Colorado School of Mines
Department of Computer Science

# CSCI 370 Final Report

MINES COMPASS

Prepared By:
Brandon Barton
Roman Downie

Revised
June 15, 2022

Advanced Software Engineering, Summer, 2022

Supervised By:
Professor Kathleen Kelly

A field session project submitted to the Department of Computer Science in partial fulfillment of the requirements for CSCI 370 field session.

Table 1: Revision History Table

| Revision | Date | Comments |
|---|---|---|
| New | May 17th | **Completed sections:**<br>Title page<br>Built template<br>Table of contents<br>References<br>Appendix |
| Rev-1 | May 18th | **Completed sections:**<br>Functional requirements<br>Team profile |
| Rev-2 | May 19th | **Completed sections:**<br>Non-functional requirements<br>Risks<br>**Updated Sections:**<br>Functional requirements |
| Rev-3 | May 20th | **Completed sections:**<br>DoD<br>**Updated Sections:**<br>Non-functional requirements |
| Rev-4 | May 29th | **Completed sections:**<br>Design |
| Rev-5 | June 5th | **Completed sections:**<br>Software Test and Quality<br>**Updated Sections:**<br>DoD |
| Rev-6 | June 14th | **Completed sections:**<br>Results<br>Future work<br>Lessoned learned<br>**Updated Sections:**<br>Software Test and Quality |
| Rev-7 | June 15th | **Updated Sections:**<br>All |

# Table of Contents

# Introduction

Every fall and spring, students build schedules for the next semester of their academic careers. In parallel, professors plan for new courses, curriculum changes, and make changes to existing course material for the latest offering. Due to these changes, among other things, students may run into problems planning their schedules.

Students may run into uncertainty and delays while making sure they are taking the right prerequisites, co-requisites, free electives, etc. In the worst case, an unplanned degree may result in a student having to enroll for an additional year which carries significant financial and social costs.

Along the same line, professors, faculty committees, and university registrars face many challenges while making curriculum changes to existing degree programs. These challenges include updating course pre and co-requisites, updating the course catalog, all while maintaining legacy course information from previous years.

The goal of this project is to reduce the headaches of planning schedules for students, assist academic advising, and help institutions such as the Colorado School of Mines get students to graduation on time through many changes in program requirements. We hope to provide a seamless software product for students to plan multiple semesters of courses, professors to make updates, and universities to maintain the latest catalog editions in a useful format.

This project is building off of an existing code base that we made in our database management class. The previous code existed as a proof of concept. Many of the components of the website as well as the schema for the database were already created when starting this project. A sizeable amount of the work on this project is to rework some of that old code to be cleaner and more modular.

# Requirements

## Functional Requirements

- Login page
  - Remove the existing sign up page and replace it with an SSO login page using DUO ideally.
- Schedule workspace
  - Allow students to build a schedule by adding semesters, and courses. The previous courses that a student has taken should also be displayed in the schedule workspace. Students should be able to determine whether a planned schedule is valid, or if a degree requirement is missing in their plan.
- Explore page
  - The explore page will allow users to make changes to their schedules based on feedback from the program. One such example would be highlighting a class and seeing what other classes could replace that class in the schedule. Another functionality would be to browse the catalog to see other classes the institution offers. Courses may be searched by which requirements they fulfill in a specific major, by name, department, or subject.
- Account page
  - The account page should display valuable user information such as,
    1. Name
    2. Class status based on credit
    3. Major(s)
    4. Minor(s)
    5. Planned graduation date
    6. CWID
    7. Academic advisor(s)
    8. Degree progress
- Administrator or faculty page
  - The administrator or faculty page will provide faculty members for administrators the ability to,

1. Add courses
2. Update course information
3. Delete courses
4. Change degree requirements (e.g. Replacing CSCI 261 with CSCI 200)

## Non-functional Requirements

- The above functional requirements are supported by non-functional requirements (also known as quality requirement), which impose constraints or design or implementation
- Examples include performance requirements, cost constraints, security or reliability.
- Login page
  - SSO and DUO should provide sufficient and necessary security for user login.
- Schedule workspace
  - Adding courses, semesters should be accessible and quick on the order of milliseconds(ms). Students should be able to easily choose from a drop-down menu of courses.
  - Validation of schedules should be on the order of ms, with guaranteed correctness for the current catalog.
- Explore page
  - A similar drop-down menu to the schedule workspace should provide students with the ability to quickly add courses
  - Students should get fast query results on questions such as,
    1. What suggested courses should I take to add a minor in computer science?
    2. What suggested courses are equivalent to MATH201?
    3. What courses satisfy the mid-level HASS requirement?
- Account page
  - The account page should only display the user's information in a secure manner. No other user's information should be accessed by any other user, which maintains user privacy.
  - If super-user abilities are granted (e.g. academic advisors), the super-user should be able to access their advisee's schedules and account information.
- Administrator or faculty page
  - The highest level of database security should be addressed to avoid any data insertion, leaks, or other malicious activity.
  - Database API will check for valid and safe queries
  - Course updates should only be done by super-users with permission to course updates and changes.

## Definition of Done

For the scope of this project, we have defined a number of critical aspects of the project that we wanted to be finished and functional by the end of field session.

These critical features include,

1. Long-term database solution
2. Reliable data for one major's requirements according to the 2022-23 Mines course catalog
3. Hosted website
4. Single sign-on capabilities

With these features, we hope to launch our product to one user in the Mines community, particularly someone in an academic advising role. While launching is a step that will occur outside of the scope of this course, our efforts throughout field session have placed the team in an ideal position to then launch our product shortly after the course.

To ensure that our team came out of the course with a functional product, we ran a comprehensive series of user tests throughout the course, and especially in weeks 3  4 leading up to the final presentation.

# Risks

## Technology risks

- Database
  - Risk of malicious software insertion in the course update web page.
  - Curriculum changes should follow a multi-step approval process outlined in the Functional Requirements section.
  - Data rights may be an issue when dealing with sensitive information. Ownership and storage of the database should be addressed.
  - Queries on the database should be quick and efficient.
- Data and User information
  - Data needs to be secure to prevent user data leaks.
  - User data should not be accessed by any users without the correct permissions
  - In the long term, there could be issues with the old data mixing with the new data. (i.e. renaming of classes, restructuring of departments, moving courses between departments, etc.)
- Website software failures
  - The database interface may need to be updated along with the database.
  - Trying to access something that doesn't exist. For example, a student whose account has been deleted.

## Skills risks

- Setting up a long-term database is a challenge because members of the team do not have much experience in this field. To address this risk, we plan to devote a large portion of the class to make sure our database is set up correctly and securely.
- Members of the team are novices in web development.
- Given the fast pace nature of the course, hitting a roadblock may result in delaying the project timeline. We plan to set realistic goals to account for these potential delays.

# Design

## System architecture

The high-level overview of the software architecture is conveyed in Figure 1. To address the requirements of students, faculty, and administrators, our solution delivers a website application. The web-app consists of multiple pages, each tailored for a specific need of the user. Each page is described in detail in the process flow below.

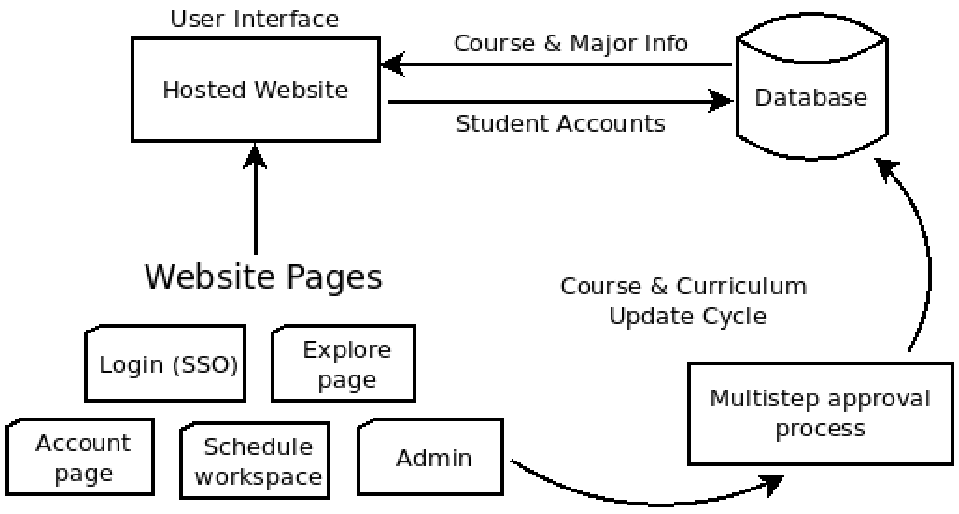


Figure 1: High level overview of the system architecture.

To make the website functional with the Mines course catalog information, the website interfaces primarily with an external database. Our choice of a PSQL database allows for the back-end software of the website to pull course information and logic from the database which houses all course, major, minor, and other relational data displayed in the database entity relationship diagram (Figure 3). Porting the database also allows information to flow into the database, such as user information (e.g. planned schedules, major or minor information).

## Process flow

As shown in Figure 1, we provide general overview of the system architecture. Here we provide detailed technical design descriptions of each webpage, and the corresponding user flow. In Figure 3, users begin at the log-in page if they have not created an account yet. As discussed in the Results chapter, single sign on capabilities with DUO mobile were not a feature added over the course of field session. Therefore, the current working implementation simply provides a sign-up and login pages for users to access their accounts. Once signed in, users are directed to the home page, also known as the schedule workspace. The schedule workspace is where users may view saved schedules statically. The explore page is one of the main productions of this course, which is the main application of the software product. The explore page addresses the primary need of students who are able to create dynamic schedule plans. Users may also view their personal account information on the account page.

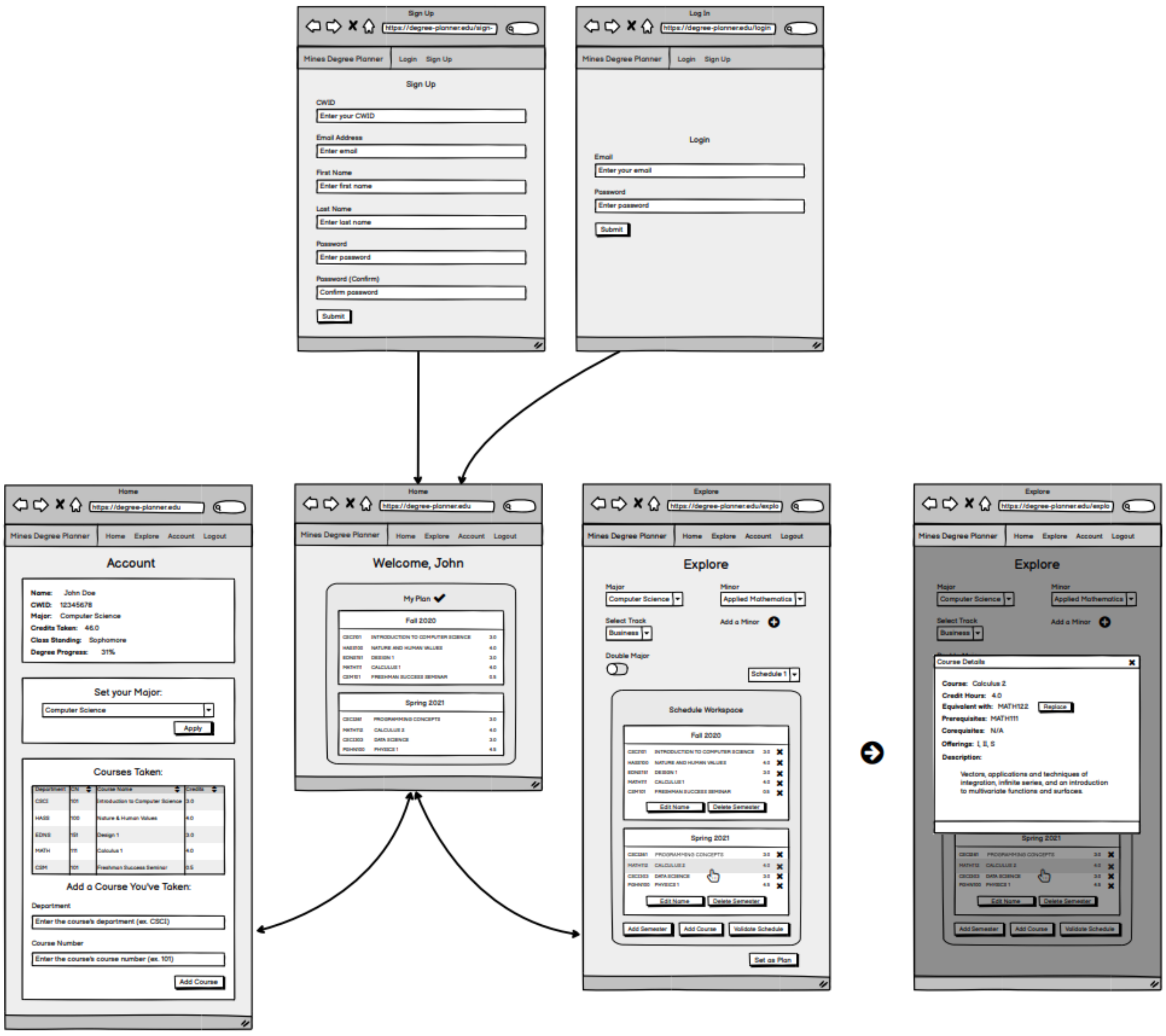


Figure 2: Process flow of webpages in wire diagram format. Edges and arrows represent user flow to each page.

## Database design

The database primarily contains course information (e.g. prerequisites, course description), and student (user) information. Captured in Figure 2, this minimal database model allows students to create schedules, with the framework we have generated in our software. After making a schedule, students can then test their plans against the major requirements, and ask business questions to understand what additional courses they should take to reach their goals.

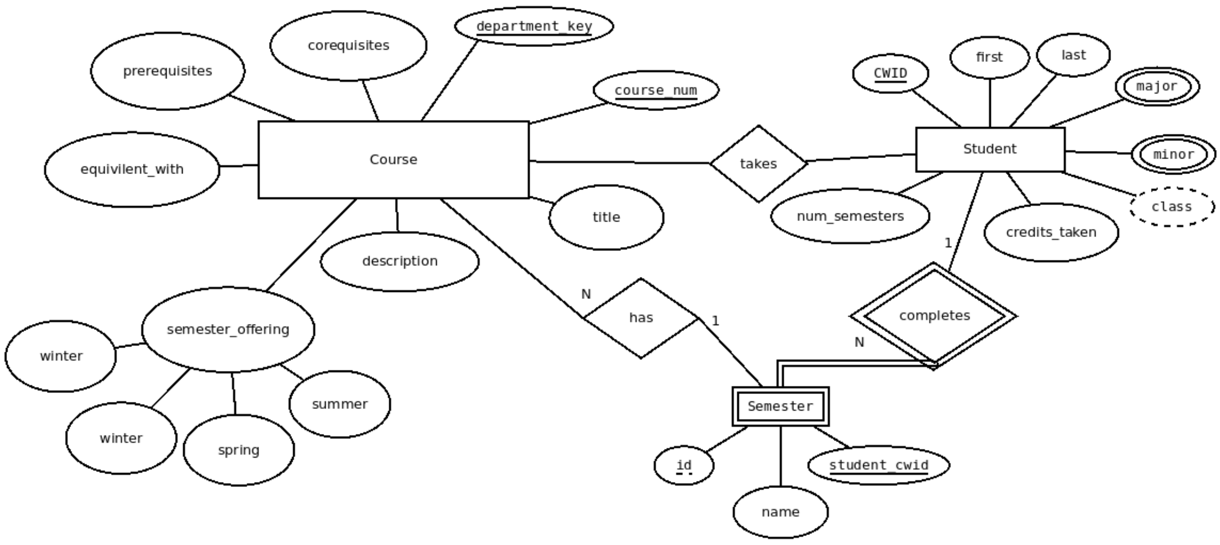


Figure 3: Database entity relationship design.

This design allowed for the team to build the software which is the main contribution of the project. Most importantly, the database port to our software is easily exchanged, ensuring the ability to scale and provide continuous updates to the curriculum requirements.

## Wire diagrams

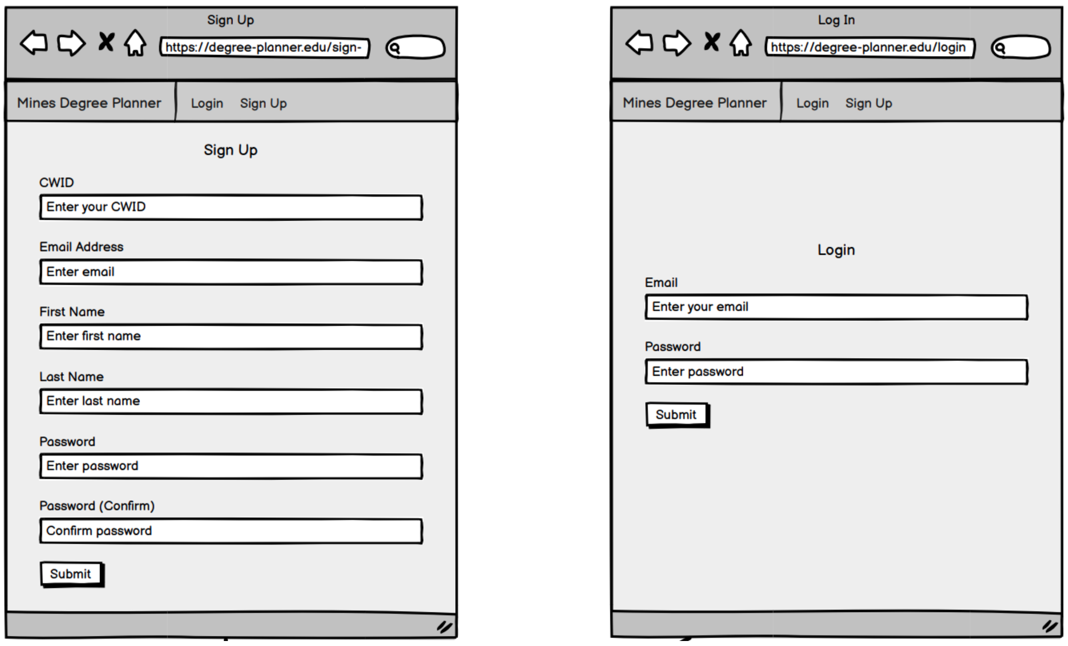
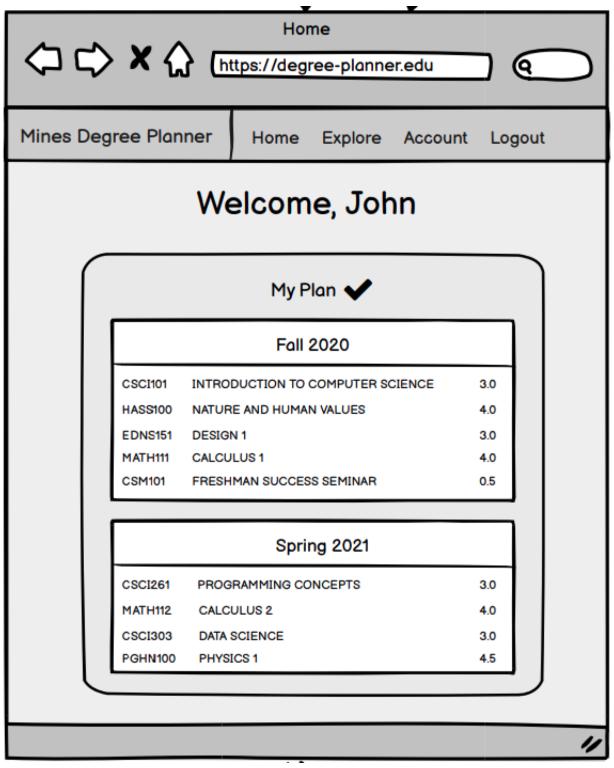


Figure 4: User sign-up (left) and login page (right)

In Figure 2, the process flow from a user perspective is provided which shows each page of the website. In this section, we provide closer looks at each of the web-pages, and discuss the capabilities of each.

Starting from the login page, this is the first point of contact between users and the application. If the user does not have an account made previously, they are directed to the sign-up page (Figure 4). New users may create an account with student information included their name, email, and other pertinent user information. If users have an existing account, they may access their accounts directly from the login page. Once logged in to their account, users may proceed to other pages as shown in Figures 5 and 6.



Welcome, John

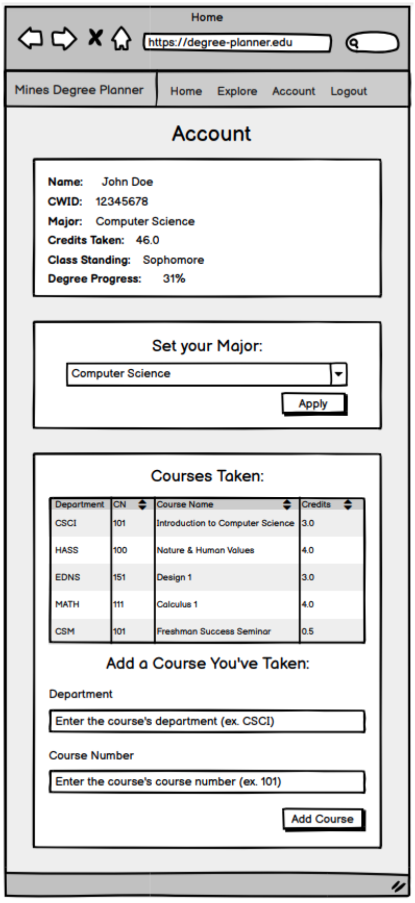
My Plan ✓

Fall 2020

|  |  |  |
|---|---|---|
| CSCI101 | INTRODUCTION TO COMPUTER SCIENCE | 3.0 |
| HASS100 | NATURE AND HUMAN VALUES | 4.0 |
| EDNS151 | DESIGN 1 | 3.0 |
| MATH111 | CALCULUS 1 | 4.0 |
| CSM101 | FRESHMAN SUCCESS SEMINAR | 0.5 |

Spring 2021

|  |  |  |
|---|---|---|
| CSCI261 | PROGRAMMING CONCEPTS | 3.0 |
| MATH112 | CALCULUS 2 | 4.0 |
| CSCI303 | DATA SCIENCE | 3.0 |
| PGHN100 | PHYSICS 1 | 4.5 |



Account

**Name:** John Doe
**CWID:** 12345678
**Major:** Computer Science
**Credits Taken:** 46.0
**Class Standing:** Sophomore
**Degree Progress:** 31%

Set your Major:

Computer Science

Apply

Courses Taken:

| Department | CN | Course Name | Credits |
|---|---|---|---|
| CSCI | 101 | Introduction to Computer Science | 3.0 |
| HASS | 100 | Nature & Human Values | 4.0 |
| EDNS | 151 | Design 1 | 3.0 |
| MATH | 111 | Calculus 1 | 4.0 |
| CSM | 101 | Freshman Success Seminar | 0.5 |

Add a Course You've Taken:

Department: Enter the course's department (ex. CSCI)

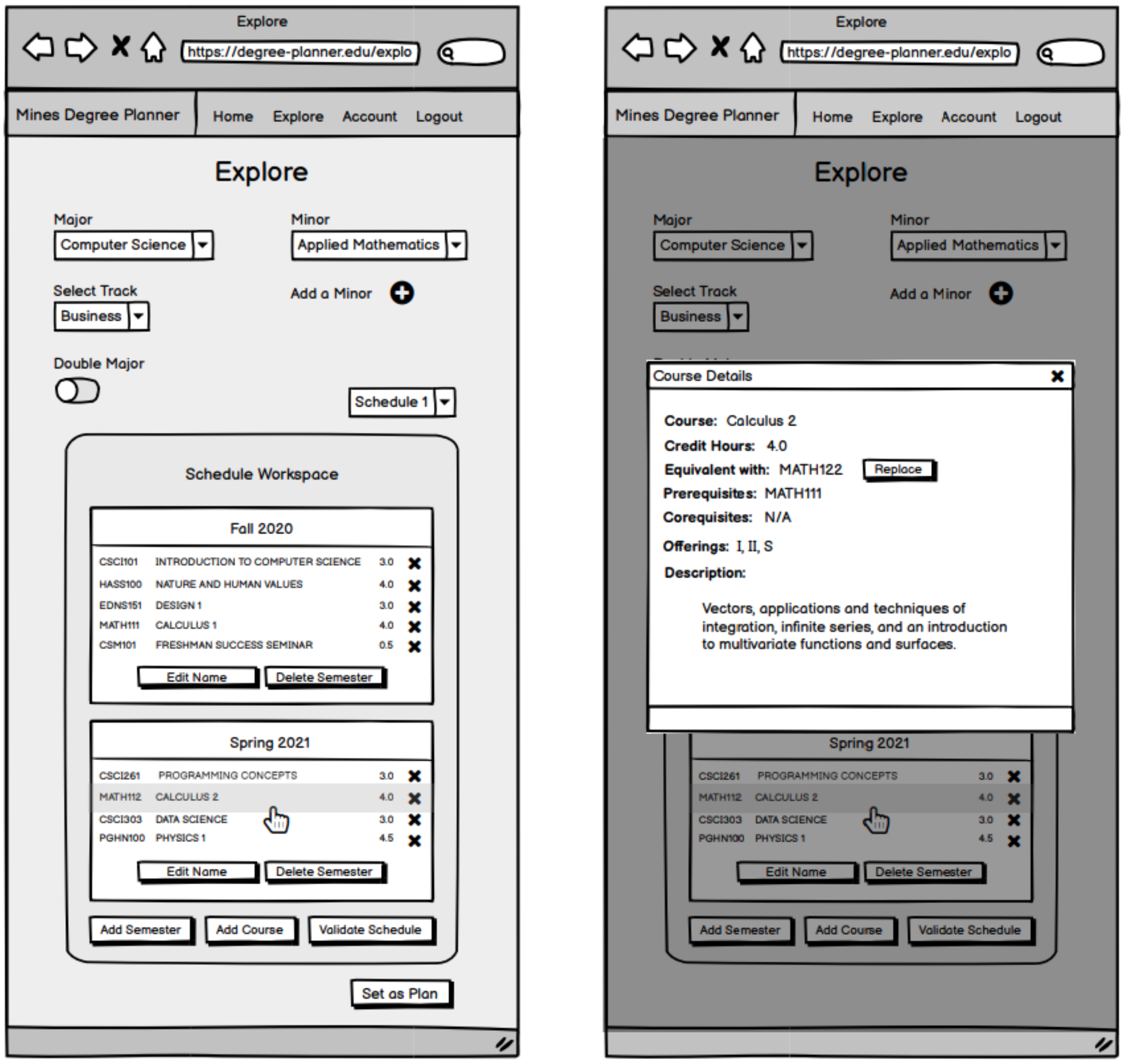
Course Number: Enter the course's course number (ex. 101)

Add Course

Figure 5: (left) User home page (aka schedule workspace), and (right) account page.

After the user has passed authentication on the login page, the user will land on the home page. The home page is also known as the schedule workspace in Figure 1. On the home page, users may access previously saved schedules, which may be loaded into "my plan" panel on the website. The home page of the website is where students may review schedules in a clean format, distinct from the explore page in Figure 6 which allows for more dynamic planning.

Users may also view account settings on the account page in Figure 5 (right), which contains user information and course history. As the website stands, users may enter previous course information manually, which is then saved in the database and populated upon login. The account page contains sufficient information, however there are various opportunities for future automation discussed in the chapter on Future Work.



# Explore

Major: Computer Science

Minor: Applied Mathematics

Select Track: Business

Add a Minor

Double Major

Schedule 1

Fall 2020

| | | |
|---|---|---|
| CSCI101 | INTRODUCTION TO COMPUTER SCIENCE | 3.0 |
| HASS100 | NATURE AND HUMAN VALUES | 4.0 |
| EDNS151 | DESIGN 1 | 3.0 |
| MATH111 | CALCULUS 1 | 4.0 |
| CSM101 | FRESHMAN SUCCESS SEMINAR | 0.5 |

Spring 2021

| | | |
|---|---|---|
| CSCI261 | PROGRAMMING CONCEPTS | 3.0 |
| MATH112 | CALCULUS 2 | 4.0 |
| CSCI303 | DATA SCIENCE | 3.0 |
| PGHN100 | PHYSICS 1 | 4.5 |

Course Details

**Course:** Calculus 2
**Credit Hours:** 4.0
**Equivalent with:** MATH122
**Prerequisites:** MATH111
**Corequisites:** N/A
**Offerings:** I, II, S
**Description:**

Vectors, applications and techniques of integration, infinite series, and an introduction to multivariate functions and surfaces.

Figure 6: Explore page.(Left) base page, and (right) pop up course information.

The explore page as shown in Figure 6 is the most dynamic aspect of the application that we provide. The explore page provides students with the capabilities of exploring tracks of study, including double majors and additional minors.

In a pragmatic approach, student may test the practicallity of adding additional degrees, certificates, and minors by adding courses to a planned schedule. Students may add an arbitrary number of semesters to their planned schedules which ensures the dynamic approach of the appplication is maintained. As many students take gap-years, work on co-op, or take internships during fall and spring semesters, the explore page allows students to plan for these gaps in course registration.

As students add courses to their schedule, they may view information by clicking on a course which displays a pop-up shown in Figure 6 (right). The pop-up shows all course information which is pulled directly from the PSQL database.

These technical designs provide student users with the capability of dynamic schedule planning over multiple years of study. The practicalities and considerations of implementing these plans are discussed in the sections that follow.

# Software Test and Quality

## Unit testing

Our team has agreed to follow the standard practice of test-driven development (TDD) for all software development in this project. TDD will address the validation aspects of our software by demonstrating that the code performs the intended task correctly.

For convenience, our team is using a common test framework called Pytest, which is a python module designed for unit testing. Pytest is very similar to many other frameworks (e.g. JUnit, unittest), and very simple to use which fits the scope of our project nicely.

As our team progressed through development, any new piece of code was first tested in a sandbox of its own environment, ensuring that the new piece of software performs well on its own. Following this step, new software ran through the existing tests before merging into the larger code base. This ensured that new development does not break any aspects of existing software.

## User interface testing

Since our product is primarily a user web app, our user interface needed to be tested extensively. The users of interest are primarily students, although advisors, administrators, and faculty also serve as user interface testers.

To streamline this process, we have developed a google form for users to give specific feedback on each page of our website. This process allows for the developers to easily access feedback in an organized and uniform manner.

Over the course of field session, we ran a series of user interface testing on our hosted website. We tested with a handful of students to begin with and plan to test with more following the conclusion of field session. Following the user interface testing, we would enter into the next stage of revisions to address specific feedback from user testing.

## Integration testing

For integration testing, we would do live testing on the hosted website with all parts running at the same time. That being the database, the website server, and the actual website itself. This gave us feedback on if there are any issues when running each of the individual parts together.

Testing like this also allowed us to test in a realistic usage environment. It allows users to emulate the environment they would most likely use it in and if any issues were found we would have a lot of information on the variables that changed for that specific situation.

## Code reviews

Coming into this project, since our existing code base was for a short-term course, we had no test framework set up. As a result, the first week of field session included an in-depth evaluation of the existing software framework.

Going forward, over the course of field session, code reviews have been performed every two weeks. This quality consideration will ensure that all parties involved in development are confident in the level of software quality we are developing, as well as ensure long-term considerations are being evaluated on a consistent schedule.

## User acceptance testing

We are having a handful of students test the software as if they were using it for themselves. We also would like to have some select faculty (i.e. advisors) test it similarly but that is beyond the scope of this course. Testing like this ensures we find any features that we might be missing that we would want to implement as well as bugs that we haven't found yet. It also gives us a lens into the eye of the end user to see what changes we could make to make the product more user-friendly.

# Ethical Considerations

## Pertinent principles

While all of the Principles in the ACM code of ethics are important to ensure that our team maintains the highest level of ethical consideration, the following principles are particularly significant for this software and project.

- 1.06. Be fair and avoid deception in all statements, particularly public ones, concerning software or related documents, methods, and tools.
- 2.04. Ensure that any document upon which they rely has been approved, when required, by someone authorized to approve it.
- 3.12. Work to develop software and related documents that respect the privacy of those who will be affected by that software.
- 3.15 Treat all forms of software maintenance with the same professionalism as new development.
- 5.09. Ensure that there is a fair agreement concerning ownership of any software, processes, research, writing, or other intellectual property to which a software engineer has contributed.
- 7.04. Review the work of others in an objective, candid, and properly-documented way.
- 8.03. Improve their ability to produce accurate, informative, and well-written documentation.

## Violation risks

As students will eventually be making career decisions based on the information that our software provides, principle 1.06 will need to be explicitly addressed. Unless our software product is maintained by someone internal in the Colorado School of Mines administration, which is beyond the scope of this course, our team will need to explicitly state that we are not liable for any decisions that students make with the use of our product. As our product is a helpful tool, any decisions made by students and faculty should be reviewed by an academic advisor.

## Michael Davis tests

Legality Test: Something we have to consider is that, ideally, we would be using student information (such as classes taken, grades, student id, email, etc.) to keep track of students and their schedules. There could be some legal issues with handling such private information such as the FERPA laws. There would be other laws that we would have to consider too such as making registration recommendations and not being liable for decisions made by students based on those recommendations.

Reversibility Test: If we were the end user we would want every single edge case to work. We would need to trust it to work on every schedule we could think of with no caveats. If there was a situation where it wouldn't work we would need to know where it breaks as well.

## Software quality plan considerations

If we don't follow the quality plan we could miss some edge cases or have bugs in the schedule validation code. At the very least it would cause confusion and waste time for people. This could be in the form of checking their schedule themselves and seeing that it doesn't work after all or going to their advisor and seeing that they were missing something. The outcomes could be much worse as well. For example, if our test cases aren't comprehensive enough we could miss an edge case that a student runs into. If that happens a student could end up needing an extra year of school because they had a bad schedule. The worst situation that we could think of is an advisor using the product and making a bad recommendation to a student based on our software. This could result in the advisor facing punishment and the student not meeting degree requirements, missing a class they have to take, etc.

# Project Completion Status

The goal of this project was to continue development on an existing project that had its genesis in a database management course at the Colorado School of Mines. Over the course of field session, goals were set for this project in terms of functional and non-functional requirements, as well as defining an end-point in a definition of done (DoD).

In our DoD, we laid out four main goals that we wanted to hit in this course. A concise summary of these four items is provided below. In the rest of this chapter, the completion status of the project will be discussed including a list of features that were implemented, others that were not implemented, a summary of testing, and an in-depth discussion on performance and usability testing.

## Features not implemented

- Single sign on (SSO)
  - While SSO is a critical aspect of deploying our web application to students in the near future, this step required not only hosting a website but also ensuring that all integration testing was to a confident point so as to not put critical user data at risk. Therefore, while we had initially hoped for implementing this feature in our application, we realized that step should be completed much later on in the project life cycle. With our hosted website, we plan to bring on a security team to ensure risks such as SQL injection are covered before bringing users at risk.
- Reliable data
  - In the future, reliable course data will need to be obtained by the Colorado School of Mines. This data is handled primarily by the Registrar's office at Mines, which we have attempted on multiple occasions to establish a contact with. Unfortunately, the registrar has been slow to respond most likely due to the course falling over the summer months when many staff are likely on vacation. Although data was not obtained, the framework of our software is now at a stage where connections (via API) or hosting of reliable data on our very own database servers is possible. Not having this data has turned out to be a benefit to our project, as we have made our application more general and applicable to institutions beyond the Colorado School of Mines.

## Summary of testing

While the methods and results of our testing strategies and frameworks are provided in detail in the Software Test and Quality Chapter, here we provide a summary and discussion of the testing we have implemented.

Our testing consisted of three main components consisting of software testing, integration testing, and user interface testing. While not exhaustive, these testing methods addressed the needs of the project for the scope of where the project currently stands. Most importantly, effective testing strategies should be scalable, which is one aspect that we emphasized throughout the scope of the project. In what follows, each aspect of our testing will be discussed, and their contribution to the success of our project.

## Software testing

The critical software backend which all front-end software of our application relies upon was tested primarily with unit tests. Since our application is developed using the Flask framework for websites, and web apps, we tested our back-end software with Pytest, an existing python native test framework.

Starting with a code base of thousands of lines of untested software, the existing code had to be chunked into sections to make unit testing possible for the existing code-base. This parsing step included scrapping some non-essential software, and saving modular sections of code for algorithms that will answer the main business questions our software provides. Once existing software was accounted for, test files were made for each aspect of the project that required test modules. This step was considered carefully, as TDD usually works by writing tests before the software exists.

After the existing software was well-tested, we continued with further development with the traditional TDD methods. In particular, one of the main validation algorithms was developed by writing a comprehensive suite of unit tests. This suite includes a number of test users which act as example student users on our web application, each with an associated set of user data. Each test user has an associated set of data consisting of course schedules, majors, minors, and class (e.g. Freshman, Junior). This suite of test users then allowed us to test our algorithms on example data which we would expect from real students using our product as well as test many edge cases. The testing of the code is far from finished but the framework that has been developed, with the ability to create test users in the database allows us to create more tests much quicker than before.

## Integration testing

Along with unit testing for the back-end software of our algorithms, unit tests were also used for ensuring that our database was ported correctly. Not only did we implement test users in separate classes in our software, but we were successful in adding test users to the actual database which will house real student information in the future. This test feature was critical to ensuring that user data is pulled from the PSQL database correctly, meaning that our algorithms interface correctly with real-time data.

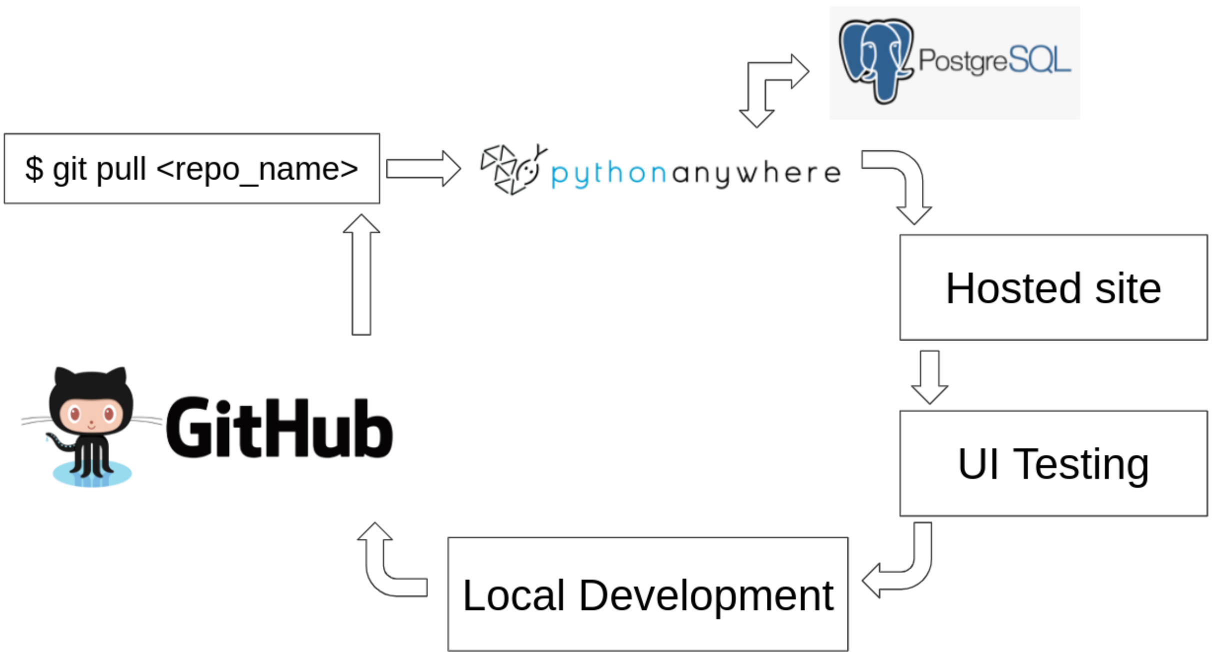


Figure 7: Cycle of integration testing between different components of the project.

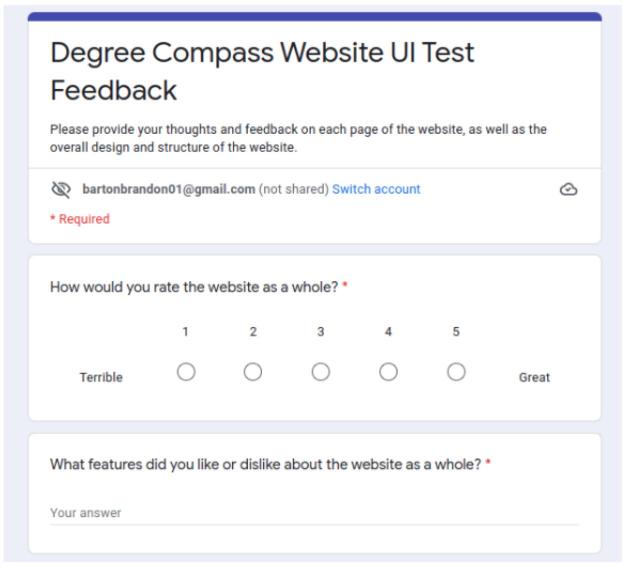
Aside from unit testing in the integration aspect of our testing methods, one goal of this project was to include a seamless life cycle of testing, development, and deployment as shown in Figure 7. Our choice of Pythonanywhere as a hosting service for our application turned out very successful. Pythonanywhere connects seamlessly to our source code hosted on Github, which gets continually updated through local development. Our database also connects to our application through Pythonanywhere providing a high-level assurance that each piece of our software is connecting and functioning properly. After hosting our website through Pythonanywhere, this cycle intuitively leads to UI testing which is discussed in detail in the following section.

## User interface tests

With the hosted website, our goal was to test our user interface with future real-life everyday users of our application. The test group consisted of a diverse set of students across multiple grade levels.

To test our user interface, we allowed students access to our public domain (brandonbarton.pythonanywhere.com) provided in a google spreadsheet. With the provided spreadsheet in Figure 8, users could give ratings on usability and helpful features for each page of the website.

Using this feedback, the design of our website was changed to address the want of saving multiple schedules while also having the freedom to explore different majors. Ultimately, this process is a continual cycle, although the user feedback was paramount in the development of the Explore page, and many adjustments to the home or schedule workspace.


Degree Compass Website UI Test Feedback

Please provide your thoughts and feedback on each page of the website, as well as the overall design and structure of the website.

bartonbrandon01@gmail.com (not shared) Switch account

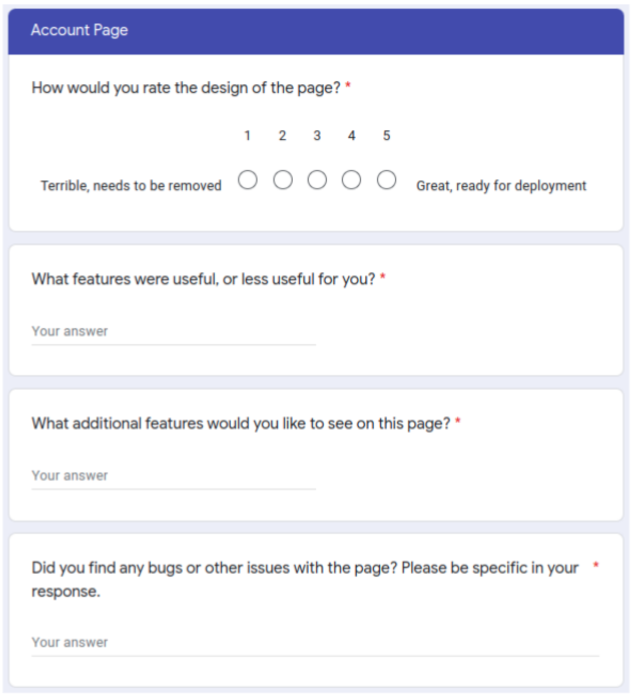
* Required

How would you rate the website as a whole? *

|  | 1 | 2 | 3 | 4 | 5 |  |
|---|---|---|---|---|---|---|
| Terrible | ○ | ○ | ○ | ○ | ○ | Great |

What features did you like or dislike about the website as a whole? *

Your answer


Account Page

How would you rate the design of the page? *

|  | 1 | 2 | 3 | 4 | 5 |  |
|---|---|---|---|---|---|---|
| Terrible, needs to be removed | ○ | ○ | ○ | ○ | ○ | Great, ready for deployment |

What features were useful, or less useful for you? *

Your answer

What additional features would you like to see on this page? *

Your answer

Did you find any bugs or other issues with the page? Please be specific in your response. *

Your answer

Figure 8: UI testing and feedback form.

# Future Work

Our project is still in its infancy. There are still many features to implement, pages to be reconstructed from the ground up, and integration with Mines systems to be done. A lot of the work done on the project during this field session was to free it from its reliance on certain temporary systems so that we could work on more permanent solutions. Here are some of the features we would like to implement in the near future.

- SSO: Single Sign On would let us remove the sign-in and sign-up pages in favor of a sign-in page similar to canvas that would allow us to create user accounts before they even know about the website's existence. It would also be much more secure and it would be integrated into the Mines system meaning we could use Mines user data much more easily.
- Reliable data: Currently we are using the Mines catalog website for course data. Ideally, we would use whatever database or API Mines currently uses to store that data so we could directly access that and avoid any bad data as well as make updates much easier.
- Mines domain: We want to make this part of the Mines community and one of the ways that we want to do that is through a Mines domain name. Integration with trailhead as well as a mines domain name would make it much more obvious to students that they could use this software to help them during their time at Mines.
- Faculty/Administrator page: This would allow faculty/admins to be able to make changes to the catalog, look at their students' schedules, make curriculum updates (i.e. new course requirements), among other things. This would give the institution more control over the website.
- Account page automation: The account page currently allows users to manually input previous course history. To make this process smoother, the web app should be capable of scraping data from transcripts directly, or connect to an existing Mines database with student information data. Another aspect of the account page that may be automated is connecting to the major declaration process currently in place at Mines.
- Extending major capabilities: While the database contains the Computer Science and a small pool of other major requirements currently, other majors and minors can be added to extend capabilities university wide.

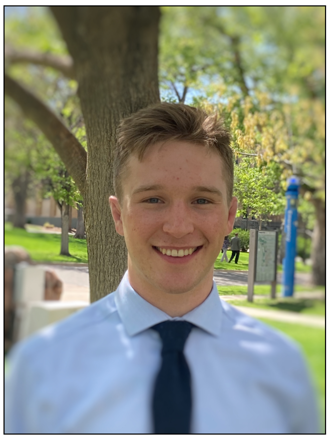
# Lessons Learned

We jumped into this project trying to do things we have never done before. We had to learn quite a few new technologies along the way. Some of those technologies were: hosting a website, hosting a sql server, flask(python library), and many of python's quirks. One of the lessons we learned by learning these new technologies is that it takes time. We would have plans to be done with something in a day or two and then we would hit a snag and have to push things back. If we were more realistic about the time it took to learn new technology we could have had better plans.

Another lesson we learned was to work more in parallel. We would try very hard and dedicate a lot of time to get one thing done but if we got stuck on it, or it was just taking a long time (like migrating the database) we would lose a lot of time that we could have spent working on other things as well. Even if the other things we would have been working on relied on the thing we were stuck on, it would probably be better to just work on them anyway just to be in a better spot once the bottleneck is cleared up.

One of the more important lessons we learned was to build with a redesign in mind. There were many times we had to rethink something or port code to a different platform. Some of these went smoother than others because of the way we had structured our design. A good example of this would be moving to a different database, it actually went very smoothly (ignoring mistyping some things that took a while to debug) because we structured the code in a way to work with any psql database. On the other hand, we had some issues with populating the database because the script we had to populate the database only worked with Mines' codd database.

## Team Profile



**Brandon Barton**
**Class:** Senior
**Major:**
BS Computer Science 2022
MS Applied Mathematics and Statistics 2023
**Hometown:** Longmont, CO
**Work Experience:** Undergraduate research
**Interests:** Rock climbing, Brazillian Jiu-Jitsu

**Roman Downie**
**Class:** Senior
**Major:** Computer Science Major
**Hometown:** Denver, Co
**Work Experience:** IT consultant at Colorado School of Mines
**Sports/Activities/Interests:** Video Games, Archery, Guitar and Bass

# Appendix

- ACM: Association of Computing Machinery
  - https://www.acm.org/code-of-ethics
- Flask: Web framework, it's a Python module that lets you develop web applications easily. It's has a small and easy-to-extend core.
- IEEE: Institute of Electrical and Electronics Engineers
  - https://www.ieee.org/about/corporate/governance/p7-8.html
  - https://flask.palletsprojects.com/en/2.1.x/
- PythonAnywhere: Hosting service to run and code Python in the cloud.
  - https://www.pythonanywhere.com
- Pytest: Python testing module
  - https://docs.pytest.org/en/7.1.x/