

ECOdgers

Project: *Colorado Plateau Cooperative Ecosystem Studies Unit (CPCESU) Project Management System*

Technology Feasibility Analysis

Overview:

The purpose of this document is to showcase our project at a high-level, present technological and integration challenges, and give our evaluation and solutions.

Team Members:

Colton Nunley
Joseph Remy, Jr.
Jasque Saydyk
Ana Paula Chaves Steinmacher - *Capstone Mentor*

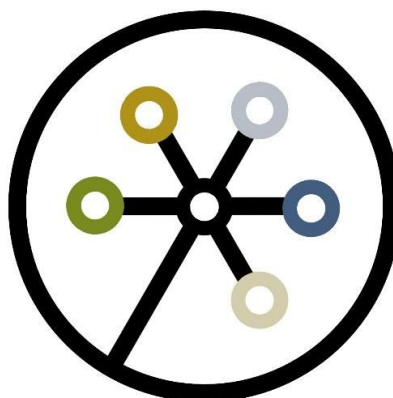
Clients:

Dr. Todd Chaudhry
Laurie Thom
Rebecca Dyball - *Intern*

Northern Arizona University
School of Informatics, Computing, and Cyber Systems

Cooperative Ecosystem Studies Unit
Colorado Plateau

November 9, 2018



ECOdgers

This page is left blank intentionally.

Table of Contents

Table of Contents	2
Introduction	4
Technological Challenges	5
Web Framework	5
Database	6
Front-end Libraries	6
Documentation Platform	7
Hosting Solution	8
Version Control	8
Testing Solutions	9
Security Testing	9
Continuous Integration and Continuous Delivery (CI/CD)	10
Issue Tracking	11
Environment System	11
Technological Analysis	13
Web Framework	13
Database	16
Front-end Libraries	18
Documentation Platform	21
Hosting Solution	22
Version Control	23
Testing Solutions	24
Security Testing	25
Continuous Integration and Continuous Delivery (CI/CD)	26
Issue Tracking	27
Environment System	29
Technological Integration	31
Conclusion	33

This page is left blank intentionally.

Introduction

The Colorado Plateau Cooperative Ecosystem Studies Unit (CPCESU) is a government agency which organizes federal agencies and Native, state, and local governments with non-governmental organizations and universities to complete a variety of conservation projects. From investigating archeology sites to conducting zoological surveys, this helps preserve the nature and history of the Southwest, allowing us to learn more about the impacts we have on our fragile ecosystem. The CPCESU gets dozens of project proposals, hundreds of modifications, and millions of dollars each year to setup, organize, track, and archive these projects. Since their founding, they have used an off-the-cuff approach to project management using email, Microsoft Excel, a Microsoft Access database, and a shared file system drive.

Due to the ad-hoc nature of the forms and data storage, the CPCESU staff are having to spend up to half of their time at work to track these projects. If the number of projects were to double, the CPCESU staff would find themselves overwhelmed and have no time to spend on their other work for the organization. In addition to this, there is no defined structure for data entry into the database, leading to partially completed rows and a lack of consistent, vital information the organization needs to renew its charter. There is no way for project leaders and organizations to modify their contracts to extend the timeline and allocate more funds without emailing or calling the CPCESU, and the CPCESU staff need to be able to search their dataset and export statistics they need to report to inquiring organizations, NAU, and the CPCESU Director. Lastly, certain parts of CPCESU projects depend on budget spreadsheets, approval documents, final reports and more, which are not directly linked to projects and need to be found manually in a massive file system.

ECODers is a senior Computer Science capstone group tasked by Dr. Todd Chaudry and Laurie Thom with developing a website and database solution for the CPCESU with three essential components: a project management system for the CPCESU; a website for showcasing public projects and information; and a system allowing organizations to submit, develop, and track projects. The group is formed under the School of Informatics, Computing, and Cyber Systems at Northern Arizona University. Team members include Joseph Remy, Jr. (Team Lead), Colton Nunley, and Jasque Saydyk and the Team Mentor is Ana Paula Chaves Steinmacher.

This technological feasibility analysis begins by listing out every challenge we expect to face in this project and the various metrics that will be used to determine the solutions we propose. This is followed by the Technological Analysis section where we go into detail about the various solutions available, their various benefits and disadvantages with how they compare to the metrics we chose, and our reasoning for the solution we decided on. We then explore how all of our solutions combine together in the Technological Integration section of the document, giving a broad overview as to how the project will be structured. This document is then concluded with an overview of every technology we are going forward with, our confidence in our solutions, and how our product will address the problems of the CPCESU.

Technological Challenges

Below are all of the various technologies and challenges we need to evaluate to make our solution for our client. For each technology, the challenges are listed with respect to priority with the top being the most important and decreasing need.

Global challenges that apply to all of our technologies include:

- Cost and Licensing
 - We do not have a budget or financing options presently so we will need to prioritize open-source software, free-tier choices, and so on.
- Long Term Support (LTS) and Maturity
 - To prevent issues overall, we want software that is mature with enough built-in features that will positively contribute to development and come with long-term support and consistent updates. Yearly updates are the minimum.
- Documentation
 - For the technologies we pick, we need to make sure there is enough support for development through official documentation, community forums, etc.

Web Framework

Since our solution will have both front and back-end parts, we need a robust web framework as the foundation for our web service and our other technologies. The framework we choose for our project will determine what challenges can be solved efficiently and effectively. Here are some of the challenges and needs for the web framework:

- Security
 - Sensitive data will be archived and needs to be protected by our system and framework.
- Scalability
 - CPCEU wants a product that will last and grow with the organization. Therefore, we need a product that can be modified and scale with ease to solve unforeseen needs and challenges.
- Interfacing with other tech (DB, APIs, libraries, all other tech mentioned)
 - Full stack, app friendly, templating built in, and deployable.
 - The ideal framework will have native database management.
- Language
 - Heavy preference for Python
 - Cuts down on time if we have developed in the language before.
 - Other potential considerations: Java, C#
- Learning curve
 - What would be easiest/hardest for the team to develop with.

- Product interest from the public will ensure longevity and up to date documentation.
- Ability of other people to edit code - Robust and/or ease of use
 - Has to have Extensive Documentation & Tutorials.
 - Large community base preferred.
- Representational State Transfer (REST) capability
 - Able to send data in a stateless, uniform way for other units in the client's national organization.
 - Easy to integrate, use, and maintain.

Database

The data of the CPCEUSU, totaling no more than 1 GB over the past twenty years, consists of various pieces of contractual paperwork and facts about the projects like the budget, project lead, and associated organizations. While the nature of the data doesn't really pose an issue for many modern databases, how the data is organized, accessed, and queried will determine the success of this project.

- ACID Compliant
 - Any database chosen must follow the principles of Atomicity, Consistency, Isolation, and Durability to ensure that every transaction with the database is valid, even in the event of errors.
- Security
 - Since this database will be holding sensitive information that cannot be leaked out to the public, it is crucial that any database chosen must be at the forefront of industry security standards.
- Performance
 - While we do not expect this web server to experience large loads, it must be able to run and grow during its expected 10 year lifespan. Thus, any database selected must be able to scale and handle perform well above what is expected.
- Portability
 - Since there may be leadership changes within the CPCEUSU organization, there is a possibility that CPCEUSU may need to redeploy their server on another system. Thus, any database chosen must be, in some way, portable.

Front-end Libraries

Front-end web libraries have numerous capabilities and can positively influence the user experience of our web application. From creating reactive interfaces to simplifying client-side

scripting, there are many potential benefits and drawbacks on the development side. Our primary needs are to simplify programming JavaScript and Cascading Style Sheets, create a unified theme for the website, and make it easier to implement modern web design principles such as responsiveness, dynamic look and feel, and direct influence by the user's actions. Below are some of the challenge we expect to face while considering front-end libraries:

- Overall Benefits
 - With enough necessity and functionality, we could use libraries to improve user experience, simplify development, and so on.
 - Leveraging the need for another library.
- Saved Time
 - If we can save time without reinventing the wheel, we should try to estimate the potential amount of time saved.
 - We should compare the library to writing our own solution.
- Performance, Usability, and Accessibility
 - The library should not hamper overall performance of our solution, such as load time, and the usability or accessibility for the end user.
- Compatibility
 - We want to aim to support as many browsers and platforms as possible. We should not allow a library to limit us without justification.

Lastly, we have one big, overall challenge to consider when using these libraries: should we use Content Delivery Networks (CDNs) for libraries or use static, local copies? In our analysis, we will explore the benefits and drawbacks of each.

Documentation Platform

Documentation about our solution is essential to two main groups of people: users of the solution and developers. It is important that the documentation is included as a part of the final deliverable to our client. Some challenges we will face include:

- Ease of:
 - Searching for material
 - Ideally, for everyone using the documents, we would like to be able to search and find the correct articles.
 - Reading - Flow and document organization
 - The documents should be tiered and categorized to make it easier to find and trace material down in a logical manner.
 - Page linking is important. We need to consider if this is a manual process or dynamic.
 - Posting and Updating

- When creating a new entry or updating an existing one, we want it to be easy to do without a lot of knowledge or overhead.
- We aim to make this process short and simple for both our client and for future developers.
- Exportability / Portability
 - We need to think about how to move the documents with the solution. This is critical for the product's longevity and upkeep.
 - Example: Google Docs would require a lot of exporting vs. built-in docs would just come with the software.
- Compare and contrast external vs. in-app documentation
 - External documentation could have more functionality and easier management than something we make.
 - In-application documentation would have it always packaged with the application and could be managed by the web app with little frustration and centralized accessibility.

Hosting Solution

Hosting our product will be a big hurdle. We need our host to support all of the tech that we intend to develop with. If we can't deploy our product, then all of our hard work developing would not exist on a solid foundation and be severely affected by hosting problems.

- Compatibility
 - The host will need to support all of our technology.
 - We want a configuration that will minimize integration challenges.
- Longevity
 - Needs to serve the CPCEU for the next ten years.
- Robust
 - Requiring little to no maintenance is a priority since the solution will be maintained by various capstone groups and individuals on and off for years to come.

Version Control

For our team, and eventually for our client, we need a way to collaborate on the project harmoniously without conflicting issues. We want to use Git with a centralized service for our team repositories. For version control, we need to keep in mind the following:

- Accessibility and access to our repositories
 - We want it to be easy and straightforward to pull, branch, and merge content.
 - We also need to consider using Secure Socket Shell (SSH) keys versus username and password authentication.

- Ability to customize and manage repo
 - Ideally, to work cohesively, we want to manage our repositories using code reviews and pull requests instead of manually merging.
- Integration and Webhooks
 - The most important integration would be our Issue Tracking software and this would be the most important aspect of this section.
 - We want to use Continuous Integration, Continuous Delivery (CI/CD) practices to test our solution and assure quality to a certain standard.

Testing Solutions

As with any project, testing is crucial to ensure that the software does what we intend it to do, ensure that future changes do not change the behavior of the software, and to ensure compliance to various requirements. Due to the web based nature of this project, there are various other tests we need to use to ensure we are delivering a high quality product that works for various people in various environments than it is presumed to run on.

- Unit Testing
 - While most modern languages come with robust testing frameworks, we will need to ensure that the language we intend to use does have a robust testing framework and to consider any third party additions that might need to be added to expand functionality.
- Compatibility Testing
 - We need a tool that automatically tests our website on various browsers and screen sizes to ensure that any changes to the website is compatible with various browsers or devices.
- Accessibility Standard Compliant Testing
 - As NAU is moving to be fully accessible online, it is crucial that our website is fully Web Content Accessibility Guidelines 2.0 compliant to ensure that it is able to work for all users and can be integrated with the rest of the NAU system, if need be.
- Linting
 - Lastly, we need CSS and JavaScript linters to ensure that we are writing good code that is free of common mistakes.

Security Testing

This web application will be holding sensitive information regarding the location of archaeological sites and endangered species. If these documents are exposed to bad actors, it will not only severely damage the CPCEUS reputation, but may also see the looting of these sites and the

poaching of these endangered species. Thus, we have to have a proactive solution to identify security faults with the project management system, which will allow us to readjust our design as early in development as possible and to disclose the security faults we are not able to account for with our client. To do this, we need a diverse set of tools for each of the main steps in auditing a web application.

- Information Gathering
 - Involves gathering as much public information about the target system as possible, like identifying the ports used and the services running
- Target Evaluation
 - Identifying and evaluating the target for vulnerabilities
- Exploitation
 - Testing and verifying if the vulnerabilities are real
- Privilege Escalation
 - A deeper attack on real vulnerabilities to see if attackers can access a wider range of sensitive data
- Maintaining foothold
 - Identifies the ability of current security measures to detect unauthorized access and tests what alternative paths can be opened to the application undetected

Continuous Integration and Continuous Delivery (CI/CD)

As with any large software project, though the pieces may work individually, when they have to be combined together in the end, it is inevitable that nothing works together and everything has to be reworked. This has occurred in countless software projects, and it will no doubt happen to ours as well. To prevent this from occurring, we need a Continuous Integration solution that is able to test every branch of the project with all of the tests created so far, and build the website to ensure that with each step of development, the entire project is functioning, or we are notified of any problems.

Related to this, we plan on delivering multiple versions of our website during its development: a stable version that is updated when certain milestones are reached and a development website that is continuously deployed when its development branch is updated. Keeping a development server up to date manually will be time consuming, thus it requires a solution to continuously deliver the website automatically to save us time and allow us to have an auto-built website that is presentable to the client at anytime to showcase the latest features we have implemented.

- Compatible and Configurable
 - Since we plan on using a wide variety of services, any solution must be able to plug into any service we plan to use.

- Flexible
 - As our project matures, our processes will surely change along with it. Thus any solution must be able to change with the system.
- Good UI
 - Must clearly show when and where tests are failing with the system.

Issue Tracking

As a team, we need to manage our project throughout the next year. That being said, we will have requirements to fulfill, features to implement, bugs to fix, and so on. We need a way to track our project and issues and below are challenges to consider when choosing a service:

- Simple project management features
 - At a minimum, we need to be able to open issues, track them using our own workflow, and be able to reference closed issues.
- Integration capabilities
 - We would like to be able to reference issues inside of and outside of the Version Control software we choose. This will give us flexibility to make issues for assignments and for our repositories.
 - Since we want to use Continuous Integration and Continuous Delivery, we want to be able to reference passing and failing builds on certain branches and issues.
- Ability to Scrum Project Development
 - Ideally, we want to apply the principles of Scrum and Agile development. This means that our issue tracking software should have some built-in capability to ease us into this practice.

Environment System

We would like to have some control over our software environments for development and for the production version for our clients. Aspects include creating localized spaces for controlling software versions, maintaining consistency between developers, and managing and testing dependencies at an application level. The three main ways we thought of are containerization (dividing a host operating system into segments), virtualization (virtual machine inside of a host operating system), and physical (dedicated hardware for development with its own operating system, no host operating system or virtualization needed). To maintain different environments, we need to look at the following metrics and challenges:

- Operating System support
 - We need this to work regardless of operating system and be robust enough to last without constant maintenance.

- We also want to make sure there are minimal dependencies (software and hardware) for our system.
- Repo-friendliness - can it be moved / update via Version Control
 - It would be great to have our system move around via some sort of version control (either inside a specific repo or the product's repo). This will help keep developers synced.
- Self-hosted vs cloud
 - Lastly, this should not be a problem for development, but we need to consider having this self-hosted by one of the team members or hosted in the cloud during development.

Technological Analysis

Now that we have gone through our challenges, we explored and selected the different technologies that will solve our technological challenges.

Web Framework

The web framework we choose will be need to have certain attributes and features which are important to us for coming up with solution. The most important features we desire in a web framework are laid out in *Figure 1* below. Some of which include: Security, Language, Dependencies, API among others. There are many frameworks that are available that could solve some of these issues, but we narrowed down to some of the most popular ones.

Django

This is a highly sought after framework due to a few big features including that its developed in Python and its full stack capabilities. Django markets itself as a “batteries included” framework in the sense that once it is installed and configured, a developer can jump right in and start developing. This along with its respected built-in website security and straight forward database management, lends consumers to flock to Django more frequently than most other frameworks.

Ruby on Rails

Rails is another popular full stack framework that makes use of Ruby as its language. Rails offers developer many things that are to be expected with a full stack framework. That includes features like built-in security, database management, and model-view-controller development style. However, there are few experts in the field who know how to efficiently and effectively develop in Ruby, which creates a hurdle for most newcomers that is not worth jumping over.

Flask

This framework is designated as a “micro-framework”. That means that there is not a lot of overhead for web developers. Since there are no restrictions on the manner in way developers develop, there is a sense of freedom and leniency that is lost amongst other restrictive frameworks. This lack of structure though can be a hindrance on larger scale projects, which is a deterrent for some developers. Therefore most of the community that use Flask are using it for personal or smaller scale projects.

Node.JS

This is a Web 2.0 framework that is developed in JavaScript. Node will provide developers with the ability to develop asynchronously providing low latency and streaming of programs. Though Node.JS is not widely considered a full web framework, it is well suited for a foundational supportive tool during development and deployment.

Figure 1: Feature Chart





				
DB Mgmt.	✓	✓	✗	✗
Security	✓	✓	✗	✗
Scalable	✓	✓	✗	✓
Language	Python	Ruby	Python	JavaScript
LTS	2.5 years (Roughly)	Full	Unknown	2 years (Roughly)
License	3-Clause BSD	MIT	BSD	MIT

Figure 1 graphically illustrates our most important technical challenges and needs that have to be solved by the framework for our project.

Why We Chose Django

Based on the results of Figure 1 and our needs as developers, we decided to choose Django as our web framework. One important reason that led us to choosing Django was the database management. Re-creating a brand new database for CPCEU is our frontline objective, and knowing that Django has built in database management is a huge plus. The built-in security for Django is unrivaled by any of the other frameworks that we researched with similar security options. Since Django is a fully featured framework, it will provide us with rapid and secure development. That means Django will already provide us with database security and an admin interface. This will solve two important requirements for our product. Django is universal and does not depend on the operating system or numerous dependencies to run.

One alternative to Django is Ruby on Rails. One of the most popular web frameworks since its initial release in 2005, Rails was a hot contender for our framework choice. The model-view-controller software pattern supported by Django was a huge selling point to us. It was appealing because it makes use of our skills and techniques that have been fostered by our education as object oriented programmers. The Ruby language is the main reason why we did not choose Rails. None of us have

ever developed in Ruby which is a big deterrent for not using Rails. We came to the conclusion after researching Flask and Node.js that there would be a large learning curve if we didn't go for a full stack framework.

Configuration & Dependencies

As an added bonus for Django, the configuration for the developer is mostly done on initial install, but it still provides the end user with ample amounts of customizable apps and admin interfaces. Configuration and dependencies will present a lot of challenges and needs for our product. If our product de-couples with other important tech halfway through development it could be detrimental to the process. One challenge that will give us trouble will be to make sure that the product is supported across every web browser. Django handles this with ease due to its extensive documentation and large community support, another huge reason why we chose Django.

Other Considerations

Here is a list of other detailed benefits of using Django as our web framework:

- Group familiarity
 - Everyone on the team has developed in Django previously
- Community-respected built-in security
- Admin interface
 - A snippet of what the built-in interface looks like:



- Could easily train Todd and employees interact with this.
- Third party plugin support
- Simple routing
 - Two main files do all the URL routing: urls.py & views.py
- Jinja templating
 - Using Python directly on HTML for data management/display
- Simplicity of adding a sub applications and modules to the project
 - Modular apps provide quick and easy modifications to our project.

- Python
 - Doesn't take an expert to learn and maintain.

Conclusion

Since Django offers us security, database management, good LTS, easy scalability, and a natural development process right out of the box, it is an obvious choice for our web framework. That being said, there could have been a move towards using Ruby on Rails, but none of us know Ruby and there is not a big community like Django has that we can rely on for assistance. Because of its success and overwhelming community support, there is no doubt that Django will be an invaluable tool for our project.

Database

The data of the CPCEU, totaling no more than 1 GB over the past twenty years, consists of various pieces of contractual paperwork and facts about the projects like the budget, project lead, and associated organizations. While the nature of the data doesn't really pose an issue for many modern databases, how the data is organized, accessed, and queried will determine the success of this project. Thus any solution must be ACID compliant, secure, good performance, have community support, and is portable.

About Databases

When it comes to choosing a database, first, we must explore whether we are going to choose between a relational database and a non-relational database. The difference between the two is that relational databases are a reliable technology with decades of development, whereas non-relational databases are a relatively new technology to handle problems with querying large datasets of partially validated data.

Relational NoSQL

	Relational	NoSQL
ACID	✓	✗
Security	✓	✗
Performance	✓	✓



As we can see from the above chart, non-relational databases (NoSQL) are not ACID compliant by their very nature, as they solve problems of storing and querying large sets of data that is more or less formatted similarly. Since non-relational databases are relatively new to the database scene, they

are also not as secure as relational databases and a lot of security must be implemented by the developer. Lastly, for the problems relational databases and non-relational databases solve, they both have similar performance, and both suffer with performance when solving the other’s problems. Thus, for this project, we will be using a relational database.

Relational Databases

There are many relational databases that exist, but we will focus on the four databases with the largest communities surrounding them. With Oracle and Microsoft SQL databases, although they can be used for free, there is a whole list of restrictions that applies to their databases that are unlocked by paying for support, and considering that we lack a budget for this project, both are unappealing because of this.

This leaves MySQL with a InnoDB storage engine and PostgreSQL to choose from, both being very popular and widely supported databases.

	 PostgreSQL	 MySQL
ACID	✓	✓
Security	✓	✗
Performance	✓	✗
Support	✓	✓
Portability	✓	✓

As the chart above shows, both PostgreSQL and MySQL are pretty similar, but the major difference between the two lies in security and performance. PostgreSQL has a lot more security features built into the database, built from the ground up will SSL support, and has far more options with its role-based approach to set and maintain permissions. MySQL has an Access Control List based approach to permission handling and some SSL support. PostgreSQL also handles large loads and complex queries better than MySQL, which tends to underperform when under stress.

Conclusion

Normally we would recommend MySQL with a website project like this as the data transactions are going to be relatively simple with no high performance needed. However, the system needs to be designed so that it can last for a decade and scale with the organization in mind. Not only this, but the security is crucial with this database as there are reports that cannot be leaked out the

public that may expose archaeological sites to looting and the locations of endangered species. With this in mind, we are going to use a PostgreSQL database for this project.

Front-end Libraries

Using front-end libraries can save development time and allow for a unified theming and easier programming. We have looked at the following libraries for JavaScript and Cascading Style Sheets (CSS) and have determined which ones we want to continue prototyping. There are two main areas for this investigation: ways to simplify programming the front-end and ways of creating the global site theme with reactive, easy to use interfaces with modern design principles such as material design. The last subsection looks at the two primary ways of delivering our libraries: content delivery networks versus local copies.

Simplifying Programming

For our primary front-end languages, JavaScript and Cascading Style Sheets, we would like to speed up development by using libraries and preprocessors. It is common to practice to use these approaches so that programming time is not spent reinventing the wheel or creating redundant code. The two JavaScript libraries we are looking at are jQuery and Dojo Toolkit, both mature and competitive solutions and offer about the same suite of functionality and benefits. The two main CSS preprocessors we have found are Syntactically Awesome Stylesheets (Sass) and LESS.

jQuery - JavaScript library

For JavaScript development, we selected jQuery, a library designed to speed up development and decrease time spent writing code. It is one of the most popular libraries and is usually a requirement for other JavaScript libraries.

Based on preliminary research and the team's previous experiences, jQuery will save us time not having to write the same helper methods, dramatically increasing our potential work output. For usability, it is a singular, minified file and is compatible with the major suite of desktop and mobile browsers including Chrome, Microsoft Edge and Internet Explorer, Firefox, Safari, and stock browsers. Lastly, jQuery guarantees support for all browsers' current and previous stable versions and also includes support for older browsers with jQuery 1.12.

The other JavaScript library we considered to reduce time spent writing code is the open source Dojo Toolkit. Though it is popular with companies like IBM and Pearson and has similar capabilities as jQuery, we could not justify using this library. Most of our theming and user interface libraries require jQuery and ultimately Dojo was not needed. It is one less library we need to learn and keep track of.

Syntactically Awesome Style Sheets - CSS Preprocessor

While CSS is extremely great in itself, it is not very development friendly. Manual CSS requires a lot of typing, separate inheritance blocks to get working, and is usually compiled into a massive single file to limit web requests. To help decrease overhead and automate compiling and minifying CSS for production, we have elected to use Syntactically Awesome Stylesheets (Sass), a CSS Preprocessor.

As an intermediate, it has better syntax to decrease the amount of writing and also includes the added benefits of nested rules, variables, functions, and the ability to make modular, flexible code. By using Sass, we expect to save a lot of time since we will have a tiered file system for styling that automatically compiles as we write and is organized without the need to keep all of our styling in a massive, single CSS file. Sass requires a build tool (like Prepros for example), however, this impact will be miniscule when compared to writing plain CSS. Sass's lifespan is directly linked to any updates for CSS, which means it is supported by all browsers that support the most recent stable version of CSS. Lastly, Sass is an open source project with frequent releases to add features and optimize the codebase.

LESS is another CSS preprocessor and is inspired by Sass. They are both very capable of what we want them to do and are almost the same except that LESS lacks in a couple areas. Sass allows us to not repeat ourselves as often with our nesting, which plays a major part of writing less code. Also LESS is not as accurate with math operations and converting between CSS units. Lastly, Sass is more distinct when using mixins by separating selectors (classes, ID's, and tags) from mixins (like an include without copying and pasting code) whereas LESS combines the two, which could be a pitfall.

Theming and User Interface (UI) Libraries

Both our clients and us developers want to make sure that the end-user experience is harmonious throughout the web application without relying completely on custom styling. If we were to implement our own solution, it would be difficult to create and maintain. By using already popular solutions, we can cut down on development time while assuring an acceptable level of support after we deliver the solution. To accomplish this, we looked at various technologies: three front-end frameworks (MDBootstrap, Bootstrap, and Materialize) and a couple of reactive JavaScript libraries (jQuery UI and Vue.js).

Material Design for Bootstrap (MDBootstrap) - UI Toolkit

A spin off of Bootstrap 4, MDBootstrap combines both modern, reactive design with simplified CSS and JavaScript without using default theming. With the popularity of jQuery and Bootstrap, MDBootstrap aims to deliver a great foundation for our product's UI with numerous benefits.

The foremost benefit is a clean looking UI with minimal effort. The modular theme can be changed and already comes with built-in cross-platform compatibility with frequent updates almost every month. We can use this to our advantage to spent less time working on the nitty-gritty CSS and

more time developing the solution for our clients. Lastly, MDBootstrap also comes with jQuery and Sass files, which we already want to use to decrease programming time and increase functionality.

The other theming libraries we considered were Bootstrap (the original) and Materialize CSS, a younger, still growing library with material design in mind. Bootstrap is a great boilerplate, but we did not want to make the theme from almost scratch. We also wanted something that came with material design at the forefront. Plus, with MDBootstrap, we have more capabilities and less time developing our own solution or having to rely on multiple other plugins. As for Materialize, it is still new and lacks functionality. Lastly, we are more familiar with Bootstrap than Materialize.

We also considered mixing Bootstrap or Materialize with a UI library like jQuery UI or Vue.js. jQuery UI is a collection of widgets, themes, and animations for web interfaces. Overall, jQuery UI is not as feature rich as MDBootstrap and could be considered another dependency with little justification. It can wrap UI elements and create theming, but we could not find a need for jQuery UI's niche.

Vue.js is a progressive JavaScript framework for building reactive user interfaces. Presently, this is something we want to still consider when designing and building the website, but does not need to be chosen at this time. It is extraneous to this document. This is also the same for React Native and AngularJS. While both great technologies, we discovered that both of these are not a part of our solution to our client at present. React Native focuses on mobile application development and AngularJS is a web framework.

Overall, MDBootstrap has great potential with minimal reliance on other software while providing enormous functionality for both the end user and us developers.

Content Delivery Networks (CDNs) vs. Local Copies

This is an overall challenge that affects all of our development. We need to choose between using content delivery networks or storing local copies on our web server to serve our front-end libraries. Below are our conclusions on both ideas:

Content Delivery Networks

By using a CDN, we do not have to keep track of the library and it can be updated automatically without human intervention. The maintenance of the library will be minimal and will always have the most recent version. However, there are a few drawbacks of this method.

The first and foremost issue is the reliance on another system that we have no control over. This means that if the CDN changes URLs or methodology of serving content, then we would be unable to assure support for our product after we deliver it next year. We could render our product completely inoperable if there is a problem with the CDN.

Local Copies

The first benefit to using local copies is version control. We have the ability to choose which version of our libraries we want to serve and it can give us time to experiment with new releases

before actually releasing them to our master version of the product. With local copies, we can also edit the libraries directly or interact with the files without relying on a CDN. Lastly, since we have a local copy to serve, we have control over how they are served, assuring that the libraries will be served server side in a predictable manner.

Some drawbacks include manual updating. Depending on how we edit or use these libraries, we might have to find ways to merge our customized code into the new version. Other problems include having a fixed version of libraries. This could be a security risk potentially, though we haven't found a way to justify this with our library picks, and could have the more serious problem of becoming old and deprecated. With progressive updating, we could mitigate these issues, but once the product is out of our hands, we cannot guarantee long-term support.

Conclusion

We want to use local copies. We want the ability to control what library versions we use and be able to modify and work directly with our toolkit and libraries in a predictable manner. Lastly, with a local copy, we don't have to worry about or rely on external resources.

Documentation Platform

Documentation is critical for modifying and maintaining, not just the application front-end, but also the back-end. Our client will need to change their workflow and, sometime in the future, will update our technology so we need to have documentation that is accessible, easy to navigate and read, and is user friendly regardless of technical knowledge. Below is a summary of our decision.

Documentation Technologies

The main solutions we have looked at are Confluence, version-control built-in wiki, Google Docs, and building our own solution.

Confluence by Atlassian is a self or cloud hosted, collaborative documentation platform. It allows teams to easily and effectively create and manage project documentation. One of the main reasons we did not choose this solution, despite its features and capabilities, is that we cannot guarantee exportability and modification in the future. We can export all of our documents as Word, PDF, and HTML, however, this does not allow our client and future developers to edit the documents nor does it make it accessible to our user groups. Lastly, Confluence is paid and though it is minimal, we do not want to pay for a system that we do not have full confidence in.

We also thought about using our Version Control software's built-in documentation software. This quickly became problematic, because it varies between web services and, for websites like BitBucket, we cannot easily export our documentation for our client. We would be saving more time and having less problems if we focused more on our in-app documentation while also satisfying more

requirements from our client. Lastly, if we kept the documentation as a part of the repository, then we cannot guarantee that it will always be accessible.

Another idea was to use Google Docs to write the documents initially then export all of them when we were ready to deliver our solution. This was quickly dismissed, because there were a lot of issues including finding ways to search for material and making things overall simple to add and change and read. With Google Docs, there is no real way of linking documents or making it searchable before and after we export our documentation.

Built-in Documentation - Our Pick

Based on our evaluation, we discovered that the best method is building our own documentation / wiki inside of our solution. The primary reason for this is that the documentation needs to be accessible by numerous user groups with varying permission levels. This approach will depend on the framework we use, but we could build this documentation application as a part of our prototype for this semester.

By creating our own solution, we can contribute our findings to the overall project. This includes URL routing and learning how we want to use our database. This can also help us learn how we want to do user permissions, create a uniform user experience, and practice implementing our clients forms and workflow. Since this solution is built into the application, it is extremely portable. We do need to consider making it easy to find, read, and add/edit material, but we do not believe this is a huge problem.

Hosting Solution

Solving the issues and challenges that come with deploying on to a host is one of our biggest obstacles. The biggest reason is that the host has to be able to serve our product to the end user by utilizing our tech. This can be extremely difficult to solve when the host system has technology requirements and restrictions. Detailed below is a plan of how we will solve these challenges.

- Development
 - We're going to use Joseph Remy Jr.'s self hosted server. During the development process, this will ensure that we will get a working product before we try to deploy.
- Testing
 - For the testing phase, we will test deployments on different OS's, browsers, and environments using Amazon Web Services (free-tier instances).
- Final Deployment
 - We're going to fully deploy our product with ITS. This may present some challenges because of restrictions so we are making sure to dedicate a lot of time to solving this.

Possible solutions for the final product:

- NAU ITS
 - NAU's ITS would be less ideal due to a lot of tech and system requirements, which are always changing and extremely restrictive which would hinder the deployment process.
 - That being said, since NAU is hosting the CPCESU, they should be responsible for the providing an environment for our product.
- Amazon Web Services (AWS)
 - Amazon Web Services would be an expensive solution, but robust enough to handle all of our dependencies and configurations.
 - Either the CPCESU or NAU will have to pay for these services including maintenance. If our product receives a lot of traffic, then it will become increasingly expensive.
- Hosting the server ourselves
 - Self hosting would make deployment easier because the lack of tech requirements, but would present the CPCESU with the obligation of maintaining a foreign server and a hand-off of the solution in this way is not feasible.

Version Control

To keep track of our contributions and allow for collaboration on this solution, we need to use version control software. We have a preference that we don't want to host our own Git server, as this would be more overhead. Other services can do a better job, so we looked at the following websites.

Version Control Services

These are the technologies in brief we looked at for Version Control:

- Bitbucket
- GitLab
- GitHub

All of these technologies meet the minimum criteria such as accessibility and access to our team repositories and customization options, but the main deciding factors were mostly based on integration capabilities and licensing.

Bitbucket - Our Pick

Overall, Bitbucket by Atlassian offers a comprehensive suite of version control features. These include simple user management, pull request and code review capabilities, integrations for our continuous integration and continuous delivery (CI/CD) server and issue tracking software, etc.

We originally were using GitLab since our team was formed. We all got very familiar with it, but ended up not wanting to use it anymore. GitLab offers more features overall than Bitbucket, but required money for most of their services. GitLab could have been our one-stop shop for version

control, issue tracking, and CI/CD, but it is too expensive to justify \$19 per user per month billed annually.

We also looked at GitHub to host our solution, but ultimately forgone this, because we wanted to keep the repository private and wanted to use something enterprise-like. We could have used our school emails for GitHub to get this for free, however, everyone already had GitHub accounts created without the school email addresses. We also wanted to focus on using something that might be seen in the workplace and GitHub did not have the same look and feel.

The access and management of the repository was the same overall of the Version Control software, but the integrations and simple licensing of Bitbucket are what stood out to the team. It is critical that we have our CI/CD pipeline to test code and show our client that we are providing a working, optimized solution. This means that our service needs to have a webhook to our CI/CD. It is also important to have an integration to our issue tracking software so that we can track our project status and statistics alongside our requirements. In "Issue Tracking," we picked Jira, which is in the same software family as Bitbucket so the integration is straight forward. Lastly, Bitbucket had a straightforward license - free perpetually for up to 5 users with no restrictions of features.

Testing Solutions

As with any project, testing is crucial to ensure that the software does what we intend it to do, ensure that future changes do not change the behavior of the software, and to uphold compliance to various requirements. Due to the web-based nature of this project, there are various other tests we need to use to verify we are delivering a high quality product that works for different users in various environments than it is presumed to run on.

Unit Tests Framework

For unit tests, since we are using the Django web framework and thus Python, we will use the robust and modern Python Unit Test framework. It has all of the features one expects out of a testing framework from simple asserts to mock objects. There no third party additions needed to expand the libraries functionality.

Compatibility Testing

For compatibility testing, there are a variety of solutions that are immediately not viable, because they are paid services in some form. This leaves either using Selenium, an API that does automated compatibility testing, or a third party tool like Katalon that is built on top of Selenium.

Selenium is a long running open source automated testing tool that is highly configurable to test any browser and is the backbone of all other automated testing solutions. Selenium can also be configured to work in the Python unit tests, so the codebase remains unified under one language.

However, the biggest downside to Selenium is that it has a big learning curve and is really painful to get setup and running.

Katalon, on the other hand, provides an IDE and some additional functionality to Selenium to make it more approachable, easier to set up, and easier to use. The downside to this is that splits apart the development environment, as all of the compatibility testing has to be through the IDE. With this in mind, we are going to be using Selenium due to its integration with Python.

Accessibility Testing

For accessibility compliance, when we exclude the variety of paid services, the top and most used API in the industry then seems to be Paypal's Automated Accessibility Testing Tool (AATT). This is an API that inspects HTML for meeting accessibility standards.

Linting

Lastly, for linting (looking for potential errors) our codebase, we will use JSLint and CSLint so that our Javascript and CSS code is robust and avoids the common mistakes.

Security Testing

Ensuring we have top of the line security for our web application is crucial for maintaining the reputation of the CPCEU and to elevate the safety of the archaeological sites and endangered species the CPCEU has information on.

While it is possible to have an automated security test suite that wraps into a CI/CD solution, the issue with these services is that they only test information gathering and target evaluation steps of a security audit, and are unable to provide information regarding the depth and ease of an attack. In addition to this, many of these automated services, along with using any security auditing company, have significant prices associated with them.

That being said, Kali Linux is a free and open source tool that is commonly used by the majority of professional security auditors. Kali Linux is a Linux distro that comes preloaded with a wide variety of tools to test the security of any computer application, including web applications. This will allow us to perform our own security audits on our website, given that we are willing to learn the various tools needed to do so. This will also allow us to identify how difficult it is to perform a particular exploit and how damaging it can be, so we may better assess whether the cost to improve that system is necessary or not.

While there are a plethora of tools available in Kali Linux, listed below are a sampling of tools we will definitely need to learn to perform our own security audits.

- Information Gathering
 - Httrack – Clones a website to allow for offline analysis
 - Webshag – Provides port scanning, URL scanning, and file fuzzing
 - FOCA – Shows the metadata attached to documents from an organization
- Target Evaluation
 - Skipfish – Prepares an interactive sitemap and runs security checks on it
 - ProxyStrike – Used to identify SQL, SSL, and XSS vulnerabilities
 - Vega – Used to identify vulnerabilities with forms
- Exploitation / Privilege Escalation / Maintaining Foothold
 - Metasploit – Used to perform server exploits, attaining privilege escalation, and log cleaning to hide one’s tracks
 - w3af – Uses found exploits to attempt to get a shell on the target system
 - DirBuster – Used to find hidden applications and pages in a website

Continuous Integration and Continuous Delivery (CI/CD)

Continuous Integration and Continuous Delivery is necessary to ensure that the product is building throughout the project and to have a constantly updated development website with all of the latest features of the website.

When it comes to the various CI/CD solutions available, any hosted solutions are out of the question as they normally have a high cost associated with them. Of the remaining CI/CD solutions, the top three are Jenkins, TeamCity, and GoCD.

	 Jenkins	 TC	 go
Agents	Unlimited	3 Build Agents	50 Build Agents
Support	✓	—	✗
Configurable	✓	✓	✗
Flexible	✓	✓	✓
Good UI	✗	✓	✓

Go is a relatively recent open source project from 2014 that has a lot of potential behind it, being a new way to visualize CI/CD and seeks to be a full CI/CD solution out of the box. The issue with this tool is that it isn't widely adopted yet and lacks resources, tutorials, and support for how to set up our project, and doesn't have a lot of plugins to modify its usage for our different tools.

TeamCity is a proprietary full CI/CD solution out of the box that can be setup quickly, but comes with a limited number of build agents in its free edition. A build agent is a process that builds the entire or a part of the project. When utilized correctly, it can speed up a slow build process or parallelize multiple builds if development is fast and there are multiple pushes at a time. Even though this is a limitation, three build agents will probably be enough for this project, as we are a team of three and we probably won't be pushing to Git enough for it to be a problem. While the community is large and there is plenty of documentation, setting up TeamCity for our project is particularly weird, as it is unable to communicate directly with the tests in Django, requiring the Python Test Runner to be overridden by a third party one to communicate with the TeamCity server. In addition to this annoyance, TeamCity requires two separate servers to function correctly, and setting that up looks like it's going to be not trivial if we decide to host the CI/CD on our hardware and not on AWS.

Last is Jenkins, the oldest open source CI solution of the three. It has a large community and set of plugins to modify its behavior in anyway we need. In addition to this, with the release of Jenkins2, its learning curve has been softened by coming prepackaged with a bunch of commonly used plugins, allowing it to be set up quickly. The only issue with Jenkins is that it can be finicky to set up, as with any CI/CD solution, but this is offset by the large community support. In addition, Jenkins is widely used in industry as a CI/CD solution still, which shows that it is very robust and battle-tested CI/CD solution.

With this in mind, we will be using Jenkins as a CI/CD solution for this project, as its only downside of having a very functional user interface is a low priority compared to having the resources to set it up in a reasonable amount of time.

Issue Tracking




Issue tracking is a major part of our project management. From creating issues from requirements to tracking project statuses and statistics, we want a comprehensive tool that can integrate with all aspects of the project and allow us to practice agile software development.

Issue Tracking Solutions

These are the issue tracking technologies in brief we looked at:

- GitLab
- Jira
- Trello

These issue tracking softwares actually varied dramatically across the board. Below is a figure comparing the main metrics of our evaluation.

			
Cost	\$10, 5 users Billed one time	\$19 per user Billed Annually	Free No Fees
Integration	✓	—	✗
Scrum	✓	✓	✗

Jira - Our Pick

Jira is a feature-rich issue tracking software by Atlassian. Overall, this service comes with agile software development built-in, extensive customizability, and numerous integration capabilities with a simple license.

We started off using GitLab at the beginning of the semester, but have moved to Jira, because the issue board did not end up working. We could not group issues effectively, it did not allow for sprints or agile software development, and did not have native integrations for our version control software or CI/CD pipeline. Our version control software is from the same developer, which made the integration seamless. Bitbucket had a native CI/CD plugin for Bitbucket repositories, but GitLab had a community-supported plugin from GitHub. Lastly, like most of our other problems with choosing GitLab in this document, there are paywalls and fees to use most of their features.

We also considered Trello at one point, but this was quickly dismissed. We could create issues and track them, but we could not organize issues enough into epics or like groups. Trello also does not have any agile software development plugins or capabilities and does not integrate well with our CI/CD and version control services. For being a free website, it is great, but lacks features to prove its worth.

Lastly, Jira and its competition can all create, track, and archive issues, however, Jira allows us to group issues together in epics, create sprints, manage our own workflow, and more. While looking at issues, we can also see the relevant branches, commits, and build status from our version control

software and Continuous Integration and Continuous Delivery (CI/CD) server. Lastly, Atlassian offers a perpetual license for ten users if you self-host Jira. Overall, Jira offers enough features to justify the one-time \$10 cost and did not take long to set up

Environment System

Since we want to control our software environments and manage our dependencies regardless of platform, we need to find a solution that can accomplish this easily without too much set up and maintenance.

Environment Software

These are the issue tracking technologies in brief we looked at:

- Dedicated machine
- Docker
- LXD
- Self-hosted virtualization (Hyper-V / Virtual Box)
- virtualenv

These issue tracking softwares actually varied dramatically across the board. Below is a figure comparing the main metrics of our evaluation.

Docker - Our Pick

Even on our small scale, we found that Docker containers would be the best way to control applications and make our environments portable and efficient to run regardless of operating system. Docker containers can standalone, for free, and can be moved inside of repositories since they are small. The biggest factor was taking the stress and time out of creating development environments and removing the possibility of mixed dependencies or misconfigurations. It is also used by companies like PayPal, Visa, and ADP, has great support, and a large community. Lastly, Docker containers can be used in numerous ways including in self-hosted and cloud environments.

virtualenv is used to create isolated Python environments. That being said, it has limited capabilities for isolating dependencies. It is not as fully featured as a Docker container, but can be harder to deploy since we don't know a lot about the tool. Overall, though both are free, Docker is simpler to set up from the get-go and has more support than virtualenv.

LXD is a way of containerizing applications just like Docker, but for Linux only. This would require everyone to use Linux and does not make sense nor do we have the resources to effectively use this. Plus, Docker runs on most systems, regardless of operating system and architecture.

Self-hosted Virtualization was one of our options, but ultimately fell through because it did not make sense for our project. We are not all working in the same location, therefore having a centralized virtual server for our development environments does not make sense.

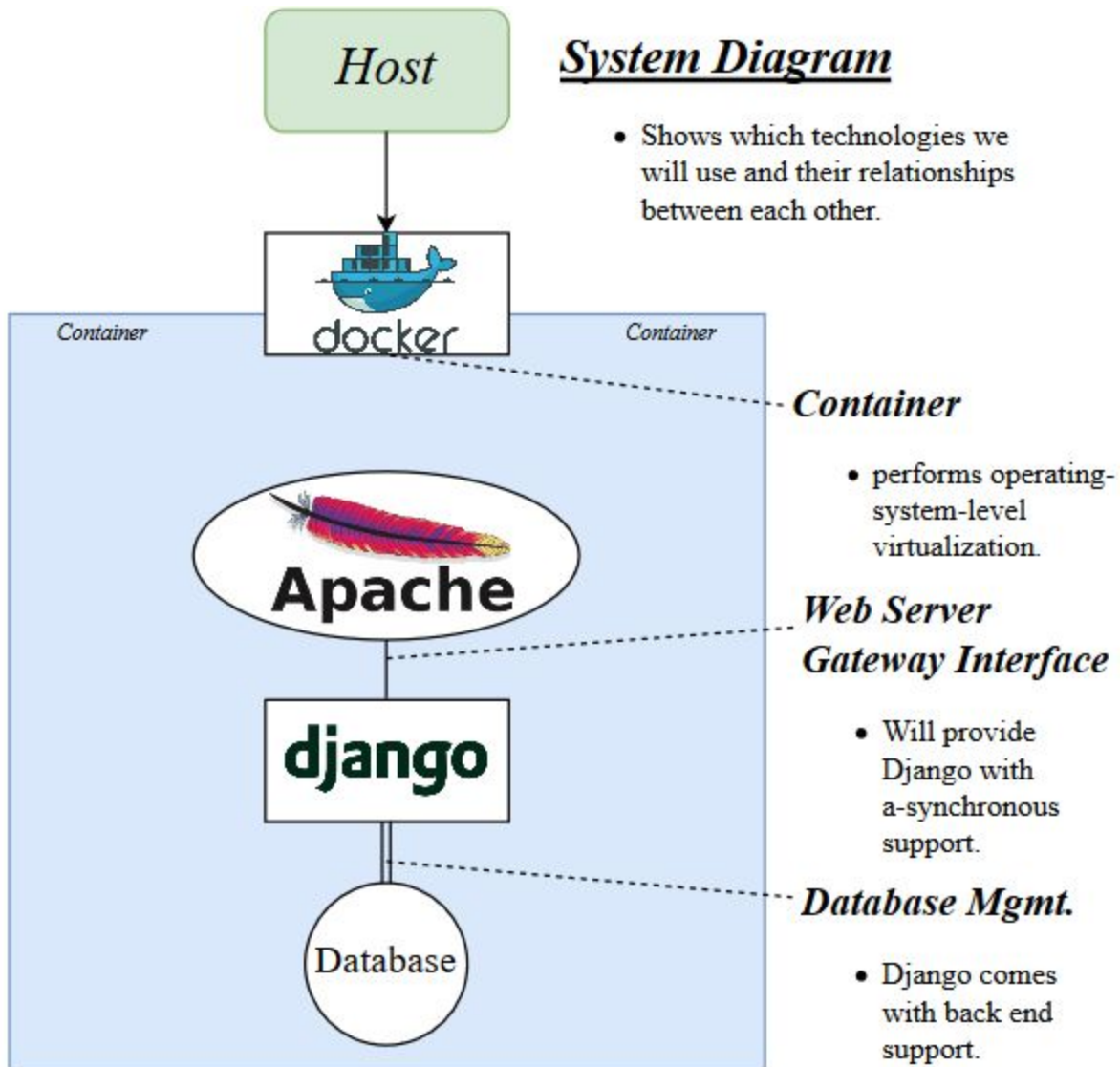
Lastly, we also had the simple idea of a dedicated machine. This would only work, again, if we all were in the same location developing together. This development machine would not really work over the Internet and we quickly retired the idea. This would work for a testing server for our clients to access and view our solution as is, but not for programming on.

Because of its features and how it compares to our other evaluated solutions, we have selected to use Docker as a part of our solution.

Technological Integration

It is not only important to find what technologies we are going to use, but also know the challenges of integrating these technologies together. Below are our considerations.

Figure 2: System Diagram



Brief of Technology Integrations:

- Problem - Hosting - Hosting software on the server
 - Solution - Host To Docker
 - Host will have to 'spin up' our Docker container on their respective system.
- Problem - Configuration - Managing application and dependency versions

- Solution - Docker Container
 - This will house all of our libraries, software, and other dependencies into one container providing system stability and fast integration into any host.
- Problem - Deployment to multiple users - Taking in numerous web requests simultaneous
 - Solution - Apache to Django
 - Apache will provide the Web Server Gateway Interface which grants our product to be delivered to multiple requests from end users asynchronously.
- Problem - Database - Communication between web app and database backend
 - Django Database Management
 - Already being built into Django, this is a very simple and well documented process.

Conclusion

Our problem we are solving in the project management and data issues of the Colorado Plateau Cooperative Ecosystems Studies Unit (CPCESU). Though they presently have a workflow, it is not efficient and more time is spent managing projects than actually working on them. We, both ECODers and CPCESU, envision a fully featured web application with the capability to increase productivity, decrease paper trails, and create a centralized database for querying and has user input verification and unified input forms.

This document outlined the technological and integrational challenges of our solution. The primary technologies were the web framework, the database, and the front-end libraries. Below in the table, we have summarized our challenges, technologies, and our confidence level with them.

Challenge	Potential Solution(s)	Confidence Score (1 to 5) 1=low confidence; 5=high confidence
Web Framework	Django	5
Database	PostgreSQL	5
Front-end Libraries	jQuery, Sass, MDBootstrap	5
Documentation Platform	In-app documentation	4
Hosting Solution	Development - self-hosted Production - ITS	3
Version Control	BitBucket	5
Testing Solutions	Python Unit Test Framework, Selenium, Automated Accessibility Testing Tool, JSLint, CSLint	4
Security Testing	Kali Linux	3
Integration	Jenkins	4
Issue Tracking	Jira	5
Environment System	Docker	4

Overall, we believe that our technological and integration challenges have been addressed, which assures us that our solution is feasible. As with most software projects, we will continue to be diligent if issues arise and will adapt as a team. Presently, our only long-standing issue is working

with NAU Information Technology Services's (ITS) requirements, which we are still not familiar with, but have started the conversation. To get this answered, we found a time to sit down with our Point of Contact at ITS and hash out both sides' requirements.

With our selected technologies, we have already begun creating a prototype application and verify our selections. Ultimately, based on our decisions, we will development a project management system that will aid the management of conservation projects in the American Southwest. These projects are vital to our understanding of our ecosystems, how we are affecting them, and finding ways to study and conserve them. Lastly, if this solution for CPCEU is extremely successful, it may become the national system for the Cooperative Ecosystem Study Units (CESU).