COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

Corn Off The Cob

Calan Barnhardt
Jordan King
Rob Hartnagel

Revised November 30, 2024

CSCI 370 Fall 2024

Prof. Kathleen Kelly

Table 1: Revision history

This will be updated with each submission

Revision	Date	Comments
New	August 21, 2024	Completed Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done XIII. Team Profile
Rev – 2	September 13, 2024	Updated Sections I. Introduction II. Functional Requirements III. Non-functional Requirements VI. System Architecture
Rev - 3	October 20, 2024	Completed Sections: VII. Software Test and Quality VIII. Project Ethical Considerations
Rev – 4	November 9, 2024	IX. Project Completion Status X. Future Work XI. Lessons Learned

Table of Contents

I. Introduction

OreCode's mission is to grow students into software engineers, great leaders, and communicators. They also want to empower local companies, Mines, and the city of Golden with software solutions by creating low-complexity, maintainable systems.

The goal of aRchitect by OreCode is to build a cloud platform to run R code in the browser in a secure way, manage assignments, and autograde R output. After encountering many issues with RStudio across multiple statistics classes, OreCode wanted to create an application where students can collaborate, teachers can manage students, and most importantly, students don't need to install any packages. No prior work has been done for this project. The stakeholders are statistics professors and students, and OreCode will be responsible for maintaining the software.

Definitions:

- **Knitting**: Converting R markdown to PDF. R markdown is stored as .rmd, and knitting is the process of getting the code out of the markdown file and running it to make it a pdf.
- **Sandboxed Code Runner**: An isolated environment so that code can run without affecting the host machine.

II. Functional Requirements

A. A code editor in the browser with a sandboxed code runner.
B. The ability to run R code with packages/libraries, and not needing to download them locally.
C. The ability to knit a pdf from markdown and R code.
D. An autograder. (stretch goal)
E. A teacher view with a login, the ability to create assignments with needed packages, and view submitted assignments. (stretch goal)
F. A student view with a login, the ability to view assignments, work on them, and turn them in. (stretch goal)

III. Non-Functional Requirements

A. The development of the project should be completely free, and post development it should be as cheap as possible to run.
B. The program should be secure so that malicious code cannot be run. There will need to be a way to make it so the code cannot harm individuals' computers in any way.
C. The website should load quickly (the website itself and the packages with R), and it should be able to run the code quickly.
D. Write easy to understand code so OreCode or other field session students can continue development after.
E. Backend code must be written in either Python, Java, or Go.
F. Frontend framework should either be Angular, Vue, or React

IV. Risks

Security Risks

- Likely
- Major
- Create a sandboxed running environment so hackers can't run code on the host machine.

High Latency

- Likely
- Medium
- At every step in creating the program, we will make sure that we keep in mind having as low latency as possible for the user.

Time Risks

- Likely
- Medium
- There is a lot to do for this project, and not a lot of time. There is a chance every functional and non-functional requirement will not be met by the deadline. The focus will be on creating a minimum viable product.

Lack of prior knowledge

- Very likely
- Major
- Do a lot of research and follow tutorials, as well as ask the client for help whenever we get blocked.

V. Definition of Done

- R Markdown files can be created on the website.
- R Markdown files can be parsed into R scripts and run.
- A pdf of compiled code can be sent back from the coderunner to be displayed to the user.
- Preventative measures for malicious code in place.
- Packages that are wanted are included in the system and allow use when the R code is run.

How the clients will test our code:

- Running code that should break the whole system to make sure it's safe.
- Running some basic R code and getting back a pdf.
- Using functions from a library/package that should have been included; ensuring this works.
- This is a ground up project, so we have some basic goals to complete. However, there are stretch goals for extended features for the web app, so many deadlines/dates are to be determined.
- Will be uploaded into Orecode github organization for them to use for their own future clients once proof of concept is completed.

VI. System Architecture

Technical Design Issues:

- Connecting backend to coderunner
- Having backend spin up code runners when code must be compiled
- Efficient design of Postgres database

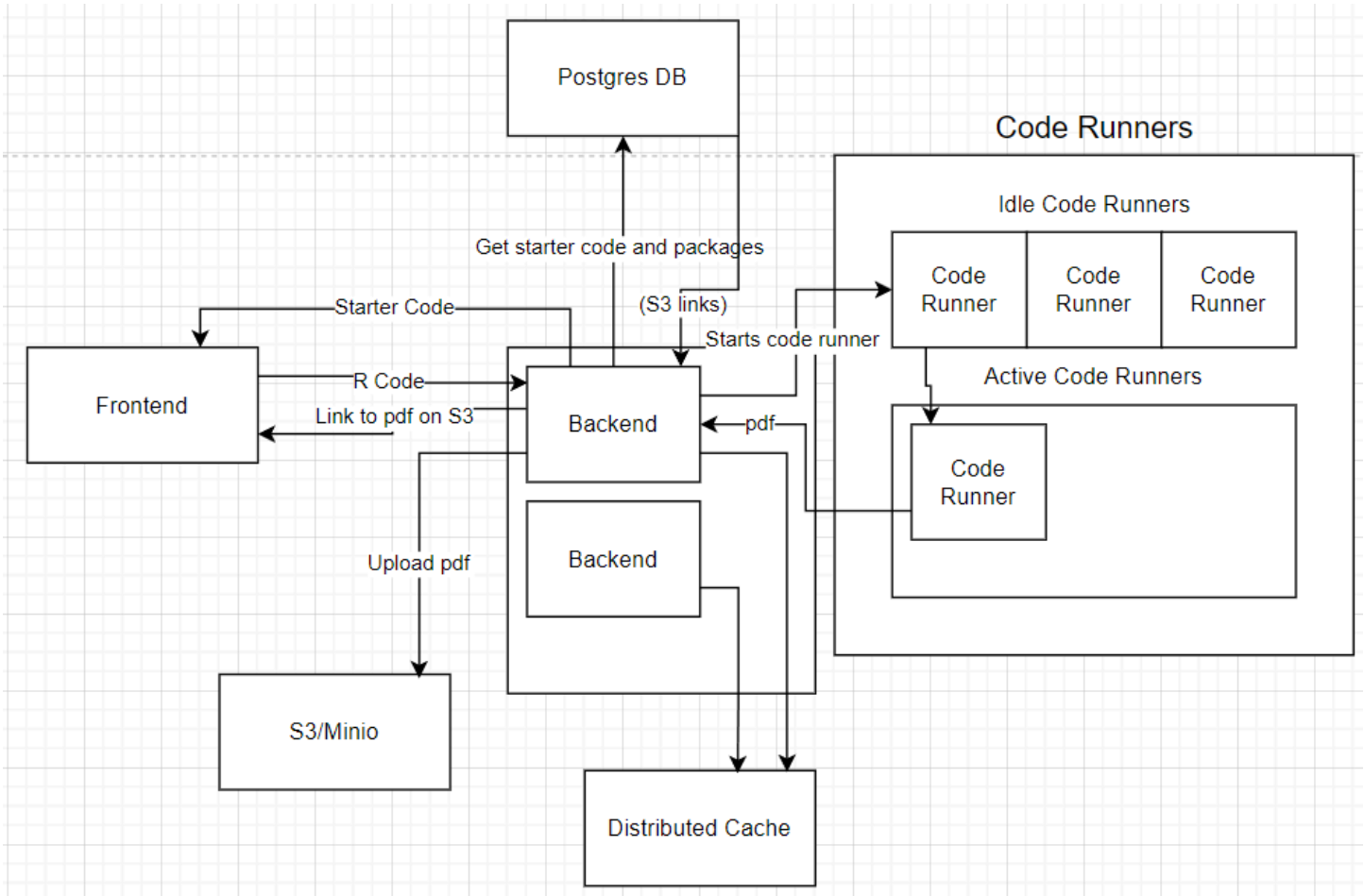


Figure 1. aRchitect Infrastructure

Figure 1 shows the aRchitect infrastructure. The frontend was created with a code editor where R code can be written. The plan for the backend was to send starter code and packages to the frontend when the user chooses a project. However, this was not implemented due to time constraints. The frontend and backend were designed such that when code must be compiled, the frontend sends the code to the backend, where a coderunner container is created. The backend sends the code to the coderunner, where it is compiled into a pdf and stored in MinIO. The coderunner returns a link to the pdf that contains the compiled code to the backend, and the backend sends this link to the frontend for display.

Figure 2 is an ERD that shows the layout of the database with all the primary keys that were used. This is the Postgres database that was used throughout the project.

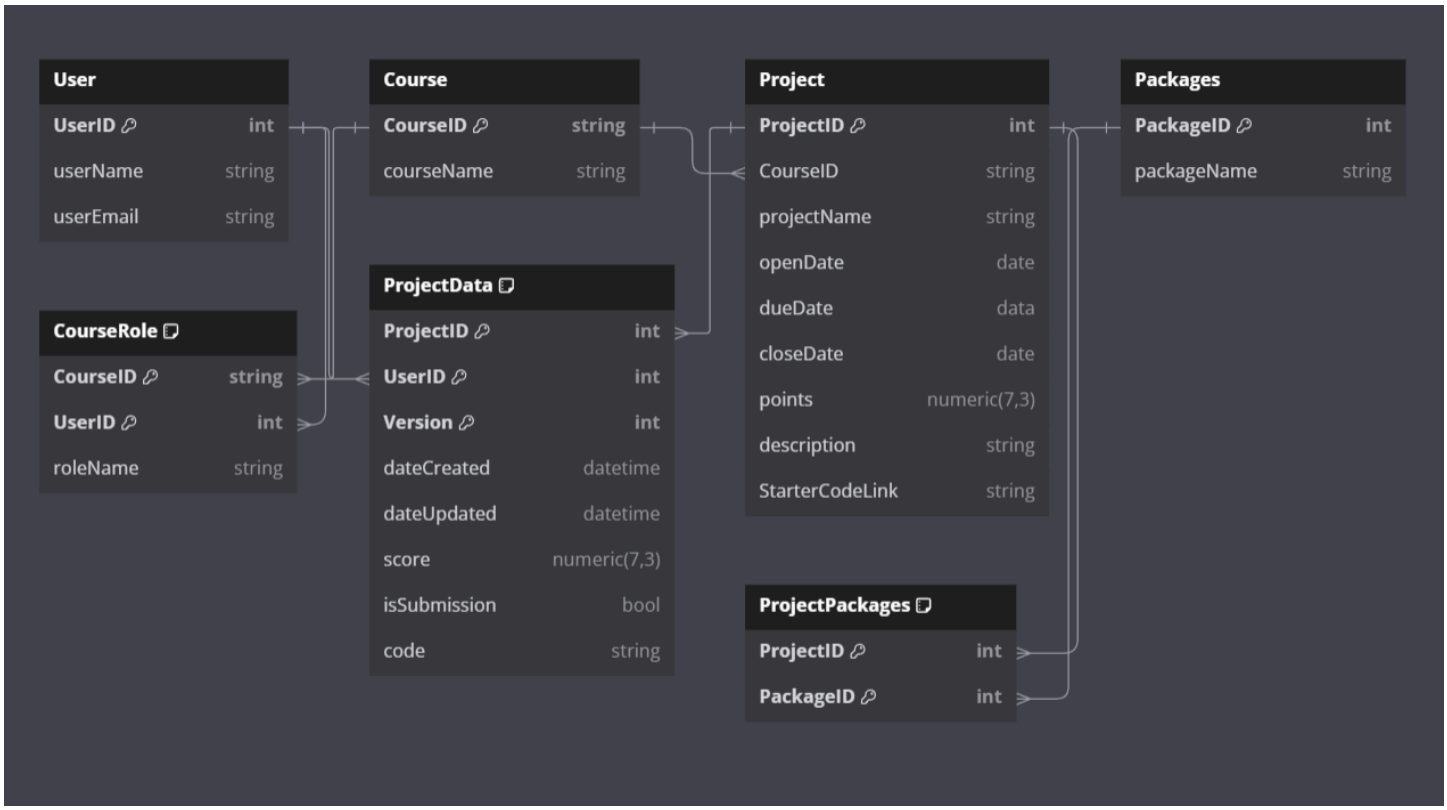


Figure 2. Database Infrastructure

VII. Software Test and Quality

Throughout the semester, the project features and requirements changed rapidly, and the clients' focus was on developing new features (rather than tests). This project was built from scratch this semester, and the tests reflect that.

Docker Containers Test:

- *Purpose*: To make sure that all the code is dockerized and can run on any machine
- *Description*: To confirm that the frontend, backend, and coderunner docker containers are all created at the correct time, and have the correct code containerized.
- *Tools Required*: Docker desktop, the entire application, and different computers with different operating systems are all required.
- *Threshold for Acceptability*: These tests will need to work perfectly for the application to work, so all the code will need to be in the Docker containers with the right packages.
- *Edge Cases*:
    - Creating the containers when the containers have already been created
    - Creating the containers when none of the containers have been created before (can be done by removing the containers or testing on a new machine)
    - Testing on different operating systems (like Windows, Mac, and in the cloud)
- *Result*: The result will be an application that we know can run on any computer.

Communication with Coderunner Test:

- *Purpose*: To make sure the frontend can send R code to the code runner.
- *Description*: To make sure that an Rmd can be sent from the frontend, to the backend. Then to make sure the backend can spin up a code runner and then send the data to the code runner.
- *Tools Required*: Docker desktop and the entire application.
- *Threshold for Acceptability*: These tests will need to work perfectly for the application to work, so the Rmd will always need to be sent to the code runner.
- *Edge Cases*:
    - Sending a blank Rmd file
    - Sending a very long Rmd file (multiple page lengths)
- *Results*: The result will be knowing that R code can be sent to the code runner.

Compiling R code to a PDF:

- *Purpose*: To make sure an Rmd can be turned into a knitted PDF.
- *Description*: To confirm that once the code runner gets an Rmd, it can be knitted into a PDF.
- *Tools Required*: Docker desktop, test Rmds, and the code runner container.
- *Threshold for Acceptability*: These tests will need to work perfectly for the application to work, so the Rmd will always need to be turned into a valid PDF.
- *Edge Cases*:
    - Testing a blank Rmd file
    - Testing a very long Rmd file (multiple page lengths)
    - Testing a Rmd file that requires packages
    - Testing a Rmd file that outputs graphs
    - Testing a Rmd file that outputs tables
    - Testing a Rmd with latex included
- *Results*: The result will be knowing that if a code runner gets a Rmd, it will be knitted into a PDF.

VIII. Project Ethical Considerations

Some ACM/IEEE principles that were pertinent to the development of the product are as follows:

1. IEEE: 3.02. Ensure proper and achievable goals and objectives for any project on which they work or propose.

It was important that we had direct and clear communication with our clients about what they wanted us to achieve and what we believed was achievable for us given the timeframe and our experience.

2. IEEE: 3.11. Ensure adequate documentation, including significant problems discovered and solutions adopted, for any project on which they work.

Because we were building this project from the ground up, it was not anticipated that it would be finished when field session ended. Therefore, it was imperative that we adequately document our solutions and any problems that we ran into so that those picking up the project are well-informed.

3. ACM: 3.5 Create opportunities for members of the organization or group to grow as professionals.

This was an excellent opportunity for us to learn and grow as computer scientists, and it was important that we were distributing the work as evenly as possible in order to provide everyone with adequate opportunities to do so.

4. ACM: 1.2 Avoid harm.

This is a tool that involves student work and grades, so it was important that we employed proper security measures to ensure that no one is harmed by information being leaked. Also, since code is run on student and teacher machines, it was important that we provided measures so malicious code cannot be run.

Of the ACM/IEEE Principles listed above, ACM 1.2 Avoid harm was in the most danger of being violated. It was important that we employed effective security measures and thoroughly tested them. If we failed to do this, student data (including assignments and grades) could be leaked, or harmful code could run on someone's computer, breaking it entirely.

Harm Test: Our option does less harm than alternatives, for example using RStudio instead of our program aRchitect. RStudio is known to a lot of students to be hard to use, clunky, and just overall not a great option for students trying to use R to complete projects. Here, our new product aRchitect, fixes the many issues of RStudio, and it benefits not only students, but TAs and Professors as there are tools for them to use.

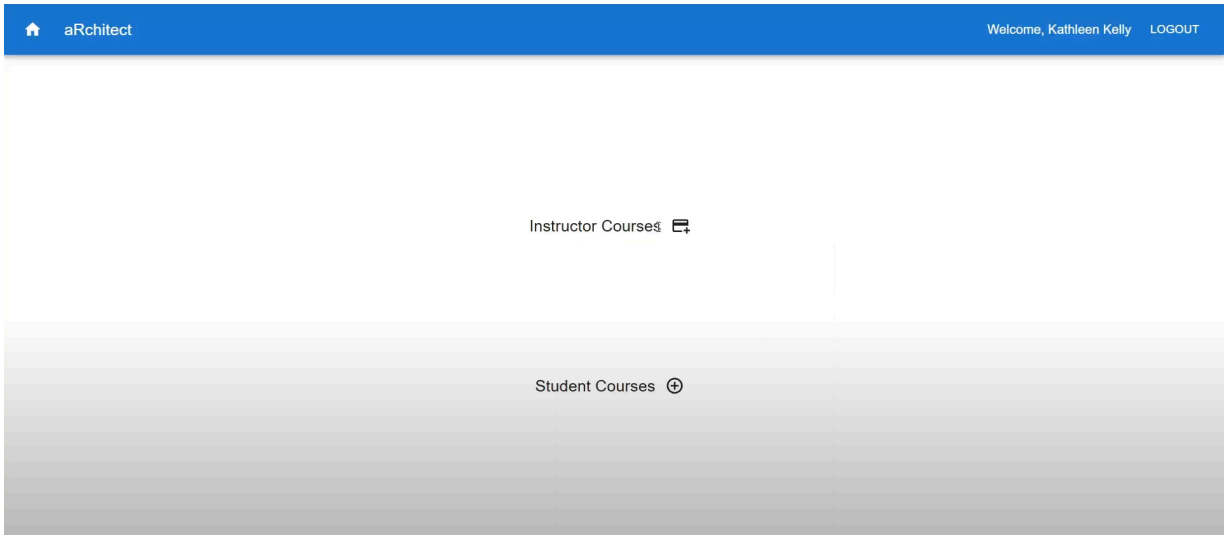
Mirror Test: We would all feel proud looking at our project, as not only were we able to build our application from the ground up, but we were able to create something that will help students, and make completing their projects easier. We also built a great tool for professors and TA's and can make their job of grading and keeping track of students work much easier. We were able to do this in a completely ethical way where we did not steal data, store passwords, or keep anything that we would not be proud of under the hood of the software.

If our software quality plan is not implemented properly, then students using the tool we are creating could experience several negative effects, such as their code not compiling correctly (even if they have correct code), their code not compiling at all (if the frontend fails to send to the backend), and inability to run the application at all (if the docker containers are not set up properly).

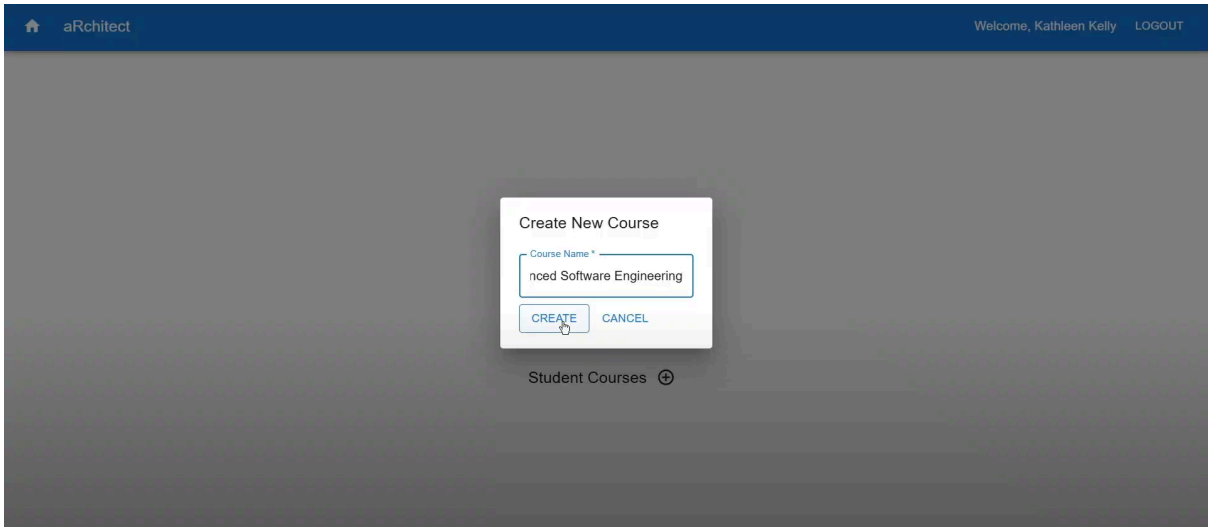
IX. Project Completion Status

The goal of this project was to create a cloud platform for running R code in the browser, managing assignments, and autograding R output. The application we created successfully meets most of our client's functional requirements.

After logging into aRchitect, you are presented with the courses page shown in figure (3). In this page, we added functionality to create a course (4), and to add a course (which has a very similar UI so an image is not shown).

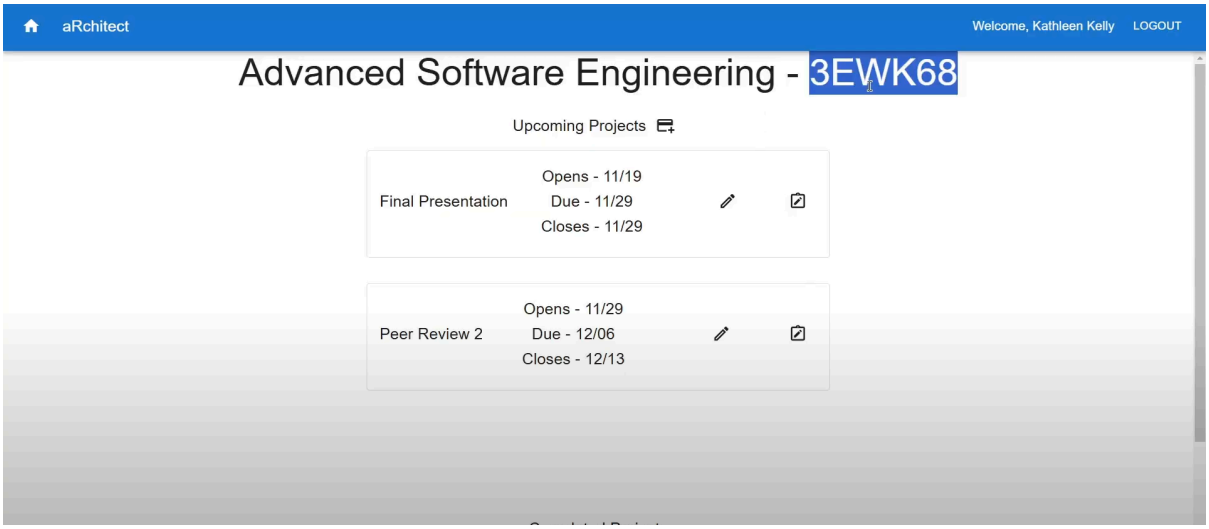


(3) Courses Page

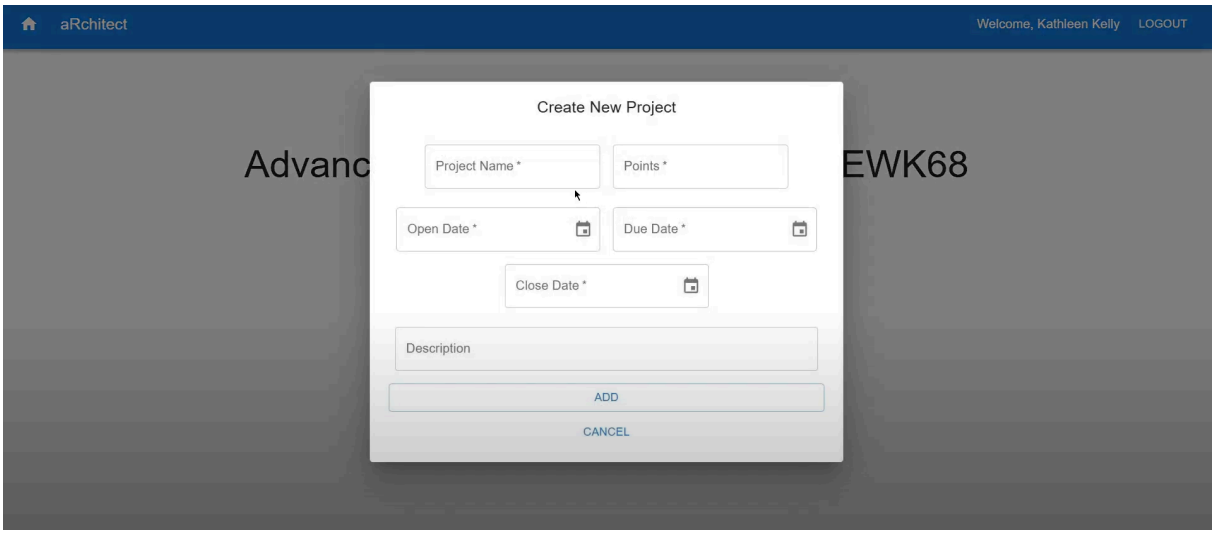


(4) Creating a new course

The website also has a projects page, which has both a teacher (5) and student view. They are very similar, but the student view cannot create new projects, and doesn't show extra information (like the buttons to edit/grade, show the open date for a project, etc). Teachers can create projects, which results in a popup that contains all the fields required for a project (6).

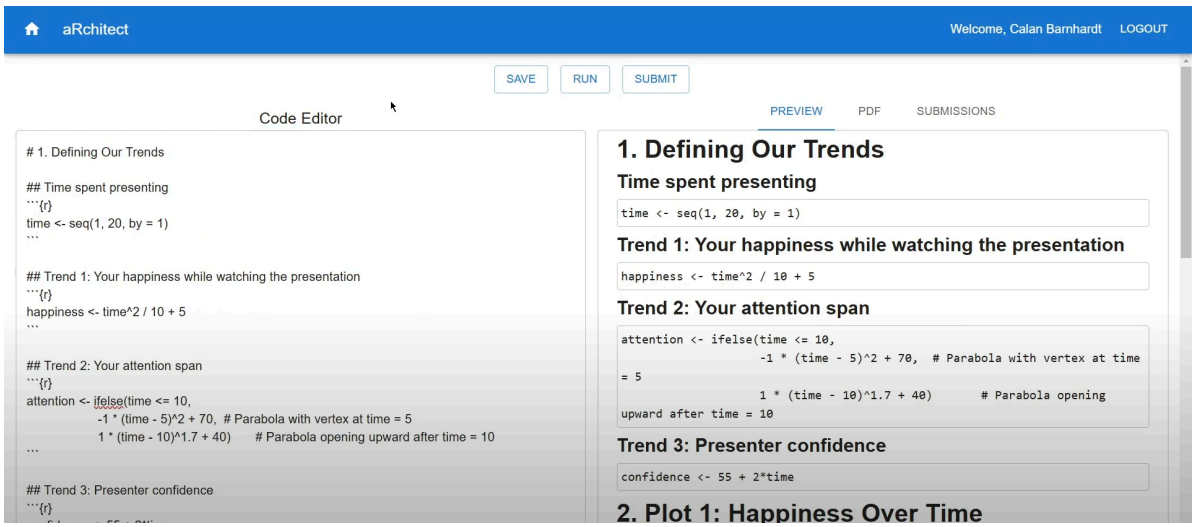


(5) Teacher view of the projects page



(6) Creating a new project on the teachers project page

Finally there is a code editor for which the students can write their code and view rendered markdown on the preview tab (which live updates whenever typing). Students can also save their code, run it, and submit their code (which will show on the submissions tab). All of the logic is fully functioning in the frontend, backend, coderunner, and database!



(7) Code editor

Features not implemented include the teacher-view function of being able to create assignments with starter code that specifies packages/libraries needed, an autograder, and manual grading.

X. Future Work

One piece of work that needs to be done in the future is to implement packages/libraries that are automatically installed per assignment so students don't need to install them manually. To do this, we need to add the ability for teachers to add required packages to projects. Then we will need to download the packages and upload them to minio (or cloud bucket storage), and fetch those packages whenever a student runs that specific project. This will require a large amount of frontend, backend, and database knowledge, and will take at least 10 hours.

There are also some features related to teachers that haven't been completed. This includes being able to grade assignments, creating an autograder so teachers do less work, being able to edit courses, and having starter code for specific assignments. All these tasks will require frontend, backend, and database experience, and will require at least 3 hours per task, with creating the autograder being a semester long project.

Once there are clients for the project, everything will need to move to the cloud. We built the project in a way that this should be easy, but still expect it to take a while (at least 20 hours). This will require cloud hosting experience, as well as an understanding of aRchitect.

For all of the future work, all that is needed is the project, an IDE, node, and Docker Desktop.

XI. Lessons Learned

- Software Development Process
    - We all got great experience developing an application and being able to make a functioning app for many to use, and all that goes into completing that.
- Architecting
    - Since we got to design our project from the ground up, we got to learn how to architect a project, and how the decisions made can affect the entire project and workflow. We also gained experience by getting to design how everything works, and becoming true experts on our application.
- Team-based Development
    - Getting to work as a team. Learning to use each person's strong suits allows the person that is most well equipped for a problem to solve it. Also being able to communicate, and notify each other of delays, and understand what each person is working on.

XII. Acknowledgments

This project was very large and would not have been possible without the help of a few key figures. The main people we would like to acknowledge are the following:

- Tyler Wright and Ethan Richards, who were our clients, helped us throughout the entire development process. We met with them every week, and they always answered our questions, gave us tasks to do, and helped us solve complicated bugs whenever we ran into them.
- Caleb Bartel was also a great help throughout the entire semester. He helped us know exactly what we needed to do for the class, helped us improve our agile workflow, and made sure we were on track to complete the project before the end of the semester.
- Kathleen Kelly was also a great help when planning all the different talks throughout the semester. These talks helped us to become better developers, and finish the project in a more clean and efficient manner.

XIII. Team Profile

Calan Barnhardt

- Computer Science Major with general focus
- Durango, CO
- Interned at Tyler Technologies and Atwell
- Hiking, camping, skiing, tennis

Jordan King

- Computer Science, general track
- Six Mile, South Carolina
- Interned at Framatome
- Reading, slacklining, snowboarding

Rob Hartnagel

- Computer Science BS + MS
- Nanuet, NY
- Intern at Tyler Technologies and Aegon
- Hiking, Climbing, Volleyball

Appendix

All the required information to continue with the project is in the documentation within the Git repository. There is documentation on coding standards/conventions, how to set up the project, as well as suggestions on how to implement the work in the future work section. There is also documentation on how the backend and coderunners work so anyone picking up the project can understand how everything works.