NAU NORTHERN ARIZONA UNIVERSITY

School of Informatics, Computing, and Cyber Systems

Technical Feasibility

November 10, 2022

Team Teacher To-Do

Project Sponsor: Chris Aungst

Faculty Mentor: Italo Santos

Faculty Mentor: Michael Leverington

Team Members:

Sam Gerstner (Team Lead)

Alexander Frenette

Noah Nannen

Shlok Sheth

Bronwyn Wedig

TABLE OF CONTENTS

Project Introduction

Arizona is currently facing a massive teacher shortage in our state. To help address this issue, the Arizona Department of Education has introduced a new program that will allow student teachers in their final semester of their undergraduate degree to fill some of these vacant teaching positions. As you can imagine, this process involves tracking many documents and requirements to ensure students meet the needed criteria. Currently the NAU College of Education spends hundreds of hours each semester tracking these requirements for their students in order for them to participate in this student-teacher program.

The current solution involves a great deal of data being input into an excel spreadsheet; this process is currently a manual one. Hence the tremendous amount of time being spent. The spreadsheet is used to track completed courses, along with other completed requirements. Some of these requirements need to be confirmed by students, yet they are unable to directly report and modify the spreadsheet. This requires them to report to an individual, who has access to the spreadsheet, which can then update said information. This interaction introduces additional work, and delays.

To streamline the process of tracking and confirming eligibility for students to participate in the student-teacher program, we will be designing a web-based application that will be able to automatically interact with student records eliminating the need to manually import data, as well as provide the functionality to self-report external information regarding requirement. Our application will allow for both staff and students to view the progress for necessary requirements, as well as save a tremendous amount of time by having a single source of truth for eligibility status.

Document Introduction

The purpose of this technical feasibility document is to outline the high-level project requirements, technologies to be used, as well as the technical feasibility and any foreseeable problems with any project requirement or chosen product or technology. This document aims to provide an in-depth analysis of all MVP requirements and the technologies and platforms that will be used to help us accomplish our goals.

This document will be used as a reference point for the remainder of the project and will be one of the primary documents that guide our development and decision-making processes. One of the primary goals of this document is to outline potential problems, their solutions, and to help all stakeholders gain a better understanding of roadblocks that may be encountered during the development process.

Technological Challenges

Our minimum goal for the project would be to at least give the student their overall progress for fulfilling all the requirements and steps to achieve other requirements. In simpler terms, we could describe the website as a professionally modeled website that imports the student information from their NAU login using CAS. The student can track and follow through their career path for their Student Teacher Intern Certificate (STIC).

STIC has a hefty list of requirements a student must fulfill; hence the website would give the student a gist of that. The model will also represent the requirements that are completed by the student. It has a dashboard where the student can see their overall progress for the program. The website will also provide a feature to share, and upload signed documents on the server. The main objective of the website is to help the student be on the path for the STIC plan and give a user-friendly insight into their overall standing making it easier and less time-consuming for the student to track their progress. The website will create a path for the student and give the student a plan to follow through in order to fulfill the requirements. We also plan to accommodate the website by importing the NES exam scores from the Pearson website which plays a vital role in making the student eligible for the certificate program. For the backend part of the website, we would like to structure it in a way that it will import data from FileMaker Pro software that will use an application program interface, and it will use the student USER ID to fetch and identify the data from the database. As specified by Dr. Leverington, the minimum requirements for our website using sensitive data should be able to create a hashed value for the student user id in order to make the website secure and then search the student's delicate information in the FileMaker Pro database. This process demands a hashing algorithm in the program.

Another focus of ours is to model the website's appearance on existing NAU systems, as well as provide an administrative panel for staff of the College of Education.

While not currently a priority, a future desired ability would be to enable e-signatures to be managed through the app. For now, we will consider this ability outside of the scope of our tasks.

As it stands, our project is somewhat of a greenfield. There are two systems in place that we hope to interact with: these being the collection of student records, as well as the school's user authentication system CAS. Besides these two systems, we are free to choose how to best implement our system. That being said, we are faced with several technological challenges:

- Selecting a robust language to build our system upon
- Selecting a robust framework for delivering web pages
- Obtaining and accessing reliable student records
- Authenticating the identity of students
- Storing auxiliary student requirement data
- Accessing our application data
- Documenting our API
- FERPA Compliance and securing the student data
- Creating a uniform user experience

Technological Analysis

Languages

One of the first challenges we faced with this project when coming up with our development plan was what backend programming language we should use. This is a critical decision, as it is the foundation of our project's development. The languages we considered using included C#, & NodeJS, and Java. While all these languages are capable of being used to create web apps, we must select the best fit for us.

Our selection criteria were based on several factors. We wanted a language we were familiar with or at least capable of learning in a short period of time and is efficient with respect to developer time spent programming. We were also interested in industry adoption, and long-term support. We did not want to be using libraries that were not production tested, as security is critical to our app. Another concern of ours was type safety. A huge source of bugs stems from a lack of type safety; to avoid any potential type mishaps we are very focused on this.

Looking at each language, we start with C#. C# offers a great deal of the features we want. It has some industry support along with extensive documentation, is type safe, and offers production tested libraries. The major downside is that the majority of the group is unfamiliar with it.

NodeJS is a server-side implementation of JavaScript. It is incredibly easy to write simple applications with and is great for those interested in expanding from front-end development to back-end development as well. The issues for our group are that even fewer members are familiar with it, and it does not provide the same robustness and type safety as other languages. As our system grows, it becomes more difficult to manage the code base of a dynamic language and ensure safety.

Java ended up our winner. It is one of the most commonly used languages in industry, has extensive libraries and documentation, the language is static and strict which will ensure safety. The group also has more familiarity with it than with other languages. To prove the feasibility of this language, we have gone ahead and further refreshed ourselves on the language and gained confidence in our ability to use it.

	C#	**Node.js**	**Java**
Developer familiarity	2	1	3
Developer efficiency	3	4	3
Industry adoption	3	2	5
Type safety	5	1	5
Total Score	**13**	**8**	**16**

Web Pages and Frameworks

Our next challenge is delivering the actual web application. As it stands, there are two dominant players in the world of web applications. Dynamic pages, and static web pages. A dynamic webpage is an architecture that involves the client sending the server a “skeleton” webpage, which is then populated on the client side though calls to a backend API. Two of the major benefits are reducing load times of pages, and minimizing data transmitted. A static webpage is an architecture involving the generation of the webpage each time upon request. While the page itself will have data that is dependent on which user is logged in, hence dynamic, once it reaches the user’s web browser, it is in its final state, hence the term static. As a group we were able to immediately eliminate the option of developing a dynamic website. This being we have limited familiarity with the technologies involved such as React.js and Vue.js.

Having made this decision, we began our search for a java framework for making webpages. Specifically, a framework designed around the MVC architecture. MVC allows for the separation of our model (data), view (webpage) and our controller (application logic). This separation allows for easy division of development tasks among our team members based on individual's strengths. The frameworks we investigated using were Spring Boot, Quarkus, and Micronaut.

While Quarkus will provide the MVC functionality that is desired, it relies on Kubernetes to function. This added complexity is something that is beyond our current resources, making it a non-viable option simply due to the ease of deployment. Spring Boot is the most popular option amongst Java developers and provides extensive documentation and tutorials for new users. Because of these clear advantages, we chose to use Spring Boot for this project.

Micronaut is a more traditional framework. It does not provide the same horizontal scalability out of the box, but it provides relative simplicity. It too is currently used in production environments and offers extremely thorough documentation. The major drawback is the complexity of the framework and lack of digestible tutorials.

Our choice was ultimately Spring Boot, as it offers a relatively streamlined development and deployment process. The Spring Boot framework will allow us to write our application in languages that are familiar to us (Java, HTML, CSS, etc.) and easily package our application into a JAR or WAR file that can easily be deployed to an Apache Tomcat web server. The documentation is excellent and offers simple and digestible tutorials to get you started. An added benefit of spring is the modules that have been developed for the framework. These modules offer features such as database connections, security, and authentication. We believe therefore Spring Boot's adoption in the industry is so high.

To prove feasibility of using Spring Boot, we will go ahead and implement the demo apps that are provided in the extensive tutorials. We will then expand upon them to further develop our app.

	Quarkus	**Micronaut**	**Spring Boot**
Documentation	2	2	5
Simplicity	2	2	4
Industry adoption	2	2	5
Deployment	1	4	5
Total Score	**7**	**10**	**19**

Obtaining Reliable Student Records

Our next challenge is to obtain reliable student records for our application. The foundation of our app is a task list, and these tasks depend on the state of the academic record of a student. Due to FERPA privacy concerns, this data is highly regulated. In order to gain access to this data our group first had to complete FERPA certification through the university. While we have been told we will have access to the data, it has not yet happened. Due to this, we must take several things into consideration. There are three scenarios we are currently faced with. First, we get access to the FileMaker Pro API, which is the system the College of Education uses to keep track of student records. This API will allow us to query sensitive records of students, to then determine eligibility, and status. The second option is that we are not given access to this API, and instead we must find a way to export this data to a file such as a CSV, that the College of Education can then import into our system. This has major drawbacks or adds additional exposure opportunities if our database is compromised. The attacker would then have access to the student record data. The final option is that we create a mock API that will generate fake student record data and treat it as though it were FileMaker Pro. This has the major downside of making the app non-functional.

Ultimately our decision is to assume we get access to FileMaker Pro's API that way we can create a functional application. It also means that administrators at the College of Education do not need to manually administrate our application, while also limiting the amount of sensitive data stored in our database.

Once we gain access to the FileMaker Pro API we will begin proving feasibility by creating simple calls to the API to confirm our ability to interact with it.

	FileMaker Pro API	**Importing Data**	**Mock API**
Admin simplicity	5	1	4
Student record security	4	1	5
Usefulness of application	5	5	0
Total Score	**14**	**7**	**9**

Authenticating Student

As our system will need to interact with student permanent records, authentication is critical and mandatory per FERPA. The ability to uniquely identify students and verify such identity will be critical in preventing individuals from accessing data that is not their own. We have several options to handle this. We could gain access to the university's authentication services, authenticate users based off student e-mails alone, or have administrators create accounts for each student.

Our main criterion for our decision is security, ease of account creation, and ease of administrator management.

If we can use the university's CAS (Central authentication service) we are guaranteed that a user is who they say they are. We also get the benefit that we do not have to manage user passwords or accounts, which is handled by CAS. It also means that the administrators at the college of education do not have to manage accounts.

If we verify users based off e-mails, we gain the same benefit of knowing they are who they say they are, but we have the downside of an additional piece of data we must protect and manage. That being their account password. On the plus side, administrators do not need to be involved in the account creation process.

Our final option is to have administrators create accounts. This is in no way ideal as it is not an automated process, and would require additional person hours being spent, somewhat defeating the purpose of our application.

As a result of our requirements, we have decided to plan on using the CAS system but will resort to student e-mail account creation if we are unable to gain access to CAS.

Our method of proving the feasibility of CAS we will have to begin by either requesting a “dummy” user account be made to test our ability to log in and authenticate a student. If this fails, we will then have to explore our secondary option.

	CAS	**Student e-mail**	**Admin created**
Admin simplicity	5	5	0
Account Security	5	3	3
Ease of account creation	5	3	0
Total Score	**15**	**11**	**3**

Storing Auxiliary Student Data

While we will not be storing student records, we will still have a need to store student information specific to our application. This includes status on tasks as well as specific documents. This leads us to two main options for databases. A simple relational database or a document store.

For simple relational databases, we have the option of MySQL or MariaDB. While for document storing, we would be using MongoDB. Our main criterion is familiarity, speed of development, feature set, and cost.

Both MySQL and MariaDB can be thought of as equivalent for our sake. They are both SQL databases, which means they are very familiar. The only downside is their feature set. Neither database is good for storing large pieces of data such as files. As such we will need to use an external blob store and reference the blob IDS within the database. As a note, the only difference between MySQL and MariaDB is that MySQL is propriety and has an additional cost.

If we were to select MongoDB we would lose familiarity with the database technology but gain the benefit of having dynamic schemas. This feature might be useful for some data sets, but as we are going to define our schemas, we do not benefit from this. We would however benefit from the ability to store large documents from within the database using GridFS.

Ultimately due to familiarity, we have chosen to go with MariaDB, which if needed can be changed in production to MySQL without issue. We shall prove feasibility by creating a sample database, and create a simple Java application to interact with said database.

	MySQL	**MariaDB**	**MongoDB**
Developer familiarity	4	4	2
Development speed	4	4	1
Feature set	4	4	5
Cost	3	5	5
Total Score	**15**	**17**	**13**

Having selected a relational database, there remains the need for document storage. Our 3 options are to use a traditional cloud storage provider such as Google Drive or Dropbox, self-hosting a filesystem on a VPS, or use a more general blob storage option. Here are concerns data integrity it in case of fault, simplicity, and cost.

There are two main differences between self-hosting and using a service such as Google Drive. The cost, as Google Drive is significantly more expensive than using a vps to run your own file system. The other being that google drive is more tolerant to faults. There is almost zero chance of losing your data due to Google's servers failing.

A blob storage on the other hand offers an even more affordable solution to data storage than google drive, while providing a simple API to interact with the data. You also have the benefit of the data being replicated, which means there is no chance of data loss. Due to these benefits, we have decided to go with blob storage, and in our case there is no difference between S3, Azure, or any other provider, as they all have the same functionality we need.

To prove the feasibility of using a blob system we will start by creating a simple application to store and retrieve a small number of blobs. This shall confirm our ability to interact with the API provided.

	VPS	**Google Drive**	**Blob Storage**
Data replication	0	5	5
Simplicity	2	2	4
cost	5	3	4
Total Score	**7**	**10**	**13**

Accessing our Application Data

Access to data is critical to our app. Due to privacy and security constraints, a great deal of our data will be held externally, but we will still have to provide an interface to access remotely held data as well as data specific to our application.

With the architecture of our system, we have decided to make the front end and back end loosely coupled. Access to the data should also be platform agnostic. This means that if we wish to swap our front end, the existing back end shall remain. This separation of logic and responsibility, will also allow for the addition of alternate front ends; an example being a mobile app.

Having our front-end logic separate from our backend logic will constitute the need for an access mechanism. Our options are a standardized remote procedure call such as gRPC, a REST API over http, or a GraphQL implementation. Our concerns are developer familiarity, industry adoption, simplicity to implement, and appropriateness for external API.

Looking at gRPC first we see that its primary focus is on communication between two servers. That being said it does have a high industry adoption rate, and is most likely the most simple to implement. The issue is that it is not appropriate for front-end to back-end communication, and our developers lack familiarity.

GraphQL on the other hand is a “replacement” for REST. It is designed to minimize the number of calls to the back-end, by allowing front-end services to query more effectively. The downside is the added complexity in developing a GraphQL implementation, and our developers lack familiarity.

REST is our winner It is currently the industry standard for communicating between back-end and front-end services, while remaining simple to implement. To prove the feasibility of our rest API we will create a rest API that returns dummy data and stores requests within the data.

	gRPC	**GraphQL**	**REST**
Developer familiarity	1	1	4
Industry adoption	3	3	5
Simplicity	5	3	5
Appropriateness	0	5	5
Total Score	**9**	**12**	**19**

Documenting our API

To ensure we know the capability of our API it is critical to document the endpoints. This includes the endpoints, the verbs, the expected schema for requests, and the response schema. This is critical to ensure that future developers are able use our API to provide additional front ends such as a mobile app. Our main concerns are consistency/ease of updating, ability to generate code from our specification, and ability to generate documentation.

Our first option is to manually manage documentation per endpoint. This option is tedious and prone to error, all while not providing any auto generation features. The second option to document our code from within our application code using Annotations. The major issue is that is that it does not allow for generation of code. The final option is to the OpenAPI spec which allows for endpoint definitions as well as schema definitions. It is clear to us, that OpenAPI is the winner as it is it provides the best representation of our endpoints, all while having the added benefit of being able to generate our server-side code using our OpenAPI specification.

	Manual	**Java Docs**	**OpenAPI**
Consistency/ ease of updating	1	2	5
Code generation	0	0	5
Documentation Generation	1	3	5
Total Score	**2**	**5**	**15**

User Interface & User Experience

For our user interface, we had to decide on what sort of style to go with to ensure that the websites generated for the project were aesthetically pleasing as well as useful to the users that would be accessing them. We already knew that we would have to use HTML to create the website, but the challenge arose when we needed to decide on a style to use. We ended up with 3 possible options for our styling. The first was Bootstrap, which allows for higher functionality and integration of complex tools into a site, as well as having complete access to whatever else needs to be added on in terms of tools and plugins. The second option is Bulma, a native HTML import that also allows for extended use and additional plugins to be added in the future, but less focus on integration of some of the higher end tools we were planning on using, like Spring Boot. The last option was Custom CSS styling, which is just using CSS stylesheets for each webpage. This is incredibly clunky and creating stylish webpages is very difficult to accomplish without large amounts of dedication, but it is still one of our options if one of the other two exclude something vital to the project.

	Bootstrap	**Bulma**	**Custom CSS**
Industry adoption	4	3	5
Ease of Use	5	4	1
Components Included	5	4	1
Development Speed	4	2	2
Project Support	5	3	2
Total Score	**23**	**16**	**11**

We chose to use Bootstrap not only because of its high functionality or inclusion of other project types, but also because NAU already has a custom Bootstrap that we can use to ensure that the website can be recognized as an NAU webpage and has all the functionality of a NAU website built in.

To prove the feasibility of Bootstrap we plan on creating a simple landing page to better familiarize ourselves with the library. This will allow us to see if it meets our needs.

Securing Student Data

Keeping the student data safe and sound from the outside world is one the biggest challenges our team is facing, it has been an anchor to our ship. One of the requirements for the website to be secure is to not have stored any type of FERPA-related information on its database. In order to minimize the exposure surface of the website to the FERPA data, we can hash the student details to some extent like test scores, student name, and GPA. To merrily carry this information securely and display it on the website we must create a hashing algorithm, this builds an environment where there is no direct relationship between the student's name and their respective data. This hashed value will be linked to the Filmmaker Pro database, so that it will be used as a unique value to fetch data in real-time every time the user logs in. Hence, it is not saved in our website's database.

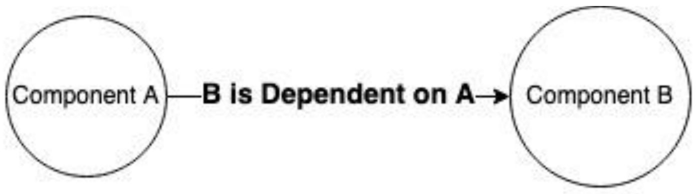
	MD5	**SHA2**	**SHA3**
Security Rating	2	3	4
Ease of Use to members	2	2	2
Industry Rating	3	4	5
Total Score	**7**	**9**	**11**

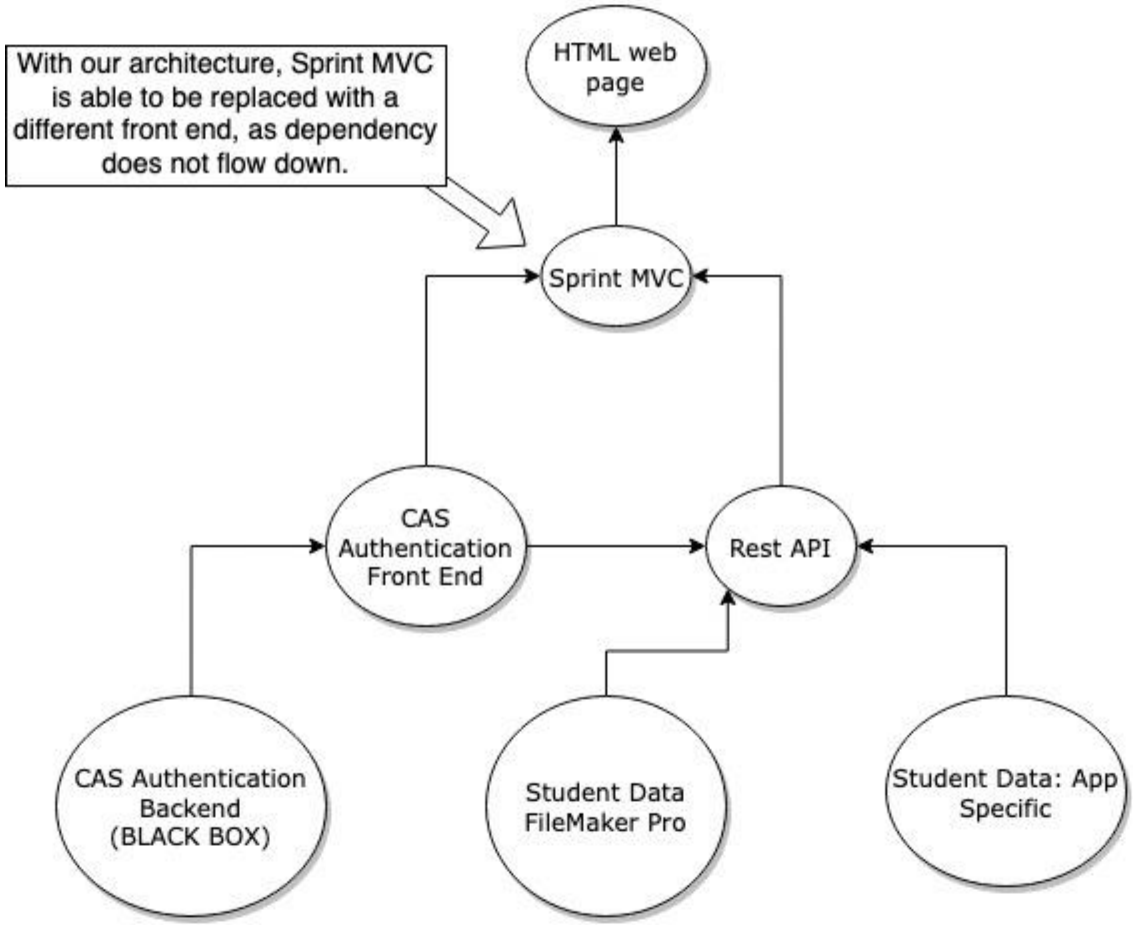
Technology Integration

During the analysis of our technical challenges we made sure to select technologies and design philosophies that would allow us to componentize our system. Each part of our system is able to be developed by separate teams, and have minimal dependency relations on other components. Broadly speaking our system can be broken up into two primary parts, front-end and back-end. The back-end is accessed through our REST API, which will be designed such that it is loosely coupled from any front-end architecture. This will allow future developers to replace the front-end system, or create additional views of our back-end system such as a mobile app.

Right now, most of our focus is into figuring out two main components of our application: CAS and the data structure of FileMaker Pro. As we are in the process of getting access to the FileMaker Pro application, we are unaware of the schema of the data, and the structure of the data and how it is being saved on the server. The closest we have been to that data is that FileMaker Pro can export all the data to an excel file in a form of CSV file, which is nothing but columns and rows of data, worst case scenario we could use this file to process our data, but we are planning on sticking to the strategy of using FileMaker Pro application in order to facilitate the idea of security and efficiency.

Integration Diagram




With our architecture, Sprint MVC is able to be replaced with a different front end, as dependency does not flow down.
HTML web page
Sprint MVC
CAS Authentication Front End
Rest API
CAS Authentication Backend (BLACK BOX)
Student Data FileMaker Pro
Student Data: App Specific

Conclusion

In these times of shortage of teachers in the education industry, there is a need for a change in the system, to enhance the quality and quantity of student teachers graduating. Our application would be able to make it easy and simple for the students and the department to track and follow through the checklist of requirements for the STIC program. Our program is not only restricted to Northern Arizona University but can be instilled upon other universities with similar program offerings across the states.

Our next steps are to confirm access to the services such as FileMaker Pro and CAS. This way if we are not able to obtain access we can then adjust our development plan accordingly. The other challenges at hand are ones that offer us an opportunity to combine our skills to create this new product.