

# Electronic Billboard Display and Management System

---

CAB302 SEMESTER 1, 2020

McKenzie Mitrov | Tim Bryan | Samir Tamang | Nigel Togel Rabia  
GROUP 90 | 30/05/20

## Contents

Statement of Completeness .....	2
Statement of Contributions.....	3
Design Description .....	4
Viewer application.....	4
Control Panel.....	5
Billboard Server/Database .....	7
Test-driven Development.....	14
Viewer application.....	14
Control Panel.....	14
Server .....	14
Setup/Installation Instructions .....	16
Server .....	16
Setting up MariaDB server and Creating Fresh Database.....	16
Create a new database and name it .....	16
Change properties file.....	17
Open the project and Run server .....	19
System Walkthrough .....	21
Control Panel.....	21
Server .....	29
Creating Properties File.....	29
Setting up tests for the Control Panel.....	30
Setting up tests for the unit test that deal with mock data .....	31
Setting up test for the unit test that deal with real data .....	32
Viewer .....	33
Run viewer .....	33
Close viewer.....	34

## Statement of Completeness

Control Panel Features	Implemented
Login/Logout	Y
Create/List/Delete billboards	Y
Get billboard info	Y
Display schedule	Y
Schedule billboards/Remove from schedule	Y
Create/List/Edit/Delete user	Y
Get/Set user permissions	Y
Set user password	Y
Preview billboard	Y

Server Features	Implemented
Billboard Viewer request	Y
Login request	Y
Get/List billboard information request	Y
Create/Edit billboard request	Y
Delete billboard	Y
View schedule	Y
Schedule billboard/Remove from schedule	Y
Create/List/Delete users	Y

Get/Set user permissions	Y
Set user password	Y
Log out	Y

Viewer Features	Implemented
Displays current billboard	Y
Server connectivity	Y
Attempt connection in 15 second intervals	Y
Displays appropriate error screen	Y
Exits on ESC key/mouse click	Y

### Statement of Contributions

Student	Contributions
McKenzie Mitrov (n10479651)	Billboard viewer application
Tim Bryan (n10468030)	Billboard control panel
Samir Tamang (n10488456)	Billboard server
Nigel Togel Rabia (n10236708)	Billboard server

## Design Description

### Viewer application

#### Main.java

This is the primary class for the billboard viewer. It contains functions which generate the appropriate Swing components using information retrieved from the server. These include the billboard title message, information message, and picture, which are dynamically altered using information from Generate.Java to fit on-screen.

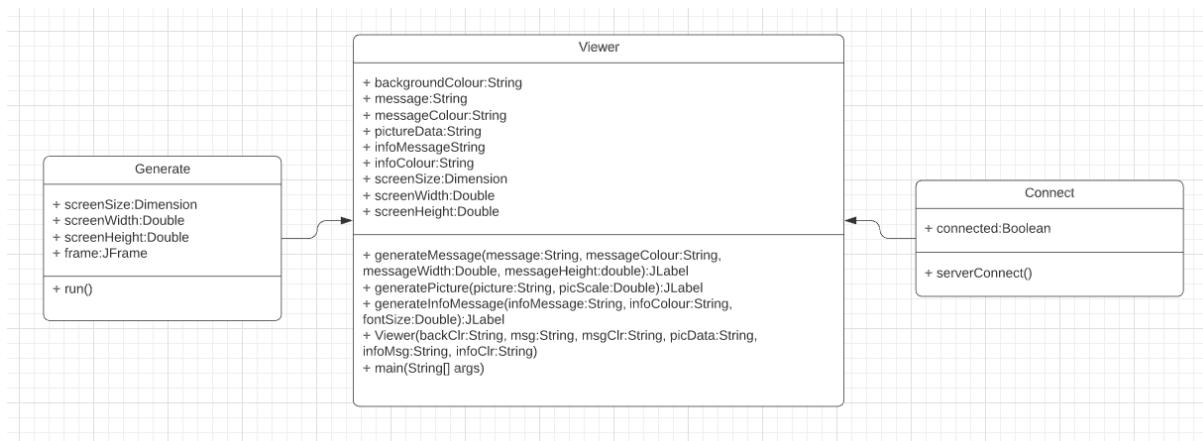
#### Connect.java

This class handles the server connectivity component of the application. The class contains a method which when called, sends a request to the billboard server for the current billboard information. This information is then stored in an array and is processed in Main.java.

#### Generate.java

This class is responsible for creating the layout of the billboard based on the retrieved information. It contains a large number of conditional statements which determine the layout and sizing of each component. A billboard is generated using the layout every 15 seconds by a timer in the main loop in Main.Java

## Viewer UML Diagram:



## Control Panel

## Billboard.java

This class is for storing the relevant data pertaining to the structure of a billboard. It allows for at least 1 of the required fields to be filled in and provides defaults for the other values.

## Schedule.java

This class is for storing the schedule, which contains data about when billboards are scheduled for, as well as for how long and how often they recur. This uses the Billboard class as it stores which billboards are scheduled for when.

## MainForm.java

This is the primary GUI class. It extends JFrame and inherits ActionListener and Runnable, so it becomes the primary GUI window. It uses both the Billboard and Schedule classes to store the data received by the ControlPanelRequests class.

## ControlPanelRequests.java

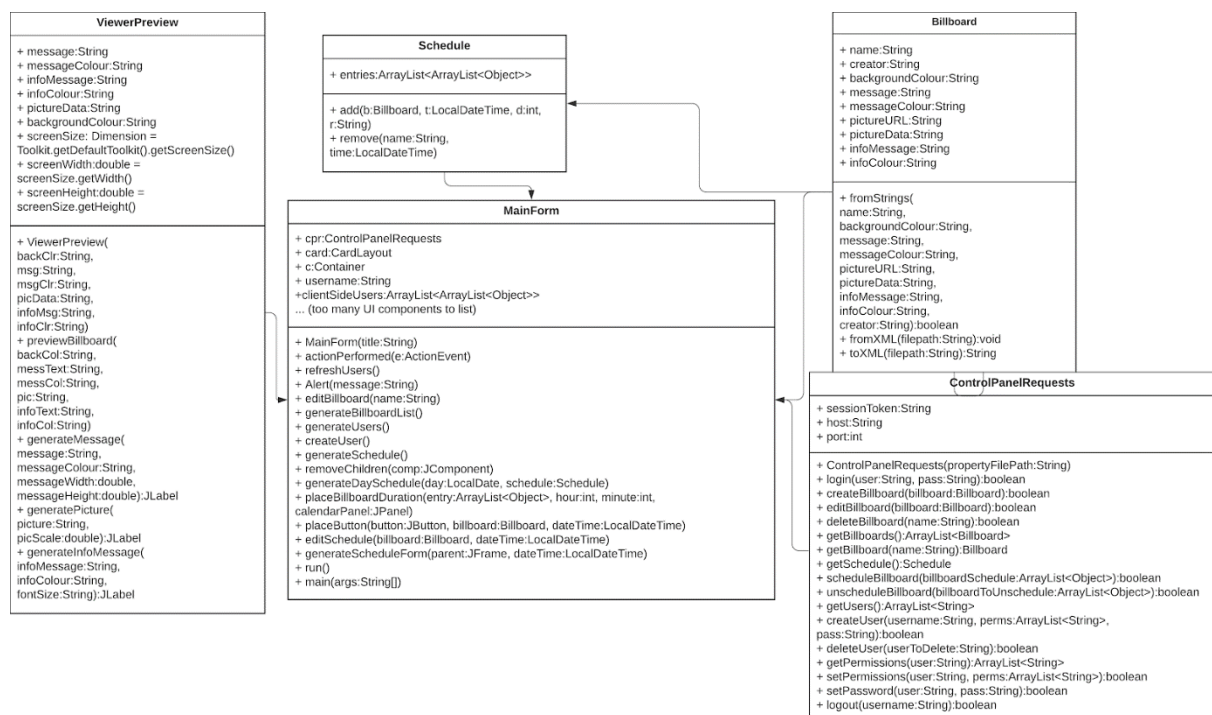
This class works as an intermediary between the server and the MainForm class. It sends requests from the MainForm to the server and receives data back, formatting it suitably for

the MainForm. It uses the Billboard, and Schedule classes to format the data received from the server.

## ViewerPreview

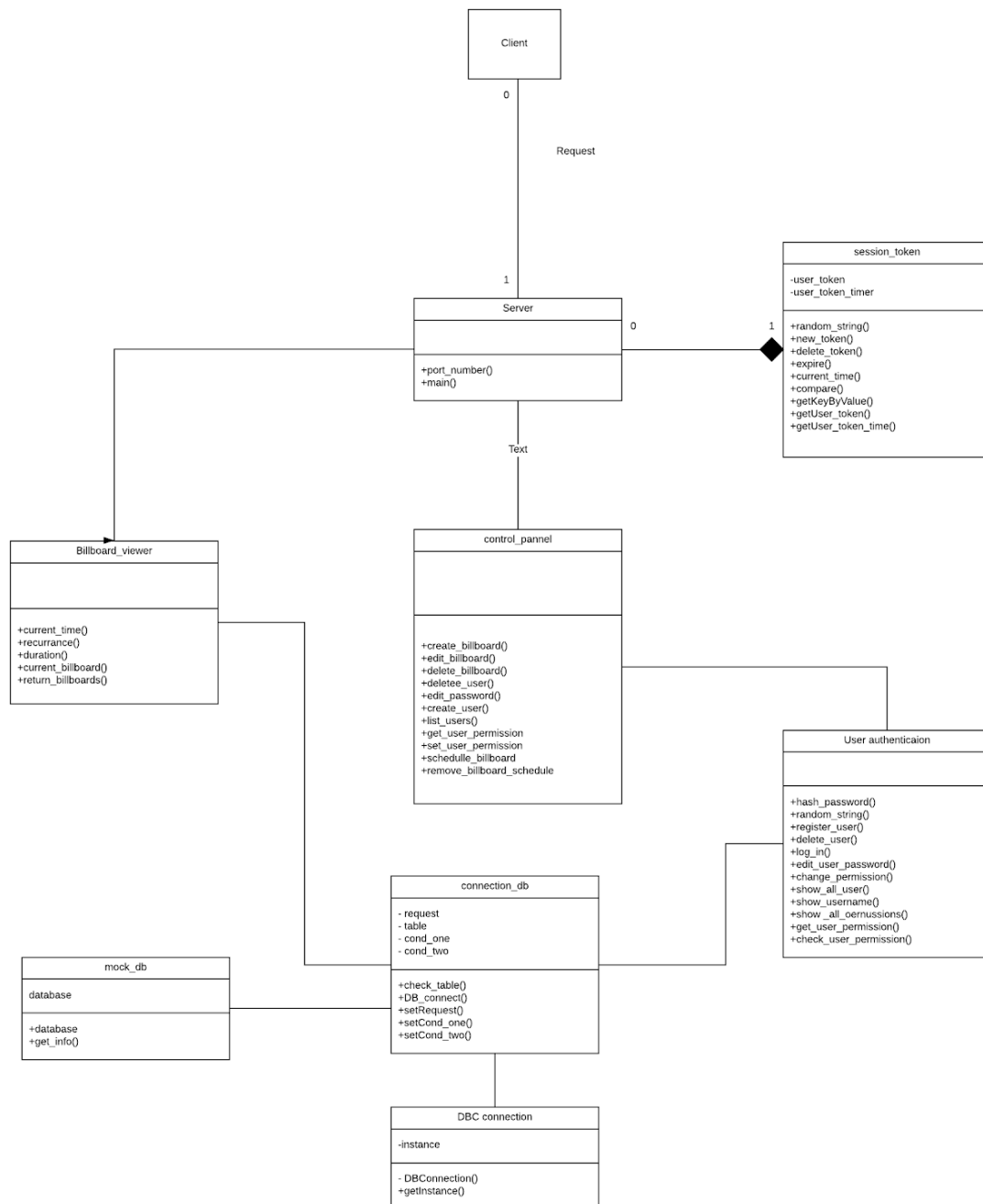
This class is adapted from the Viewer's main class to work as a non-full screen viewer for editing billboards.

## Control Panel UML Diagram:



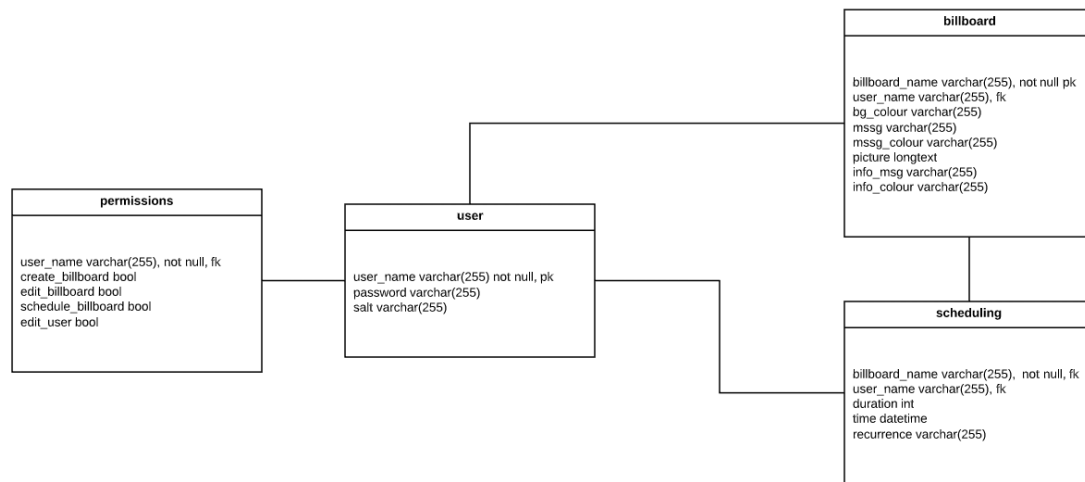
## Billboard Server/Database

Server UML Diagram:





Database relational schema:



## Billboard\_viewer

Contains classes, interface and unit test for server for handling request from billboard viewer.

## billboard\_viewer

Contains function and methods required for billboard viewer request. The main purpose of this class is to return the billboard scheduled at current time and its duration. This class also handles the recurrence of the billboard.

## billboard\_viewer test

Contains unit tests to test methods for billboard\_viewer. This unit Test was used to test billboard\_viewer which uses functions that affect the billboard directly.

## billboard\_viewer\_interface

Contains method name and its parameters for billboard\_viewer\_mock class and billboard\_viewer class

### billboard\_viewer\_mock

Contains functions and methods that are used for testing using mock data to simulate real function and data. The methods in this class were created first and after error handling, the methods were implanted in the billboard\_viewer class and were father tested on the billboard\_viewer\_test.

### billboard\_viewer\_mock\_UnitTest

These class contains Unit tests that were used for testing billboard\_viewer\_mock

### Control\_panel package

This class contains methods that are used for control panel requests from the control panel.

### control\_panel\_interface

Contains method name and its parameters for control\_panel\_server class and control\_panelMock class that are used for control panel requests

### control\_panel\_server

Contains all methods for control panel requests. This method also calls User\_authentication class objects for handling user data such as changing password, logging in, registering, etc.

### control\_panel\_unitTest

Contains unit tests for control\_panel\_server. As control\_panel\_server sends and receives data from the database, the tests are used to observe if the methods are functioning properly. And misusing of the unit tests can change the databases drastically as it contains functions that deletes, adds and edits the database. Therefore, the tests should be run separately and with care.

### control\_annelMock

Contains method that was helped to build the control\_annel\_server. Uses mock data for better unit tests. To modify code or add features on the control\_annel\_server class, the code was first tested on control\_annelMock before being implemented to control\_annel\_server.

### control\_annelMock\_unitTest

Contains unit tests for control\_annelMock.

### Database package

#### Connction\_db

This class sends different types of SQL commands to the database as string. The class contains DB\_Connect() which calls DBConnection.java class to connect to the MariaDB server and creates objects of Mock\_db.java class to store any piece of data received from the database. This class also contains a method called check\_table() that creates tables if it doesn't exist in a database, which is called by server.java class. The primary purpose of the class is sending and receiving data from the database so that the data can be used by other classes that need to interact with the MariaDB server such as control\_annel\_server.java, User\_authentication.java and billboard\_viewer.java.

#### Database\_commands

This class contains methods that are called in Connction\_db.java class where it is used to send SQL commands.

#### DBConnection

This class reads the properties file called db.props, creating a singleton instance of the database connection.

### Mock\_db.java

This class includes methods that help storing data in a 2d array list. This array contains several overloading of a method for different numbers of data to be stored. Also contains a method that calls the array list.

### Server\_package package

#### Billboard\_viewer\_client\_mock

This class creates a mock for billboard viewer requests. The main purpose of this class is to test how the server and the billboard viewer client communicates. And build the server.java class around it

#### Control\_pannel\_client\_mock

This class creates a mock for billboard viewer requests. The main purpose of this class is to test how the server and the billboard viewer client communicates. And build the server.java class around it

### Server

This class creates a server where it takes the request from the billboard viewer and control panel. This class calls user\_authentication.java, control\_pannel\_server.java and billboard\_viewer.java class which contain methods used to send and receive data depending on the requests. The requests from clients are handled by a switch case statement where the switch is the request. The request is a string sent by the client.

### User\_authentication\_package

Contains interface, classes and unit tests for user authentication and methods required for any user relating requests

### User\_authentication\_interface

Contains the methods used for mock\_user\_authentication and user\_authentication.

### Mock\_user\_authentication

Contains all the methods that are used for handling anything with the user using mock data.

For example, login user, registering user, changing user password, checking user permissions, etc. Which were later used for User\_authentication class

### Mock\_user\_authentication\_test

Contains a variety of unit tests for mock\_user\_authentication so the methods could be implemented to user authentication after checking for any kind of errors.

### User\_authentication

Contains methods from mock\_user\_authentication after unit tests but using data from the database instead of mock database, following the same or similar logic structure.

### User\_authentication\_exception

Exception class used throughout the project to handle exception not only from user\_authentication\_package but also from different package

### User\_authentication\_test

Contains unit tests to test methods for user\_authentication. This unit Test was used to test user\_authentication which uses functions that affect the billboard directly.

### Session\_token

This class contains methods for storing, deleting, adding session token, etc. The session tokens are stored in a hash map where the key is the username and the value is the token. It also

stores another HashMap where the key is the username and the value is the expiration time of the token. The class contains methods to determine when a token is expired.

Properties file

Db.props

Contains information that are necessary to connect to a database

## Test-driven Development

### Viewer application

Test-driven development was utilised in each class throughout the development. The first tests I created were to test the various 'generate' functions in Main.Java (found in ViewerTest.Java). These tests ensured that the data retrieved from the server was handled correctly and made into JLabels and did not create any exceptions. The next test I made was to ensure that the JLabels were then correctly placed in the GUI, including all information required to generate a billboard (found in GenerateTest.Java). The final test was made later in the development, and its purpose was to verify the server connection function. This test ensures that the function retrieves the correct data and connects to the server as intended without creating any exceptions. (found in ConnectTest.Java).

### Control Panel

Test-driven development was used for the ControlPanelRequests class. At first it was created to verify it was receiving data from the mock when it should. Now with the server working, the tests were edited to include stricter checking of actual functionality. It involves logging in, creating new entries in the database, checking that they exist. It also includes negative checks, such as wrong credentials being unable to log in.

### Server

Test-driven development was used for creating methods using mock data. The mock data is a 2D array String list. Using test driven development, I was able to create a mock class and use the mock data and do a wide range of unit tests on each method. So, implementing methods and classes following the same or similar logic structure that use the real data was easier.

Unit tests also ran on the class and methods that use the data from the database. These Unit tests were done to test if the methods were working correctly and also used to give database values without the need of a server. As the Unit tests for the classes that send and receive data from the database, running the tests with caution is advised as these tests can change the database.

The classes that were made using Test-Driven Development were: `User_authentication.java`, `control_panel_server.java`, `billboard_viewer.java`. And their mock class that used mock data are: `billboard_viewer_mock.java`, `control_panelMock.java`, `mock_user_authnticaion.java`.



## Setup/Installation Instructions

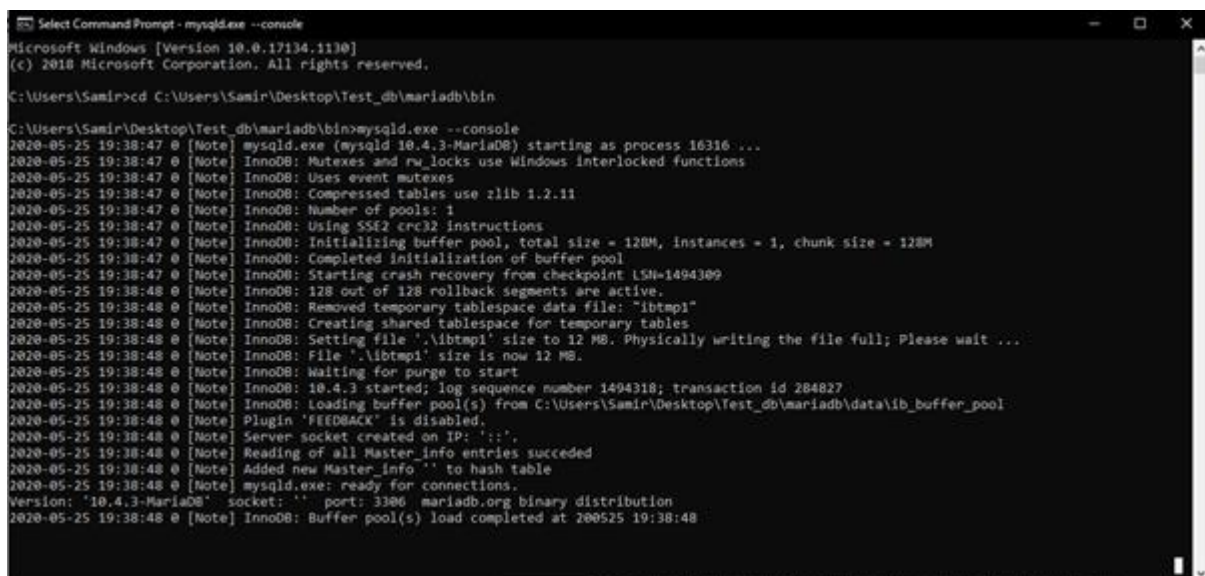
### Server

#### Setting up MariaDB server and Creating Fresh Database

Open up a command prompt and change the directory to MariaDB bin folder.

```
C:\Users\Samir>cd C:\Users\Samir\Desktop\Test_db\mariadb\bin
```

Run MariaDB server using "--console" command



```
Select Command Prompt - mysqld.exe --console
Microsoft Windows [Version 10.0.17134.1130]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Samir>cd C:\Users\Samir\Desktop\Test_db\mariadb\bin

C:\Users\Samir\Desktop\Test_db\mariadb\bin>mysqld.exe --console
2020-05-25 19:38:47 [Note] mysqld.exe (mysqld 10.4.3-MariaDB) starting as process 16316 ...
2020-05-25 19:38:47 [Note] InnoDB: Mutexes and rw_locks use Windows interlocked functions
2020-05-25 19:38:47 [Note] InnoDB: Uses event mutexes
2020-05-25 19:38:47 [Note] InnoDB: Compressed tables use zlib 1.2.11
2020-05-25 19:38:47 [Note] InnoDB: Number of pools: 1
2020-05-25 19:38:47 [Note] InnoDB: Using SSE2 crc32 instructions
2020-05-25 19:38:47 [Note] InnoDB: Initializing buffer pool, total size = 128M, instances = 1, chunk size = 128M
2020-05-25 19:38:47 [Note] InnoDB: Completed initialization of buffer pool
2020-05-25 19:38:47 [Note] InnoDB: Starting crash recovery from checkpoint LSN=1494309
2020-05-25 19:38:48 [Note] InnoDB: 128 out of 128 rollback segments are active.
2020-05-25 19:38:48 [Note] InnoDB: Removed temporary tablespace data file: "ibtmp1"
2020-05-25 19:38:48 [Note] InnoDB: Creating shared tablespace for temporary tables
2020-05-25 19:38:48 [Note] InnoDB: Setting file 'ibtmp1' size to 12 MB. Physically writing the file full; Please wait ...
2020-05-25 19:38:48 [Note] InnoDB: File 'ibtmp1' size is now 12 MB.
2020-05-25 19:38:48 [Note] InnoDB: Waiting for purge to start
2020-05-25 19:38:48 [Note] InnoDB: 10.4.3 started; log sequence number 1494318; transaction id 284827
2020-05-25 19:38:48 [Note] InnoDB: Loading buffer pool(s) from C:\Users\Samir\Desktop\Test_db\mariadb\data\ib_buffer_pool
2020-05-25 19:38:48 [Note] Plugin 'FEEDBACK' is disabled.
2020-05-25 19:38:48 [Note] Server socket created on IP: '::'.
2020-05-25 19:38:48 [Note] Reading of all Master_info entries succeeded
2020-05-25 19:38:48 [Note] Added new Master_info '' to hash table
2020-05-25 19:38:48 [Note] mysqld.exe: ready for connections.
Version: '10.4.3-MariaDB' socket: '' port: 3306 mariadb.org binary distribution
2020-05-25 19:38:48 [Note] InnoDB: Buffer pool(s) load completed at 2020-05-25 19:38:48
```

Create a new database and name it

After establishing the MariaDB server. Now create a fresh database by opening up another command prompt and changing the directory to the bin folder again. Now using the '-uroot' command we can use MariaDB to create a database.

```
C:\Users\Samir>cd C:\Users\Samir\Desktop\Test_db\mariadb\bin
```

Now using the "create database (name)" command create a database with a suitable name.

For example:

```
MariaDB [(none)]> create database tutorial;  
Query OK, 1 row affected (0.029 sec)
```

Change properties file

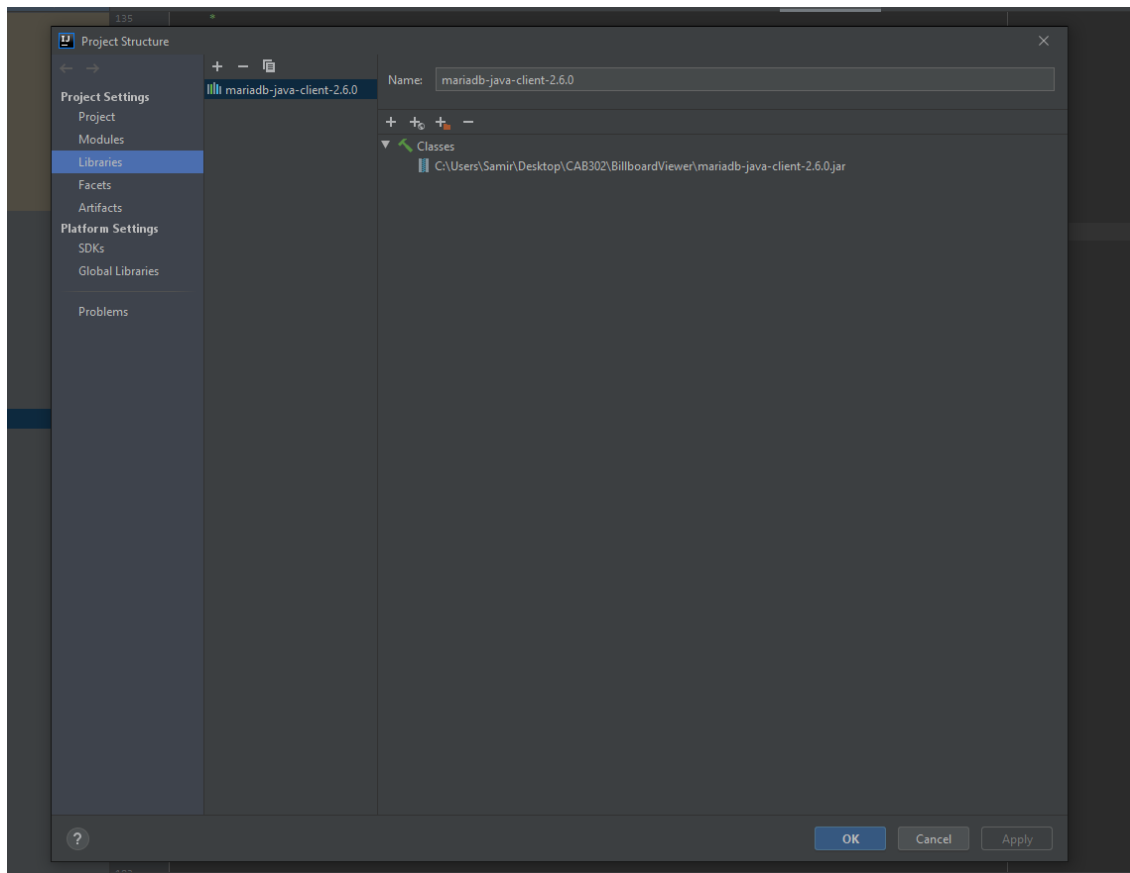
Now in the project file. Open the properties file called db.props.

As shown in this image. Set the jdbc.schema to tutorial or the given name to the database to connect to.

```
jdbc.url=jdbc:mysql://localhost:3306  
jdbc.schema=tutorial  
jdbc.username=root  
jdbc.password=
```

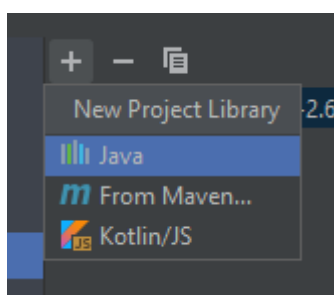
Open the project and add/check for MariaDB client in the library

Now open the project using IntelliJ and add/check the MariaDB to the library so the project can connect to the database. This can be done by going to the file and clicking on the project structure. After that go on the Libraries in the Project Setting and check if it [has MariaDB java-client or not.

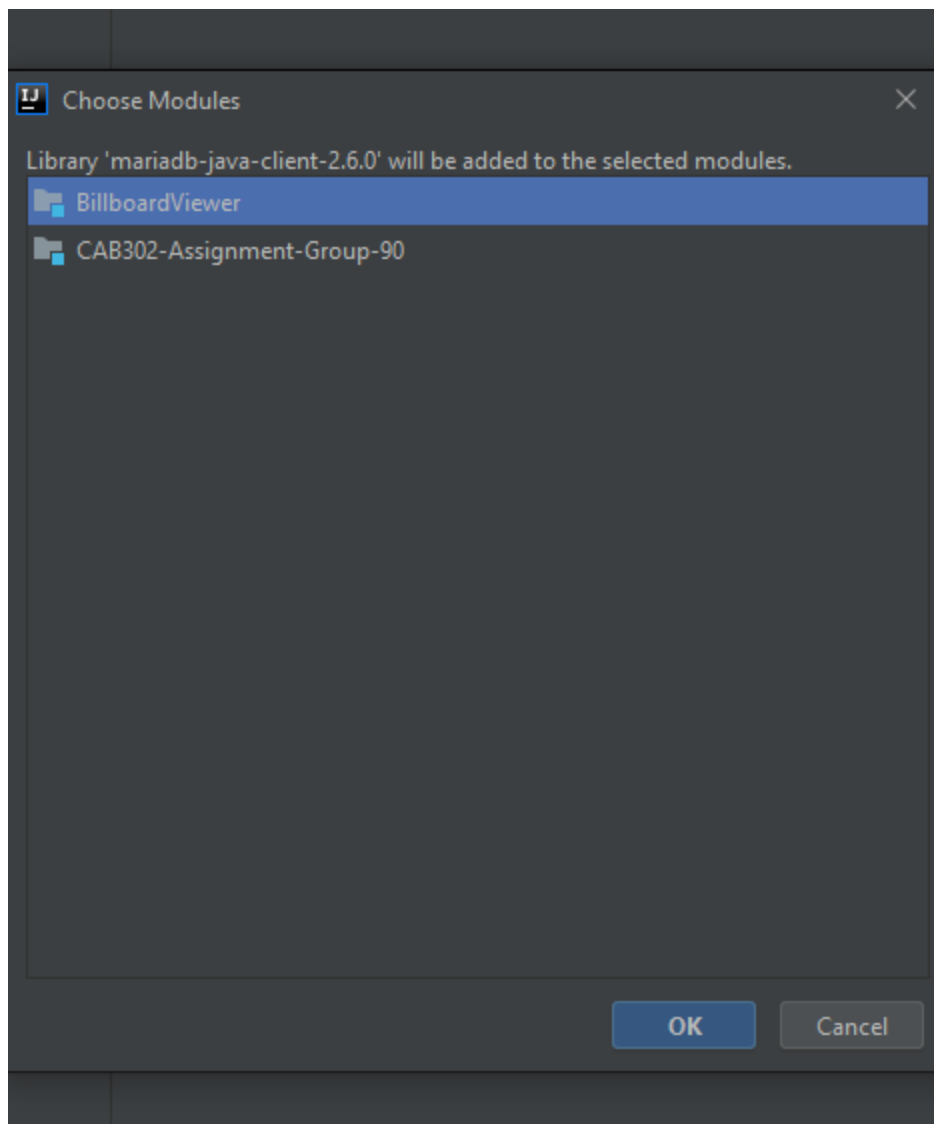


As we can see from the image above it shows if MariaDB is in the project library, if not follow these instructions.

Click on the '+' sign and press the java button. After that navigate to the project directory then open the BillboardViewer folder. After then select the maraidb-java-client-2.6.0.jar then press ok.

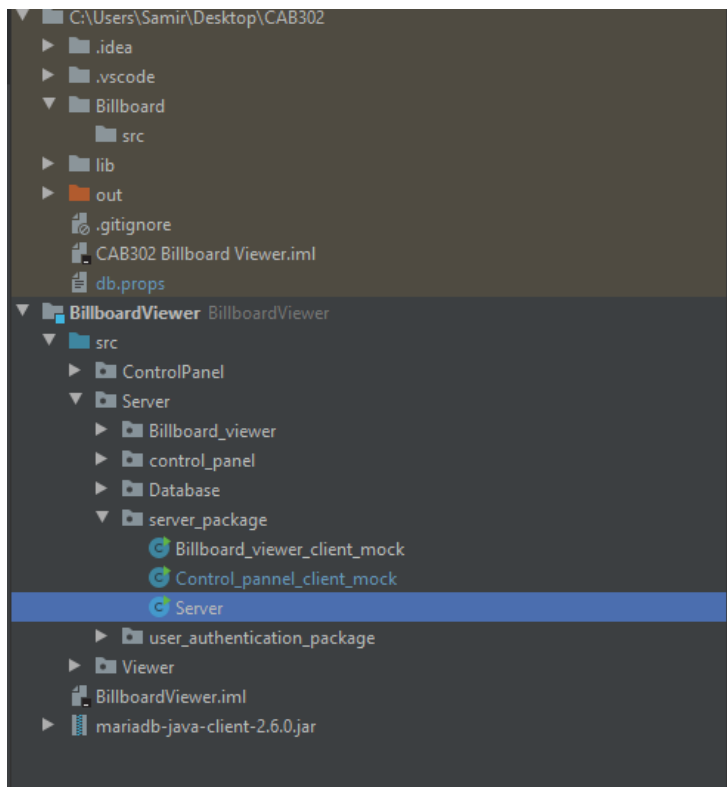


The next screen should look something like this. Then select on the BillboardViewer press ok and press ok. Now the MariaDB client is successfully added to the library.



Open the project and Run server

After creating a new database and changing the properties file `jdbc.schema` to the database and having added a MariaDB client on the library. Open up the server package located inside the `src` of `BillboardViewer` and open the `server.java` file located inside `server_package`.



Now run the server class. This should automatically create new tables from the database and also create a user with all the permissions called “admin” and password “password”.

```
MariaDB [tutorial]> select * from user;
+-----+-----+-----+
| user_name | password | salt |
+-----+-----+-----+
| admin    | 404116458 | ZUszG |
+-----+-----+-----+
1 row in set (0.000 sec)

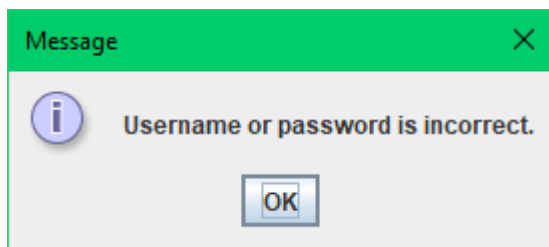
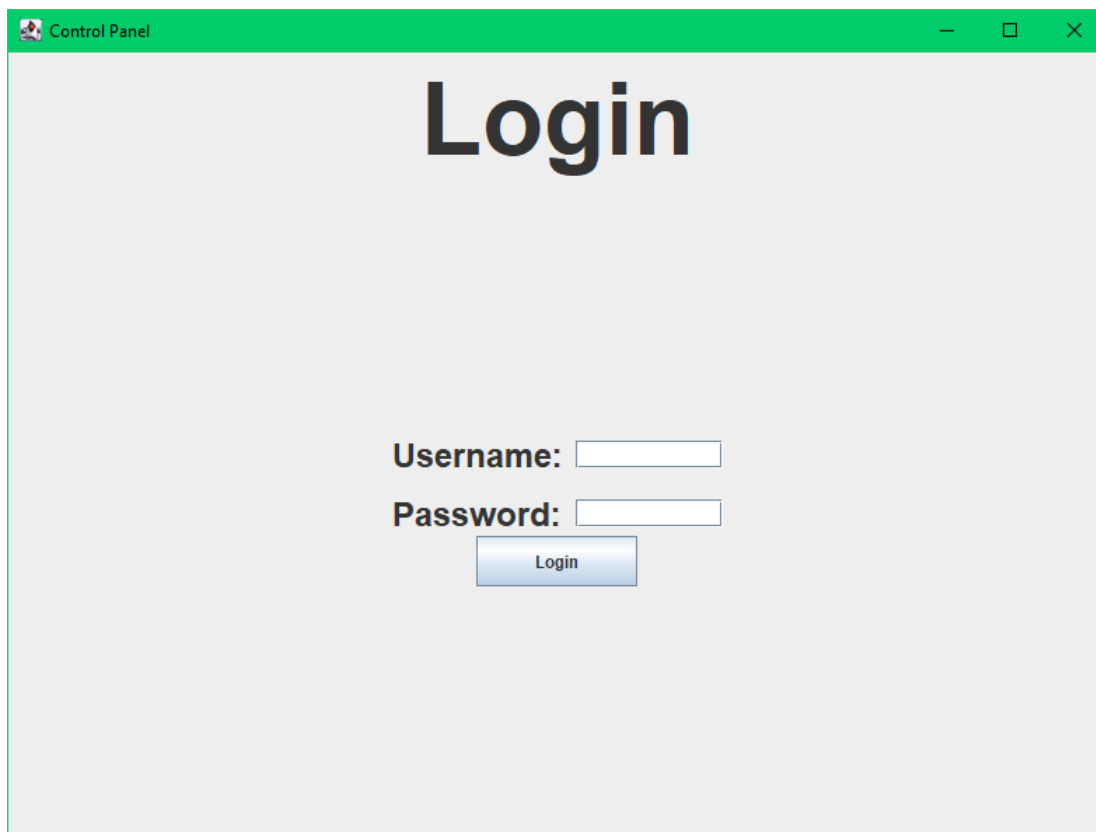
MariaDB [tutorial]> select * from permission;
+-----+-----+-----+-----+-----+
| user_name | create_billboard | edit_billboard | schedule_billboard | edit_user |
+-----+-----+-----+-----+-----+
| admin    | 1 | 1 | 1 | 1 |
+-----+-----+-----+-----+-----+
1 row in set (0.000 sec)
```

Now the next time to start up the server, only run the server.java file and leave it running.

## System Walkthrough

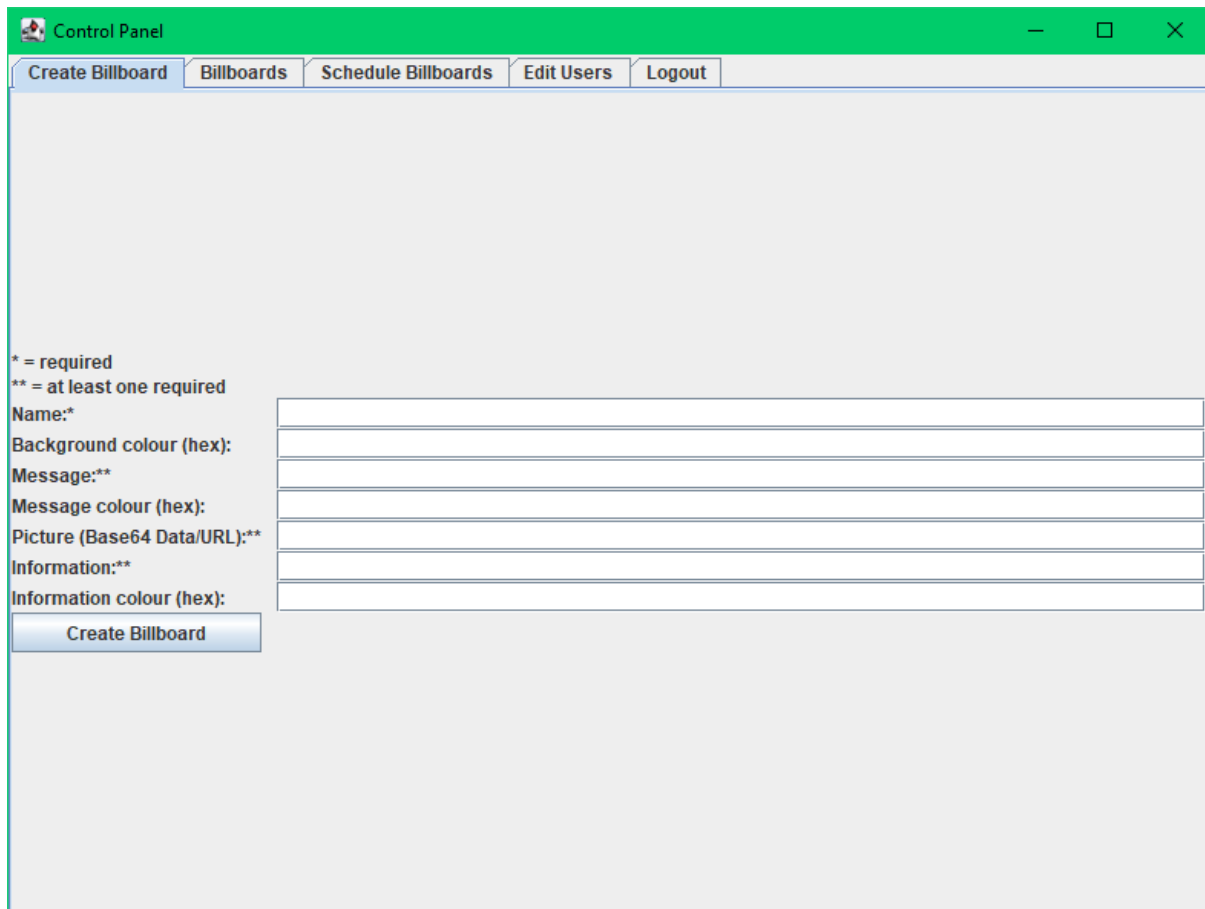
### Control Panel

Upon launching of the Control Panel, a login screen will be displayed. Enter the default username and password and it will proceed. An incorrect username and password will bring up an alert.



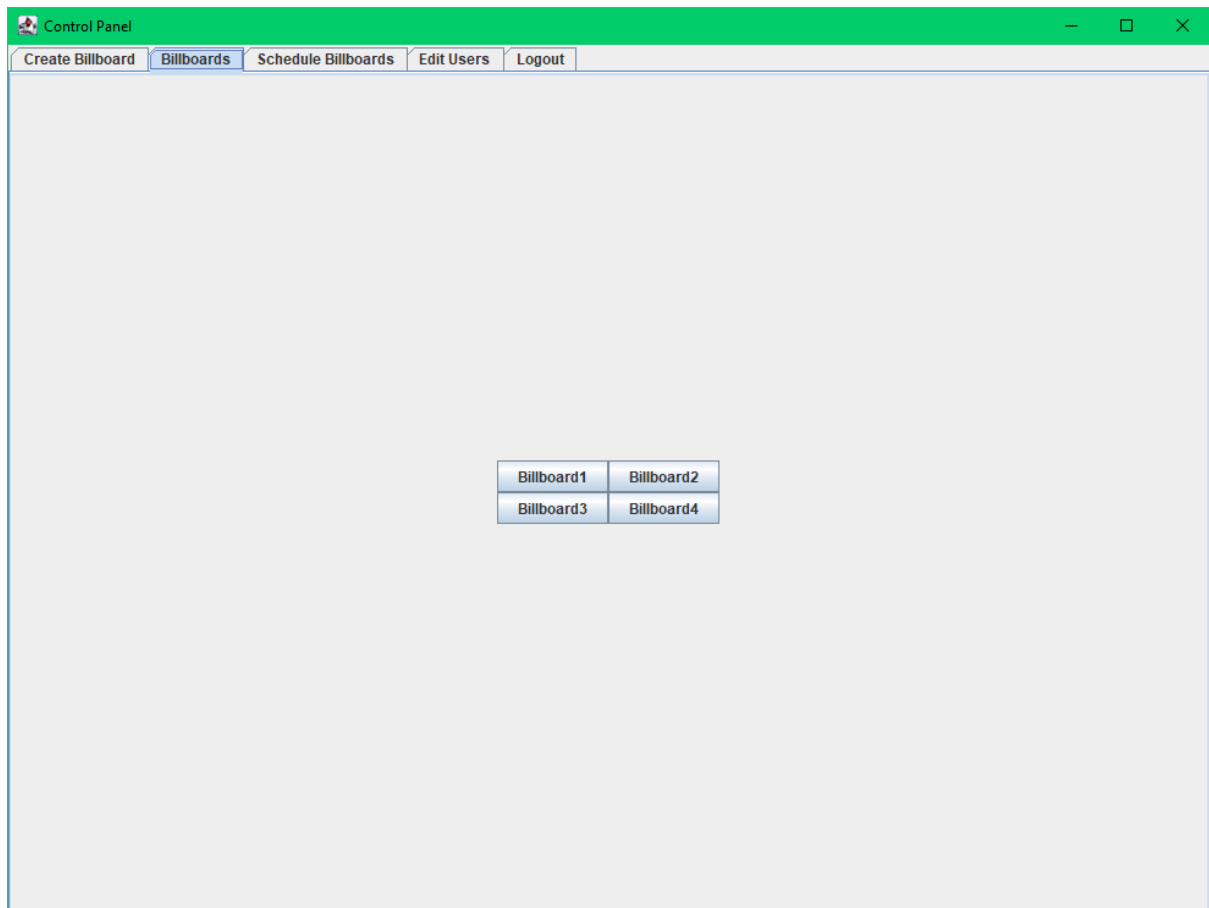
On the main screen there will be multiple tabs available to navigate.

The “Create Billboard” tab allows the user to create a billboard, by filling in the relevant information.



The screenshot shows a web application window titled "Control Panel" with a green header bar. Below the header is a navigation menu with five tabs: "Create Billboard" (selected), "Billboards", "Schedule Billboards", "Edit Users", and "Logout". The main content area is light gray. On the left side of the main area, there is a legend: "\* = required" and "\*\* = at least one required". Below the legend are seven input fields with labels to their left: "Name:\*", "Background colour (hex):", "Message:\*\*", "Message colour (hex):", "Picture (Base64 Data/URL):\*\*", "Information:\*\*", and "Information colour (hex):". Each label is followed by a text input field. At the bottom left of the main area is a blue button labeled "Create Billboard".

The “Billboards” tab allows the user to browse the created billboards. They can click on a billboard to edit it, if they are the user that created them, or have the relevant permission.



On the edit billboard window, the user can choose to edit the billboard, delete it, or preview it.



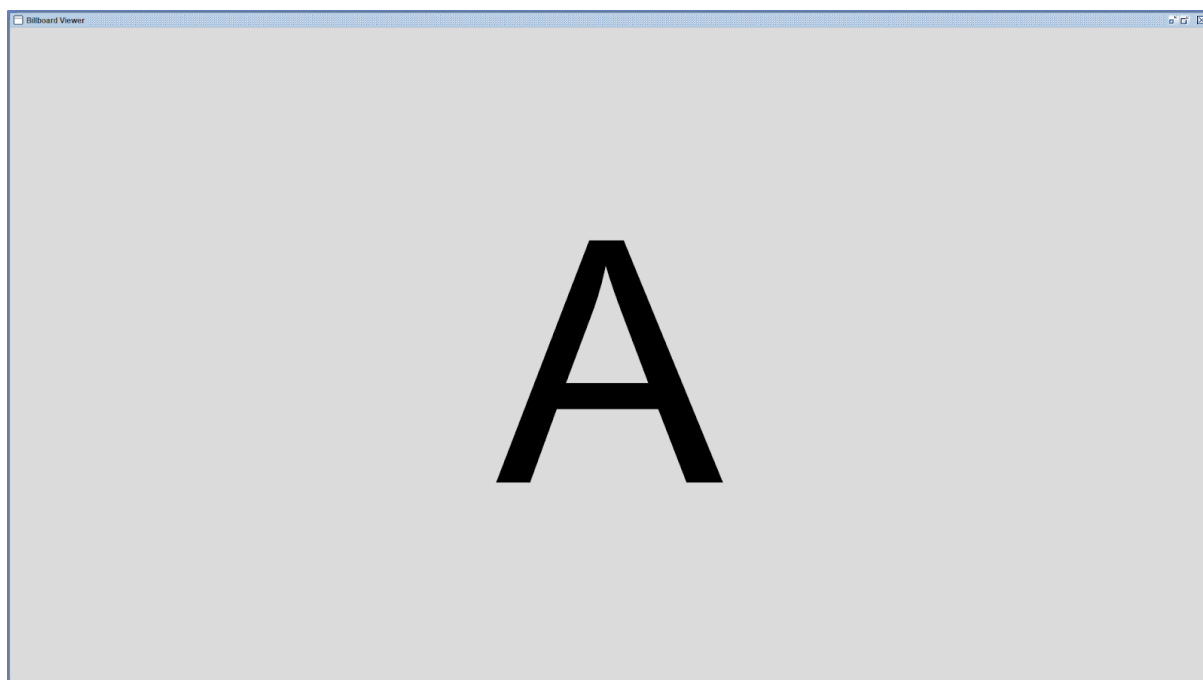
\* = at least one required

Name:*	Test
Background colour (hex):	#dbdbdb
Message:*	A
Message colour (hex):	#000000
Picture (Base64 Data/URL):*	
Information:	
Information colour (hex):	#000000

Edit Billboard

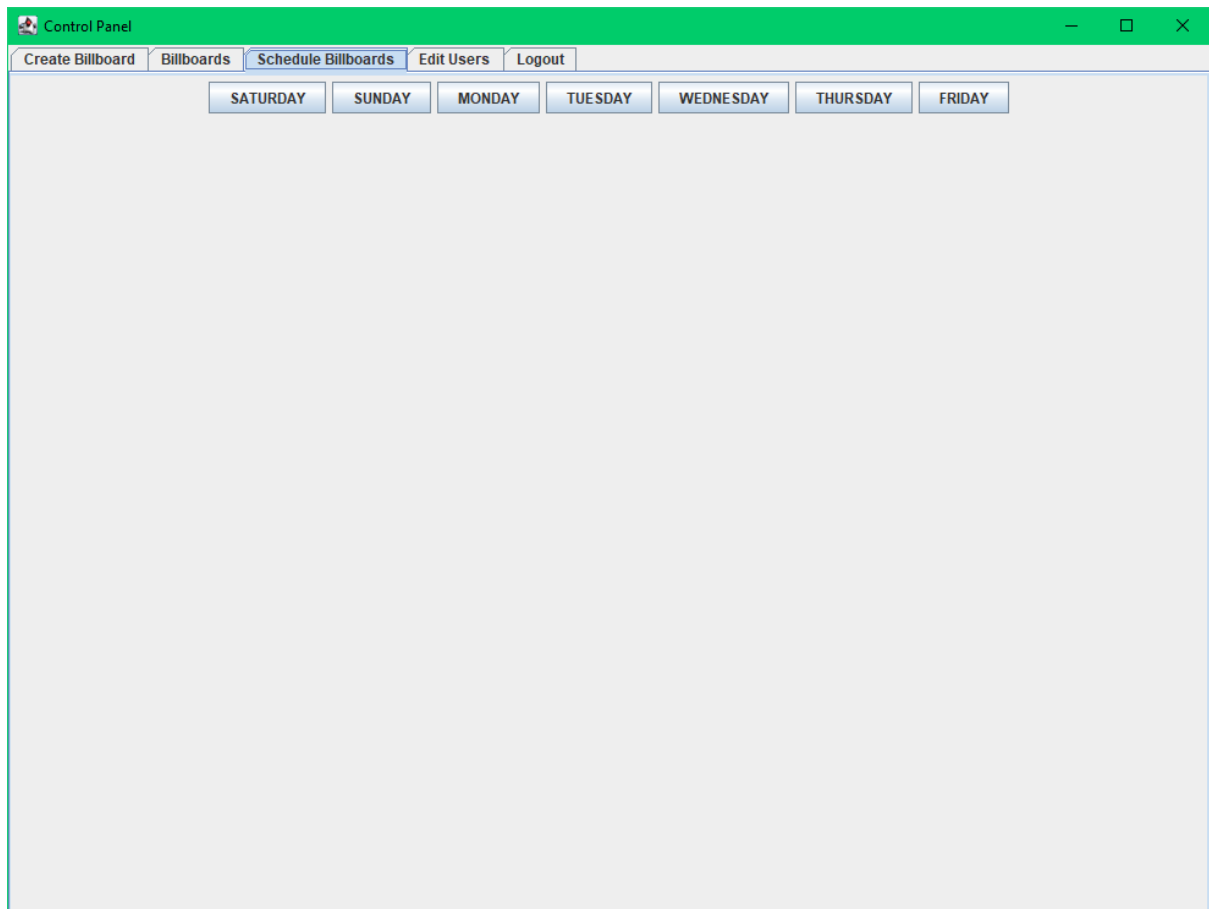
Delete Billboard

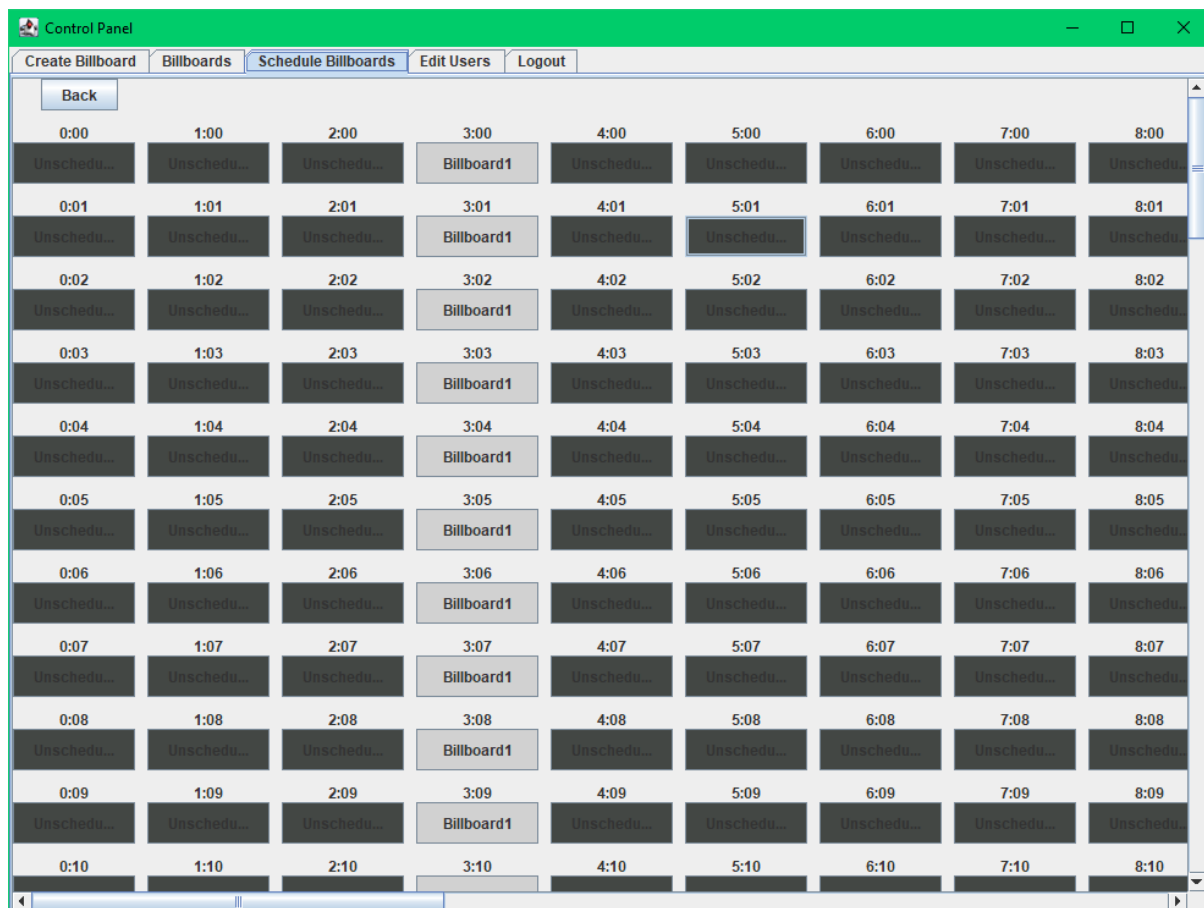
Preview Billboard



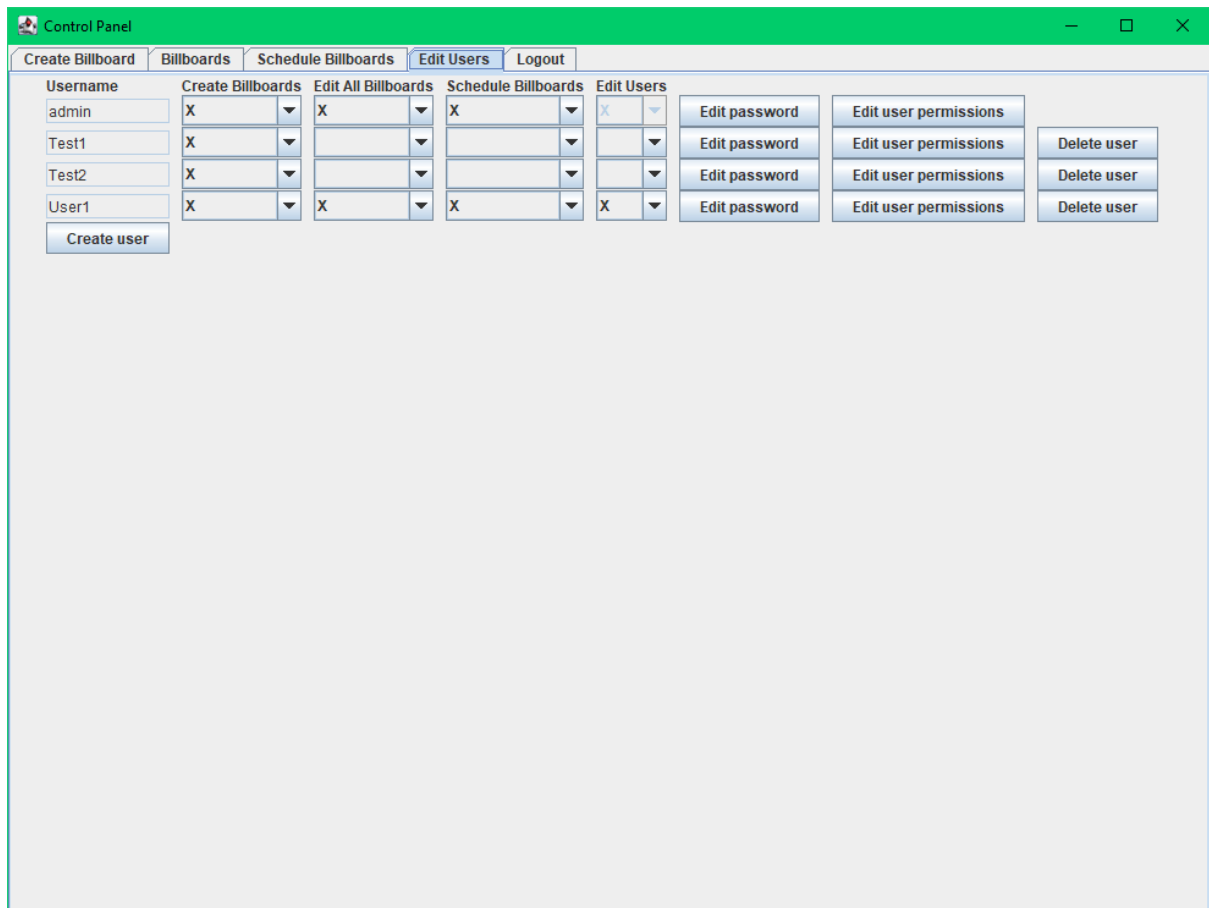
The “Schedule Billboards” tab allows the user to select a day from the next week to browse the schedule of. Once a day has been selected, a grid containing the scheduled billboards will be displayed. The columns represent the hours of the day, going from 00:00 to 23:00, and the

rows being minutes in each hour. The user can click on any grid square to either schedule a new billboard at that time slot or to delete an existing billboard scheduled.





The "Edit Users" tab allows the user to edit their password if they don't have permissions to change other user's details. If they do have the permissions the user will be presented with a list of users and their permissions. They can change other user's permissions as well as their own except their edit other users' permission. They can also change other user's passwords as well as their own. They also have the option to create a new user.

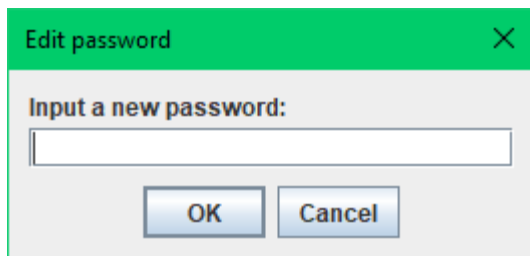


Control Panel

Create Billboard Billboards Schedule Billboards **Edit Users** Logout

Username	Create Billboards	Edit All Billboards	Schedule Billboards	Edit Users			
admin	X	X	X	X	Edit password	Edit user permissions	
Test1	X				Edit password	Edit user permissions	Delete user
Test2	X				Edit password	Edit user permissions	Delete user
User1	X	X	X	X	Edit password	Edit user permissions	Delete user

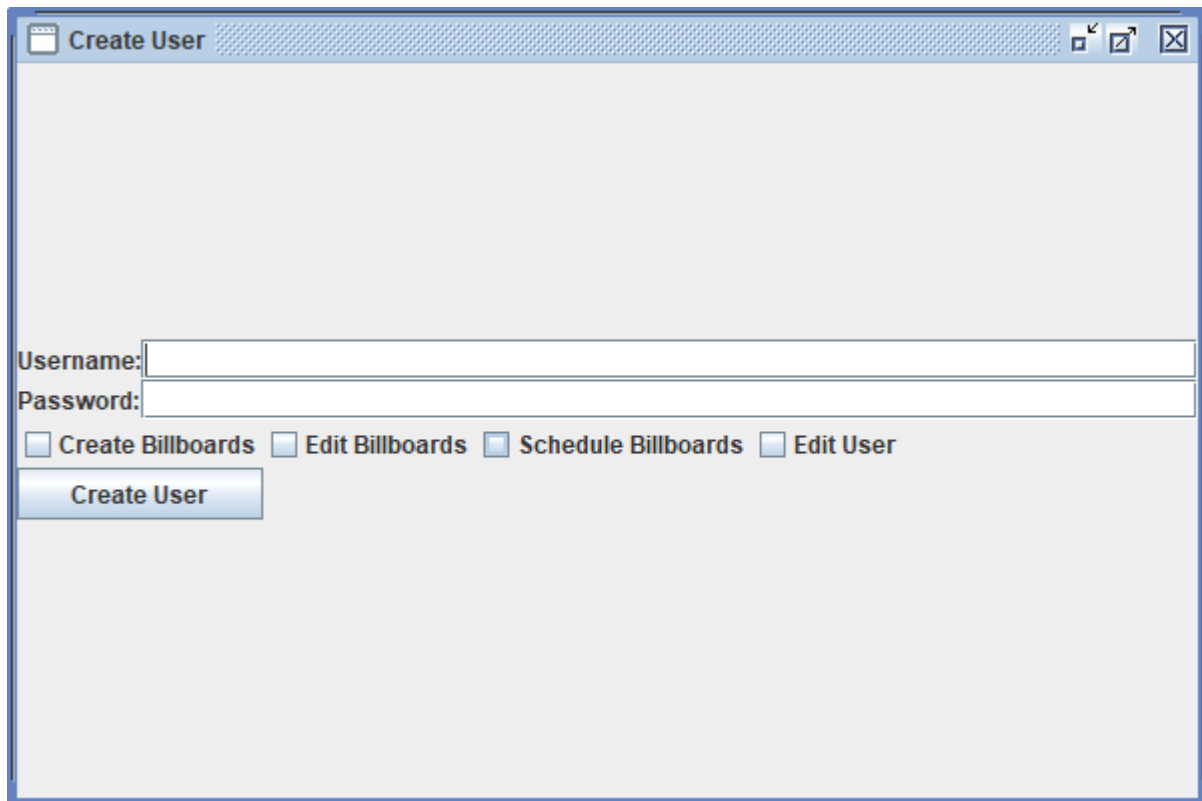
Create user



Edit password

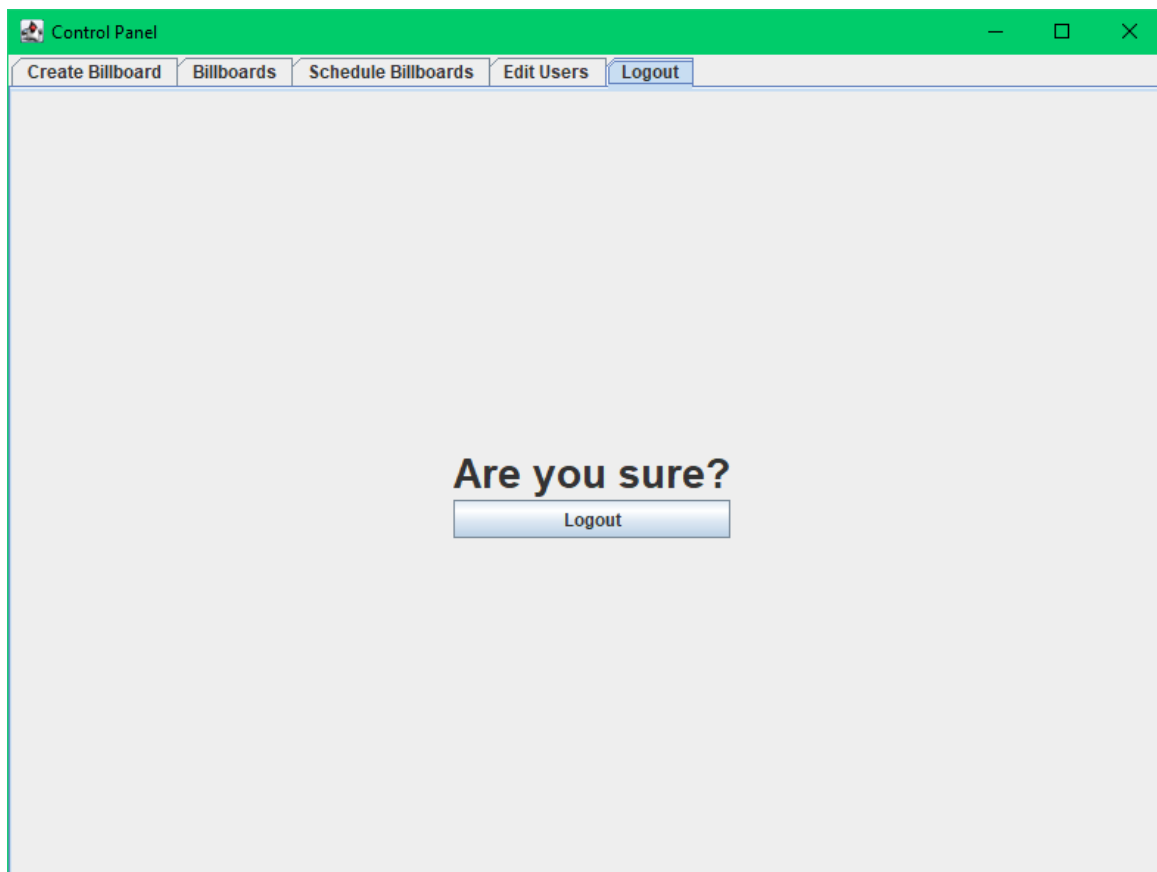
Input a new password:

OK Cancel



A screenshot of a web application window titled "Create User". The window has a light blue header bar with the title and standard window controls (minimize, maximize, close). The main content area is light gray. It contains two text input fields: "Username:" and "Password:". Below these fields are four checkboxes: "Create Billboards", "Edit Billboards", "Schedule Billboards", and "Edit User". At the bottom left of the form is a blue button labeled "Create User".

The "Logout" tab simply logs the user out.



A screenshot of a web application window titled "Control Panel". The window has a green header bar with the title and standard window controls. Below the header is a tabbed interface with five tabs: "Create Billboard", "Billboards", "Schedule Billboards", "Edit Users", and "Logout". The "Logout" tab is currently selected. The main content area is light gray and displays a confirmation dialog with the text "Are you sure?" in bold. Below the text is a blue button labeled "Logout".

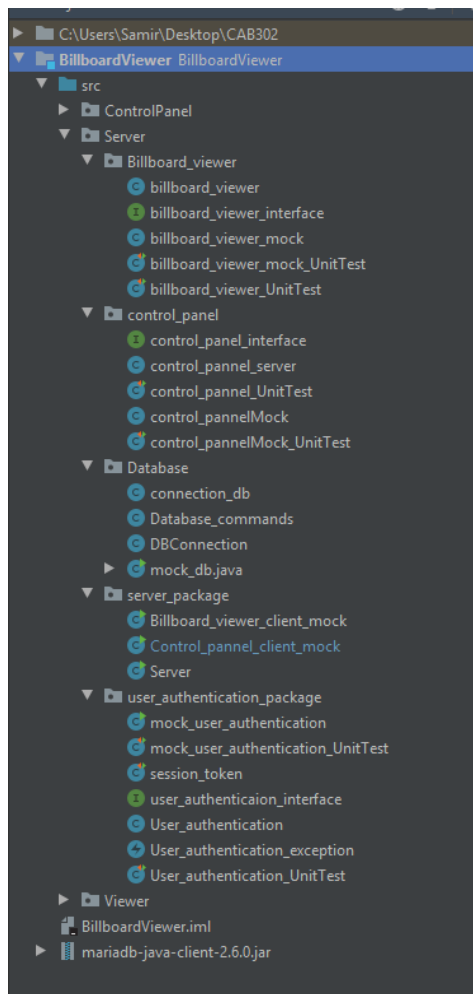
## Server

### Creating Properties File

```
jdbc.url=jdbc:mysql://localhost:3306  
jdbc.schema=mydb_v2  
jdbc.username=root  
jdbc.password=  
  
server.hostname=localhost  
server.port=12345
```

- The jdbc.url takes the URL of the database server it needs to connect to.
- The jdbc.schema take the name of the database that the server need to connect to.
- The jdbc.username is the jdbc.username and the jdbc.password is the username and password of the database server.
- The serve.hostname takes the host name.
- The serve.port takes the port number of the server.

## Setting up tests for the Control Panel



As we can see from this image that there are two unit tests done. The Unit test with mock on its name is the Unit Test for the mock class that uses mock data which doesn't require the connection to the database to run these tests. These tests were mostly used for TDD to create classes that uses data from the database

While the unit tests without the "mock", tests the functions that use the data from the database. Therefore, testing these units should be done with care as these tests changes the data in the database. These tests were used to test the function on if the methods are interacting with the database as it is supposed to. As these unit tests test the classes that interact with the database, connection to the database is required.

Setting up tests for the unit test that deal with mock data

To run unit tests for the mock data. Select a class with “uniTest” and “mock” in its name. For demonstration I will be using “mock\_user\_authentication\_UnitTest” class. As the name suggest its unit test for mock\_user\_authentication class which uses the mock data.

```

}
@Test
public void test_show_all_user_n_permission() throws User_authentication_exception {
    user_test.show_all_user();
    System.out.println(user_test.show_all_user());
    System.out.println(user_test.show_all_permissions());
}

@Test
public void test_edit_password() throws User_authentication_exception{
    user_test.edit_user_password( user_name: "Zishan Fry", new_password: "password");
}

@Test
public void test_edit_permission() throws User_authentication_exception{
    ArrayList<String> permission = new ArrayList<>(){Arrays.asList("Edit Users", "Schedule Billboards", "Create Billboards", "Edit All Billboards")};
    user_test.change_permission( user_name: "new user", permission);
}

@Test
public void test_new_table() throws User_authentication_exception{
    try {
        connection_db.check_table();
    }
    catch (Exception e){
        System.out.println("asdasd");
    }
}
}

```

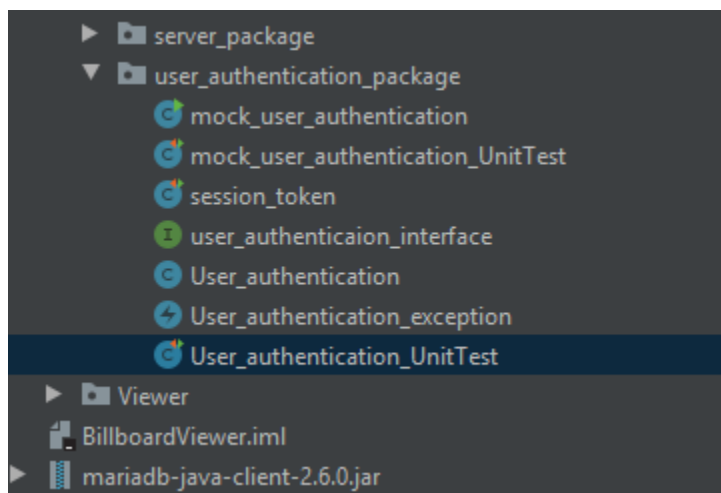
As we can see from the test method names it explains what it is testing. If we run the class, this is the result.

Test Method	Time	Output
mock_user_authentication_UnitTest	37 ms	
test_new_edit_password()	10 ms	
test_login_no_user()	2 ms	
test_edit_user()	3 ms	
test_showusernames()	1 ms	[[user_name, hashed_password, salt], [Keiron Graves, -1893898963, salt], [ Jess Philip, -1671085674, salt], [Dixie Lott, -11458989336, salt]]
test_delete_nouser()	2 ms	[[user_name, hashed_password, salt], [Keiron Graves, -1893898963, salt], [ Jess Philip, -1671085674, salt], [Dixie Lott, 1962173743, salt]]
test_delete_user()	1 ms	
test_get_user_permission()	1 ms	
test_register_permission_greater()	6 ms	[user_name, Keiron Graves, Jess Philip]
test_show_all_user()		
test_register_permission_nolist()	2 ms	
test_register_user()	3 ms	
test_login_password_fail()	1 ms	
test_show_all_permission()	1 ms	[Dixie Lott, 1, 0, 1, 1]
test_change_permission()	1 ms	
test_register_permission_wrong()	1 ms	
test_check_user_permission()	2 ms	
setUpTest()	1 ms	
test_login()		

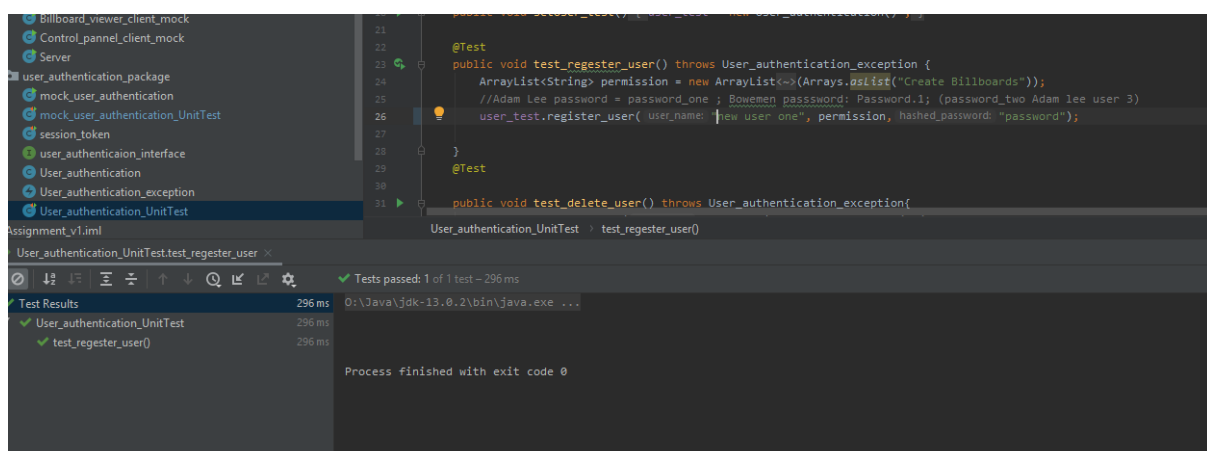


Setting up test for the unit test that deal with real data

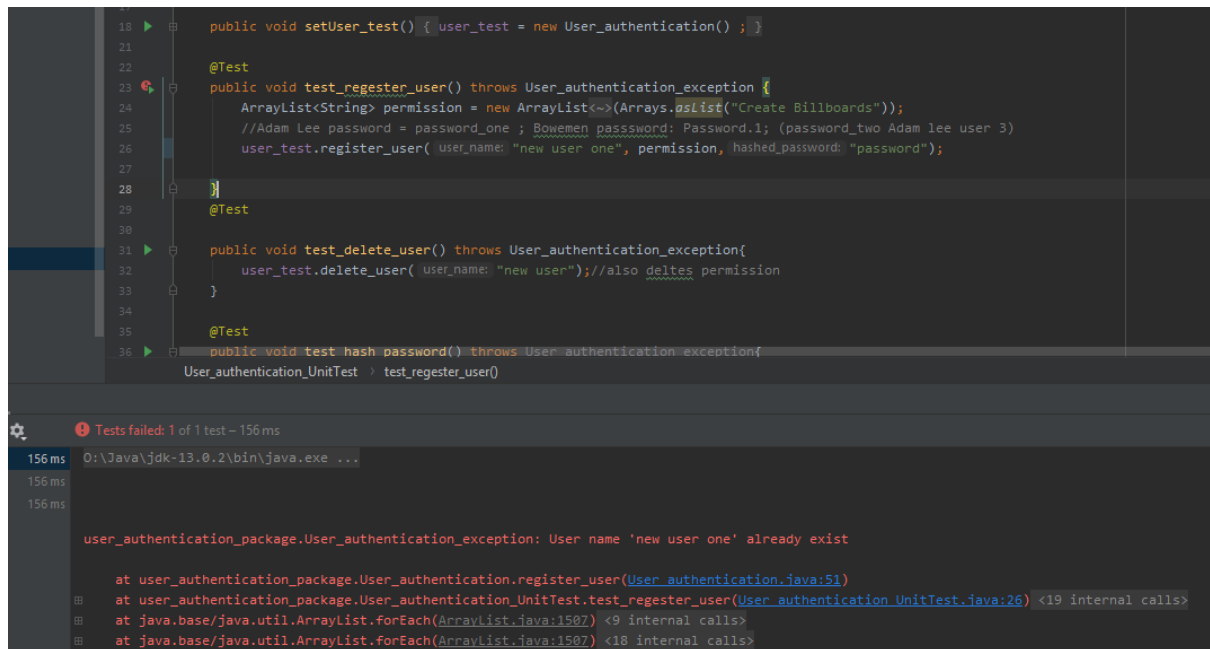
For setting up the unit test that tests classes that deals with real data, connection to the database is required. For this demonstration I will be using User\_authentication\_UnitTest to test User\_authentication



Like for the previous demonstrated unit test the methods name explain what the methods tests. But for this type of unit test, the user must ONLY run one test at a time as it changes the database and the variables, and the parameters of the tests keep changing depending on the database for example for registering a user. Lets register a user called "new user one" with some permissions and a password.



At first the test might pass but running it again will fail it as the user is already registered like shown in the image below. Therefore, running these unit tests should be done with care



```
18 public void setUser_test() { user_test = new User_authentication(); }
21
22 @Test
23 public void test_regester_user() throws User_authentication_exception {
24     ArrayList<String> permission = new ArrayList<>();
25     //Adam Lee password = password_one ; Bowemen password: Password.1; (password_two Adam lee user 3)
26     user_test.register_user( user_name: "new user one", permission, hashed_password: "password");
27 }
28
29 @Test
30
31 public void test_delete_user() throws User_authentication_exception{
32     user_test.delete_user( user_name: "new user"); //also deltes permission
33 }
34
35 @Test
36 public void test hash password() throws User_authentication_exception{
37     User_authentication_UnitTest test_regester_user()
38 }
```

Tests failed: 1 of 1 test - 156 ms

156 ms O:\Java\jdk-13.0.2\bin\java.exe ...

156 ms

156 ms

user\_authentication\_package.User\_authentication\_exception: User name 'new user one' already exist

at user\_authentication\_package.User\_authentication.register\_user(User\_authentication.java:51)

at user\_authentication\_package.User\_authentication\_UnitTest.test\_regester\_user(User\_authentication\_UnitTest.java:26) <19 internal calls>

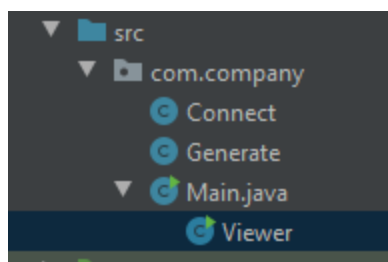
at java.base/java.util.ArrayList.forEach(ArrayList.java:1507) <9 internal calls>

at java.base/java.util.ArrayList.forEach(ArrayList.java:1507) <18 internal calls>

## Viewer

### Run viewer

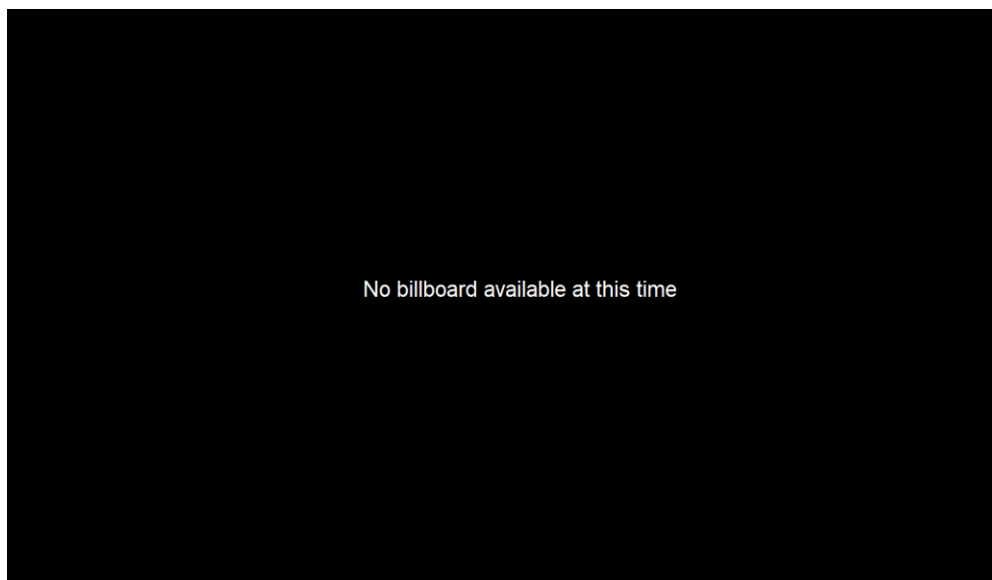
To start the viewer application, open the viewer project file and run the “Viewer” class found in Main.Java:



If a connection to the server is established and a billboard is currently scheduled, it will be displayed:



Otherwise, an error screen will be displayed instead:



[Close viewer](#)

To close the application, simply press the escape key (ESC) or click anywhere on the screen with the left or right mouse button.