# CP2011 Assignment – Weather Tracking System

## Jake Dixon (12828309)

### Project Scope:

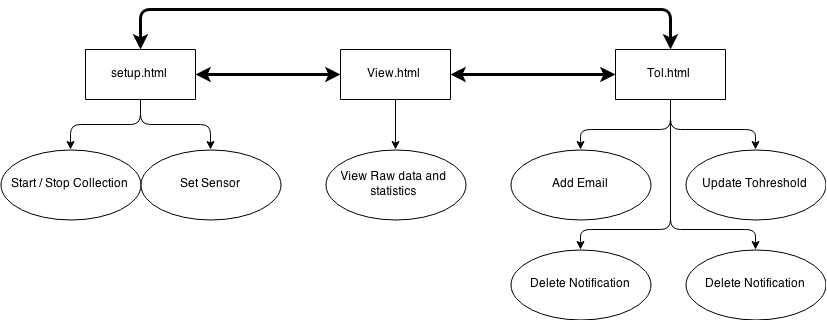
The Goals of the assignment is to create a weather tracking system that logs active sensor data to a database.

* The use has to be able to toggle which sensors are actively receiving new data and the user can set the time interval been records.
* The program will calculate the Sum, mean and standard deviation of each of the sensors data. The user should be able to view these statistics as-well as see the raw data as it is being generated.
* The user can also set tolerances which, when reached an email will be sent to any email addresses the user has added into the program.
* Once a tolerance is reach the program will also record this occurrence for later review / deletion by the user.
* Once a sensor is turned on all previous data is cleared and the tracker starts recording the data.
* The user can start or stop the generation of data in the setup page.
* The web GUI would need to grab data from the database using Javascript to interact with a Java serverlet which would communicate with a Data Access Object.
* The user and program can add stuff to the database by POSTing the data to a serverlet which would then communicate with a Data Access Object

### Project Tools:

* XAMMP – A popular PHP development environment, XAMPP includes a free Apache distribution containing MySQL, PHP and pearl. XAMPP is also open source.
* MySQL – A popular open source Relational Database Management System used in the assignment to grab and store data for the weather Tracker.
* Apache Tomcat - An Open source implementation of java EE which allows us to run Java serverlets and java classes on our localhost server.
* phpMyAdmin – A free tool written in PHP to handle the creation and administration of MSQL databases. This was sued to create our initial database and for testing our program by inputting dummy data.
* IntelliJ IDEA – An Intelligent Java Interactive Development Environment. This is the program used to create all the HTML files, javascript files, serverlets and java classes.
* JDBC – A database connectivity API that was used to interact with the database.
* JavaMail – An API used to send emails from the program.
* JavaBeans Activation Framework – An API used in conjunction with the JavaMail API to send emails.

### Requirements Analysis:

* Setting a Sensor
* Run the setup.html
  + The Javascript controller contacts Serverlet for current sensor data
    - Serverlet contacts DAO to get data
      * DAO grabs current sensor states and returns it to the Serverlet
    - Serverlet returns states as a sext string
  + Javascript sets the states of the checkboxes
* Tick a sensor
* Check box auto posts to Serverlet
  + Serverlet forwards data to the Sensor DAO
    - DAO adds data to the database
* Page Refresh and JavaScript controller runs
* Starting and stopping data collection
  + Run setup.html
  + Select interval
  + Press start
    - Button Posts to Serverlet
      * Serverlet executes timer to record data at certain intervals
  + Press stop
    - Button posts to Serverlet
      * Serverlet destroys timer
* Viewing raw data
  + Run view.html
    - View controller requests data from statistics Serverlet
      * Serverlet calls the statistics DAO
        + DAO grabs raw data and statistics from the database
        + Returns the data
      * Serverlet returns the data as a text string
    - Controller formats data into tables and dynamically updates the HTML
* Setting thresholds
  + Run tol.html
    - Tolerance controller requests current threshold data
      * Serverlet contacts DAO class
        + DAO grabs data from databse and returns it
      * Serverlet return it as text
    - Controller formats the text
  + Set tolerences
  + Press update
    - * Posts to serveerlet which contacts DAO with updated data
        + DAO updates database
  + Page refreshes
* Adding emails
  + Run tol.html
  + Input email in text field
  + Press add button
    - * Button posts to serverlet which call DAO with new email
        + DAO add email to database
  + Page refreshes
* Deleting email address
  + Run tol.html
    - email controller requests current emails
      * Serverlet contacts DAO class
        + DAO grabs data from databse and returns it
      * Serverlet return it as text
    - Controller formats the text
  + Press delete on an email entry
    - * Button posts to serverlet with email which call DAO
        + DAO deletes email from DAO
  + Pages refreshes
* Deleting Notification
  + Run tol.html
    - notification controller requests current notifications
      * Serverlet contacts DAO class
        + DAO grabs data from databse and returns it
      * Serverlet return it as text
    - Controller formats the text
  + Press delete on an notification entry
    - * Button posts to serverlet with notification id which call DAO
        + DAO deletes notification from DAO
  + Pages refreshes

### Implementation:

#### -Client-Side

Data structures:

* Strings – These were returned from the Serverlet, these contained formatted data.
* Arrays – The Strings of data were converted into arrays using the Split method to split them at a certain point.
* DOM objects – These objects were grabbed and created using JavaScript so that parts of the website (tables) could be dynamically generated without refreshing the page. JavaScript could interact with current objects and change their properties and also add set properties of new objects
* XMLHttpRequest object – is an API that Javascript can sue to interact with the Serverlet.
* Function objects / prototype objects– used to control data flow and variable scope.

Algorithms:

* Split() – to separate strings into arrays
* Document – to interact with the web page
* LocalStorage() – to store JavaScript variables across webpages
* forEach – to format data from an array into HTML elements
* Interval – to update tables programmatically with raw data

#### -Server-Side

Data Structures:

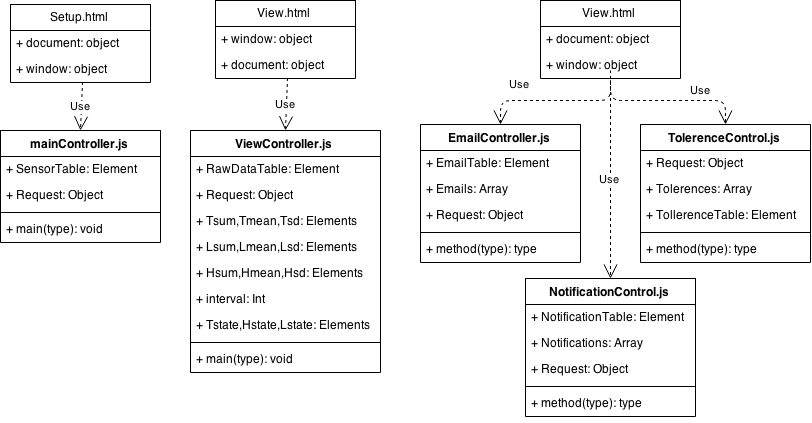
* ArrayList – These were used to store information in the java classes
* HashMap – these were used similarly to ArrayList but theses allowed varables to be stored with different types and nesting
* Java class Objects – Initialised classes as object with methods and can also contain some static variables that can be accessed even without initialising the object (timer)
* String – these store text data, the Serverlets also return pure text.
* Primitive types – int, float and Boolean were used to store variables and used as conditioning for statements and control structures.
* Connection – An object created with the JDBM API sued to intereact with the database
* PreparedStatement – used to dynamically change a sql query statement
* Properties object – has system properties
* Session object – for use with JavaMail
* PrintWriter object – used to respond by a serverlet to respond to a request
* MimeMessage object – data in an Email sent via JavaMail
* Exception object – sued when something goes worng ☹

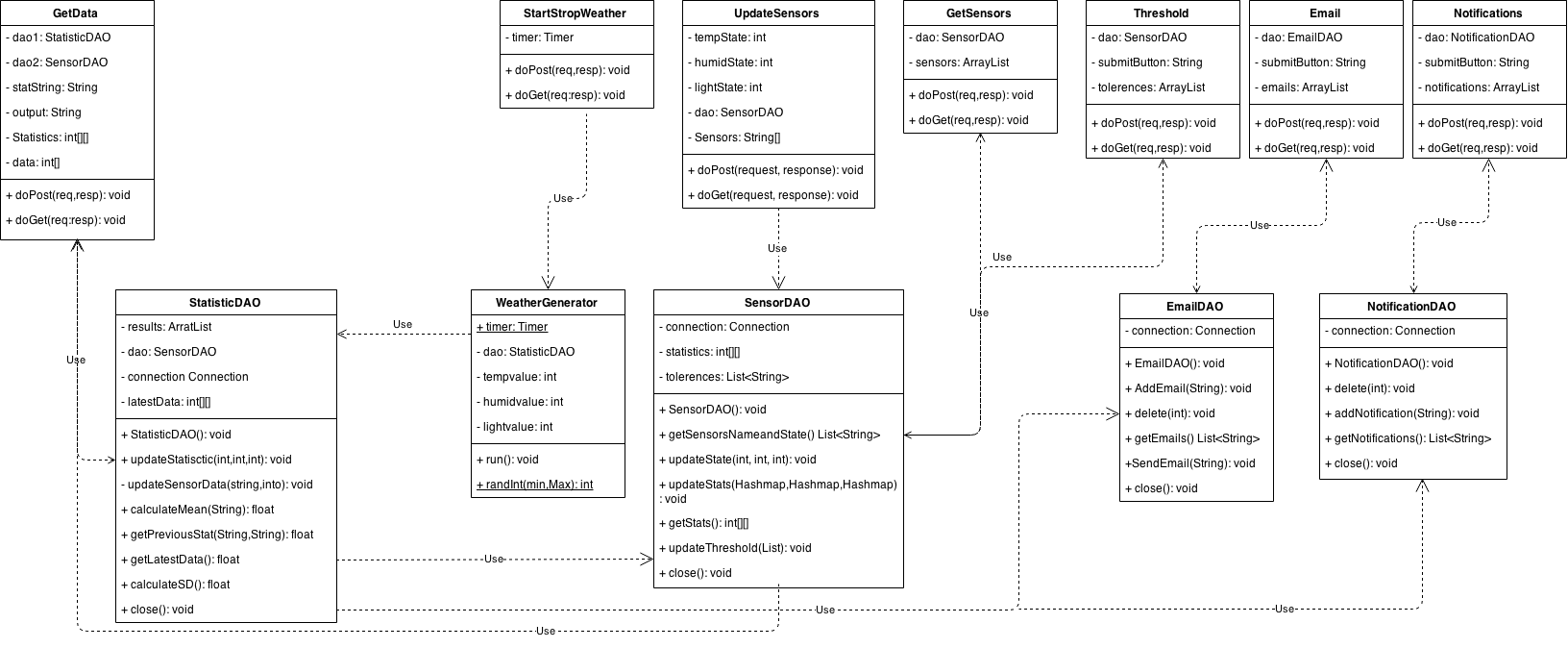
Algorithms:

* If statement – conditional statements to tell the program when to do something
* While loop – conditional loop tell the program to do something until a condition is false
* For loop – conditional loop tells the program to do something a number of times
* Functions – modulate code and number of times code is repeated
* Class – objects that store a number of methods and vales when initiated
* Try / catch – allows the program to avoid crashing and continue to run
* SD formula – calculate the mean, take the square of each element minus mean and then calculate the mean of that square difference and square root that square difference mean
* Mean formula – sum of the elements divide by the number of elements
* Sum formula – n1 + n2 + n3 + … Nn
* Sql statements – INSERT INTO example VALUES eg1, eg2
* TimerTask – controls the generation of data
* switch statement – Conditional statement used when strings are involved instead of an if statement

### Design:

#### -Client-Side





### Modularity:

#### -Client-Side

The JavaScript files in this project are used in a proxy / Adapter pattern. A Proxy / Adapter pattern is a design pattern functioning as an interface to something else. In this case the Javascript allows the html to interface with the serverlet’s. It allows the page to by dynamically updated with data from a server without POSTing from the HTML itself.

#### -Server-side

* Adapter Pattern – All of the Serverlets in this project are Adapter patterns. They adapt POST requests into data that is then fed to or gotten from the Data Access Objects.
* Façade pattern – The Data Access Objects are using a Fascade pattern. The database Management system is a library of java classes and a fascade system is supposed to apply a simplified interface to the larger body.
* Decorator pattern - The weather generator in this case is a decorator pattern. It Extends (decorates) the functionality of the TimerTask task class.

### Database:

