**Remote Weather Station**

**Andrew Schaaf, Adam Baldwin, Hieu Luong**

**Draft #2**

**October 7, 2016**

**CNT 4104 Software Project in Computer Networks**

**Instructor: Dr. Janusz Zalewski**

**Department of Software Engineering**

**Florida Gulf Coast University**

**Ft. Myers, FL 33965**

Copyright © 2016 by Florida Gulf Coast University

**1. Introduction**

Having access to real-time weather data is imperative for both businesses and individuals and is something that many applications and programs depend on. Users demand that data be accurate, up-to-date and dynamic in order to suit their personal needs.

We will be extending the Remote XBee Weather Station which is an extension of three previous projects. The project was started in 2009 by Brad Konert, was extended by Christopher McCoy in 2011 to have internet accessibility [1] and was last worked on by Sergio Pais in Spring 2015 who showed that data could be stored on a simple server and sent to a client. [2] We will be creating a cloud-enabled weather station and an web application to serve as a client. We will be extending the previous work both at the hardware and software level. We will be using a new sensor to get extended dynamic weather data to our client. The sensor (Figure 1.3) will transmit data to the Arduino board (Figure 1.1) via the XBee transceiver (Figure 1.2). The server will connect to the Arduino board via USB and store data on the cloud using Amazon RDS. The client web application will receive weather data from the server via Web API calls which will request both real-time data from the server and long-term data from the Amazon database.

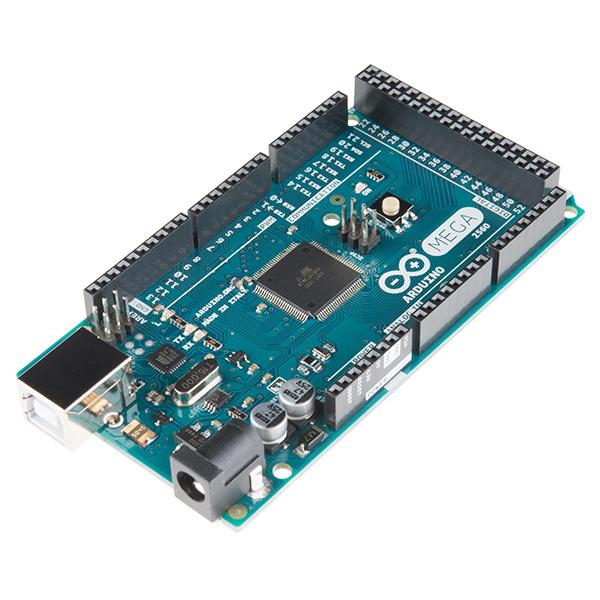


Figure 1.1: Arduino Mega 2560



Figure 1.2: XBee on shield

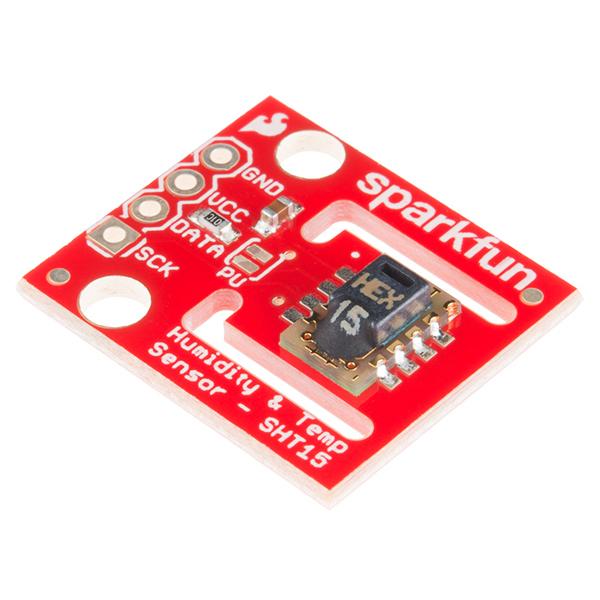


Figure 1.3: SHT15 Temperature and Humidity Sensor

**2. Software Requirements Specification**

**2.1 Project Objectives**

This project is a continuation of three previous semesters of work. The previous work served as a proof of concept that weather data could be read using a sensor, stored in a database, and be sent to a client. Currently, the sensor reads temperature and humidity data, sends it a simple server set up, and sends it to an ASP.NET web form which serves as the client. This semester we will be reconfiguring the sensor, the server and database system, and the client application. As shown in Figure 1.1, the Arduino board will receive data from the sensor and via XBee and will connect to a server via USB. The data will be stored both on the local server and on an Amazon RDS database. A client will then be able to access the data via the internet.



Figure 1.1 Physical diagram

**2.2 Context Diagram**

The weather will be polled by the temperature and humidity sensor and transmitted to the Arduino board via the XBee transceiver. The Arduino board sends sensor data to the server which stores it both locally and sends long term weather data to the Amazon RDS database. The server will receive API calls via HTTP from the client and will query the Amazon RDS database if long-term data are requested. The server the sends the weather data to the client in JSON format. This is illustrated in the context diagram in Figure 1.2.



Figure 1.2: Context diagram for Remote Weather Station

**2.3 Requirements Specification**

Best on the description of functionality in the previous section, software requirements can be formulated as follows.

**2.3.1 Arduino Software Requirements**

A1) The Arduino software shall poll the temperature and humidity sensor every 15 minutes.

**2.3.2 Server Requirements**

S1) The server software shall store the most recent 24 hours of data locally.

S2) The server software shall send an average of the previous hour data to the Amazon RDS database every 24 hours

S3) The server software shall respond to API requests from the client and send data in JSON format back to the client.

S4) The server software shall restart when the hardware is restarted.

**2.3.3 Client Requirements**

C1) The client software shall display current weather by making an API call to the server each time the page is loaded.

C2) The client software shall display weather data based on parameters given by user.

C3) The client software shall display the weather in a graphical and linear format.

**3. Design Description**

**3.1 Software Architecture**

As seen in the software architecture diagram (Figure 3.1) below, the user will interact with a web application through a web browser. The web browser will make HTTP requests to a Web API and will receive data in JSON format. The Web API will get data from both the local database housed on the server and an Amazon RDS database which will store long term data. The parameters sent to the Web API will determine if the Amazon database is queried. There will also be two C# applications deployed on the server. One will periodically send data to the Amazon RDS server every 24 hours. The other will interact with the Arduino Mega and will poll sensor data every 15 minutes.

****

Figure 3.1: Software architecture diagram

**3.2 Detailed Design**

**3.2.1 Web Application**

The Web Application’s structure is shown in Figure 3.2. The main objective of the application is to display weather data to the user. This will be done both graphically and linearly. First, the application must get input from the user regarding the parameters of the data they would like to see (date range, temperature or humidity, etc.). The application will the make an HTTP request to the Web API and will receive a response in JSON format. The application will then display the results in it’s the main window of the application.



Figure 3.2: Web application structure diagram

**3.2.2 Web API**

The Web API’s structure is shown in Figure 3.3. The main objective of the Web API is to send data from a database to the client. It first receives an HTTP request which holds parameters specified by the user. The Web API will use these parameters to determine which database needs to be queried. It then converts the data into JSON format and sends the response to the client.



Figure 3.3: Web API structure diagram

The behavior of the Web Application and Web API is shown in Figure 3.4. The Web API will first receive parameters from the user. If the parameters request long-term data, the Web API will query the Amazon RDS database for the data. If the request is only for short term data, the Web API will only query the local database.

C:\Users\Andy\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Flowchart.png

Figure 3.4: Web application/Web API flow chart

**3.2.3 Server Software: Amazon RDS Application**

The server software is composed of two C# module, one to periodically send collected data to the Amazon RDS database for long term storage, another to poll the sensor. This is to ensure that short term data can be quickly found and long-term data can be stored in a more elastic way. The structure of the part responsible for sending long-term data is shown below in Figure 3.5.



Figure 3.5: Amazon RDS application structure chart

The behavior of the server software is shown in Figure 3.6. The software will run idle while data are collected. Then, every 24 hours, it will average the previous 24 hours of data and form a SQL query. It will then insert the data into the Amazon RDS server and return to idle.



Figure 3.6: Amazon RDS application state chart

**3.2.4 Sensor Polling Application**

The other C# application that will run on the server is the sensor polling application that was built by a previous group [2]. The application will need to be slightly altered to store onto SQL server database rather than a Microsoft Access but otherwise it’s function will remain largely the same. The basic structure of the sensor polling application is shown in Figure 3.7.



Figure 3.7: Sensor Polling Application structure chart

The sensor polling application will be set on a timer and will send a request to the Arduino software every 15 minutes. When the application receives the data from the Arduino board, it will store the data as strings and build a SQL query to store them in the local database. The behavior state chart is shown in Figure 3.8.



Figure 3.8: Sensor Polling Application state chart

**3.2.5 User Interface Design**

The User interface design will allow a user to enter desired parameters (start date, end date, and data types) and update a chart and linear data table. There is a toolbar on top which directs to the main page (Figure 1.3), a documentation page housing all of our project documents, and a contact page for all group members. The main page shows a side bar where a user can enter parameters, a chart area where a line graph is generated with a series representing each data point, and a linear table with the raw data listed.

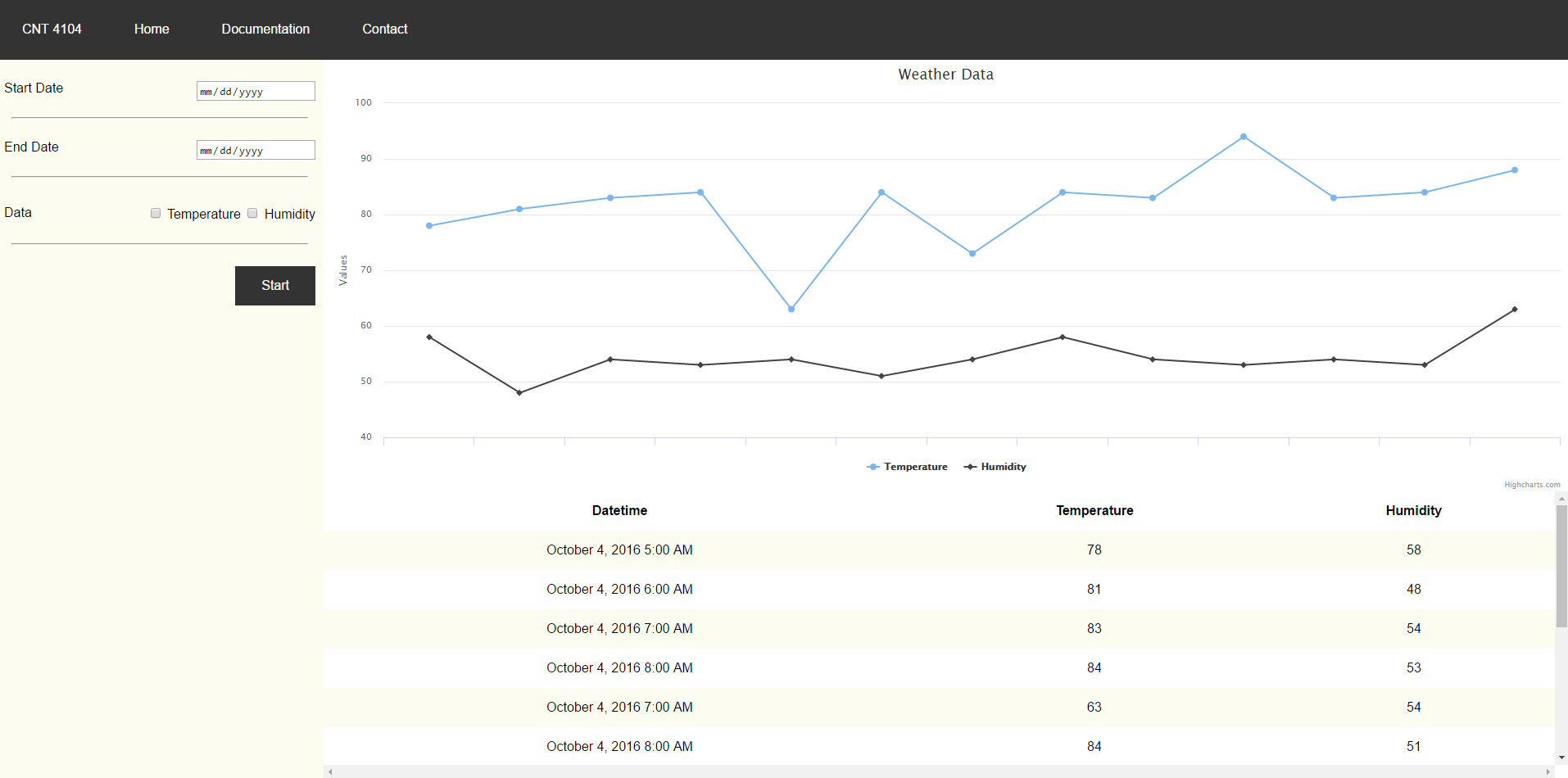


Figure 1.3: UI Design

**6. References**

[1] C. McCoy, Remote XBee Weather Station, FGCU CNT 4104 Software Project In Computer Networks, December 2011.

[2] S. Pais, Remote XBee Weather Station Maintenance Report, FGCU CEN 4935 Senior Software Engineering Project, April 2015.