4.2 Design Exploration

4.2.1 Design Decisions

Our first key design decision that we made for our project was to implement auto graded questions. This enables instant feedback for students while saving professors and TAs time on grading. This decision is crucial as it supports one of our goals of providing students with immediate feedback about their assignments, while also allowing professors and TAs to focus more on teaching rather than grading. Another key design decision was to incorporate randomized questions by using randomized parameters within questions, making each attempt unique for students. This is essential, as it ensures students have unlimited practice opportunities with varying question parameters, allowing them to gain a further understanding of core concepts. Finally, our last key design decision was using Prairielearn as the project's main platform, as it allows for us to use question randomization and autograding. Choosing Prairielearn for our project allows us to utilize its fully modular capabilities to create questions that are randomly generated with randomized parameters that can be automatically graded by Prairielearn, unlike other alternatives like Canvas.

4.2.2 Ideation

Options to autograde questions:

- Option 1: manual grading by professor or TAs
- Option 2: not asking code questions that can't be autograded
- Option 3: write own compiler + unit tests to run in PL
- Option 4: all multiple choice
- Option 5: use PrairieLearn's built-in C and Python autograder
- Option 6: create our own custom autograder container in addition to PrairieLearn's version
- Option 7: make harder autogradable questions, like programming ones, graded based on participation
    - Ex: if you submit code, no matter if it's right or wrong, you will get participation points for that question

The decision we had the most choices with was the goal of having almost all questions able to be autograded. Although an achievable goal, we had to determine if it was worth the risks and costs, and if so, how it should be implemented. The obvious alternative is to allow manual grading, and forego our goal. Or, we could remove all questions that cannot be simply autograded- mainly paragraph response and student-written code questions. We also considered adapting them by taking the core concepts of these questions and turning them into multiple choice, or other format, questions.

Autograding the C and ARM coding questions is the most complex part to implement. Some choices for achieving this included using PrairieLearn's built-in autograders for C and Python, or using these versions in addition to adding a custom autograder container. If given more time and materials, we could even have decided to build our own compiler and unit testing solution from the ground up to run in PrairieLearn. Finally, we could have utilized a "challenge question" approach, where more difficult programming questions are asked, but students receive participation points as long as they tried in good faith.

4.2.3 Decision-Making and Trade-Off

To identify the pros and cons of which options to use based on our list of brainstormed options, we had to think about our ultimate goal with our design and user needs. We want to have questions that are autogradable so professors and TAs don't have to worry much about manually grading assignments, but we also want to develop complex questions that are unique and engaging for students.

	Pros	Cons	Trade-offs
Option 1	• Professors and TAs have more say in how a question gets graded • Don't have to worry about making tests and making questions autogradable	• Doesn't need our professor's and TA's need of questions being autogradable • Students won't get instant feedback when answering a question • Answers can be mistaken and points can be wrongfully deducted	By picking this option over the rest, we are giving up: • saving time for professors and TAs • Instant feedback for students • Incorrect grading by human error for: • More say in how a problem gets graded • The convenience of not having the extra step to autograde questions
Option 2	• Saves time in developing new strategies for autograding difficult coding questions • Don't have to worry about creating custom autograder containers for questions that PrairieLearn can't handle	• Can only ask coding questions in a certain format, which can hinder uniqueness	By picking this option over the rest, we are giving up: • Uniqueness of problems for: • Not having many autograder problems
Option 3	• We know the whole process from submitting an answer to how it gets autograded • We can customize how questions get autograded	• Will take a lot of time to get working properly	By picking this option over the rest, we are giving up: • Time that we could use to develop and improve problems for: • Making our own method to autograde questions
Option 4	• Easy to autograde • Will have almost no autograder issues or debugging	• Questions will only be in a multiple-choice format • Hinders the engagement of	By picking this option over the rest, we are giving up: • Problems that our engaging for students and encourage learning

		questions	for: • No difficulties with making problems autogradable
Option 5	• This is great at autograding C code submitted by students • Also works for calculation based questions	• Does not support ARM assembly language input • Does not simulate the actual hardware that students use in lab	By picking this option over the rest, we are giving up: • ARM assembly question autograding • Simulations of real 2880 hardware for: • Ease of implementation
Option 6	• More types of questions can be autograded, such as ARM assembly questions • We'll know how to use the custom autograder	• It will take some time to create a functioning autograder container	By picking this option over the rest, we are giving up: • Time that could be used for problem development or improvement for: • More variety in questions that can also be autograded
Option 7	• Easier question design since you do not need to write code to grade the question • All students will get points if they attempt the question, regardless of correctness	• This will not provide feedback to students to let them know if their solution is correct • Students may submit low-effort and non-functioning solutions.	By picking this option over the rest, we are giving up: • Instant feedback to students. for: • Easier question design and participation grading.

4.3 Proposed Design

4.3.1 Overview

Our current design is a server with PrairieLearn running as a Docker container. Users access the server through their web browser with the server URL. Inside PrairieLearn, we have a list of homework assignments and their respective questions. A question is composed of a JSON file with basic information about the question, an HTML file that structures the visual component of the question, and a Python script that supports the internals. Questions can get autograded through either the Python script or an autograder image that checks programs written in C or ARM assembly. TAs can view student responses and provide feedback if answers are not autograded.

4.3.2 Detailed Design and Visual(s)

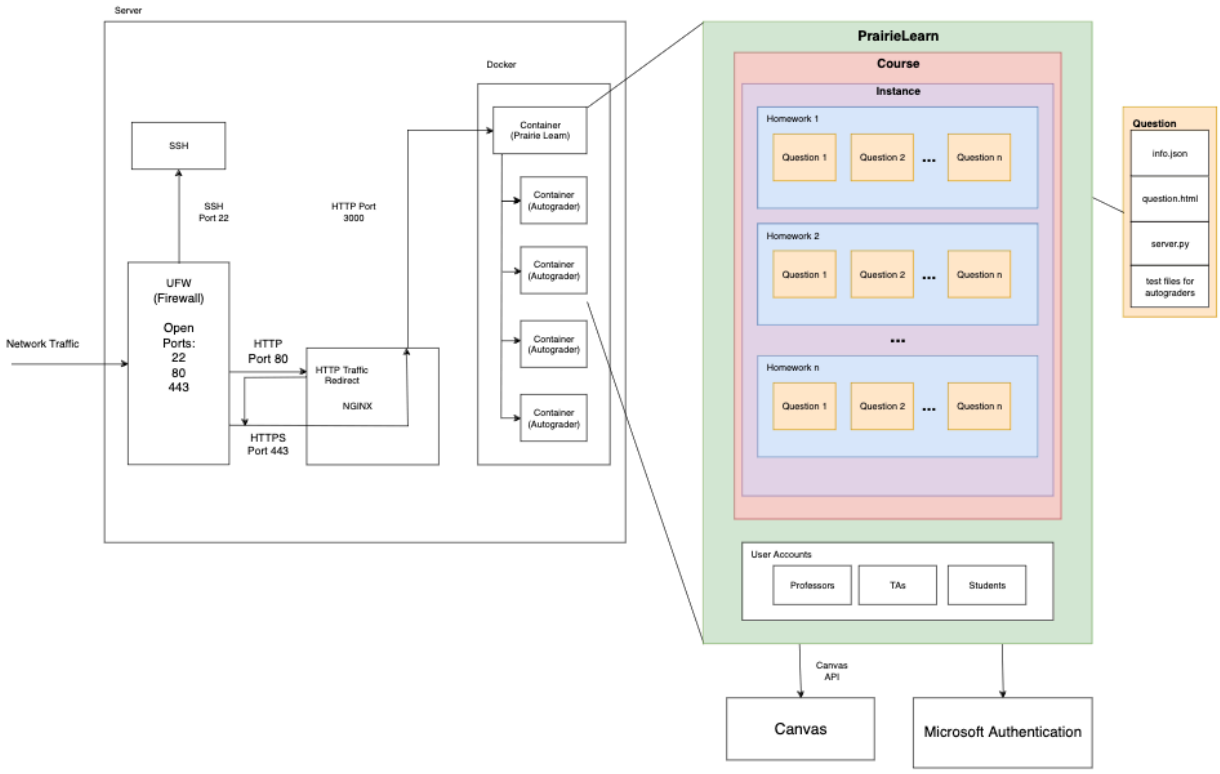

Figure: Block diagram of PrairieLearn components

The server hosting our PrairieLearn application is an Ubuntu 22.04 Linux machine. It has port 22 open for SSH which allows us to access the server for maintenance. Ports 80 and 443 are also open for HTTP/HTTPS traffic to allow users to access the site through the URL. User traffic is redirected with NGINX to port 3000, the PrairieLearn application port. PrairieLearn is hosted in a Docker container, which functions like a virtual machine. The ARM assembly and C autograders are separate containers connected to the PrairieLearn container through a socket and input/output directories.

PraireLearn is structured with courses with instances. An example of how those are used is “CPRE 2880” as the course listing and “Fall 2024” as the instance. Inside each instance are homework sets with individual questions. Each question has a “info.json” file that includes a unique ID, a title, a category, additional tags, and autograding options. Questions also have an HTML that allows the designer to create the visual components of the question, like diagrams, prompts, and entry boxes. There is also a “server.py” that randomizes question parameters and grades the question if it is not programming based. If it is programming based, there is a separate folder called “tests” that has a correct program and scripts to initialize the autograder.

There are three types of users for PrairieLearn: students, TAs, and professors. Students can view their assignments and submit answers. TAs can view those answers and provide feedback if necessary. Professors can design and modify the questions/course. Users login with their Iowa State credentials through Microsoft Authentication.

Eventually, PrairieLearn will be integrated into Canvas so that grades are automatically synced and the PrairieLearn application window is embedded into the Canvas site.

4.3.3 Functionality

Our design would ideally become a quiet staple in CPRE classes. Professors with little technical skill would be able to set up a virtual machine to host a PrairieLearn instance just by following our documentation. Professors, TAs, or senior design groups could then implement course specific questions. Both professors and students will be able to log in with an account connected to their university account, making access restriction easy. Professors can then release a course for their students to access, customize which questions will be graded, and publish assignments.

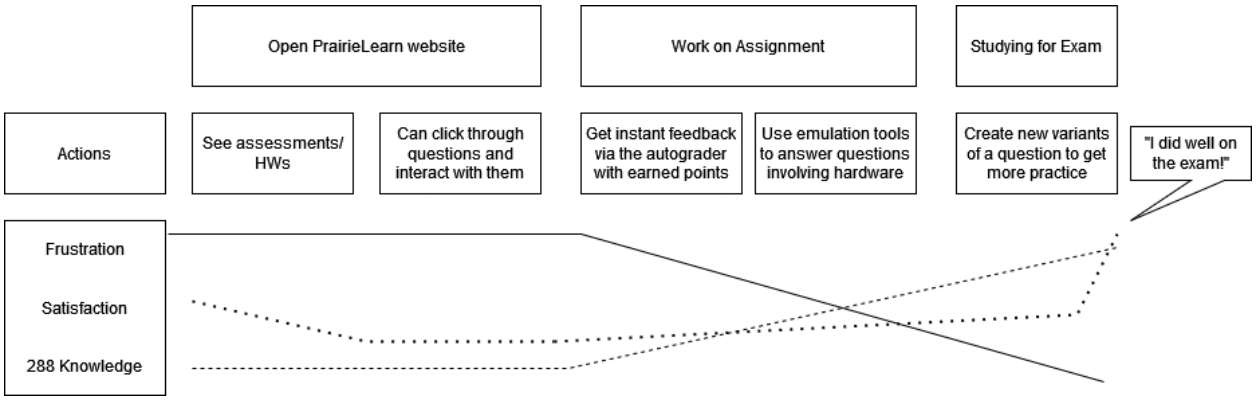

Figure: Journey Map describing student use of PrairieLearn

PrairieLearn should be an effective platform for students to complete homework assignments and practice concepts. Questions will be more than just short answers or multiple choice- we will implement questions that are more interesting and require more effort so students must think through each question, even if they have seen a variant of the problem before. We aim for our solution to become a standard for a flexible engineering homework platform that teaches students well. Our design should have minimal bugs and cause minimal frustration as students work through problems. From there, the course will interface with Canvas to automatically update student grades. Overall, the effect will be reduced stress on professors and students, reduced time spent grading, and improved student test scores.

4.3.4 Areas of Concern and Development

At the moment, our design is functioning and nearing a version ready for Beta testing, however, it does have some noted faults. One is that all homework questions are not implemented or are implemented improperly. There are also more manual graded questions than preferred, since manually graded questions create more work for TAs that we wish to avoid. Additionally, many questions are simple and need to be more engaging for users.

Our concerns in addressing the above faults come from several factors. We have a time restraint because our client requests a Beta version to use in the upcoming semester, and we also only have so much time to contribute to the project as college students. We also have not tested our application with actual students, so it is hard to find what all will need changed or improved. There is also a limit of already implemented question elements in PrairieLearn, so creating unique questions requires significantly more effort from the designer.

The plan for this project is to have a Beta version ready for Spring 2025 that allows students that semester to take a first look at our application and provide us feedback for improvements. We will also create further documentation to support future PrairieLearn developers that work on this course or another. We will also create new PrairieLearn question design elements that allow for robust and engaging questions.

4.4 Technology Considerations

The technologies used in our design include Prairielearn, QEMU ARM emulator, and Git. Our first technology, Prairielearn, integrates well with our project as our goal is to create a dynamic, customized, interactive learning environment. Through use of its completely customizable questions, we can provide auto grading and question randomization, which will help students learn through practice. However, Prairielearn does require a certain degree of technical skill to program a course in Prairielearn. In order to create questions in Prairielearn, there's no simple interface you can use, you have to hard code it from scratch, which could dissuade potential users. Another option we could've used is the technology that was originally used, Canvas. Professors are already familiar with it, and it's easy to create homework assignments. However, the autograding abilities of Canvas are mediocre and the question customization is almost non-existent, meaning Prairielearn is better for our project.

Our next technology was the QEMU ARM emulator, which enables hardware emulation of a TM4 microcontroller. This technology allows for students to interact with embedded systems without needing a physical device such as the lab robots. Some drawbacks of this is the difficult setup of the emulator and the maintenance required to ensure compatibility with Prairielearn. An alternative to using the QEMU ARM emulator would be to use the Cybot emulator which was made for CPRE 2880. Since it was specially made for this course, it has all the features that the QEMU ARM emulator has. However, it has less documentation than QEMU while also having issues syncing to Prairielearn, meaning that the QEMU ARM emulator was the better option for us. However we haven't given up on using the Cybot emulator, and we have hopes to use it in the future for other aspects of our project.

The final technology we used was Git. Prairielearn has the ability to pull courses from Git repos, meaning any code we've worked on for our project can be pulled by the server whenever a new update is pushed. It also helps us with tracking code changes and managing our project as a whole. The only downside of Git is that it could be challenging for professors unfamiliar with Git. An alternative to using Git would be to just host the course files on the Prairielearn server, but this makes it harder to make changes to the course, as you will have to manually upload the files. For these reasons we choose to use Git to host our course files.

4.5 Design Analysis

With the current progress of our design, we almost have a polished beta version of our application ready for CPRE 2880 students to use and experiment with. Almost all homeworks have been implemented into our design, where each problem aligns with existing homeworks and is autogradable. There are still two homework assignments that need work done to their questions, such as making some questions autogradable or adding images to the questions. Our proposed design does work and it has delivered us a platform that will increase engagement and learning for students, as well as make it easier for questions to be graded without any manual intervention. The only problem that we have with our design is that there is a random bug that occurs when autograding assembly-based programming questions. However, it is totally feasible to get our design functioning correctly for the CPRE 2880 courses and other courses as well.

The future plans for our design is to polish up the remaining bit of our design before the end of the Fall 2024 semester. This involves finishing the two homeworks mentioned in the previous paragraph. This will allow our design to be in a beta version that we can have CPRE 2880 students experiment with in the Spring 2025 semester, and give us meaningful feedback to improve our design. We also want to add more question types into our design, mostly ones that utilize the emulation tools. This will provide students with problems that cover different topics in the course.