

**GROUP PROJECT DECLARATION**

We the undersigned confirm that the work completed on this project is all of our own work and has not been plagiarised. We also confirm that each member of the group has made a significant contribution to the successful completion of the project and that the project does not represent the work of any one individual or subset of the project team. The contribution of the team members will be reflected in the project report and also the demonstration and presentation associated with the project.

|  |  |
| --- | --- |
| Title of Project |  |
|  |  |
| Student 1 Name | Nicholas O’Donnell |
| Student 1 ID Number | P11012039 |
| Signature & Date |  |
| Student 2 Name |  |
| Student 2 ID Number |  |
| Signature & Date |  |
| Student 3 Name |  |
| Student 3 ID Number |  |
| Signature & Date |  |

|  |  |
| --- | --- |
| Project Supervisor Name |  |

Table of contents

[1 Introduction 4](#_Toc382984621)

[2 Problem Definition and Background 4](#_Toc382984622)

[3 Literature Review and Research 4](#_Toc382984623)

[4 System Design and Configuration 4](#_Toc382984624)

[5 Example of use 4](#_Toc382984625)

[6 Testing & Implementation 5](#_Toc382984626)

[7 User manual (where appropriate) 5](#_Toc382984627)

[8 Group Project Input 5](#_Toc382984628)

[9 Critical analysis and Conclusions 5](#_Toc382984629)

[10 Bibliography 6](#_Toc382984630)

[11 References 6](#_Toc382984631)

# Introduction

The purpose of this project is to provide a user friendly GUI (Graphical User Interface) to query a MYSQL database and draw the results to line charts. The database holds information parsed from excel files which are generated by water and energy monitors as well as from a weather station on campus at LIT Thurles. Once the files have been read into the database they are archived and deleted from the parent folder to preserve storage space. The project has been written in Java and JavaFX was used for the front-end design and implementation. The raw data files from the monitors will be stored on a low-powered PC in classroom E103 and the backend system for reading the files to the database will be stored on the same computer. The front-end GUI will access the database over the internet to allow for real-time reading and updating of the chart classes. A separate monitoring app will also access the database and give real time information on the most recent data.

The proposed use of this system will be to produce visually useful reports from the database and to allow an inexperienced user to produce queries without needing an understanding of coding or structured query language (SQL). The information contained within the raw data files or in the database are quite dense and do not provide an intuitive understanding of energy/water consumption and how these relate to external factors such as temperature, rainfall etc. By charting the data, useful readings are immediately identifiable and can be further investigated by month, day or hour.

We used the Eclipse IDE to work in as it was familiar to us and we had to add a number of external java archives these included jfxrt, the javafx runtime environment, poi, which allows us to read to excel files and jdbc, the java/ mysql connector.

# Problem Definition and Background

The aim of this project is to graphically display real time energy consumption over a chosen period and increase awareness to the peaks of energy use at various times of the year which will help staff and students reduce their energy consumption.

From the time we got final approval of the project to the completion date on 31st March, we needed to break the project into two main areas, the backend which will read in the various types of data being as water, electricity and weather data. We also needed a frontend which will graphically display this data.

This further aided us to break down the project objectives as follows;

* Extract useable information from electricity, water and weather files.
* Create and implement a database to store the data from the files.
* Create a zip archive to store the files once they have been read.
* Display information from the database to a GUI (Graphical User Interface).

Kevin Doherty and Paul Mackey were assigned to the file handling of electricity and water data. More specifically, Kevin was assigned to linking the frontend GUI to the backend SQL statements. Paul was assigned the role of handling the weather files and providing a mechanism for the users to view the chosen data in Excel instead of on the graph.

Nicholas O’ Donnell was in charge of researching the frontend GUI and creating the zip archive to store files once they have been passed to the database.

## Timeline

In the first two weeks, Kevin and Paul were assigned the task of researching how to make a connection to the database and reading different types of files being csv and .txt files.

Nicholas was researching how to display the information to an end user and how to implement a GUI (Graphical User Interface) for the finished project.

Once these tasks were completed, we then began implementing the research and began writing the code to read csv files and .txt files and after a discussion with Nicholas we decided to use JavaFX to create charts to display data sets.

Once the problem of reading in various files was overcome, we then began inserting this data into the database. In week 5, nick had a basic charts class that drew test data. We then began integrating the backend with the frontend and started selecting real time data and displaying it on the charts.

At this stage, we encountered problems with the design of the front end GUI because when we added real time data, the displaying of this data became unreadable. This led us to having to completely redesign the GUI. Nicholas was placed in charge of the design of the new GUI, Kevin was placed in charge of modifying the select methods to accommodate the new design by adding in the ability to group and Paul was placed in charge of dealing with missing data from the files.

When all these tasks were completed the next set of tasks were to integrate the data within the database with the newly designed GUI and also to insert the weather data into the database. Paul was placed in charge of handling the weather data while Kevin and Nick were placed in charge of connecting the MySQL data to the charts drawn by JavaFX. Once these tasks were completed, we then added additional functionality being Print Screen buttons, Exporting to Excel and the development of a real time Monitoring application. Paul and Kevin were assigned to the development of the additional buttons and selecting the data from the database and exporting it to excel, Nicholas was placed in charge of the monitoring application.

The last few weeks of the project were spent removing bugs within the code that produced inconsistent data on the charts as well as linking the monitoring application with the database. We all worked on these last few tasks and after identifying the cause of the inconsistent data being related to out MySQL queries, we were easily able to resolve the problem.

# Literature Review and Research

**JavaFX.**

As stated earlier, we implemented this project in Java, MySql and JavaFX. Throughout the course of our studies at LIT Thurles we have been using both Java and MySql, so both were a familiar starting point for the project. To implement the front-end GUI and to draw the charts we chose to use the latest version of JavaFX, 2.2.which is bundled with all versions from the Java SE 7 Runtime environment onwards. There was very little literature available on JavaFX 2.2 as it is a new technology so the main source was the Oracle website which contained background information, tutorials in many aspects of coding in JavaFX and explanations of the architecture and framework of the API. *(*[*http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm#CJACGDDE*](http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm%23CJACGDDE)*)*

JavaFX is a collection of graphics and media packages written as a java API (Application Programming Interface) which means that it can reference all the API’s available to the Java platform. This allows it to access native capabilities and connect to server-based middleware. It is intended to replace swing as the primary java GUI widget toolkit. Features of JavaFX which make it so powerful are;

* **Swing compatibility**. Much of our coding in first and second year has been in Java as such our first experience of GUI development was in Swing. The familiarity of code for controls and layouts between this and JavaFX made it easier to get started developing the front-end.
* **Cascading style sheets**. The ability to use CSS means that the look and feel of the front-end GUI can be styled separately from the application code. This allowed changes to be made to individual components or to set an overall theme which ties disparate parts of an application together. It allowed us to implement changes to the Human computer interface design at all stages of the development process without having to change any of the underlying code.
* **Hardware accelerated graphics pipeline**. Graphics are pipeline rendered in JavaFX using Prism if a machines GPU (Graphical Processing Unit) supports it. This provides quick and smooth rendering of 3D animations. If running on a low end machine JavaFX will use Swing’s 2D rendering. Accelerated media playback with low latency is also provided by a high-powered media engine.
* **User Interface controls**. A large library consisting of all the major UI controls that are needed to provide a rich interface are provided and can be styled using CSS or directly in the application code. If the correct hardware is present then the User Interface controls can be multi-touch enabled.
* **Availability**. JavaFX applications which are compiled in JDK 7(Java Development Kit) will run on all major personal computer operating systems such as Linux, Mac OS and Microsoft windows.
* **JavaFX Charts package.** While the other features outlined above were all contributing factors in why we chose JavaFX, the charts package was by far the most important for our project. The features provided by javafx.scene.chart allowed us to provide visually striking graphs onto which we could draw water, energy and weather data. A large range of graphing options were available to us such as pie charts, bar charts scatter plots and line charts. We chose to use line charts for our application as it best suited the type of information we wished to display. A data set is defined as a series and this series is drawn to the chosen chart. For our application we chose to create single series charts but multiple series can be added to a chart. Each data-set can be individually labelled and assigned a distinct colour.

**File handling and text manipulation**.

The ability to handle files and folders containing raw data was of paramount importance to the success of the project. In order to do this we relied on a number of resources. Text books such as “*How to program Java”, Deitel and Deitel (7th edition)* and “*Java Gently”, Judith Bishop (3rd edition)* provided a good overview of the basics of file and text handling in java.

However for more complicated tasks such as parsing information from csv filenames and extracting their contents or creating zip archives, we turned to the internet. The java programming language is currently an incredibly popular choice for many developers and this has led to a thriving online scene of amateurs and professionals who provide mutual aid to other java users.

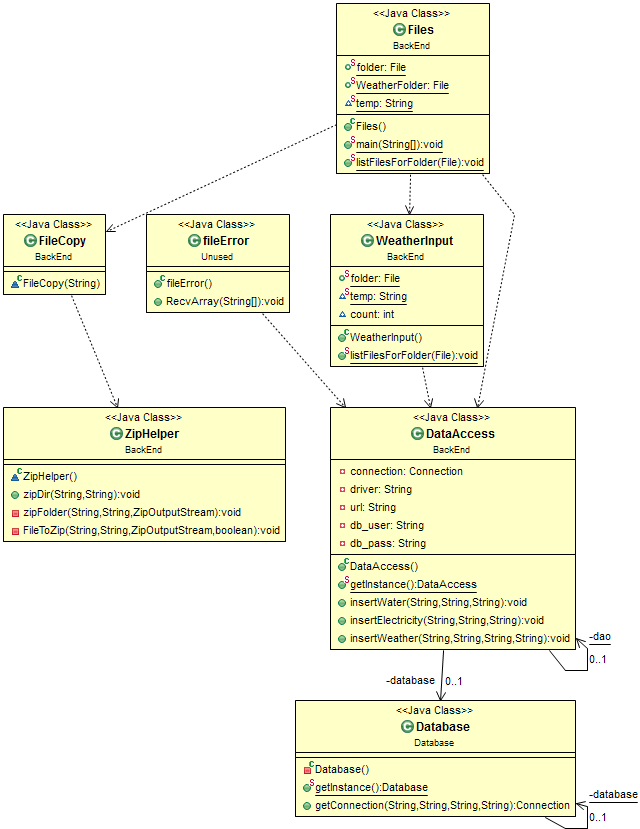
Sites such as <http://stackoverflow.com/> and <http://www.coderanch.com/> provide a forum for novice users to seek advice from more experienced coders. There are strict rules about plagiarism and these sites exist only to help with existing code or projects. A number of threads (to many to reference) on these forums enabled us to solve difficult problems which were encountered. Without this resource the project would have proved far more difficult and would have been missing some core functionality.

**MySql research.**

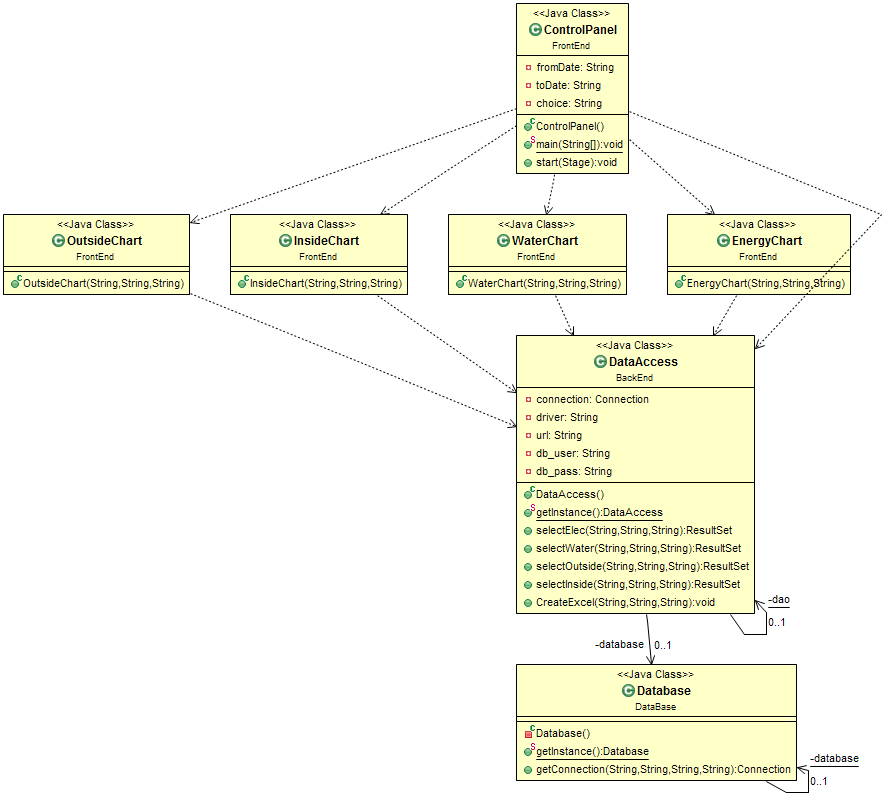
During our studies we had a semester in Database planning and design and a semester of Database implementation. For both of these subjects we were working in MySql. This gave us a good grounding in the subject but for more detailed information we used web resources like <http://www.w3schools.com/sql/default.asp> and tutorials on YouTube with a video provider called ‘The new Boston’ at <http://www.youtube.com/watch?v=1e_zEVlh_xc&list=PL32BC9C878BA72085> . The internet was our main resource for database related information because the actual database and queries were within our ability. W3schools.com was also very useful for the CSS styling of the JavaFX

# System Design and Configuration

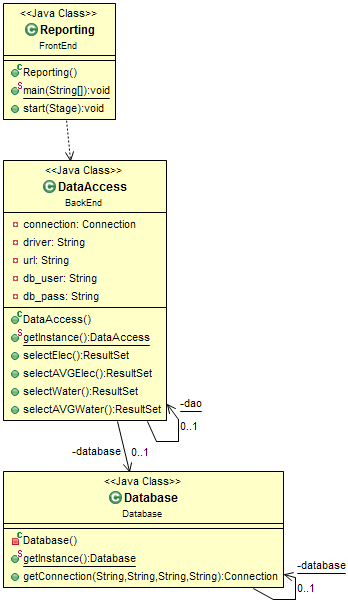
## Class Diagrams

**

* *Class Diagram of Backend Project showcasing the main class Files which handled Water and Energy files, WeatherInput which handled Weather files, the file zipper classes ZipHelper and FileCopy and the database related classes DataAccess and Database. The class diagram also shows the unused fileError class used for generating missing values.*

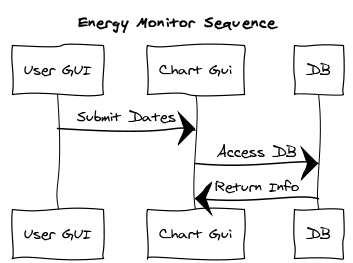


* *Class diagram of the FrontEnd project which shows the relation between the main GUI ControlPanel and the Chart classes for Water, Energy, Outside Temperature and Inside Temperature.*



* *Class diagram showing the relationships between the Reporting GUI, the DataAccess class and the Database class.*

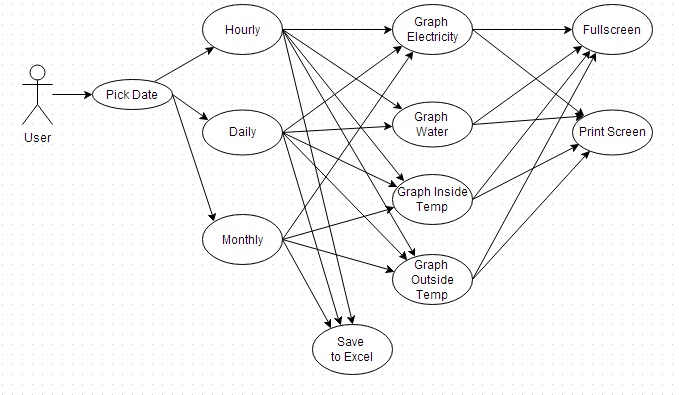
## Sequence Diagram

**

# Example of use

In practice, this system will allow staff and students to graphically view real time energy and water use and allow them to make decisions on how to reduce energy consumption within the college. The system will also allow comparisons with external factors monitored by a weather station such as temperature, rainfall etc.

It will also allow for the possibility to compare with other colleges’ energy use and enables us with the possibility to consult other colleges on how to reduce their energy consumption based on our energy performance by using this system.

****

**Figure 1.0 *Use Case***

From analysing figure 1.0 above, one possible scenario would be for the user to pick a date from 2pm to 4pm on 25th December and look at the electricity, water, inside and outside temperature. The user can then compare these values and determine whether the temperature outside is effecting the electricity usage.

**User stories**

As a user, I want to see energy, electricity, and water data graphed from 1st of February to 28th February.

As I user, I want to see energy, electricity and water data in an excel file from 1st march to 31st march.

As a user, I want to see inside and outside temp for the month of December.

As a user, I want to see electricity and water usage from 12 am to 6am.

# Testing & Implementation

## Class Functionality and Implementation

#### Back End Classes

*Files.java*

The java class ‘files’ is responsible for breaking up text and csv files into usable information and passing values to the database. The files class uses the listFilesForFolder () method to list through every file in the folder, which is designated at the beginning of the program. For each file that it reads in, it extracts the data from the file using FileReader which is then passed into a String and then split up based on new lines and spaces and stored into an array. All these elements are then inserted into the database using a method defined in the user defined class DataAccess. The File class also separates the Delta files from the other files such as Power Factor and Daily Files by containing if statements that looks for the substring Delta in each file name.

*DataAccess.java*

This version of DataAccess class contains a number of methods that insert data into the database based on Strings passed from the Files class. Both the insertWater() and insertElectricity() take three parameters from the Files class, : the name of the meter, the timestamp and the value measured by each sensor. These are then inserted into PreparedStatements which contain MySQL statements which allow the data to be inserted into the electricity and water tables in the database. The insertWeather() method takes four parameters : the name of the meter, the timestamp and the inside and outside temperature. These are all inserted into a PreparedStatement as described above and inserted into the weather table in the MySQL database.

*Database.java*

The Database class is used to establish a connection to the MySQL database using the JBDC java connector. It does this by taking in four parameters being : the driver , the URL (which contains the IP address of the database, the port number and the name of the database), the user and the password.

*WeatherInput.java*

The WeatherInput class is called after both the electricity and water data is finished being inserted. It operates the same way as the Files class, by inserting each file in the folder one at a time. It begins by reading a LineNumber using BufferedReader, this prevents the program from inserting duplicates as it will only insert once the counter in the program exceeds the line number in the file. Once the program reaches the last entry for the month, it writes the file name to a separate document, this prevents the program from reading duplicate entries.

*FileCopy.java*

This class takes a String containing the file name as a parameter. This file is then copied and deleted from the parent directory to a folder containing the file type, month and year. When a new monthly folder is created, the previous month is passed to the ZipHelper class.

*ZipHelper.java*

Java does not provide methods for zipping folders, this program allows files and folders to be archived.

FileError

This class accounts for the fact that some data may be missing and resolves the problem by generating an average of what data it has received. This class has since been made redundant due to causing errors when more than one line of data is missing.

#### FrontEnd Classes

*ControlPanel.java*

The Control Panel is the front end GUI presented to the user. It takes input from combo boxes, a radio group and UI controls and converts the values to strings. These values are then used to construct MySQL queries. These values are locked once the submit button is pressed. Once the values have been submitted, they can be passed to the respective chart class selected by the user. The button controls allow the user to choose which chart to display. There is also a Save Data button which allows the user to export data to an excel file.

*Chart Classes*

Using the JavaFX charts library provided a means to draw charts containing data sets as series. These series were drawn dynamically from information provided by the database and requested by a query formulated in the FrontEnd GUI, ControlPanel. The values passed from ControlPanel are then passed into the respective methods within the front end version of DataAccess which has MySQL select statements that return ResultSets which contain the data to be drawn to the chart.

*DataAccess.java*

This version of DataAccess contains methods for selecting data from the database rather than inserting. For each data type there is a method that takes in the three parameters from the ControlPanel GUI being: the “From” date, the “To” date and the “Choice” variable which is used to group the data by hour, day or month. The “Choice” variable determines which meway the data is grouped by. The “From” and “To” values are set in a PreparedStatement containing the MySQL statement for hourly, daily or monthly data and executed. The data queried from the database is then passed into a ResultSet and returned back to a chart to be graphed.

*Database.java*

Carries out the same functionality as described above.

#### Reporting App Classes

*Reporting.java*

The reporting application is used to give persistent feedback on the college energy and water usage by getting the mean usage for all relevant data in the database and showing our current usage. If the current values are five percent below the average this is the safe zone and the text colour is set to green. If the value is within the five percent of the average the text colour is set to orange as a warning. If the value is 5 percent over the average then the text colour is set to red showing that it has greatly exceeded the acceptable range. These values are constantly updating using a Thread. The DataAccess class methods are used to access the necessary data.

*DataAccess.java*

This version of DataAccess contains methods similar to those described previously except they are modified to only return a single value each time, the average water value, the most recent water value, the average energy value and the most recent energy value.

*Database.java*

Same functionality as described before.

## Accomplishments

We successfully completed all of our main tasks set for the project. While there were peripheral goals we would have liked to achieve, we were happy that we hit all our targets that we set at the start. We were able to create three separate programs being the Reporting application (ControlPanel.java and Charts), Monitoring application (Reporting.java) and the BackEnd applications (DataAccess, Files, ZipHelper etc). All the classes we created were able carry out their intended purpose and were also designed in a way that allowed them to be easily expanded on to add additional functionality.

## Problems encountered

### Redesign of ControlPanel GUI

In the original version of the FrontEnd GUI, all the graphs would be drawn to the screen within the same window. (Where the image is now) This proved problematic as the amount of space provided for each graph would be reduced dramatically if all four graphs were present at once, leading to the data being graphed being unreadable as it was so small. This meant that we would have to redesign the GUI entirely to merely act as a Control Panel that would then draw graphs in separate windows each time a chart button was pressed. This made it easier to compare certain dates against each other which was impossible to do previously unless a screenshot was taken as the charts were redrawn every time.

### Redesign of Files class

When we first began the project we incorrectly assumed that the data from the censors would be arriving once an hour and contained only one row of data. Due to the Delta File actually containing four rows in each file, we would have to re-write the files class entirely to utilize two loops that would loop through the folder to look at each file and then loop through each file and insert the rows one at a time.

### Text Fields in ControlPanel GUI

Another problem we encountered when designing the ControlPanel GUI was caused by our use of text fields for taking in user input. We would have to ensure that only date related information was entered, meaning that we would have to have a massive amount of if statements to ensure that no letters were entered or attempts at inserting MySQL injections were made. This was resolved by replacing the text fields with combo boxes which limited the user to only being able to set dates by selecting from a list of values, which was the desired functionality.

## Testing

White Box Testing

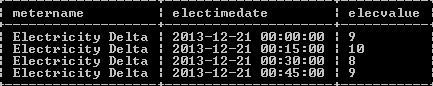
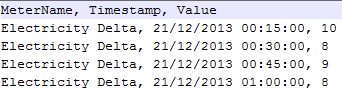
White box testing involves analysing the code to identify errors within the program. As our programs utilized a number of loops for looping through the files and using their file names to access their data, we would use print statements to test if we had entered certain loops and to test that we were receiving the correct data from the files. This would give us a good idea of the data being handled at specific points in the code e.x the file being read from and its contents and also showed if the program was entering loops correctly. An example test for the File handler class would be to insert three or four files into a folder run the program. A number of print statements should then be able to print out the name of the file being read, its contents and whether it had been inserted to the database or not. If all the correct information was printed and the data was found in the database then we could remove the print statements and test other areas of the program.

When testing the GUI interfaces of the project, we used the same methods. We would use print statements to test the various handlers for the buttons and the combo boxes. An example of this would be putting a print statement inside the Submit button. This statement would then print out the From date and To date as well as the choice of grouping when then user presses submit. This was also necessary to test if the dates were valid and that the To date was not greater than the From date. Print statements were used to test what if statements the program had entered after the user pressed submit to show if the dates were valid or not. If the dates were valid then the three variables would be set and printed out and if they weren’t then the variables would not be set and the print statement would state that the dates were incorrect. We later used a text field on the GUI to show that the dates were not valid in order to notify the user.

Black Box Testing

*The File Handler*

While designing the project we would perform a black box test on the current build weekly in order to ensure that the different classes within the project were running the way they were intended to. We carried out a number of test cases to make sure that the program would carry out the functionality that it was intended to do and produce the correct output without actually looking at the code. These types of tests should be able to be carried out by anyone, whether they have code experience or not. In the beginning when we were working on getting the data into the database we would run our code and then examine the output in the database. If the data was incorrect we would make alterations to the code until the data finally formatted correctly and matched the raw data in the original files.



* *These images show what the output should be if the data from the files was inserted correctly after carrying out a test case. All the values for 21-12-2013 at midnight were present in the database.*

*The Reporting Application*

In terms of the reporting application, tests cases involving inserting random data and graphing it would be carried out. Earlier versions of the application utilized text boxes instead of the combo boxes found in the final product. To test these we entered queries using the text boxes to select data from the database. We were able to plot the data but the text boxes also allowed people to enter whatever they wanted into the text fields, sometimes causing the program to crash. This in the end led us to changing the overall design for user input to incorporate combo boxes which limited what users could enter, thus preventing MySQL injections, unsupported date formats and also making it easier to use the program overall. An example test for both versions of the GUI would be entering a number of dates and attempting to graph them. We would then analyse the graph that was generated by the program and compare it to readings found in the original files and the database. If the data on the graph was consistent with the data found in both the database and the .csv files then the test was a success and the program was working correctly. If the data was incorrect then analysis of the code or the MySQL statements would have to be carried out and changes implemented to try and produce the correct output.

*Monitoring Application*

The functionality of the monitoring application was the easiest to test. The test case for the Monitoring app involved comparing the most recent value displayed on the screen against the most recent value in the database as well as comparing the average values. The colours of the current value must also be the correct colour depending on whether it is greater than or less than the average value. To test if the values were in the correct threshold of 5 percent we simply tested the values on a calculator. If the Monitoring app always contained the correct values and the text fields were the appropriate colour after the database was updated then the monitoring apps functionality was complete.

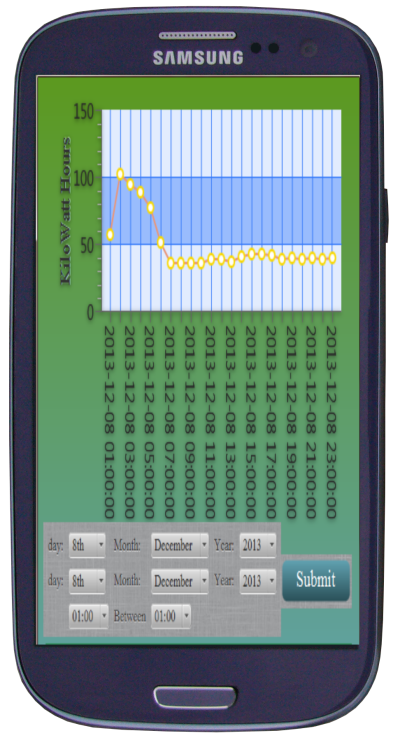
Stress Testing

Due to our project using live data to graph and export information we would have to perform a number of stress tests on the database and the file handler class. This was done by placing thousands of files in the one folder and making the program work with them. The file handler would have to be able to handle reading the data from the files and inserting it without crashing. We found that the program ran reasonably well, being able to insert thousands of files without crashing though once the type of file was changed suddenly the program would break. Luckily the File handler would never have to deal with this scenario as it only starts running once an hour and would only have to insert 2 files every time being one energy file and one water file.

The weather file handler also had to be stress tested but thanks to the way it was implemented it would not have to deal with thousands of files as each file is constantly updated instead of new files coming in every hour, this means that there will be way less files to deal with, reducing the amount of work the program would have to carry out. The program could be tested by running it and getting it to insert a number of monthly data files. The program ran well as it only had to insert all the data from a file and once it was completed then this file would never be looked at again. When a file is completed its name is stored in a file which would allow the program to ignore it when the Weather file handler is started. This means that only one file will have to be worked on at a time and it will only have to insert six rows of data to the database each time it is ran every hour.

## Future Improvements and Features

Due to time constraints we weren’t able to branch into other areas of functionality. Given the time there was a number of features we would have liked to implement. The main one was a mobile application for Android phones. The application would provide the user with a lightweight version of the reporting app that would allow staff and students on campus to get information about water,energy and weather data and display it on their phones. The app could also notify facilities management if a sensor is producing massive readings.



*This diagram shows up a mockup of the Mobile app version of the project running on an android phone.*

### Daily and Power Factor Files

Due to time constraints we only used the data from the Delta Files into the database. If we had more time we would have stored data from other files such as the Daily files and Power Factor files for additional functionality or simply for testing purposes. We decided early on to not utilize these files as they contained less useful data and time would have been wasted trying to modify the File handler to read and insert from these files. Examples of use for these files could be comparing the values contained within them against the data already contained in the database, the Delta and Daily files are a prime example as the Daily files contain a sum of all the Delta files for a particular day. Seeing as we had access to the Delta files we could simply generate this information instead of having to read it from the Daily files.

### Additional Weather File usage

In the current system we are only using two of the fields found within the Weather files, the Inside Temperature and the Outside Temperature. The weather files had up to 20 fields in total which contain data such as rain fall, Solar Radiation and Humidity. The reason we left out all these fields was largely due to time constraints but if we had more time we could have added them into the system and allowed people to access them on graphs. Adding this additional data would also have allowed us to create a weather GUI and display all the information at once and show all the data relates to each other.

### Specific Queries

With the current GUI the user is able to get all data between two different dates/time frames. One example of how this could be expanded further would provide an option to select a time frame and two dates and select all data between these two times for each day between the two dates. This would allow users to compare the energy or water use between these times in order to identify areas where energy or water could be saved.

### MySQL Statements for creating Database and Tables

Instead of the user having to set up the Database and tables before they ran the program for the first time, methods could be created to set up the database if it was not present. This is easily achievable thanks to PreparedStatements as they carry out any MySQL statement.

### Alerts

Similar to the notifications found in the possible mobile app, the Monitoring application could be modified to send email alerts to the facilities staff to inform them when excessive data or water has been used in the last 15 minutes. This could allow the staff to look for the problem and return the usage levels to the average use.

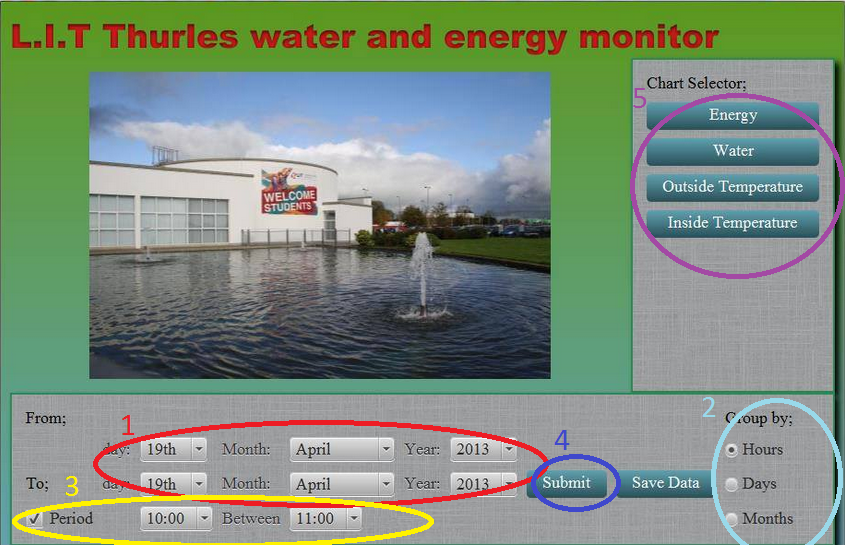
### Multiple Series on a Single Chart

Another useful feature that could have been implemented if there was more time was having multiple series lines on a single chart allowing a user to compare different data types against each other. For example a user could compare water usage to the outside temperature, this would be particularly useful when analysing the warmer times of the year.

# User manual (where appropriate)

The user will be provided with an executable jar that can be launched directly from a desktop icon.

## Using The GUI

****

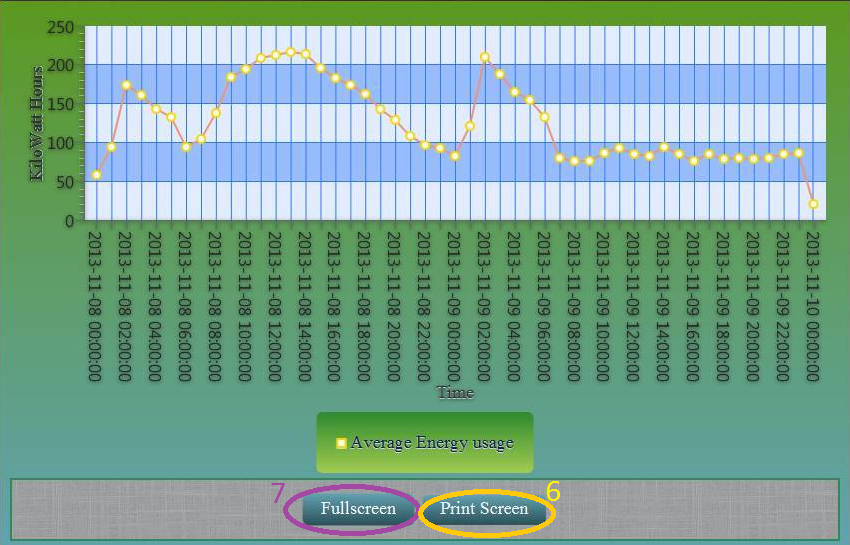
**Step 1 –** Choose a date range for which you want to see data for. “From” is the earliest date you wish to see the data for. “To” is the latest date you wish to see the data for.

**Step 2 –** Choose date range to be displayed in Hours, Days or Months.

**Step 3** – If you have chosen hours from step 2, choose particular hours during that range.

**Step 4** – Submit data. – Ensure “From” date is earlier than “To” date.

**Step 5** – Choose a particular type of data you would like to view.

****

**Step 6 –** This option will provide a screen shot of the presented graph.

**Step 7 –** This option will expand the window to full screen.

**Exporting data to Excel**

To export the chosen range to excel, Follow steps 1 to 3 as mentioned above and choose Save Data. This will provide a menu as to where to save this data in your directory.

## How to add new weather data using rainfall as an example

This example will illustrate how to add new weather category to the overall project. In this example we will be using rainfall but this value can be substituted for any value the user wishes to add.

**Step 1**

Launch MySQL and sign in. Password = password. (Change as appropriate).

Sign into the correct workspace with “Use assignment;”

Create a new table in the database called RainFall.

Create table Rainfall (Rainmeter Varchar(10), raintime datetime, reading varchar(10));

**Step 2**

Open WeatherInput class

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **0** Date | **1** Time | **2** Outside Temp | **3** Outside Humidity | **4** Dew Point |
| **5** Wind Speed | **6** Wind Gust | **7** Average Wind Bearing | **8** Rainfall Rate | **9** Total Rainfall |
| **10** Sea Pressure Level | **11**Rainfall Counter | **12** Inside Temp | **13**Inside Humidity | **14** Latest Gust |
| **15** Wind Chill | **16** Heat Index | **17** Uv Index | **18** Solar Radiation | **19** ET |

Lines 111 and 118 contain the strings that point to the data to be inserted. Add a new line here. In this case were adding rainfall so we are choosing position 9 as seen above and adding this to a string. Eg. *String TotalRainfall=valuearray[9];*

**Step 3**

Then go to the Backend version and go to the DataAccess class and create a new statement that will insert into the database**.** An example statement for our rainfall example is seen below.

**public** **void** InsertToRain(String one, String two,String three) {

**try** {

PreparedStatement ps = connection.prepareStatement("INSERT into RainFall(Rainmeter, raintime, reading) VALUES(?, STR\_TO\_DATE(?, '%d/%m/%Y %k:%i:%s'), ?);");

ps.setString(1, one);

ps.setString(2, two);

ps.setString(3, three);

ps.execute();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**Step 4**

Next go to the weatherInput class and go to line 127, and insert a new line which will call this statement and insert the values to the database. In this example we will be adding inn.InsertToRain (tablename, DateTime*, TotalRainfall*);

## How to add new data for a new monitor

For this example we will be adding a new heat meter.

**Step 1**

Create a new heat meter table in the database. For this example we will call the table HeatMeter eg, Create table HeatMeter(MeterName Varchar(10), Metertime datetime, reading varchar(10));

**Step 2**

Go to the backend version and go to the DataAccess class and create a statement that will insert into the heat meter table, eg,

**public** **void** InsertToHeat(String one, String two,String three) {

**try** {

PreparedStatement ps = connection.prepareStatement("INSERT into HeatMeter (MeterName, Metertime, reading) VALUES(?, STR\_TO\_DATE(?, '%d/%m/%Y %k:%i:%s'), ?);");

ps.setString(1, one);

ps.setString(2, two);

ps.setString(3, three);

ps.execute();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**Step 3**

Go to line 82 in the files class and add a new line which will insert the heat meter readings to the heat meter database. For this example, we will be adding

**Else if**(name.contains("heat"))

{

in. InsertToHeat(name, date, value);

FileCopy fc = **new** FileCopy(*temp*);

}

## How to Add Buttons to GUI

At line 68 in the ControlPanel class, the Vbox container “buttonbox” is declared. This is a container that stacks buttons vertically and any new buttons should be added here. Declare a button to add to the button box at line 127.

Button heatBtn = **new** Button();

energyBtn.setMaxWidth(Double.*MAX\_VALUE*);

energyBtn.setText("Heat");

energyBtn.setOnAction(**new** EventHandler<ActionEvent>() {

**public** **void** handle(ActionEvent event) {

**new** HeatChart(fromDate, toDate, choice);

//primaryStage.toFront();

}

});

buttonBox.getChildren().addAll(select,energyBtn,waterBtn,tempBtn,rainBtn, heatBtn);

## Creating a Chart

To create a new Charts class, simply copy the code from any of the other charts classes and change rename anything to do with the type of reading eg.

stage.setTitle("Energy Line Chart"); = stage.setTitle(“Heat Line Chart”);

DataAccess energy = **new** DataAccess();

ResultSet energyrs = energy.selectElec(fromDate, toDate, choice);

DataAccess heat = **new** DataAccess();

ResultSet heatrs = heat.selectHeat(fromDate, toDate, choice);

The selectHeat method must be defined within the DataAccess class which will be described in the next section.

## Modifying DataAccess

To create a method for selecting heat, simply copy the code for another type of data within the DataAccess class and change the MySQL statements within the PreparedStatements to select from the new table.

Example

PreparedStatement ps = connection.prepareStatement("select DISTINCT metername , metertime, SUM(reading) from heatmeter where metertime >= ? AND metertime <= ? GROUP BY YEAR(metertime), DAY(metertime), HOUR(metertime), MONTH(metertime) ASC; ");

# Group Project Input

## Kevin Doherty

My main goals of the project were handling the Water and Energy files in terms of extracting data from the raw .csv files and inserting them into the database as well as linking the FrontEnd GUI’s with the backend MySQL statements.

##### Water and File Handling Class

My first task was to ensure that we were able to read from multiple files, one at a time and insert every line into the database. I achieved this by creating a loop that would read in the names of every file contained within a folder and then inserted the four different rows contained within the file into the database.

##### MySQL Classes and linking with Reporting App

To insert the data into the database I had to create a class that would contain all the methods for inserting into the database. I learned that this could be achieved by using PreparedStatements which allow you to carry out MySQL statements and also allow you to set specific areas within the string to insert variables into the statement. This proved useful as it allowed me to pass the variables containing sensor data into the MySQL statements which then inserted them. Once the first version of the Reporting GUI was completed it was my task to write a class that would contain a number of methods that also used PreparedStatements to select data from the database and graph this data. I would have to link the ResultSet data containing the information from the database with the chart classes. This involved getting the distance between the first and last positions in the result set which would allow me to set the overall size of the JavaFX graph. I then had to create a loop that would set all the array positions for the chart series to the data contained within the ResultSet. Another feature I implemented was a Screenshot button which would allow the user to take screenshots of charts that were drawn. This also gave me a chance to test the FileChooser class which allowed us to select the location where a file we generated could be saved. This would later prove useful for exporting the Excel files.

##### Monitoring Application (Reporting.java) link with MySQL

Another one of my tasks was to also link the Monitoring app with the database and produce the desired output. This was done by utilizing a similar class similar to the one described before but in this case the class would return the average and most recent values from the database. I would then have to constantly retrieve the most up to date values from the database as the Monitoring app is a live application, this was achieved by implemented a Thread which would be able to constantly retrieve the most recent value for both energy and water and the average values for each. The most recent values would then be compared within the thread and if the most recent value was greater or less than the average it would be set to green or red on the GUI depending on the situation.

##### Stress Testing and bug fixes

Another one of my tasks would be to carry out stress tests on the file handling classes and the chart classes. For the file handling classes I made the programs try to insert a massive amount of files to judge if the program would crash under stress and I also carried out massive queries using Reporting app to test if the programs could graph a large amount of data at once. I also aided the rest of the team when it came to identifying bugs and helping when it came to implementing different features.

Working as part of a team

I found working as part of a team to be a greatly enjoyable experience, due to the fact that everyone on the team carried their own weight and helped each other when someone ran into a problem. It gave me a chance to develop my team working skills, as I have rarely been placed in a position like this before so it gave me a new perspective on developing software.

## Nicholas O Donnell

**Tasks**

For this project I took responsibility for archiving all files once they are read in to the database. I also investigating how to design and then implement the front-end GUI which would provide input controls to the user and allow us to dynamically draw data-sets. All parts of the project that were assigned to me were included in the final project hand up and I believe they had a positive contribution.

I was chosen to take on the graphical element of the project because in the first term of 3rd year, as a student of computing rather than smart energy systems, I had studied both mobile application development and computer graphics.

During the research phase I identified the Java integrated API, JavaFx, as a potential candidate for the designing a graphically rich display for our project. At the time JavaFX was a new technology and any documentation that I could find was contradictory and confusing. This was due, in part, to the fact that any books that had been published were usually on JavaFX 1, which was designed as a scripting language to be embedded in java applications. Whereas JavaFX 2 onwards was a stand-alone API which could run alongside other java libraries (since the end of this project JavaFX has been updated again to version 8 which is deployed with Java SE 8). The main attraction of JavaFX was its graphics and charts libraries which would allow us to create a visually impressive front-end application.

**Team work**

I was aware that it was a risk using JavaFX for our project because of the lack of literature and other information so I brought it before the group to make a decision. Having reviewed the tutorials and information provided by the Oracle website we all agreed that JavaFX strengths outweighed its weaknesses and that the project as a whole would benefit from its use.

All decisions made by the group including how work would be assigned and the direction the project would take, were reached by discussion rather than any one person taking a leadership role. This worked well within a small group but might not have been appropriate in a larger setting. We all had input into every aspect of the project, not just those that were assigned to us and this helped to establish a sense of ownership and pride in the overall product.

Working closely with both Paul and Kevin while designing the front-end application was essential in order to construct a GUI that could collect information in a format that would be of the most use when constructing queries for the database back-end application.

**Reflections on group projects**

Generally, I would have had a negative opinion of group work as a whole, however having worked with a team that has a shared goal, dedication and enthusiasm to produce a good project, I have revised my opinion. This has been a largely positive experience and has given me an insight into how a project moves from a loose idea on paper to a full realised end-product.

## Paul Mackey

One potential issue I suggested about the water and electricity files being received is if for some reason, the system went down for a certain length of time resulting is missing data. At the time I felt this would cause issues with the graphing of data and could result in inaccurate graphs.

Based on this, one of my roles was to come up with a solution for this issue. I came up with a mechanism that will recognise when there is missing data such as what type of data is missing, what time is missing and also get the average of the values that have been received and fill in the blanks.

To do this, I needed to alter the file reader class for water and electricity and if there has been less than 4 values received, It will insert into the database what values have been received and call a class that can generate the missing line.

I also felt that a particular user requirement might be to see the chosen date range in Excel as opposed to seeing it on the charts. For example, a user may graph a range of data and if there was a particular peak, they could save that data to an excel file to take a look at more accurate readings. I then created a button on JavaFx to give the user an option to save the date for their query to Excel rather than view it graphically.

Another Role that was assigned to me was to read in the weather files. This was a very important aspect in the project as the weather files contain 20 different readings and it would also be important to look at inside and outside temperature and compare that to electricity usage throughout different times of the year to see, for example, would cold weather result in higher electricity usage.

These files contain 10 minute interval readings and are sent every hour. This hourly data is appended to the same monthly file as opposed to water and electricity that have a separate file for each hour. One particular issue I came into was preventing duplicate lines. For example, after one hour of data is appended to the file and the next hour comes in, when the program runs, it will insert the previous hour and current hour into the database. To fix this issue, I needed to create a file which will store the current line number. When the program runs, it will then only insert when the count in the program exceeds the line number in the file which will prevent duplicate lines.

Another issue I then came into was preventing duplicate files being inserted. For example, when the end of October is reached and the first hour of November comes in in the NovLog, the program would insert all of October again. To fix this issue, I tried moving the file to a different location and deleting it from its current location when it reaches the last monthly reading but I was unsuccessful due to not being able to close the buffered reader before the delete function.

I then decided to create a file which will store the names of the completed files. When the program starts up and is running through the files, it will do a check of the filenames file to see has it been read yet.

Working as part of a team was also enjoyable as each member shared a common goal and were enthusiastic about the bringing the project forward. Regular meetings also proved to be invaluable as each member at some point may have been relying on another member to get their task done before they could start theirs.

# Critical analysis and Conclusions

If we use the four main objectives, as outlined in chapter 2, as benchmarks we can say that we successfully achieved our goal of delivering a set of applications that satisfy any number of use cases (see chapter 5) for the proposed use of this system. The system provides charts that show accurate data when compared to raw data files, this process is outlined in the testing section of chapter 6 and is reasonably error-free. When we demonstrated or project, the UI was intuitive to use and provided useful and usable charts that were easy to read and understand.

While the project went quite smoothly we did encounter a number problems along the way. These are outlined in chapter 6.3. Some of the problems we encountered were due to us not properly understanding the problem but pushing on with coding and becoming inflexible when it came to making changes, it is sometimes necessary to take a step back from a problem to fully understand it and be open to constructive criticism. Any large project will inevitably run into problems but overcoming them is part of the learning process.

Some of the problems that we encountered were from lack of planning. We had not prepared an adequate timeline (chapter 2.1) before undertaking the project and this led to stalls and unnecessary work being carried out (see fileErrors.java in section 6.1).

Having only worked on small scale projects up to now we did not realise the importance of using class diagrams (chapter 4) and source control (GitHub) to keep track of a project when it begins to grow in size. Without them we would have had a hard time reintegrating individual aspects of the projects into a whole.

To successfully undertake a project of this size it is important to work with people who are committed and enthusiastic and to plan and prepare before beginning any work. If you don’t want to do the work or if you just don’t understand the scope of the problem, it is easy to become frustrated or to disengage from the process. It is far easier to sustain enthusiasm by setting realistic goals and having a timeframe within which to achieve a known outcome. There were many features that we would have wished to add (chapter 6.5) given more time but it is important to maintain a realistic timeframe and to exercise scope control.

# Bibliography

# References