AN ANDROID MOBILE APP TO VISUALIZE ASU WIND ENERGY PRODUCTION

by

Mikeal C. Anderson

Honors Thesis

Appalachian State University

Submitted to the Department of Computer Science
in partial fulfillment of the requirements for the degree of

Bachelor of Science

May, 2017

Approved by:

James B. Fenwick Jr., Ph.D., Thesis Director

Dee Parks, Ph.D., Second Reader

Dee Parks, Ph.D., Departmental Honors Director

James T. Wilkes, Ph.D., Chair, Computer Science Department

Abstract

Mikeal C. Anderson: An Android Mobile App to
Visualize ASU Wind Energy Production
(Under the direction of James Fenwick Jr.)

Appalachian State University prides itself on its commitment to utilizing renewable energy sources to minimize its carbon footprint. One of the most iconic examples of this commitment is the Broyhill wind turbine. As it stands, there are few access points for citizens to learn about the turbine's active state: its current output, rotational speed, orientation, etc. The goal of this thesis is to build upon preexisting frameworks and outline the implementation of an Android application that could fill this role and be distributed by the University. The application consists of a tabbed interface including several topical pages, most notably being the animated turbine tab which gathers and animates a turbine using close-to-real-time data acquired via a constructed system of a webserver and a backend database. The resulting application could easily be distributed by the University, following a few server-moves and modifications, as well as become the foundation for several other possible projects.

Acknowledgements

Many hands were required to iron out all the details of this thesis, and concluding without a few special mentions about their contributions would be unfit. I'd like to thank Dr. Brian Raichle of the ASU Department of Sustainable Technology and Jim Dees of the ASU Sustainability team for providing information regarding acquisition of turbine data, Nathan Davis and Michael Zanga for their previous groundwork regarding mobile applications for the wind turbine, Joseph Anderson for his role as my graphic artist, providing much of the application's iconography, and finally, Dr. James Fenwick Jr. and Dr. Dee Parks for their patience in editing this document.

Table of Contents

Chapter I - Introduction 1

Chapter II - ASU Renewable Energy Initiative 3

A. ASU Renewable Energy Initiative History 3

B. The Broyhill Wind Turbine 3

Chapter III - Android Mobile Device Programming 6

A. A Brief History of Android and the Mobile Revolution 6

B. Writing Android Applications 7

Chapter IV - Application Implementation 9

A. Overview and Issues 9

1) Database: 11

2) Data Acquisition Script: 12

3) HTTP Request Server: 13

B. Implementation Details 14

1) The FragmentTabActivity: 14

2) History and Project Fragments: 14

3) The TurbineFragment: 15

4) The DetailsFragment: 19

C. Challenges 20

Chapter V – Conclusion and Future Potential 22

References 24

Appendix 26

I. FragmentTabActivity.java 26

II. ProjectFragment.java 26

III. TurbineFragment.java 27

IV. DetailsFragment.java 30

V. TurbineData.java 33

VI. DBInterface.java 35

VII. InternetDataAdapter.java 35

VIII. HTTPServer.py 37

IX. dataReq.py 39

Chapter I - Introduction

Installed on the highest point of the Appalachian State University campus, the Broyhill wind turbine is a visible landmark rising above the campus and surrounding town. The result of an innovative partnership between the Appalachian State Renewable Energy Initiative (ASUREI) and New River Light and Power (NRLP), the wind turbine is a constant reminder of the dedication of the Appalachian student body to engaging sustainable and renewable energy. As the largest turbine of its kind in the state of North Carolina, students walking across campus and even citizens traveling through town often gaze up to the turning blades with pride. Then, for many, that feeling of pride is followed by a sense of curiosity. How fast are those blades turning? How much power is actually being generated right now? Unfortunately, there is no convenient way for such curious citizens to access live metrics regarding the turbine.

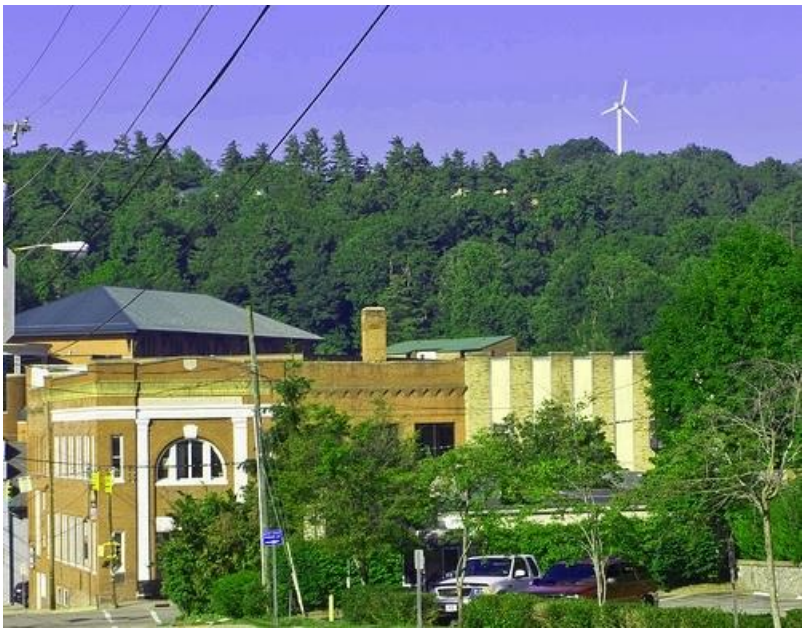

Figure 1 - ASU REI Broyhill wind turbine rises above campus

As a computer science major and graduating senior at Appalachian State University, opportunities to branch out into different areas of programming are valuable towards diversifying my knowledge of the field. One area that I had not yet delved into was mobile application development, which could be an effective means of delivering publicly accessible information regarding the turbine. The two major players in the current smartphone industry, iOS and Android, are both quite popular; however, due to time limitations only one could be chosen. A combination of previous Java experience, openness of development, and lower distribution hurdles were the primary factors leading to the decision for this project to direct its focus to Android devices.

Thus, the goals of this thesis are threefold: to research and learn Android programming as an educational opportunity for myself, research the Appalachian State Renewable Energy Initiative and its on-campus wind turbine, and finally to put these topics together into a convenient Android application that will be released to the public as an access point for live or close-to-live telemetry of the turbine for those curious about the Broyhill wind turbine and the ASUREI.

Chapter II - ASU Renewable Energy Initiative

When talking about the Broyhill wind turbine, it would be impossible not to also cover the Appalachian State University Renewable Energy Initiative (ASUREI) that played a critical role in its acquisition and plays a continuing role in its operation.

A. ASU Renewable Energy Initiative History

In 2002, the state of North Carolina enacted laws that established energy reduction goals and requirements for universities [3]. In response, the University and its students voted in 2004, and reaffirmed in 2007, in favor of a self-imposed fee that would support the establishment of a committee of students and staff which would be responsible for allocating funds towards renewable energy projects on the University's campus. Thus, the mission statement of the ASUREI was decided: "Reduce the environmental impact of Appalachian State University by implementing renewable energy technologies, investing in energy efficiency projects, and promoting campus engagement [12]." Other ASU REI projects include a half dozen photovoltaic array (solar panel) projects, the solar vehicle project, AppalCart biodiesel project, and numerous solar recycling and trash compactors around campus.

B. The Broyhill Wind Turbine

A historic and well-known source of renewable energy, the wind currents that flow across the Earth's surface are one of the fastest growing sources of new electricity generation in the world today. Free from pollution, inexhaustible, and increasingly affordable, wind, and the wind turbines used to harvest its power, is an attractive source of clean energy for

organizations like Appalachian State University that are located in the blustery mountains of North Carolina [10]. Indeed, wind energy and Appalachian State have a relationship that extends as far back as 1979 when one of the first major industrial wind turbines in the United States was installed on Howard's Knob as a NASA experiment [4]. The MOD-1 windmill, as it was called, was the second generation of large-scale industrial wind turbines designed by NASA in response to the OPEC oil crisis of the 1970s in an effort to reduce American dependence on foreign energy [4]. Borrowing from a similar German design, the MOD-1 turbine housed a 2 MW generator set on a cross-braced tower and was powered by two downwind steel blades. The turbine itself was positioned on Howard's Knob on July 11, 1979 and "was hailed as one of the experimental energy technologies that would 'break the OPEC stranglehold'" [4]. Unfortunately, the MOD-1 was subject to a number of adversities ranging from design flaws to community backlash related to noise and ultimately was decommissioned due to mechanical failure and cheaper Reagan-era energy prices [4].

The Boone area wouldn't see another wind turbine for nearly three decades until the ASUREI and New River Light and Power raised the funding for a new model in 2009. Dubbed the Broyhill Wind Turbine due to its location adjacent the Broyhill Inn and Conference Center, which is now demolished, the current wind turbine is a Northern Power Systems Northwind 100 model and touts a more modest 100 kW generator on a 120-foot tall cylindrical tower. Three 34-foot upwind blades harvest wind energy. On June 22, 2009, the unit 154-foot tall structure costing $533,000 was installed at the highest point on the Appalachian State campus, 755 Bodenheimer Drive [16].

The unit is rated to produce between 110,000 kWh and 145,000 kWh per year [13]. The average North Carolina home consumes 1,113 kWh per month or 13,356 kWh annually meaning the Broyhill Wind Turbine is capable of providing power for approximately 10 - 15 Southern homes [9][2]. At peak output however, the 100-kW generator could theoretically power around 66 homes given 730 hours in a month and the 1,113-kWh figure from above leading to the average household consuming around 1.5 kW at any given time. In addition, the unit is estimated to offset an annual 200 metric tons of carbon dioxide [6].

A prominent feature on the Appalachian State campus, as well as in the general Boone area, the wind turbine is a beneficial reminder to residents of the progression of renewable energy and just one of example of the many ways ASU is moving towards a smaller carbon footprint. At this time, unfortunately, there is no easily accessible and user-friendly method of viewing and visualizing data being produced from the turbine. Nonetheless, the turbine has several access points for data and a more public utility featuring a graphic dashboard is in the early-stages of production. This situation provides ample opportunity for this thesis to benefit the curious people of Boone who may be interested in how a small, community-scale wind turbine like the Broyhill wind turbine can impact the energy demands of a city like theirs.

Chapter III - Android Mobile Device Programming

This section of our journey through this thesis requires a preface to make clear the scope of how far and how detailed we will cover Android programming. The breadth of features, implementations, and intricacies contained in the Android platform is vast and many textbooks are filled with dozens of chapters covering every nook and cranny in excruciating detail, several of which will be referenced in this paper. The scope of this project, however, is much more concerned with a higher level, practical overview of this project that would be sufficient to produce the application described previously. Therefore, while a decent portion of the Android development process will be covered here, it will be such that the entry level computer science student or apt layman could follow along and stand a good chance at reproducing much of the final product. With that in mind, we can begin our coverage.

A. A Brief History of Android and the Mobile Revolution

The mobile computing revolution has been nothing short of awe-inspiring in terms of where it started and where it is now. A total paradigm shift has occurred in the ways we access the internet and interact with each other. What used to require bulky, hardwired desktop computers and later laptops can now be accomplished effortlessly via the literal wireless supercomputers that many of us carry in our pockets and (anecdotally) take for granted on a day-to-day basis. There are, by some estimates, more mobile devices in circulation now than there are people on Earth, and Google reports that, today, more of its queries come from mobile users than from conventional computers [15]. We use our smartphones for everything

from traditional voice communication and text messaging to email, web browsing, video streaming, social networking, gaming, and hundreds -if not thousands- of other uses provided by applications being developed every day. Android, by far the most popular mobile operating system on the market with a share of 86.8% in the third quarter of 2016 (the next closest being iOS at 12.5%) and more than one billion users, has played a huge role in this revolution for a multitude of reasons, namely its open-source and royalty free release [14] [7].

Acquired by Google Inc. in 2005, Android was intended to be Google's dedicated venture into the mobile phone market [7]. Android was released under the Apache license, in which "each Contributor grants to [the user] a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form [1]." Two years later, the first Android mobile device, aptly named "The Android", was released by Google. In the decade that followed, Android would become the most popular mobile platform. Thanks to its open-source platform, many device manufacturers could install the Android operating system on their devices which could then be sold at relatively low prices [7]. Naturally, these inexpensive and powerful devices were very attractive to consumers, leading to its widespread adoption.

B. Writing Android Applications

Android applications are written using a Java-like, object-oriented programming language. The language is "Java-like" in that it contains a relatively large subset of Java's functionality, with some unnecessary bits, such as printing, being left out [8]. The result is

essentially “a slimmed down Linux/JVM stack” [11]. Additionally, the layout of an Android application is defined using the Extensible Markup Language (XML) [7][5]. Aside from its overwhelming popularity, these aspects of Android development led it to be the frontrunner when considering what platform to develop this project on, as Appalachian State specializes in teaching its computer science students Java and object-oriented programming practices. Android may be developed in many integrated development environments (IDEs) and, for reference, this project was developed in Android’s official IDE: Android Studio.

Chapter IV - Application Implementation

A. Overview and Issues

Now that we have sufficient background information on both the Broyhill Wind Turbine and Android, we can begin to coalesce the two into a usable and informational application. The implementation for this application is heavily influenced by a previously constructed application of similar functionality created by Nathan Davis and Michael Zanga and used with their permission, continuing what may become an interesting collaboration between students in the future. Many UI elements are borrowed from the Davis and Zanga application. However, the previous application had no data gathering functionality. Whether due to time constraints or similarly encountered difficulties that will be discussed in later chapters, the previous version generated data exclusively using random number generators and its only network functionality was limited to requesting and displaying charts via the Google Charts API. This laid an important foundation for the next obvious step: live data collection. However, adding this feature proved to be a challenging task due to the specifications of the wind turbine, its somewhat mysterious and not-well-known accessibility, and the technologies that would be required to gather and serve such data reliably. As mentioned before, the difficulties of this project will be covered in a later chapter, but suffice to say that Android programming would not be the only technology necessary for fulfilling our goal. Several other “mini-projects” will need to be discussed alongside our main Android-based application. These include:

- A database that will be used to aggregate data retrieved from the wind turbine.

- A data acquisition script that will periodically query a device attached to the wind turbine for data.
    - This script will also need to insert the retrieved data into the database
- An HTTP web server that will accept and handle requests from the application for data.

Naturally, in software development there are a vast number of potential implementations to any given set of requirements, and the reasoning behind these decisions will be covered in their respective chapters.

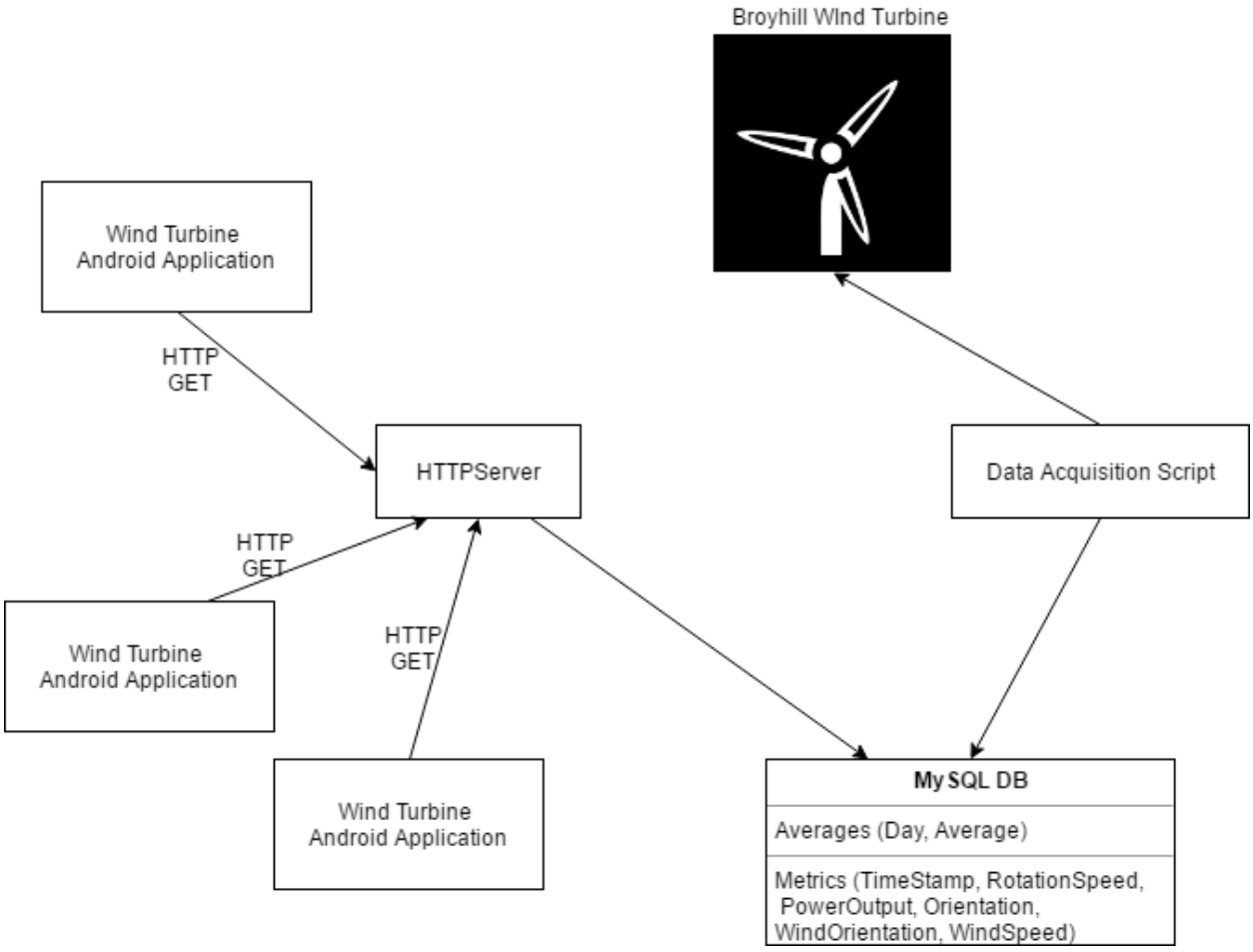

Figure 2 – An overview diagram of the proposed system

1) Database:

To accommodate for calculating and deriving information about the turbine's output over a range of time, it is necessary to turn this standalone project into a database-interfacing one. The values we will be storing for this application will be placed into two different tables of a relational MySQL database: one for raw metrics and one for derived daily average outputs. The "metrics" table will have the following columns: a primary key TimeStamp field of datatype timestamp for keeping track of what day and time the entry was created, a RotationalSpeed field of type float for rotations per minute, a PowerOutput field of type float for the current kW real power output, Orientation and WindOrientation float fields for keeping track of the turbine and wind direction in degrees from North, and finally a WIndSpeed float field for the current wind speed at the turbine. The second table, "averages" will have columns for Day (date type, primary key) and Average (float type) kW output for that Day. The values in the "averages" table will be derived from an SQL query that will be executed from the data acquisition script mentioned earlier. This query is

```
INSERT INTO averages (Day, Average) SELECT DATE(TimeStamp), AVG(PowerOutput)
FROM metrics WHERE DATE(TimeStamp) = UTC_DATE() ON DUPLICATE KEY UPDATE
Average = (SELECT AVG(PowerOutput) FROM metrics WHERE UTC_DATE() =
DATE(TimeStamp));
```

The query takes the all entries in "metrics" for the current UTC date and averages their power outputs, which it then inserts into the "averages" table. The second query, responsible for inserting rows into the table, is dynamically built by the acquisition script.

For this portion, MySQL was chosen due to previous familiarity with the technology, its ability to easily interface with the python scripting language which we will be using later, and its installation on the Appalachian State Computer Science Department student server that we will be coordinating this project via.

2) Data Acquisition Script:

Now that we have a database defined, the next step is populating it. As mentioned previously, we will create a python script for this purpose. Python's simplicity and versatility make it an excellent choice for this job; however, other routes and languages could have been taken to achieve comparable results.

Therefore, this script will have the express purpose of populating our "metrics" and "averages" tables with pertinent data gathered from the turbine and the Yahoo Weather API (for local wind direction). This data acquisition script will be listed in its entirety at the end of this thesis, but the key features are:

- Create a thread to request an XML document from an Acquisuite data acquisition server that contains values retrieved from the turbine Modbus device, and compile an insert query from its contents, along with a wind direction gathered from the second thread, and execute it on the database. This thread then sleeps for 15 seconds before repeating.
- Create a second thread responsible for retrieving the wind direction data from the Yahoo Weather API and storing it in a volatile global variable accessible by the first thread. This thread then sleeps for 10 minutes in order to stay under the Yahoo Weather API's limit of 2,000 calls per day [17].

3) HTTP Request Server:

The final, non-Android part of this project is the HTTP request server. Because Android does not natively support accesses to a remote MySQL database, it is necessary to create a middleman adapter that Android applications *can* interface with. For this project, it was decided that an HTTP server capable of handling GET requests would be used and which Android does provide functionality for. Though there are multiple plug-and-play options for creating simple web servers, this represented an opportunity to explore the details of web server design; therefore, a python server was built from the ground up to handle such requests. In the response, this server will return a string in a format that is similar to the popular comma separated values (CSV) format. The string is free of line breaks, and contains different chunks of data to be processed, separated by semicolons. The first of these chunks contains the timestamp and other columns from the latest row in the table, while the subsequent chunks are up to thirty days' worth of rows from the "averages" table, which will be used to generate a chart inside the application. An example response would be

> 2017-04-23 4:58:34,35.178,6.051,100.0,100.0,11.5,;2017-03-29,11.0;2017-03-30,6.0;2017-03-31,7.0;2017-04-01,8.0;2017-04-02,7.0;2017-04-03,8.0;2017-04-04,9.18172;2017-04-05,5.14134;2017-04-06,50.6082;2017-04-07,55.7892;2017-04-08,4.55224;2017-04-09,6.02344;2017-04-10,1.90635;2017-04-11,3.49516;2017-04-12,3.90634;2017-04-13,1.17005;2017-04-14,2.42275;2017-04-15,2.43265;2017-04-16,5.04;2017-04-17,11.2929;2017-04-18,5.83449;2017-04-19,0.584334;2017-04-20,7.23123;2017-04-21,8.78767;2017-04-22,6.14533;2017-04-23,6.03669;

This format, while somewhat tedious for humans to analyze, is easily parsed with Android/Java's accommodating String library.

B. Implementation Details

With the previous necessary supporting items now in place, we can begin our description of the actual Android application. As mentioned previously, much of this application draws inspiration from the Davis and Zanga application, namely its identity as a subclass of the FragmentActivity class and its four tabbed fragment pages: "History", "Turbine", "Details", and "Project." As described earlier, internet communication has been added and many of the tabs have undergone significant overhauls, visually and technically. In this section, we will tour the various aspects of the application and their detailed implementations.

1) The FragmentTabActivity:

The FragmentTabActivity is simply an extension of the FragmentActivity which overrides the superclass' onCreate method. In this subclass, a TabHost of type FragmentTabHost is instantiated and the four tab fragments are added in the same order as listed above. This acts as the "main" for the program and the onCreate method is the first chunk of the program that is executed when the user starts the turbine application.

2) History and Project Fragments:

The History and Project tabs, shown in Figure 3, are fairly simple and were mostly left unaltered from the Davis and Zanga precursor. The History fragment contains a few simple

paragraphs outlining the history of the turbine and the Project fragment holds a clickable link that opens the user's browser on activation.

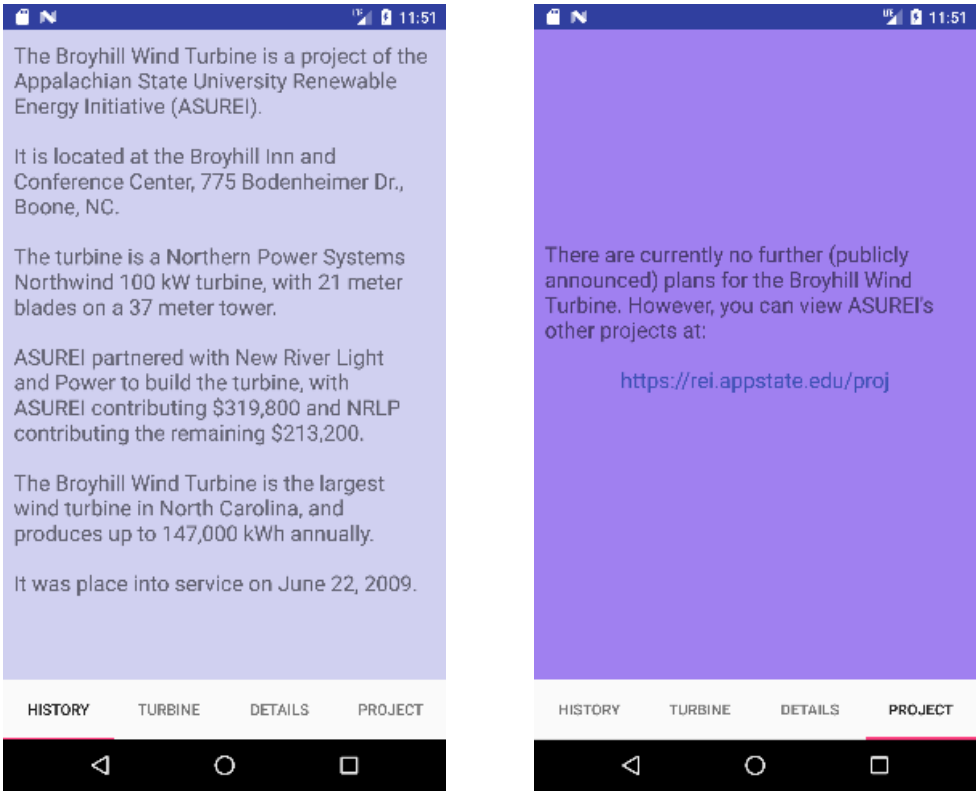

Figure 3 – The History and Project Tabs

3) The TurbineFragment:

Arguably the most important of the tabs in this application, the TurbineFragment will hold the graphically animated turbine, the wind speed and direction indicator, the current turbine RPM counter, as well as the turbine's current power output in kW. Figure 4 shows the Turbine tab of the application. The values and animation are additionally controlled by a Java class that is responsible for updating the current display with information gathered by the turbine. The turbine Java class utilizes several other helper classes to attain this functionality: a singleton TurbineData class and a InternetDataAdapter class that implements a separately defined DBInterface interface.

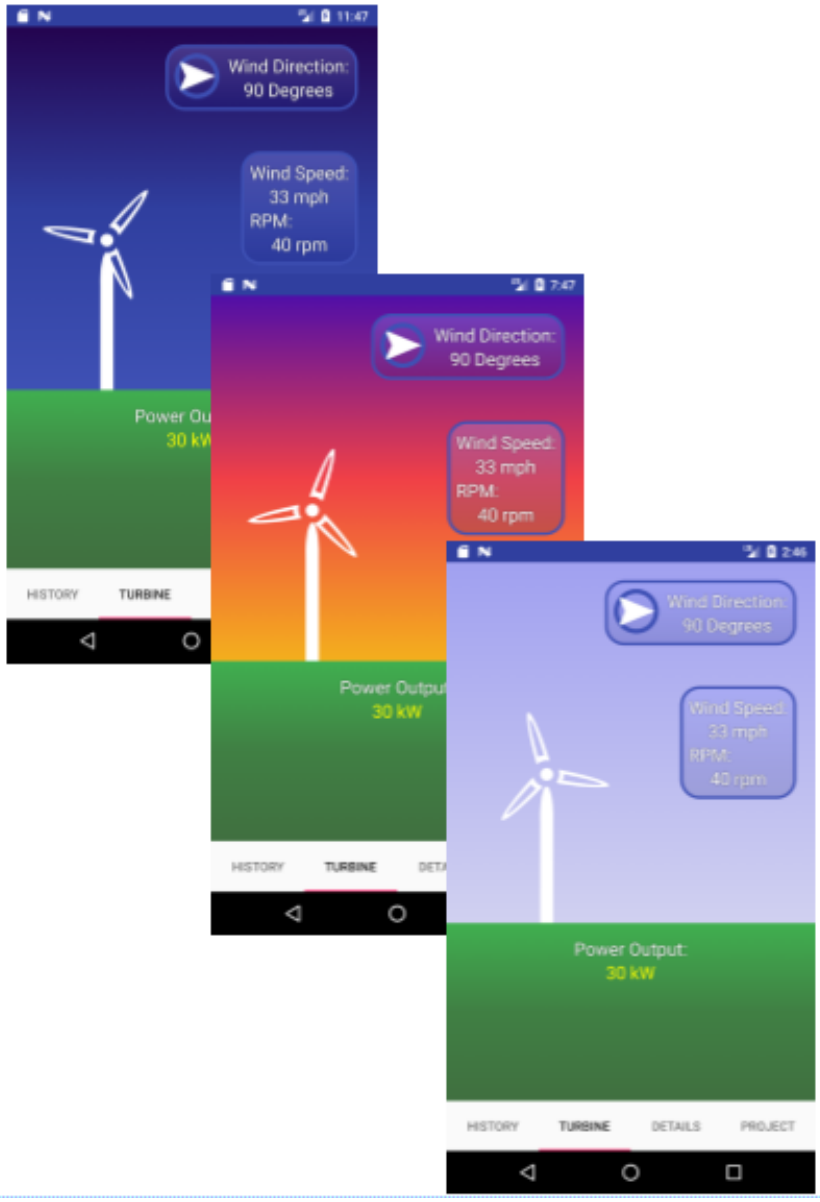

Figure 4 - The Turbine tab with its time-based backgrounds

The InternetDataAdapter class is responsible for connecting and gathering data from the HTTPServer, providing methods for getting rotation speed, power output, orientation, wind orientation, wind speed, and monthly data points. Additionally, the InternetDataAdapter holds an ArrayList of Strings which represent the various data to be returned, an update method for, as the name implies, updating the values from the server, and a private

getConnection method which is responsible for handling the details of interfacing with the server.

Moving up the chain, the TurbineData class, as mentioned earlier, is a singleton class that acts as the interface between the TurbineFragment and the InternetDataAdapter. As such, it consists largely of a collection of wrapper methods for the latter which deal with error handling and logging. The headlining feature of the TurbineData class is its nested private inner class, UpdateDataTask, an extension of Android's AsyncTask class, which allows for asynchronous execution of the InternetDataAdapter's networking functionality. Because host-to-host internet communication can be delayed by an indeterminate amount of time depending on the medium and throughput of the links connecting them, it would be unwise to block the activity's UI thread while we wait for the InternetDataAdapter to finish its business. This would result in stuttering at best and a completely frozen display at worst, depending on the delay. To solve this, we delegate the server communication and data acquisition to its own thread via an AsyncTask which has a doInBackground method that executes the update and an onPostExecute method which advertises to the UI thread that an update is available via a LocalBroadcastManager. This preserves the UI thread and allows the transaction to take however long it needs to finish. The UpdateDataTask is executed every 15 seconds as this is also the delay between new input being inserted into the database, so any sooner would be a waste. An extension to this delay, however, could very well be considered if load on the server became too great at the 15 second interval.

Back in the TurbineFragment class being run in the UI thread, data is gathered from the TurbineData class and fed via mutator methods to a number of Views and Layouts that

have been inflated by the class' onCreateView method. While most of these methods are straightforward, the animateTurbine method is worth noting for how it takes the turbine's RPMs and translates it into a rotating animation. To begin, the animateTurbine method stores the current rotational position of the drawable turbine blades and performs a mod operation on this value by 360 (the number of degrees in a circle). The animateTurbine utilizes an ObjectAnimator object to execute the animation which takes an Android View object, animation style, start position, end position, and animation duration to carry out the desired action. Since we already have the view, animation type ("rotation"), and start position, we can find the end position by simply adding 360 degrees and completing one rotation. This works well since we are given the rotational speed of the turbine in rotations per minute and from there we can find the duration of one rotation by dividing the number of milliseconds in a minute (60000) by the speed in RPM. This results in the number of milliseconds that one rotation of the turbine will take and all that's left is to animate the turbine using the INFINITE repeat setting to keep it running at that duration until a new speed update is delivered.

The result of these details is a smooth and accurate turbine animation that sits on a page detailing the various dynamic aspects of the wind turbine. There are several minor features to this fragment that help polish the aesthetic of the application and these include time-based sky background colors, rotation direction based on prevailing wind orientation, and a "Brake Engaged" notification on detection of a negative power output. In the event of a connection error, the TurbineData class will catch and log the exception as well as create a "Server Error" notification informing the user of the connection problem.

Finally, the issue with the TurbineFragment in its current state is that, even with the multithreaded approach, the turbine still stutters on the retrieval of new data. This is likely due to the nature of changing the duration of the ObjectAnimator rotation animation on the fly without a full stop to cover up the difference; but, in the end, it is barely noticeable and does not negatively impact the UI significantly.

4) The DetailsFragment:

The final tab/fragment is the DetailsFragment, which is shown in Figure 5. This tab is responsible for visualizing the percentage of total power possible output from the turbine in the form of a progress bar, detailing the number of houses and lightbulbs that the turbine could power at the moment, and displaying a dynamically generated chart containing the previous thirty days' individual average power output in kW.

This tab is quite similar to the original Davis and Zanga version with a few modifications, the highlights being: a recalculated formula for deriving the number of possibly powered objects based on the 1.5-kW figure previously mentioned for houses and a typical 60-watt bulb; and, an improved access to the Google Charts API for generating the output chart.

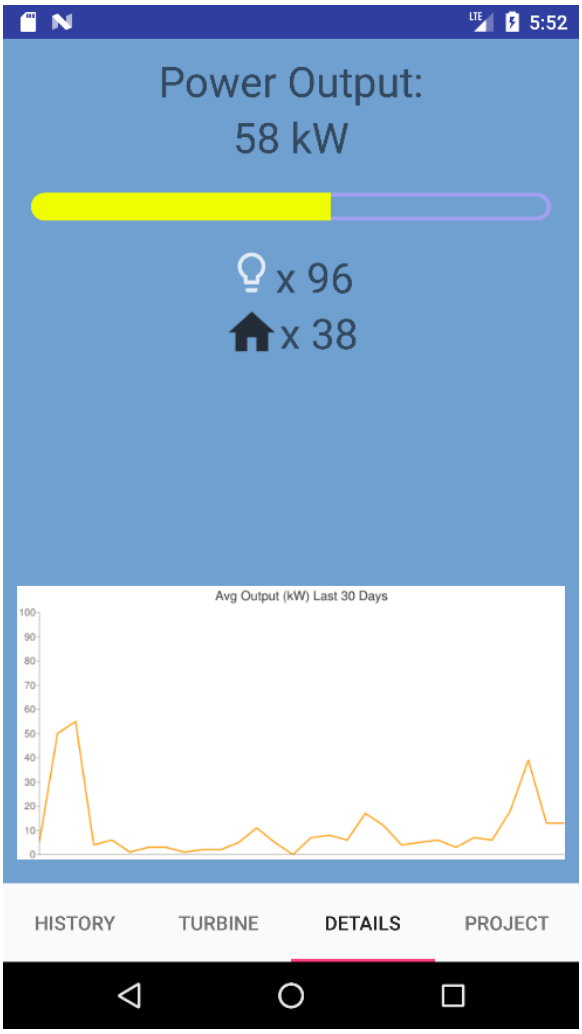

Figure 5 – The Details tab with current output and chart

C. Challenges

As hinted at previously, there were several challenges faced in this project that are worth mentioning. It was a logistical endeavor to determine the ideal way to interface the number of devices contained within the scope of this application. The turbine's data transmission device primarily runs a Modbus serial communication protocol which had to be dealt with via a specialized Obvius AcquiSuite data acquisition server. A relatively straightforward concept at this point; however, numerous REI members needed to be

consulted with over the course of nearly two months to identify the server and determine the ideal way to gain access to and retrieve fresh data. In order to aggregate this data and insulate the acquisition server from overload, a database needed to be kept that could be queried; yet, the available MySQL database proved to be incompatible with Android directly. As such, an adapter needed to be created using a common protocol (in this case HTTP) to facilitate their interaction across many simultaneous sessions. The scope of this project grew rapidly from a simple Android application to one that required knowledge of databases, networking, and parallelism; an excellent exercise is a number of pertinent computer science fields.

Chapter V – Conclusion and Future Potential

The final product of this thesis is a concise and lightweight application capable of accurately conveying information regarding the Broyhill Wind Turbine's current state. The animation provides a human-friendly way to visualize exactly how quickly the turbine is rotating or how many lightbulbs it might power at a given moment, while those interested in more technical details will find a reasonable amount of quantitative data available in the various measurement fields. A chart of the previous thirty days' outputs can give a snapshot of how recent energy production is going and allow users to extrapolate about how wind conditions in Boone have been over that timeframe.

There is plenty of room for future development on this project, especially given its current constraints to the student server in Appalachian State's Computer Science Department. The backend services could, with reasonable ease, be transferred to a more public server under the control of the University's Technology Department, where they would be far better suited to handle access requests from a city's-worth of residents. At that point, a move to the Android market would be the next natural progression, allowing for its distribution to those interested in its capabilities. Furthermore, the backend could be used by other HTTP-capable platforms, of which there are many, to represent the data in other formats. For example, a web application could be built to display this data in a browser, freeing it from device specific constraints. Several of those whom I have spoken with have mentioned a potential scale model of the wind turbine being built and placed in the campus

library which could be outfitted with an Arduino, a few servos, and be connected to my HTTP server to spin its blades according to the speed of its full-scale counterpart.

References

[1] Apache License Version 2.0, Apache Foundation, 2004 [Online]. Available: www.apache.org/licenses/LICENSE-2.0.html

[2] “Appalachian installs wind turbine on campus,” ASU University News, June 2009 [Online]. Available: www.news.appstate.edu/2009/06/24/wind-turbine-on-campus/

[3] “Appalachian State University recognized for reducing energy use,” ASU University News, Dec. 2016 [Online]. Available: www.news.appstate.edu/2016/12/22/energy-use/

[4] B. Kovarik, “Remembering the Wooshies of ‘79,” Appalachian Voices, no. 3, June 2009.

[5] B. Phillips et al., “Your First Android Application,” Android Programming: The Big Nerd Ranch Guide, 2nd ed., Atlanta, Big Nerd Ranch LLC., 2015, ch. 1, pp 1-32.

[6] Broyhill Wind Turbine, ASU Renewable Energy Initiative Info Page, 2005 [Online]. Available: rei.appstate.edu/pagesmith/61

[7] C. Hoisington, “Voilà! Meet the Android,” in Android Boot Camp for Developers Using Java, 1st ed., Boston, Cengage Learning, 2013, ch. 1, pp. 1-9.

[8] E. Burnette. (2008). “Java vs. Android APIs,” ZDNet [Online]. Available: www.zdnet.com/article/java-vs-android-apis/

[9] FAQ: How much electricity does an American home use?, U.S. Energy Information Administration, 2015 [Online]. Available: www.eia.gov/tools/faqs/faq.php?id=97&t=3%5d%5b

[10] Introduction to Wind Power, ASU NC Wind Energy Site, 2015 [Online]. Available: wind.appstate.edu/wind-power

[11] J. Rose S. (2008). With Android and Dalvik at Google I/O [Official Blog]. Available: blogs.oracle.com/jrose/with-android-and-dalvik-at-google-io

[12] Renewable Energy Initiative, ASU REI About Page, 2017 [Online]. Available: rei.appstate.edu/about-the-rei

[13] Renewable Energy Project: Broyhill Wind Turbine, ASU Campus Renewable Energy Systems Spec. Sheet [Online]. Available: sustain.appstate.edu/_documents/Wind%20Turbine%20system%20data.pdf

[14] Smartphone OS Market Share, 2016 Q3, International Data Corp., 2016 [Online Press Release]. Available: www.idc.com/promo/smartphone-market-share/os

[15] S. Pichai. (2015). You say you want a mobile revolution… [Official Blog]. Available: googleblog.blogspot.com/2015/05/io-2015-mobile-revolution.html

[16] Wind Power, ASU University Sustainability, 2017 [Online]. Available: https://sustain.appstate.edu/initiatives/renewable/wind/

[17] Yahoo Weather API for your apps, Yahoo Developer Network, 2017 [Online]. Available: https://developer.yahoo.com/weather/

Appendix

I. FragmentTabActivity.java

```
public class FragmentTabActivity extends FragmentActivity {

    private FragmentTabHost mTabHost;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        StrictMode.ThreadPolicy policy = new
StrictMode.ThreadPolicy.Builder().permitAll().build();
        StrictMode.setThreadPolicy(policy);

        setContentView(R.layout.activity_fragment_tabs);

        mTabHost = (FragmentTabHost) findViewById(android.R.id.tabhost);
        mTabHost.setup(this, getSupportFragmentManager(), android.R.id.tabcontent);

        mTabHost.addTab(

mTabHost.newTabSpec("history").setIndicator(getString(R.string.history)),
                HistoryFragment.class, null
        );

        mTabHost.addTab(

mTabHost.newTabSpec("turbine").setIndicator(getString(R.string.turbine)),
                TurbineFragment2.class, null
        );

        mTabHost.addTab(

mTabHost.newTabSpec("details").setIndicator(getString(R.string.details)),
                DetailsFragment.class, null
        );

        mTabHost.addTab(

mTabHost.newTabSpec("project").setIndicator(getString(R.string.project)),
                ProjectFragment.class, null
        );

        mTabHost.setCurrentTabByTag("turbine");
    }
}
```

II. ProjectFragment.java

```
public class ProjectFragment extends Fragment {

    TextView mLink;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
```

```
savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_project, container, false);
        mLink = (TextView) view.findViewById(R.id.rei_link);

        mLink.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String url = getResources().getString(R.string.asurei_url);
                Uri address = Uri.parse(url);
                Intent i = new Intent(Intent.ACTION_VIEW, address);
                startActivity(i);
            }
        });

        return view;
    }
}
```

III. TurbineFragment.java

```
public class TurbineFragment extends Fragment{

    private static final String TAG = "TurbineFragment";
    private static final String COMPASS_ROTATION = "compass rotation";
    private static final String TURBINE_ROTATION = "turbine rotation";

    private TurbineData mTurbineData;
    private BroadcastReceiver mReceiver;
    private ObjectAnimator mTurbineAnimator;

    private View mTurbineView;
    private ImageView mCompassView;
    private TextView mWindOrientView;
    private TextView mWindSpeedView;
    private TextView mPowerOutputView;
    private TextView mRPM;
    private RelativeLayout mContainerView;
    private float compassRotation;
    private float turbineRotation;
    private int origColor;

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

            mTurbineData = TurbineData.getInstance(getContext());

        mReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                updateValues();
            }
        };
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_turbine, container, false);
        mContainerView = (RelativeLayout) view.findViewById(R.id.container);
        mTurbineView = view.findViewById(R.id.blades);
```

```
        mCompassView = (ImageView) view.findViewById(R.id.compass);
        mWindOrientView = (TextView) view.findViewById(R.id.wind_orient);
        mWindSpeedView = (TextView) view.findViewById(R.id.wind_speed);
        mPowerOutputView = (TextView) view.findViewById(R.id.power_output);
        mRPM = (TextView) view.findViewById(R.id.rpm);
        origColor = mWindSpeedView.getCurrentTextColor();
        setBG();
        if (savedInstanceState != null) {

mCompassView.setRotation(savedInstanceState.getFloat(COMPASS_ROTATION));

mTurbineView.setRotation(savedInstanceState.getFloat(TURBINE_ROTATION));
        }

        compassRotation = mCompassView.getRotation();
        turbineRotation = mTurbineView.getRotation();

        return view;
    }

    @Override
    public void onResume() {
        super.onResume();
        updateValues();
        setBG();
        LocalBroadcastManager.getInstance(getActivity())
                .registerReceiver(mReceiver, new IntentFilter("update"));
    }

    @Override
    public void onPause() {
        super.onPause();
        mTurbineAnimator = null;
        LocalBroadcastManager.getInstance(getActivity())
                .unregisterReceiver(mReceiver);
        compassRotation = mCompassView.getRotation();
        turbineRotation = mTurbineView.getRotation();
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putFloat(COMPASS_ROTATION, compassRotation);
        outState.putFloat(TURBINE_ROTATION, turbineRotation);
    }


    private void setBG(){
        try {
            Drawable day = getResources().getDrawable(R.drawable.daysky);
            Drawable dusk = getResources().getDrawable(R.drawable.dusksky);
            Drawable eve = getResources().getDrawable(R.drawable.eveningsky);
            Calendar cal = Calendar.getInstance();
            Log.d(TAG, "It is " + cal.getTime() + " Time");
            Calendar dayCal = Calendar.getInstance();
            dayCal.set(Calendar.HOUR_OF_DAY, 6);
            dayCal.set(Calendar.MINUTE, 30);
            Calendar duskCal = Calendar.getInstance();
            duskCal.set(Calendar.HOUR_OF_DAY, 19);
            duskCal.set(Calendar.MINUTE, 0);
            Calendar eveCal = Calendar.getInstance();
            eveCal.set(Calendar.HOUR_OF_DAY, 20);
            eveCal.set(Calendar.MINUTE, 0);
```

```
            if(cal.getTime().after(eveCal.getTime()) ||
dayCal.getTime().after(cal.getTime())){
                Log.d(TAG, "It is after " + eveCal.getTime() + " Time");
                mContainerView.setBackground(eve);
            }
            else if(cal.getTime().after(duskCal.getTime())){
                Log.d(TAG, "It is after " + duskCal.getTime() + " Time");
                mContainerView.setBackground(dusk);
            }
            else {
                mContainerView.setBackground(day);
            }
        } catch (Exception e) {
            Log.d(TAG,"BGERROR",e);
        }
    }
    private void updateValues() {
        try {
            mTurbineData.update();
            setWindOrientation(mTurbineData.getWindOrientation());
            setWindSpeed(mTurbineData.getWindSpeed());
            setPowerOutput(mTurbineData.getPowerOutput());
            animateTurbine(mTurbineData.getRotationSpeed());
        } catch (Exception e) {
            Toast.makeText(getContext(),"Connection
error",Toast.LENGTH_LONG).show();
            Log.d(TAG, "Error", e);
        }
    }

    private void setWindOrientation(float orientation) {
        float start = mCompassView.getRotation();
        if (start - orientation > 180)
        {
            start -= 360;
        }
        if (orientation - start > 180)
        {
            start += 360;
        }
        String text =
String.format(getResources().getString(R.string.direction_format), orientation);
        mWindOrientView.setText(text);
        ObjectAnimator compassAnimator = ObjectAnimator.ofFloat(mCompassView,
                "rotation",
                start,
                orientation)
                .setDuration(1000);
        compassAnimator.start();

    }

    private void setWindSpeed(double speed) {
            String text =
String.format(getResources().getString(R.string.speed_format), speed);
            mWindSpeedView.setText(text);

    }

    private void setPowerOutput(double output) {
        if (output > 0) {
            String text =
```

```
String.format(getResources().getString(R.string.power_format), output);
            mPowerOutputView.setText(text);
            mPowerOutputView.setTextColor(Color.YELLOW);
        }
        else{
            mPowerOutputView.setText("Brake Engaged");
            mPowerOutputView.setTextColor(Color.RED);
        }
    }

    private void setRPM(double rpm) {
        String text = String.format("%.0f rpm", rpm);
        mRPM.setText(text);
    }
    private void animateTurbine(float speed) {

        float start = mTurbineView.getRotation() % 360;
        float end = 0;
        if(mTurbineData.getWindOrientation() >= 200)
            end = start - 360;
        else end = start + 360;
        setRPM(speed);
//        Log.d(TAG,"Starting At: " + start);
//        Log.d(TAG,"Ending At: " + end);
        if (mTurbineAnimator == null) {
            Log.d(TAG, "Animator is null");
            mTurbineAnimator = ObjectAnimator.ofFloat(mTurbineView, "rotation",
start, end)
                    .setDuration((long) (60000/speed));
            mTurbineAnimator.setRepeatCount(ValueAnimator.INFINITE);
            mTurbineAnimator.setInterpolator(new LinearInterpolator());
        } else {
            Log.d(TAG, "Animator is not null");
            mTurbineAnimator.cancel();
            mTurbineAnimator.setFloatValues(start, end);
        }
        if (speed != 0) {
            Log.d(TAG, "Speed is " + speed + "RPM,  Duration:" +
mTurbineAnimator.getDuration()
                    + " rotation: " + mTurbineView.getRotation());
            long currentDur = mTurbineAnimator.getDuration();
            long newDur = (long) (60000 / speed);
            mTurbineAnimator.setDuration((long) (60000 / speed));
            mTurbineAnimator.start();
        }
    }
}
```

IV. DetailsFragment.java

```
public class DetailsFragment extends Fragment {

    private static final String TAG = "DetailsFragment";
    private static final String CHART_BYTES = "CHART_BYTES";

    private TurbineData mTurbineData;
    private BroadcastReceiver mReceiver;
    private byte[] mChartBytes;

    private ImageView mChartView;
    private ImageView mPowerBar;
```

```
    private TextView mPowerText;
    private TextView mTextBulbs;
    private TextView mTextHouses;
    private int origColor;

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mTurbineData = TurbineData.getInstance(getContext());
        mReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                updateValues();
            }
        };
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_details, container, false);
        mPowerBar = (ImageView) view.findViewById(R.id.power_bar);
        mPowerText = (TextView) view.findViewById(R.id.power_text);
        mTextBulbs = (TextView) view.findViewById(R.id.light_bulbs);
        mTextHouses = (TextView) view.findViewById(R.id.houses);
        mChartView = (ImageView) view.findViewById(R.id.chart_view);
        origColor = mPowerText.getCurrentTextColor();

        if (savedInstanceState != null) {
            mChartBytes = (byte[]) savedInstanceState.getSerializable(CHART_BYTES);
        }
        if (mChartBytes != null) {
            Log.d(TAG, "Reloading saved chart");
            mChartView.setImageBitmap(BitmapFactory.decodeByteArray(mChartBytes, 0,
mChartBytes.length));
        }
        ClipDrawable pbar = (ClipDrawable) mPowerBar.getDrawable();
        pbar.setLevel((int) (mTurbineData.getPowerOutput() * 100));
        updateValues();
        return view;
    }

    @Override
    public void onResume() {
        super.onResume();
        LocalBroadcastManager.getInstance(getActivity())
                .registerReceiver(mReceiver, new IntentFilter("update"));
    }

    @Override
    public void onPause() {
        super.onPause();
        LocalBroadcastManager.getInstance(getActivity())
                .unregisterReceiver(mReceiver);
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putSerializable(CHART_BYTES, mChartBytes);
    }

    private void updateValues() {
```

```
        double power = mTurbineData.getPowerOutput();
        setPower(power);
        setLightBulbs(power);
        setHouses(power);
            Log.d(TAG, "Setting Chart");
            setChart(mTurbineData.getMonthData());
    }

    private void setPower(double power) {
        if (power > 0) {
            String text =
String.format(getResources().getString(R.string.power_format), power);
            mPowerText.setText(text);
            mPowerText.setTextColor(origColor);
        }
        else{
            mPowerText.setText("Brake Engaged");
            mPowerText.setTextColor(Color.RED);
        }
        ClipDrawable pbar = (ClipDrawable) mPowerBar.getDrawable();
        ObjectAnimator animator = ObjectAnimator.ofInt(pbar,
                "level",
                pbar.getLevel(),
                (int) (power * 100))
                .setDuration(1000);
        animator.start();
    }

    private void setLightBulbs(double power) {
        int bulbs = (int) (power / .6);
        mTextBulbs.setText( String.format(
                getResources().getString(R.string.number_of), bulbs));
    }

    private void setHouses(double power) {
        int houses = (int) (power / 1.5);
        mTextHouses.setText( String.format(
                getResources().getString(R.string.number_of), houses));
    }

    private void setChart(List<Double> data) {
        StringBuilder builder = new StringBuilder(
                "https://chart.googleapis.com/chart?" +
                "cht=lc&" +
                "chs=600x300&" +
                "chtt=Avg%20Output%20(kW)%20Last%2030%20Days&" +
                "chxt=y&" +
                "chxl=1:|kW&" +
                "chd=t:");
        builder.append(data.get(0).intValue());
        for (int i = 1; i < data.size(); i++) {
            builder.append(',');
            builder.append(data.get(i).intValue());
        }
        builder.append("&");
        Log.d(TAG, "Target URL: " + builder.toString());
        new ChartFetcherTask(builder.toString()).execute();
    }

    private class ChartFetcherTask extends AsyncTask<Void, Void, byte[]> {
        private String mURL;

        public  ChartFetcherTask(String url) {
```

```
            mURL = url;
        }

        @Override
        protected byte[] doInBackground(Void... params) {
            Log.d(TAG, "Getting Chart");
            try {
                return new URLFetcher().fetchURL(mURL);
            } catch (IOException ioe) {
                Log.e(TAG, "Error loading bytes: " + ioe);
                return new byte[0];
            }
        }

        @Override
        protected void onPostExecute(byte[] bytes) {
            mChartBytes = bytes;
            Bitmap bmp = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
            mChartView.setImageBitmap(bmp);
        }
    }
}
```

V. TurbineData.java

```
public class TurbineData {
    private static final String TAG = "TurbineData";
    private static TurbineData sInstance;
    private static Random sRNG;

    private Context mContext;

    private DBInterface mData;

    private TurbineData(Context context){
        mData = new InternetDataAdapter();
        sRNG = new Random();
        mContext = context;
        final Handler dataHandler = new Handler();
        Runnable dataRunnable = new Runnable() {
            @Override
            public void run() {
                new UpdateDataTask().execute();
                dataHandler.postDelayed(this, 15000);
            }
        };
        dataRunnable.run();
    }

    public static TurbineData getInstance(Context context) {
        if (sInstance == null) {
            sInstance = new TurbineData(context);
        }
        return sInstance;
    }

    public void update(){
        try {
            mData.update();
        } catch (Exception e) {
            Toast.makeText(mContext,"Server error",Toast.LENGTH_LONG).show();
```

```
            Log.d(TAG, "Error", e);
        }
    }
    public float getWindOrientation() {
        try {
            return (float) mData.getWindOrientation();
        } catch (Exception e) {
            Log.d(TAG, "Error", e);
            return 0;
        }
    }

    public float getWindSpeed() {

        try {
            return (float) mData.getWindSpeed();
        } catch (Exception e) {
            Log.d(TAG, "Error", e);
            return 0;
        }
    }

    public float getPowerOutput() {

        try {
            return (float) mData.getPowerOutput();
        } catch (Exception e) {
            Log.d(TAG, "Error", e);
            return 0;
        }
    }

    public float getRotationSpeed() {

        try {
            return (float) mData.getRotationSpeed();
        } catch (Exception e) {
            Log.d(TAG, "Error", e);
            return 0;
        }
    }

    public List<Double> getMonthData() {

        try {
            return mData.getMonthData();
        } catch (Exception e) {
            Log.d(TAG, "Error", e);
            return new ArrayList<Double>();
        }
    }

    private class UpdateDataTask extends AsyncTask<Void, Void, DBInterface> {

        @Override
        protected DBInterface doInBackground(Void... params) {
            // Connect to server and get data
            InternetDataAdapter data = null;//DataAdapter("");
            try {
                data = new InternetDataAdapter();
            } catch (Exception e) {
                Log.d(TAG, "mySQLAdapter err", e);
            }
```

```
            return data;
        }

        @Override
        protected void onPostExecute(DBInterface data) {
            // Update local data
            mData = data;

            // Tell active fragments that new data is available
            LocalBroadcastManager.getInstance(mContext).sendBroadcast(new
Intent("update"));
        }
    }
}
```

VI. DBInterface.java

```
public interface DBInterface {
    public void update() throws Exception;
    public double getRotationSpeed();
    public double getPowerOutput();
    public double getWindOrientation();
    public double getWindSpeed();
    public List<Double> getMonthData();
}
```

VII. InternetDataAdapter.java

```
public class InternetDataAdapter implements DBInterface{

    private static final String TAG = "InternetDataAdapter";
    Connection conn;
    ArrayList<String> resArr;
    public InternetDataAdapter() {

        try {
            resArr = getConnection();
        } catch (Exception e) {
            Log.d("MYSQLADAPTER", "CONST ERR");
            resArr = new ArrayList<String>();
            for(int i = 0; i < 10; i++) {
                resArr.add("0");
            }
        }
    }

    private ArrayList<String> getConnection() throws IOException {
            URL url = new URL("http://student.cs.appstate.edu:15005/Wind");
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            connection.setConnectTimeout(3000);
            connection.setRequestMethod("GET");
            connection.setRequestProperty("Content-Type", "text/plain");
            InputStream is = connection.getInputStream();
            if (connection.getResponseCode() != HttpURLConnection.HTTP_OK) {
                throw new IOException(connection.getResponseMessage() + ": with
Wind");
            }
            BufferedReader rd = new BufferedReader(new InputStreamReader(is));
            String line;
```

```
            StringBuffer response = new StringBuffer();
            while((line = rd.readLine()) != null){
                response.append(line);
                response.append('\r');
            }

            rd.close();
            connection.disconnect();
            String res = response.toString();
            String[] resArr = res.split(";");
            ArrayList<String> result = new ArrayList<String>();
            for (String i:resArr[0].split(",")) {
                result.add(i);
            }
            for(int i = 1; i < resArr.length - 1; i++){
                result.add(resArr[i].split(",")[1]);
            }
            return result;

    }
    @Override
    public void update() throws IOException {
        try {
            resArr = getConnection();
        } catch (IOException e) {
            Log.d("MYSQLADAPTER", "CONST ERR");
            resArr = new ArrayList<String>();
            for(int i = 0; i < 10; i++) {
                resArr.add("0");
            }
            throw e;
        }
    }
    @Override
    public double getRotationSpeed() {
        return Double.parseDouble(resArr.get(1));
    }
    @Override
    public double getPowerOutput() {
            return Double.parseDouble(resArr.get(2));
    }
    @Override
    public double getWindOrientation() {
        return Double.parseDouble(resArr.get(4));
    }
    @Override
    public double getWindSpeed() {
        return Double.parseDouble(resArr.get(5));
    }
    @Override
    public List<Double> getMonthData() {
        ArrayList<Double> data = new ArrayList<>();
        for(int i = 6; i < resArr.size(); i++){
            data.add(Double.parseDouble(resArr.get(i)));
        }
        Collections.reverse(data);
        return data;
    }

}
```

VIII. HTTPServer.py

```
serverPort = 15005                                          #Set server port
serverSocket = socket(AF_INET,SOCK_STREAM)                  #Define and bind the server's
socket to port
serverSocket.bind(('',serverPort))
serverSocket.listen(1)                                      #Listen for incoming connection
requests
print('The server is ready to receive')


def windHandler(connectionSocket, addr):
    try:
        cnx = mysql.connector.connect(user=<USER_ID>, password=<PASSWORD>,
                                      #host=<HOST IP>,
                                      port='3306', database='<DBName>')
        cursor = cnx.cursor()
        query = "SELECT * FROM metrics ORDER BY TimeStamp DESC LIMIT 1"
        cursor.execute(query)
        result = cursor.fetchall()
        res = ""
        for row in result:
            for i in row:
                res += str(i)
                res += ","
        res += ";"
        query = "select * FROM averages ORDER BY day DESC LIMIT 30;"
        cursor.execute(query)
        result = cursor.fetchall()
        for row in result:
            for i in row:
                res += str(i)
                if i != row[len(row)-1]:
                    res += ","
            res += ";"
        print(res + '\n')
        respStr = ('HTTP/1.1 200 OK\r\n'          #build response header
              + 'Content-Type: text/plain\r\n'
              + 'Connection: close\r\n'
              + '\r\n')
        respStr += res
        connectionSocket.sendall(respStr.encode())
        cursor.close()
        connectionSocket.close()
    except mysql.connector.Error as err:
        print(err);
        print('Database connection error')

#Function for interacting with a client
def handler(connectionSocket,addr):
    respCode = '200 OK'
    request = connectionSocket.recv(1024).decode()      #receive request from client
    print(request);
    gIndex = request.find('GET')                        #find GET index
    if(gIndex == -1):                                   #check if GET request
        respCode = '400 Bad Request'
```

```
        path = '400errorDoc.html'                       #this server only handles GETs so
400 error
    else:
        path = request[gIndex+4:]                       #grab start of path
        fIndex = path.find('HTTP/1.1')                  #find end of path
        path = path[:(fIndex - 1)]                      #grab actual path
        cwd = os.getcwd()                               #store cwd
        if (path == '/'):                               #if no path given, default to
index.html
            path = cwd + '/index.html'
        elif (path == "/Wind"):
            windHandler(connectionSocket, addr);
            return;
        elif (os.path.isfile(cwd + path)):              #if file exists, create complete
path
            path = cwd + path
        else:                                           #else 404: no file found
            respCode = '404 Not Found'
            path = cwd + '/404errorDoc.html'

    respType,respEncodeing = mimetypes.guess_type(path) #get file type and encoding
    respSize = os.path.getsize(path)                    #get size in bytes
    rType = respType[:(respType.find('/'))]             #get more specific file type
    isImage = (rType == 'image')                        #find if ^^^^ is an image
    if(isImage):
        respFile = open(path, 'rb').read()              #if so read with rb
    else:
        respFile = open(path).read()                    #else just read file

    respStr = ('HTTP/1.1 ' + respCode + '\r\n'          #build response header
              + 'Content-Type: ' + respType + '\r\n'
              + 'Content Length: ' + str(respSize) + '\r\n'
              + 'Connection: close\r\n'
              + '\r\n')
    if(not isImage):                                    #if non-image, append file to
response
        respStr += respFile
    connectionSocket.sendall(respStr.encode())          #send response

    if(isImage):                                        #if image...
        connectionSocket.sendall(respFile)              #send image separately
    connectionSocket.close()                            #close socket
#Main
#Listen for new connection requests
def main():
    try:
        while (1):
            connectionSocket, addr = serverSocket.accept()      #accept requests
            try:                                                #create new thread per
client
                _thread.start_new_thread( handler, (connectionSocket,addr))
            except:
                print('Error: unable to start thread')
    except KeyboardInterrupt:                                   #catch keyboard interrupt
        print('Terminating server. Goodbye!')
        sys.exit(0)

if __name__ == "__main__":
    main()
```

IX. dataReq.py

```
payload = {'ADDRESS':'4','TYPE':'DATA'}
host = 'http://<AcquisuiteServerIP>/setup/devicexml.cgi'

newWind = False
windDirection = 0
data = {}

def wind():
    global newWind, windDirection, data
    while(1):
        try:
            print("New Wind Data")
            baseurl = "https://query.yahooapis.com/v1/public/yql?"
            yql_query ="select wind from weather.forecast where woeid in (select woeid
from geo.places(1) where text='boone, nc')"
            yql_url = baseurl + urlencode({'q':yql_query}) + "&format=json"
            result = urlopen(yql_url).readall().decode('utf-8')
            data = json.loads(result)
            #newWind = True
            windDirection = data['query']['results']['channel']['wind']['direction']
            #json.dumps(data)
            time.sleep(600)
        except Exception as e:
            print(str(e))
            time.sleep(60)
def dbInsert():
    print('dbinsert called')
    global newWind, windDirection, data
    time.sleep(1)
    while(1):
        try:
            cnx = mysql.connector.connect(user='<UserName>', password='<PASSWORD>',
                                                    #host='<HOST_IP>',
                                                    port='3306', database='<DBName>')
            cursor = cnx.cursor()
            r = requests.get(host,params=payload,auth=('<USERNAME>','<PASSWORD>'))
            root = ET.fromstring(r.text)
            data = ""
            data += "'" + root.find('./devices/device/records/record/time').text + "',"
            data += root.find("./devices/device/records/record/point/[@name='Rotor
Speed']").get('value') + ","
            data += root.find("./devices/device/records/record/point/[@name='Inverter Real
Power']").get('value') + ","
            data += str(windDirection) + "," + str(windDirection) + ","
            data +=
root.find("./devices/device/records/record/point/[@number='13']").get('value')
            query = "INSERT INTO metrics VALUES(" + data + ") ON DUPLICATE KEY UPDATE
TimeStamp = now(); "
            cursor.execute(query)
            cnx.commit()
            query = ("INSERT INTO averages (Day, Average) SELECT DATE(TimeStamp),
AVG(PowerOutput) FROM metrics "
                        "WHERE DATE(TimeStamp) = UTC_DATE() "
                        "ON DUPLICATE KEY UPDATE Average = (SELECT AVG(PowerOutput) "
                        "FROM metrics WHERE UTC_DATE() = DATE(TimeStamp));")
            print(query)
            cursor.execute(query)
            cnx.commit()
```

```
            time.sleep(15)
        except ConnectionError:
            print("Connection error")
            time.sleep(10)
        except KeyboardInterrupt:
            print('Bye\n')
            sys.exit(0)

        except Exception as e:
            print(str(e))
            time.sleep(10)
def main():
    try:
        _thread.start_new_thread(dbInsert,())
        _thread.start_new_thread(wind,())
        while(1):
            time.sleep(60*60)
            pass
    except Exception as e:
        print(str(e))
        print('error unable to start thread in dbHandler')
if __name__ == "__main__":
    main()
```