CS 307 -- Software Engineering
Design Document

Personal Pi in the Sky

Team 7

Anna Benjamin, Kathryn Frankewich, Austin Klasa, Bridgette Kuehn, Matt Molo

Table of Contents

PURPOSE:

Weather is currently communicated to the general population through means of weather channels and websites that display forecasts for a broad locale. However, some people may prefer a more accurate, personal weather data for their specific location (i.e. people who live in rural areas). Additionally, weather enthusiasts may want to gather their own weather data separate from weather channels and websites. Our weather utility and corresponding web application would allow for such personal gathering of the weather. By having the weather utility interact with a web application, our system would enable its users to interact with each other.

This design document thoroughly describes our functional requirements.

1. Data will be collected from users' raspberry Pi weather stations
    a. The data will be sent from multiple Pi stations to a local server
    b. The data will be hosted in a database
    c. The data will be displayed in both graphic and tabular form
2. Users can look at historical data extracted from the database
    a. Users will search by date
    b. Users will search by location
    c. Users will search by temperature
    d. Users will search by pressure
    e. Users will search by wind speed
    f. Users will search by humidity
    g. The database will be able to scale to work with as many stations as possible
3. The user interface will be implemented as a web application
    a. On the app, users will be shown the current temperature
    b. On the app, users will be shown the location of the said temperature
    c. On the app, users will be shown the current humidity
    d. On the app, users will be shown the current pressure
    e. On the app, users will be shown the current wind velocity
    f. Users will be able to view the app both on mobile and desktop
4. The web application will be user-friendly
    a. Pages will load in 3 seconds or fewer
    b. No ads will be present on the web application
    c. Layout will be neat and organized for efficient use
    d. Users will be able to learn how to create their own weather station

DESIGN OUTLINE

Overview and Components

- Clients
    - Raspberry Pi Weather Station
        - The Raspberry Pi client will collect data using components attached to the weather station using weather sensors.
            - The weather sensors will get data from the environment and will be logged inside of the Pi.
            - There will be multiple sensors attached to each individual Pi.
        - The Raspberry Pi will be able to send data that was obtained through the weather sensors to the server.
        - The Raspberry Pi will publish data to the server.
    - Web Application
        - The web application will be refreshed by the server.
        - The web application will display information to the user.
        - The web application will give the user the ability to search through data and view both historical and current data from the server.
- Server
    - The server will handle data sent to it by the Raspberry Pi.
    - The server will be able to ping the Raspberry Pi to get current data.
    - The server will send data to the database.
    - The server will make calls to the database based on requests selected by users in the web application.
    - The server will handle all algorithms and calculations necessary for the data.
- Database
    - The database will store all information sent to it from the server.
    - The database will store information associated with each weather station, including the generated ID, the owner, alias, location, and whether the owner has decided to share data with other users or not.
    - The database will store weather data such as the date, time, temperature, humidity, barometric pressure, and wind speed.

Architecture

The Personal Pi in the Sky will use a combination of two models in order to efficiently collect, manipulate, and show large amounts of data. The first model that we will implement will be that of Client-Server. We will have one singular server that will communicate between two different types of clients. The clients will be the Raspberry Pi weather stations that collect the data and the front end web application. Additionally, we will be implementing the Model View Controller structure in order to effectively load the application in very quick manner.

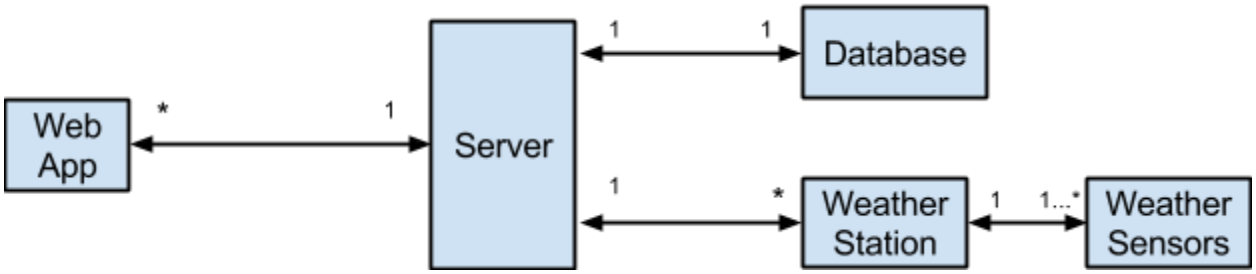

Figure 1. UML Diagram

We will have multiple Raspberry Pis configured as weather stations. These Raspberry Pis will have multiple sensors connect to each one. The sensors will send data to the Pi weather stations. The Pi configures the data and then sends the data to the one server. The server then formats and sends the data into the database. When a user uses the web application, the app will request weather data from the server. The server then handles the request and in turn requests the selected data from the database. The database will then respond to the request. Once the server has the applicable data, the server will send the updated information to the web application page.

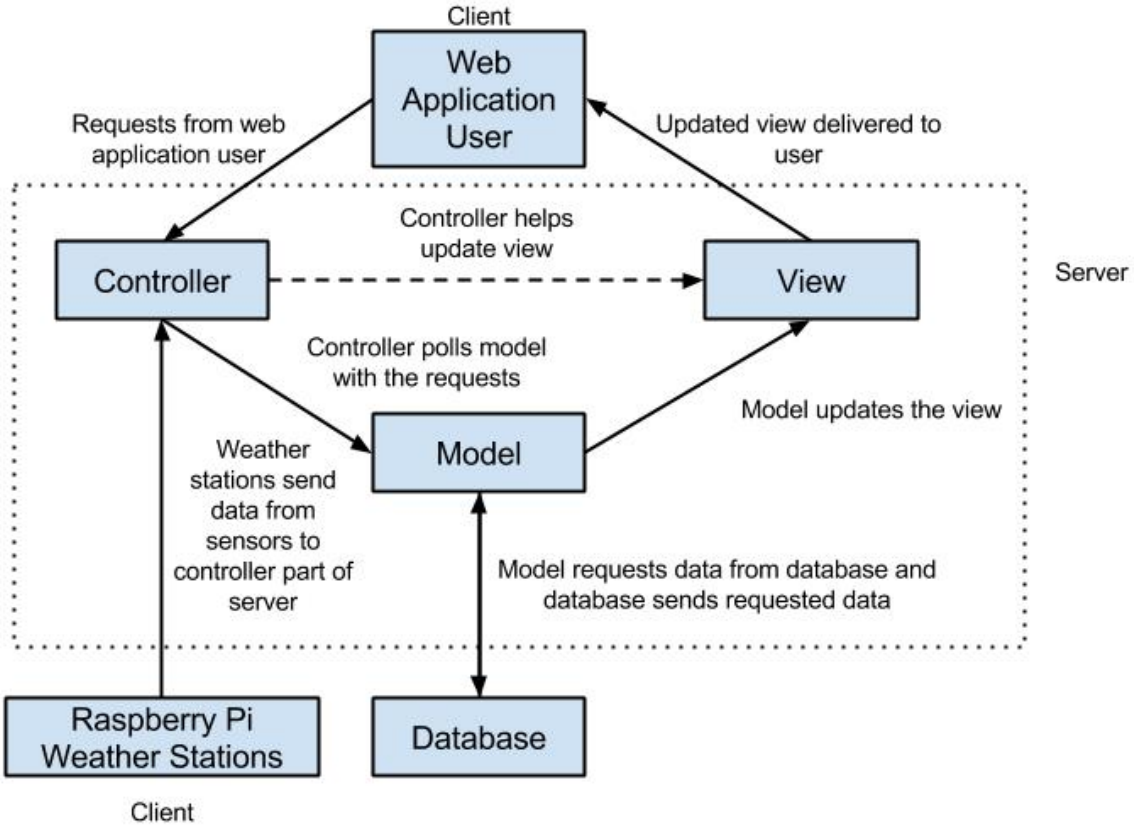

Figure 2. Client Server and Model View Controller Integration

The above figure depicts how the Personal Pi in the Sky will integrate the Model View Controller and Client-Server models in its implementation. As previously mentioned, there are two types of clients, the weather stations and the front-end web application users. The server for these two clients will contain the Model, View, and Controller. The server interacts with the weather stations to get data and then sends the data to the database.

DESIGN ISSUES

Issue 1: Database Selection

1. MySQL
2. Oracle
3. MSSql
4. mongodb

Decision: Overall, our first decision was to use a relational database, because our group was more experienced with them, so mongodb was out. Out of the last 3, we chose to go with MySQL specifically, because while all of them use the SQL language, we were more experienced with setting up and using MySQL on the server.

Issue 2: Server requesting information from weather station or weather station sending information

1. Only get data by calling an API on the Pi
2. Only get data by publishing the data to the database
3. Have both options available

Decision: At first, the option we chose was to have the data collected on the Raspberry Pi be accessible via a public REST API. This has a few advantages because people can build their own web apps without the need for any PHP or backend and use JavaScript to get the data. However, in our case having the Pi publish data to us is easier to manage, since we don't need to have a registration to add the Pi to our list of what we poll and don't have to set up the script to poll the devices. However, now we don't get the advantage of an open REST API. In the end, we decided both of these things were valuable, so we are having both options available to publish and request data from the Raspberry Pi.

Issue 3: Web application implementation

1. JavaServer Faces
2. PHP

Decision: The group has decided to use PHP, HTML, CSS, and JavaScript to implement the web application. Using JavaServer Faces would create more overhead for the group since it would take longer to set up and learn how to use effectively. Utilizing PHP allows the HTML forms to gather query information from the web application user. PHP will then be able to form a query to be sent to the database.

Issue 4: How frequently the Raspberry Pi will send its data to the server

1. Every 1 minute
2. Every 5 minutes
3. Every 10 minutes
4. Every hour
5. Every day

Decision: We have decided that our Raspberry Pis should output their data every ten minutes, because ten minutes would allow for the data to be up-to-date, and for the server to not be too constantly updated.

Issue 5: Raspberry Pi API implementation

1. Java
2. C
3. NodeJS

Decision: In this case, we needed a language that could talk to the hardware sensors and interact with the web. This will be one application that will get a request over the web, grab its sensor data and return it in json form. C works very well for hardware and grabbing data from the sensors. NodeJS works very well for the REST API, but would not work well with the sensors. We decided on Java, because it works well as in between. It's low level enough to talk to the sensors and has great libraries for working with the web.

Issue 6: How Users Register their Raspberry Pis

1. Users register their Pi via the web application
2. Users register their Pi during setup of their Pi

Decision: We have decided that users will register their Raspberry Pis when they are initially setting up their weather station. They will be able to set a unique alias for their Raspberry Pi and decide whether to make their data public or private in the web application. If the owner decides to set their weather station to private, the owner can view their data in the web application by entering their weather station's alias. This would be a better option to implement than having a user register their Raspberry Pi through the web application, because they will have more control of their data from the Raspberry Pi. Registering the Raspberry Pi upon setup would also be easier to implement since it requires less additional communication between the system components.

DESIGN DETAILS

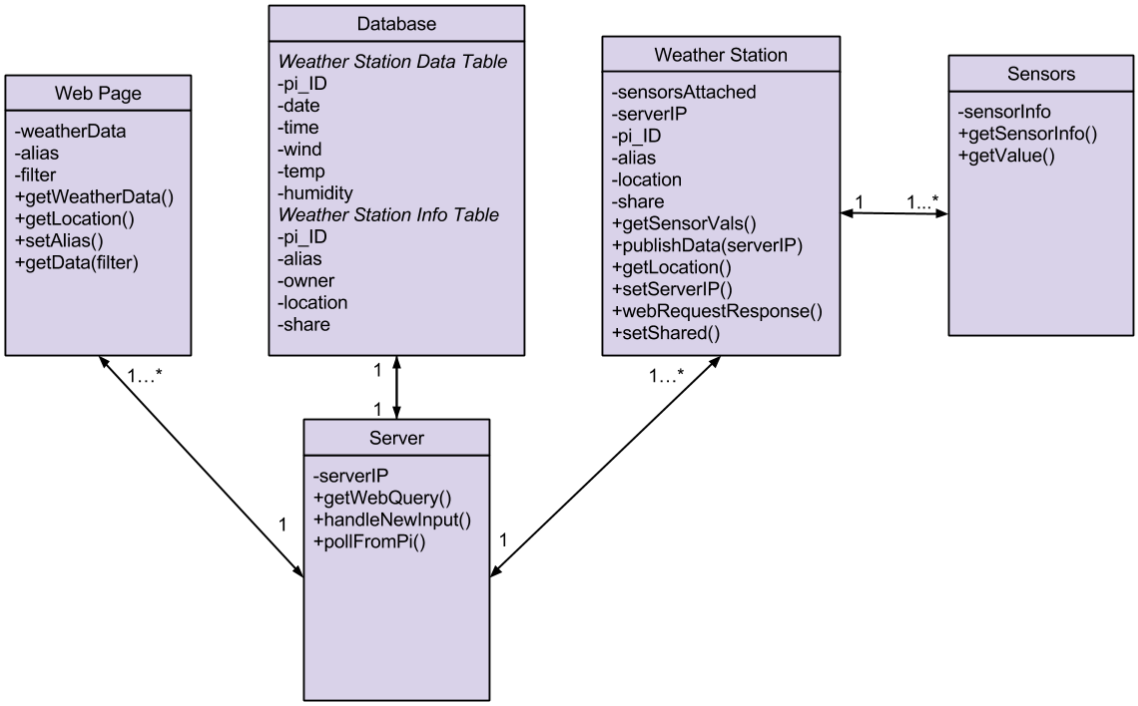

Figure 3. Class Diagram

The figure above is the class diagram for the Personal Pi in the Sky. It includes the classes that will be used in the implementation of the project. Each of the classes includes member variables and the methods that will allow communication between classes.

Web Page: The web page is the main view of the application. It uses the weather data for the display of the webpage. The webpage may also contain a weather station alias, which is submitted from a user that has chosen to keep their weather station data private. The webpage has multiple functions to update values. getWeatherData() will query the server for weather data based on the weather station that is closest to the user. setAlias() will be used internally when a weather station owner, who has kept their data private, wants to view their data. getData(filter) will let users search for more specific data in our database, depending on dates, temperatures and other values.

Server: The server is the entity that will manage the database, handle the website, and the data being pushed from the weather stations. It is identified by its own IP address, serverIP. Web requests will be queries via the member function getWebQuery(), which will connect to the database and return the requested data to the webapp. handleNewInput() handles weather data that is pushed from the weather stations to the

server, and updates the database. If needed, pollFromPi() will be able to poll a specific weather station for its data, instead of waiting for data to be pushed.

Database: The database stores all of the information about a weather station client by providing an info table and data table for the weather station. The information table will have basic information on each weather station, and no duplicates. The data table will hold every record of weather station data collected, which can be filtered through by a specific weather station or other parameters. The weather station info has the following features: ID, alias, owner, location, and share. The weather station data table has the following features: ID, date, time, humidity, pressure, temperature, and information about the wind.

Weather Station: Weather stations involve communicating to sensors and responding to api requests. getSensorsVals() updates the weather station's values for each sensor that it has. webRequestResponse() will respond to API requests. Weather stations also have all of the features that will be stored inside the database as mentioned before.

Sensors: Sensors is a simple class that is meant to be extended to the specific sensors. Depending on the type of sensor (temperature, humidity, etc), getValue() return format will be different as well as its implementation of reading the sensor's value. getSensorInfo() will return basic information about the weather station's sensors.

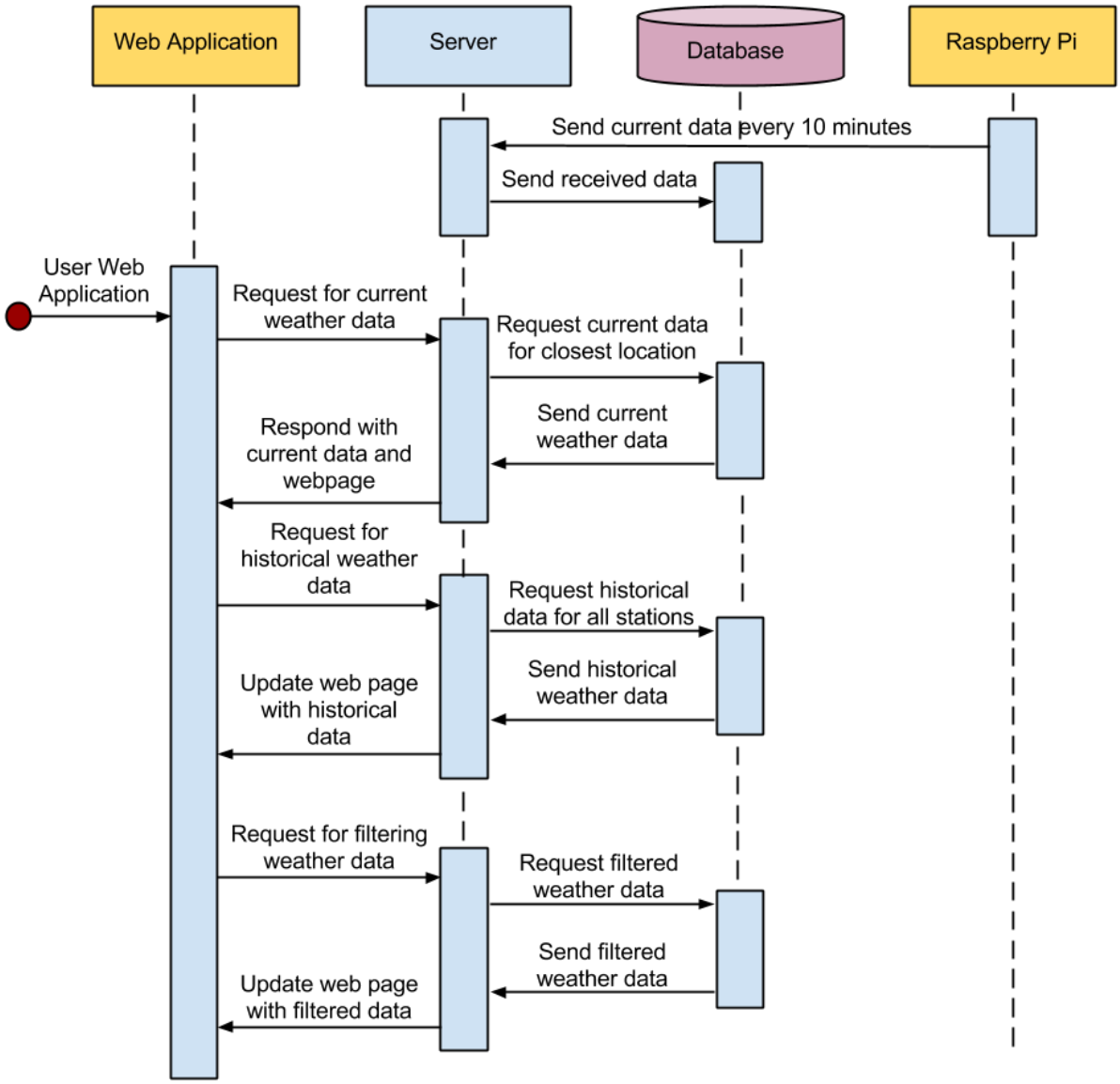

Figure 4. Sequence of Events Overview

The above figure shows the interactions of components as our system is in use. The Raspberry Pi weather station will be on a continuous loop sending data to the server every 10 minutes. The server will then store that data in our database. While these actions are running, a user may start the web application. Once the user starts the application, the application can make requests to the server, i.e. for current, historical, or filtered weather data, which will in turn request data from the database and then update the web application view.

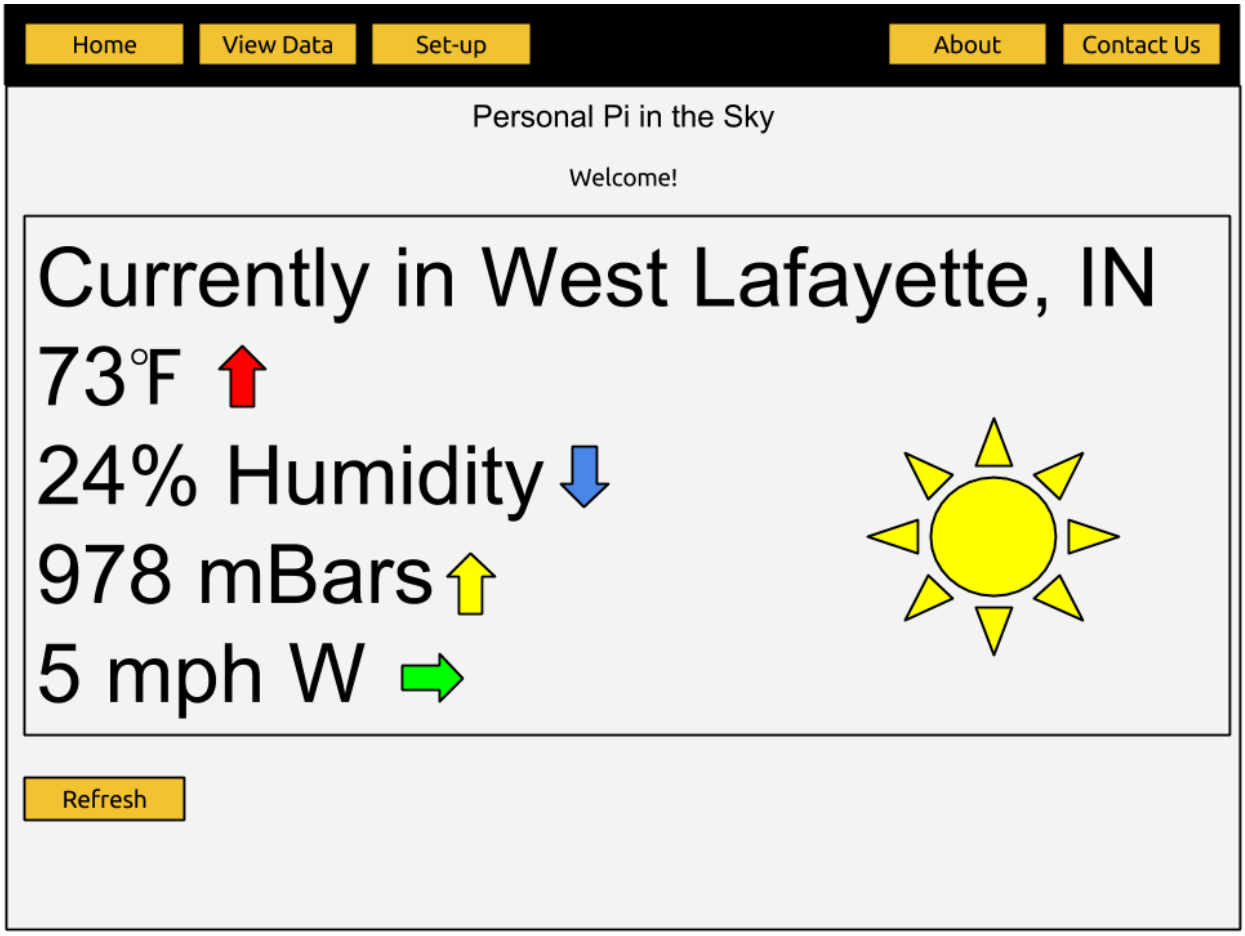

Figure 5. UI Mockup

The figure below represents the homepage of the web application. This page will show the current weather data for the weather station that is located nearest to the web application user. It displays the location of the weather station providing the current data. It also includes the temperature, humidity, barometric pressure, wind speed, and wind direction. Furthermore, the page will display graphics that correspond to the current weather conditions.