PrairieLearn Implementation for CPR E 288

Design Document

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: 11/30/23 Version 2

Executive Summary

Development Standards & Practices Used

- IEEE 610 Standard Glossary of Software Engineering Terminology
- IEEE 830 Software Requirements Specifications
- IEEE 1016 Software Design Description
- IEEE 1074 Software Development Life Cycle
- IEEE 2050 RTOS for embedded systems standard

Summary of 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 emulator for running student bot code in a virtual environment

Applicable Courses from Iowa State University Curriculum

- CPR E 288

New Skills/Knowledge acquired that was not taught in courses

- Python / NodeJS / HTML / C Programming
- PrairieLearn Framework
- Okta SSO, OAuth2, LTI Integration
- Server Management / Security
- Git Integration

Table of Contents

List of Figures

List of Tables

List of Symbols

SSO	Single-Sign On
JS	JavaScript
PL	PrairieLearn
CPR E 288	Computer Engineering 288
ISU	Iowa State University
CyBot	OpenConnect Roomba used by students
UFW	Uncomplicated Firewall (Firewall software)

1 Team

1.1 Team Members

- Carter Murawski
- Chris Costa
- Matt Graham
- Tyler Weberski
- William Hudson
- Andrew Winters

1.2 Required Skill Sets for Your Project

Embedded Systems Programming

Python/JS/C Programming Language

1.3 Skill Sets covered by the Team

Embedded Systems Programming - Carter Murawski, Tyler Weberski, Chris Costa, Matt Graham

Python - Mitch Hudson, Carter Murawski, Matt Graham, Chris Costa

JavaScript - Mitch Hudson, Tyler Weberski, Matt Graham, Chris Costa

C - Mitch Hudson, Carter Murawski, Andrew Winters, Tyler Weberski, Matt Graham, Chris Costa

1.4 Project Management Style Adopted by the team

GitLab

1.5 Initial Project Management Roles

Project Lead(Tyler Weberski): Coordination, Scheduling, assembly, and delivery of products

Scrum Master(Carter Murawski): Documents, Organizes, and tracks development.

Domain Expert(Mitch Hudson): Discover and explain what the team needs to know about the application area, context, and adjacent systems

Quality Assurance(Chris Costa): ensure that delivered products accurately and completely describe/satisfy the requirements

Consultant(Matthew Graham): Advises, strategizes, and plans for the development of the product

Construction(Andrew Winters): Oversees the implementation and execution of the project

2 Introduction

2.1 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 questions that are customizable and randomized, where students can answer the questions, and have them be autograded once submitted.

2.2 Requirements & Constraints

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 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

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

2.3 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 key in order to speed up certain 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, as well as 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 key for the flow of the project. Although we don't have a specific lifecycle model set yet, it will be a decision made soon in order 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 success for the team.

2.4 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.

3 Project Plan

3.1 Task Decomposition

First, we need to create questions and assignments in the PrairieLearn development environment that can be auto graded and used by students. Then, once we have our course content, we can create and integrate an emulator for testing labs.

Once the course content is completed, we will set up a production server through the university that is accessible by students. This server will need to be secured and configured.

Finally, we will test the system against our performance and non-functional requirements and ensure everything is in order.

Create questions for homework assignments / labs / quizzes / exams

- Going through the course material and create questions to match course content
    - Binary, Decimal, Hex conversion
    - Digital logic
    - Clock functionality
    - Register bit initializing
    - Datasheet questions
    - C and assembly coding questions
- Set up auto grader for compiling code and testing it
    - Research PrairieLearn auto grading functionality
    - Test with basic programming questions
    - Once tested, implement it with real questions

Create / Integrate CyBot emulator for

- Create an emulator for the CyBot and microcontroller, that will take in uploaded student code and show the results of running the code.
    - Research the TIVA microcontroller and possible emulator solutions
    - If needed, create C code for emulating the TIVA microcontroller
    - Test microcontroller emulation
- This will have to be connected to the auto grader docker environment because the code needs to be compiled
    - Create environment to auto grade
    - Connect student code to emulator
    - Test student code using the emulator

Implement Okta SSO

- PrairieLearn by default does not support Okta as a login method.
- We need to do research and develop a method for PrairieLearn to use Okta
    - Research Okta and OAuth2 login flow
    - Create a basic Okta login program and test outside of PrairieLearn
    - Create fork of PrairieLearn repository
        - See how PrairieLearn uses other login methods
        - Update the codebase with the new method

    - Test the new code with Okta development server
    - Once working with dev server, test with ISU login servers

Create Production Server

- Have the university create a virtual machine
- Secure the virtual machine with public key authentication and multi-factor authentication
- Install required software (Git, PrairieLearn, Nginx, UFW, Docker)
- Configure PrairieLearn
    - Set up config.json to include login parameters
    - Set up course syncing with Git
    - Configure auto grader Docker connection
- Configure UFW
    - Allow HTTPS, HTTP, and SSH
    - Disallow all other incoming traffic
- Get SSL certificates through the university
- Configure Nginx
    - Set up HTTPS server
        - Reverse proxy to the PrairieLearn port
        - Connect to SSL certificates from university
    - Set up HTTP server to redirect all traffic to HTTPS port

Test Production Server

- Test security
- Test under load
    - Make sure the server fits performance requirements
    - Check to see if multiple servers are required
- (If needed) Set up load balancer and multiple PrairieLearn servers

3.2 Project Management/Tracking Procedures

We will be using an agile project management style due to this style being non-linear and incremental. This style fits our project as our goals are determined by decisions made throughout the development process. Many decisions that we make regarding what is to be done are dependent on previous tasks that are completed; however, this process is not linear. We will prioritize the development of different features based on what we feel is the most important at that time. For example, a stretch goal of our project is to have microcontroller emulators built into the PrairieLearn framework that can be compatible with the autograder. If we are able to achieve this goal. then we are able to continue incrementally building in compatibility with physical microcontrollers in addition to the emulators. This process of microcontrollers and emulators can be done before, during, or after the development of a better auto grader or different question types.

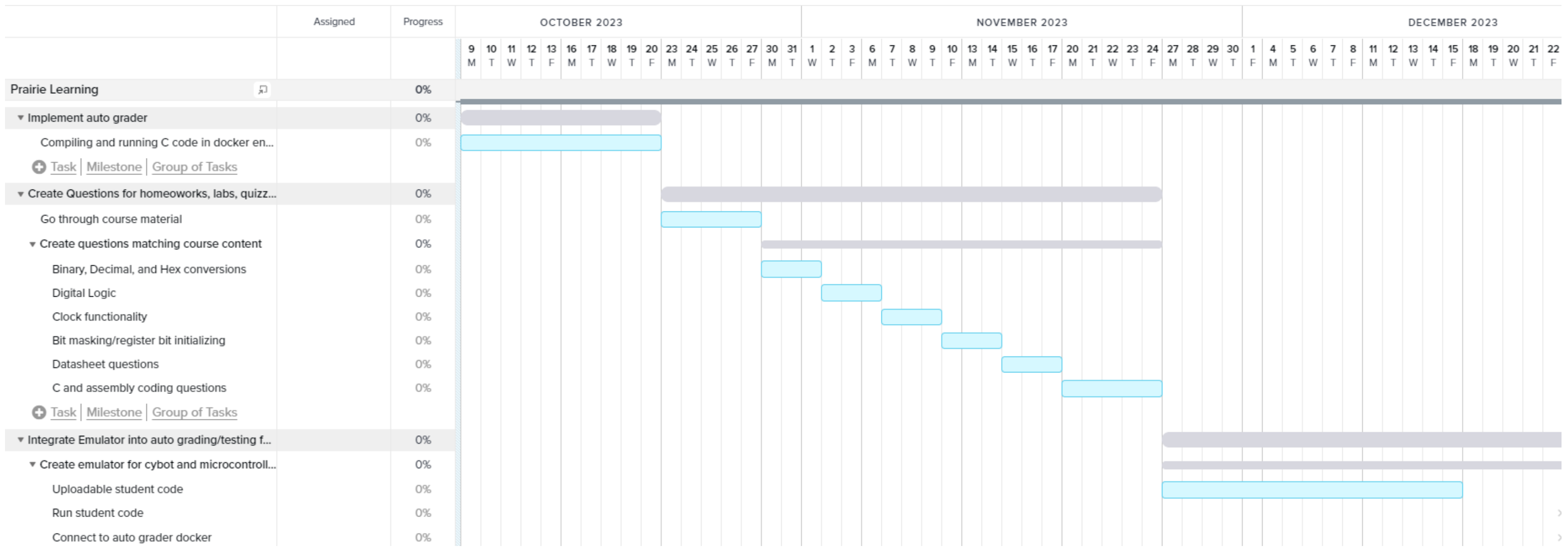
We will be using Git through GitLab to track progress, which includes their issue boards. We will also do collaboration through git merge requests and an external Discord server for communication.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

- Improve Auto Grader
    - The Auto Grader must take in questions and output students score with 100% accuracy.

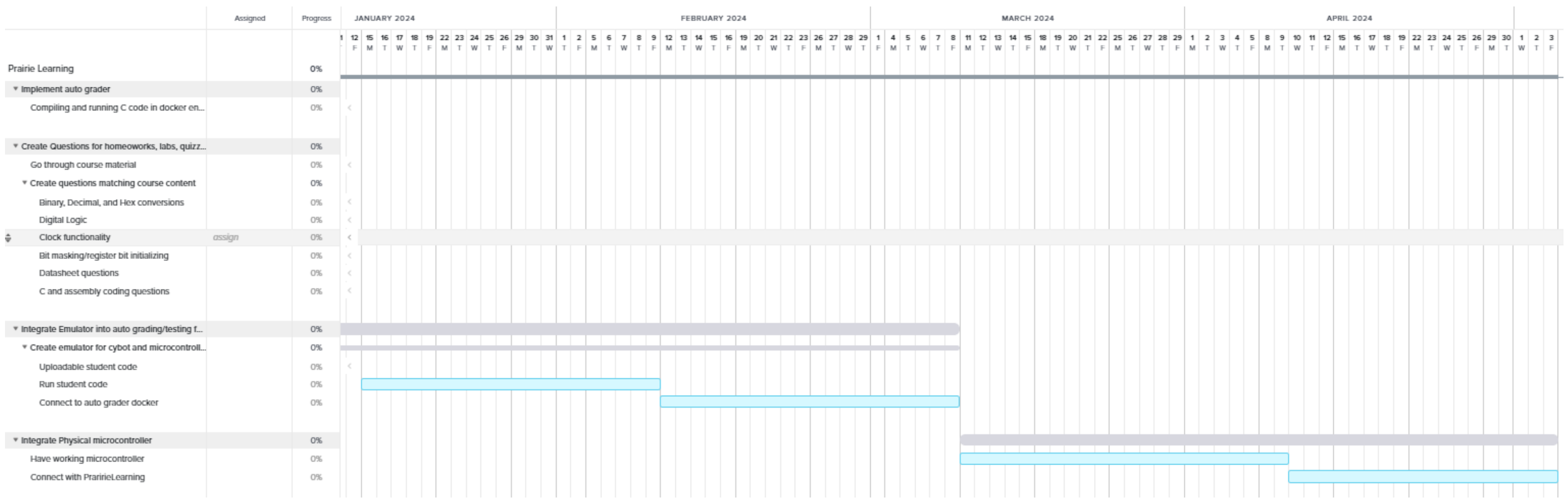
- Interactive Question Development
    - Develop 5-10 new interactive question formats that can be easily changed for each assignment that integrate with the auto grader.
    - Develop a randomization function that can change numbers and other specific parts of a question while keeping the same format. This will integrate with the auto grader with 100% accuracy.
- Implement Emulator
    - Create an Emulator that can take in user code and simulate the real world Cybot.
    - Simulate the driving functionality with 95% accuracy.
    - Simulate the object detection functionality with 80% accuracy
- Create Microcontroller
    - Have a working microcontroller that allows user interaction. Further functionality will be determined as the project progresses.

3.4 Project Timeline/Schedule



	Assigned	Progress
Prairie Learning		0%
▾ Implement auto grader		0%
Compiling and running C code in docker en...		0%
Task | Milestone | Group of Tasks		
▾ Create Questions for homeoworks, labs, quizz...		0%
Go through course material		0%
▾ Create questions matching course content		0%
Binary, Decimal, and Hex conversions		0%
Digital Logic		0%
Clock functionality		0%
Bit masking/register bit initializing		0%
Datasheet questions		0%
C and assembly coding questions		0%
Task | Milestone | Group of Tasks		
▾ Integrate Emulator into auto grading/testing f...		0%
▾ Create emulator for cybot and microcontroll...		0%
Uploadable student code		0%
Run student code		0%
Connect to auto grader docker		0%

Figure 1. First Semester Timeline



	Assigned	Progress
Prairie Learning		0%
▾ Implement auto grader		0%
Compiling and running C code in docker en...		0%
▾ Create Questions for homeoworks, labs, quizz...		0%
Go through course material		0%
▾ Create questions matching course content		0%
Binary, Decimal, and Hex conversions		0%
Digital Logic		0%
Clock functionality	assign	0%
Bit masking/register bit initializing		0%
Datasheet questions		0%
C and assembly coding questions		0%
▾ Integrate Emulator into auto grading/testing f...		0%
▾ Create emulator for cybot and microcontroll...		0%
Uploadable student code		0%
Run student code		0%
Connect to auto grader docker		0%
▾ Integrate Physical microcontroller		0%
Have working microcontroller		0%
Connect with PrairieLearning		0%

Figure 2. Second Semester Timeline

3.5 Risks And Risk Management/Mitigation

Our project is entirely software based, which means the only risks we need to contend with are performance targets and 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 is going to 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 auto grading. 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 for tools not meeting expectations is probably around a .4 considering no one in the team 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.

3.6 Personnel Effort Requirements

Task	**Projected Effort (High/Medium/Low)**	**Total Person Hours**
Improve Auto Grader	Medium	30
Create Questions for Assignments/labs/quizzes/exams	Medium	60
Integrate Emulator	High	120

Task	**Projected Effort (High/Medium/Low)**	**Total Person Hours**
Integrate Physical Microcontrollers	High	180

Table 1. Personnel Effort Requirements

The first task to be completed is to improve the existing autograder through the PrairieLearn framework. This is a medium effort task to complete because the auto grader is already implemented in the system, but needs various improvements such as allowing for more grading and automating options. This will take about 20 hours of work to complete and test in the system. The next task is to create questions for multiple assignments, quizzes, labs, and exams for students to utilize. This is a medium effort task for the team because with course material, questions are already created and just need to be implemented. Some assignments are completed from the previous group, but can be improved with more variability and better grading. Task 2 will take about 60 hours of total time to complete and include all questions available through the course material. Task 3 involves integrating an emulator for the Cybot and microcontroller. Implementing this requires taking students' code and giving the results of the running code. This is a high effort task because it involves creating an emulator and configuring it to the required needs. Task 3 also has an estimated time of 120 hours for completion because of the large amount of development and testing that will be involved. The final task is to integrate Physical Microcontrollers. The goal is to have a working microcontroller that will allow users to interact with. This is a high effort level because it involves configuring and connecting a microcontroller to work with PrairieLearn. This task will take the most time because it will involve a large amount of development and testing to be completed.

3.7 Other Resource Requirements

Other potential resources we may need for our project are the CPR E 288 course material, example questions, access to the entire PrairieLearn framework, and access to the microcontrollers used in CPR E 288.

4 Design

4.1 Design Content

We are designing many new interactive questions that will test students' knowledge in more interesting ways. These questions must be creative and test students in new ways beyond simple multiple choice or fill in the blank.

We will also design an emulator to simulate the CyBot in 288. The emulator must accurately represent the real world so that students can more easily simulate their code without needing a physical bot and test space.

4.2 Design Complexity

For this project, there are a number of components that work together. One component is the PrairieLearn framework and course content. We need to create content that tests students' understanding of the course material. Along with that, we need to create an autograder that runs in Docker containers. The autograder needs to be able to compile the C code given by students, and determine the output. This will require an understanding of C and Docker to create robust software following many engineering standards. Another piece is implementing an emulator for the microcontroller used in class, and connecting it to a simulated CyBot. These simulations need to accurately represent what would happen if the code was run on physical hardware. The simulation should also allow the students to see a representation of each lab and the CyBot itself. This will require the combination of computer, electrical, and software engineering knowledge.

4.3 Modern Engineering Tools

The modern engineering tools we used for this design are Git/Gitlab and Ubuntu. We utilize git for our project in order to track changes and to have version control for all additions that are made. We will make use of Gitlab to track progress and keep a backlog and activity board to measure tasks and progress. VirtualBox and Ubuntu are tools used to create a virtual environment in order to run PrairieLearn.

4.4 Design 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 in order to generate revenue for the owners of the project. The development of this project also offers job opportunities for developers.

Table 2. Design Context

4.5 Prior Work/Solutions

A lot of similar products are in place for the autograder, such as Renaissance, MasteryConnect, and Gradescope, just to name a few of the bigger ones. These groups have assessment problems which teachers/professors can create questions and set the answers for students to fill out. This would include homeworks or exams, with the option for manually graded questions for long response. These competitors, however, do not have any form of emulator built in for microcontrollers, something that can set PrairieLearn apart.

We are following previous work, and the advantages we will gather are having a few items to work off of and using previous documentation to help us get started. The previous group had good documentation on the basic autograder function and made videos guiding us through setup. The shortcomings are that there wasn't as much done in the further development from the group last year, so past the initial autograder fixes we are doing, they have nothing we can build off of there.

A few pros for our solution would be help for the CPR E 288 class and an emulator that will allow students to have better access to the course labs and testing, which was a bulk of the work and learning in the class. This emulator will allow students to work from home without having to have a microcontroller with them physically. Some sections of CPR E 288 had an emulator in the past. The previous emulators used were not very user-friendly and are in need of a replacement, something we believe PrairieLearn has the ability to improve upon. This also gives students different ways to learn with more interactive questions to really help with understanding of different concepts. Doing more problems in different ways can make students succeed more and more.

A few cons for our solution come more from the autograder side, and the competition having UI that people may prefer more. We also know that these other products are more user-friendly and well-respected than the PrairieLearn framework. We can also see that the autograder must be independent for each question due to how the framework is structured, something that limits the ease of implementation. Another con is user friendliness in general, getting access to PrairieLearn is a little difficult and might be a hassle for some students.

4.6 Design Decisions

- We are going to use PrairieLearn as the framework for distributing course content
- We will use Docker containers to compile and run code for the autograder
- We will use Git and GitLab to keep track of versioning
- We will use a campus server for the final production server

4.7 Proposed Design

We have all set up the PrairieLearn environment which we will be using to have questions and the autograder working. In this we have implemented a few different forms of questions including multiple choice and fill in the blank, which would then be autograded for the correct response. These questions have randomized numbers in order for students to get slightly different questions, but still keep the concept the same.

4.7.1 Design 0 (Initial Design)

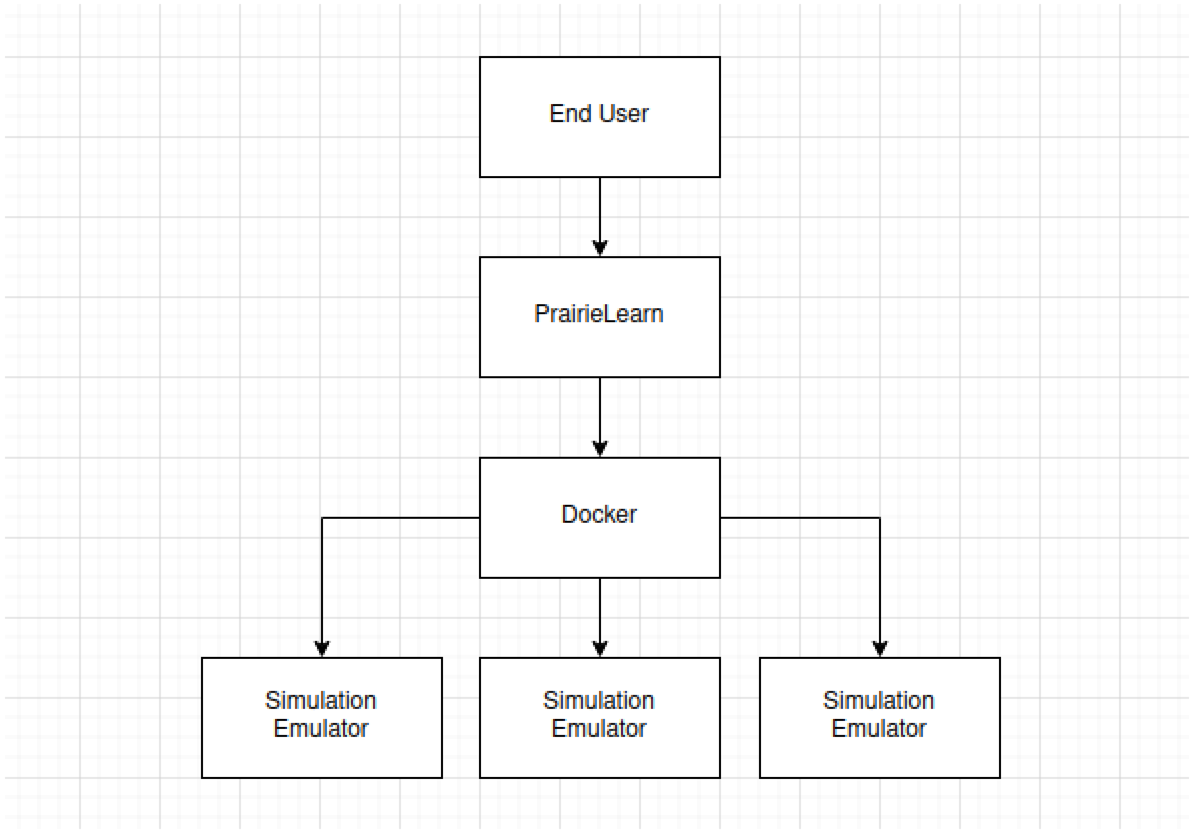


Figure 3. Initial Design Diagram

The current 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 in order 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 in order to integrate it onto the PrairieLean site. This emulator will be used to 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.

Functionality

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 as it makes it easier and faster to test code without needing a physical CyBot and testing space. Both of these are used increasingly throughout the course as the labs and lecture material becomes more complex.

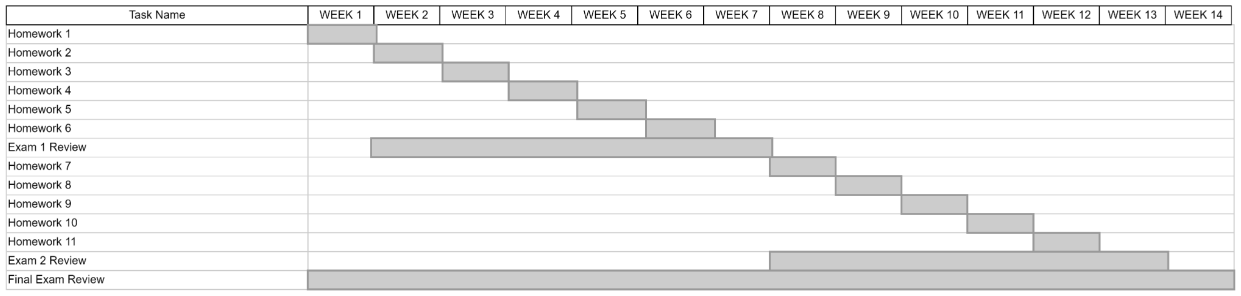


Figure 4. Functionality Gantt Chart

The current design satisfies these functional requirements as it lays out a plan to finish the auto grader, create more questions for students, and then create an improved emulator to be used in the lab.

4.7.2 Design 1 (Design Iteration)

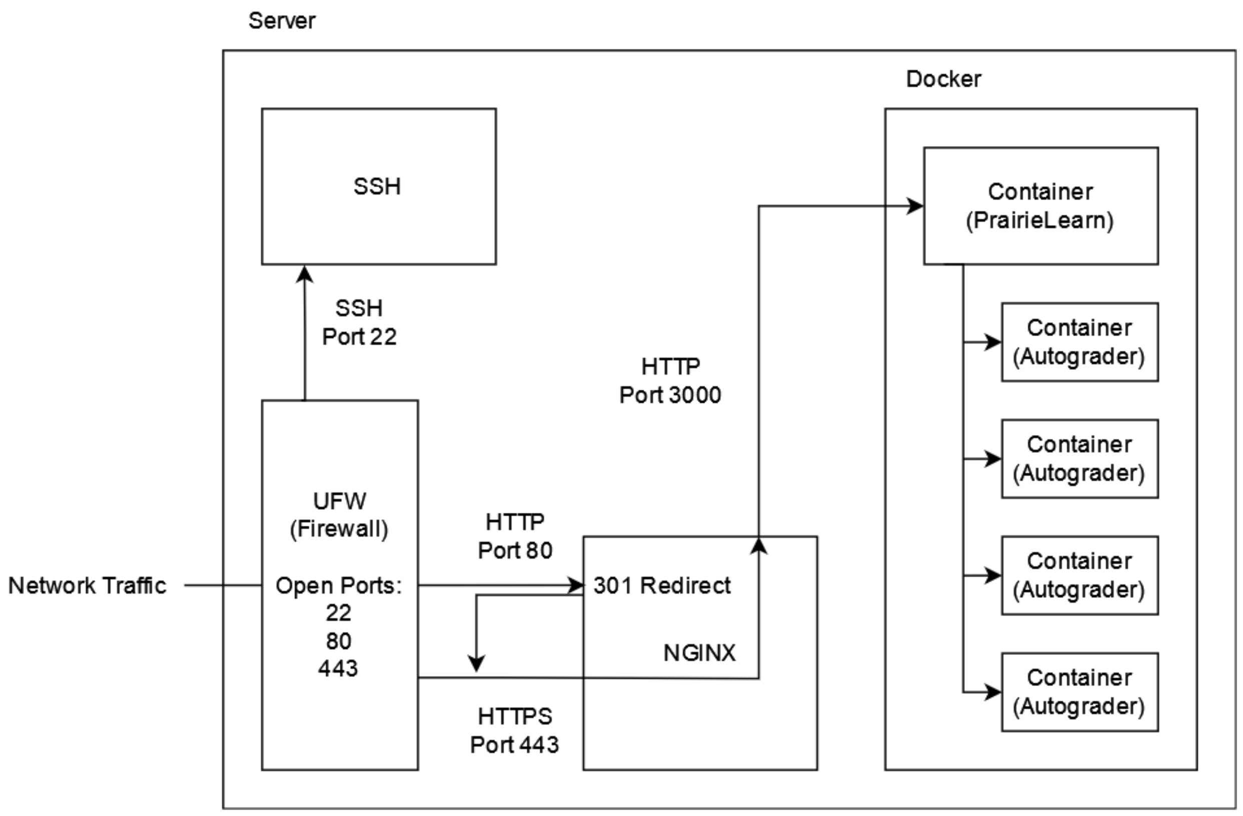


Figure 5. Design Iteration Diagram

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

What led to this iteration was a more thorough understanding of the requirements for PrairieLearn as well as research into the design of production servers. The major changes from Design 0 are 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. To protect this important part of the server, we added multi-factor authentication and public key authentication. These changes fulfill the security requirements related to connecting and accessing server resources.

Finally, 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 at the same time, helping with handling multiple users at once, 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 in to PrairieLearn, such as organizing sections, creating / modifying questions, and displaying pages in visually appealing ways.

4.8 Technology Considerations

There are many strengths for the technologies we have decided on. UFW is a simple and effective firewall program that comes built-in with our operating system, and the SSH security configuration offers a strong balance of security and user-friendliness. Docker is an extremely efficient way of creating sandboxed environments without much overhead, improving security and efficiency. Finally, NGINX is a well-known and actively maintained web server program, which neatly slots into our web stack.

There are only a few weaknesses with our design, and they mostly come from the tradeoffs made in our SSH configuration. Due to the need for easy access from new computers put forward by our advisor, we still allow password authentication, which is much less secure than public key authentication alone. We have compromised by adding multi-factor authentication, but SSH still sends passwords in tunneled plaintext, which can be quite dangerous.

Some alternatives we can look into are switching out NGINX for a different reverse proxy, such as Apache, and running PrairieLearn natively instead of in a Docker container. The docker container is much more difficult to update with our own code, but stays up to date with PrairieLearn's official repository much more readily. Running PrairieLearn natively can solve the issue of running our own code, but can make it more cumbersome to update it with the official repository.

4.9 Design Analysis

Our design proposed in 4.7.1 was fairly basic and undefined due to our lack of knowledge surrounding PrairieLearn and its requirements. Since then, we have revised our design and updated it to what it is now in 4.7.2. We have also started implementing a server for production. During this implementation we ran into several issues and have refined our design to resolve them. These include security requirements, and the introduction of a firewall and Nginx for handling incoming requests. Making these changes led us to updating our design diagram and improving it to what is seen in 4.7.2.

The design proposed in 4.7.2 is promising and seems to be handling what we need it to, at least during development. The design is able to keep up with the load of developers, but it remains to be seen if it will be enough to handle the students in the future. Our security testing has also shown the design to be fairly robust and secure.

In the future, we will iterate on the design and improve efficiencies, as well as implement quality of life improvements, such as integration to Canvas and ISU's Okta Single Sign-On (SSO) servers. Adding Canvas integration will allow grades from PrairieLearn to automatically be reflected in the gradebook used by the University, updating students' grades immediately. Connecting our app to ISU's Okta SSO allows users to log in with the same account used for every other ISU app, including Canvas, Microsoft Office, and AccessPlus. It also keeps track of sessions, so if you've signed into Canvas, it will automatically sign you in to every other Okta app as well, such as PrairieLearn.

5 Testing

5.1 Unit Testing

For our project, the ‘units’ being tested are the individual questions. These questions are tested within PrairieLearn where they are compiled and previewed. The built-in grading system allows us to test and ensure that the questions are working as intended. Most of the tools used are built into the PrairieLearn framework as files and questions can be edited directly through the web interface. The other way we are testing our units is by showing them to our advisor at each weekly meeting to make sure 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 are the auto grader Docker containers. Since each question will create a new container to auto grade, we can test that the grading is working 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.

5.2 Interface Testing

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

5.3 Integration Testing

The most important connection paths we have is making sure each individual question works on its own, and can be integrated with no issues into one big assignment, so that the students going through the homework can easily finish it. These will be tested by individually going through the Unit testing for each question, and ensuring proper formatting there. After that, we will go through the interface testing and connect the questions into 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.s

5.4 System Testing

The system can be tested by testing homework assignments as a whole and making sure they all work properly each time a question is attempted. This will ensure that when students go to attempt the homeworks that it all works properly and their learning time will be maximized. A lot of the questions within the homework are randomized so testing the homework assignments multiple times will be beneficial to make sure that random value in a question doesn’t mess up the student.

5.5 Regression Testing

For regression testing, we will ensure that any new addition will be contained in its own 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 in order to ensure that PrairieLearn will continue to run is to have every component of the system 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 because it is a requirement driven by PrairieLearn requirements

5.6 Acceptance Testing

In order to demonstrate the design requirements were met, we will ensure 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 go through each of our courses, assessments, and questions to ensure that 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, in order 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 on a regular basis after deliverables have been completed as well as at the termination of development.

5.7 Security Testing

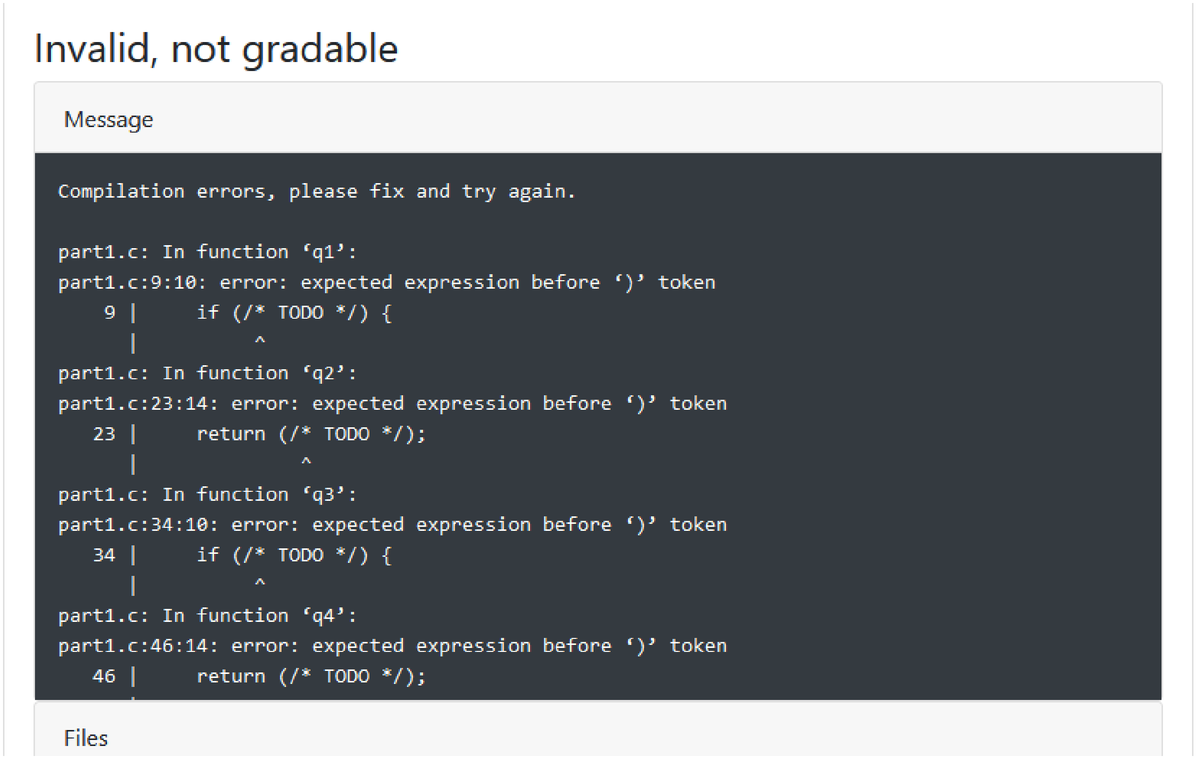
A suggestion from our advisor was to give the production server to one of the cybersecurity classes for them to attempt to break in. We plan to use this to both give the cybersecurity students an interesting project and also test our security measures.

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.

To protect the server, we use SSH with public key authentication and multi factor authentication when using a password, as well as 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 that are 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.

5.8 Results



Invalid, not gradable

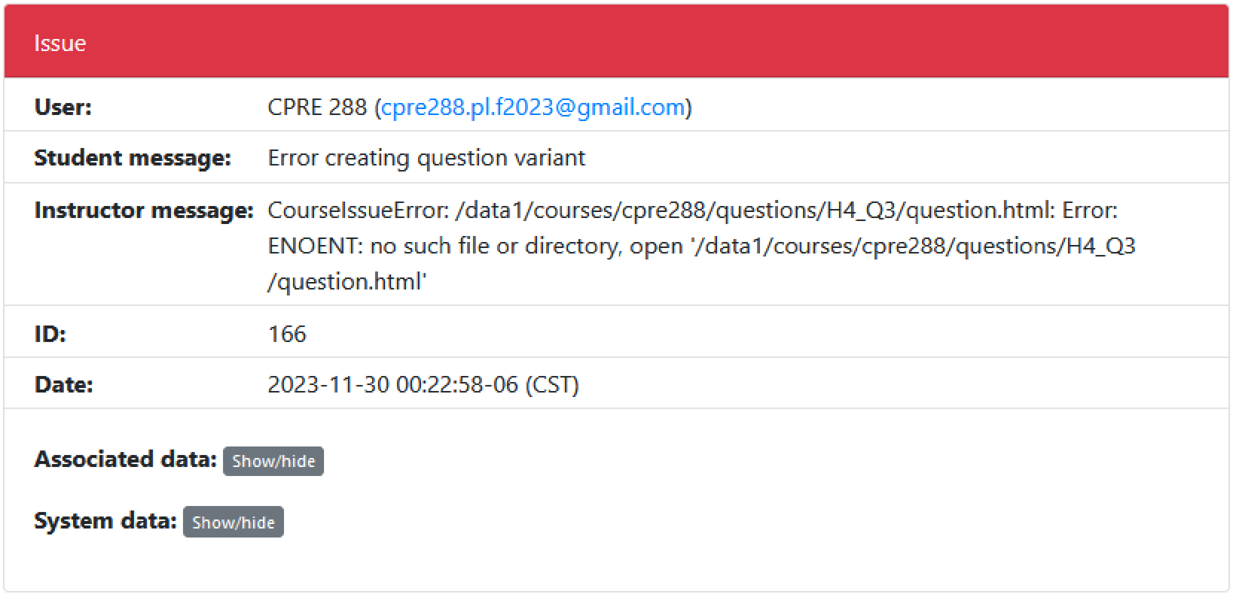
Message

```
Compilation errors, please fix and try again.

part1.c: In function ‘q1’:
part1.c:9:10: error: expected expression before ‘)’ token
    9 |     if (/* TODO */) {
      |          ^
part1.c: In function ‘q2’:
part1.c:23:14: error: expected expression before ‘)’ token
   23 |     return (/* TODO */);
      |              ^
part1.c: In function ‘q3’:
part1.c:34:10: error: expected expression before ‘)’ token
   34 |     if (/* TODO */) {
      |          ^
part1.c: In function ‘q4’:
part1.c:46:14: error: expected expression before ‘)’ token
   46 |     return (/* TODO */);
```

Files

Figure 6. PrairieLearn reporting on compilation errors in question code.



Issue

User:	CPRE 288 (cpre288.pl.f2023@gmail.com)
Student message:	Error creating question variant
Instructor message:	CourseIssueError: /data1/courses/cpre288/questions/H4_Q3/question.html: Error: ENOENT: no such file or directory, open '/data1/courses/cpre288/questions/H4_Q3/question.html'
ID:	166
Date:	2023-11-30 00:22:58-06 (CST)

Associated data: Show/hide

System data: Show/hide

Figure 7. PrairieLearn issue page with a broken question.

For questions that have been developed so far, the Prairie Learn framework has been able to catch all mistakes in code as they are compiled. Our requirements are that the questions compile, display, and auto-grade. All of these are implemented within the PL environment. For non-software related testing, our advisor/client has helped catch any mistakes in formatting or material content and has guided us on finalizing questions as desired.

As seen in Figure 6, PrairieLearn will report compile errors in code and log errors whenever a course question or user-submitted code is broken. In Figure 7 you can see when a question creates an error, PrairieLearn automatically creates an issue and notifies the instructor and administrators. This makes it extremely easy to keep track of issues and test new questions.

6 Implementation

Our implementation plan starts with the creation of several virtual machine servers to run PrairieLearn, and a load balancer. On each of the PrairieLearn servers, we will apply our image from Design 1, and get all of them up and running. We will also set up the load balancer to direct traffic between the PrairieLearn servers. Finally, we will implement each question in the course Git repository, and create a shared database server for each PrairieLearn server to access. The shared database server will allow every PrairieLearn server to be connected with the same user database.

To set up a server we first need to get the university to create a virtual machine for each one. This virtual machine will be using Ubuntu 22.04 LTS as that is what PrairieLearn specifies it is made to work with. Then, we need to ask the university for a signed SSL certificate so that students aren't greeted with warnings whenever they access the website.

Once we have the certificate and virtual machine, we can set up our server. First, we will set up SSH and the firewall. The firewall will only allow port 22 (SSH), 80 (HTTP), and 443 (HTTPS). SSH will be set up to use multi-factor authentication and public key authentication. Then, we will install the requisite software for each server. This includes Nginx (a reverse proxy software), Docker (a virtualization program), and PrairieLearn itself (installed through Git). To configure Nginx, we will set up the port 80 server to redirect to port 443, forcing users to use HTTPS, and set up the port 443 server to be forwarded to our PrairieLearn server running on port 3000. We will also set up the 443 server to use our signed certificates we got from the university. Finally, setting up PrairieLearn only requires running a startup script, creating an account, and adding a course that is linked to our GitLab repository.

Once each server is set up, we will set up a load balancing server and a database server. The load balancing server will act as the entrypoint into the network for our users. It will direct traffic and keep track of how many users are connecting to each PrairieLearn server. This maintains as low of a load as possible on each server, improving performance. Finally, the database server will hold a PostgreSQL database that each PrairieLearn server connects to. This database stores information like users, grades, and other PrairieLearn background data. A singular database is important for keeping each PrairieLearn server in sync.

The last step for our implementation is to upload the course content we have been creating throughout the semester to the production server and begin serving it to students. Once this step is completed, we will have a fully functioning instance that can be used by students and instructors to create comprehensive course material.

7 Professionalism

7.1 Areas of Responsibility

Area of Responsibility	**IEEE-CS/ACM Code of Ethics [1]**
Work Competence	"2.01. Provide service in their areas of competence, being honest and forthright about any limitations of their experience and education." This code addresses work competence by insisting workers work within their areas of competence and being forthcoming about limitations. This differs from the NSPE Canon by going into more detail about what constitutes deceptive acts and what to do to avoid them.
Financial Responsibility	"5.05. Ensure realistic quantitative estimates of cost, scheduling, personnel, quality and outcomes on any project on which they work or propose to work, and provide an uncertainty assessment of these estimates." This code addresses financial responsibility by insisting followers ensure the outcomes of their products follow realistic expectations in terms of costs. It also insists that followers provide uncertainty assessments for any estimates. This differs from the NSPE Canon by going into more detail about quantitative estimates and financial responsibility.
Communication Honesty	"1.06. Be fair and avoid deception in all statements, particularly public ones, concerning software or related documents, methods and tools." This code addresses communication honesty by directly enforcing fair and honest depiction in all statements, especially ones directed at the public. This differs from NSPE Canon by being more detailed about avoiding deception and applies these rules to all communication, not just public statements.
Health, Safety, Well-Being	"1.03. Approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good." This code addresses health, safety, and well-being by describing how approved software should be tested and ensured that the outcome would be to the public good. This differs from NSPE Canon by insisting that checks for safety, health, and well-being are done before the software is even approved.

Area of Responsibility	**IEEE-CS/ACM Code of Ethics [1]**
Property Ownership	"2.03. Use the property of a client or employer only in ways properly authorized, and with the client's or employer's knowledge and consent." This code addresses property ownership by insisting its followers only use property in authorized ways. This means that employees should acquire consent and the knowledge of the employer before using any property. This differs from NSPE Canon by further detailing the relationship between employee and employer.
Sustainability	"1.04. Disclose to appropriate persons or authorities any actual or potential danger to the user, the public, or the environment, that they reasonably believe to be associated with software or related documents." This code addresses sustainability by explicitly mentioning the environment when addressing health and safety requirements. This differs from NSPE Canon by mentioning the environment whereas the Canon doesn't.
Social Responsibility	"Software engineers shall act consistently with the public interest. In particular, software engineers shall, as appropriate: …" This code addresses social responsibility by explicitly calling for engineers to act in accordance with public interest and society as a whole. This differs from NSPE Canon by describing different situations in which software engineers in particular handle social responsibility.

Table 3. Areas of responsibility IEEE-CS/ACM CoE

7.2 Project Specific Professional Responsibility Areas

Area of Responsibility	**Applies to us?**	**Team Performance**	**Justification**
Work Competence	Yes, because we need to provide a service competently to the students of ISU.	High	We are working hard within our means to produce a meaningful product that will improve the course and student learning.
Financial Responsibility	No, because our project is entirely run using free resources.	N/A	We are not paying for any part of the project. Finances do not affect our project.

Area of Responsibility	**Applies to us?**	**Team Performance**	**Justification**
Communication Honesty	Yes, we need to communicate honestly with students and the advisor.	High	We have never lied or otherwise been dishonest to the clients or public.
Health, Safety, Well-Being	No, our project doesn't affect the health and safety of anyone.	N/A	Our project does not impact the physical world in any meaningful way.
Property Ownership	Yes, we are using the property of several organizations and must be respectful of that.	High	We are using PrairieLearn within its license's use cases. We are also being respectful of the computing resources our servers borrow from the university by keeping them secure.
Sustainability	Yes, our project uses electricity for running servers and software, so we need to be mindful of its impact.	Low	Due to the small footprint of our project, we have not looked into the environmental impacts of our project very much.
Social Responsibility	Yes, our project seeks to improve the lives of ISU students and instructors.	Medium	Our project focuses on a small niche of the university, only affecting one course's students and instructors. This makes it difficult to track how much we improve the public good.

Table 4. Project specific responsibility areas

7.3 Most Applicable Professional Responsibility Area

The most applicable professional responsibility area for our project is the work competence responsibility. As our project will be affecting students and instructors in future sections of CPR E 288, we need to ensure our project is up to standards. We also need to ensure our project will fulfill requirements put forth by our advisor and the course's original intent. If our work does not meet standards it will become useless for the client and have wasted everyone's time. Thus, we need to keep our standards high and provide an effective service for the students of CPR E 288.

8 Closing Material

8.1 Discussion

Our plan for the project fulfills every requirement we've set forward. All of the security requirements are fulfilled by our implementation of Design 1. Every functional requirement has been fulfilled, bar the ongoing effort to produce course content and the stretch goals of the emulators. For next semester, we have a clear plan set in motion that will create a meaningful project at the end.

8.2 Conclusion

We started this project setting out to create a learning environment for the course CPR E 288. To do this, we were given a framework by our advisor and tasked 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, as well as set up secure production environments for the content to be hosted on.

The biggest roadblock to achieving the goal of creating content for every assignment was a lack of understanding and familiarity with the environment. It took us around 3 weeks to really get to know PrairieLearn, and to this day we are still uncovering new techniques with the framework. This lack of understanding slowed progress, and a lot of early on issues could have been solved by better documentation and communication.

8.3 References

[1] IEEE, "Code of Ethics | IEEE Computer Society," Computer.org, 2017. https://www.computer.org/education/code-of-ethics (accessed Dec. 01, 2023).

[1] "PrairieLearn," ReadTheDocs.io. https://prairielearn.readthedocs.io/en/latest/ (accessed Dec. 01, 2023).

8.4 Appendices

PrairieLearn Documentation: https://prairielearn.readthedocs.io/en/latest/

8.4.1 Team Contract

Team Members:

Carter Murawski	Chris Costa	Matt Graham
Tyler Weberski	William Hudson	Andrew Winters

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:
    a. Face to face Thursday 12:30 pm at Library
2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):
    a. Discord and in-person at meetings
3. Decision-making policy (e.g., consensus, majority vote):

    a. Majority vote
4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):
    a. Gitlab will store notes, note taking will rotate between members.

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:
    a. Everyone will be in attendance, on time, and participating. Members will notify the team ahead of time if they are unable to attend.
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:
    a. Each member will contribute to assignments and ensure everything is completed on time and will communicate if something is at risk of being delayed.
3. Expected level of communication with other team members:
    a. Being available every day, and able to respond within a timely manner to whatever necessary task/issue needs to be resolved (within 24hrs).
4. Expected level of commitment to team decisions and tasks:
    a. When the majority votes, that will be the final decision for the group unless there is a vote following the previous one to modify the decision.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):
    a. Each member will share leadership roles
    b. Different members will take the lead on more specialized tasks as needed
2. Strategies for supporting and guiding the work of all team members:
    a. Weekly meetings where we can set tasks for the next week to complete before the next meeting.
    b. Progress checks to ensure a weekly goal has been met
    c. Everyone knows who is supposed to do what task and when.
3. Strategies for recognizing the contributions of all team members:
    a. Checking assigned tasks within Gitlab to see who has completed an issue

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.
    - Chris Costa: Software Engineering
    - Matt Graham: Software Engineering, Cyber Security
    - Carter Murawski: Electrical Engineering, Embedded systems
    - Tyler Weberski: Software Engineering
    - Andrew Winters: Electrical Engineering
    - Mitch Hudson: Cyber Security
2. Strategies for encouraging and support contributions and ideas from all team members:
    a. Everyone participates in active communication and collaboration on tasks.
3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)
    a. Communicate with the team if any issues arise before they become a problem.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:
    a. Reach milestones on time.
    b. Be in a position for a smooth transition from planning to development.
2. Strategies for planning and assigning individual and teamwork:
    a. Within GitLab, staying on top of the storyboard, assigning tasks to individuals and making sure everyone has something to work on at all times.
3. Strategies for keeping on task:
    a. Keeping everyone up to date and informed on tasks through Discord and Gitlab

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

Communicate with a friendly reminder

    a. Meet as a group and discuss the issue and why it happened, then bring it to a TA to seek further guidance and assistance.

2. What will your team do if the infractions continue?
    a. First meet as a team and discuss why these issues may occur and figure it out there
    b. Next steps would be involving TA as well to continue
    c. If previous steps don't resolve the issue, coming to the professor and trying to resolve the issue

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.
b) I understand that I am obligated to abide by these terms and conditions.
c) I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

NAME	Carter Murawski	DATE	8/31/2023
NAME	Matt Graham	DATE	8/31/2023
NAME	Chris Costa	DATE	8/31/2023
NAME	Tyler Weberski	DATE	8/31/2023
NAME	William Hudson	DATE	8/31/2023
NAME	Andrew Winters	DATE	8/31/2023