COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

;DROP TABLE teams; –

Jeremy Gillett

Andy Nguyen

Duy Nguyen

Alex Rogers

Revised June 18, 2026

CSCI 370 Summer 2026

Dr. Phil Romig III

Revision	Date	Comments
Started working on Field Session	May 18, 2026	Completed Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements XI. Team Profile Reviewed Sections: References Appendix A – Key Terms
Sprint 1	May 22, 2026	Completed Sections: All previous and: IV. Risks V. Definition of Done
Sprint 2	May 30, 2026	Completed Sections: All previous and: VI. System Architecture
Sprint 3	June 1, 2026	Completed Sections: VII. Software Test and Quality VIII. Project Ethical Considerations Updated Sections: III. Non-functional Requirements V. Definition of Done VI. System Architecture
Sprint 4	June 8th, 2026	Completed Sections: IX. Project Completion Status X. Future Work XI. Lessons Learned
Sprint 5	June 15th, 2026	Completed Sections: XII. Acknowledgments Updated Sections: All Sections

Table 1: Revision history

Table of Contents

I. Introduction

Hazen Research & Development is a company specializing in mineral processing, precious metal refining, and rare earth element extraction. The company performs a wide variety of laboratory analyses to determine the composition and characteristics of mineral samples, generating large amounts of scientific data that must be stored, organized, and retrieved efficiently. At the inception of the project, much of this information was maintained through spreadsheets and manual recordkeeping processes, making long-term organization, searching, and secure management increasingly difficult as projects continue to grow.

The primary objective of this field session project was to design and develop a secure, locally hosted database management system that modernizes Hazen's existing workflow while remaining compatible with the company's current computing environment. The proposed solution provides a web-based graphical user interface (GUI) that allows authorized users to upload, manage, search, and export laboratory data without requiring direct interaction with the underlying database. Because the client operates on older Windows-based systems, primarily Windows Vista, the software was designed with simplicity, compatibility, and ease of deployment as key priorities.

The system was developed in a 3-jointed full-stack development process. In doing so, simplicity and efficiency are maintained throughout the development process to ensure the scope of the project was not exceeded in the allotted five-week period.

The primary stakeholders for this project are the engineers, technicians, and administrators at Hazen Research who rely on accurate laboratory data throughout the research project. Upon project completion, the software and source code were delivered to the client, enabling their internal IT personnel and developers to maintain, expand, and customize the application as future laboratory requirements evolve.

II. Functional Requirements

All listed elements were engineered to be objective, discrete, and verifiable through standard software testing methodologies. While the application must accommodate varied testing technologies through dynamic schemas and data storage, it must enforce strict validation rules such as filtering and inheritance. This framework ensures data integrity while role-based access control protects from unauthorized user modifications.

Table 2, shown below, outlines the formal, testable functional requirements for the system:

Requirement	Description	Importance
Dynamic Schema and Flexible Entry	The system must allow authorized users to define custom field labels and data schemas to accommodate any future technologies and testing methods.	High
Project Filtering	The system must provide an interface that filters data records based on a 5-digit numeric identification code.	High
Data Support	The database must support upload, storage, and retrieval of multiple data formats including numeric, images, and PDFs.	High
Inheritance	The system must establish a link between an original sample, its subsamples, and its replicates, enforcing automatic inheritance from the parent record.	Medium

Export Functionality	The system must enable users to export data tables into a standard Comma-Separated Values (.csv) file format.	High
Role-Based Access Control (RBAC)	The system must enforce three tiers of user permissions: Superuser (admin overrides and schema changes), Manager (advanced editing and user assigning), and Technician (read and flagging).	High
Timestamps	The system must automatically log timestamps for the initial record creation, user-entered test run time, and most recent modification.	High
Graphical User Interface (GUI)	The system interface must be deployed as a web-based GUI, abstracting database operations away from the command line interface (CLI).	High

Table 2: Functional Requirements

III. Non-Functional Requirements

1. Accuracy and Precision

All numerical calculations and statistical outputs must be calculated and displayed up to three decimal places or four significant figures where applicable to maintain data integrity.

2. Deployment and Network Architecture

The project must be implemented as a locally hosted web application utilizing the Streamlit framework in Python as well as other necessary imports like Pandas or Matplotlib.

3. Database Integrity

All relational data, parameters, and metrics must be persistently stored and managed using a PostgreSQL database.

4. Performance and Scalability

The user interface must render updates within 3 seconds of a database query execution under standard conditions. The codebase and database schema will be documented to facilitate future development.

5. Security

The database must be configured with password authentication.

6. Usability

The user interface must be a responsive, dynamic design that adapts cleanly to differing resolutions without breaking the layout.

7. Windows-Based

The software must be capable of running on a Windows-based computer without overwhelming the computer. This is due to the fact that the company has computers running on old Windows versions such as Windows Vista. The database must be able to be stored on a Windows computer locally.

IV. Risks

1. The clients are predominantly using Windows Vista machines, which have significantly limited capabilities for accessing newer software. The risk of building a web-based database that is too modern for an older operating system is high, mandating that the database is created with a process that works with older software. To mitigate this, Streamlit was selected for the front-end interface because of its compatibility with Windows Vista.

2. The product will take an input field from a user that queries a database, so it must be ensured that the database uses query parameters to avoid the risk of SQL injection (see Appendix A). This risk is extremely high and quite likely if a user wants to be malicious, but with a well-designed architecture using parameterized queries, it can be easily mitigated.

3. If a user searches for a large amount of data, it could take a long time for the front end to populate this data into the GUI, leading to a risk of inefficiency. The risk of this is likely, and its impact is moderate, as it will create a less positive user experience. To mitigate this risk, smart SQL queries were written and tested for maximal efficiency.

4. It is possible that the front-end tools being used cannot fulfill all the functional requirements. This risk was mitigated through extensive research of the Streamlit documentation and early implementation of the most technically challenging requirements. Ultimately, the team developed the front-end using Streamlit after evaluating several available frameworks.

5. Creating a product with a database that has not been properly tested could cause the users to lose important data. This risk has a major impact, as providing an unstable, broken product to the client will not meet the definition of done. It is expected that initially these risks will be likely, but as development continues fewer bugs will be present. To mitigate the risks, frequent testing was performed using a test database populated with client-provided dummy data to ensure that queries were returning the required data without crashing or deleting data.

6. The team's experience in creating front-end applications could be a limiting factor in meeting development goals. The team has proficient experience in developing and working with database software but has only slight experience in developing production-level front-end applications. This risk has major implications for the viability of the project, as a front-end application is required for the definition of done to be met. There is a high likelihood that the team will be affected by this lack of experience, but this risk can be greatly reduced and mitigated through proper research and learning. The team will also address this risk by using software that is more developer-friendly, namely Streamlit. Using this approach will allow the team to learn basic web development without requiring full mastery of high-end development tools such as React or Angular.

7. The client requires that each user is assigned a role of either admin, manager, or technician, each with different capabilities in the database, which leads to the risk of having improper security around these roles. It is paramount that these roles are properly implemented to avoid the risk of data being deleted or modified by an unwanted user. This risk has a major impact, and without any security in place, unwanted users could likely perform disallowed tasks. However, this risk was easily mitigated by careful implementation within the database using RBAC (role-based access control) features.

8. The software may not be intuitive for technicians and employees at Hazen to quickly and easily learn. This risk is likely, as many employees at the company are inexperienced in using computers frequently and their current architecture is out of date. This risk has major implications because if a clunky, unintuitive product is produced, employees will not desire to use it or take the time to learn how to use it. This was mitigated by clear UI, tooltips, and conversations with the client about how to best train new users of the software.

9. Locally hosted servers and databases may not have cloud backup, which could lead to massive amounts of data being lost if the main server were lost. This is highly likely and has major implications because losing research

data would be a massive inconvenience to the client. To mitigate this, the team recommended that the database be backed up to the cloud through the client's IT procedures.

10. Multiple users could try to access, edit, or delete data simultaneously. This risk is unlikely, as the client's workforce is small, but it has major impacts on the credibility of the data within the database. To mitigate these risks, database scheduling procedures were put in place in the event of concurrent operations.

V. Definition of Done

The software must do the following:

1. Take CSV data as input
2. Extract desired wt% data from the provided input
3. Insert data into a relational database
4. Filter data within the database based on elements or sample ID
5. Interact with the data using a front-end GUI on a locally hosted website.

For testing, the software must accurately display the dummy data provided in an easily readable table. The product was delivered to the client at the end of the 5-week field session. The client received the source code along with clear instructions for properly installing the software to run on a locally hosted network.

VI. System Architecture

The architecture was designed to be simple while maintaining data security and accessibility. The main GUI has two forms of input: human input when inserting new data from experiments to store into the database and the ability to pull existing data from the database for technicians to reference. Afterwards, the GUI was able to export selected data into a .csv format to be exported into an Excel spreadsheet.

For the database, the selected element was a locally hosted server on the client's personal computer utilizing PostgreSQL. PostgreSQL was selected due to background knowledge of its functionality and capabilities in storing data. By locally hosting the database, data storage will be dependent solely on the client and the size of imported data.

For the middleman API, the selected element was FastAPI. FastAPI provided a framework for creating highly efficient RESTful APIs with limited prior experience required. It is simple to create and implement secure endpoints that will allow for a layer of separation between the database and the GUI frontend. Additionally, FastAPI is built on top of Pydantic, a Python library built for data validation. Because of this, using FastAPI to connect endpoints to a database is extremely secure, as the Pydantic framework allows for proper validation of inputs before making calls to the database. Furthermore, FastAPI has extensive documentation and is widely used, so it proved to be an appropriate choice for the API necessary for the client's requirements.

For the GUI frontend, the selected element was Streamlit. Streamlit allowed for the creation of user interfaces through Python commands, enforcing simplicity in building the GUI while ensuring compatibility with Hazen's systems. Additionally, Streamlit permits fast, simple connection to an API, allowing components to be separated and enforced robustly.

Finally, the predominant risk would be the security of the database, with an emphasis on its integrity. The database was used to store vast amounts of vital research data, so it must be strongly protected from deletion or loss of large amounts of data. To ensure this protection, the team developed the database to avoid accidental data loss, along with strongly advising the client to store the database in the cloud. With these protections in place, the likelihood of losing vast amounts of crucial data becomes significantly lower.

Multiple factors were considered in determining how to protect the database for a local web application. After a discussion with the group advisor, the preferred option was the use of Hypertext Transfer Protocol Secure (HTTPS). However, the client was presently concerned with receiving a product that is functional without the need for HTTPS security. The team built the application using HTTP routing, with a future goal of implementing HTTPS protections using client-side certificates to ensure the legitimacy of the web application.

Shown below are Figures 1 and 2, which represent how the system was implemented and the way in which a user interacts with the application.

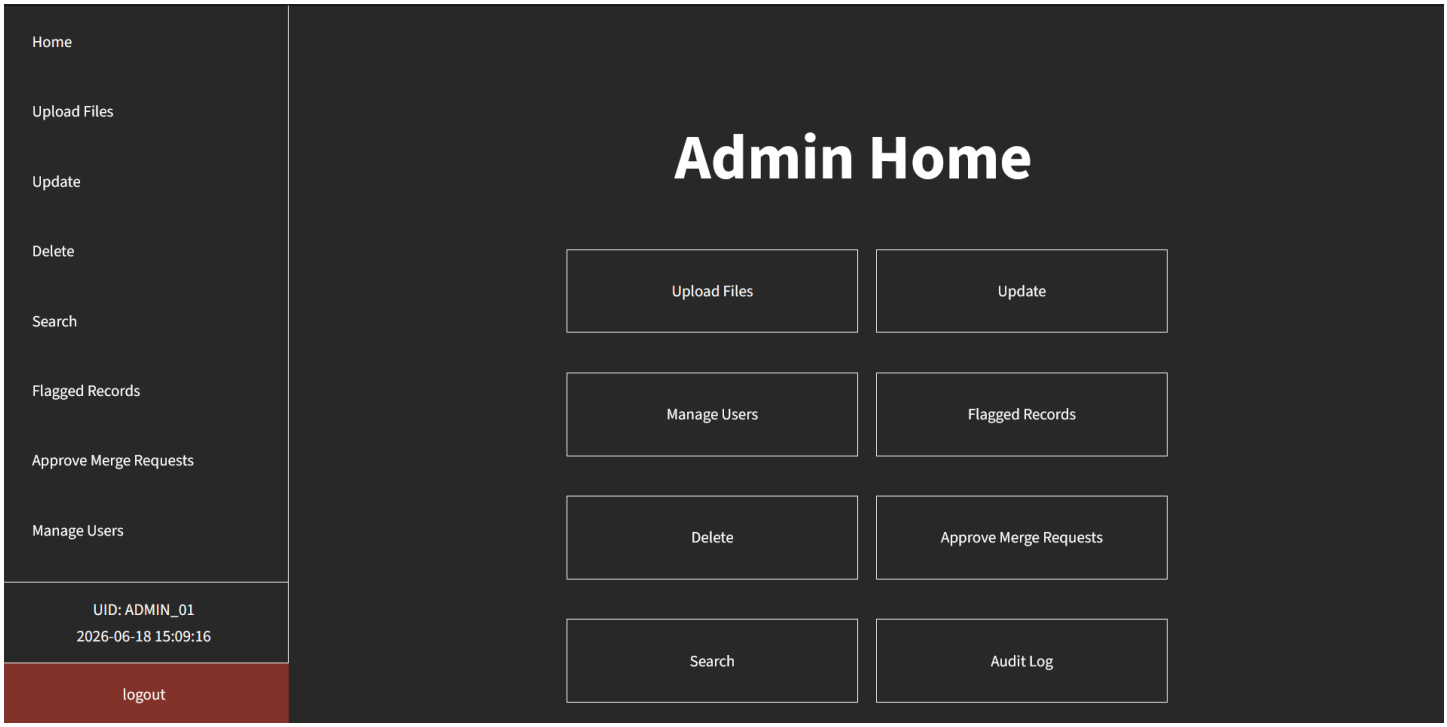

Figure 1. Wireframe of the Admin homepage in the GUI, offering certain capabilities for the admin to use.

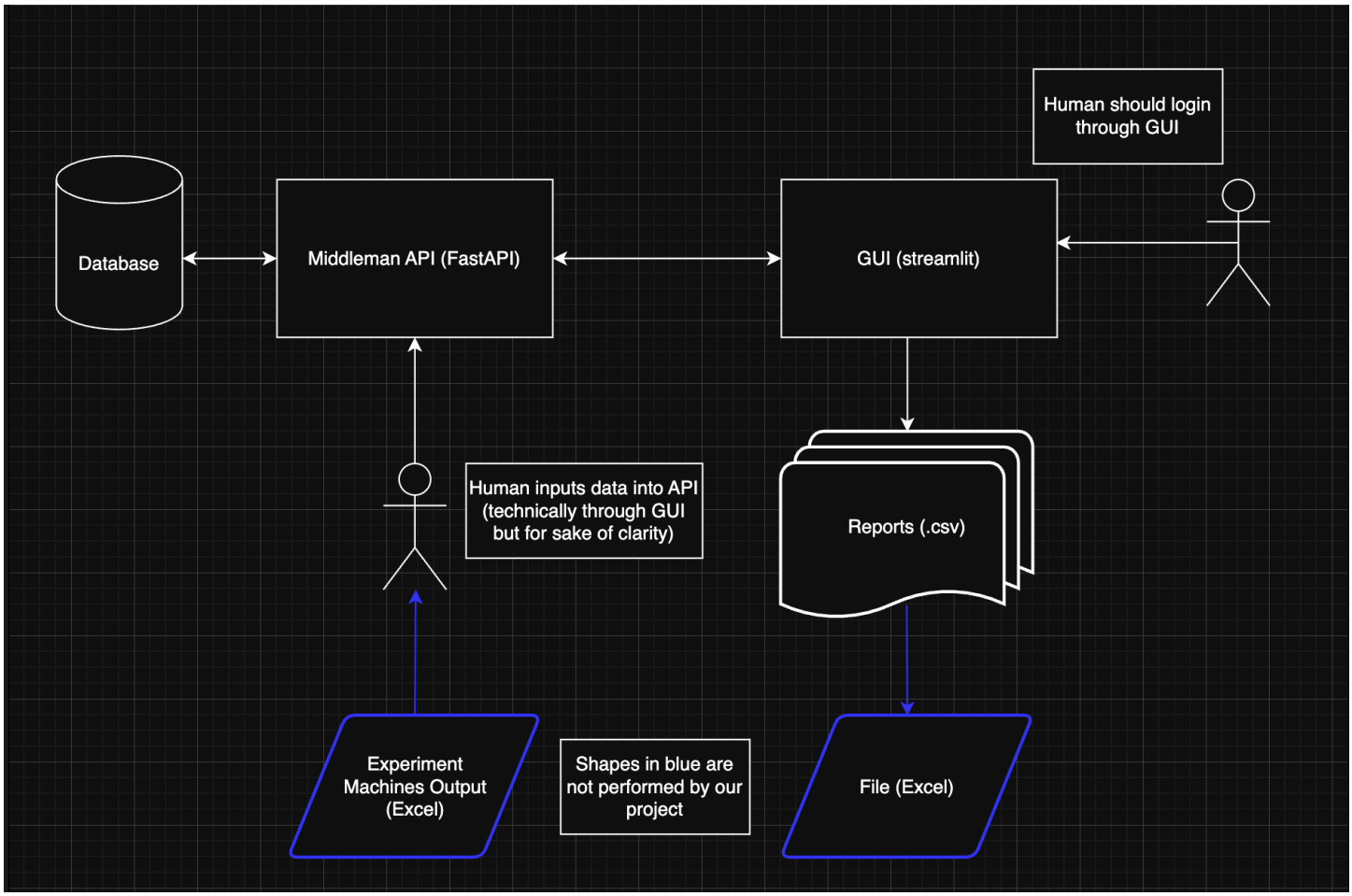

Figure 2. Architecture Diagram

The risk assessment and mitigation table summarizes the primary risks identified during the design phase of the project. Each asset was evaluated for potential vulnerabilities, associated threats, likelihood of occurrence, and potential impact on the system. A risk score was assigned to prioritize mitigation efforts. The team then identified appropriate treatment strategies and security controls to reduce the likelihood or impact of each threat. Particular emphasis was placed on protecting the scientific database, ensuring proper authentication and authorization, maintaining audit logs, and providing recovery mechanisms to prevent accidental data loss.

Asset	Vulnerability	Threat	Likelyhood	Impact	Risk
One centralized computer holding database	Lack of redundancy	Central Computer dies due to old age	Likely	Doomsday	20
SQL Database	Vulnerability	Database may be exposed to SQL injections	Likely	Catastrophic	16
FastApi backend server	Weak Authentication handling	Unauthorized api access	Medium	Catastrophic	12
Streamlit frontend application	Modern Web Front End	Older machines may not be able to access it	Medium	Serious	9
audit logging system	logs can be altered or deleted	someone can maliciously delete data to hide their tracks	Medium	Serious	9
file upload system	improper input handling / no input handling	can input malicious files or corrupted files	Highly Likely	Catastrophic	20
pending requests system	improper approval of requests	unauthorized changes in the scientific database	Highly Likely	Minor	5
session authentication tokens	token theft / session hijacking	unauthorized account access	Unlikely	Serious	3
HTTPS certificates/security configuration	misconfigured certificates	Unsecure encryption / man in the middle attacks	Low	Catastrophic	8
role permission system	improper role configuration	unauthorized data manipulation	Medium	Catastrophic	12
Roll back system / undo button	undo history or system may fail	inability to recover data that was undone	Likely	Serious	12

Treatment	Control	Mitigated Risk
Transfer	Host database in the cloud as well as locally	5
Reduce Likelihood	Paramaterized sql queries and fastapi validation	0
Reduce Likelihood	Use something like OAuth and securely store the API keys	5
Acceptance	Have the database be accessed on modern computers	1
Reduce Likelihood	Only allow admin users to edit logs	2
Reduce Impact	Fast API to handle incorrect file types	2
Transfer	Have more than one person confirm the requests	3
Reduce Likelihood	HTTPS and secure token expiration	2
Reduce Likelihood	proper certificate management and renewal	3
Reduce Likelihood	backend role enforcement through fastapi	0
Reduce Likelihood	Confirmation box to ensure deletion and a history kept of the database	5

Figure 3. Risk Assessment and Mitigation Table

The client provided a risk tolerance of 20. Currently, all of the mitigated risks are assessed as below this threshold, as shown in Figure 3.

The application workflow begins when a Hazen employee accesses the Streamlit graphical user interface and logs into the system. Login credentials are passed to the FastAPI backend, where authentication is performed, and the user's role is determined. Depending on the assigned role, the user is granted access to the appropriate functionality within the application. Administrators, project managers, and technicians each have different permissions that govern which actions they may perform within the system. After authentication, users may perform actions such as creating, updating, deleting, flagging, approving, or searching records. These requests are sent to the FastAPI backend, which validates user input and verifies that the user has the required permissions before allowing any interaction with the database. Once validated, the requested operation is performed on the database and the result is returned to the user through the graphical interface. The system then displays either a confirmation message or an error message depending on the outcome of the request. This workflow ensures that all database interactions are properly validated, authorized, and recorded before modifications are made to scientific research data, as reflected in Figure 4, shown below.

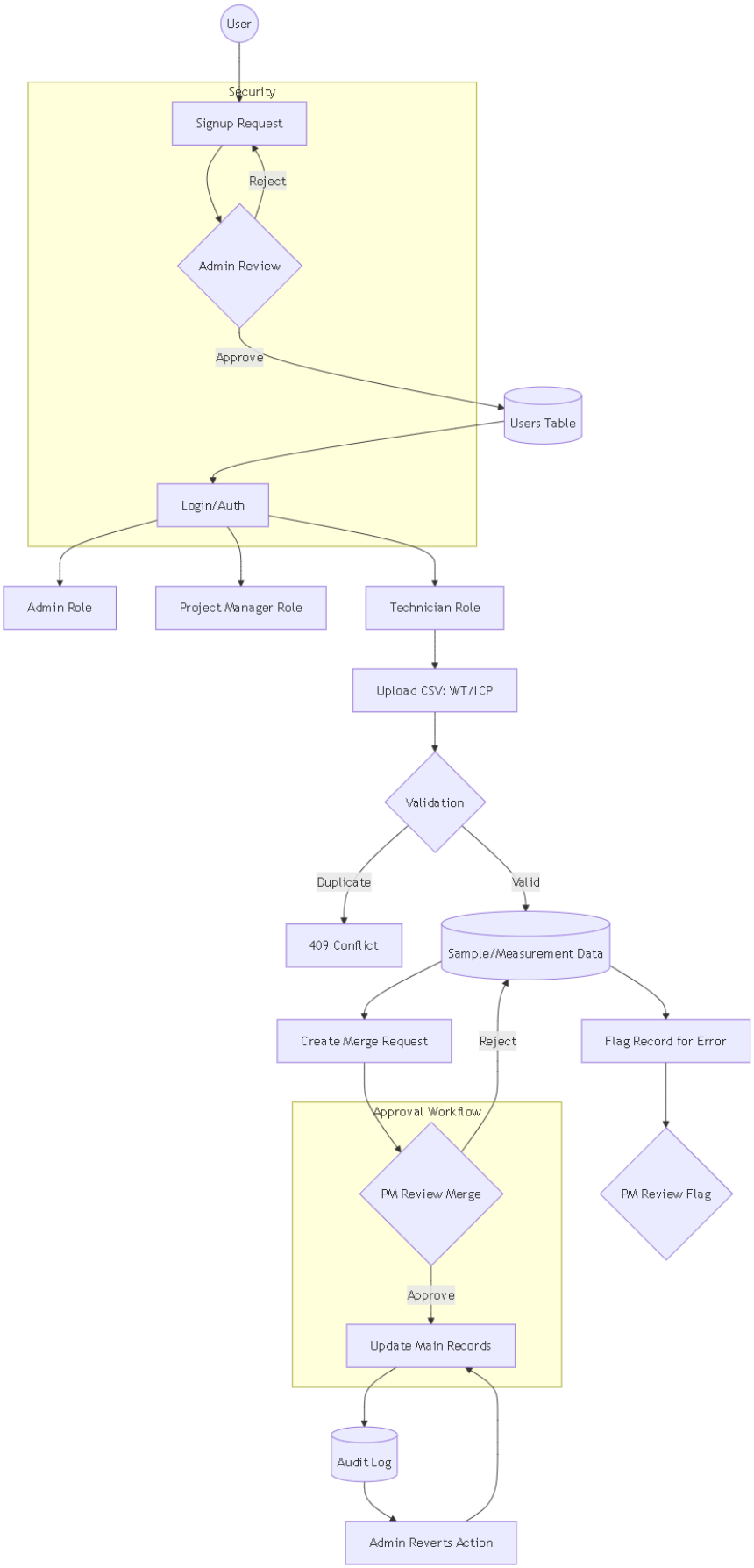

Figure 4. System User Signup Workflow

The database schema was designed based on the Entity Relationship Diagram (ERD) shown in Figure 5. The Users and Pending_Users tables present in the ERD are used for user authentication and RBAC. The remaining data tables were carefully constructed for efficient storage and retrieval while storing all the necessary data as specified by the client. The merge requests and flag records tables were stored separately rather than as fields of the Sample data to ensure temporary changes to the database would require administrator approval. The Sample_Element table serves as a bridge table between Element and Sample, establishing a many-to-many relationship in the case of experiments with different elements present. The Audit_Log table provides a historical record of all changes to the database, including uploads, updates, deletions, and merge requests.

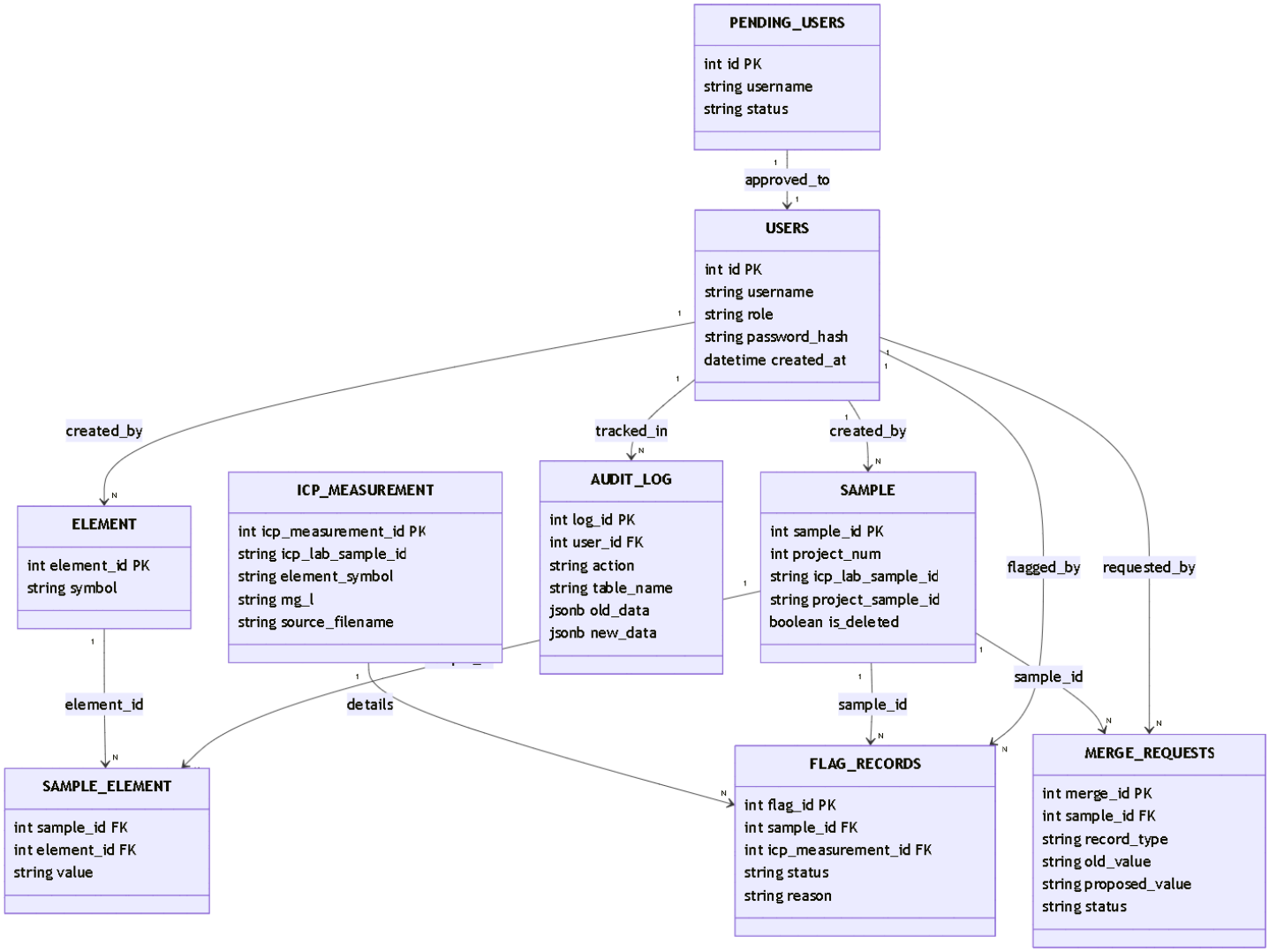

Figure 5. Table specifications for the Database Schema

VII. Software Test and Quality

The table below represents the testing procedures that the team undertook during the development phase of the project. Each functional requirement was addressed during the testing phase to ensure that all requirements were being fulfilled in the final product delivered to the client.

Requirement	Purpose of Test	Description of Test	Tools	Threshold for Acceptability	Edge Cases	Results
Dynamic Schema and Flexible Type Entry	Integrity of Database	Data from many different types will be attempted to be uploaded to the database	Testing Server; SQL Script	The database must have correct data at the end, without any erroneous deletes or omitted data. The data that is produced as an output must match the data present on the data sheet the client has provided.	Uploading the same data under different formats/different types	The only accepted file type is .csv. Additionally, the database was checked for duplicate sample_ids regarding WT%.
Project Filtering	Correctness of Outputs	Various inputs will be entered into the search field in the database	SQL Script	The desired filtered results must be outputted, without any changes to the database. The correct results will match the data with the inputted project ID on the data provided to the team by the client.	Capitalization; foreign language characters; extra spaces	Searches produced the correct outputs, matching the expected outputs present in the data provided by the client. Users can search by a variety of fields, which correctly filtered the project data.
Data Support	Correctness of Outputs	Various PDFs, images, and numeric data types will be uploaded to the database	Manually	The uploaded data must be correctly stored in the database, and it must be accessible through a search.	Blank images and PDFs; large file sizes; uploading nothing	The team was not able to implement PDF/image uploading within the specified timeframe. However, CSV file uploads using the standard

						format provided by the client were correctly uploaded and accessed.
Inheritance	Correctness of outputs	Various sub-projects will be uploaded to the database	Manually	The uploaded sub-project must be correlated underneath the main project type. The correctness will be compared against the sub-projects present in the data provided by the client.	Changing the order of sub-projects; foreign characters; searching for a project that does not exist	The team was unable to implement project-subproject inheritance within the specified timeframe.
Export Functionality	Correct Input Handling	Various data will be exported using the export button	VSCode Diff Log; Script to Compare	The exported data must be in the correct file format and store all the relevant data. There must not be any data added or lost in the process.	Exporting a large file; exporting a file with null data; Trying to export an empty file	Files were exported in CSV format with the correct data in the format the client desires.
RBAC (Admin)	Correct Data Modification	In an admin account, the user will attempt to delete data	Manually	The desired data to be deleted must be removed from the database. There must not be any extra data removed or unintended side effects.	Deleting something referenced by a foreign key; removing data that doesn't exist; trying to delete something twice	No unintended side effects have been seen from delete operations in the database. The front-end correctly deleted and displayed the new data.
RBAC (Admin)	Correct Role	In an admin	Manually	The new user	Setting up a	The signup

	Assignment	account, the admin will set up a new user account along with its role		must be able to log in to the correct role screen. They must not be able to do anything that their role would not permit them to do.	user with the same username; trying to reassign your own role	page prevented users with the same username from signing up. The user could only perform actions associated with their role. A user could reassign their own role, which caused them to see the correct GUI after signing out and signing back in.
RBAC (Admin)	Correct Password Resets	In an admin account, the user will approve a password reset from another user, creating a new password	Manually	The new user will attempt to log in. The old password should not work, and the new one must work.	The user presses forget, and then goes back and does their normal password, and then admin resets because of request; user pressing forget without a username; user pressing forget multiple times; different admin users try to reset same password	The password was correctly reset, and a user could not use a previous password to log in. If multiple requests were made, the user was informed that a request had already been made for that account.
RBAC (Manager)	Correct User Permissions	In a manager account, a user will not be able to delete data or reassign users	Manually	A user will log into a manager account. The GUI should not display the	A user's role is reassigned by the admin; users with the same username	The manager screen displayed the correct actions. When a user was

				option to delete data.	under different role groups	reassigned, the new screen correctly reflected their new role.
RBAC (Technician)	Correct User Permissions	In a technician user account, the user will only be able to add records, flag records, and make merge requests	Manually	A user will log into a technician account. The GUI should only display the options to add records, flag records, and make merge requests	A user's role is reassigned by an admin	The GUI correctly displayed the actions that the technician can perform, and did not display actions associated with the other roles. When a technician's role was reassigned, that account correctly reflected the change in the accompanying GUI screens.
Timestamps	Accuracy of Data Entry	Time data will be added/updated in the database in all 3 levels of user account	Clock	The timestamp in the database must match the actual time the user entered the data. Correctness can be ensured by comparing with the clock present in the sidebar.	Changing the local clock; Undoing an update/add	The display clock automatically updated. The audit log correctly reflected the time of all changes. If the local time was changed, all database operations still reflect Mountain Time Zone.
GUI (Search)	Correct Abstraction and Display	The user will search for various projects by ID	Manually	The data that is returned and displayed on the GUI must be the same data	Searching for a project that doesn't exist; searching for a duplicate project	The searching functionality worked as intended, returning the desired

				that is present in the database that would be returned from an equivalent SQL query.		results. Searching with null data returned error messages prompting the user to input real data for search.
GUI (Sidebar)	Correct Navigation	The user will attempt to navigate through the sidebar of the GUI	Manually	The user arrives at the destination they expected from the sidebar without having bugs.	Clicking on an item and clicking on another; clicking on the same button twice	The user arrived at the intended selected page. Clicking on a button multiple times or clicking quickly between buttons did not crash the app or cause unintended behavior.
GUI (Login Screen)	Correct Authentication	The user will attempt to sign in with a correct password and username. Then, the user will enter an incorrect password and correct username; and then an incorrect username and correct password	Manually	The user should only be able to enter the app if both the correct password and username are entered. The user should only see the page of their role.	Pressing submit with an empty username / password; case sensitivity; resetting a password	The user could only sign in with both the correct username and password. Trying to submit with empty fields prompted the user to enter responses into the fields.
GUI	Web-accessible	The user will attempt to open the web-based GUI on several	Safari, Chrome, Firefox, Edge, Mac Computer, Windows	The web-app should be accessible from all different browsers and	Using outdated OS or browsers	The web-GUI was accessible on Safari, Edge, Firefox, and Chrome from a

		browsers across several different operating systems	Computer	operating systems		Windows-based machine. It was also accessible from a computer running MacOS. A Linux-based computer has not yet been tested.
GUI	Intuitiveness / Ease of Use	Several tests will have users try and use the app without the team providing instructions on how to use it to see if it is intuitively designed	Test subjects	The app must be considered “generally intuitive” from the test subjects who tried it out.	Users who have limited app experience; users with disabilities; users from different cultures	The GUI was tested and demonstrated to the client as was determined to be intuitive and easy to use.

Table 3: Tests and Results

VIII. Project Ethical Considerations

Designing responsible technology requires looking beyond functionality. The following examines the core ethical challenges that were considered in this project:

1. Data Integrity and Manipulation

The project allows the right users to easily modify, delete, or inject data if strict access controls and logs are not implemented. This can make test results or processes seem more efficient or more compliant than they actually are. Falsifying data to artificially inflate asset value directly violates ACM 1.3 (Be Honest and Trustworthy) [1] and IEEE 1.1 (Public Welfare and Safety) [2], requiring strict role-based access control (RBAC) and data validation protocols.

2. Environmental Responsibility/Public Safety

A software system for a company doing industrial and chemical research processing must treat environmental safety metrics, such as runoff and chemical contaminants, with the highest priority. Failing to transparently report critical environmental hazards to shield an entity from liability or to artificially inflate value violates ACM 1.2 (Avoid Harm) [1] and IEEE 1.1 (Public Welfare and Safety) [2]. Reporting features that ensure transparency and resist censorship of hazardous metrics are necessary.

3. Conflicts of Interest and Data Security

Proprietary research data, like that at Hazen Research, is highly valuable. Thus, engineers or administrators could face bribery and coercion to grant unauthorized access to database backends, leak formulas and proprietary data, or ignore when security protocols are breached. Failing to secure the database backend against external breaches or internal

corruption violates ACM 1.7 (Honor Confidentiality) [1] and accepting incentives to bypass security or overlook data leaks violates IEEE 1.4 [2], which strictly prohibits bribery and corruption in all engineering practices.

4. Proprietary Models and Property

Integrating external models, open-source projects, and schemas without attribution violates ACM 1.5 (Respect Creative Work) [1] and IEEE 1.3 (Honest Representation of Claims) [2]. Software engineers must trace the origins of all third-party code and applications used in the project, ensuring that the creators are properly credited and their intellectual property rights are respected.

IX. Project Completion Status

The product fulfilled all the points laid out in the definition of done. The software allows a user to upload a CSV file as an input, from which wt% and ICP data are correctly inserted into a local relational database. This data is filterable on elements present and sample ID, along with many other fields. Additionally, all of these features are present in a web-based GUI with RBAC.

The team was also able to complete several of the stretch goals laid out at the inception of the project. These features include the following:

1. Update/Delete Capabilities - An admin user can update and delete data within the database from the GUI
2. Audit Log - A detailed audit of every change to the database, stamped with the time and the user, is visible from the admin user page.
3. Merge Requests - A technician user can “edit” the data, creating a merge request that an admin user either approves or rejects.
4. Flagged Records - A technician user can flag a record with a comment, to which the admin user can delete or manually edit the data based on the comment.
5. Auto Logoff - All users will be logged off from the application if they have been inactive for more than 30 minutes.

The performance of the features present within the software meets the team’s non-functional requirements. The team required that querying data through searching would return results in less than 3 seconds, which has been met and tested. Additionally, all of the features have been tested rigorously, particularly the RBAC, with few errors present. The most significant remaining issue was that the front-end, Streamlit, is occasionally unstable or slow in displaying content. This seems to be a quirk of Streamlit, and although frustrating, it is rare enough that the team is pleased with the final software.

There were several features that the team was unable to implement. These features include the following:

1. Sample Progress Tracking - The client provided a stretch goal to track a sample across several buildings where data collection would occur, marking in the database which buildings had been visited. The client desired that a progress bar would be visible for each sample, reflecting which buildings still required to be visited in order to collect all the necessary data.
2. Inheritance/Subprojects - The client voiced a desire to have projects and subprojects be correlated, and subprojects to be accessible from parent projects by navigating through the parent project. The client also provided a stretch goal of representing this inheritance structure in a tree-like diagram.
3. Flexible type Entry - The client requested that the software could permit many different types of data to be uploaded to the database, such as PDFs, Excel files, CSV, and images. Presently, the software can only upload CSV files of a specific format to the database.

X. Future Work

There is a significant amount of possible future work related to this project. Currently, the team’s software permits uploading, querying, and storage of data from ICP testing. ICP testing is only one of around 60 tests that are frequently performed at Hazen Research. The client’s hope would be that the database would be able to support

uploading and querying of many more testing types in the future. Implementation of these tests would likely vary widely from test to test, but a similar structure would be present that the team utilized for ICP testing. There would be a significant amount of knowledge required for each additional test-type added to the software, particularly with understanding what data needs to go into the database from each kind of test. The team's rough estimate was that adding each additional test-type could take anywhere from 15-30 hours, depending on the complexity of the test. Much more frequent communication with the client would also be necessary in order to understand the nature of each additional test-type being added.

Another source of future work is the features the team was unable to implement in the time frame. In order to implement these using the current structure, someone would need to understand Streamlit, FastAPI, and PostgreSQL quite well, which the team estimates would take around a week or two to learn without prior knowledge. The team estimates that the sample progress tracking could be implemented in 3-5 days, inheritance in 2-4 days, and flexible type entry in 1-3 days. In total, assuming limited prior knowledge, the learning and implementation of these features could take 2-3 weeks.

Another possibility for future work would be to rework the front-end framework currently used in the project. For this project, Streamlit was utilized for its convenience and straightforward learning curve. However, Streamlit has many limitations in developing production-level software. The team encountered these issues and found workarounds, but it would be much better to rework the full front-end in a more dedicated framework, such as React or Angular. This would require significant familiarity and proficiency with web-development frameworks or significant time and effort spent learning these frameworks. It is estimated that the knowledge required to learn one of these frameworks and rework the entire front-end could take as long as 4-5 weeks, a significant amount of time and effort.

A final possibility for additional work is implementation of the database and web application in the cloud. The current software utilizes a local database and a locally-hosted website. The client has some desire for expanding this to a cloud-based database with a web app hosted on a locally hosted server accessible by users within the company. This would require a significant amount of networking knowledge, particularly with security and HTTPS certification, along with frequent cooperation with the client's IT department. It is likely that to implement this, an NDA would have to be signed as proprietary network architecture data would be necessary in order to correctly implement the solution. It is estimated that this would take around 2-3 weeks to correctly implement for a group with limited prior networking experience. However, this work is the least feasible to implement, as it would require frequent testing at the client's building on the client's network, a significant barrier to development.

XI. Lessons Learned

Python-based frontend libraries, while simple to implement, offered limited functionality compared to modern web frameworks. The team selected Streamlit as the selected frontend framework due to the lack of experience in web development using more advanced frameworks. Streamlit was highly comprehensive to learn and to implement, but the further into development the team progressed, the more its limiting factors became apparent. In the end, the team needed to do numerous CSS injections to produce the desired results. Additionally, to implement some of the stretch goals, using Streamlit would not have been a viable framework. For the scope of a five-week project, the team likely made the right choice in selecting a development framework given prior experience, but using React or Angular would be the right choice for a longer development cycle.

FastAPI is a highly effective, powerful API framework. FastAPI was selected after the team researched several options, determining it to be the best choice for the scope of the project. Accordingly, as the team worked with FastAPI more, it quickly became apparent that this tool was the best choice on both the large and small scale. It has extensive documentation and capabilities, while being extremely efficient to get a functional RESTful API up and running using this framework.

Understanding what data is necessary and what is not is extremely vital when creating database systems. In this project, the team was given a large amount of dummy data, much of which was not strictly necessary for the final

implementation. It took several weeks and meetings with the client to determine which data was strictly necessary for the database and how it correlated with other data points across the dataset. Determining this was crucial for correctly implementing the database structure, and once the team was able to identify the necessary data, development speed was greatly increased.

Project backlogs are remarkably effective in organizing and assigning work for a team project. The team used the backlog very little in the first few sprints, but once full development had begun, the backlog was instrumental in staying on track. Use of the backlog permitted the team to collaborate and communicate effectively during development. Additionally, the backlog helped execute daily standups, as it permitted the team to point to work that had been completed or work that still needed to be done.

Being able to concisely and clearly communicate the features and development of a product is a crucial skill. The team learned this especially in the creation of the two large presentations where both specificity and brevity were required. Learning how to effectively communicate and present the work the team had completed was invaluable learning for all team members.

XII. Acknowledgments

We would like to express our sincere gratitude to Hazen Research & Development for providing us with the opportunity to work on this project and gain valuable real-world consulting experience. In particular, we would like to thank Viktor Lefevre, our client, for his guidance, expertise, and continued support throughout the project. His insights and feedback were instrumental in helping us understand the project's objectives which in turn helped us develop effective solutions. We would also like to extend our appreciation to Dr. Phillip Romig III for his mentorship, encouragement, and academic guidance during the consulting process. His expertise and dedication helped ensure the success of our work and enriched our learning experience. We are grateful to all those who contributed their time and knowledge to make this project both successful and rewarding. Additionally, we would like to thank the other professors and instructors involved in the course for sharing their knowledge, providing valuable feedback, and helping us develop the professional and technical skills that contributed to the completion of this project.

XIII. Team Profile

Alex Rogers

Sophomore

Computer Science

Hometown: Littleton, CO

Work Experience: Subway, Restaurant Server, Intramural Referee

Clubs: AI/ML, ACM, Club Volleyball, SASE

Andy Nguyen

Junior

Computer Science

Hometown: Aurora, CO

Work Experience: Car Repairman

Clubs: Society of Asian Scientists and Engineers

Duy Nguyen

Junior

Computer Science

Hometown: Denver, CO

Work Experience: Developer for Summit Assembly, Restaurant Host

Jeremy Gillett

Senior

Computer Science

Hometown: Longmont, CO

Work Experience: Restaurant Host, Missionary

Clubs: Mines Catholic, Mines Hacky Sack Club

Hobbies: Guitar, Running, Basketball

References

[1] Association for Computing Machinery, “ACM Code of Ethics and Professional Conduct,” ACM, New York, NY, USA, 2018. [Online]. Available: https://www.acm.org/code-of-ethics

[2] IEEE, “IEEE Code of Ethics,” IEEE, Piscataway, NJ, USA, 2020. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-14.html

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
XRD - X-ray Diffraction	*By using X-Ray beams on a crystalline sample, the light beams diffract off electrons and produce patterns indicating which elements are present*
ICP – Ion Coupled Plasma	*Injection of ionized argon gas into liquidized rock sample to observe atoms and ions*
AA – Atomic Absorption Spectroscopy	*Setting liquidized rock sample on fire, tracking how much light the samples absorb to determine concentrations of certain elements in sample*
wt%	*Measurement of concentration of a substance in a mixture.* *Formula:* $\frac{mass\ of\ component}{total\ mass} \times 100$
RBAC - Role Based Access Control	*A security model that restricts system access based on an user's role*
GUI (Graphical User Interface)	*A visual interface that allows users to interact with software through buttons, menus, forms, and other graphical elements*
API (Application Programming Interface)	*A set of rules and endpoints that permit software components to connect through a middleman*
Streamlit	*A Python-based library used to develop web apps*
FastAPI	*A Python framework used to create web APIs that process requests between the user interface and the database.*
PostgreSQL	*An open-source relational database framework used for storage, retrieval, and modification of data*
SQL (Structured Query Language)	*A programming language used to create, retrieve, update, and manage information stored in relational databases*
CSV (Comma-Separated Values)	*A text-based file format that stores data where values are separated by commas*
RESTful API	*An API that follows REST architectural principles and uses standard HTTP protocols.*
HTTP (Hypertext Transfer Protocol)	*The standard protocol used for communication between web browsers and web servers.*
HTTPS (Hypertext Transfer Protocol Secure)	*An encrypted version of HTTP that protects data transmitted between users and web applications.*
Pydantic	*A Python library used for validating and managing data*

Audit Log	*A record within a system keeping track of what updates to the database have occurred, noting at what time these updates occurred and who performed the updates*
SQL Injection	*A security attack where malicious SQL commands are inserted into user inputs to destructively modify a database*