# CAB302: Software Development

**Assignment 1: Electronic Billboard Management Software**

**Due Date: 31/05/2020**
**Weighting: 60%**
**Group Number: 60**

**Team Members:**

| Student ID | Student Name | Github Username | Git Commit Email |
|---|---|---|---|
| **n09763872** | Patrice Cotter | PCotter-dev | Patrice.cotter@connect.qut.edu.au |
| **n09800689** | Kanupriya Agarwal | KanuAgarwal | kanu10@live.com.au |
| **n09808876** | Yu Yi Alan Yu | yyay15 | yuyi.yu@connect.qut.edu.au |
| **n09954619** | Jacinta Roberts | JacintaRoberts | jacinta.roberts@connect.qut.edu.au |

## Contents

## 1 Statement of Completeness

The table below shows the statement of completeness on the features implemented in the electronic billboard management software.

| Criteria | Item | Done | Criteria | Item | Done |
|---|---|---|---|---|---|
| 1 | Source control – Git History | Yes | 14 | Message, Image, URL integration for billboard | Yes |
| 2 | Report Manual: Work distribution | Yes | 14 | Billboard Preview in Control Panel | Yes |
| 3 | Report Manual: high-level design description | Yes | 14 | Billboard stored in database | Yes |
| 4 | Report Manual: Setup instructions and software usage guide | Yes | 14 | Control Panel display schedules within week via calendar style, showing users too. | Yes |
| 5 | Java Doc Comments and generation of documents | | 15 | Control panel lists all users | Yes |
| 6 | Code Quality | Yes | 15 | Certain users can create new users with default salted/hashed password | Yes |
| 7 | Test-Driven Development | Yes | 15 | Certain users can edit user permission | Yes |
| 8 | Junit-5 test on non-GUI code | Yes | 15 | Certain users can edit user password | Yes |
| 9 | User Authentication process between Server and Control Panel | Yes | 15 | Certain users can delete users | Yes |
| 9 | User Credentials Stored in Database | Yes | 16 | Control panel communicate via network protocol via prop. File for address and port | Yes |
| 9 | Password Hashing and Salting | Yes | 16 | Server listens on port | Yes |
| 10 | Session Management generation and storage (Session Token) | Yes | 16 | Network is stateless connection | Yes |
| 10 | Session Management expiration and prompt (Session Token) | Yes | 16 | Connections only established by viewer and control panel | Yes |
| 11 | User permissions with protection | Yes | 17 | Server connection via JDBC | Yes |
| 12 | Billboard Display connect via network prop. file | Yes | 17 | Database stores billboards, Schedule, User info | Yes |
| 12 | Billboard Viewer displays billboards | Yes | 17 | Default user login created if not found in database | Yes |
| 12 | Billboard Viewer standby billboard | Yes | | | |
| 13 | Import Billboard via XML | Yes | | | |
| 14 | Schedule stored in Database | Yes | | | |
| 14 | Control Panel list billboard | Yes | | | |
| 14 | Control Panel Delete billboard | Yes | | | |
| 14 | Control Panel list billboard | Yes | | | |
| 14 | Control Panel Edit billboard | Yes | | | |
| 14 | Control Panel Create billboard | Yes | | | |

## 2 Statement of Contribution

The below table summarizes the contribution of each member of the team and describes the items they have completed for the project. The Agile and Software development aspects of the project will not be explicitly covered in the report. Instead, these may be found in the Appendices if the reader wishes to learn more about the development process and the UML diagram and the wireframe mock-up made.

| Student | Agile and Software Development | Software Implementation | Report |
|---|---|---|---|
| Patrice Cotter **(n09763872)** | • Wireframe development<br>• Trello | • Main GUI maintainer for Control Panel (Control, Views and models) | |
| Kanupriya Agarwal **(n09800689)** | • UML – Server<br>• UML – Database<br>• Trello | • Main Maintainer for Viewer Application GUI | |
| Yu Yi Alan Yu **(n09808876)** | • UML – Server<br>• UML – Database<br>• Data flow table for software<br>• Trello | • TDD Test – Control Panel<br>• Main Maintainer for MySQL Server Database<br>• Main Maintainer for Schedule Module for server and control panel<br>• Main Maintainer for Billboards Module for server and control panel | • Main Report maintainer |
| Jacinta Roberts **(n09954619)** | • UML – Control Panel<br>• UML – Viewer<br>• Software interaction exemplar<br>• Data flow table for software<br>• Trello | • TDD Test – Server and Networking<br>• TDD Test – Helper<br>• TDD Test – UserAdmin and Session Validation<br>• Main Maintainer for Users Module for server and control panel<br>• Unit Testing for Billboards for server<br>• Unit Testing for Users for server | |

# 3 Software Architecture

## 3.1 Software Hierarchy

The billboard application software was designed to be adaptable to changes. Methods and classes were created to ensure generalize the coding so that the code may be adapted and reused. One prime example is the use of the helper's module, where network interfacing methods are set so that the code may be reused in all three applications. This includes functions such as port validation, reading property files and serialization of objects for TCP/IP transmission.

For each application, a main top-level class exists as the main application. Multiple class files were created to group relevant functionality together at a lower level. This ensures proper hierarchy and code readability, and the ability to extend and reduce functionality according to specification. The hierarchy of the software classes for each application and a brief description of each may be seen as follows:

### 3.1.1 Server

Main server top level class application that contains the main loop. This will run any additional and relevant class files for functionality. Three main modules may be found, which consists the functionality of:

- Schedule
- Billboard
- User

Server Application Class

The server application class may be executed to initialize the server for the billboard application. All server functions will be called as appropriate as the server receives instruction and information from the control panel. The communication between the server main and control panel class is also serialized to allow objects to be sent to and from both applications.

Main Method Class File

- DB Connection: This class file contains all the code required for the server to read in, connect to and interact with the Database.
- ScheduleAdmin: The ScheduleAdmin class file contains code that will run anything related to the billboard schedules, from storing the information in the database and parsing them to the Control Panel Application. These will include server methods which will have checking logic for sessionToken, and permission checks relevant to each method. SQL specific methods are also stored in this file.
- UserAdmin: The UserAdmin class file contains code that will compute and process commands to do with User control. This includes setting user profiles in the database, and controls to retrieve and check user permissions and parsing back to the Control Panel Application. All methods in this class relies on DbUser, which class the appropriate SQL commands to communicate with the database. These will include server methods which will have checking logic for sessionToken, and permission checks relevant to each method.
- DbUser: The DbUser class file contains methods which directly communicates from server across to the database. These are parsed into UserAdmin methods.

- DbConnection: The DbConnection class contains methods which creates instances of connection to the database server. These are called upon for all SQL queries in the different modules.
- BillboardAdmin: The BillboardAdmin class file is responsible for any code that interacts with Billboards, such as storing billboards into the database and parsing them to the Control Panel Application. These will include server methods which will have checking logic for sessionToken, and permission checks relevant to each method. SQL specific methods are also stored in this file.

Main Object Class File

- DbBillboard: Db Billboard class is used when the server requires to get a billboard information from the database. It contains fields such as creator, billboardName, xmlcode, picture data and serverResponse.
- CpBillboard: The CpBillboard is a class created specifically for the ease of serialization when sending objects from Control Panel to server.
- BillboardList: BillboardList class contains two fields, an ArrayList field of BillboardName strings and a serverResponse. This class is mainly responsible for the list billboard command in BillboardAdmin. This object is serialized and is passed to the ControlPanel to the GUI to display active billboards in the database.
- CurrentSchedule: This class has fields which shows the billboardName, StartTime and CreationDateTime which will be imputed on the Server and sent to the Control Panel to be displayed onto the GUI when the user is viewing the current schedules of billboards.
- ScheduleList: This class will return all information about the schedule when the user requests it. This is used when the user is editing a schedule in the GUI, which the information prepopulates the field for the current schedule.

### 3.1.2 Control Panel

Main Control Panel top level class application that contains the main loop. This will initiate the GUI and run relevant class files for specific functionality. Similar to the server application, three main modules may be found for the functionality of Scheduling, Billboard and Users. However, additional class files are also found in the application which builds up the GUI interface using the Model, Controller, View system.

Main Control Panel Class

The main class for the control panel application allows the control panel to be started. This will then open up the GUI application, where the user will be able to interact with the server, if a server is running.

Main Modules for Control Functionality

- BillboardControl: The Billboard Control class controls all the methods required to send the user input from control panel about billboards (such as creation) to the server.
- ScheduleControl: The Schedule Control class contains code which allows the Control Panel to communicate anything schedule related to the billboard server.
- UserControl: The User Control class is responsible for any code that interacts with user controls and user list for the Control Panel and relays them to the server for processing.

Model, Controller and View Class

- AbstractView: Abstract View defines the generic style of the application, abstract methods in which child classes are required to extend and frame set up common to all views. Abstract View extends JFrame.
- AbstractGenericView: Abstract Generic View is designed to provide all generic functionality extended by all views excluding LogInView and BBFullPreview. Functionality includes a Profile, Log Out and Home Button embedded in a Navigation and Profile panel. Abstract View is extended to gain generic style and set up of the GUI frame.
- AbstractListView: Abstract List View is designed to create the main structure for displaying the User and Billboard data in a scrollable list. Child classes extend this structure and can customise for the specific data employed.
- AbstractUserView: Abstract User View is designed with the key features for displaying with User data including username, permissions and password. Child classes, including Profile, Edit User, Create User and View User inherit this common structure. Abstract Generic View is extended to gain generic functionality.
- BBCreateView: BB Create View is designed to allow users to create a Billboard by selecting the billboard name, title, text and picture. The billboard can be exported as an XML or an existing billboard XML can be imported. The billboard is displayed in the middle of the view, however a full view preview can be seen by selecting 'Preview Billboard' button.
- BBFullPreview: Preview the full Billboard from the BBCreateView or the BBListView, this shows the Billboard as it would be on the Viewer. This extends the Viewer class and overrides the exit functionality.
- BBListView: List of Billboards allowing user with valid permissions to Edit, Delete or View billboards. This extends the AbstractListView and is customised to display Billboard data.
- BBMenuView: Menu for navigating to the Billboard list or Billboard create/update view.
- Controller: Controller Class is designed to manage user inputs appropriately, sending requests to the Control Classes and updating the model and GUI views when required. The constructor stores a reference to the model and views and adds listeners to each GUI view.
- HomeView: Home view to allow users to navigate to all areas of the application including Users, Billboard and Schedule menus.
- LogInView: Log In View to allow users to log in with their username and password. An error message is displayed for incorrect credentials. LogIn View extends Abstract View.
- Main: Main starts up the application by instantiating the Model, Views and Controller. The main and AWT event thread are joined to run the application safely.
- Model: Model class contains all data that persists during a session. This includes the current username, session token and the current view seen by the user.
- ScheduleMenuView: Menu View to allow users to navigate to the schedule week calendar view or create/update schedule view.
- ScheduleUpdateView: Schedule Create View allows users to select an existing billboard and create or update the schedule. Users can set the start and end time, days scheduled and the recurrence setting (I.e. minutes, hourly, no repetition). Users can also clear the existing schedule.
- ScheduleWeekView: Schedule Week View to view the weekly schedule of billboards. This displays the billboard name and creator, including the start and end time.

- UserCreateView: User Create View is designed to allow users to create new User accounts. A unique username, password and permissions are required to create a new account. This extends the AbstractUserView to acquire key elements.
- UserEditView: User Edit View is designed to allow users to edit user accounts (assuming valid permissions). The username cannot be edited, only the permissions and password can be updated. This extends the AbstractUserView to acquire key elements.
- UserListView: User List View displays a list of all users in the database displaying buttons to Edit, Delete or View the user. This extends the AbstractListView and is customised to display User accounts.
- UserPreviewView: User Preview View displays a read only view of user information including the username and permissions selected.
- UserProfileView: User Profile View allows the current user to view personal information. This is a read-only view of the profile information and extends the AbstractUserView.
- UsersMenuView: Menu View to allow the users to navigate to the user list view create user view.

### 3.1.3 Viewer

The viewer was built with two main class files. There are no user inputs to the viewer application, except for closing the application via the ‘esc’ key or a mouse click.

Main Viewer Class

The main viewer class contains the main function which allows the viewer to be run. This class will call upon the Viewer class for GUI elements, and the Server class to get the latest current schedule to be displayed on the GUI.

Viewer Class

The Viewer class consists of GUI elements which are called upon on the Main Viewer class.

### 3.1.4 Helpers

Network protocol and generic methods may be found here. As this module contains various methods from different packages with no specific use-case, there are no subclasses created under helpers.

## 3.2 Test Driven Development

Test Driven Development (TDD) was actively used throughout for the development of the billboard management software, to ensure the code is up to specification and ensures a thorough implementation of the program. This may be reflected in the *.git* commit history attached in the submission. TDD was carried out for all the classes inside the ***server*** application. Within the server, additional tests were also laid out, namely ViewerSenderAdmin to test out interactions between the viewer and the billboard server. All classes whose methods encapsulated behind the ***control panel*** application and the helper class were also implemented with a TDD fashion. To ensure an unbiased-development, separate members of the team were assigned to either implement the initial test cases or implement the final software solution.

It should be noted that although test was written initially for the methods, these were written originally to not accept sessionToken to test out the raw functionality. As all methods in server will take in sessionToken and checks for this requires the active memory of the server, these tests were commented out due to the logistics of time for rewriting the entire test cases. Instead, Unit Testing was conducted for further verification, which is described later throughout the report.

The following tables details the tests carried out for each Application and application classes.

| Server Class | Tests | Server Class | Tests |
|---|---|---|---|
| **Server** | 1) Set up Server Object<br>2) Listen for connection (Pass)<br>3) Listen to Specific IP Connection (Pass)<br>4) Send Acknowledgement (Pass)<br>5) Login Response (Pass)<br>6) Login Response (Fail: Incorrect Password)<br>7) Login Response (Fail: Incorrect Username)<br>8) Logout Response (Pass)<br>9) Validate Session Token (Pass)<br>10) Validate Session Token (Fail: Fail Token) | **DB Connection** | 1) Create DB Connection Object<br>2) Read Database Properties File (Pass)<br>3) Read Database Properties File (Fail: File not found)<br>4) Creation of Tables (Pass)<br>5) User Table with correct columns (Pass)<br>6) Billboards Table with correct columns (Pass)<br>7) Schedules Table with correct columns (Pass)<br>8) Check creation of initial user (Pass)<br>9) Deletion of DB connection (Pass) |
| **UserAdmin** | 10) Construct User Admin Object<br>11) Check User exists (Pass)<br>12) Get Other User permission (Pass)<br>13) Get Own User permission (Pass)<br>14) Get Other User permission (Fail: Calling Username Deleted)<br>15) Get Other User permission (Fail: Insufficient User Permission)<br>16) Get Other User permission (Fail: Username does not exist)<br>17) List Users (Pass)<br>18) List Users (Fail: Calling Username Deleted)<br>19) List Users (Fail: Insufficient User Permission)<br>20) Set Own User Permission (Pass)<br>21) Set Own User Permission (Fail: Cannot remove edit user permission)<br>22) Set Own User Permission (Fail: Calling Username Deleted)<br>23) Set Own User Permission (Fail: Insufficient User Permission)<br>24) Set Other User Permission (Pass)<br>25) Set Other User Permission (Fail: Calling Username Deleted)<br>26) Set Other User Permission (Fail: Insufficient User permission) | **ScheduleAdmin** | 1) Create Schedule Admin Object<br>2) Declare Database Mock Object<br>3) Set Schedule Table Mock<br>4) Create Schedule in Schedule Table (Pass)<br>5) Create Schedule in Schedule Table (Fail: Billboard does not exist)<br>6) Create Schedule in Schedule Table (Fail: Insufficient User Permissions)<br>7) Create Schedule in Schedule Table (Fail: Duplicate Start Date Time)<br>8) Remove Schedule from Table (Pass)<br>9) Remove Schedule from Table (Fail: Insufficient User Permission)<br>10) Remove Schedule from Table (Fail: Billboard does not exist)<br>11) Remove Schedule from Table (Fail: Schedule does not exist)<br>12) View All schedules (Pass)<br>13) View All schedules (Fail: No Schedule Exist)<br>14) View All schedules (Fail: Billboard Does not exist)<br>15) View A Billboard Schedule (Pass)<br>16) View A Billboard Schedule (Fail: No Schedule Exist)<br>17) View A Billboard Schedule (Fail: Billboard Does not exist)<br>18) View A Billboard Schedule (Fail: Insufficient User Permission) |

| | | | |
|---|---|---|---|
| | 27) Set Other User Permission (Fail: Username does not exist)<br>28) Get Password (Pass)<br>29) Get Password (Fail: Username does not exist)<br>30) Set Own Password (Pass)<br>31) Set Own Password (Fail: Calling Username Deleted)<br>32) Set Other User Password (Pass)<br>33) Set Other User Password (Fail: Calling Username Deleted)<br>34) Set Other User Password (Fail: Insufficient User Permission)<br>35) Set Other User Password (Fail: Username Does not exist)<br>36) Delete User (Pass)<br>37) Delete User (Fail: Calling Username Deleted)<br>38) Delete User (Fail: Insufficient User Permission)<br>39) Delete User (Fail: Username Does not Exist)<br>40) Delete User (Fail: Cannot Delete Yourself)<br>41) Create User (Pass)<br>42) Create User (Fail: Calling Username Deleted)<br>43) Create User (Fail: Insufficient User Permission)<br>44) Create User (Fail: Username Already Taken) | | |
| **BillboardAdmin** | 1) Construct Billboard Admin Object<br>2) Declare Database Mock Object<br>3) Construct Database Mock object for billboard table<br>4) Create Billboard in billboard Table (Pass)<br>5) Create Billboard in billboard Table (Fail: Duplicate Name)<br>6) Create Billboard in billboard Table (Fail: Contain Illegal Characters)<br>7) Edit Billboard (Pass)<br>8) Edit Billboard (Fail: Insufficient User Permission)<br>9) Edit Billboard (Fail: Billboard Name Does Not Exist)<br>10) Delete Billboard (Pass)<br>11) Delete Billboard (Fail: Billboard Name does not exist)<br>12) Delete Billboard (Fail: Insufficient User Permission)<br>13) List Billboards (Pass)<br>14) List Billboards (Fail: Billboard Name Does not exist)<br>15) Get Billboard Information (Pass)<br>16) Get Billboard Information (Fail: Billboard Name does not exist)<br>17) Get Billboard Information (Fail: Insufficient User Permission) | **ViewerSender Admin** | 1) Construct Viewer Sender Object<br>2) Declare Database Mock Object<br>3) Construct Database Mock Object<br>4) Get Current Billboard from Schedule (Pass)<br>5) Get Current Billboard from Multiple Schedule (Pass)<br>6) Get Current Billboard from Schedule (Fail: No Schedules) |

*Table 1: Table of Test for TDD on Server Application Class*

| Control Panel Class | Tests | Control Panel Class | Tests |
|---|---|---|---|
| **BillboardControl** | 1) Create BillboardControl Object<br>2) Create Billboard (Pass)<br>3) Create Billboard (Fail: Name Already Exists)<br>4) Edit Billboard (Pass)<br>5) Edit Billboard (Fail: Insufficient User Permission)<br>6) Edit Billboard (Fail: Billboard Does not Exist)<br>7) Delete Billboard (Pass)<br>8) Delete Billboard (Fail: Billboard Does not Exist)<br>9) Delete Billboard (Insufficient User Permission)<br>10) List Billboard (Pass)<br>11) List Billboard (Fail: No Billboard Exists)<br>12) Get Billboard Information (Pass)<br>13) Get Billboard Information (Fail: Billboard Does not Exist)<br>14) Get Billboard Information (Fail: Insufficient User Permission) | **ControlPanel** | 1) Creation of ControlPanel Object<br>2) Listen for connection (Default IP)<br>3) Listen for connection (Specific IP)<br>4) Send Acknowledgement to Server<br>5) Login (Pass)<br>6) Login (Fail: Incorrect Password)<br>7) Login (Fail: User Does not Exists)<br>8) Logout (Pass) |
| **UserControl** | 1) Construct User Control Object<br>2) Log user out (Pass)<br>3) List Current Users (Pass)<br>4) List Current Users (Fail: Calling Username Deleted)<br>5) List Current Users (Fail: Insufficient User Permission)<br>6) Change Password Own (pass)<br>7) Change Password Own (Fail: Calling Username Deleted)<br>8) Change Password Other (Pass)<br>9) Change Password Other (Fail: Calling Username Deleted)<br>10) Change Password Other (Fail: Insufficient User Permission)<br>11) Change Password Other (Fail: User Does not Exist)<br>12) Set Own User Permission (Pass)<br>13) Set Own User Permission (Fail: Cannot Remove own edit users)<br>14) Set Own User Permission (Fail: Calling Username Deleted)<br>15) Set Other User Permission (Fail: Insufficient User Permission)<br>16) Set Other User Permission (Pass)<br>17) Set Other User Permission (Fail: Calling Username Deleted)<br>18) Set Other User Permission (Fail: User Does Not Exist)<br>19) Get User Permission (Pass)<br>20) Get User Permission (Fail: Calling Username Deleted)<br>21) Get User Permission (Fail: Insufficient Permission)<br>22) Get User Permission (Fail: Username Does not exist)<br>23) Delete User (Pass)<br>24) Delete User (Fail: Calling Username Deleted)<br>25) Delete User (Fail: Insufficient User Permission)<br>26) Delete User (Fail: User Does not exist)<br>27) Delete User (Fail: Cannot Delete Yourself)<br>28) Create New Users (Pass)<br>29) Create New Users (Fail: Calling Username Deleted)<br>30) Create New Users (Fail: Insufficient User Permissions)<br>31) Create New Users (Fail: Username Already Taken) | **ScheduleControl** | 1) Create ScheduleControl Object<br>2) Update Billboard Schedule (Pass)<br>3) Update Billboard Schedule (Fail: Billboard does not exist)<br>4) Update Billboard Schedule (Fail: Invalid Date Time Format)<br>5) Update Billboard Schedule (Fail: Insufficient User Permission)<br>6) Create Schedule (Fail: Repeats Exceed Duration - SS)<br>7) Create Schedule (Fail: Repeats Exceed Duration - HH)<br>8) Create Schedule (Fail: Repeats Exceed Duration - DD)<br>9) Create Schedule (Fail: Duplicate Start Time)<br>10) Remove Billboard Schedule (Pass)<br>11) Remove Billboard Schedule (Fail: Insufficient User Permission)<br>12) Remove Billboard Schedule (Fail: Billboard does not exists)<br>13) Remove Billboard Schedule (Fail: Billboard Schedule does not exist)<br>14) View Billboard Schedules (Pass)<br>15) View Billboard Schedules (Fail: No Billboard)<br>16) View Billboard Schedules (Fail: Insufficient Permission)<br>17) View A Billboard Schedule (Pass)<br>18) View A Billboard Schedule (Fail: No Schedule Found)<br>19) View A Billboard Schedule (Fail: Billboard Does not exist) |

*Table 2: Table of Test for TDD on Control Panel Application Class*

| Helpers Class | Tests |
|---|---|
| **Helpers** | 1) Construct Helpers Object<br>2) Read Network Properties (Pass)<br>3) Validate Network Port (Pass)<br>4) Validate Network Port (Fail: Bad Port Number)<br>5) Validate Network Port (Fail: Non-existent File Name) |

*Table 3: Table of Test for TDD on Helper Class*

## 3.3 Unit Testing

To further verify the implementation of the software solution, unit testing via Junit5 was conducted on server methods. The unit tests may be found in "CAB302-Billboard/src/test/java/server/UnitTest.java". Specifically, the methods being tested is contained within the following files:

- MockSessionTokens.java
- MockUserTable.java
- MockBillboardTable.java
- MockScheduleTable.java

The original implementation of the code found in their respective method classes was extrapolated without the SQL methods. These SQL methods were replaced by mock methods which allows the testing of the actual software implementation instead of testing the usability of SQL methods. Items that were covered for unit testing includes:

1) Session Token Functionality
    a. Generate SessionToken
    b. Get Username via SessionToken
    c. Check Permission via SessionToken
2) User Based Functionality
    a. Check User Exists
    b. Create User
    c. Create User - PrimaryKeyClashException
    d. Create User - InsufficientPermissionException
    e. Retrieve User
    f. Get Own Permission
    g. Get Other Permission
    h. Get Other Permission – InsufficientPermissionException
    i. Delete User
    j. Delete User – CannotDeleteSelfException
    k. Delete User - InsufficientPermissionException
    l. Delete User – NoSuchUserException
    m. List Users
    n. List Users – InsufficientPermissionException
    o. Set Permissions
    p. Set Permissions - InsufficientPermissionException
    q. Set Permissions - CannotRemoveOwnAdminPermissionException
    r. Set Permissions - NoSuchUserException
    s. Set Password

    t. Set Password – InsufficientPermissionException
    u. Set Password - NoSuchUserException

3) Billboard Based Functionality
    a. Create Billboard
    b. Create Billboard - PrimaryKeyClashException
    c. Create Billboard - InsufficientPermissionException
    d. Delete Billboard
    e. Delete Billboard - BillboardNotExistsException
    f. Delete Billboard - InsufficientPermission
    g. Get Billboard Information
    h. Get Billboard Information – BillboardNotExistsException
    i. List Billboard
4) Schedule Based Functionality
    a. Update Schedule (creation)
    b. Update Schedule (creation) - BillboardNotExistsException
    c. Update Schedule (creation) - InsufficientPermissionException
    d. Update Schedule (editing)
    e. Delete Schedule
    f. Delete Schedule – BillboardNotExistsException
    g. Delete Schedule – ScheduleNotExistsException
    h. Delete Schedule – InsufficientPermissionException
    i. Get Schedule Information
    j. Get Schedule Information - ScheduleNotExistsException
    k. Get Schedule Information – InsufficientPermissionException

# 4 Software Usage Guide

The electronic billboard management software was developed to ensure ease of navigation of the software. This section will detail the relevant packages and software for the solution, steps required to setup the software and a brief usage guide.

## 4.1 Software Requirements

To ensure the program running successfully, please ensure the correct versions of the following software are installed. Other versions may be compatible; however, they are not tested and may bring unexpected errors when running the application.

- The program is written in JDK13, with Junit5 testing.
- MariaDB was used as the main database server, with the version 10.4.12 x64.
- A MariaDB Connector for Java was used and is contained within the software program Classpath. The version of the driver is version 2.6.0.

Testing and development of the software was conducted on windows 10 64-bit machines.

## 4.2 Software Setup

The following section will detail the steps required to configure the network properties file and default user setting to ensure the software runs correctly with the billboard viewer and billboard control panel connection to the correct billboard server.

### 4.2.1 Network and Database Properties file

A networks properties file named *"name"*, may be preconfigured to setup the listening address for the both billboard viewer and control panel.

The default network properties file format and the default entry used for this report is shown as follows:

```
// ControlPanel/Viewer to Server Properties
port = 4444
ip = 127.0.0.1
```

*Figure 1: Configuration of Network Properties File*

Database access is also setup in a similar manner, with a db.props file that can be formatted to set database connection and default user setting. The default database properties file format and details that are used for this report is shown as follows:

```
// Server to DB Properties
jdbc.url=
jdbc.schema=
jdbc.username=
jdbc.password=
```

*Figure 2: Configuration of Server DB Properties File*

## 4.3 Software Usage

The software was developed into three main software packages: Billboard Viewer, Billboard Server and the Billboard Control Panel. The respective usage guide may be seen in the following sections.

### 4.3.1 Initializing and Running the Database

A database is required to run the program, and store the user, billboard, and schedule information. To do so, please install MariaDB files and relevant software packages listed in the previous section. If the program is installed correctly, the database server can be initiated by running the following code in the *bin* folder of MariaDB:

```
mysqld --console
```

*Figure 3: Command for running MariaDb database*

- Note: For windows Power-shell users, you will need to add "./" in front to run the server.

If the server runs successfully, the following message may be seen in the console. Please ensure that the port number matches up the port number specified in the *db.props* file.

```
C:\mariadb-10.4.12-winx64\bin\mysqld.exe: ready for connections.
Version: '10.4.12-MariaDB'  socket: ''  port: 3306  mariadb.org binary distribution
```

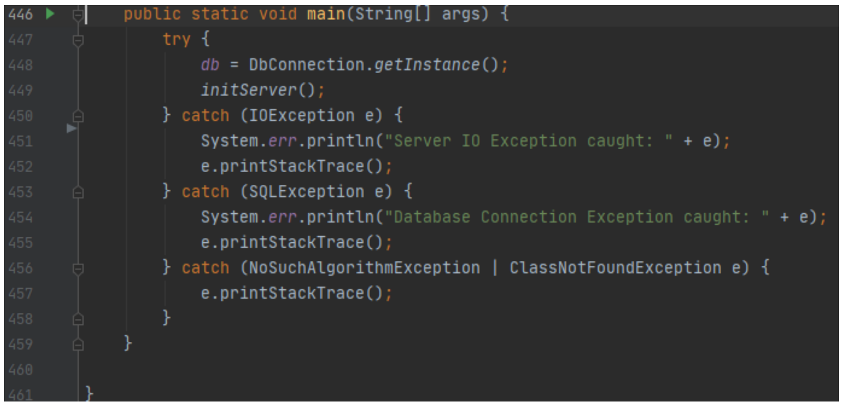
*Figure 4: Success message of running database server*

### 4.3.2 Billboard Server

The billboard server was designed to be a console application which prints out server interactions and processes. The server has no user inputs, with the only exception of running and closing the server. The server must be run first to both initialize a server which the viewer and control panel can connect to, and also creating the database table if it does not exist within the MariaDB database.

#### *4.3.2.1 Server: Running Billboard Server*

To run the billboard server, the user needs to navigate to "CAB302-Billboard/src/main/java/server/Server.java". In this file, the server application may be run by running the play button (Green Triangle) at around line 446 which will start the server.

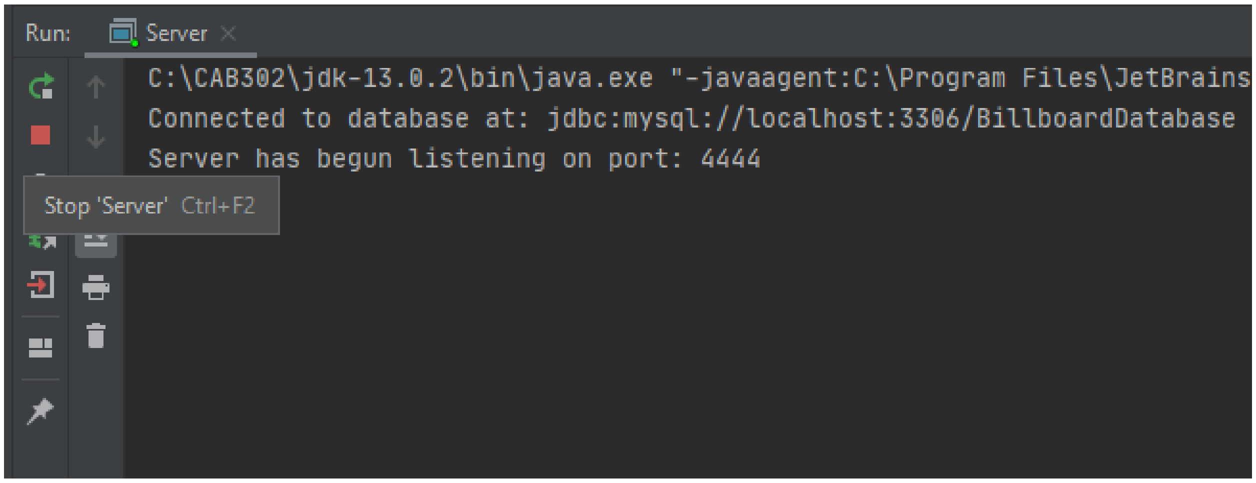


```
446     public static void main(String[] args) {
447         try {
448             db = DbConnection.getInstance();
449             initServer();
450         } catch (IOException e) {
451             System.err.println("Server IO Exception caught: " + e);
452             e.printStackTrace();
453         } catch (SQLException e) {
454             System.err.println("Database Connection Exception caught: " + e);
455             e.printStackTrace();
456         } catch (NoSuchAlgorithmException | ClassNotFoundException e) {
457             e.printStackTrace();
458         }
459     }
460
461 }
```

*Figure 5: Running Server Application*

#### *4.3.2.2 Server: Closing Billboard Server*

To close the billboard server, the user will be able to do so in the console by either pressing the red square, or press cntrl+f2 to close the server.



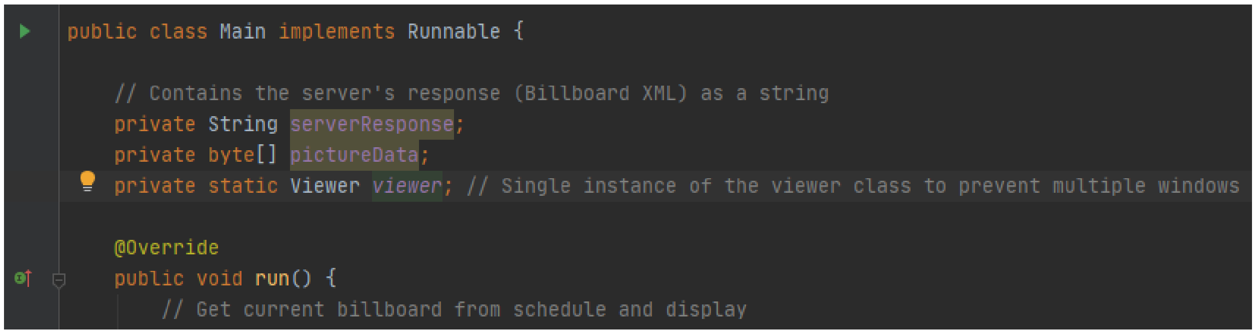
```
C:\CAB302\jdk-13.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains
Connected to database at: jdbc:mysql://localhost:3306/BillboardDatabase
Server has begun listening on port: 4444
```

*Figure 6: Closing Billboard Server Application*

Additionally, if the server exception is raised, the server will also printStack the exception and close.

### 4.3.3 Billboard Viewer

The Billboard viewer was designed to have no user inputs. Once the program is executed, a viewer frame will be created. The program may be executed by navigating to "CAB302-Billboard/src/main/java/viewer/Main.java". In this file, the viewer application may be run by running the play button (Green Triangle) at around line 12 which will start the viewer application.



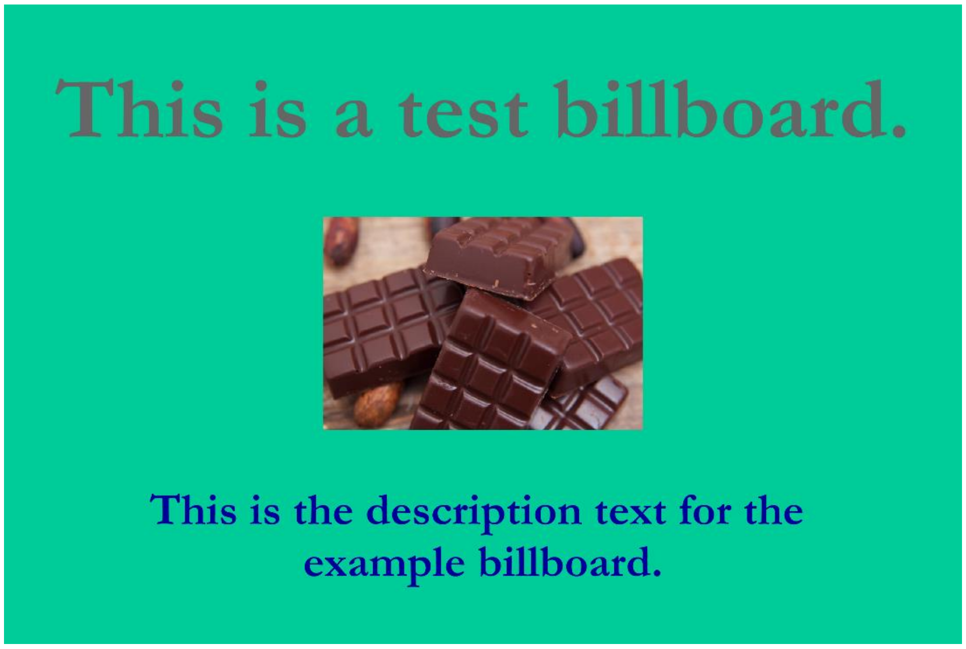
```
public class Main implements Runnable {

    // Contains the server's response (Billboard XML) as a string
    private String serverResponse;
    private byte[] pictureData;
    private static Viewer viewer; // Single instance of the viewer class to prevent multiple windows

    @Override
    public void run() {
        // Get current billboard from schedule and display
```

*Figure 7: Executing Viewer Application*

Once the program is executed, the program will automatically connect to the preconfigured billboard server. For every 15 seconds, the viewer billboard will refresh to the next scheduled billboard.

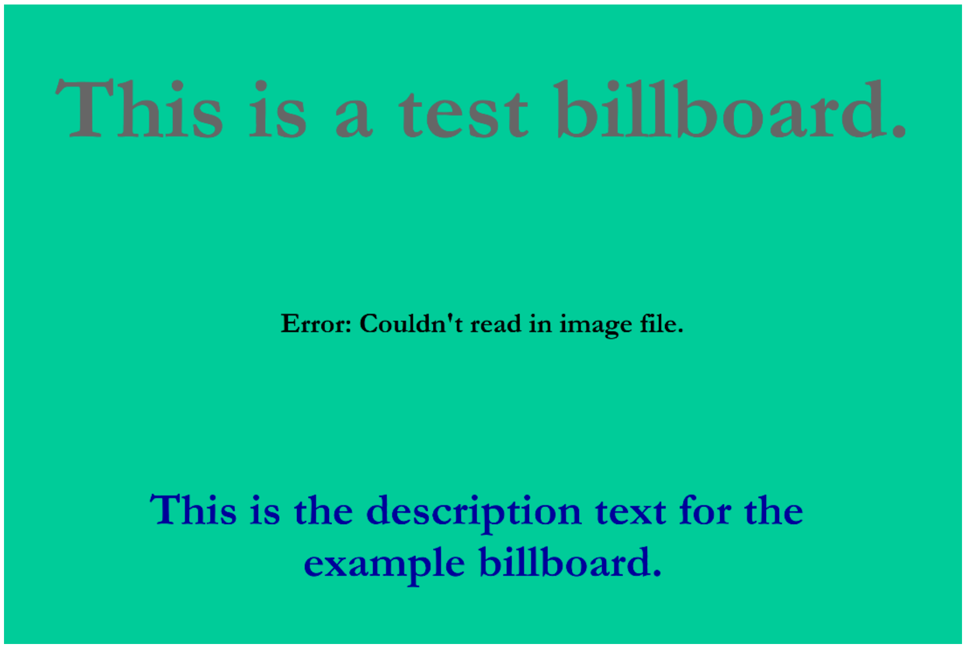


This is a test billboard.

This is the description text for the example billboard.

*Figure 8: Viewer Display with Billboard*

If a billboard is scheduled, but there is an error reading the picture, the viewer will display the other elements of the billboard and an error message in place of the picture.



This is a test billboard.

Error: Couldn't read in image file.

This is the description text for the example billboard.

*Figure 9: Viewer Display with Picture Error*

If a billboard is scheduled, but there is an error reading in the XML, the viewer will display an error message.

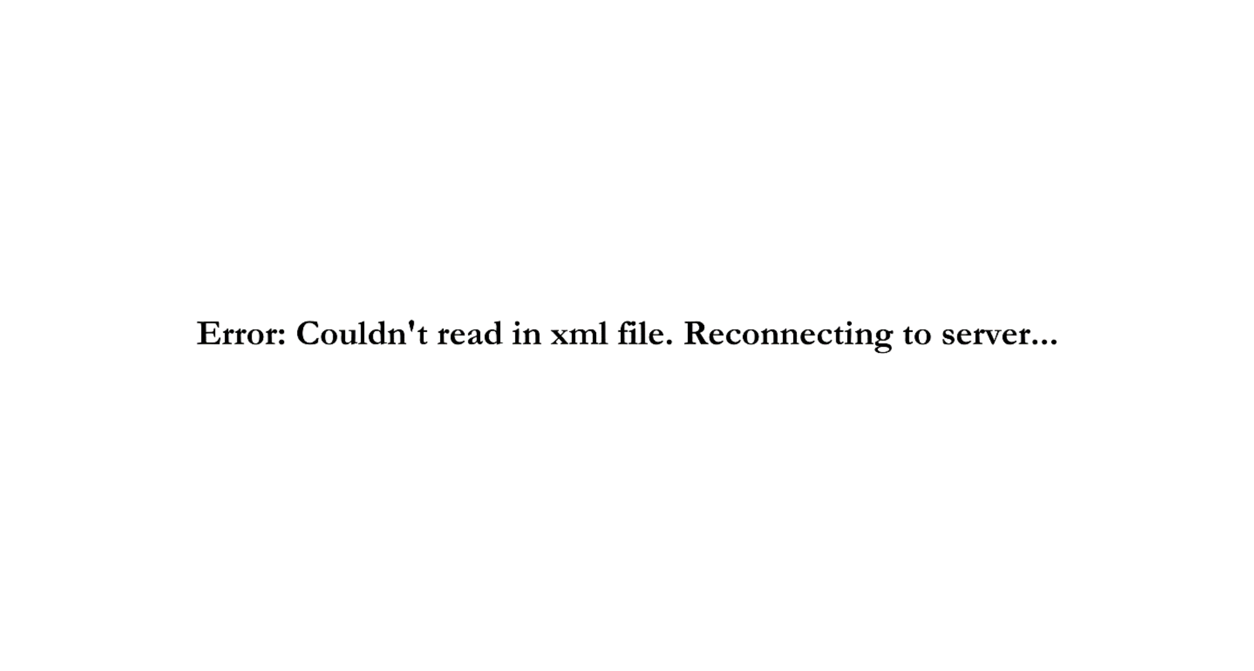

Error: Couldn't read in xml file. Reconnecting to server...

*Figure 10: Viewer Display with XML Error*

If no billboard is scheduled, a default screen with a message will be shown.

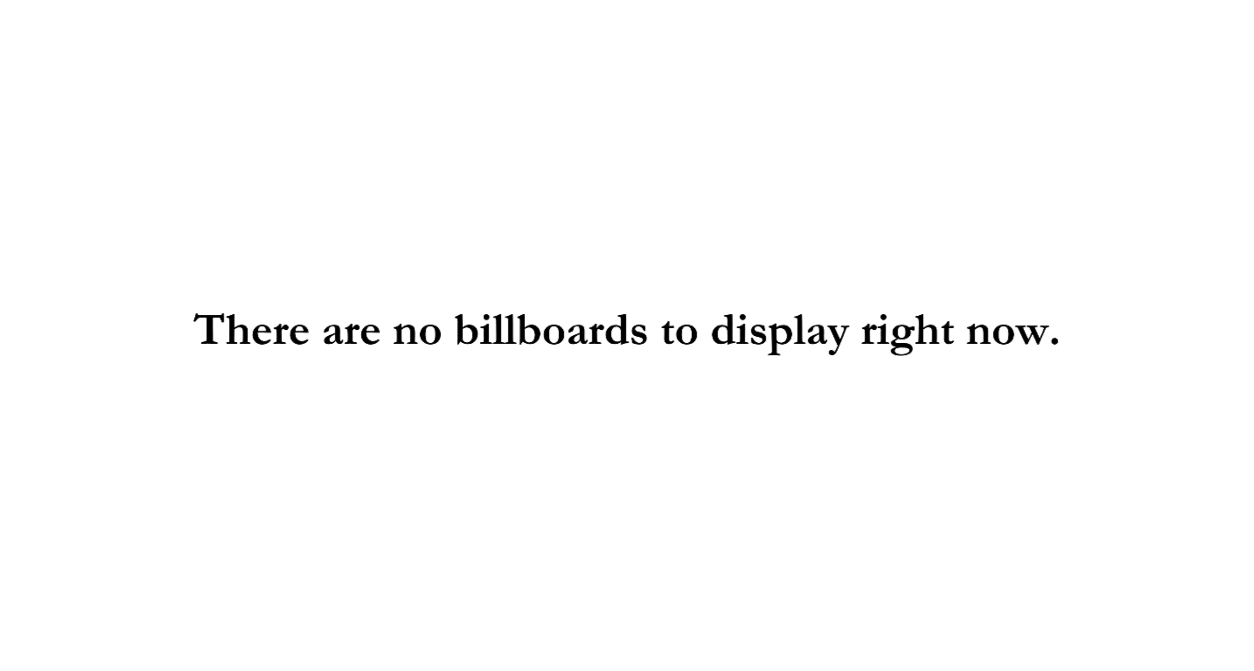

There are no billboards to display right now.

*Figure 11: Viewer Display with No Billboard*

If the viewer can't connect to the preconfigured billboard server, it will show an error message.

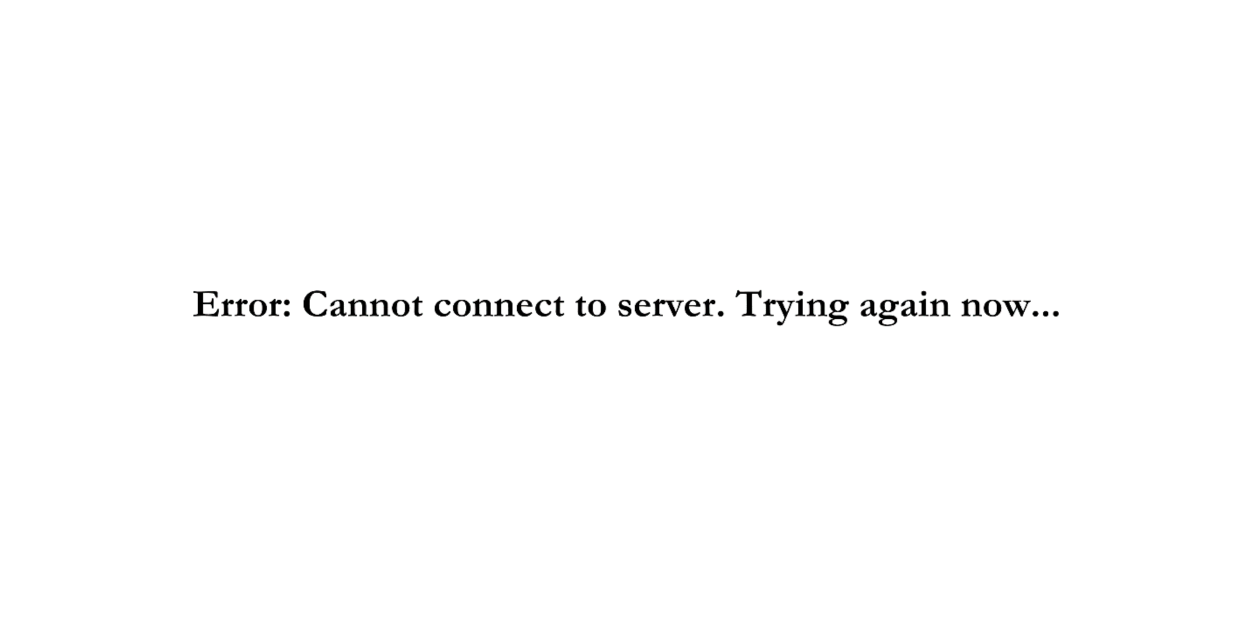

Error: Cannot connect to server. Trying again now...

Figure 12: Viewer Display with Server Connection Error

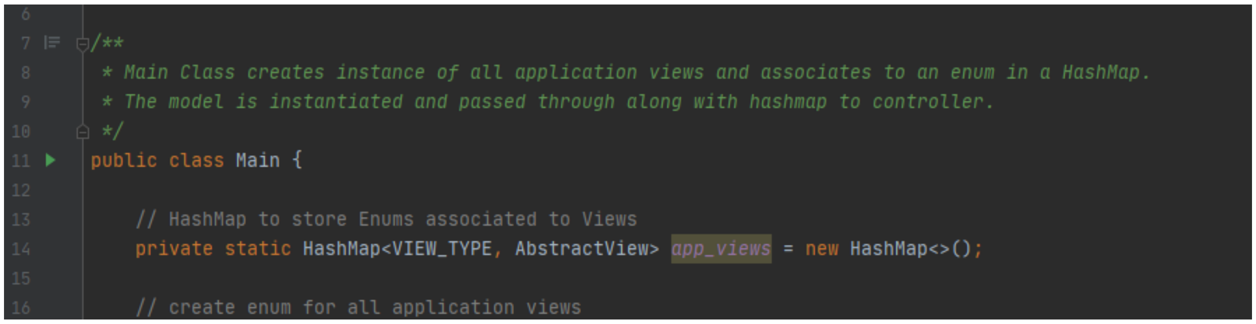
The user can close the billboard via either clicking on the Viewer screen or pressing the escape key.

### 4.3.4 Control Panel

The control Panel is the third application that is implemented for the software solution. It provides an interface for the user to interact with the server and the database, to both set up and edit billboard display data and scheduling for the billboard viewer application. Only registered users may be logged into the control panel applications. New users may be created by preexisting users with suffice permission. It should be noted that users have certain permission values, and if the requirements are not met, features will not show up on the GUI. However, for every action the user takes, a permission check will be ran to ensure that any changes within user permissions within the session is reflected upon the users requested action.

#### *4.3.4.1 Starting up the Control Panel*

The program may be executed by navigating to "CAB302-Billboard/src/main/java/controlPanel/Main.java". In this file, the control panel application may be run by running the play button (Green Triangle) at around line 11 which will start the viewer application.



```
6
7   /**
8    * Main Class creates instance of all application views and associates to an enum in a HashMap.
9    * The model is instantiated and passed through along with hashmap to controller.
10   */
11  public class Main {
12
13      // HashMap to store Enums associated to Views
14      private static HashMap<VIEW_TYPE, AbstractView> app_views = new HashMap<>();
15
16      // create enum for all application views
```

*Figure 13: Executing Control Panel Application*

#### *4.3.4.2 General User Logon interaction*

*Control Panel Login Interface*

As the control panel starts, the user is first greeted with the login screen as seen in Figure 14. The user can type in the default user details (Username: root, pass), or if a user was created in previous sessions or otherwise preconfigured, the user may login through their own username password.

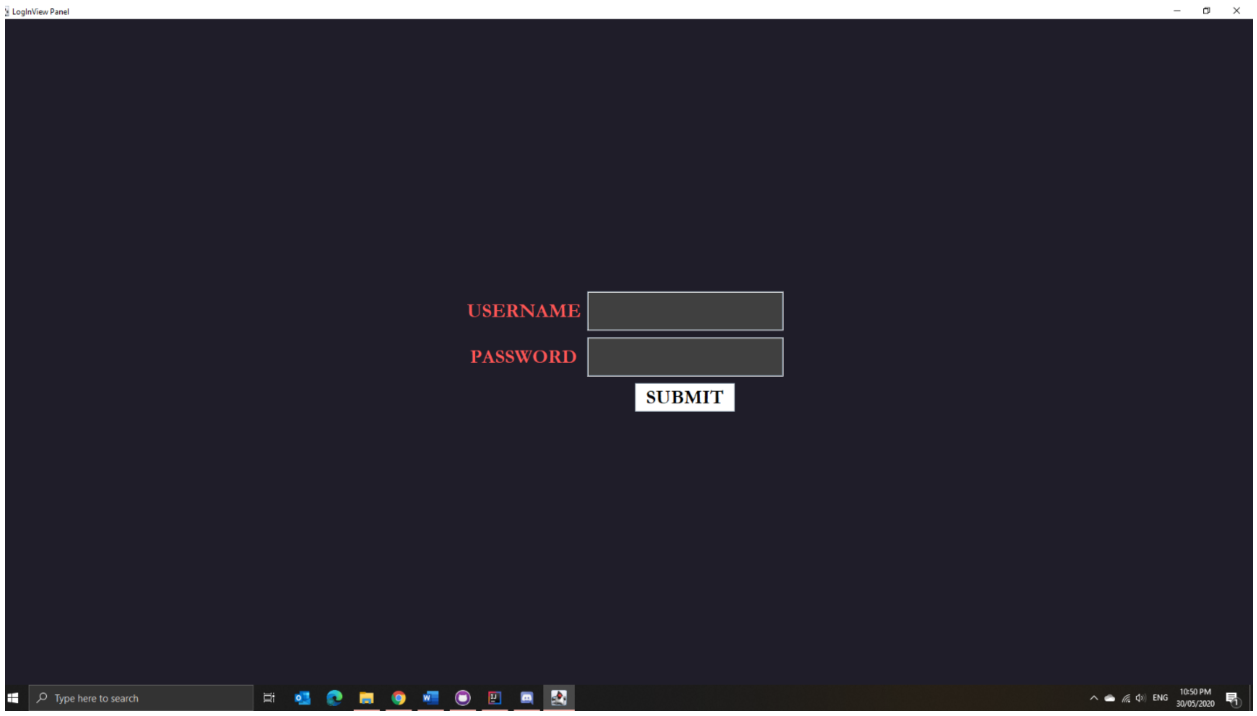


*Figure 14: User Login for Control Panel*

*<u>Successful Login</u>*

If the login was successful, the user will be redirected to the home page where they can then direct to other services. This landing page may be seen as follows Figure 15

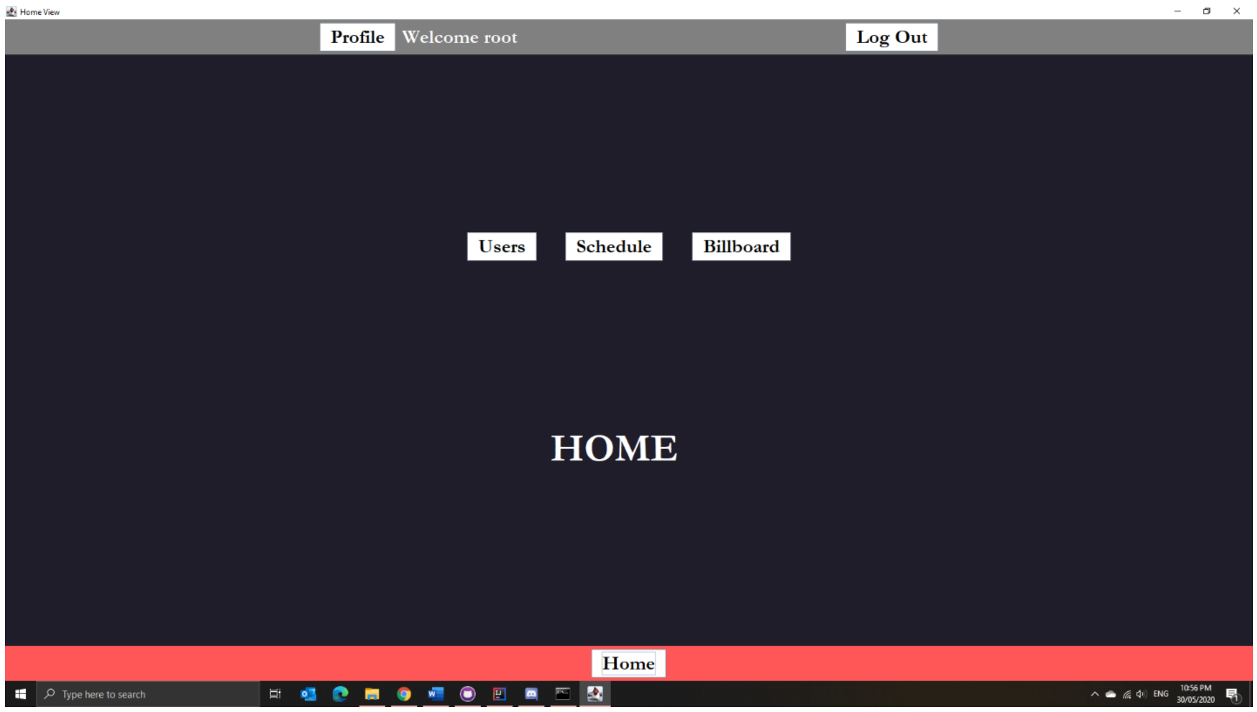
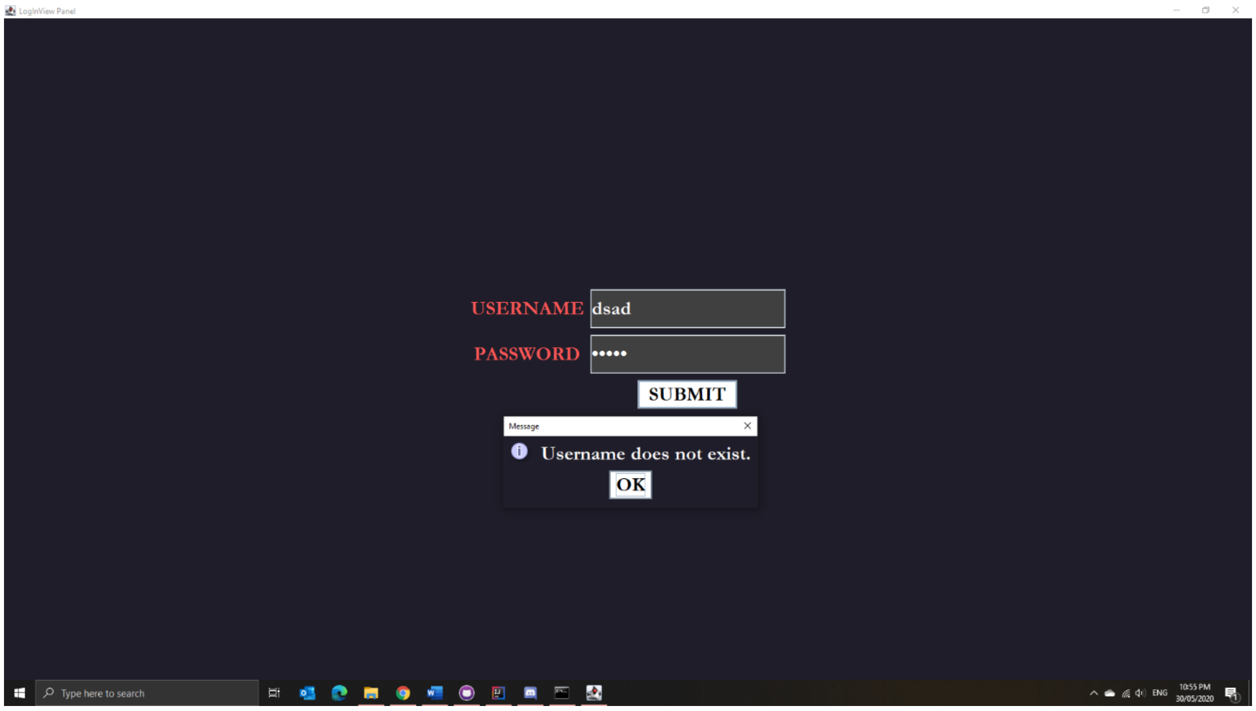


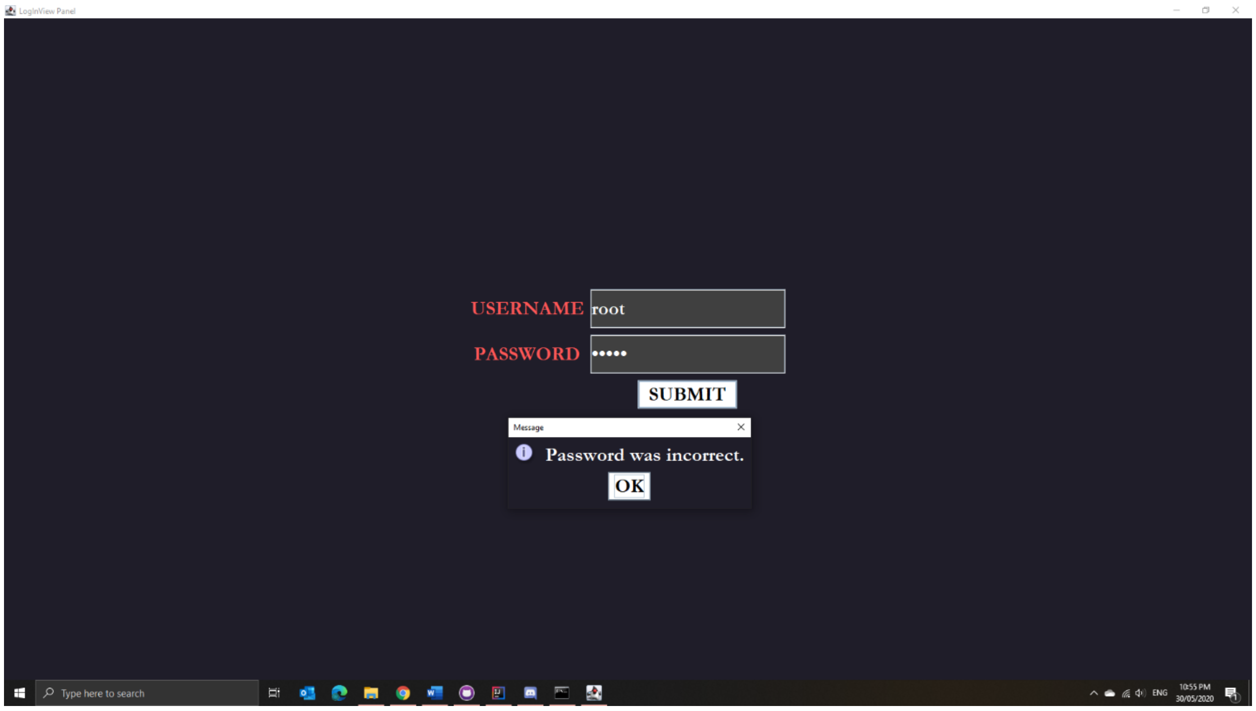
*Figure 15: Successful Login Page Screen*

## *Unsuccessful Login*

If the login was unsuccessful due to incorrect user details (such as wrong passcode or username), then an error message will be displayed on screen, which may be seen in Figure 16.



Username does not exist.



Password was incorrect.

*Figure 16: Incorrect Details login prompt*

*Expired Session Token*

For every interaction, a user requests from the control panel, a validation of session token is conducted. If the user session token exceeds the 24-hour validation period, the user will be taken back to the user login screen to reauthenticate. Once this is done, the login process is the same as it would be, and the user will receive a new session token which expires in 24 hours.

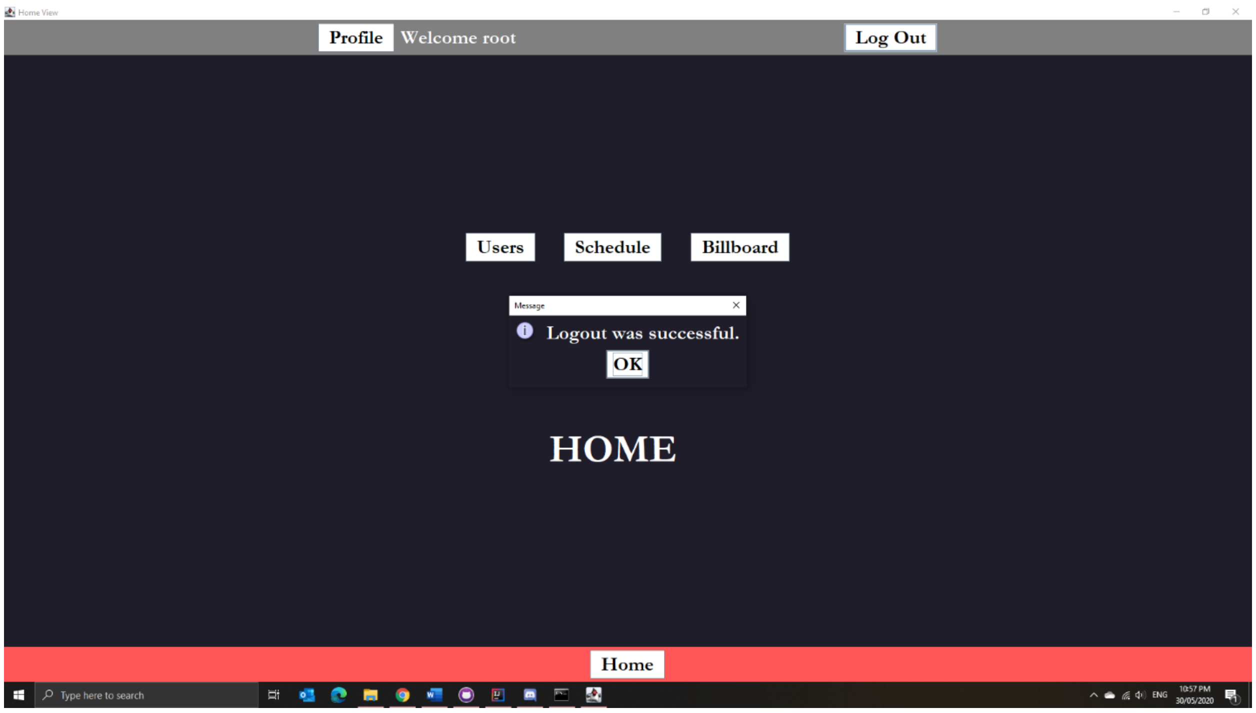
*Condition of User changed during Interaction*

Given a scenario where the user has permission to interact with a specific feature (such as editing a billboard, or editing user permission), assuming that after the user clicks the  interactive button on the Control Panel to request the server to complete the method, and that the user account was deleted midway during the request, a "*Calling Username Deleted"* Error will pop up. This signifies that before the request is completed, the username has been deleted. This will bring the user back to the login screen.

Similarly, if a user edit permission for billboards was altered whilst the user is updating a billboard, the request will not proceed, with the user being brought back to the home screen of the control panel application

*User Logout*

The user can logout from the application from the Control Panel home screen through the dedicated logout button as seen in Figure 17. Once the system receives the user request to log out, the session token will be expired, and the user will be sent back to the user login screen.



Logout was successful.

*Figure 17: Dedicated User Logout Button*

### 4.3.4.3 User Control Interface

<u>*User Control Interface*</u>

Once the user is logged in and authenticated, they may select the users button to interact with the user list in the database if they have permission (Figure 18). All users will be able to list users stored in the database, where they can view user accounts and with suffice permission, be allowed to edit and delete permissions. This may be done via the *View Users* button, which will send a query to the server to pass onto the database. If there are existing users, the user list may return, which may be seen in Figure 19

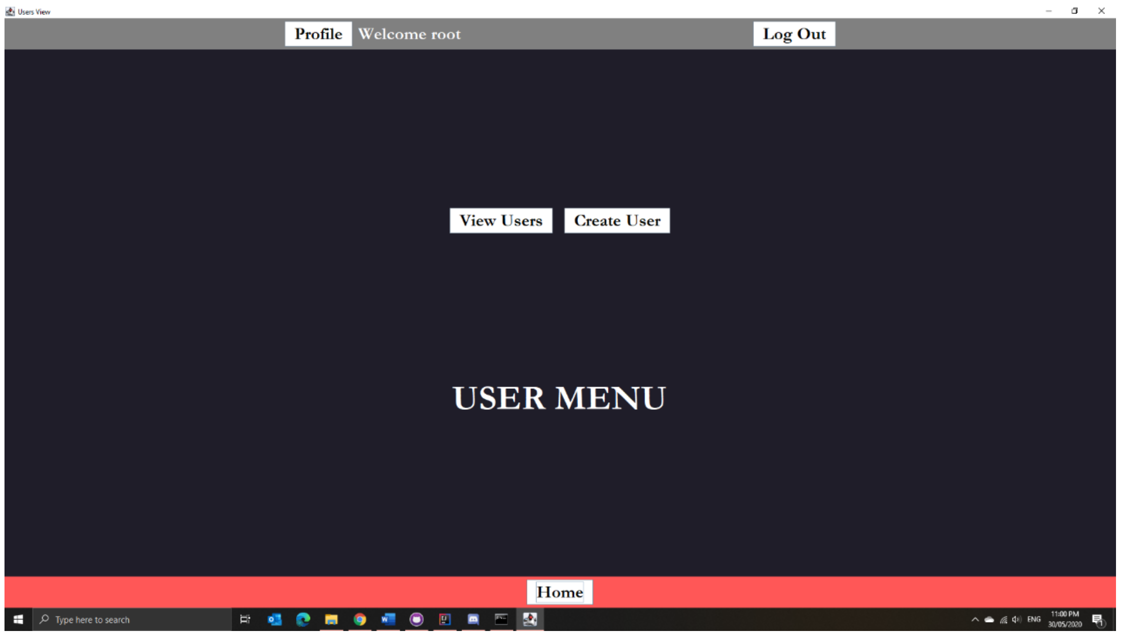


Figure 18: User Menu

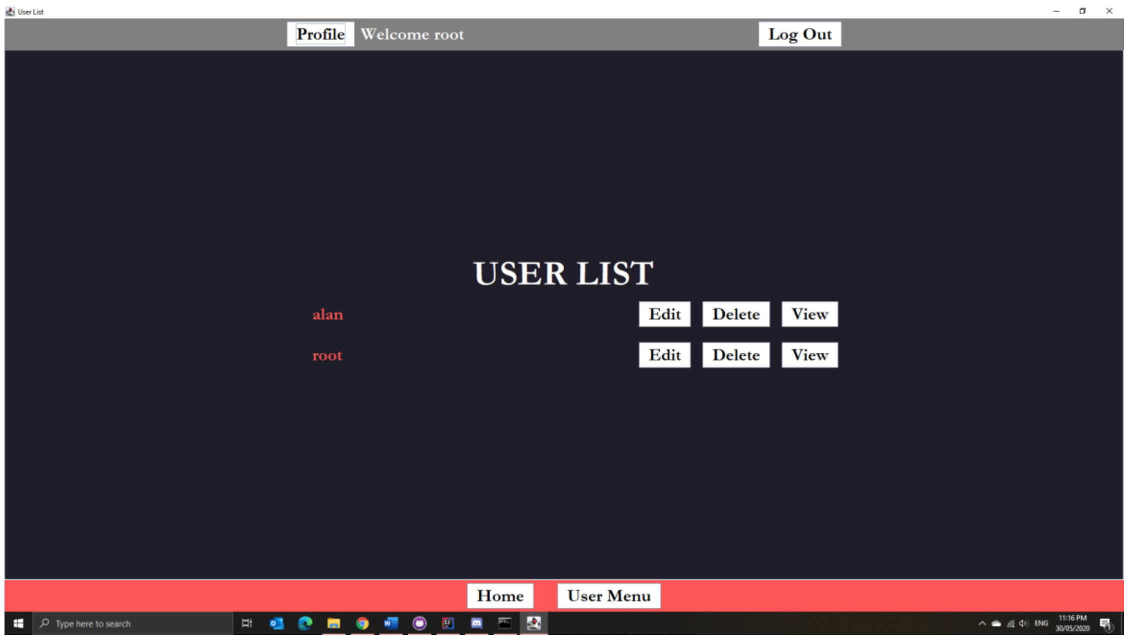
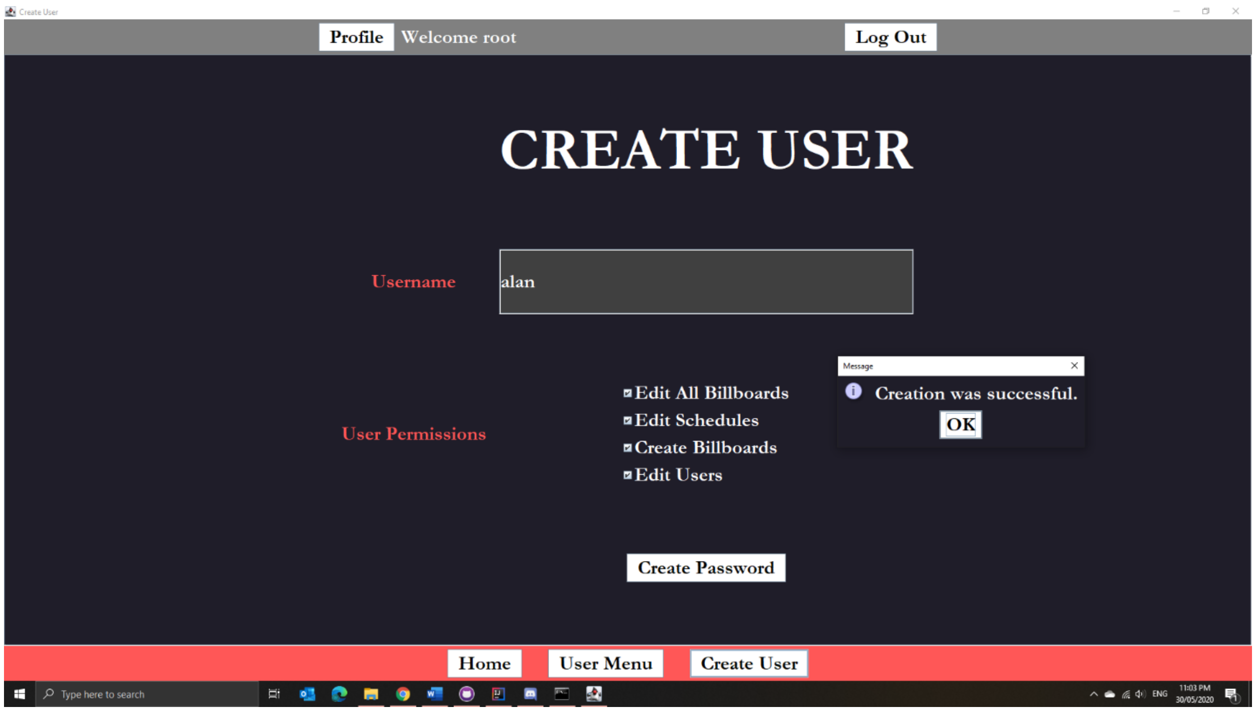


Figure 19: User List

*Create New Users*

Within the *Users* screen, users that has the *edit users* permission enabled allows them to create new users that can be used to log into the control panel. To do so, the *add user* button may be pressed which takes the user to a user creation screen as seen in Figure 20.



Creation was successful.

*Figure 20: Creation of a new user*

From this screen, the user may input a new username to be stored into the database. The user will also get the option to create a password and set the permission level of the new user.

## *Edit Own User Permission*

For each user, they will be able to change their own permission within the control panel application. User permissions may be changed via a toggle on the interface and will be saved to the database once the user confirms the changes.

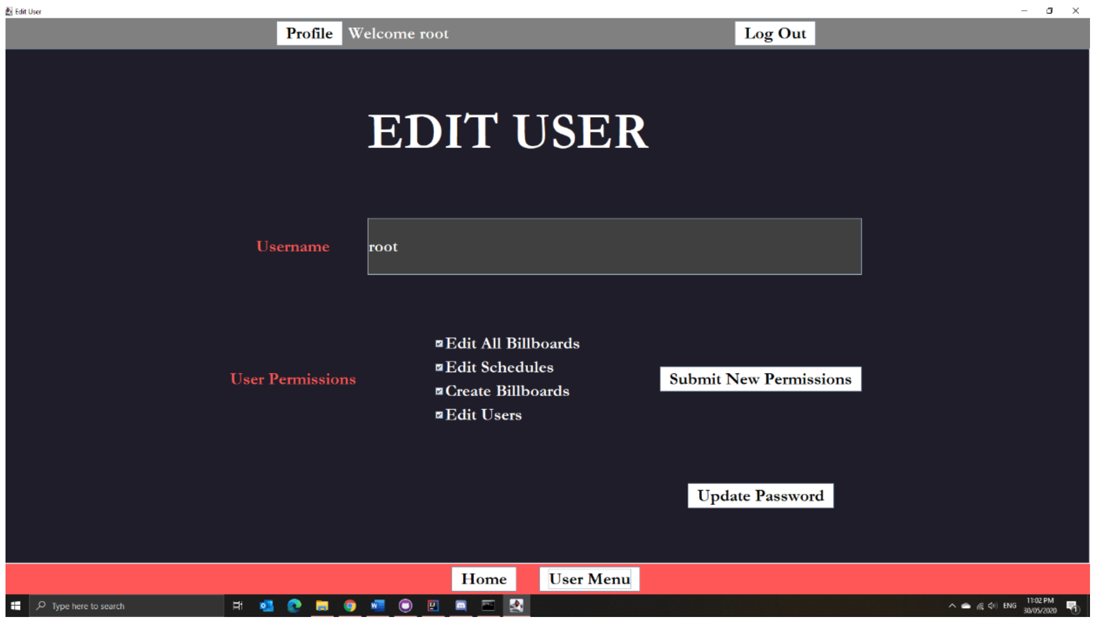
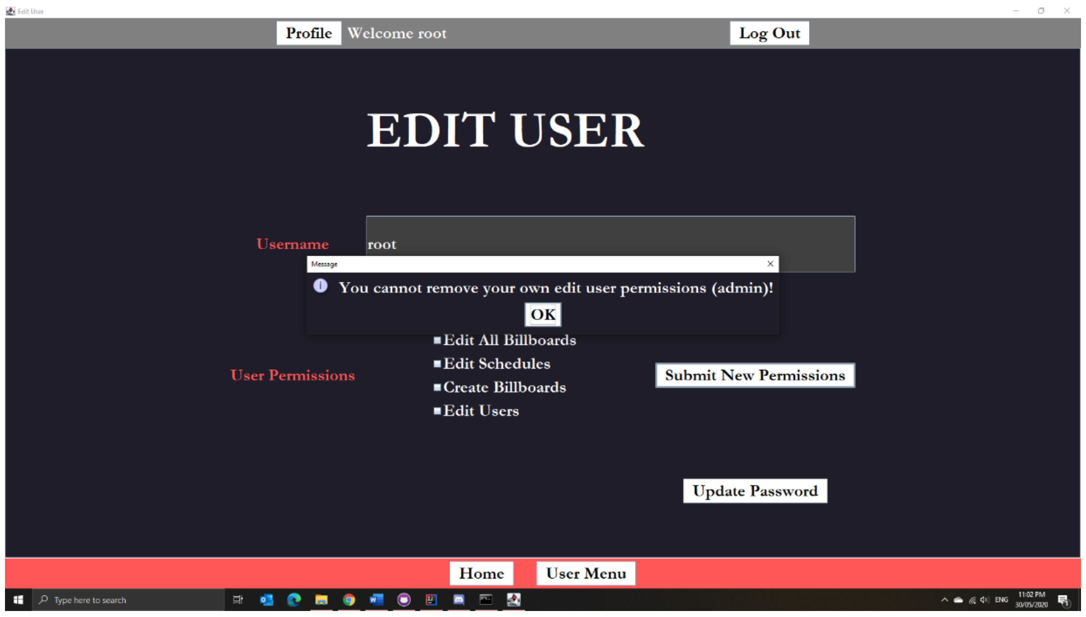


*Figure 21: Screen of editing user permission and password*

It should be noted that the edit user permission is enabled by default for registered administrators Specific user permissions such as edit other users may not be toggled by yourself to prevent undesired scenarios, and must be edited by another user / administrator.

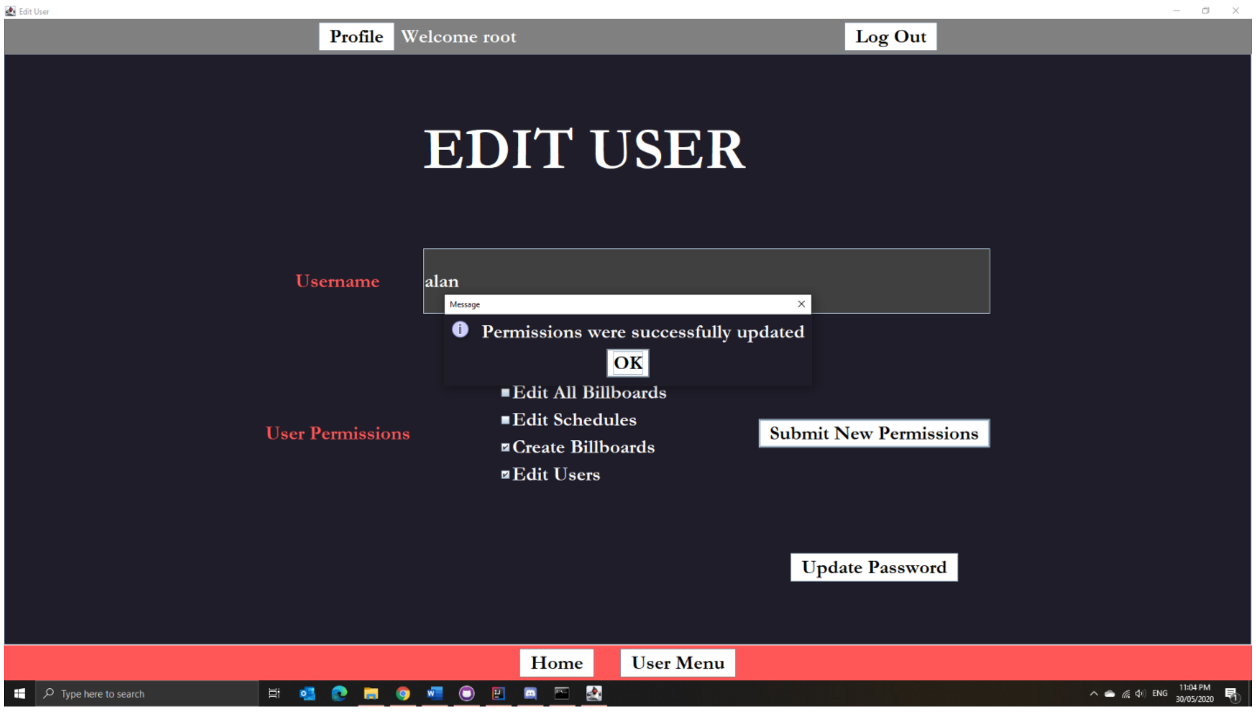


You cannot remove your own edit user permissions (admin)!

*Figure 22: Error when removing own Edit Users Permission*

### *Edit Other User Permission*

Within the *Users* screen, users with the edit user's permission enabled allows them to also edit other user permissions. This permission value is enabled for administrators by default. To edit other user's permission, the administrator or the user with edit user privilege will first need to go to the user list. From there, a specific user account can be selected and a button to request for edit user permission can be interacted with. Once the button is pressed, a request will be sent to the server to check if the user has permission to edit. Once verified, the user will be able to edit the other user's permission with a similar interface of editing their own user permission.


Edit User

Welcome root

EDIT USER

Username: alan

Message: Permissions were successfully updated

User Permissions:
- Edit All Billboards
- Edit Schedules
- ☑ Create Billboards
- ☑ Edit Users

Figure 23: Edit other user permission

## *Edit Own User Password*

Within the users interface, the user will be able to view their own profile and edit their own user passcode. This can be done by inputting the original password once and the new desired password twice. The server will verify the response and update the database. The user will then be required to use this passcode in future subsequent login sessions.

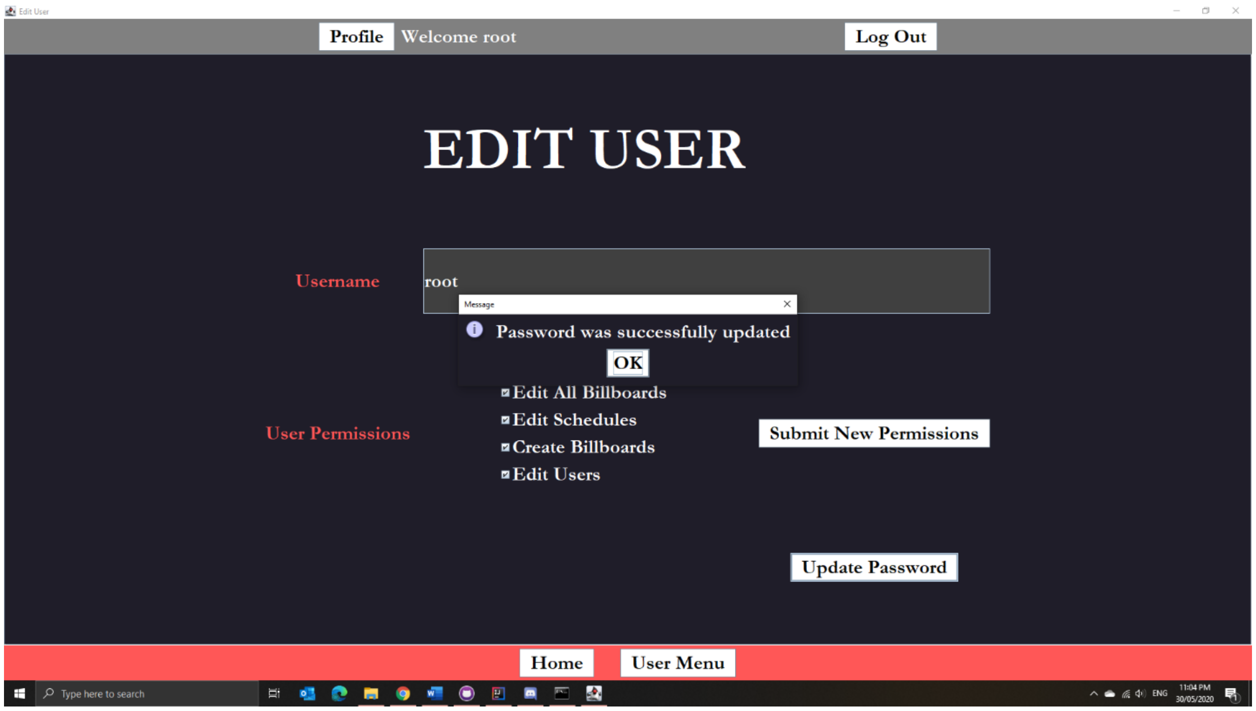
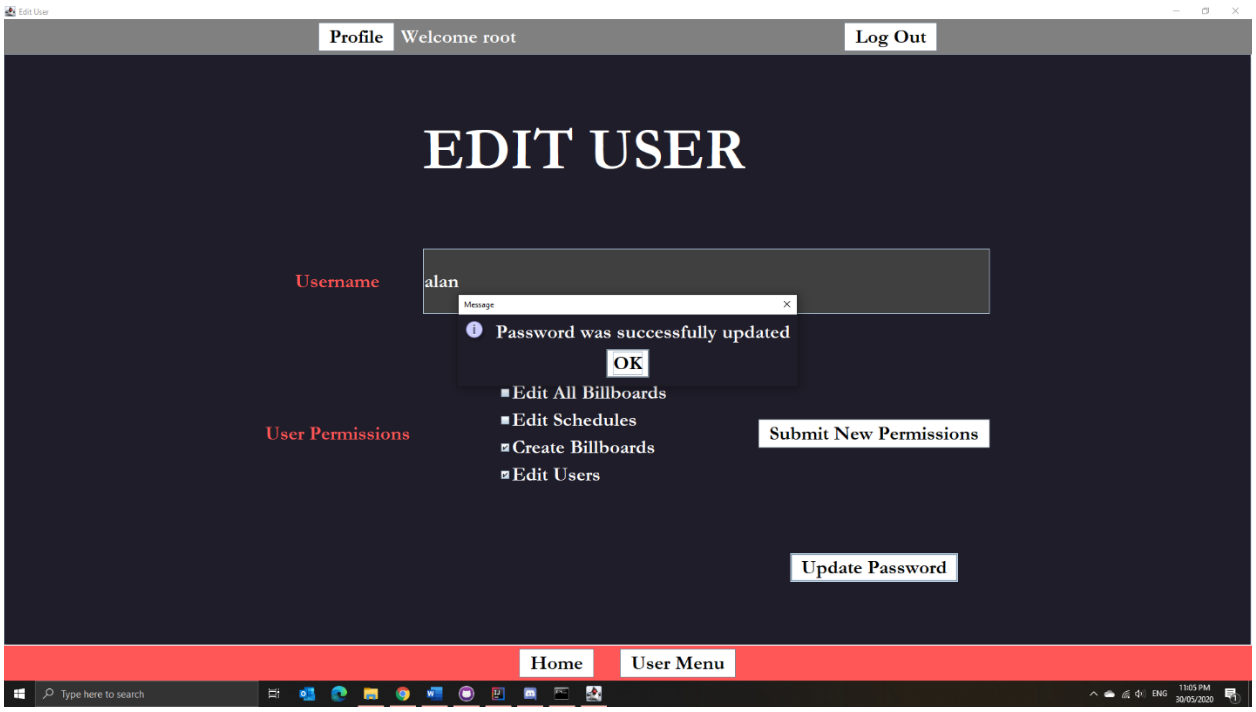


*Figure 24: Edit own password*

*Edit Other User Password*

Within the *Users* screen, users with the edit user's permission enabled also allows them to edit another user's password. This permission value is enabled for administrators by default. Once inside the view of editing User Password, the User will be required to input the original password once and the new desired password twice for verification process. The server will verify the response and update the database. The user will then be required to use this passcode in future subsequent login sessions.



alan

Message

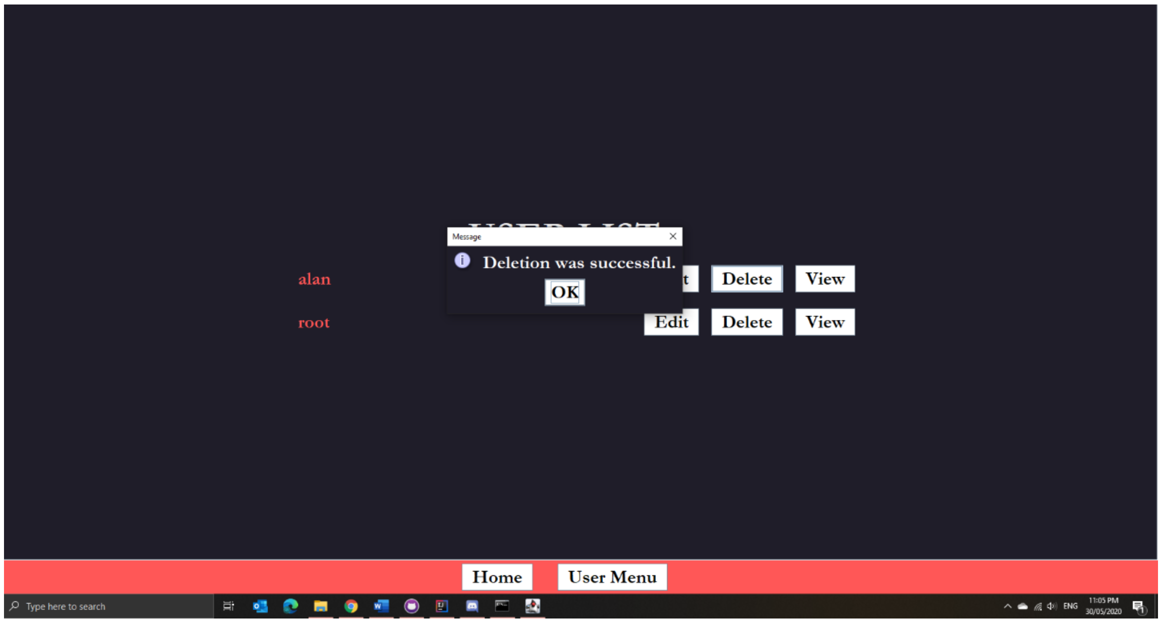
Password was successfully updated

OK

*Figure 25: Editing other user password*

*Delete Other User*

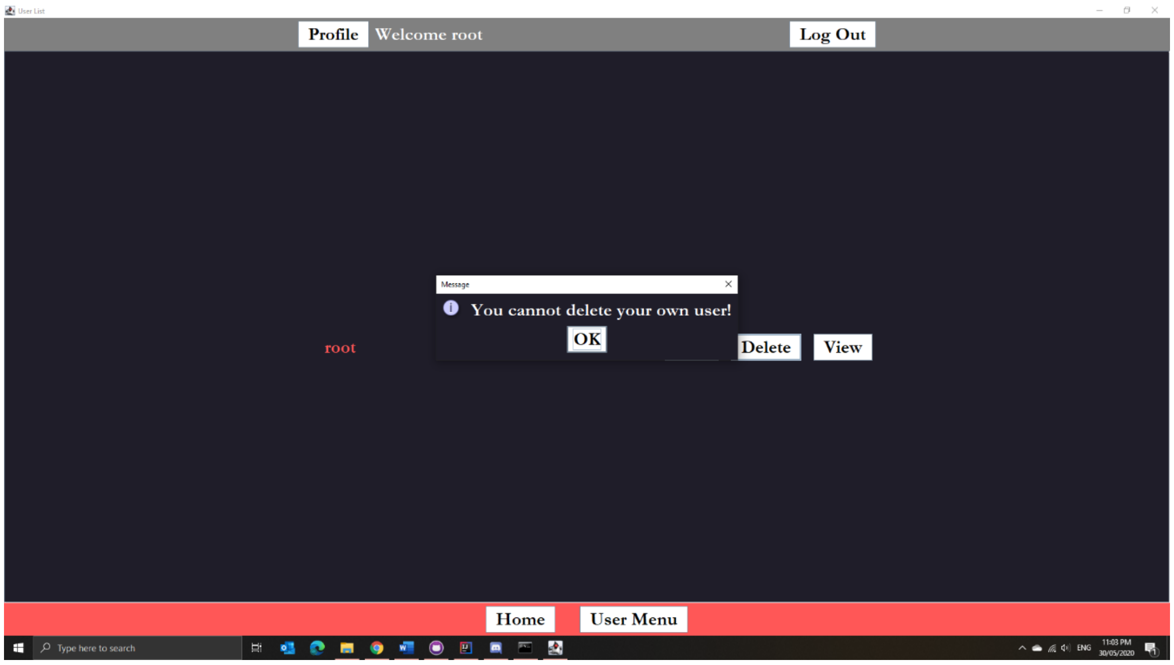
User with sufficient edit user permission will be allowed to delete other users within the server. This is enabled by administrators by default. The user will not be able to delete themselves but will be able to delete anyone else on the server. The user will be able to delete other users through viewing the user list via view all user, select a user they want to delete and once inside their profile, a delete user button will be present.



Deletion was successful.

OK

Figure 26: Deleting another user from the database



You cannot delete your own user!

OK

Figure 27: Error of deleting yourself from the database

### 4.3.4.4 Billboard Interaction Control Interface

<u>*Billboard Interface*</u>

Once the user is logged in and authenticated, they may select the *Billboard button* to interact with the billboards that are stored in the database if they have permission. This will take the user to the billboard screen.

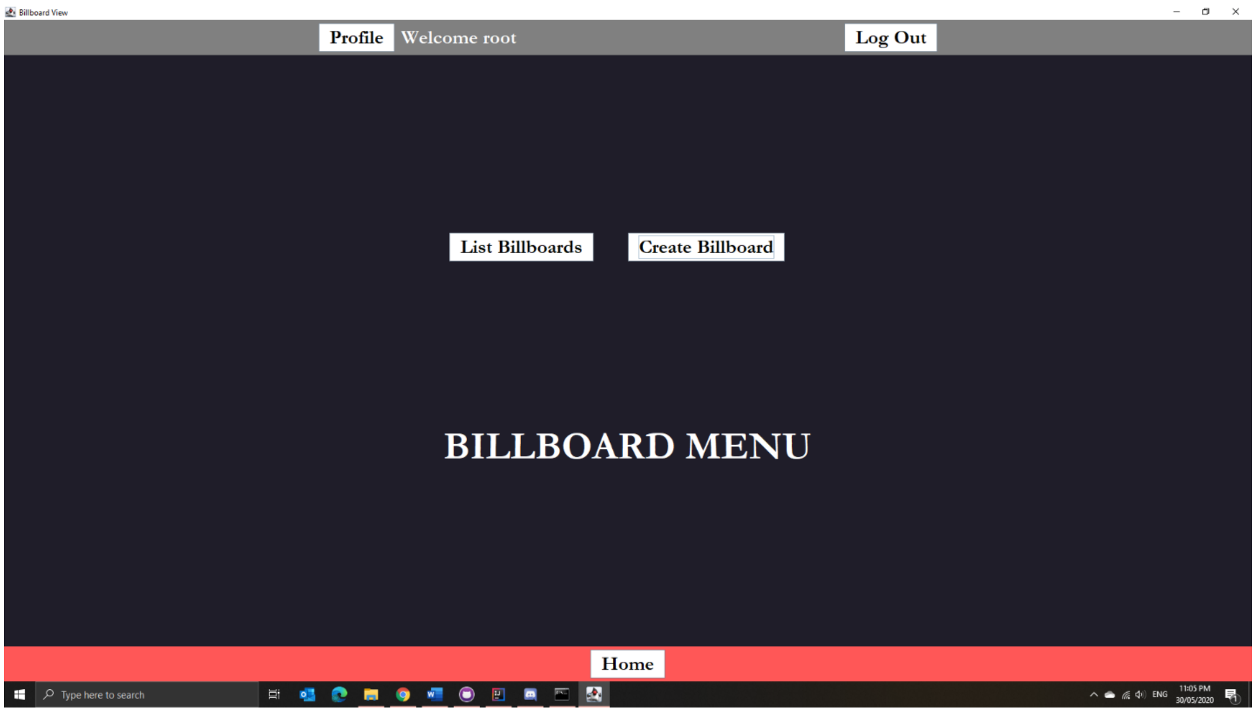
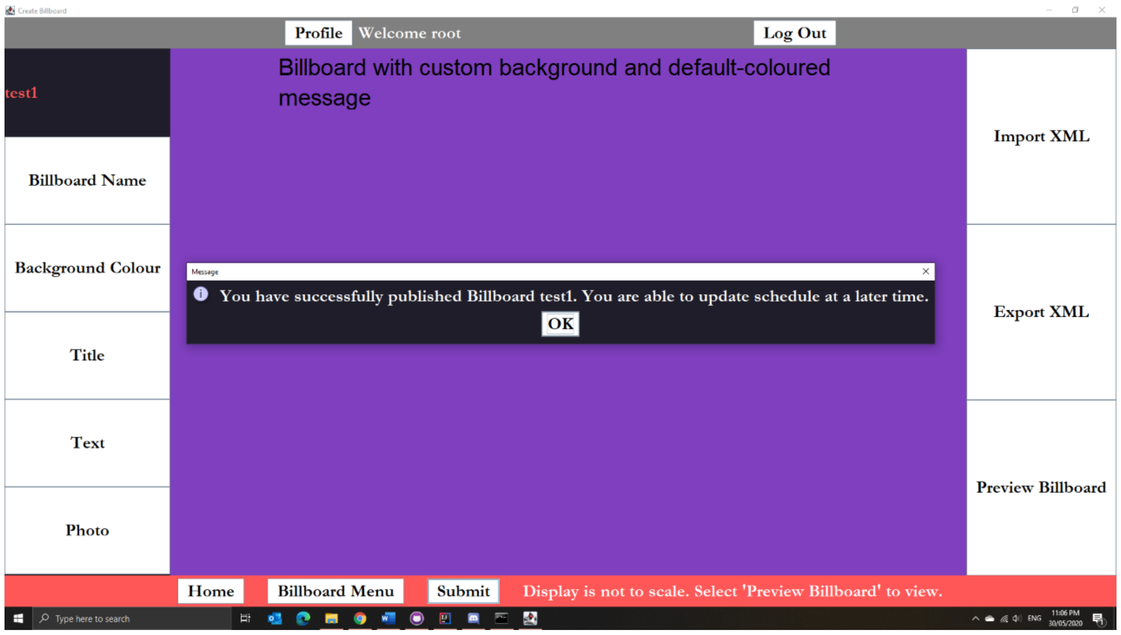


*Figure 28: Billboard Menu*

## *Add Billboard*

Users can add new billboard into the database by first going into the billboard view interface. Once inside, users with suffice permission will be able to add billboard via the add billboard button. Once inside, the user will be able to create a billboard via uploading an xml code and supplying a title.  Alternatively, a generic billboard creation tool can be used, where users will be able to create a billboard through a preset template by supplying the required fields. Once the required elements are filled in, users will be able to preview how the billboard will look like to make any adjustments as required



Billboard with custom background and default-coloured message

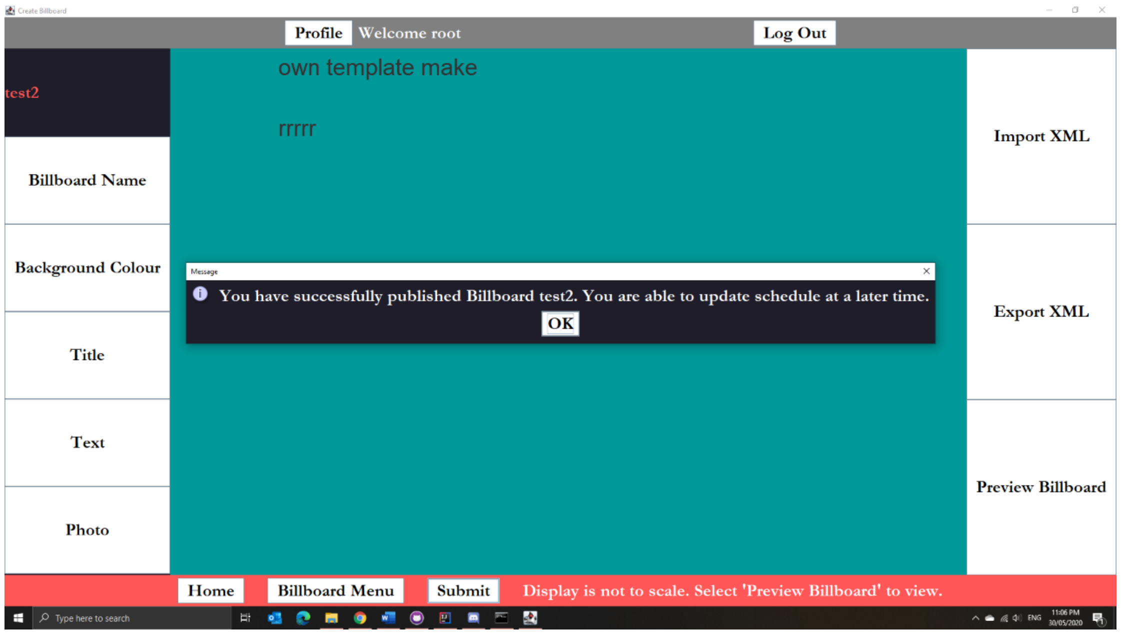
Message

You have successfully published Billboard test1. You are able to update schedule at a later time.

OK

Display is not to scale. Select 'Preview Billboard' to view.

*Figure 29: Adding a new billboard via XML*



own template make

rrrrr

Message

You have successfully published Billboard test2. You are able to update schedule at a later time.

OK

Display is not to scale. Select 'Preview Billboard' to view.

*Figure 30: Adding a new billboard via a generic template*

*View All Billboard*

All users will be able to view all billboards stored on the server database. This can be accessed under the billboard interface, and through the View All Billboard button. Once the user interacts with the button, the view will update to a list of all billboard stored in the database, displaying the billboard name and creator.

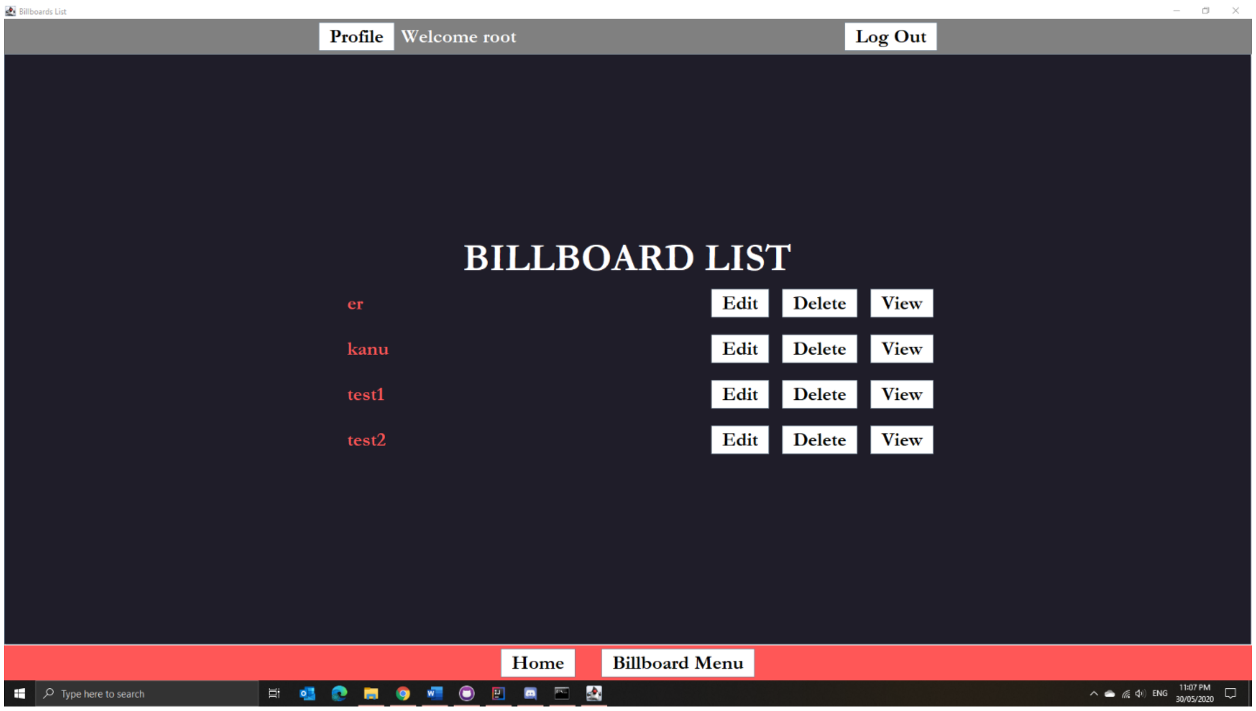


*Figure 31: View to show all billboards in database*

*View A Billboard*

All users will be able to preview a particular billboard stored on the server database. This can be accessed through the View All Billboard interface and selecting a specific billboard. The user will be able to interact with a View Billboard button, which will bring the user to a preview state and with additional billboard information displayed, such as creation date.

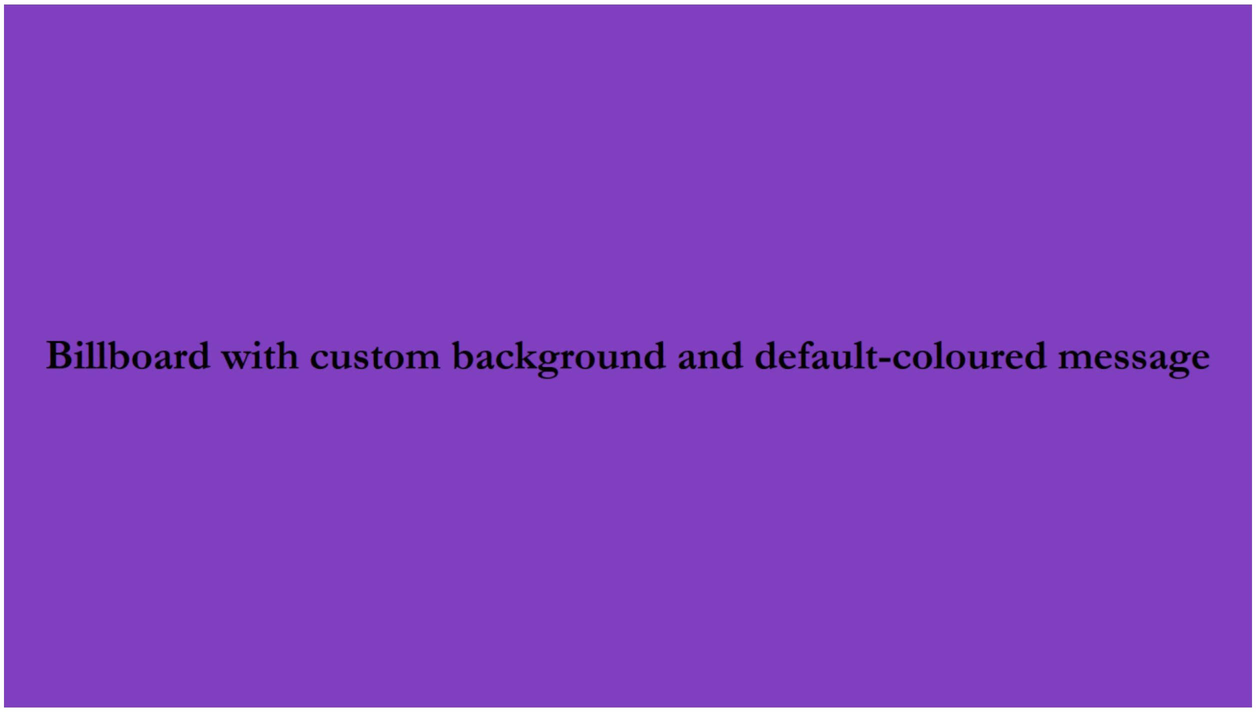

Billboard with custom background and default-coloured message

*Figure 32: Preview a specific billboard from the database*

*Edit Billboard*

Users with suffice permission will be able to edit billboards. Users can select a particular billboard they want to edit through the View All Billboard interface if they have sufficient permission, the edit billboard interface will be similar to the add billboard interface, where users will be able to upload a new xml code, title, or edit the preset fields on a billboard. Users will be able to edit their own billboard if they have the create permission, and other people's billboard or billboards that have been scheduled with the edit billboard permission. Users will also be able to preview billboards in this interface through a popup window enabled by a preview billboard button. Once the user finishes editing the billboard, the user can save the changes and exit the view.

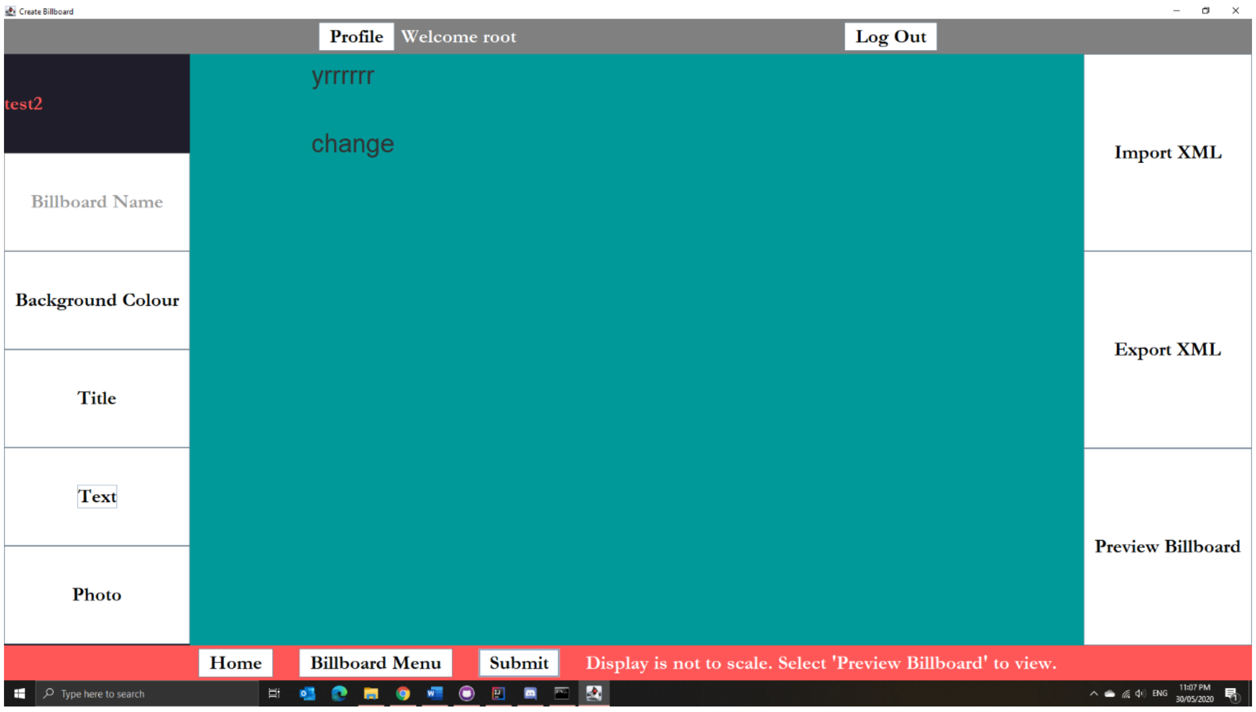
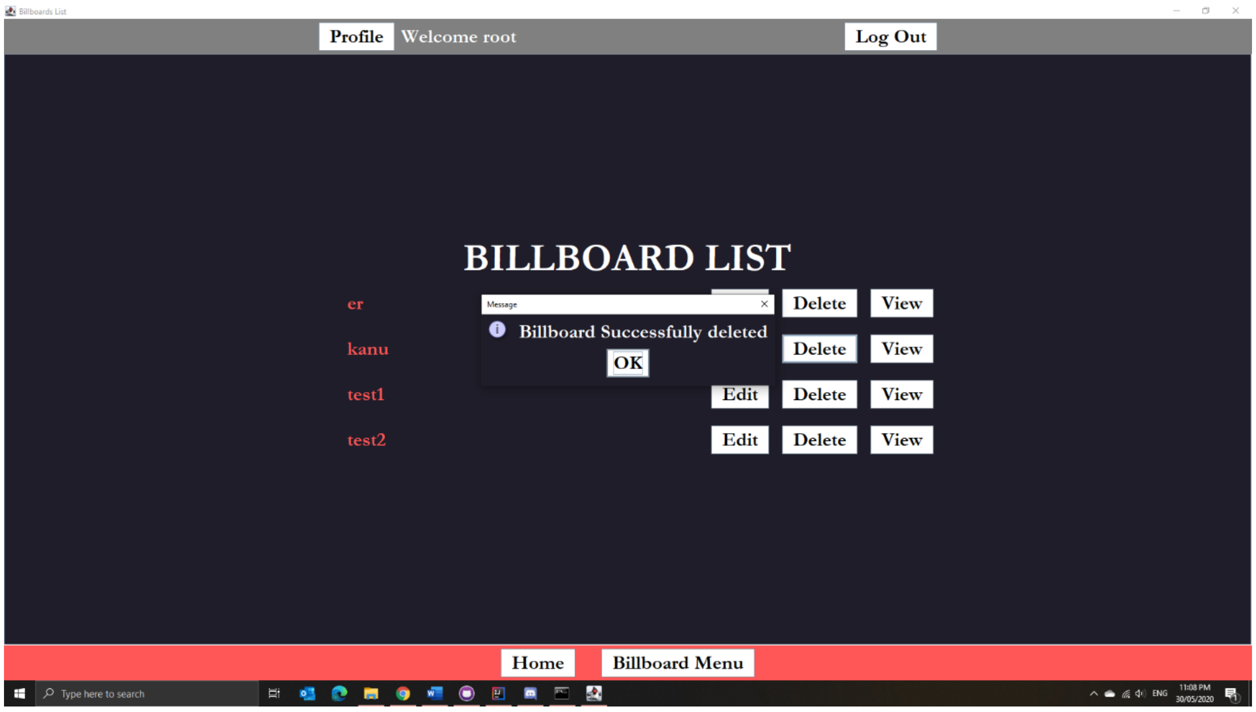


*Figure 33: Editing a preexisting Billboard*

*<u>Delete Billboard</u>*

Users with Edit billboard permission will also have the delete billboard option available for them. This will allow users to remove billboards off the database. Users who made the billboard will also be able to remove their own billboard by only having the create billboard permission if the billboard is not scheduled. The user can do so by first going into the view all billboard interface, then select a billboard to delete by interacting with the delete billboard button. This will send a request to the server and delete subsequent schedules on the server as well.



Billboard Successfully deleted

*Figure 34: Delete Billboard from database*

#### 4.3.4.5 Schedule Interaction Control Interface

<u>*Schedule Interface*</u>

Lastly, users with a valid session token will have the option to interact with the schedules in the server. Some options may not show up to the user as further permission is required (Schedule Billboard). Users can interact with schedules and view schedules through the Schedule interface. This can be done by interacting with the Schedule button on the home screen.

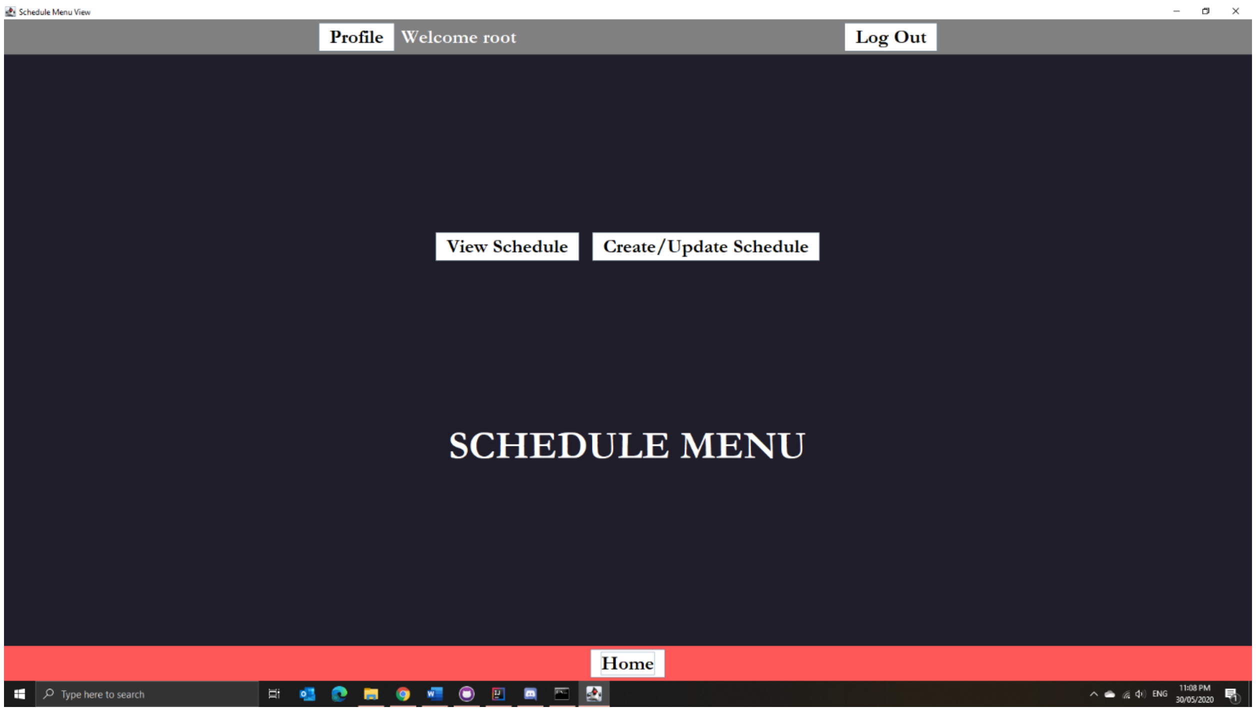
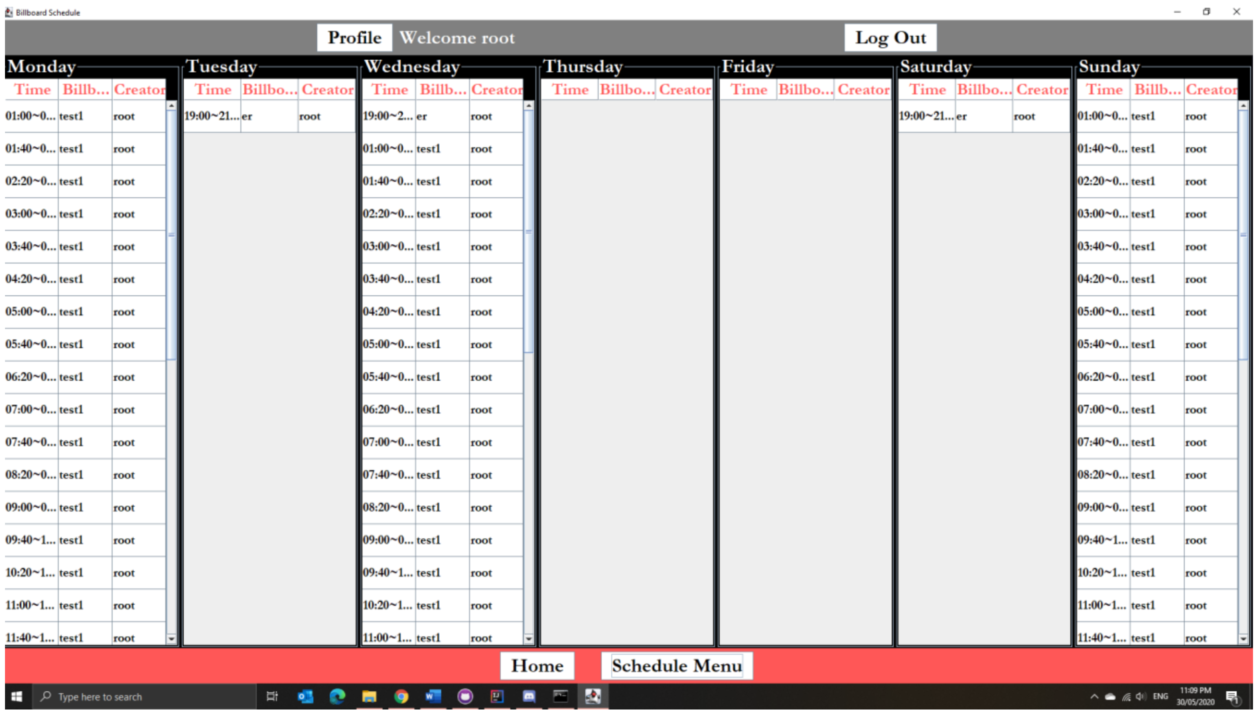


*Figure 35: Schedule Menu View*

*List Schedule*

Within the Schedule Interface all users will be able to interact with the list schedules button, which will take them to the list schedule interface. From here, users will be able to see the upcoming schedule for the week. The order of the Schedules is first sorted by alphabetical order so users who created their billboard will be able to find their schedule times easily. This is then followed by the chronological order.



Monday

| Time | Billb... | Creator |
|---|---|---|
| 01:00~0... | test1 | root |
| 01:40~0... | test1 | root |
| 02:20~0... | test1 | root |
| 03:00~0... | test1 | root |
| 03:40~0... | test1 | root |
| 04:20~0... | test1 | root |
| 05:00~0... | test1 | root |
| 05:40~0... | test1 | root |
| 06:20~0... | test1 | root |
| 07:00~0... | test1 | root |
| 07:40~0... | test1 | root |
| 08:20~0... | test1 | root |
| 09:00~0... | test1 | root |
| 09:40~1... | test1 | root |
| 10:20~1... | test1 | root |
| 11:00~1... | test1 | root |
| 11:40~1... | test1 | root |

Tuesday

| Time | Billbo... | Creator |
|---|---|---|
| 19:00~21... | er | root |

Wednesday

| Time | Billb... | Creator |
|---|---|---|
| 19:00~2... | er | root |
| 01:00~0... | test1 | root |
| 01:40~0... | test1 | root |
| 02:20~0... | test1 | root |
| 03:00~0... | test1 | root |
| 03:40~0... | test1 | root |
| 04:20~0... | test1 | root |
| 05:00~0... | test1 | root |
| 05:40~0... | test1 | root |
| 06:20~0... | test1 | root |
| 07:00~0... | test1 | root |
| 07:40~0... | test1 | root |
| 08:20~0... | test1 | root |
| 09:00~0... | test1 | root |
| 09:40~1... | test1 | root |
| 10:20~1... | test1 | root |
| 11:00~1... | test1 | root |

Thursday

| Time | Billbo... | Creator |
|---|---|---|

Friday

| Time | Billbo... | Creator |
|---|---|---|

Saturday

| Time | Billbo... | Creator |
|---|---|---|
| 19:00~21... | er | root |

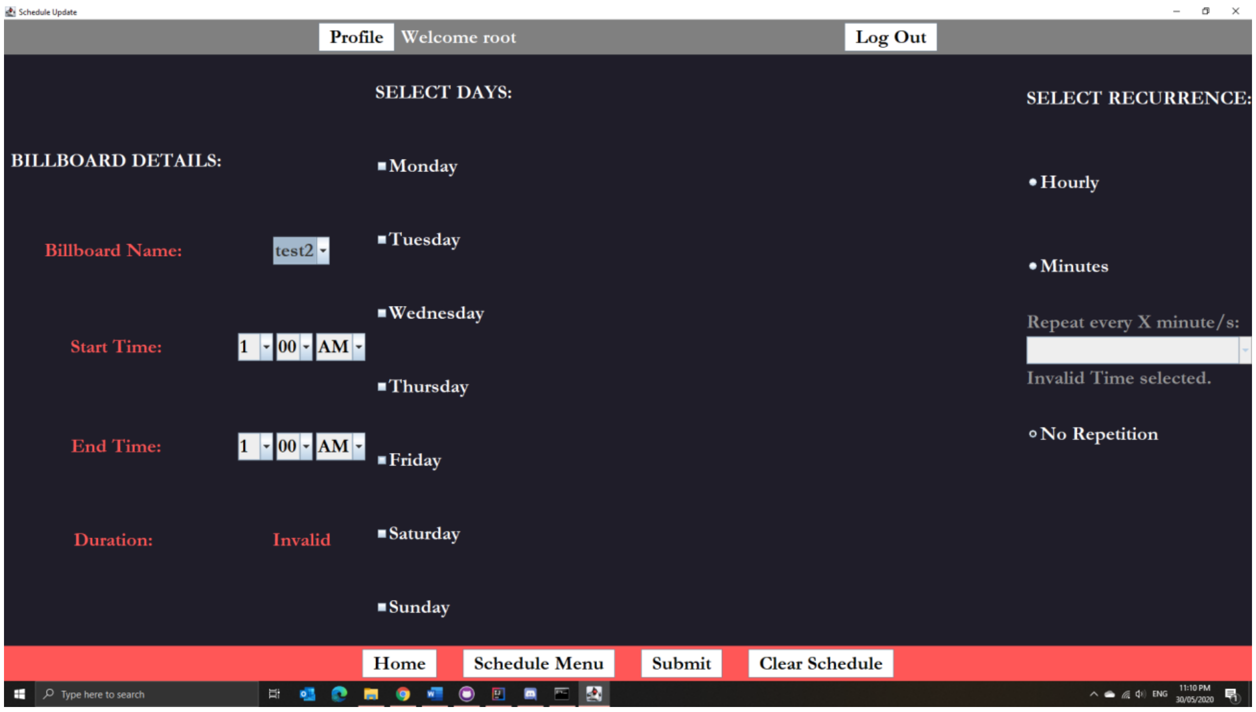
Sunday

| Time | Billb... | Creator |
|---|---|---|
| 01:00~0... | test1 | root |
| 01:40~0... | test1 | root |
| 02:20~0... | test1 | root |
| 03:00~0... | test1 | root |
| 03:40~0... | test1 | root |
| 04:20~0... | test1 | root |
| 05:00~0... | test1 | root |
| 05:40~0... | test1 | root |
| 06:20~0... | test1 | root |
| 07:00~0... | test1 | root |
| 07:40~0... | test1 | root |
| 08:20~0... | test1 | root |
| 09:00~0... | test1 | root |
| 09:40~1... | test1 | root |
| 10:20~1... | test1 | root |
| 11:00~1... | test1 | root |
| 11:40~1... | test1 | root |

*Figure 36: Show list of Schedules*

*Add Schedule*

Users with permission the Schedule Billboard will have the option to add schedules. Users will be able to interact with the add schedule button which will take the user to an add schedule interface. The interface will allow users to select a billboard which exists in the server database, then schedule the billboard via a calendar interface system. The users will be able to set the start date, duration of each schedule and if there are any repeats.



BILLBOARD DETAILS:

Billboard Name: test2

Start Time: 1 00 AM

End Time: 1 00 AM

Duration: Invalid

SELECT DAYS:

- Monday
- Tuesday
- Wednesday
- Thursday
- Friday
- Saturday
- Sunday

SELECT RECURRENCE:

- Hourly
- Minutes

Repeat every X minute/s:

Invalid Time selected.

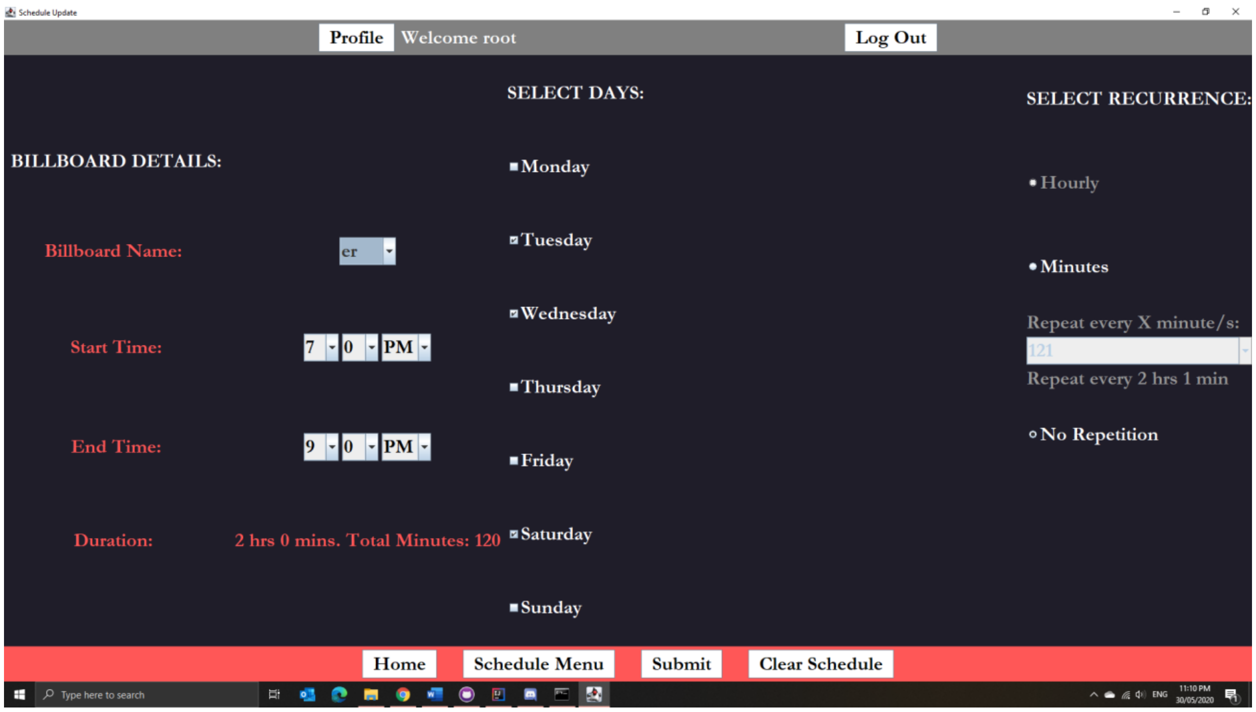
- No Repetition

*Figure 37: View for adding new schedule*

It should be noted that the GUI will limit the duration and repeats to avoid any conflicts in scheduling (such as the duration exceeding repeats).

## *Edit Schedule*

Users with permission the Schedule Billboard will have the option to edit schedules. The interface will be similar to the add schedule interface, where users can set a new schedule for the billboard. It should be noted that the same restrictions apply where no schedules can start at the same time and that schedules should not have conflicting duration and repeat parameters.



Billboard Name: er

Start Time: 7 0 PM

End Time: 9 0 PM

Duration: 2 hrs 0 mins. Total Minutes: 120

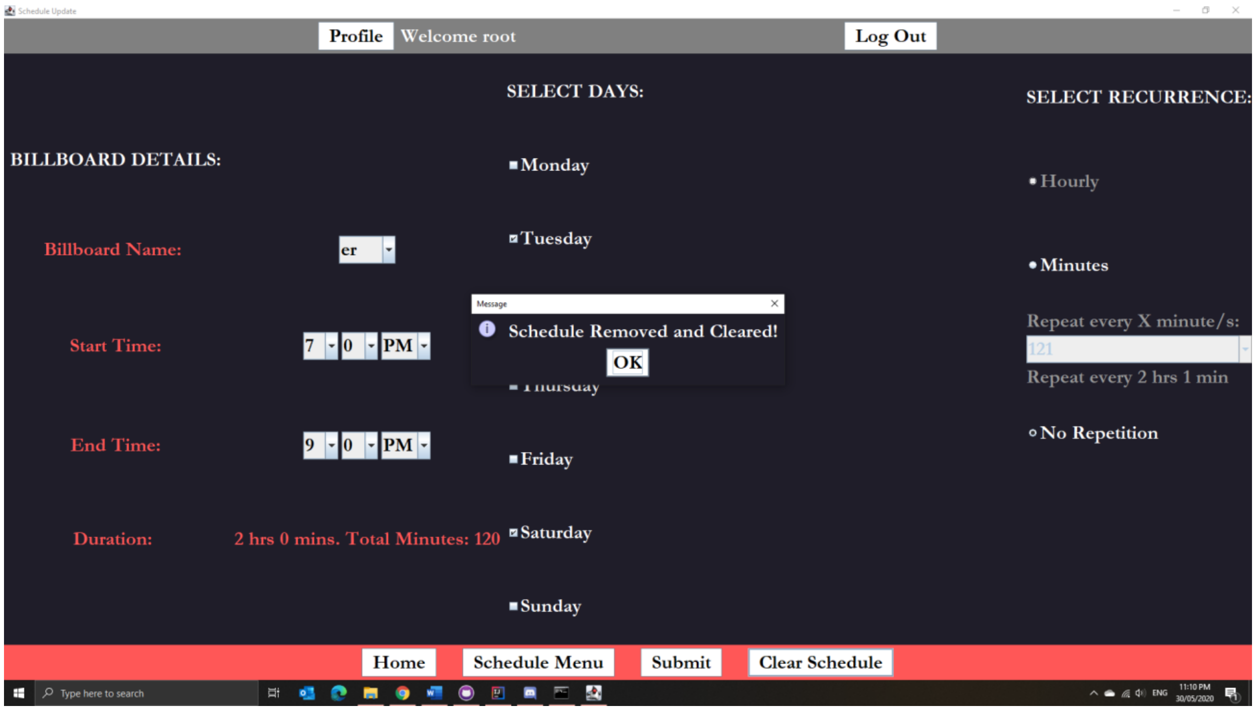
Monday
[✓ Tuesday]
[✓ Wednesday]
Thursday
Friday
[✓ Saturday]
Sunday

Hourly
[✓ Minutes]
Repeat every X minute/s: 121
Repeat every 2 hrs 1 min
No Repetition

Figure 38: View for editing a schedule

*__Delete Schedule__*

Users with permission the Schedule Billboard will have the option to delete schedules. This is accessible by clicking a Schedule in the schedule list and delete a schedule.


Schedule Update

Profile Welcome root Log Out

BILLBOARD DETAILS:

Billboard Name: er

Start Time: 7 0 PM

End Time: 9 0 PM

Duration: 2 hrs 0 mins. Total Minutes: 120

SELECT DAYS:

■Monday

☒Tuesday

Message

Schedule Removed and Cleared!

OK

■Thursday

■Friday

☒Saturday

■Sunday

SELECT RECURRENCE:

● Hourly

● Minutes

Repeat every X minute/s: 121

Repeat every 2 hrs 1 min

○ No Repetition

Home Schedule Menu Submit Clear Schedule

*Figure 39: View of removing Schedule*

# 5 Appendices

The billboard electronic management software was developed in an Agile manner. This ensures the product developed fits to specification and is to spec as per user stories. Various Agile methodology was deployed to ensure the success of development. Prior to the implementation of the software, low and high-fidelity versions of the software were generated through a UML diagram, wireframe and data flow table.

## 5.1 UML Diagram

The user stories and solution requirements were first broken down into three main components:

- Billboard Server
- Billboard Control Panel
- Billboard Viewer

An UML diagram was drawn to visualize the classes and methods present in the software solution. A sample diagram may be seen as follows.

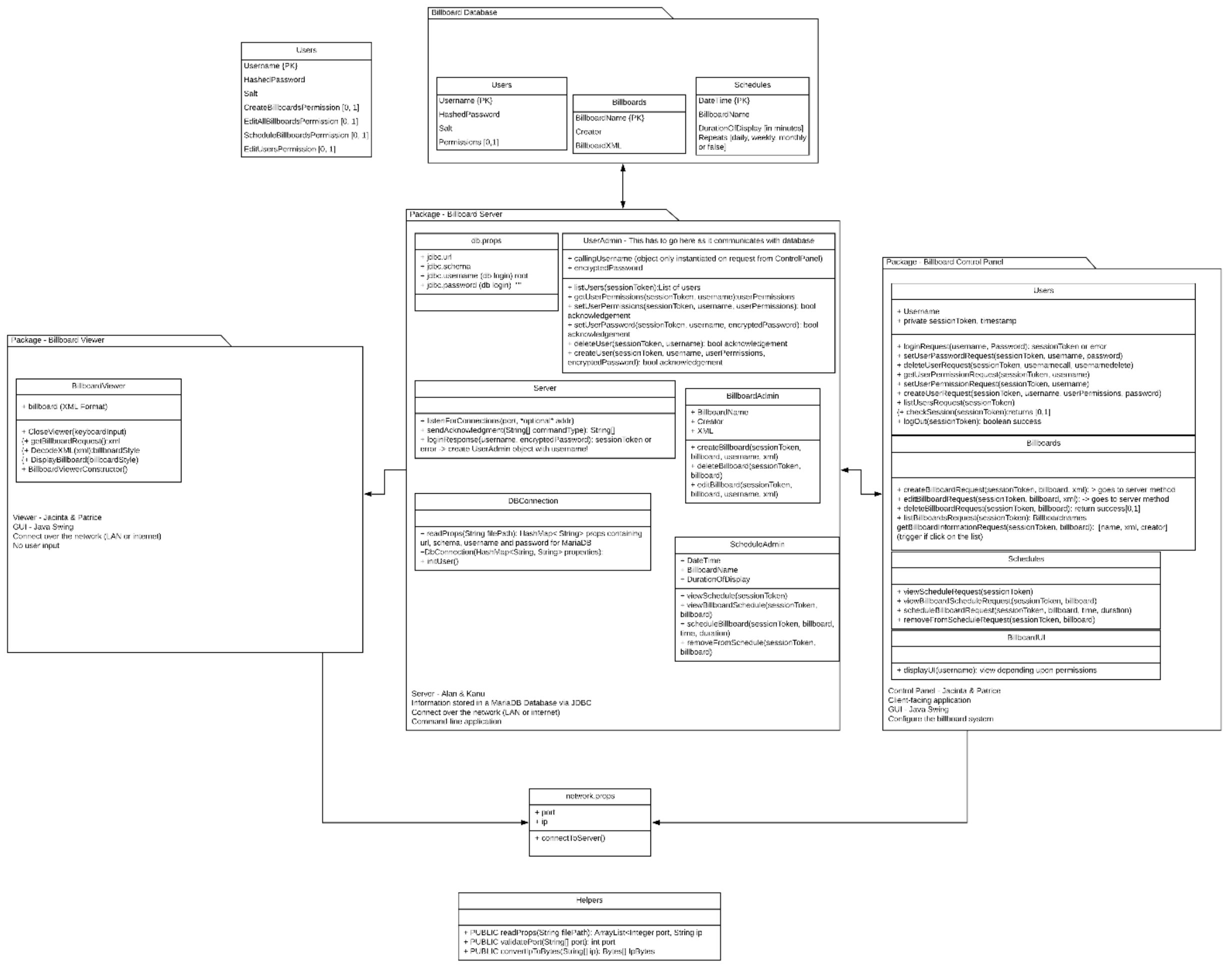

Users
Username {PK}
HashedPassword
Salt
CreateBillboardsPermission [0, 1]
EditAllBillboardsPermission [0, 1]
ScheduleBillboardsPermission [0, 1]
EditUsersPermission [0, 1]
Billboard Database
Users
Username {PK}
HashedPassword
Salt
Permissions [0,1]
Billboards
BillboardName {PK}
Creator
BillboardXML
Schedules
DateTime {PK}
BillboardName
DurationOfDisplay [in minutes]
Repeats [daily, weekly, monthly or false]
Package - Billboard Server
db.props
+ jdbc.url
+ jdbc.schema
+ jdbc.username (db login) root
+ jdbc.password (db login) ""
UserAdmin - This has to go here as it communicates with database
+ callingUsername (object only instantiated on request from ControlPanel)
+ encryptedPassword
+ listUsers(sessionToken):List of users
+ getUserPermissions(sessionToken, username):userPermissions
+ setUserPermissions(sessionToken, username, userPermissions): bool acknowledgement
+ setUserPassword(sessionToken, username, encryptedPassword): bool acknowledgement
+ deleteUser(sessionToken, username): bool acknowledgement
+ createUser(sessionToken, username, userPermissions, encryptedPassword): bool acknowledgement
Server
+ listenForConnections(port, *optional* addr)
+ sendAcknowledgment(String[] commandType): String[]
+ loginResponse(username, encryptedPassword): sessionToken or error -> create UserAdmin object with username!
BillboardAdmin
+ BillboardName
+ Creator
+ XML
+ createBillboard(sessionToken, billboard, username, xml)
+ deleteBillboard(sessionToken, billboard)
+ editBillboard(sessionToken, billboard, username, xml)
DBConnection
- readProps(String filePath): HashMap< String> props containing url, schema, username and password for MariaDB
-DbConnection(HashMap<String, String> properties):
+ initUser()
ScheduleAdmin
- DateTime
+ BillboardName
- DurationOfDisplay
- viewSchedule(sessionToken)
+ viewBillboardSchedule(sessionToken, billboard)
- scheduleBillboard(sessionToken, billboard, time, duration)
+ removeFromSchedule(sessionToken, billboard)
Server - Alan & Kanu
Information stored in a MariaDB Database via JDBC
Connect over the network (LAN or internet)
Command line application
Package - Billboard Viewer
BillboardViewer
+ billboard (XML Format)
+ CloseViewer(keyboardInput)
(+ getBillboardRequest():xml
(+ DecodeXML(xml):billboardStyle
(+ DisplayBillboard(billboardStyle)
+ BillboardViewerConstructor()
Viewer - Jacinta & Patrice
GUI - Java Swing
Connect over the network (LAN or internet)
No user input
Package - Billboard Control Panel
Users
+ Username
+ private sessionToken, timestamp
+ loginRequest(username, Password): sessionToken or error
+ setUserPasswordRequest(sessionToken, username, password)
+ deleteUserRequest(sessionToken, usernamecall, usernamedelete)
+ getUserPermissionRequest(sessionToken, username)
+ setUserPermissionRequest(sessionToken, username)
+ createUserRequest(sessionToken, username, userPermissions, password)
+ listUsersRequest(sessionToken)
(+ checkSession(sessionToken):returns [0,1]
+ logOut(sessionToken): boolean success
Billboards
+ createBillboardRequest(sessionToken, billboard, xml): > goes to server method
+ editBillboardRequest(sessionToken, billboard, xml): -> goes to server method
+ deleteBillboardRequest(sessionToken, billboard): return success[0,1]
+ listBillboardsRequest(sessionToken): Billboardnames
getBillboardInformationRequest(sessionToken, billboard): [name, xml, creator] (trigger if click on the list)
Schedules
+ viewScheduleRequest(sessionToken)
+ viewBillboardScheduleRequest(sessionToken, billboard)
+ scheduleBillboardRequest(sessionToken, billboard, time, duration)
+ removeFromScheduleRequest(sessionToken, billboard)
BillboardUI
+ displayUI(username): view depending upon permissions
Control Panel - Jacinta & Patrice
Client-facing application
GUI - Java Swing
Configure the billboard system
network.props
+ port
+ ip
+ connectToServer()
Helpers
+ PUBLIC readProps(String filePath): ArrayList<Integer port, String ip
+ PUBLIC validatePort(String[] port): int port
+ PUBLIC convertIpToBytes(String[] ip): Bytes[] IpBytes

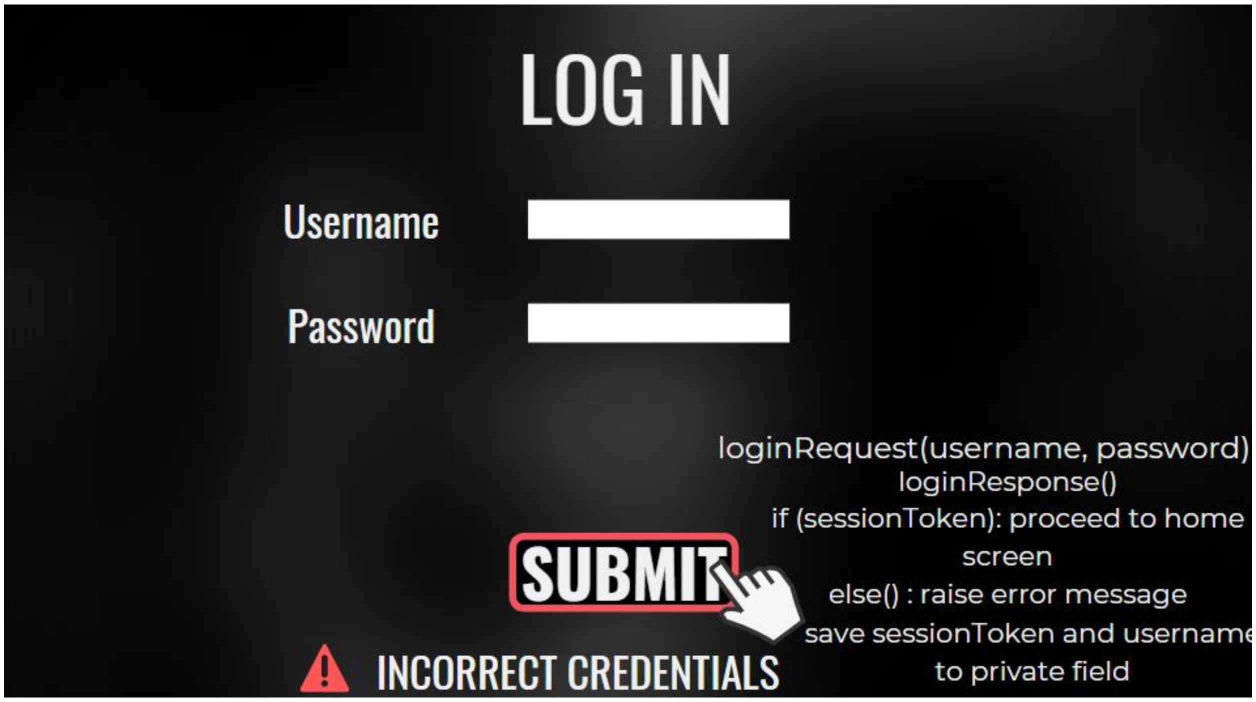
*Figure 40: Preliminary UML Diagram*

Given the requirements from the specification requires a viewer application, server application and a control panel application, it was decided that the software will be split into three main applications. Amongst the three applications (Viewer, Control Panel and Server), everything will be relayed and communicated to the server. The server also acts as the only application to interface with the database.

During the development of the software, it was decided that an additional module will be created – named "*helpers*". This module will contain helper codes such as network protocol interfaces which are used amongst all three applications.

## 5.2 Wireframe

A Wireframe of the software was then developed to visualize how the solution might look like. These also map out the requirements and interactions between the GUI and control methods for the Billboard Control Panel. A sample wireframe diagram may be seen in Figure 41, with the full wireframe present in the following link (https://bit.ly/2ZQWY7W).



loginRequest(username, password)
loginResponse()
if (sessionToken): proceed to home screen
else() : raise error message
save sessionToken and username to private field

*Figure 41: Wireframe mockup*

The wireframe has provided a better insight on the integration and requirements of the encapsulated methods within the control panel server, and the front-end GUI which the user interacts with.