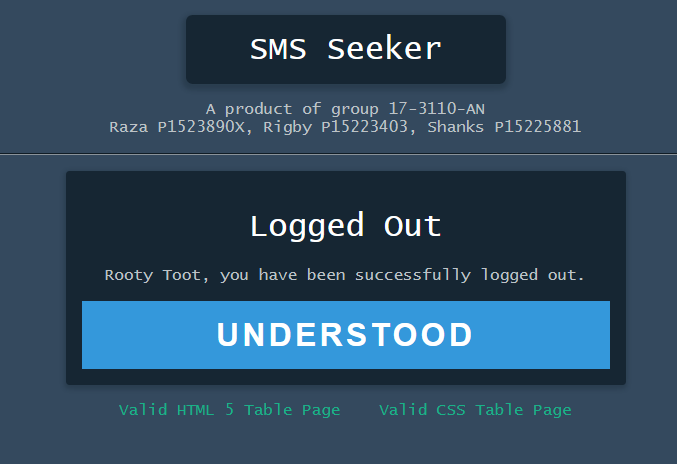
# 



*Figure 8 -*

# formlet_layout.html.twig – Figure 9.PNG

*Figure 9 -*



*Figure 10 -*

# Strengths

The group is proud of the design of the system. The system follows a distinct house style and colours, reserving important features where data loss may occur to the colour red.

Another strength is that the project is HTML validated. This system could be implemented into a larger system if necessary, provided the bugs are further tested.

The system makes use of SLIM and TWIG extensively. SLIM has significantly reduced the workload as it takes care of the routing within the system.

A key strength of the system is that it is functional. No immediate errors could be seen and only minor errors were seen when interacting with the system multiple times.

Regular meetings were carried out. All of the meetings had been successful, and as a whole, the group got along well. This was because of the variety of skills within the group.

Accounted for each others weaknesses. Documentation is thorough and complete and proofread. We found that having two people in charge of documentation has allowed us to provide a more varied documentation process.

# Weaknesses

Usernames are not encrypted within the username and some glitches occur, these coding glitches are displaying an incorrect message. According to the system, messages are only ever sent once and display an error on later attempts. Despite the error message, the messages do send every-time. Logging in and out prevents this problem but it is still an inconvenience. This problem was realised late into the project with no time to fix it which leads into the next weakness of the project, testing.

Testing has been a weakness of the project, many problems were revealed towards the end of the project. In larger scale projects where methodologies are used, testing methods would be considered very early on. For example, groups would decide if black box or white testing or use case testing would be the best way of analysing the system.

Originally, we had problems with beginning the project. Timetabling issues allowed two group members to convene and begin the groundwork of the assignment early on.

A time-plan would have helped the project significantly. Group members would be absent for long periods to catch up with other assignments. This is due in part to assignment deadlines on some modules having similar deadline dates.

An error in Git allowed a loss of work. Documentation for the testing is available at Github and on local files but did not make it into a larger system, the testing documentation that was lost due to merging errors underpinned the thought and consideration taken with the testing process.

# Summary and Conclusion

Overall, the project has been a success in several areas. We are able to download and save messages from an M2M server as well as create a login portal. During the project, the team has kept a careful eye on encryption, providing security features such as using encryption keys and password hashing where necessary.

Our system also extended beyond the initial specification, as login/logout features are available. With security in mind, we believe it was essential to include the system despite it not being mandatory for completion of the project.

Despite some minor visual glitches i.e passwords displaying as incorrect, the core functionality is still available. A larger problem is that username encryption is non-existent. We believe it would be more detrimental for passwords not to be hashed.

All data within the system has been sanitised and validated, password hashing in session files as well as the database to allow for protections against basic attacks such as DDOS and SQL injection attacks

From a managerial point of view, the project has run smoothly. Most of the coding was undertaken in a lab environment. This allowed for group members to understand how the system works as a whole.