PrairieLearn Implementation for CPR E 288

Final Report

Group Number: 33

Client: Philip Jones

Advisor: Philip Jones

Scrum Master: Carter Murawski

Consultant: Matt Graham

Quality Assurance: Chris Costa

Project Manager: Tyler Weberski

Technical Lead: Mitch Hudson

Construction: Andrew Winters

Email: sdmay24-33@iastate.edu

Website: https://sdmay24-33.sd.ece.iastate.edu/

Revised: 4/25/24 Version 1

Final Report **1**
Introduction/Background 3
● Problem statement 3
● Intended users and uses 3
● Place work in the context of related products 3
Revised Design 3
Functional Requirements 3
Security Requirements 3
Non-Functional Requirements 4
UI Requirements 4
Performance Requirements 4
● Engineering standards 4
● Security concerns and countermeasures 5
● Description of how your design has evolved since 491 5
Implementation Details 6
Testing 7
● Process 7
Unit Testing 7
Interface Testing 7
Integration Testing 7
System Testing 8
Regression Testing 8
Acceptance Testing 8
Security Testing 8
Black & White Box Testing 9
● Results 9
Broader Context 9
Conclusions 10
Appendix 1 - Operation Manual 10
Set-Up 10
Demo 10
Testing 11
Appendix 2 - Alternative/Initial Version of Design 12
Appendix 3 - Other Considerations 13
Appendix 4 - Code 13

Introduction/Background

- ## Problem statement

  Our project is to make a more intuitive and engaging learning experience using the PrairieLearn Framework for students in the CPR E 288 course. Our project will need to create a new course with new customizable and randomized questions, where students can answer the questions and have them auto-graded once submitted.

- ## Intended users and uses

  Professor Jones and the CPR E 288 classes benefit from our project, as they would use our results to have an alternative to their homeworks with the autograder.

  An instructor needs to be able to create courses for students to register for. The instructor must be able to add exams and homework assignments. The instructor must be able to create questions for both the exams and homework assignments they create.

  A student must be able to register for a course. In this course, a student must be able to complete exams and homeworks assignments assigned to them by the instructor. The student should be able to see the grades they receive on these exams and homework assignments.

- ## Place work in the context of related products

  The PrairieLearn software competes with products like Canvas, Gradescope, and Renaissance. The way PrairieLearn sets itself apart and why we used their open-source project is the highly customizable autograding features, especially for higher-level education. We developed a baseline for c-autograding and general questions. On top of this, our development of an emulated microcontroller can be integrated into PrairieLearn, which allows for intuitive randomized learning for students to learn along with their labs.

Revised Design

Functional Requirements

- Questions must be autograded as much as possible
- Must have questions and assignments for each existing assignment in the course
- Organize students into sections and semesters for easy grading
- Connect with ISU's existing Okta SSO for easy log-ins
- Connect with Canvas to let students see grades immediately
- Integrate an emulator for running student bot code in a virtual environment

Security Requirements

- The firewall should prevent undesired access to the system
- SSH needs to be secured to prevent unauthorized access to the system
- Traffic between the users and the server must be encrypted and protected from modification
- User-submitted code should be sandboxed to prevent it from affecting the system as a whole

- The load balancer should protect the system from undesired downtime

Non-Functional Requirements

- Updating questions should be easy and pain-free for TAs and instructors
- Assignments should load quickly
- Questions should be intuitive and understandable
- Grading should be efficient and give feedback in a reasonable time frame
- Grading feedback should be understandable and helpful to the student.

UI Requirements

- Pages should adhere to existing web standards
- UI should be easy to use and intuitive for new users
- Web pages and interfaces should be visually appealing

Performance Requirements

- The system should be able to handle at minimum 30 consecutive users compiling code.
- The system should have less than 5% downtime.

Engineering standards

IEEE 610 Standard Glossary of Software Engineering Terminology - Having technical terms for ourselves to communicate, as well as future groups to use, will be essential to speeding up specific processes rather than describing what should be known when attempting to further this project

IEEE 830 Software Requirements Specifications - This standard is heavily involved with what we are doing now, finding out the different functional and nonfunctional requirements and the use cases.

IEEE 1016 Software Design Description - Throughout our process, documenting our designs specifically within the Prairie-Learning work will be critical as the UI look and functionality will need various documentation aspects.

IEEE 1074 Software Development Life Cycle - Going through any of the lifecycle models will be essential for the flow of the project. Although we don't have a specific lifecycle model set yet, it will be decided soon to have a fluid project when software development starts.

IEEE 2050 RTOS for embedded systems standard - Using this standard is with the various topics we will need to cover from the CPR E 288 Intro to Embedded Systems class.

Programming Languages: Python, Javascript, C - These languages will be the main focus of our software development, and being comfortable with all 3 will be vital to the success of the team.

Security concerns and countermeasures

Our project is entirely software-based, which means the only risks we need to contend with are performance targets, availability requirements, and the functionality of our tools. Since our project uses docker containers for compiling and simulating code, the server would need to be fast enough to accommodate several classes' worth of students at a time. We also need to ensure the server has

as little downtime as possible to allow students to work on labs and assignments at all times, especially near due dates. The last risk we could have is PrairieLearn, or the emulator could fail to meet specifications and will not work for what we need it to do.

The performance risk will be relatively low, probably around .1, since PrairieLearn is a mostly client-based web framework that doesn't take much processing power on the server side beyond autograding. The availability risk should be around .2 because we can set up multiple servers as redundant backups for when downtime is needed. Finally, the risk of tools not meeting expectations is probably around a .4, considering no team member has experience with them. Thanks to that risk, our mitigation plan is to do some market research for tools that may be better suited to our needs. Using that research, if we ever find out PrairieLearn cannot fulfill our needs, we can quickly find a tool to fill that area in.

- ## Description of how your design has evolved since 491

  Since 491, we have completed homework 1 through 12 with all questions Implemented. These questions now range from fully randomized and auto-graded fill-in-the-blank to writing C code that is submitted and auto graded. The server is now fully operational with the security implemented. We have created documentation and videos detailing how to implement all of the work we have done. We also integrated Okta sign-ins to allow students to sign in to PrairieLearn with their school accounts. Finally, we created emulators using a modified Wokwi emulator for the Raspberry Pi Pico and a custom-built board implementation for QEMU. The two emulators allow code to be executed on the server as though it was running on the microcontroller board.

Implementation Details

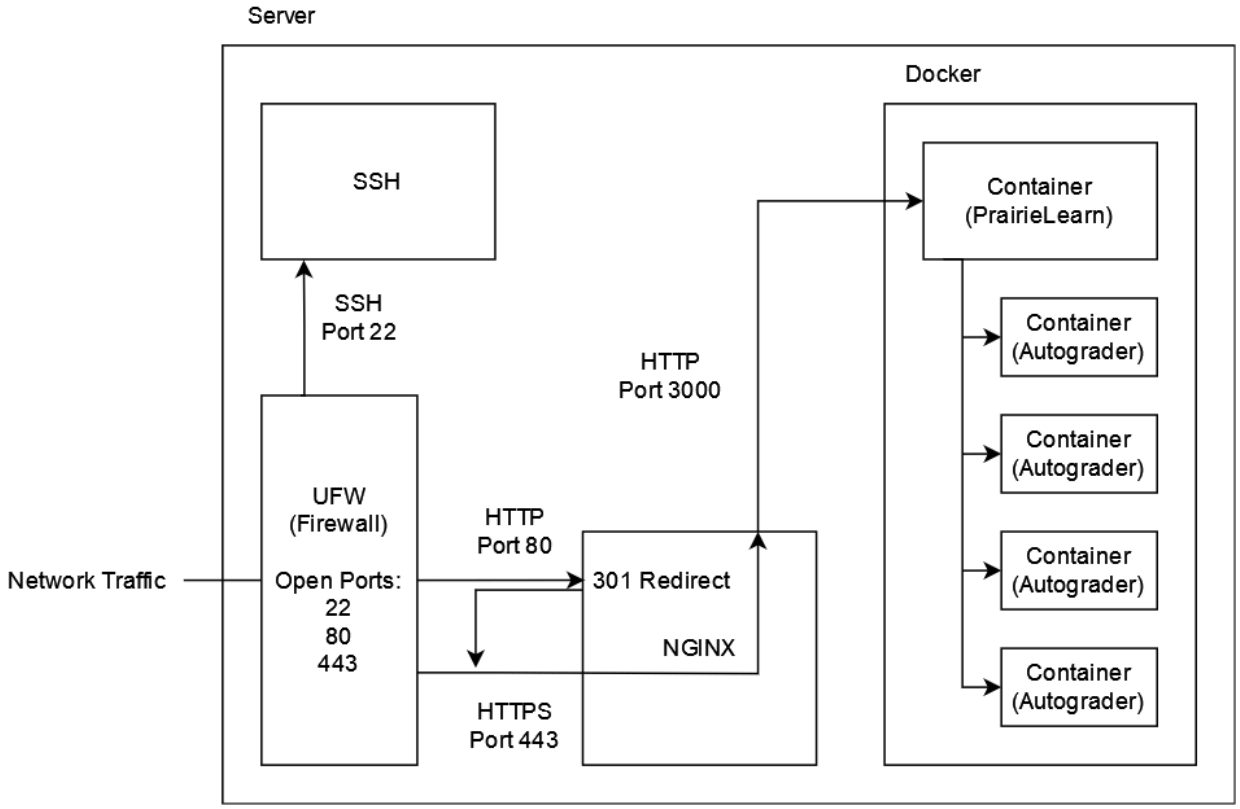

Figure 1. Design Diagram

In this matured design, we have a cemented view of the overall system design. Several servers will be set up, as shown above, and each will be behind a load balancer that ensures no single server is being overloaded.

What led to this iteration was a thorough understanding of the requirements for PrairieLearn and research into the design of production servers. The significant change from Design 0 is the addition of security features. These include the firewall, Nginx, and SSH security changes. The firewall prevents connections from any port except 22 (SSH), 80 (HTTP), and 443 (HTTPS) and fulfills the security requirement for the firewall. Port 22 is handled by the SSH daemon, and ports 80 and 443 are handled by Nginx. Nginx was added to provide encrypted traffic between the user and the server through HTTPS on port 443, fulfilling that security requirement. This traffic is then internally routed to the Docker container running the PrairieLearn server. Setting it up this way allows us to utilize the robust security and efficiency provided by NGINX while still using PrairieLearn's server. SSH is how we, as the developers, access the server for things like maintenance and updates. We added multi-factor authentication and public key authentication to protect this important part of the server. These changes fulfill the security requirements related to connecting and accessing server resources.

The docker containers running the autograders are used for sandboxing user code. Sandboxing protects the system from user-submitted code that might be dangerous and fulfills that security requirement. Thanks to Docker, multiple autograder containers can run simultaneously, helping handle multiple users at once and fulfilling the performance requirements. The autograder containers perform all of the autograding tasks, and the PrairieLearn container runs PrairieLearn. A lot of our functional requirements are built into PrairieLearn, such as organizing sections, creating / modifying questions, and displaying pages in visually appealing ways.

Since 491, we have also implemented new Docker images for running student code using the emulators. This was used to create homework assignment 12, which focused on teaching students how to write ARM assembly code. Having it run with the emulator means that the assembly code can be emulated perfectly as though it were running on an actual ARM processor, and all of the functionality for the microcontroller board is also implemented.

Testing

- ## Process

Unit Testing

For our project, the 'units' being tested are the individual questions and the emulator. The questions are tested within PrairieLearn, where they are compiled and previewed. The built-in grading system allows us to test and ensure the questions work as intended. The emulator was testing by testing a variety of user inputs and comparing them to the expected results. Most tools are built into the PrairieLearn framework as files and questions that can be edited directly through the web interface. We also test our units by showing them to our advisor at each weekly meeting to ensure they look and function as he wants. He has also helped with making sure the material is correct, as the homeworks we are implementing was created by him.

Another set of unit tests is the auto grader Docker containers. Since each question will create a new container to auto-grade, we can test whether the grading works as intended. To do this, PrairieLearn provides a helpful interface for working with the grader containers and testing their functionality. For example, if a compilation error shows up, it will be reported when submitting a response to the question.

Interface Testing

The interface in our design is within the PrairieLearn environment. Within each question, we have a JSON, HTML, and Python file with works with each other. Then, we make an assignment that combines multiple questions made. We test how they communicate and ensure the output works as desired. This is done within the PrairieLearn environment, giving us feedback on which issues are occurring within the questions.

Integration Testing

The most crucial connection path we have is making sure each individual question works on its own and can be integrated with or without the emulator with no issues into one big assignment so that

the students going through the homework can quickly finish it. These will be tested by individually reviewing the Unit testing for each question and ensuring proper formatting. After that, we will go through the interface testing and connect the questions to an assignment. This, in turn will be finishing the integration testing. The tools we will be using are what are available within the PrairieLearn environment we are using.

System Testing

The system can be tested by testing homework assignments and ensuring they all work correctly each time a question is attempted. This will ensure that when students attempt the homeworks, it all works properly, and their learning time will be maximized. Many of the questions within the homework are randomized, so testing the homework assignments multiple times will be beneficial to ensure that the random value in a question doesn't mess up the student.

Regression Testing

For regression testing, we will ensure that any new addition is contained in its container for each question, homework, and class. Containerization is a tool that was implemented in PrairieLearn previously and helps ensure that everything interacts with the necessary components. One critical feature we need to ensure that PrairieLearn will continue to run is to have every system component scalable. Scalability is required for this project because PrairieLearn is a constantly growing system that needs to be adjusted for size and requests over time. After all, it is a requirement driven by PrairieLearn requirements.

Acceptance Testing

To demonstrate the design requirements were met, we ensured that all pages made on the PrairieLearn website match the initial format given. Part of this is ensuring that all code is in a readable format and that each page has a consistent layout. To test this, we will manually review each of our courses, assessments, and questions to ensure they match the existing framework. To demonstrate that our functional requirements have been met, we will ensure that each question is auto-graded to an appropriate level for a subset of randomized outputs. Finally, to show that our non-functional requirements have been met, we will take a holistic view of the system and meet with our advisor to ensure that the user experience and system properties have been provided. We will involve our client in the acceptance testing by allowing them to use our site and give feedback on each of the three above kinds of requirements. Specifically, we will meet with the client regularly after deliverables have been completed and at the termination of development.

Security Testing

For integrity, the course information is retrieved using Git, so any changes are visible in the Git history. To maintain confidentiality, we use encryption, and logins are handled using OAuth2. This allows us to use the security measures of Google and Okta in our app to protect user information. We also use HTTPS to encrypt traffic between the client and server, making it very difficult to see what the user is doing. Finally, for availability, we plan to run multiple servers behind a load balancer that can prevent our servers from being overloaded and ensure downtime is minimal. We use SSH with public key authentication and multi-factor authentication to protect the server when using a password and a firewall that only accepts traffic from SSH, HTTP, and HTTPS. We also use NGINX to reverse proxy the PrairieLearn server and enable HTTPS, encrypting traffic between

client and server. NGINX is also set up to redirect all HTTP traffic to HTTPS, ensuring encrypted traffic. Finally, all user-submitted code is compiled and run in separate docker containers isolated from the rest of the system. This means that any and all user input is sandboxed and will be unable to affect the rest of the system.

Black & White Box Testing

We performed internal and external testing to ensure that the homework and assignments met the outlined requirements. Part of our internal testing involved analyzing and running the code for each question to ensure efficiency and a standard shared format. To track our internal testing, we split our team into groups to review the code developed by the other subgroups, ensuring each group member reviewed their own and others' work. We continued internal testing until each member had reviewed all homework and questions, deeming them ready for external review.

To perform external testing, we granted access to PrairieLearn for the TAs and professors of CPR E 288. We created a feedback form for the external testers to populate with questions, comments, or concerns regarding any functional requirements that were not met. External testers were only given student-view access to the course to ensure each homework and their questions were functionally correct, and were only able to access the given course. External testing was additionally performed to ensure the login and access to the course were safe and secure. The internal and external testing cycle continued to help guarantee that the course was correctly implemented and efficiently used all available resources.

- ### Results

  From all of the testing we did, we found issues that opened our perspective to problems that we could go in and mitigate that problem and future issues that would sprout from that problem. This testing process worked very well for us and saved us a lot of headaches from running into constant problems that could have been avoided.

Broader Context

Area	Description	Examples
Public health, safety, and welfare	Our project does not have any impact on the public health, safety, or welfare of our stakeholders	Our project does not have any impact on the public health, safety, or welfare of our stakeholders
Global, cultural, and social	The project can reflect the values of the faculty creating coursework. It will also improve the learning experience of students, promoting participation.	The implementation allows the users on the faculty level to create questions with a high level of customization, allowing them to express themselves in many ways.
Environmental	Our project does not have any impact on environmental factors.	Our project does not have any impact on environmental factors.

Area	Description	Examples
Economic	This project has the ability to create jobs and provide a product to make a profit. It also trains new engineers who will join the workforce in the near future.	The product provides an opportunity for the sale of a product to generate revenue for the project owners. The development of this project also offers job opportunities for developers.

Table 1. Design Context

Conclusions

We started this project to create a learning environment for the course CPR E 288. To do this, our client gave us a framework and tasked us with creating a course in PrairieLearn that would sufficiently test student knowledge over every assignment from the original course. Over the course of this first semester we have successfully created content for nearly half of the homework assignments and set up secure production environments for the content to be hosted on.

Through the second semester, we made a large amount of progress toward completing the assignments and making them ready for student use. Multiple new technologies were implemented to create more variety and better grading for the assignments. The next group that takes on this assignment can now take this project to the next level and produce technology that makes the learning environment for 288 students even more advanced. One thing the next group could implement is an online lab environment, which would take a lot of work but would make students able to work in the lab when space in the lab is limited.

Appendix 1 - Operation Manual

Set-Up

To start with setup, you will need to have a version of Ubuntu 22.04. We have used Oracle VM virtualbox to have Ubuntu installed. Along with this, you need docker installed to use the various containers we have developed, as well as the necessary containers to run PrairieLearn itself. Along with this, having our PrairieLearn course cloned allows you to see the progress we made. To see a more detailed explanation of the setup, a link to detailed instructions can be found here**.**

A brief overview of the steps in order to the setup are as followed:

- Install Oracle VM virtual box (install link here)
- In the setup- make sure you select the type field as Linux, and select the version to be Ubuntu 22.04 LTS (Jammy Jellyfish)
- Go through the setup steps with launching the VM.
- After setup steps, install docker, this link takes you to the correct commands to input into the terminal, or simply input these commands:
    - sudo apt-get update

    - sudo apt-get install ca-certificates curl
    - sudo install -m 0755 -d /etc/apt/keyrings
    - sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
    - sudo chmod a+r /etc/apt/keyrings/docker.asc
    - echo \ “deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \ $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
    - sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
    - sudo apt-get update
- After docker is installed, you should clone the course repository
    - To use our groups sdmay24-33 CPRE288 course, click here
- There are a few different ways to run a course
    - To run an example course made by the PrairieLearn devs, run the command below
        - sudo docker run -it --rm -p 3000:3000 prairielearn/prairielearn
    - To run your own course, run this command, and replace REPLACE_ME with your directory to the course
        - sudo docker run -it --rm -p 3000:3000 -v $HOME/REPLACE_ME:/course prairielearn/prairielearn
    - To run our groups sdmay24-33’s CPRE288 course, run this command to get all necessary docker containers
        - sudo docker run -it --rm -p 3000:3000 -v /var/run/docker.sock:/var/run/docker.sock -v $HOME/pl_ag_jobs:/jobs -v $HOME/sdmay24-33:/course -e HOST_JOBS_DIR=$HOME/pl_ag_jobs prairielearn/prairielearn
        - To run this course, make sure you create a directory pl_ag_jobs in the root directory, and the sdmay24-33 folder is located in the home directory, or change the command above to locate the correct directory location of the cloned course.

Demo

To see a demo video, click here

To see our server, click here (This shows a student view if you log in through Google or Okta). You must be on the Iowa State WiFi network or VPN to access the site.

Student View:

As a student, you can see the list of homework assignments, when enrolled in the class.

- Clicking into any of the homework assignments will pull up another menu, which shows a list of all the questions within the homework assignment.
- Clicking into a desired question will allow you to work on the question.
- When you hit Save & Grade, the grading process will happen, and depending if the question is a manually or automatically graded question, the process will start.
    - If it is a manually graded question, the server will save the response for the instructor or TA to manually insert a grade.
    - If it is an auto-graded question, then you will receive your score right as you submit it.

Developer View:

As a Developer, If you are trying to make changes to the course, you will need to:

- locally run PrairieLearn with the setup steps mentioned above.
- After running your course, and going to http://localhost:3000, you will need to hit the load from disk button on the top right corner. This will load the course you listed in the directory, through the PrairieLearn docker container.
- Clicking into the course will bring you to the Course Instances, where you can either access the course where Assignments are listed as a whole, or click into a different tab.
- If you are trying to make a new question, you go into the Questions tab on the top.
- Here, all questions made are listed, and if you want to make your own question, simply click + Add Question.
- When making a new question, you want to change the QID to the way you desire to name the question, and edit the info.json.
    - In this info.json, you can change the Title to add a title for the question (We usually have that the same as the QID), and add tags which adds a tag to the question, such as who developed the question, or when the question was starting to be developed.
- Changing to the Files tab is where you can customize the question as desired.
- The question.html file is where you write the HTML code that allows you to add elements which appear on the site visually for the students to see.
- The server.py file is where you write the scripts to randomize questions, autograde questions, and add any other necessary logic the question needs to operate.

Testing

To test your questions, PrairieLearn lets developers get a preview of the question, by clicking the Preview tab. Here you can see what everything looks like, and hitting Save & Grade will allow you to see the output just as students would. Here, you will also get debugging errors, in any of the json, html, or python files. On top of this, if these errors don't let the question pop up, then the error will display and be put into the Issues tab, with a log of what question, as well as the error message for you to debug. Past this, peer reviews to try and break or find errors in questions is another vital testing point in order to get a question working as desired.

Appendix 2 - Alternative/Initial Version of Design

- Design 0 (Initial Design)

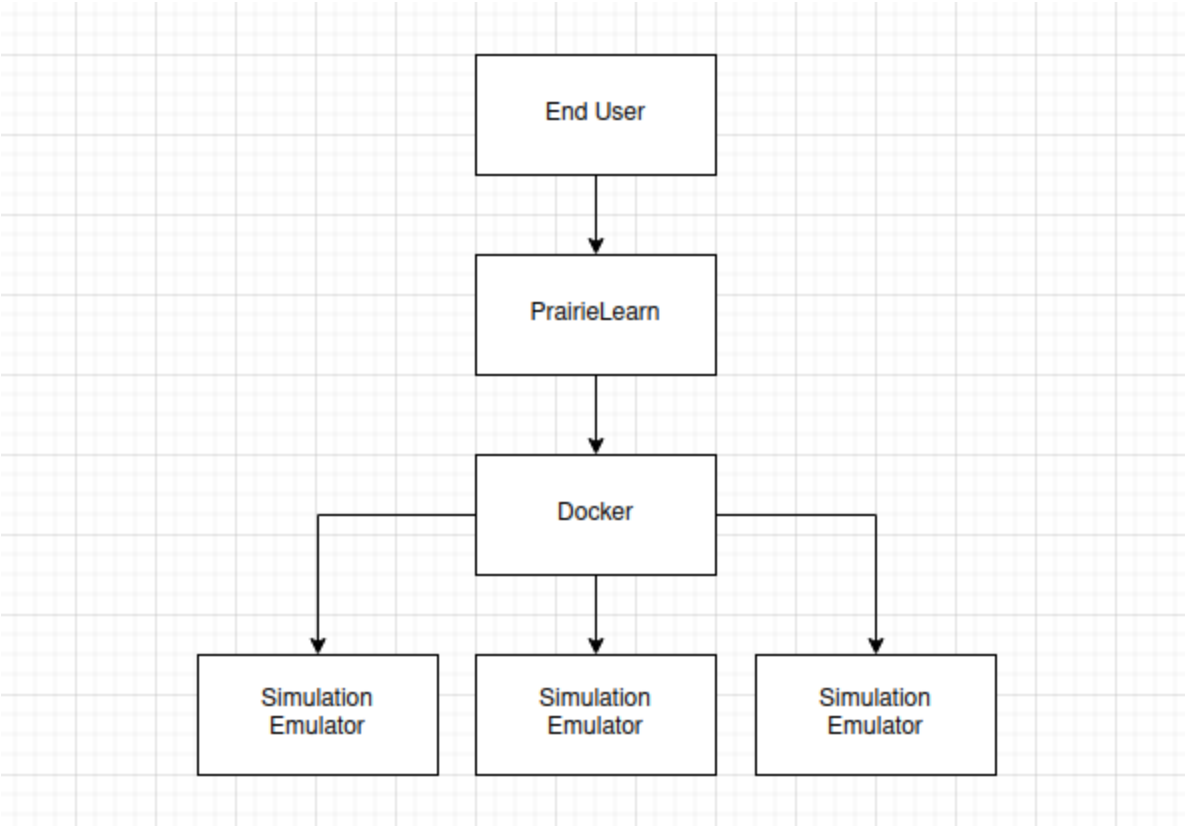

Figure 2. Initial Design Diagram

The initial design uses PrairieLearn as the interface between the simulations and the end user. PrairieLearn uses Docker containers for each question and the code given by the user is put into the emulator / CyBot simulation. To implement this, we will utilize the existing PrairieLearn framework to create questions through the UI and built-in text editors using JavaScript and Python. These questions are then published to students through the framework's implementation. We will then source an emulator and make any necessary modifications to integrate it onto the PrairieLean site. This emulator will simulate microcontrollers and allow questions to be made that utilize this. The emulator will aim to allow students to upload their code and run it on what would be the microcontroller and see its behavior. Our design is intended to create a better learning experience for students throughout CPR E 288. The questions and auto grader create a more engaging learning experience that will help grasp lecture material. The emulator then helps students in the lab, making it easier and faster to test code without needing a physical CyBot and testing space. Both are used increasingly throughout the course as the labs and lecture material become more complex. The current design satisfies these functional requirements as it lays out a plan to finish the auto grader, create more questions for students, and create an improved emulator for the lab.

This design was revised to better account for the design complexity added by the server and autograder. What led to this iteration was a more thorough understanding of the requirements for PrairieLearn and research into the design of production servers. The significant changes from Design 0 is the addition of security features. These include the firewall, Nginx, and SSH security changes.

Appendix 3 - Other Considerations

This project presented many challenges and learning experiences for us. We learned many things about developing code in a collaborative and innovative environment, especially for those of us who had limited coding experiences.

Appendix 4 - Code

PrairieLearn Course Code:
https://git.ece.iastate.edu/sd/sdmay24-33

Edited PrairieLearn Code with Okta Integration:
https://github.com/myriath/PrairieLearn

Original PrairieLearn Repository:
https://github.com/PrairieLearn/PrairieLearn

ARM Assembly Auto-grader Docker Image Source:
https://github.com/myriath/PrairieLearnARMGrader

QEMU Tiva TM4C123GH6PM Board Implementation:
https://github.com/myriath/qemuTIVA