# Green Smart Gardening System Technology Review

Jack Neff

Oregon State University

CS 461, Fall 2017, TR 12pm

Turned in: 11/21/2017

Abstract

This report contains a review of possible technologies for use in the Green Smart Gardening System designed by Jack Neff, Brandon Ellis, and Jiayu Han. The specific areas of focus are the implementation of the web-based client component of the gardening system, which will display and store data. Within these are the possible selections for a web server stack package, a database, and an application to map database information to graphs and tables. The purpose of this document is to convey our selections for these components, as well as why we chose them specifically from a field of possible candidates.

# 1 Introduction

My group is tasked with building a Green Smart Gardening System that automates gardening and farming processes. A scaled up version of our design would tend vineyards or orchards. The hardware used will consist of a sensor array, a solar panel, and a connected microcontroller, as well as a remote desktop computer connected to the microcontroller via wi-fi.

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 an application stack with an HTTP server, a database, and a PHP interpreter. We also need to select the correct database storage engine for our application, to store data in a way that can be graphically displayed. Finally we need a graphics application that will allow us to display our data in clear and interesting ways. Users will login to our web application where they can view interesting and colorful displays of data sorted in our database that came from wi-fi transmissions from a microcontroller.

# 2 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 four stack candidates.

- XAMPP: popular stack usable by any operating system.
  - OS: Windows or Mac
  - Server: Apache
  - Database: MariaDB
  - Languages: PHP, Perl
- WAMP: a similar stack specifically for Windows
  - OS: Windows
  - Server: Apache
  - Database: MySQL
  - Languages: PHP
- WPN-XM: boasts broad compatibility with third-party applications
  - OS: Windows
  - Server: NGINX
  - Database: MySQL
  - Languages: PHP

- LAMP: considered at the request of our clients because of its Google Cloud compatibility.
  - OS: Linux
  - Server: Apache
  - Database: MySQL
  - Languages: PHP/Python/Perl

  [1][2][3]

To simplify the selection process, I broke the stack into three parts: server, database, and operating system. The language will be PHP, but the other three categories were up for discussion. I hoped that by deciding on an ideal server, database, and OS, I could then select the stack that most closely matches my preference.

## 2.1 Server

Candidates: Apache, NGINX

Starting with the web server, all members of my team 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.[4]

## 2.2 Database

Candidates: MySQL, MariaDB

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 the database's size restricts begins to restrict its speed. When this happens, old data can easily be exported to storage files and then deleted from the database. The storage files could still be used by the web application to generate views.

With our database requirements being relatively light, experience becomes a key factor. My team members 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.[5]

## 2.3 Operating System

Candidates: Mac OS, Windows, Linux

All three of these operating systems will meet performance requirements, but Windows and Mac OS are significantly easier to use then Linux. However, some of the members of our group run Windows and some run Mac OS, and each OS excludes the other, while Linux has the unique property that it can be easily run on either of the other two. In this way, a Linux-based stack is similar to a cross-platform stack line XAMPP. Because of this, Linux is the preferred OS of our stack.

To summarize, our ideal web platform would consist of an Apache server, a MySQL database, and a Linux OS. LAMP is the perfect candidate, and its Google Cloud capability aligns with our clients' specifications, and provides easy scalability if our design is ever implemented on a large scale.

**Selection: LAMP**

# 3 Database Storage Engine

Candidates: InnoDB, MyISAM, NBD

Our database has few outstanding requirements. As noted above, speed of access and load are unimportant. 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.[6]

NBD was a slightly better candidate than myISAM. Like InnoDB, it supports transactions and foreign keys, and it reportedly has better crash recovery. 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. Also, 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. Because of this ease-of-use advantage, we have opted to use InnoDB, despite NBD's slightly better crash recovery and speed.[7]

**Selection: InnoDB**

# 4 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.[9]

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.[8]

D3js, unlike Tableau and Gephi, which are applications, is a Javascript library that creates dynamic data displays in the web browser. In its creators' own words, it allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document."[9] 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.[9]

Of the three, we opted to go wth D3js. We like the fact that it is in Javascript, and open source. And we just plain like the way it looks. Gephi is simply too new of an application to use because we don't know how reliable it will be over the next six months. While Tableau is an impressive application, we would prefer not to have to pay for a subscription.

**Selection: D3js**

# References


[1] AlternativeTo. (2017). WampServer Alternatives and Similar Software - AlternativeTo.net. [online] Available at: https://alternativeto.net/software/wamp/ [Accessed 21 Nov. 2017].

[2] Bitnami.com. (2017). LAMP. [online] Available at: https://bitnami.com/stack/lamp [Accessed 18 Nov. 2017].

[3] Koch, J. (2017). WPN-XM/WPN-XM. [online] GitHub. Available at: https://github.com/WPN-XM/WPN-XM/wiki [Accessed 21 Nov. 2017].

[4] Garrett, O. (2017). NGINX vs. Apache: Our View of a Decade-Old Question. [online] NGINX. Available at: https://www.nginx.com/blog/nginx-vs-apache-our-view/ [Accessed 21 Nov. 2017].

[5] Sarig, M. (2017). MariaDB vs MySQL. A Comparative. [online] Blog.panoply.io. Available at: http://blog.panoply.io/a-comparative-vmariadb-vs-mysql [Accessed 21 Nov. 2017].

[6] Dev.mysql.com. (2017). MySQL :: MySQL NDB Cluster 7.5 :: 3.5.1 Differences Between the NDB and InnoDB Storage Engines. [online] Available at: https://dev.mysql.com/doc/mysql-cluster-excerpt/5.7/en/mysql-cluster-ndb-innodb-engines.html [Accessed 21 Nov. 2017].

[7] En.wikipedia.org. (2017). Comparison of MySQL database engines. [online] Available at: https://en.wikipedia.org/wiki/Comparison_of_MySQL_database_engines [Accessed 17 Nov. 2017].

[8] Gephi.org. (2017). Features. [online] Available at: https://gephi.org/features/ [Accessed 21 Nov. 2017].

[9] Sullins, B. (2017). Tableau vs D3 - Which one should I use? - Ben Sullins — Data Geek. [online] Ben Sullins — Data Geek. Available at: https://bensullins.com/tableau-vs-d3-one-use/ [Accessed 18 Nov. 2017].