

**Intelligent Manhole Cover system**

Code Document 1



2021-3-28

Wenzhou-Kean University

Rui Wang 1064083

Xuanyu Pan 1063964

Zehong Ye 1064167

Zeyu Guo 1063769

Zhihao Zhao 1064238

Index

[1 Sensor Calibration Description 2](#_Toc67866385)

[1.1 Humidity & Water Level Sensor Calibration 2](#_Toc67866386)

[1.2 Wi-Fi & GPS Sensor Calibration 2](#_Toc67866387)

[2 Database Description 2](#_Toc67866388)

[2.1 Databases 2](#_Toc67866389)

[2.2 User Class 3](#_Toc67866390)

[2.3 DBmanager Class 3](#_Toc67866391)

[2.4 UserDAO Class 3](#_Toc67866392)

[2.5 Login Class 4](#_Toc67866393)

[3 Java Utils Description 4](#_Toc67866394)

[3.1 UDP Class version 1.0 4](#_Toc67866395)

[3.2 Password Class version 0.1 4](#_Toc67866396)

# Sensor Calibration Description

## Humidity & Water Level Sensor Calibration

The C++ code for calibrating the humidity and water level sensor describes the basic function of humidity and water level sensor. From the first line to the *void setup(){}* method, some parameters are declared. In this example, we use pin ‘*2’* on the Arduino board for data entrance on the humidity sensor and use pin ‘*A5’* for the water level sensor, and set an int value ‘resval’ for holding the water level value returned from the sensor. Then, the Arduino board starts by method *void setup(){}*. In the loop method, the first part is the loop for the humidity sensor. After declaring two-byte value *temperature* and *humidity,* the loop returns the value collected by the humidity sensor with the 1-second delay because the sample rate of the humidity sensor is 1 second. The second part of the loop is for the water level sensor, which retunes the water level ‘Empty,’ ‘Low,’ ‘Medium,’ or ‘High’ by the differentiation from the if-else statements. In the further implementation, we will calibrate it to make the return value correspond to the real-time water depth such as 1 cm, instead of Low or medium.

Source code: <https://github.com/WKU-CPS4951/Intelligent-Manhole-Cover-System/blob/main/Sensor%20Test%20%26%20Calibration/humidity_water_level.ino>

## Wi-Fi & GPS Sensor Calibration

The Wi-Fi module calibration describes the connection process and retunes the device's IP address. The loop method is left empty temporally for future implementation. First, two values are declared, one is the SSID of the Wi-Fi network, the other one is the corresponding password. The setup begins with the notification states the connection process has been initiated. If the password is correctly entered, a message will be displayed, and the IP address is returned. The calibration of the GPS module hasn’t started because we don’t have a SIM card with a proper size that fits into the module, we will test and calibrate it as soon as we get a SIM card with the right size.

Source code: <https://github.com/WKU-CPS4951/Intelligent-Manhole-Cover-System/blob/main/Sensor%20Test%20%26%20Calibration/wifi.ino>

# Database Description

## Databases

The database uses **Mysql** to store information about the user to register and login, and information about the manhole cover.

Two databases are created:

**users**, and the table: **user\_Info**

**devices**, and the table: **manhole**

CREATE TABLE userInfo(

email VARCHAR(40) NOT NULL,

user\_name VARCHAR(40) NOT NULL,

password VARCHAR(40) NOT NULL,

primary key(email)

)ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE manhole(

manhole\_id INT NOT NULL AUTO\_INCREMENT,

time DATETIME NOT NULL,

temperature DOUBLE NOT NULL,

humidity DOUBLE NOT NULL,

water\_level VARCHAR(40),

Longitude DOUBLE NOT NULL,

latitude DOUBLE NOT NULL,

primary key(manhole\_id)

)ENGINE=InnoDB DEFAULT CHARSET=utf8;

The register and login function uses **Tomcat** and **Servle**t.

Clients send requests to tomcat at the port 8080. Tomcat send requests to Servlet to perform login and register actions.

To perform the functions, there are four classes applied in Servlet.

## User Class

User defines variables: email, user\_name, password, define methods to obtain and change the variables.

Source code: https://github.com/WKU-CPS4951/Intelligent-Manhole-Cover-System/blob/main/server/User.java

## DBmanager Class

DB manager connects and closes the database. Define variables: userName, passWord to store the database username and password. Define url to get the url “jdbc:mysql://120.27.146.176:3306/users” and connects to the database **users**. After this, a connection is established to connect the database. Then a method **closeAll** is created to detect the sql statement, connection, and the query result. If they are not null, then close the connection.

Source code: https://github.com/WKU-CPS4951/Intelligent-Manhole-Cover-System/blob/main/server/DBmanager.java

## UserDAO Class

UserDAO creates the mysql statement and resultSet to store the result. Use preparedStatement to execute the statement: select the entered email in user\_Info. Then create a user to store the returned email and password.

Source code: https://github.com/WKU-CPS4951/Intelligent-Manhole-Cover-System/blob/main/server/UserDAO.java

## Login Class

Login gets the email and password from the request. Then verify the information.

Source code: https://github.com/WKU-CPS4951/Intelligent-Manhole-Cover-System/blob/main/server/Login.java

# Java Utils Description

## UDP Class version 1.0

UDP contains three data fields: target IP, port and own port, as well as the corresponding getters and setters.

Before using it, you need to instantiate UDP. To send a message to a fixed address port, please call send (String[ ]) function. To listen to the fixed port, please call the bytes [ ]: receive() function.

Note that receive causes thread blocking and needs to be called in work threads. In addition, automatically requesting and releasing of a socket after each call will cause a thread conflict, so receive () no longer provides the function to automatically request or release of a socket; you need to manually call init () to get the UDP receive socket before use and release () to release the socket after you stop using it.

The internal class ReceiveThread takes the Handle parameter and uses the handler thread mechanism to implement receive, which can be used as a demo reference or called directly. The handleMessage() function needs to be overloaded in the Handler to handle the data.

Also note that you need to apply for Internet access before use. Theoretically, UDP can be used to transfer types other than String, but it should be rewritten according to your needs.

This class is based on Billy’s HCI group’s work and is used for temporary tested. In the future, the work will focus on TCP, which is safer and more stable.

https://github.com/WKU-CPS4951/Intelligent-Manhole-Cover-System/blob/main/Android-utils-code/UDP.java

## Password Class version 0.1

The Password class provides functions for encryption. It allows random generation of RSA public and private keys, Base64 encryption, and decryption of private keys with public key encryption.

The process of login authentication is:

1. the client clicks submit password and requests the public key from the server.

2. the server generates the public and private keys and transfers the public key back to the client via Base64 encryption.

3. the client returns the password account to the server via public key encryption.

4. the server decrypts it with the private key and compares it with the database.

The future work will focus on the migration to server, and the implementation of token mechanism.

https://github.com/WKU-CPS4951/Intelligent-Manhole-Cover-System/blob/main/Android-utils-code/Password.java