**Technological Feasibility Document**

LabRats

Project Sponsors: Dr. Terry E. Baxter & Michael Kelly

Team Mentor: Jun Rao

Julian Bell

Remy Brandriff

09 November 2018



**Table of Contents**

1. Introduction
2. Technological Challenges
3. Technology Analysis
	1. Introduction
	2. Web Frameworks
	3. Databases
	4. Hosting
4. Technology Integration
5. Conclusion

1
2
3
3
3
6
6
7
8

**Introduction**

We are a senior capstone team at Northern Arizona University, consisting of team members Julian Bell and Remy Brandriff, with team mentor Jun Rao. Our project is entitled ‘Environmental Laboratory Informatics and Management System,’ with sponsors Dr. Terry Baxter, a professor in the Department of Civil and Environmental Engineering (CENE), and Michael Kelly, from NAU’s Environmental Health and Safety team. They oversee the Environmental Engineering Lab, where students and faculty reserve space and use the provided chemicals and tools to perform experiments. Essentially, the current problem with this setup is that the client does not have a reasonable, consistent tool or application with which to store information about the lab they run for both students and faculty. We are constructing a web application with a robust database backend to allow for students to request lab space and materials, while using the database side of things to prevent conflicts, inventory issues, and the like.

This document outlines our analysis of the project’s technological feasibility–what we need to accomplish with technology, what challenges we may face and how we plan to solve them, and ultimately, what technologies we will be using to create our software. In ‘*Technological Challenges*,’ we outline the high level requirements for the project, and the major technological needs and challenges these create. We address these needs and challenges in ‘*Technological Analysis*,’ in which we analyse and select technologies that will solve fulfill these needs–the web app frameworks, databases, programming and scripting languages, and hosting. Following this analysis, ‘*Technological Integration*’ discusses how we will combine these technologies, integrating the solutions to the individual needs into the final product. Finally, our ‘*Conclusion*’ reviews and summarizes our problems and findings.

**Technological Challenges**

* **We will need a highly reactive user interface in order to request lab space, check lab chemical inventory, and facilitate user registration.**

The web application that we design for this project must satisfy the three above requirements, at an absolute minimum, to be fully functional in the eyes of our client, Terry Baxter (who represents the Environmental Engineering department and its related lab). Upon registration, each user should be able to request use of and space within said lab, in addition to being able to access the entire chemical inventory that exists within.

* **We will need secure user registration, authentication, and management on our web app, as well as different levels of user access and a way for the admin to easily manage that.**

As protection of data is one of the most importantly stressed points by the client, we will have to take steps to ensure the data is both securely stored and processed within our web application. Because the data contained within the application can be quite sensitive -- chemical data, students’ names, and so forth -- it is imperative to make sure that we take steps to protect all of the data coming into and being stored within the framework that we design. In addition, we need an additional “dimension”, of sorts, to the web application that an administrator class can access -- that way, they can make crucial edits to the system, and adjust aspects as needed without any required input from developers or ITS personnel, etc.

* **We will need a secure way for communication between our web app and the database backend, and between our databases and the existing BioRaft health and safety and A.C.I.D. inventory systems.**

The integration of our web framework and database backend with the two systems ITS and the Environmental Engineering Lab already employ, A.C.I.D. and BioRaft, must be completed to satisfy the needs of the client for the project. This should not be an incredibly difficult task, but we will have to think and plan ahead carefully on how we intend to integrate the systems together so that, in future years, the EE department and ITS can continue to build on what we create in the coming six months.

**Technology Analysis**

**Introduction**

As just mentioned, the major issues involve creating an intuitive, responsive, and robust web application to handle requests related to the lab, inventory data, and registration of users, in such a way that is completely secure and facilitates communication between the web application and its related database backend. In the following sections, we detail our thought process and decisions for which web framework and database management software to use, as well as what programming and scripting languages will be used, and where the information will be hosted. In the next sections, we weigh what is the best choice for each respective category, and provide justification for each.

**Web Frameworks**

This software will be a web-based application for maximum utilization and ease of access, so we’ll need a web framework to build it and support it in development and deployment.

We compared fifteen web application frameworks (Angular.js, Apache Tapestry, Apache Wicket, Catalyst, CodeIgniter, Django, Flask, Grails, Kepler, Laravel, Meteor, Pyramid, Ruby on Rails, Spring, Symfony) across ten features (database migration; testing, template, caching, security, admin, and form validation frameworks; Ajax; MVC structure; and open source) to get us started, before looking more in-depth with the frameworks that didn’t get eliminated after the first round. These initial fifteen frameworks span across many different applications; some are used primarily for mobile applications, some can be used for both web and desktop, and others are intended only for web application usage. They also represent nine languages, some we already have experience in and others that may require a learning curve should we choose to utilize them; while having a familiar language is preferable, it was not one of the features we used to determine our framework.



As you can see, we color-coded the cells according to their answer to one of the ten features, and used those to score the initial fifteen frameworks:

* Green cells (yes) = score + 1
	+ This indicates the framework definitely has this feature, either directly with the download or in an easily extensible library. We add up the ‘yes’s first and subtract any ‘no’s from that.
* Yellow cells (plugin) = score + 0
	+ This has no effect on the score; these features were available through a plugin or extension, and have neither a positive nor negative effect on the analysis right now.
* Red cells (No) = green score - 1
	+ These features were absolutely not available for this framework, or would be extremely difficult to obtain. We took this score away from the ‘yes’ score.

After this initial round, we eliminated any framework with a score less than 8, leaving us with nine frameworks with scores of 8, 9, or 10 to examine further. We then eliminated two more (Flask and Angular.js) by narrowing our criteria down to full stack frameworks; Flask is a microframework, which works well for smaller sized projects but maybe not for one of our size, and Angular.js is primarily a front-end framework, which is not expansive enough for our purposes. This leaves us with seven frameworks to investigate: CodeIgniter, Django, Grails, Laravel, Meteor, Ruby on Rails (RoR), and Spring. We looked at these more in depth, ultimately ranking and eliminating them to determine which framework we would use. Our investigation was based on extensive research into each framework, including its documentation, projects that have used it, and user reviews, and ultimately led us to conclude Django was the ideal framework for the project--a strong online community, a familiar programming language, and built-in frameworks and templates useful for our purposes.

After meeting with the client and the ITS team working with him, we were led to consider .NET Core as the software’s framework; NAU ITS will ultimately be supporting and maintaining the product following our capstone, and it’s critical they are able to do so. ITS primarily uses the .NET Core framework with an extensive ecosystem and support team, and raised the concern that they lack Django or the needed Python support in the department. With this in mind, we now compare and contrast Django and .NET Core, to determine the best framework for our purposes.



Ultimately, we will be going with .NET Core as our web app framework out of necessity. We have little to no experience in C# as opposed to extensive work in Python, and this will be a steep learning curve, but we don’t have much freedom in which framework we choose. While using Django would be easier for us and our development, NAU ITS will ultimately have to support and maintain it after we’re gone, and they aren’t able to support Django in the long run as they move towards Microsoft and away from Oracle, so we’ll have to learn and adapt quickly. This will also make it easier to integrate with the existing A.C.I.D. inventory system, which is also built on .NET Core; NAU ITS has presented the option of creating an API for the integration so we aren’t creating an entirely new inventory system that will likely be scrapped in favor of BioRAFT once that becomes an option to ITS.

**Databases**

The project is going to require extensive use of databases in addition to a web framework in order to store inventory systems, chemical data, and so forth; it is essentially one of the only two primary elements of the application. As far as databases go, our options are somewhat more limited; we agreed early on that the best option to use for the project was likely going to be a variant of SQL. At first, after examining Django’s documentation and looking at what is most widely used within that framework, we decided to go with MySQL as our database of choice. There were multiple reasons to this decision:

* We have previous classroom and personal experience with MySQL (via NAU’s CS345 course and other projects)
* It is relatively easier to use compared to most other database systems
* Django documentation shows native support for a small handful of databases, of which MySQL is the most prominent one
* MySQL is ubiquitous throughout the world of software; it is currently, more or less, an industry standard for database management, and as such, has a lot of documentation and resources online to consult for help, suggestions, etc.

Upon meeting with our client on Tuesday, however, we unanimously moved forward with the decision to use .NET Core instead of Django as a web framework tool, due to the fact that most of ITS relies on it as opposed to a Python-based framework and it would make support and design much easier for all parties involved. As a result, we moved from deciding on MySQL to **SQL Server**, a relational database system also using SQL that is developed by Microsoft and integrates perfectly with .NET Core. We did some analysis of SQL Server regardless to get a feel for what some pros and cons might be; we found that although it can be expensive, resource-heavy at times, and their Integration Service can cause headaches for some users, the fact that NAU offers these services to developers at no cost, and that it is fast, stable, and integrates extremely well with other Microsoft products makes it a win from nearly every angle. We will be using this database management system going forward for the entirety of the project for these reasons.

**Hosting**

All web applications, whether software or just websites, need to be hosted on a server somewhere. While there are many options for this, both free and commercial, due to the nature of the project, it will be hosted on an NAU server provided by ITS. We examined other web hosting alternatives, but ultimately determined that NAU’s own services will suffice.

There are plenty of web hosting services that could start us off without needing an application framework, or that would integrate with our frameworks–some free, most paid–but NAU’s services are a blank slate with local support; we decided this was preferable, given the circumstances, than potential issues later on with cost and fine print, whether it affects us specifically or our client. While Linux servers, such as the ones used in the CEFNS college, are an option because .Net Core can run on the Linux operating system, the NAU ITS team affiliated with the project have mentioned the possibility of using Microsoft servers such as Azure, which they already have set up and supported. This also opens up the possibility of using the CAS system for user authentication.

**Technology Integration**

Combining the use of .NET Core and SQL Server will ultimately be a powerful pair of tools with respect to achieving the goals of the project. .NET Core will be able to cover both the front end (the “website” end of things, what users will be seeing), and SQL Server serves as the majority of the back end to store all of the information like inventory data and lab use requests, among other things. In addition to some of the drawbacks mentioned before that we may potentially encounter, we may have difficulty getting initially used to two extremely extensive, wide frameworks. Fortunately, NAU ITS has offered resources for support in learning these tools, and we intend to take advantage of using them to maximize our learning potential with respect to these frameworks.

Fortunately, as SQL Server is fully supported by .NET Core and they are both made by the same company, we don’t anticipate too many integration issues between the two platforms. We can consult resources from both sources, as well as other users and websites online if we encounter any problems that serve as obstacles to progress within the project, or find ourselves struggling with either of the frameworks. Overall, the benefits of using these two in tandem, coupled with the incredibly high amount of support online, as well as NAU ITS’ thorough support for Microsoft services, leads to these products being a more than solid choice for the purposes of this project.

**Conclusion**

The overall goal of the project is to provide the Environmental Engineering Lab with a robust system for managing lab use requests, storing inventory data, and achieving both of these ends in a secure, reliable manner. In addition, our project needs to be built in mind with integrating with BioRaft, an inventory management system already in use by NAU’s Environmental Engineering department, and allowing for an extensive database of chemical storage and information for students and faculty. We begun looking at options for how to achieve these requirements, knowing that we will need both a solid web framework and database management software.

We put careful consideration into both and weighted our options, but given the nature of NAU ITS, we have decided that .NET Core and SQL will be the most effective choices for the purpose of this project. Compounded further to support this decision is the necessity of using .NET Core to facilitate the project’s integration into the NAU software ecosystem. Hosting will be done through NAU’s servers due to the university-centric, insular nature of the project; it will be a free, secure, and reliable choice.

We strongly feel that the use of SQL and .NET Core to provide a solid database backend and web framework will allow us to construct a web application that meets and exceeds these requirements for the project. We’re quite confident and optimistic that we can use this pair of tools to create a valuable end product that has longevity, scalability, and relevance in future semesters as a robust web application to handle all aspects of using the lab and storing valuable related information.