COLORADOSCHOOLOFMINES
EARTH • ENERGY • ENVIRONMENT

InnoHub Innovators

CSCI 370 Report

Jeremy Utt
Justin von Rosenberg
Marina Feller
Jayden Pahukula

Revised June 12th, 2025

CSCI 370 Summer 2025

Prof. Donna Bodeau

Table of Contents

I. Introduction

The Labriola Innovation Hub at Mines is a campus makerspace, providing a wide range of workshops such as a Metal Shop and Wood Shop to students and staff at little to no cost. Each workshop contains many different machines, each of which have their own specific training.

For this field session project, the Labriola Innovation Hub wanted our team to help them build a system for easily looking up the trainings that Mines community members have completed when they enter workshops throughout the building.

Currently, when a student enters a workshop in the innovation hub, a TA must scroll through the training quizzes in Canvas, manually checking which are complete. After a TA looks at the trainings they can approve the student to use specific machines. This is a very slow process requiring the TA to scroll through a list of all possible trainings and requiring students to ask about each machine they want to use.

Ultimately, the goal of this project is to improve the process of students checking in to workshops. The Innovation Hub would like TAs and staff to be able to log into an application, search up students by a unique identifier such as an email or CWID, and view each student's training on a tab for the particular workshop they are checking in at. In order to build this application, we created a web application for a Raspberry Pi kiosk system. The interface allows users to login and provides all of the necessary functionality.

The Innovation Hub has extensive ideas for future improvements, such as tracking sign-ins, managing tool checkouts, and potentially even writing a new training system to extend upon the application we started to build for them.

II. Functional Requirements

As a minimum viable product, we were required to build a web application with a graphical user interface (GUI), with the primary function of querying user training data for Labriola Innovation Hub. The graphical interface produced needed to be able to perform the following tasks:

1. Search for and display a student's completed shop training
2. Separate completed training by which workshop it applied to
3. Authenticate users to ensure that only authorized people could view data

In addition to these requirements, the following optional functionality can be added in the future:

4. Blastercard scanner integration, so that authentication for accessing the training data can simply be done by tapping the Blastercard against a scanner.
5. Integration with a name tag printer, which would print a tag with the users name, and what training they have completed.

III. Non-Functional Requirements

In order for the application we built to be as successful as possible, it had to minimize the cost to the client. This meant keeping internet access from the application as lightweight as possible. The system we designed also needed to be scalable and able to incorporate additional functionality in the

future should it be desired. Our application is designed to use the following technologies for programming and hosting the application in order to achieve these design goals:

6. The application as a whole will be hosted using a server managed by Mines ITS so that it is allowed to connect to Okta and Canvas through IT in the future
7. Nginx as our web server running on the hardware provided by IT
8. Express.js as our backend framework for handling and sending web requests
9. Vue.js as our frontend framework for the web application to simplify GUI development
10. Canvas API integration to get the training modules and their associated data from Canvas
11. Okta Mines SSO service for user authentication

IV. Risks

One of the main risks with our application is the handling of sensitive information. Our application deals with Canvas and student records, so we needed to make sure that we were responsible and did not leak any private data. This was important because of FERPA regulations, which are explained more in the ethical considerations section.

Another big risk that stems from using the Canvas API is the security of the API key. This key that the application uses is extremely sensitive because it has teacher permissions, and if a malicious person gained access to it, they could theoretically view and tamper with student records.

One more risk that we needed to consider was authentication. Users will be logging in to this app, and authentication (especially on the Mines network) introduces new complexity. If something is implemented incorrectly, we could potentially compromise a lot more than just our application. While many of the security-crucial elements were not implemented by us directly, we still needed to make sure to utilize them correctly. All of these concerns factored into what we could consider a usable application.

V. Definition of Done

In order to provide valuable functionality to the clients, our application needed to simplify the lookup process for student training data. Specifically, the application was designed to let these users access individual student Canvas records that are tied to a specific training course. This provides real-time access to student progress and completion data, helping ensure that training requirements are being met. Access to the system had to be restricted to users within the Mines network domain and only those with proper authorization could be able to log in. The system was designed to clearly identify people who are authorized, ensuring that sensitive student information is only available to the right people.

At the end of the project, code that fulfilled these purposes was delivered to the client by copying the code into a repository owned by the Innovation Hub GitHub account. This also needed to include writing extensive documentation for the users of this application, as well as any future developers that might work on this project after us.

VI. System Architecture

The overall design of our application, as shown in Figure 1, is composed of a frontend user interface, a backend for accessing user data, a server for hosting both the frontend and backend, and integration with Canvas and Okta. Lab TAs and students have access to the frontend and sign in with Mines SSO in order to view the training data. The backend has access to Canvas to get the relevant information to be displayed on the frontend.

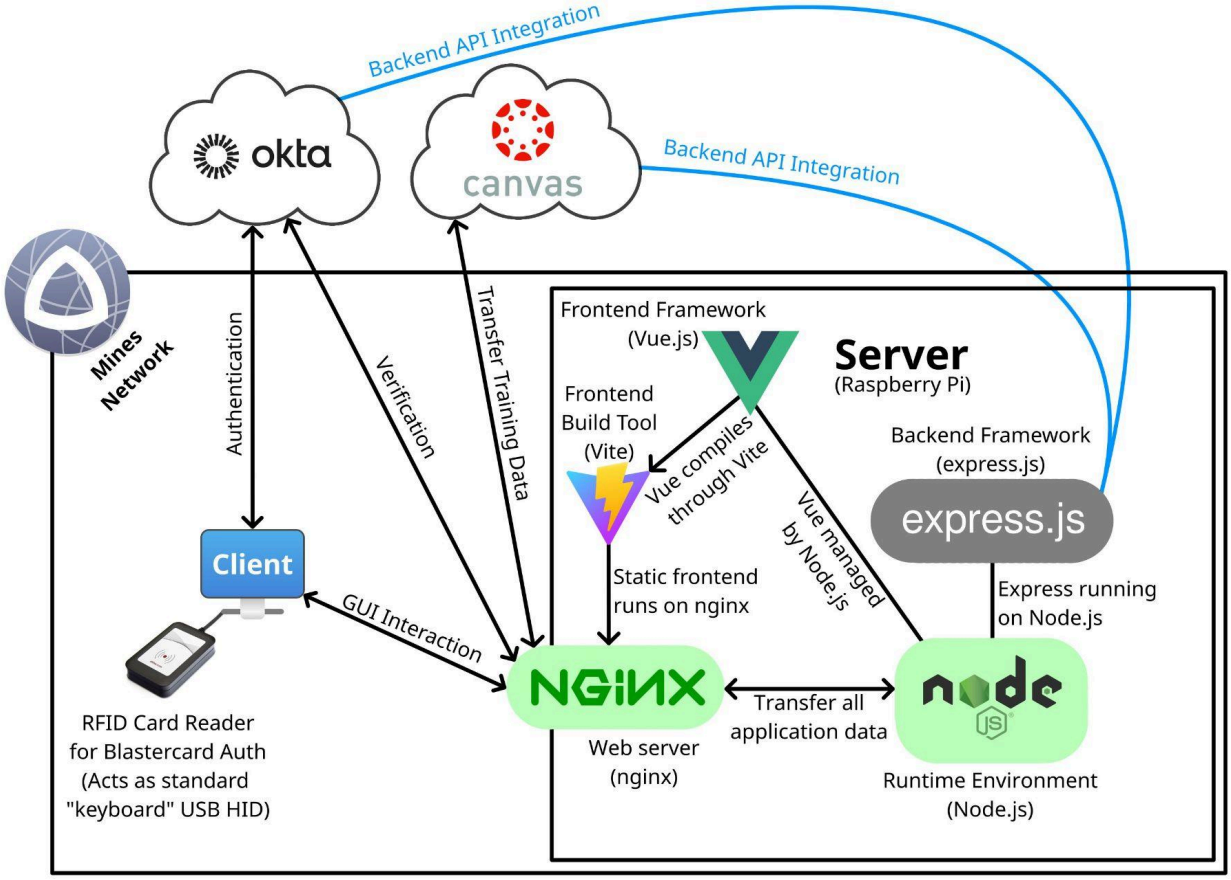

Figure 1. System Architecture

Frontend

The frontend for our application is built using the Vue.js framework so that it can use custom components and templating. This way, it was easier to build a modular application with data that updates in real time from JavaScript variable changes. The frontend can be built out of regular HTML tags but subcomponents can be defined in separate files and JavaScript variables can be used directly in the HTML code instead of needing to write code for specific updates that need to be made to DOM content. Using this framework, we created two pages for our application.

The search page, shown in figure 2, allows TAs or staff to search for students and see which trainings they have completed. They can also filter by which workshops' trainings to display using toggleable workshop buttons.

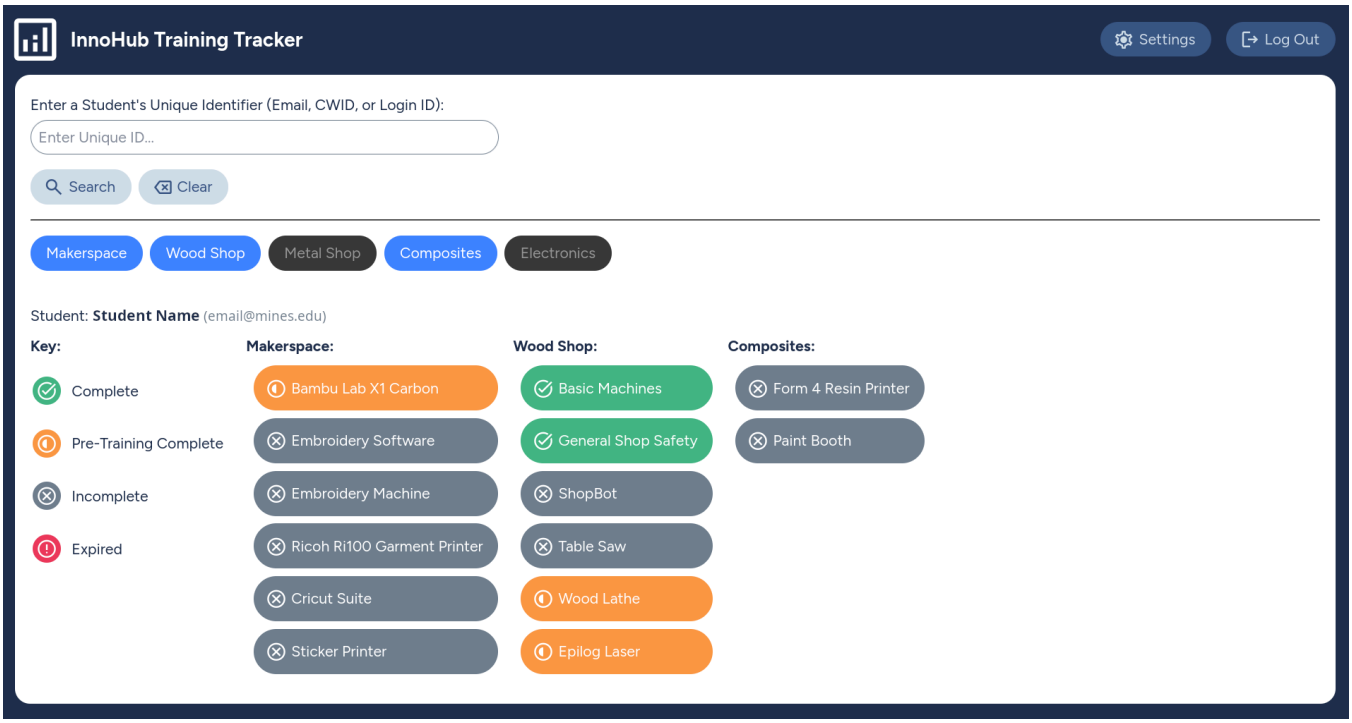

Figure 2. Search page

The settings button in the top right takes you to the configuration (config) page, shown in figure 3. This page allows staff of the Innovation Hub to add or remove tools, specify which workshop a tool belongs to, and assign Canvas quizzes to a certain tool.

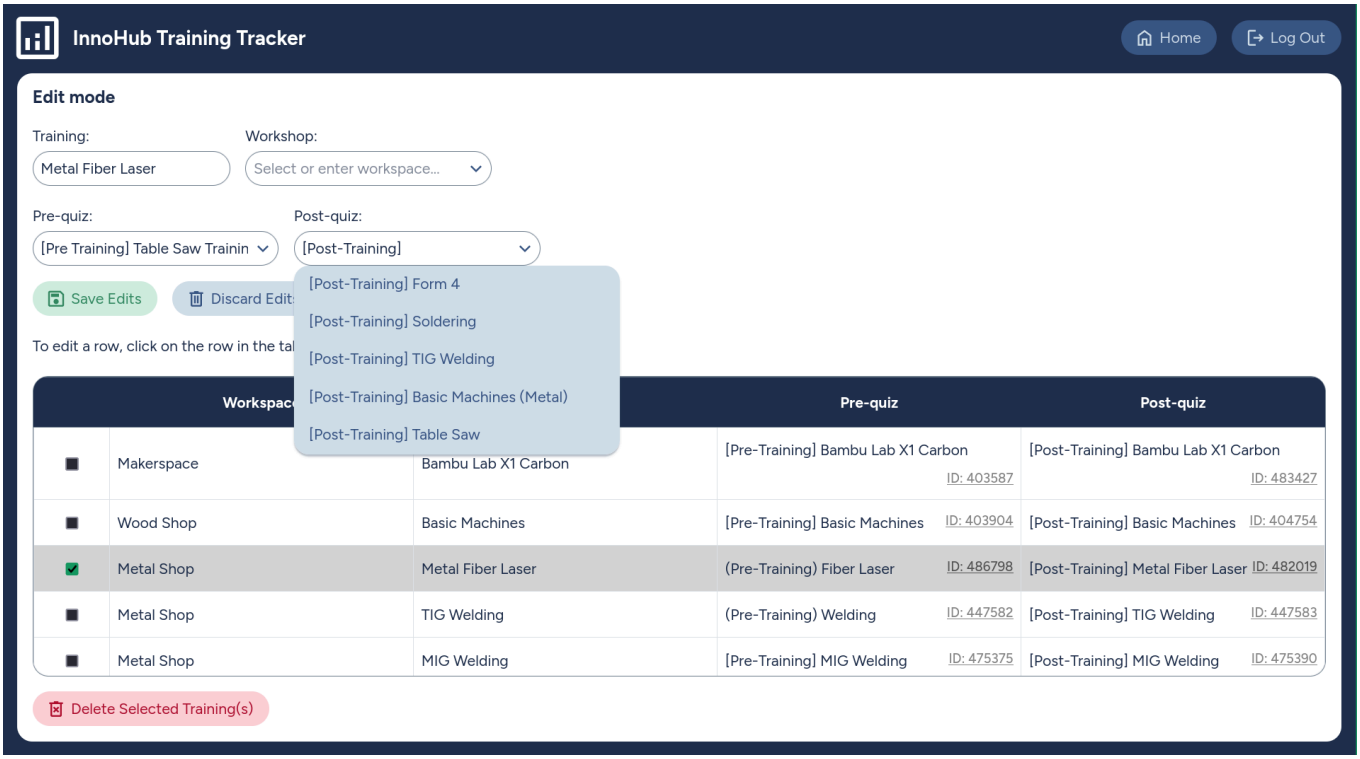

Figure 3. Configuration Page

Server

The web application our team created will eventually run on a server managed by Mines ITS using Nginx to load balance and host web files. Our Vue.js code is able to compile down into static html and js files which can then be served by Nginx to run the frontend on client computers. In order to run our Express JavaScript endpoints, a linux program called pm2 starts multiple instances of our express code running on Node.js. From there, Nginx uses a reverse proxy to handle any incoming requests to our backend API and balances the load across the backend instances started by pm2. In addition to load balancing, Nginx and its reverse proxy allow us to keep the server more secure since there is no need to directly expose the backend services' ports to the internet. The hosting methodology used allows the web server to be secure and performant while remaining relatively inexpensive.

Backend

The backend is where most of the business logic of the application is handled. Its main job is to expose API endpoints that the frontend uses to get the data to display. This data is queried from the Labriola InnoHub Canvas page, using the Canvas API. It is authenticated by its own Canvas API key that has teacher level permissions in the course.

The endpoints that are provided by the backend API are listed below, and specific implementation details and data formats are available in the code documentation.

POST	/auth/login	Logs into the web app, and returns the session token.
POST	/auth/logout	Logs out.
GET	/api/student/:id	Get the quiz completion status of the student with the given ID. The ID can be CWID, email, or login id.
GET	/api/config	Get the current configuration, which is a list of machines, which workshops they belong to, and their corresponding Canvas quiz ids.
PUT	/api/config	Update the current configuration.
GET	/api/config/quizzes	Get a list of available Canvas quizzes, used for matching quiz names and ids in the config page.

Table 1. Backend API endpoints

Of course, in order to query any data, the client needs to be authorized to access that data. The backend expects a session token for all non-auth queries, and it expects admin credentials before modifying the config. The backend verifies this token with Okta before returning anything.

For the sake of easy development, the backend was built as an Express API, running with Node on the server. It is configured as a systemd service that continuously runs and serves the incoming requests.

Authentication and Security

The system supports two types of authentication: a long-term OAuth2-based login through Mines SSO (Okta), and a temporary Basic Authentication method designed for testing and demo purposes while waiting on IT approvals.

Oauth Authentication

Figure 4 depicts the authentication process.

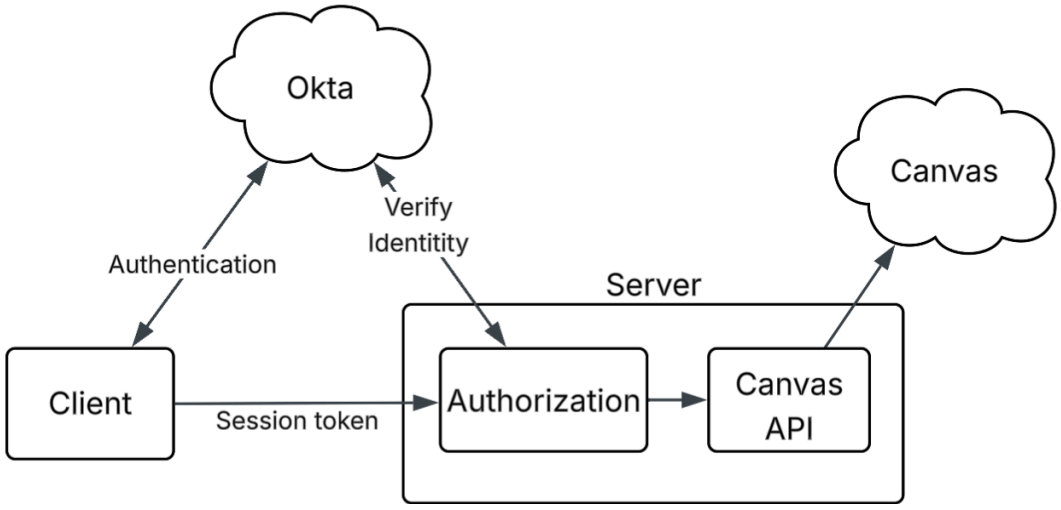

Figure 4. Authentication process flowchart

OAuth2 is implemented using Passport.js with the Authorization Code grant type. As shown in Figure 5, Users are redirected to the Mines SSO page, where they log in using their standard campus credentials. Once the user logs in, their information is returned to our backend through a secure callback, which then queries Canvas using their email to determine what role they have in the system:

- Teacher → Level 3 (Full access, including system config)
- TA → Level 2 (Access to all student data)
- Student → Level 1 (Access only to their own records)

The backend saves this session data and creates a JWT (JSON Web Token), which is returned to the frontend via an HTTP-only cookie. The frontend uses this cookie for future authenticated API calls, with no need to handle the token directly in JavaScript. This process is shown in more detail in Figure 5.

Until IT completes the Okta integration process, we are using a self-hosted OAuth2 provider called Authentik for local development. Authentik supports all the necessary protocols (OAuth2 and SAML) and can be swapped out for Okta without major code changes.

The backend accepts this token either from the cookie or an `Authorization: Bearer <token>` header. All tokens are verified on each request, and no sensitive data is sent without validation. To prevent common security issues like XSS, we use HTTP-only cookies. All login-related traffic should go over HTTPS, and no authentication details are stored in frontend JavaScript or local storage.

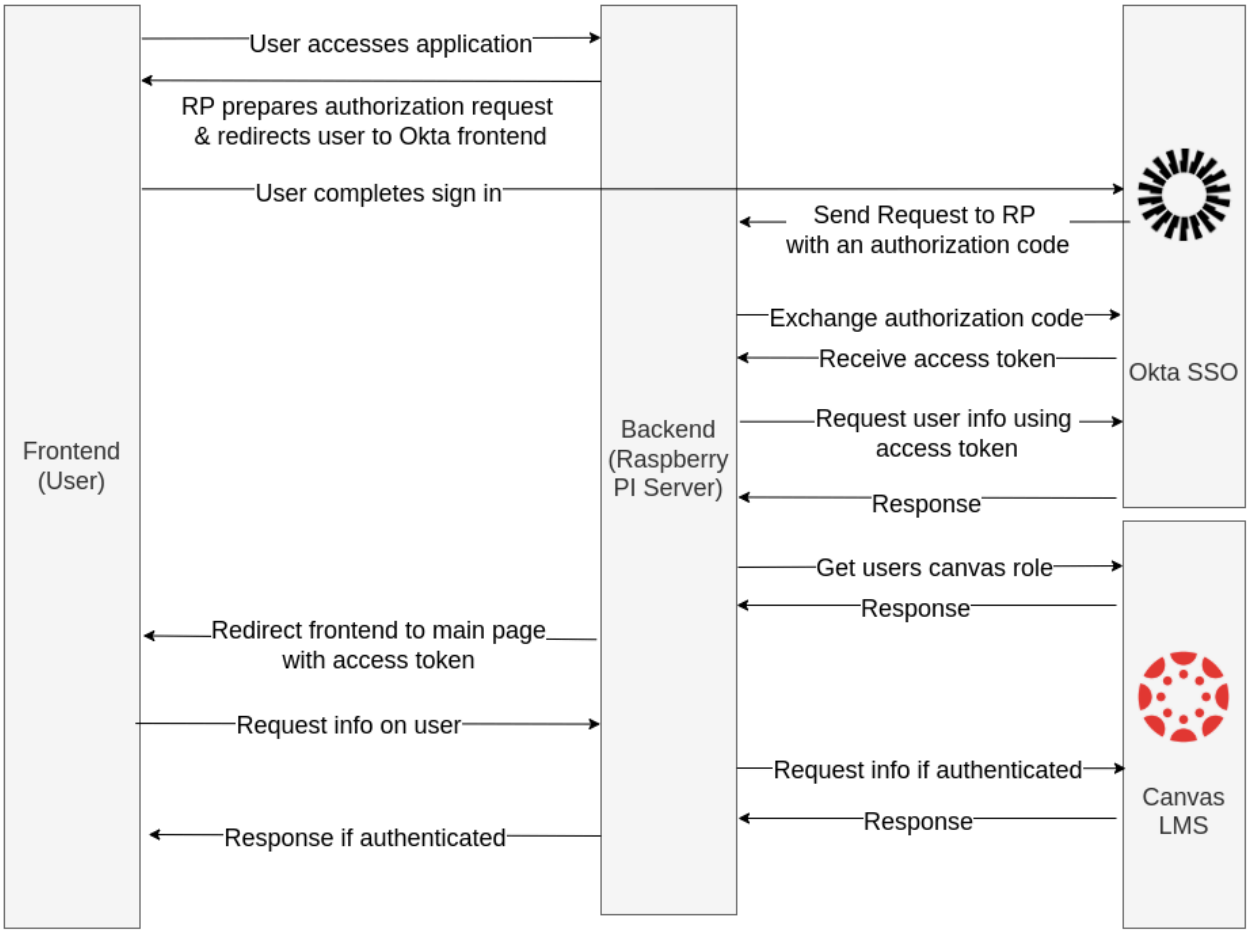

Figure 5. Detailed Okta authentication process flowchart

Basic Authentication

As a short-term solution, the system supports a basic login flow that allows Teaching Assistants (TAs) and Admins to use shared passwords for access. This method exists mainly for development and testing before full integration with Mines SSO through OAuth2. It is intentionally simple but does include basic security practices like password hashing and token-based sessions.

When a user visits the login page, they can enter a shared TA password. If the password is correct, the backend signs a JWT (JSON Web Token) in the same way that it would when using the Oauth implementation. This allows the two different authentication schemes to be interchanged without much extra effort.

The `/config` page, which is considered sensitive, is protected by an extra password prompt. Even after logging in as a TA, users must enter a second admin password to unlock configuration features. The system does not promote the user to a full Teacher role. Instead, the admin password is held temporarily in JavaScript memory and sent along with every config-related request. This password is not saved between sessions, so users have to re-enter it after each page refresh.

On the backend, both the TA and Admin passwords are stored in hashed and salted form. A secure hashing algorithm (bcrypt) is used, and the secret used for signing JWTs is also stored in the environment file. This ensures that even if the environment file was compromised, raw passwords would still be protected.

After login, the user's token is used to identify them for all protected routes. The backend checks whether the token is valid and if the user's role is high enough for each API endpoint. This protects the system against unauthorized access, even if someone tries to bypass the frontend.

While this setup works well enough for limited use, it's not designed for long-term deployment. Passwords are shared among users, no individual identities are tracked, and the admin flow depends on client-side memory. Once OAuth2 is eventually in place, this method can be removed entirely.

Frontend Route Protection

Vue Router handles page navigation and enforces access control through route metadata. For example, pages like `/config` include a `meta: { requiredAuth: 3 }` tag, which tells the router to only allow access to users with Level 3 permission. A global navigation guard (`router.beforeEach`) checks this level before allowing users to enter a page. If the user doesn't meet the required level, they're redirected to `/login` or a no-access page. However, this frontend logic is purely for user experience. All real enforcement happens on the backend, where every API call checks the user's authentication level before returning any data.

VII. Software Test and Quality

In order to ensure that our final product functioned as expected, we implemented several procedures to test our code in different capacities. We also created team rules to dictate how code should be reviewed and incorporated into our application.

To start off with, we wrote unit tests for code blocks that could function independently. When writing helper functions for backend logic, we got them to a state where they could function without extra test code or input generators. Then, we wrote tests to try various edge cases and ensure the correct functionality of the functions. By following this process to test standalone helper functions immediately, we largely prevented broken code from being used in other parts of the program.

Beyond unit testing of small pieces of our backend functionality, we also made larger tests to verify the expected output from our backend's API endpoints. By passing fake Canvas responses into our API endpoints through Vitest mock functions, we were able to isolate the functionality of our own code from changes in the Canvas API response. This ensured that all different responses that Canvas might deliver were processed correctly by our code without having to actually produce these responses from a third party program. Specifically we tested that the API endpoints returned the expected response under different circumstances, such as if the Canvas API is down, if the API token is no longer authorized, etc. We checked that the endpoints returned to the correct error statuses and information.

The last backend testing method we used was to manually run full tests of our endpoints that included contacting Canvas and pulling actual data. These tests were run manually since the Canvas API falls outside of our control and may change over time. When the tests produced errors, we

documented and quickly fixed the code before pushing updates so that errors did not make it into the final application.

For our frontend, since it is difficult to automate testing of graphical display features, our testing entirely consisted of manually using the interface. As we developed features for the frontend of our application, we constantly looked at the results of our code. Only code that produced correct visual results was committed to the GitHub repository for our application. This made sure that the code in the main branch was always functional and did not display broken elements.

We had a couple of other ways beyond testing to verify that our application functioned according to the client's specifications. First of all, we regularly demonstrated the functionality we had implemented at weekly meetings with our client contacts. By providing this frequent opportunity to give feedback on the current state of the application, we were able to quickly correct any misunderstandings and get back on track with implementing features.

As one last method of ensuring overall code quality, we followed our team rules that all code must be reviewed by a second individual on the team before it is pushed to the main branch of our GitHub repository. By asking the other members in the group to review code, any potential bugs or code cleanliness concerns that the author may have missed were easily corrected before they affected the quality of the application.

After development finished, we were able to deliver code to the clients that passed thirteen different automated tests. The delivered product has also been extensively tested by manually searching for specific users with useful training data and checking that config changes from the GUI have actually applied to the config file. These manual tests as well as automated test results have shown that the application is working as intended with no obvious bugs.

VIII. Project Ethical Considerations

The main ethical consideration for our project was making sure that student data was handled responsibly. The application needs to comply with the Family Educational Rights and Privacy Act (FERPA), as it accesses Mines students' personally identifiable information (PII) such as their name, CWID, email, and more. In the original proposal, this information would have been stored in a database. This approach would have required us to make sure that everything was encrypted, and that we safely stored the keys, which would have added security complexity. We would not have been able to use the CWID as a unique identifier in the database because it is considered sensitive data. This is the primary reason we decided to not use a traditional database, and have decided to never store PII persistently.

We also needed to consider responsible usage of sensitive information during development and testing, because we needed to work with real Canvas data. In order to be as ethical as possible, we used the personal info of our own team members when possible; otherwise we asked for the consent of close colleagues to use their training information. We also created dummy data populated with fake values when we were not able to use any real Canvas data.

These issues are a perfect example of ACM Code of Ethics section 1.6, which is to respect privacy. It states, "a computing professional should become conversant in the various definitions and forms of privacy and should understand the rights and responsibilities associated with the collection and use of personal information" [1]. Because we are dealing with private information, we have a

higher ethical responsibility to be responsible with users' data. It is for this reason that we took as many precautions as possible.

IX. Results

We were able to complete almost all of the main goals that we had for this project, and all of the main functionality works as expected. The biggest obstacle that we couldn't get over during this project was coordinating with Mines IT.

Specifically, we did not have enough time to go through the steps required to integrate Mines SSO into our application, or to get it running on a proper production server. The process for getting these things approved is lengthy and complicated, involving many different rounds of reviews for security and accessibility. When we realized that this was not going to be feasible in the five week time frame, we consulted with the client about options moving forward. The client decided that they would rather get a working proof of concept, even if it meant cutting some of the features they were originally hoping for.

The main feature that we had to compromise on was the Okta sign in. In order to get a working product released, we switched to a system that uses static passwords. To access the application, the TA signs in with the TA password. This lets you use the main features of checking the training status of a student. There is also a separate teacher password that is used for accessing the configuration page. The two separate passwords ensure that the configuration can still only be changed by teachers. Both of these passwords are set by the user when the application is installed, and can be changed later.

The static passwords are not stored in plaintext, and they are also used to encrypt the Canvas API token. This means that even if the physical device is compromised, the API token cannot be recovered without the TA password. Despite these security measures, this system is not meant to be a replacement for Okta and Mines SSO.

This change made it impossible to allow students to log in to the application to check their own data. It also means that everything has to run locally since we cannot get a static server to host on. The TAs can access the application on the Raspberry Pi itself, like a kiosk, and look up student information on that device. The Raspberry Pi can then be kept in a secure location so that API keys and other files are protected.

Aside from the work with OAuth that was not able to be completed, we had started working on some extension features that did not end up reaching completion. Particularly in the case of Blastercards and writing a kiosk display mode for the application we had started researching these features in case we had time to work on them. We made sure our findings on these extra features are documented for future developers since we did not get around to implementing them.

Despite the functionality that did not end up getting completed, we were able to take a functional application to our clients at the end of our five week development period. By logging in with the temporary password security system users are able to search for student trainings and modify the application's configured tool trainings in a decent looking graphical interface. While the alpha deployment that we delivered does not encompass the application's full desired functionality, it provides a good basis that the clients can use and collect feedback on for the next development team.

X. Future Work

Given that we made some sacrifices on functionality to get a working alpha of our application released to our clients, we have a significant list of well defined steps for the continued development of our application. The following list highlights what we believe are the next key steps in moving the application towards its final form:

1. The application should be placed on a Mines ITS managed server as soon as possible. This enables much of the continuing feature development of the application that cannot currently happen.
2. With the help of an Okta development environment on a Mines server, the application should be modified to support OAuth SSO logins instead of the temporary login solution it currently uses. This would make the training tracker more secure and allow for enhanced functionality with fewer concerns.
    a. Most of the code for OAuth already exists in the application and is documented in GitHub READMEs. It simply needs to be uncommented and put to use in the new development environment.
3. Hosting the application on a Mines server would also allow Mines ITS to give the application a static IP, a domain name, and an SSL certificate so that it is compatible with access from more user devices.
4. The training tracker application should undergo the correct code reviews with Mines ITS to be certified for long term use. With this, ITS will give the application a development Canvas key so that it no longer has to be run with an individual user's access token.

Beyond the necessary steps for the app's continued development outlined above, we were given several stretch goals for future functionality to implement. The features below would help make the application even more impactful for its users:

5. The application could have Blastercard support in the future so that students could simply tap a card upon entering a workshop rather than having to type in their information.
6. Logging for times when students enter or leave workshops could provide valuable data on attendance, workshop usage, and machine usage.
7. Current users in a workshop could be displayed on TVs to allow Innovation Hub TAs to monitor machines being used in a workshop even more easily.
8. Tools that students check out from the tool crib could be tracked in the application for easier viewing and management.
9. A system of physical kiosk machines could print nametags and allow students to easily view their own trainings without visiting a workshop.

Between the clear next steps and the many opportunities for expansion, the training tracker application we have started has the potential to become a useful or even integral tool for managing operations in the Innovation Hub.

XI. Lessons Learned

One of the biggest things we realized was how much time it takes to get approval from IT, especially when it comes to authentication. Even though we knew how to technically implement Mines SSO using Okta and OAuth, we didn't expect the approval process to take so long. There were multiple steps, including security reviews and policy checks, and we just didn't have enough time in the five-week field session to get through all of it. In the future, we'd start these approval steps much earlier or make sure the project timeline gives us enough room to wait.

We also saw how important it is to design around security and privacy when working with student data. Since we were dealing with names, emails, and potentially CWIDs, we had to think about FERPA compliance from the start. That's why we decided not to use a database to store training records. Instead, we pull data in real-time from Canvas and avoid storing anything long-term. This reduced risk and helped us avoid having to worry about encrypting data or securing a database.

Another thing we learned was how helpful it is to have regular client meetings. Talking with the Innovation Hub every week gave us clear direction and helped us make quick decisions, like choosing to use static passwords instead of full SSO so we could get a working demo done in time. Their feedback helped us stay focused and made sure we were building something useful, not just what we thought they wanted.

Finally, this project showed us the value of testing and good documentation. We found that writing unit tests early helped prevent bugs later on, especially in backend functions. We also made sure that only working code made it into our main GitHub branch. And by the end, we had clear setup instructions so that even if we weren't around, someone else could pick up the project and run it.

XII. Acknowledgments

We would like to express our gratitude to everyone who supported us throughout this project. We want to thank our clients from the Labriola Innovation Hub, Victoria Bill and Julia Roos, for providing us with this opportunity and for their continuous guidance and feedback. We are especially grateful for their understanding and flexibility when we faced roadblocks during development. We would also like to thank our project advisor, Professor Donna Bodeau, for her mentorship and support over the course of the field session. Her expertise in engineering gave us helpful guidance that was crucial to the direction and final design of our application. Finally, we would like to thank Ric Santiago Oquendo from Mines IT, for working with us through the required processes of IT approval for our project, and providing us with crucial information and resources, which greatly influenced our development decisions for this project.

XIII. Team Profile

Jeremy Utt

Jeremy is a rising senior studying computer science with a minor in electrical engineering. Has experience with software engineering at SEAKR engineering., and is part of the Battery Workforce Challenge. His hobbies include Radio communications, Cyber security, Networking, and Video Games

Justin von Rosenberg

Client Point of Contact

Justin is a rising Junior studying CS at Mines. He enjoys doing audio engineering for Mines Little Theater in his free time as well as experimenting with Linux and various personal coding projects.

Marina Feller

IT Point of Contact

Marina is a rising senior pursuing a BS in Computer Science at Mines. She has been a part of Phi Theta Kappa, oSTEM, and the Concert & Marching Band at Mines. In her free time, she enjoys Linux, computer networking, software development, electronics repair, table tennis, camping, and hiking.

Jayden Pahukula

Jayden is a graduating senior with his bachelors in computer science from Mines. He has work experience as a software engineer at Lucid Software, and is a member of Mines Oaks, and ACM. His hobbies include competitive programming, Linux, volleyball, and piano.

References

[1] Association for Computing Machinery, “ACM Code of Ethics and Professional Conduct,” *Association for Computing Machinery*, Jun. 22, 2018. https://www.acm.org/code-of-ethics

Appendix A – Key Terms

Term	Definition
GUI	Graphical User Interface
TA	Teacher Assistant
API	Application Programming Interface
SSO	Single Sign On
framework	A modular library of JavaScript code that allows for easier development of web-based user interfaces
DOM	Document Object Model, a structure allowing interfaces to be changed dynamically with code
FERPA	Family Educational Rights and Privacy Act
PII	Personally Identifiable Information

Appendix B – Documentation

In order for the client to make full use of this project, we have developed comprehensive documentation for all levels of involvement. First, we created a user manual, intended for TAs and staff that will be using the application day-to-day. This contains basic instructions on how to use the web interface.

The next set of documentation goes along with the installation script that we developed. The install script is a bash script that automatically installs everything that the application needs to run on a fresh Raspberry Pi. The accompanying documentation details how to run the install script, as well as setting up all the necessary security secrets.

We also developed documentation for any developers who continue this project in future semesters. This includes many things like how to set up the development environment, what the code is doing in certain places, how different parts of the code work together, etc. We also tried to explain

many of our design decisions, since many things changed throughout the course of the project. Ideally, any future developers won't have to make the same mistakes that we did, and can get caught up relatively quickly.

All of this documentation has been handed off to the clients, and is available with the project source code.