1 Technical Review

Jack Neff CS 461 Green Smart Gardening System

2 Introduction

I am in charge of the display and storage of data, as well as the implementation of our application as a web-based UI. Our project involves collecting environmental data from a garden with a microcontroller and then displaying this data to users. For starters we need a stack with an HTTP server, a database, and a PHP interpreter. We also need to select the correct database storage engine for our application, and a graphics application that will allow us to display our data in interesting ways. Users will login to our web application where they can view interesting and colorful displays of data sotred in our database that came from wi-fi transmissions from a microcontroller.

3 Server Database Interpreter Stack

Candidates: XAMPP, WAMP, WPN-XM

To build and test our site, we will need a database, an HTTP server, and an interpreter for our source code. Nowadays packages called web server solutions stacks like contain all these things in one application. This greatly eases configuration and simplifies the interconnectivity of this type of stack.

We had three stack candidates. XAMPP (Cross-Platform Apache MariaDB PHP Perl), which is a popular stack usable by any operating system, WAMP (Windows Apache MySQL PHP), and WPNXM, which is similar to XAMPP but lacks Perl, and uses NGINX instead of Apache for the server. It boasts compatibility with a number of useful third-party utilities.

Starting with the web server, we all agree that we are more familiar with Apache than NGINX, and so choosing NGINX would represent an opportunity cost of learning a new API. According to NGINXs web site, their engine is lighter and faster than Apache and is actually designed specifically to address the shortcomings of Apache servers, mainly in terms of speed and efficiency. We will not have vast amounts of data moving to and from our server, so speed not high on our list of priorities for a server. In the pursuit of spending our design period as efficiently as possible, we would prefer an Apache server.

Regarding the database, we are faced with a similar situation. All three of us have more experience with MySQL than MariaDB, but MariaDB is like a MySQL+, basically a copy of MySQL (even written by MySQLs original author) but with added features and better performance. After consideration, we decided we would prefer to use MySQL over MariaDB, but not to the same degree that we preferred Apache over NGINX. If faced with a situation where we needed to choose one of the latter options, we would pick MariaDB over NGINX.

At this point WAMP is looking like our most suitable choice. We are going to be using PHP, which all three stacks support, so this is not a factor. Additionally we can disregard XAMPPs Perl capability, as we have no use for it. All that said, WAMP has the features we prefer: MySQL and Apache. However, WAMP (and WPN-XN) are both Windows-only, while XAMPP can be used on any platform. We plan to discuss whether it is more important to be cross-platform, or comfortable in the developing environment.

We will be using a relational database with our main focus on clear organization and ease of access. Speed of access will not be important because our sensors are optimized for long battery life, so additions will take place at long intervals, and deletions will happen very rarely or not at all until memory constraints restrict the size of the database. Data sets used to generate views such as tables and graphs will be small enough that generation wont take longer a few seconds. In order to implement our preferred structure, foreign key support will be required, as well as transaction support.

Because our database management system criteria are not very stringent, we have opted to go with the system we are most familiar with, MySQL. A different DBMS would have such a minute effect on the implementation of our project that we decided the opportunity cost of installing and researching a new system was too high to choose anything else.

4 Database Storage Engine

Candidates: InnoDB, MyISAM, NBD

Our database has few outstanding requirements. Speed of access will not be important because our sensors are optimized for long battery life, so additions will take place at long intervals, and deletions will never occur at a faster rate than additions. Data sets used to generate views such as tables and graphs will be small enough that generation wont take longer a few seconds. For the purposes of our project, we will focus less on building a perfectly suited database and more on using the database as an organizational tool to help us design user interfaces and analysis algorithms. In order to implement our preferred structure, foreign key support will be required, as well as transaction support.

Looking at our candidates, we first looked at what set each of them apart. InnoDB is the only one we have all worked with before, and it has both foreign key and transaction support, so we started from there as looked to see if MyISAM or NBD had any notable advantages. Right out of the box we realized MyISAM was probably not a good fit, as it doesnt have foreign key or transaction support. It also does not recover from crashes as well as InnoDB or NBD. Its defining feature is full text search within a database, but for this project that will likely never be needed.

NBD was a slightly better candidate. It supports transactions (non-transparently) and foreign keys, and has a better crash recovery than InnoDB. It is also faster than InnoDB and is designed with real time performance in mind. However, speed is not paramount to our project and real time performance wont carry

any advantages in our particular case. As a kicker, NBD does not have MVCC (multi-view concurrency control) meaning if more than one person is manipulating the database at once, it can result in the access of incorrect or corrupted data. InnoDB does have MVCC, which will allow all three of us to work in the database simultaneously.

5 Graphical Display

Candidates: Gephi, D3js, Tableau

One of our primary goals is to display our collected data to the user in a way that is easy to understand and can inform their agricultural decisions. Data displays should be easy to create, easy to understand, and be viewable on the web-based user interface. In addition, graphs should be automatically generated from data in the database. A user should have the capability to create their own graphs from the data as well, if they want to view a specific time period or examine a certain factor.

Our three candidates for a graphing API are Tableau, a widely known data display platform, D3js, a javascript-based platform, and Gephi, an open-source platform with functionality such as link analysis that searches for and reports relationships between data sets.

Tableau boasts incredible speed and efficiency. In fact, it is especially suited to applications with massive and fast-changing data sets. They have also spent years researching how to efficiently graph data, and how to present data the easiest possible format for humans to understand. The well-researched graphing approach would be a boon for us, but we dont have much need for the speed capability that Tableau is known for. Tableau also requires a subscription, while the other two candidates are open source.

Gephi is a new, open source platform that is on the verge of a 1.0 release (it is currently at .9). Gephi provides most of the capability we are looking for but we are concerned about its stability and lack of community support because of how new it is. In addition, the one functionality we need that Gephi does not appear to provide is some kind of method for the automatic creation and display of graphs.

We are most seriously considering choosing D3js as our data display API. Unlike Tableau and Gephi, which are applications, D3js is a javascript library that creates dynamic data displays in the web browser. In their own words, it allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document (https://d3js.org/). This sounds promising because it could allow us to create fully automated graphs that reflect database data. The one disadvantage compared to the others is that it may be more difficult for the user to create their own graphs, since not every user is familiar with Javascript. It is also open source.