A Simple Weather Monitoring System

Rafael A. Okiishi Paulucci
CS 4365 – Introduction to Enterprise Computing – Spring 2020
Georgia Institute of Technology

***Abstract*—We discuss the design and implementation of a near-real-time weather monitoring system, *MicroWeather*, based on a web application, open hardware and open software.**

***Index Terms*—meteorology, weather, web and internet services**

I. Introduction and Motivation

Weather monitoring is critical for aviation, agriculture, construction and several other industries, as well as in everyday life. While many online weather services exist (many of which are free), not every part of the world is covered by accurate weather models, high-resolution satellites or a dense network of weather stations. The infrastructure conditions of some regions, as well as funding available in certain governments and companies, preclude universal usage of commercial weather monitoring services.

Free-of-charge websites (for example, *Weather Underground* and *The Weather Channel*) feature intrusive ads, are not friendly to mobile users, often require registration, and offer results derived from forecast models, not on-site monitoring.

Commercial weather station hardware in the U.S. ranges from around 120 USD for a basic model on Amazon to several thousand dollars a month for a professional subscription service like *WeatherSTEM*. However, even basic stations may only be available at a higher cost in other countries, and be locked to a vendor platform. This lock-in makes long-term support, customizability and expansibility challenging.

In this context, a combination of open-source software and open hardware can be employed to reduce costs (especially with unofficial, spec-conforming hardware) and compensate for the lack of local and instant accuracy in weather monitoring, particularly in regions far from urban centers.

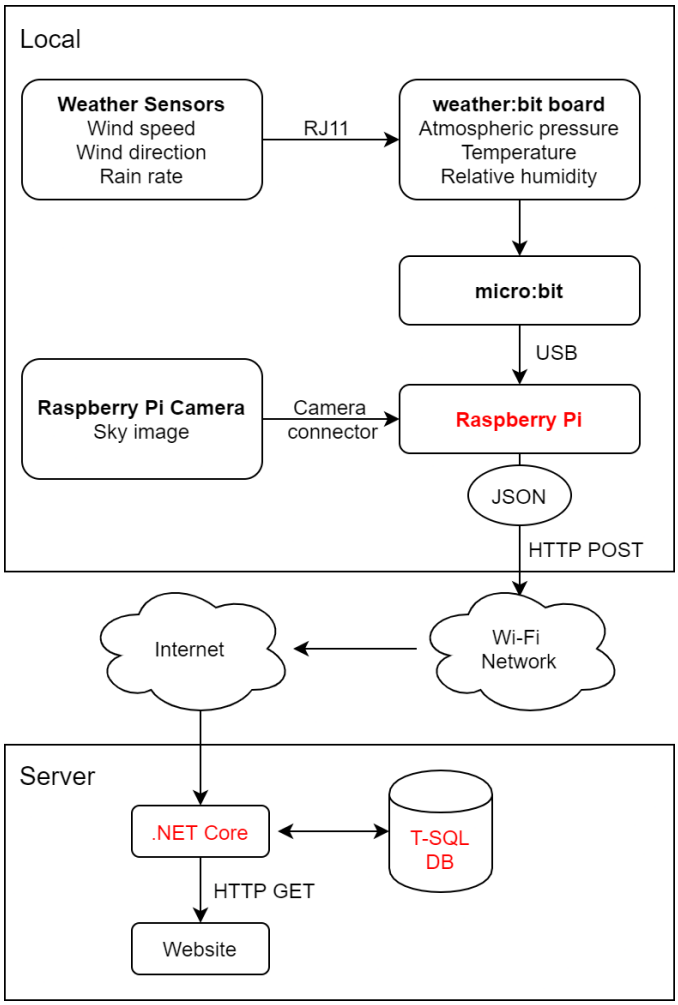

Figure 1: MicroWeather Architecture

II. Implementation

A. Hardware

The basic MicroWeather hardware is comprised of the following parts:

1) BBC micro:bit, a small single-board computer powered by an ARM Cortex M0 CPU with connectivity via 2.4 GHz radio, micro USB, Bluetooth Low Energy, a serial interface, I2C and a 20-pin edge connector for accessory boards. *Sensors: compass, accelerometer, light level*
2) Raspberry Pi 3B+, a small single-board computer featuring Wi-Fi, Bluetooth, micro USB, HDMI, GPIO, I2C, SPI and a camera module.
3) Raspberry Pi Camera Board v2, an 8 MP camera module.
4) SparkFun weather:bit Board (DEV-14214): the interface between the micro:bit and weather station. *Sensors: temperature, relative humidity, atmospheric pressure*
5) Weather Meter Kit (SEN-15901): wind vane with 8 positions, anemometer and rain gauge connected via RJ-11 to the weather:bit board.
6) SparkFun Soil Moisture Sensor (SEN-13322): connected via analog 3-wire to the weather:bit board.
7) Waterproof Temperature Sensor (SEN-11050): soil temperature sensor, connected via analog 3-wire to the

weather:bit board.

The last four components above are included in the SparkFun micro:climate kit (KIT-15301). Parts 6 and 7 are currently unused.

The following values can be monitored with this setup:

- Temperature
- Relative humidity
- Barometric pressure
- Wind speed
- Wind direction
- Rain rate *(disabled)*
- Sky picture
- Soil moisture *(disabled)*
- Soil temperature *(disabled)*

B. Software

Note: the full MicroWeather code is available at https://github.com/rpaulucci3/microweather and licensed in BSD 3-clause. Please read the included README on GitHub for the folder structure and build instructions.

Information is presented to the user via a responsive web application, hosted at https://microweather-dev.azurewebsites.net. The application is based on the .NET Core 3.1 framework with a 32 MB T-SQL database, and uses the "Web App + SQL" profile on Microsoft Azure.

MicroWeather's code aims for extreme simplicity – the public API has only 3 methods, and the database has only one table, which corresponds to a list of weather observations.

The first is `AddObservation()`, which receives a JSON bundle of information periodically pushed via HTTP POST by the Raspberry Pi, and saves it in the T-SQL database using the Dapper micro-ORM.

640x480 JPEG-encoded sky images are stored as Base64 strings for easy embedding of the most recent picture in HTML. A sample JSON for this method is below:

```
{
 "timestamp": "2020-02-14T01:12:25.193Z",
 "altitude": 700,
 "wind_speed": 0.0,
 "wind_direction": "NW",
 "rain_rate": 0,
 "temperature": 0,
 "pressure": 0,
 "humidity": 0,
 "soil_moisture": 0,
 "soil_temp": 0,
 "image": ""
}
```

The other two methods are `GetObservations()`, which returns a list of all saved observations (with the attributes above), and `GetLatestImage()`, which returns a Base64 string that represents the latest JPEG-encoded sky picture.

The front-end is designed for fast loading, and based on Bootstrap and jQuery, with some additional components for styling data tables. There is only one MVC View and Controller, "Home", as the .NET Core app is single-page and very simple.

The micro:bit runs a binary script compiled from a Blocks/JavaScript recipe built with Microsoft MakeCode. An extension developed by SparkFun abstracts away pins needed for data collection from the weather sensors, making the code more readable. The script then does the appropriate math needed to convert data into metric units and concatenates the response into a JSON-like string, which is printed to `/dev/ttyACM0` on the Raspberry Pi.

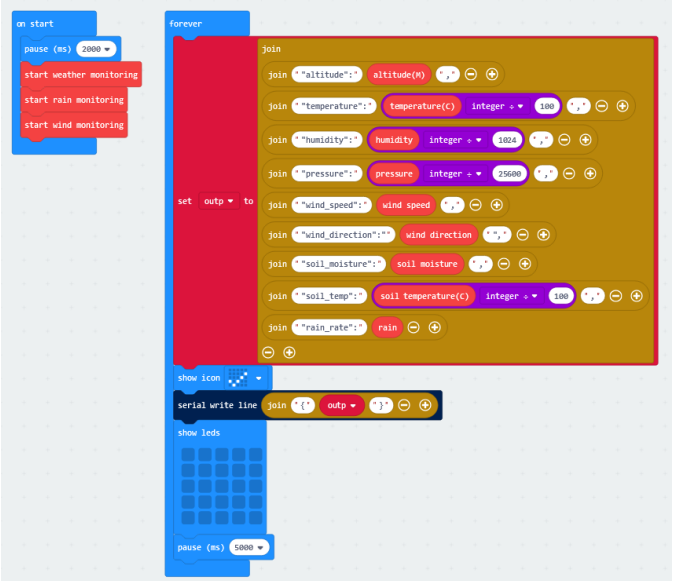

Figure 2: MakeCode Blocks Script

(If Fig. 2 is too small, the code is available in this format at https://makecode.microbit.org/10986-04839-99956-28471.)

The Raspberry Pi runs a single Python script, `microweather_rpi.py`, which consists of an infinite loop that queries the micro:bit via USB (`/dev/ttyACM0`) for data needed to build the JSON that is sent to `AddObservation()`. It then performs the required HTTP POST.

Upon accessing the website, the user sees a page similar to the one on Fig. 3. The latest sky picture is prominently featured, with a sortable and searchable table of observations below it. The top right corner has a link to the project's GitHub repository.

III. Evaluation and Issues

During development, the following major changes happened:

1) The rain rate sensor was excluded from the initial deliverables
2) The backend was switched from Flask + SQLite to .NET Core + T-SQL
3) The Raspberry Pi 3B+ was used instead of the Pi Zero W

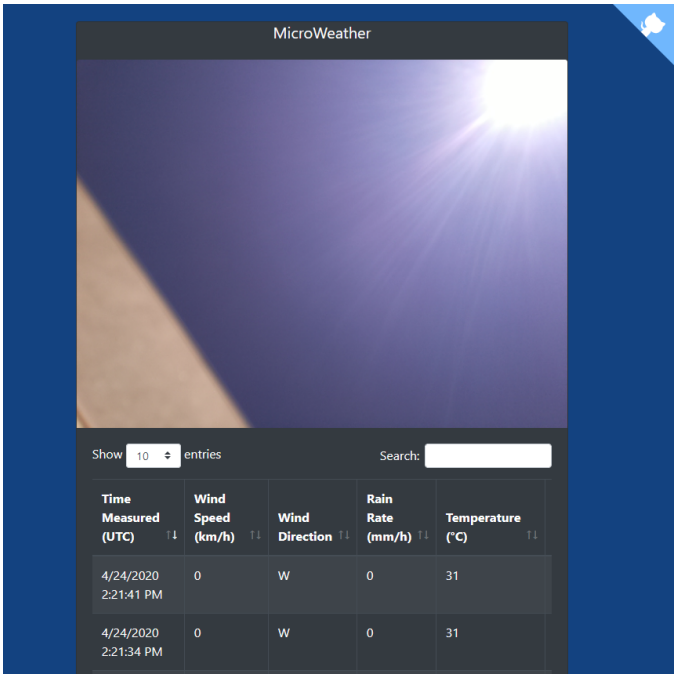

Figure 3: Website Screen Capture

4) It was not possible to use a cell phone power bank with the above change
5) Planned UI improvements were cancelled due to time constraints

It was not possible to operationalize the rain rate sensor because its mechanism proved to be somewhat unreliable with water tests, and forced travel due to COVID-19 reduced available time for basic component waterproofing, a requisite for tests with real rain.

SQLite lacks important types, such as `DateTime`, and does not interface well with Flask on Microsoft Azure. Since .NET Core supports T-SQL (a much more robust DB) with very few extra packages, the SQLite approach was not pursued further. Moreover, Azure offers much better monitoring and performance with a .NET Core application than with a Python one.

The Raspberry Pi Zero W had some Wi-Fi range and processing power issues, and was removed in favor of a Pi 3B+. However, the latter is less energy-efficient, which prevented the usage of a cell phone power bank.

The addition of charts and trends (possibly via *Highcharts*) had been initially planned, but had to be scrapped in the minimum product due to time constraints.

Due to repeated disassembly and reassembly of the weather station, it was not possible to gather a consistent, long-term evaluation profile. However, as of the writing of this report, the sensors reported a temperature of 31 degrees Celsius, relative humidity of 38%, and wind speed peaking at around 2 km/h. The official governmental report from *CPTEC*, Brazil's federal weather prediction agency, stated a max. temperature of 29 degrees Celsius, relative humidity of 41%, and wind speed peaking at around 4 km/h for the nearest region.

The measured sensors were working nominally and within expected bounds. Since no rain or severe weather was experienced, this aspect remains to be evaluated, along with the currently disabled sensors.

IV. Conclusion

The MicroWeather platform has the potential to serve as a model for a simple, low-cost, customizable and reproducible weather monitoring system. The expected use case is to enable individuals, small farmers and companies to perform near-real-time, local weather monitoring with a reasonable level of accuracy without depending on a proprietary platform.

The KISS ("keep it simple, stupid") principle, applied to both the back-end and front-end, coupled with the open nature of the project, opens up possibilities for future work by third parties both in software and hardware. System builders can build upon the basis given here with different aims: cost reduction, increased accuracy, component integration, and others.

Open-schematic components theoretically allow implementers to build their weather station locally, as long as they have access to small-scale circuit printing facilities, and to the needed integrated circuits. Specification-conforming "clones" of the micro:bit and Raspberry Pi might also be a factor in driving down cost for those who cannot afford officially branded hardware.

Further evaluation is required to determine the feasibility of reproducing and modifying the electronic components used for MicroWeather. One avenue of research could be to remove or streamline select components and make each weather station model tailored to a specific use case.

Appendix

Demonstration website: https://microweather-dev.azurewebsites.net

Demonstration video: https://youtu.be/GA5nxOMETFs

GitHub repository: https://github.com/rpaulucci3/microweather

MakeCode Blocks Script: https://makecode.microbit.org/10986-04839-99956-28471