Final Documentation, Design Document/Charter

Project 3: Building an Autograding Question System using PrairieLearn

University of British Columbia Okanagan

COSC 499 - Summer 2022

Date: August 17th, 2022

Contacts:

Emiel van der Poel (emielvdpoel@gmail.com)

Luis Lucio (llucio99@icloud.com)

Prajeet Didden (prajeetdidden@gmail.com)

Siqiao Yuan (siqiaoyuan@gmail.com)

1. **Design Document**

1. Project Description

1.1 The Problem

The current solution for the ER design question that was previously created by UBCO students known as AutoER works well when it is only being used for this sole purpose. If someone were to want to host other types of questions on that platform it would not be possible. As well, having a UI for a user to navigate that project does not exist.

This is where PrairieLearn comes in. PrairieLearn is an open source problem driven learning platform for creating homework and tests. It is designed that questions can be randomly generated and can be auto graded upon completion. This platform would seem like an ideal platform to host this ER design question.

1.2 Project Description

For this project, we will be using PrairieLearn, an open source software for online learning, to create a scalable system for question generation and answering. Also, the existing AutoER system for answering ER related questions for the Introduction to Databases Course, will be integrated into PrairieLearn.

There are two major steps in this process. Firstly, PrairieLearn must be dockerized and deployed in a production environment on an existing UBCO server for campus wide usage. Secondly, integration of AutoER will be done as a PrairieLearn element to allow easy creation of new ER questions for any instructor with minor HTML knowledge and guiding documentation.

Thorough testing will be done on the generation and grading scripts to ensure accuracy. PrairieLearn supplied tests will be used for all tests related to user authentication, course, assessments, and question generation.

2. Requirements

2.1 Non-functional Requirements

- Deploy dockerized PrairieLearn.
- Grading system that will correctly follow set grading scheme
- Instructor can create a visual representation of an answer and convert that into the text solution.
- Front-end diagram renders maintain high readability standards set by the client.
- Compatible with Mermaid-js with backwards compatibility with nomnoml.
- Use Python as the primary language for the back-end.
- Extensibility and modifiability for future updates.

2.2 Technical Requirements

- Questions will be randomly generated.
- Correct autograded answer corresponds with the question.

2.3 User Requirements

2.3.1 Student User Requirements

- Login to PrairieLearn system.
- Users can view questions.
- Users can answer questions.
- Users can view feedback
- Users will receive a grade.
- Users can self register.
- ~~Users can register via CWL login.~~

2.3.2 Instructor User Requirements

- Login to PrairieLearn system.
- Users can select specific question types.
- Users can retrieve grades from student users.
- Users can see a student's submission.
- Users can set a time limit for taking the test.
- Users can set the number of times the students can retake the test.
- Users can bulk add accounts for students.

2.4 Functional Requirements

- Create documentation for PrairieLearn deployment on docker.
- System auto generates ER questions.
- System auto grades ER questions.
- ~~Expand auto generation scripts to also auto generate SQL questions.~~
- ~~Expand auto grading script to also auto grade SQL questions.~~
- Explore different options for more readable front-end diagram rendering.
- ~~Canvas integration.~~
- Implementation of front-end diagram rendering software.
- Create the ability for instructors to bulk sign up students.

2.5 Additional Requirements

- Convert nomnoml drawing model to Mermaid.js

2.6 Incomplete Requirements

- Users can register via CWL.
- Expand auto generation scripts to also auto generate SQL questions.
- Expand auto grading script to also auto grade SQL questions.
- Canvas integration.

3. User Groups

3.1 User Groups

1. Students.
2. Instructors.
    a. Professors.
    b. Teaching Assistants.
3. Admin.

3.2 User Persona Examples

Student:

Figure 1: *Example User Persona for a student.*

- For Renee, she wants to be able to practice more ER diagrams within less time, as drawing ER diagrams can take lots of time. It would be best for her to be able to receive feedback whenever she finishes a practice because she likes to study alone.

Instructor:

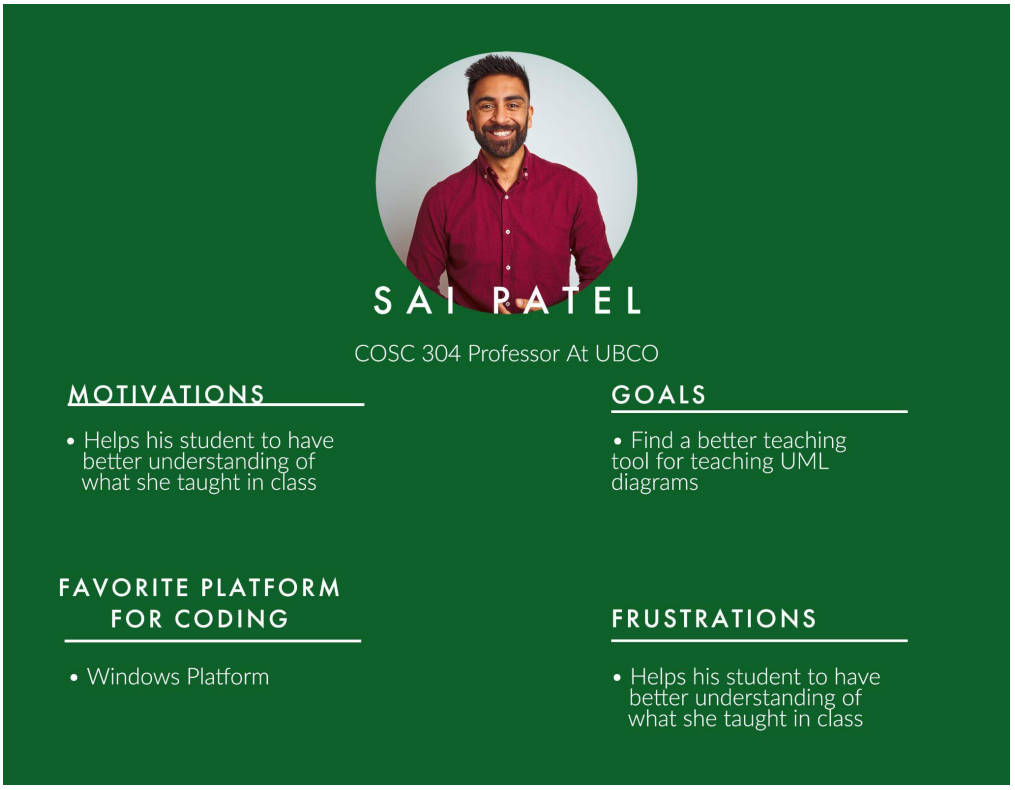

Figure 2: *Example User Persona for an instructor.*

- For Sai, being able to grade ER diagram questions easier is one of the priorities when he chooses his teaching tool for his Database course. At the sametime, he believes practice makes perfect, so being able to provide more examples for practice is also what he is looking for.is also what he is looking for.

3.3 Usage Scenarios

- Introduction to Databases Course
    - ER Diagrams are difficult to complete and very time consuming.
    - Since they are hand drawn the output can be difficult to grade.
    - Also difficult to test these types of questions and practice as it requires time and commitment to craft each question specifically.
- Normal users trying to increase their skill with these diagrams.
    - User wants to be able to practice ER diagrams.
    - There are not a lot of examples to try out these questions.
    - This platform would be good as anyone could access an open course just to try out these questions on their own and with the random generated problems, it increases the number of ER diagram questions a user can practice.

3.4 Student User Group Use Case

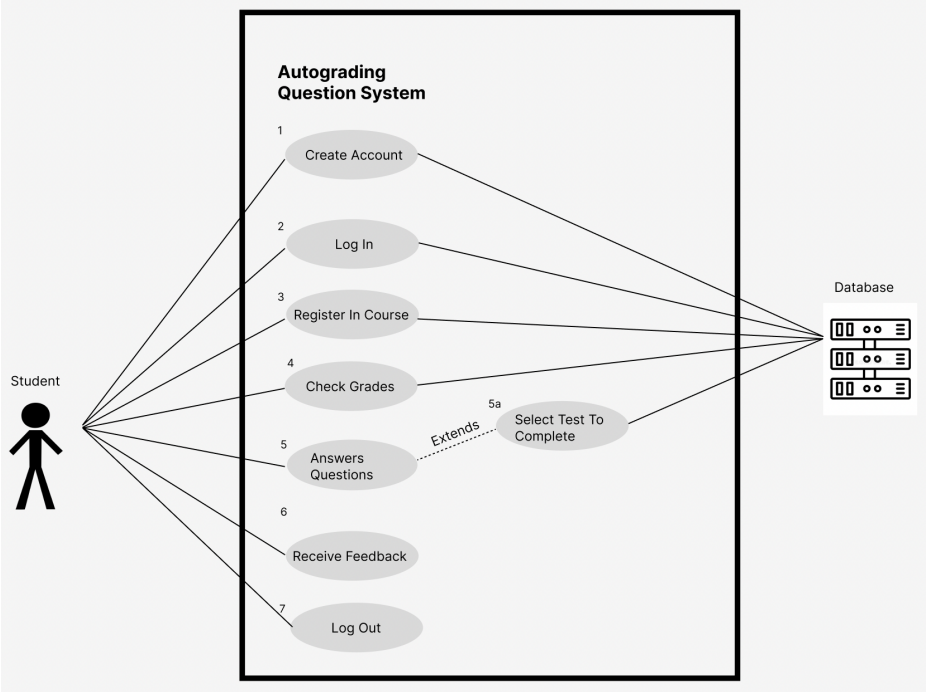

Figure 3: Student Use Case

1- Create Account
Precondition- *To create an account user's are required to have a stable internet condition and a valid email address for signup. Server must be online to sign up.*
Description- *Users that are wanting to use this system are required to create or have an account. The Create account will be an event where users can create an account using a valid email and password set up.*
Postcondition- *Once a user has created an account they will be able to log into the system to start using the services. It is required that they maintain a stable internet connection to use the systems services. As well the new account will be stored in the database.*

2- Log In
Precondition- *Required to get into the log in state users are required to have a stable internet condition, and an account in the system. Server and database must be online. Users account must be in the database for log in with matching email and password.*
Description- *Users that are wanting to use the system must log into the system prior to use. This will be a basic authentication with the ability to use CWL log in credentials if the student is from UBC. Once logged in, students will be capable of checking their grades, answering questions or logging out.*
Postcondition- *Once users have entered this state they will be logged in and will be required to maintain a stable internet connection to stay logged in. Once logged in they will be able to access more features.*

3- Register in Course
Precondition- *Users are required to have a stable internet connection. The server and database must be online. There must be a course available to register. User must have a valid account in the system and must be logged in.*
Description- *Students will be able to register for courses if they were not registered into a course by an instructor.*
Postcondition- *Once a user has registered for a course the database will be updated to give access to the course. Server and database must be online for this.*

4- Check Grades
Precondition- *Users are required to have a stable internet connection. Users are also required to be logged in. Server and database must be online to check grades.*
Description- *This feature is used so the student will be able to check their grades on the questions they have completed.*
Postcondition- *They will be back in a logged in state. Server must be online and the user must have a stable and reliable connection*

5- Answer Questions
Precondition- *Users is required to maintain a stable internet connection. Users are required to be in a logged in state. The instructor of the course must have set the question type. Server and database must be online.*
Description- *This feature is for the students to answer the questions that the instructor has set as a test. Questions will be shown to the students on the PrairieLearn platform where the students can answer the questions. The student will be able to select a test from all available tests in the course.* ***(5a)***.
Postcondition- *Once the student has anwered the question, the student solution will be sent to the autograding system where it will be checked for accuracy. The Server must remain online for this to occur and the user must have a stable internet connection. The database must be online to publish results from question.*

6- Receive Feedback
Precondition- *Server must be online to complete this feature. The system must have received a students answer to a question and the question must have been graded by the autograding system. The User must have a stable internet connection to receive feedback.*
Description- *This feature is designed to give a student feedback on the question they completed. If they got parts wrong the system will display where the user lost marks.*
Postcondition- *Once a user has received feedback the user must maintain a stable internet connection. The user will be able to preview there solution and read the feedback generated by the system.*

7- Log Out
Precondition- *User must maintain a stable internect connection to logout. As well the server must be online.*
Description- *The logout is for the users to be able to leave the system and lock their account until they log in again.*
Postcondition- *User is not connected to the system. Account is locked and can only be accessed once logged in again.*

3.5 Instructor User Group Use Case

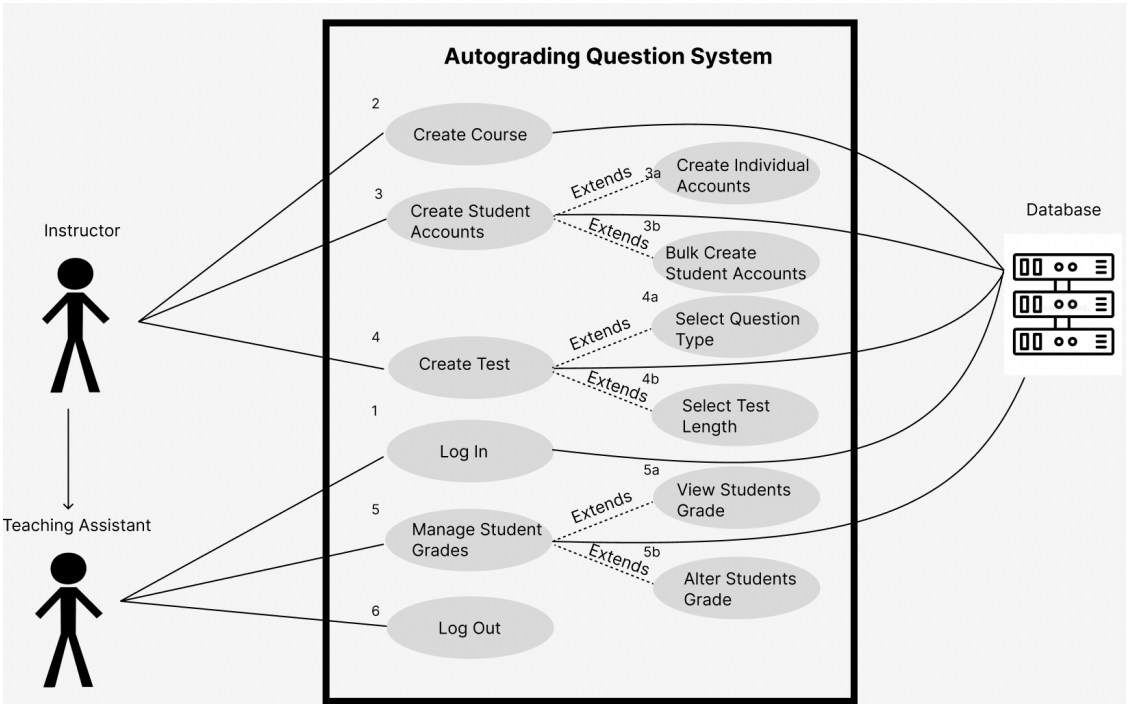

Figure 4: Instructor Use Case

1- Log In
Precondition- *Required to get into the log in state users are required to have a stable internet condition, and an account in the system. Server must be online to log in. The database must also have the associated account registered in the system.*
Description- *To be able to use the system both teaching assistants and instructors will be required to log in.*
Postcondition- *Once users have entered this state they will be logged in and will be required to maintain a stable internet connection to stay logged in. Once logged in they will be able to access more features.*

2- Create Course
Precondition- *To create a course the user is required to have a stable internet connection, the server must be online, and they must be logged into their account with instructor privileges.*
Description- *To have students associated in a course an instructor must create a course. This will be a way for instructors to group students and create tests for those students respectively.*
Postcondition- *Once an Instructor has created a course, that course will be stored in the database, which requires the server to be online and the database to be online and functional.*

3- Create Student Accounts
Precondition- *To create student accounts an instructor is required to have a stable internet connection, the server must be online, and they must be logged into their account with instructor privileges*
Description- *An instructor will have the ability to create accounts for their students in there courses. Once an account is created it will be assigned to the course the student is registered in. Instructors will have the ability to create account for individual student or they can bulk add account by importing a excel file or csv.* ***(3a/3b)***
Postcondition- *Once an Instructor has created student account the student accounts will be stored in the database and the student accounts will be registered to a course. This requires the server and database to be online and functional.*

4- Create Tests
Precondition- *Users are required to have a stable internet connection to create a test. As well their account must be linked as an instructor to access these features. The User is also required to be in the logged in state. Server must be online for the user to select a question type. The instructor must be creating a test for a specific course that has been created.*
Description- *This feature is for the instructor of a course to create a test of random questions with a selected question type, such as UML or SQL and they will be able ot set the test length.* ***(4a/4b)***.
Postcondition- *Once a user has created a test, it will be stored in the database and become available for all students in that course. Tests can be changed at any point in the future.*

5- Manage Student Grades
Precondition- *Users are required to have a stable internet connection. Users are also required to be logged in as an instructor or TA. Server and database must be online to check student grades.*
Description- *This feature is used for teaching assistants and instructors to see how students are doing in their courses. Both are able to view or alter the students grades in the event of an unfair question.* ***(5a/5b)***.
Postcondition- *Any changes will be updated in the database, which requires the server and database to be online.*

6- Log Out
Precondition- *User must maintain a stable internect connection to logout.*
Description- *The logout is for the users to be able to leave the system and lock their account until they log in again.*
Postcondition- *User is not connected to the system. Account is locked and can only be accessed once logged in again.*

3.6 Admin User Group Case

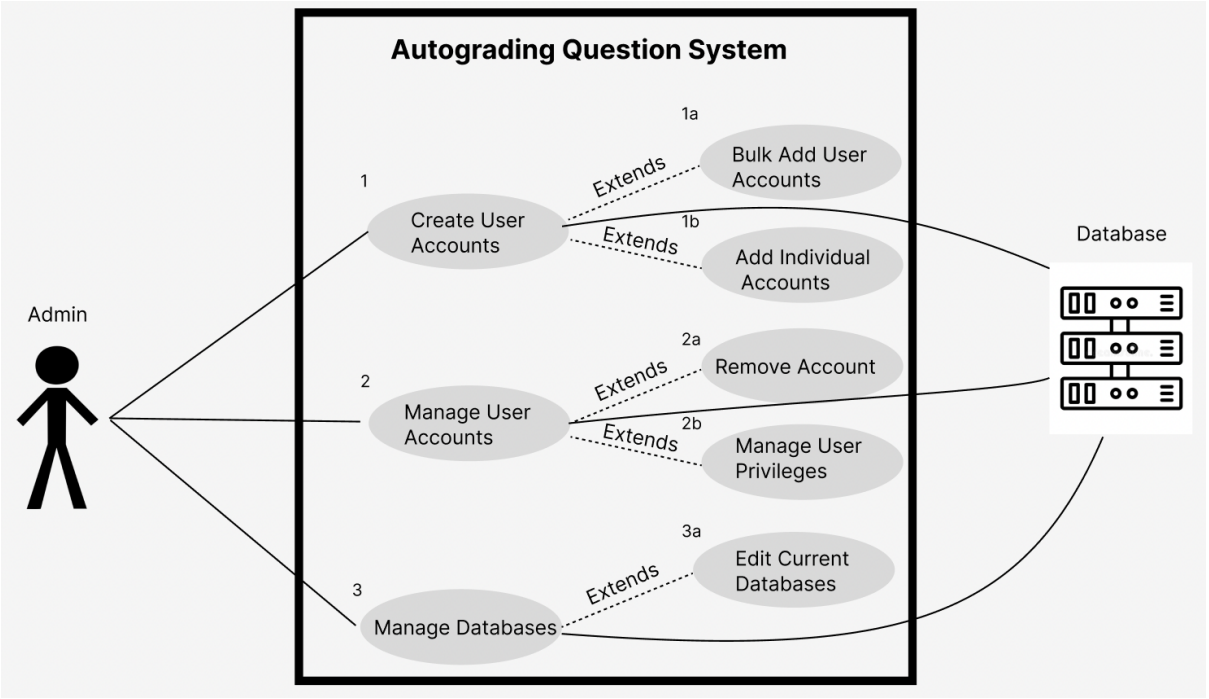

Figure 5: Admin Use Case

1- Create User Accounts
Precondition- *To create an account user's are required to have a stable internet connection. The user must be an admin account. Server and database must be online to sign up.*
Description- *The Admin will be able to create accounts for instructors or students by doing so individually or in a bulk fashion such as import an excel sheet or csv.* ***(1a/1b)***.
Postcondition- *Once user accounts have been created they will be stored in the database, which requires the server to be online and the database to be online.*

2- Manage User Accounts
Precondition- *To manage user accounts the admin must have a stable internet connection. The user must be an admin account. The server and database must be online to use this feature.*
Description- *Admin's will be able to handle different aspects of all users of the system. In the event that someone's privileges needs to be changed from student to teaching assistant this can be done. And an account were to become deactive than the admin would be able to remove these accounts from the database.* ***(2a/2b)***.
Postcondition- *After an account has been altered or removed this will update the database, which requires the server and the database to be online.*

3- Manage Databases
Precondition- *Users are required to have a stable internet connection. The user must be an admin account. Server and the database must be online.*
Description- *In the event that something needs to happen to the databases, the admin will have the capability to make the changes necessary.*
Postcondition- *All changes to the database can only occur if the database is online.*

4. System Architecture

4.1 System Architecture Diagram

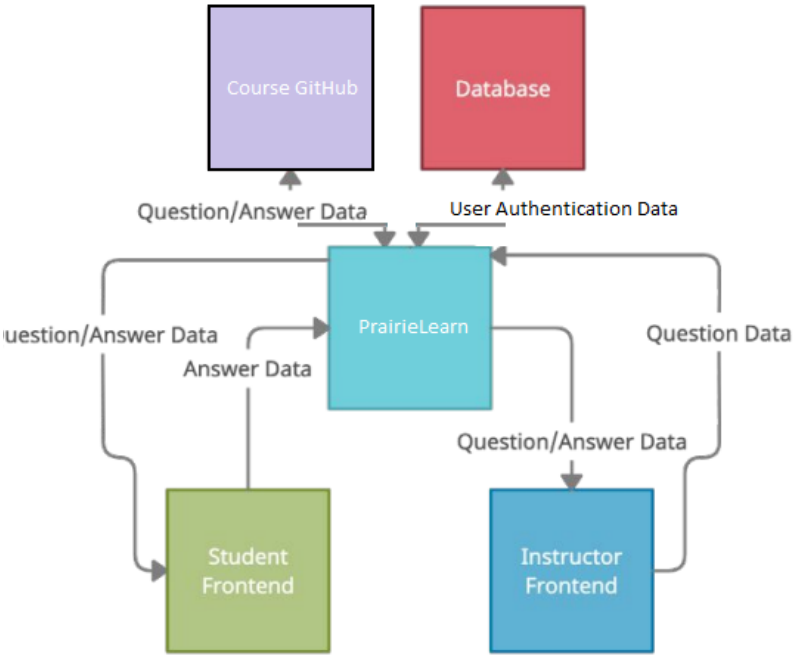

Figure 6: *System architecture for PrairieLearn system*

The instructor frontend will allow users with instructor privileges to enter question data and send that information to PrairieLearn, which will then sync up to the existing GitHub repository for the course. The student frontend will give users questions generated by the instructor, which they will be able to answer through the use of the frontend interface in order to also send that data to PrairieLearn, which will then store it in the database for grades. PrairieLearn will receive both question templates and individual student answers, and will store all of this data on a database that is capable of linking each answer to its corresponding student. PrairieLearn will be responsible for both communicating with the database as well as sending/receiving all data to corresponding frontend interfaces.

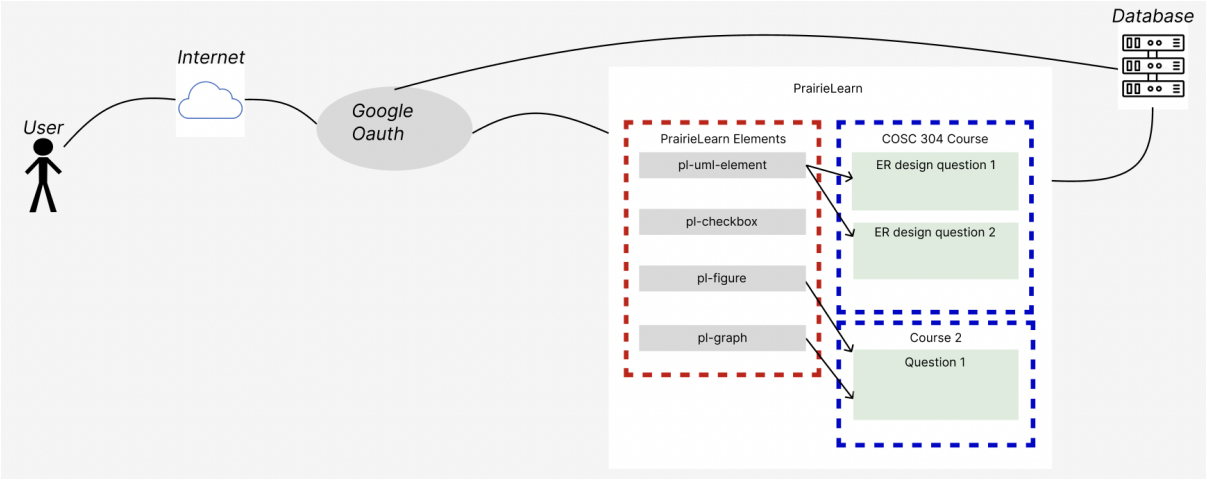

Figure 7: *Here we have a high level architecture diagram with detail on courses and elements.*

This high level architecture diagram describes the way users will connect through the internet to Google Oauth for authentication. Through Google Oauth they will be able to log into the system. The system consists of elements. These elements can be used in courses as questions and assessments. The system is also connected to a postgres database for system storage.

4.2 Level 0 Data Flow Diagram

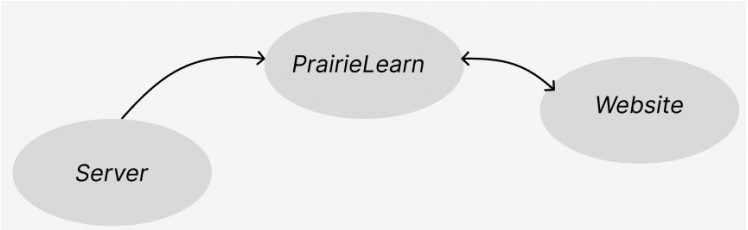

Figure 8: *Level 0 Data Flow Diagram.*

4.3 Level 1 Data Flow Diagram

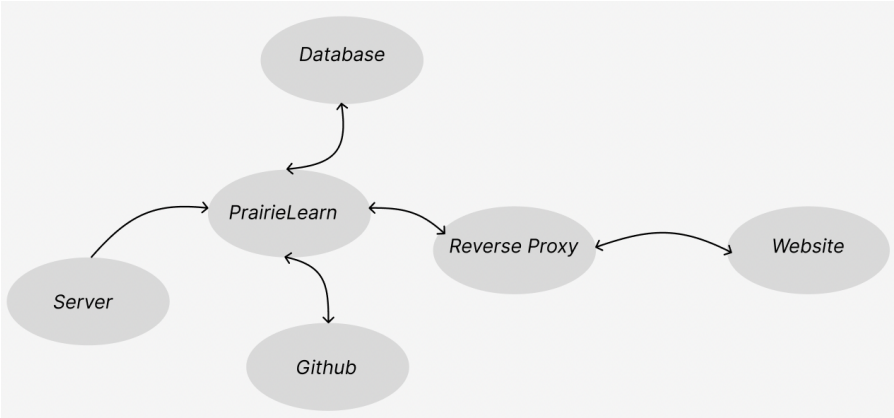

Figure 9: *Level 1 Data Flow Diagram.*

4.4 Sequence Diagram

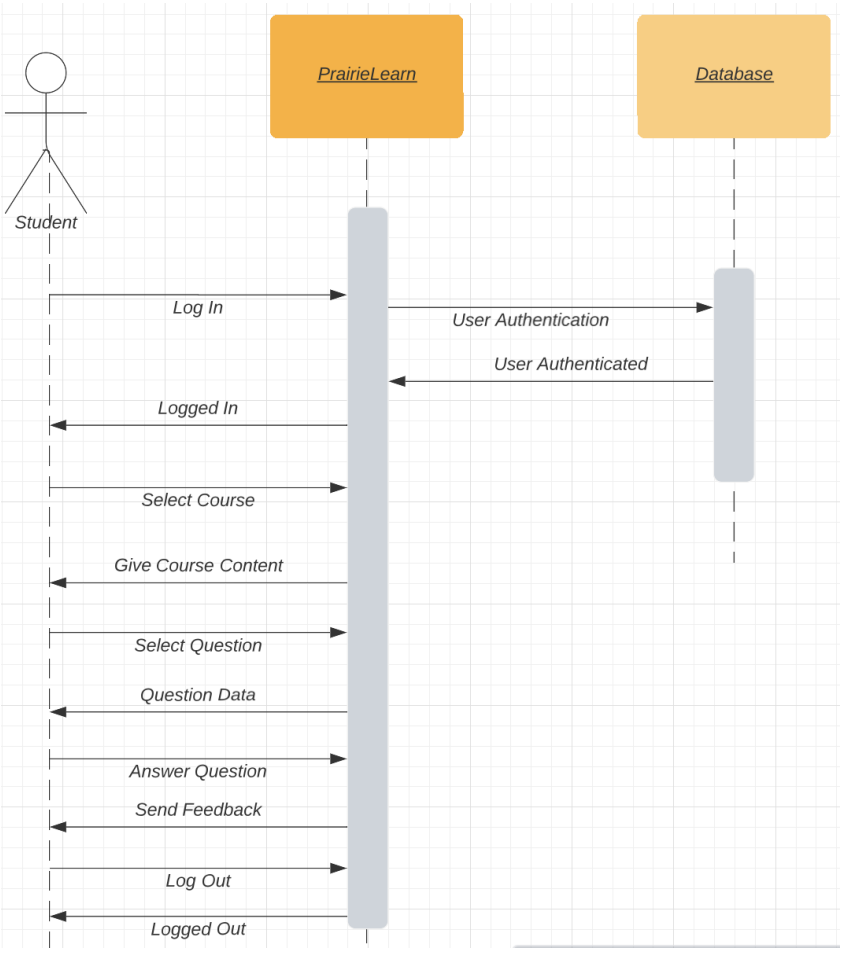

Figure 10: *Sequence Diagram showing how a student user interacts with the system.*

5. UI

5.1 Log In UI

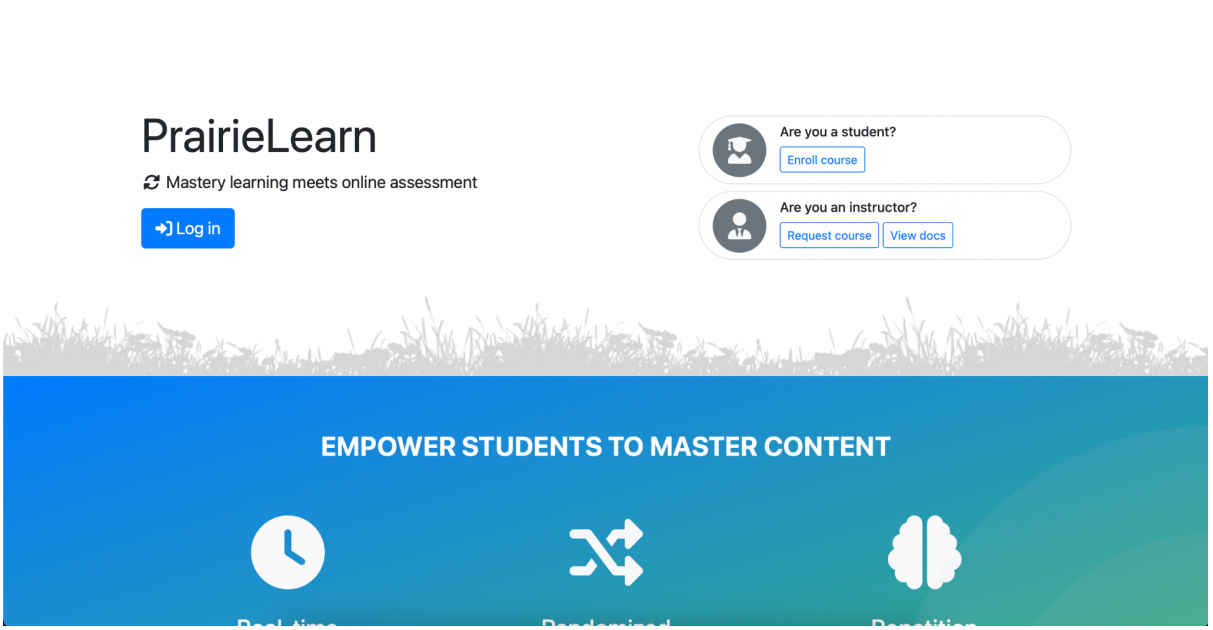

Figure 11: *Landing Page for platform.*

The landing page is a quick introduction to the system and how it functions. It allows a user to login with the Login button. The current login implementation uses Google OAuth2, meaning if you have a google account, you can access the courses. Enrolling in a course will ask you to login if not done, then will allow a user to enroll in a course of their choice.

An instructor is able to request a course, which then goes to the administrators of the system, who are then able to approve of their request which will automatically create a github repository in the organization and add their GitHub credentials to that repository.

View docs allow a user to view documentation pertaining to PrairieLearn.

5.2 Student Assessments View

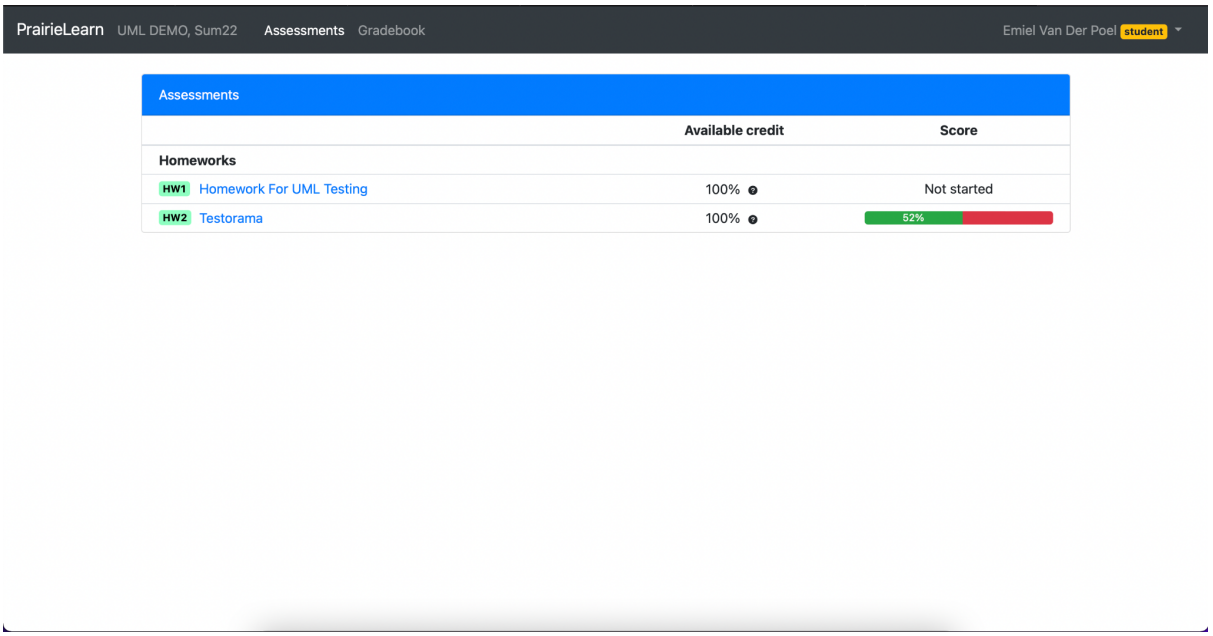

Figure 12: *Assessment page where students can find homework and exams.*

5.3 Student Questions View

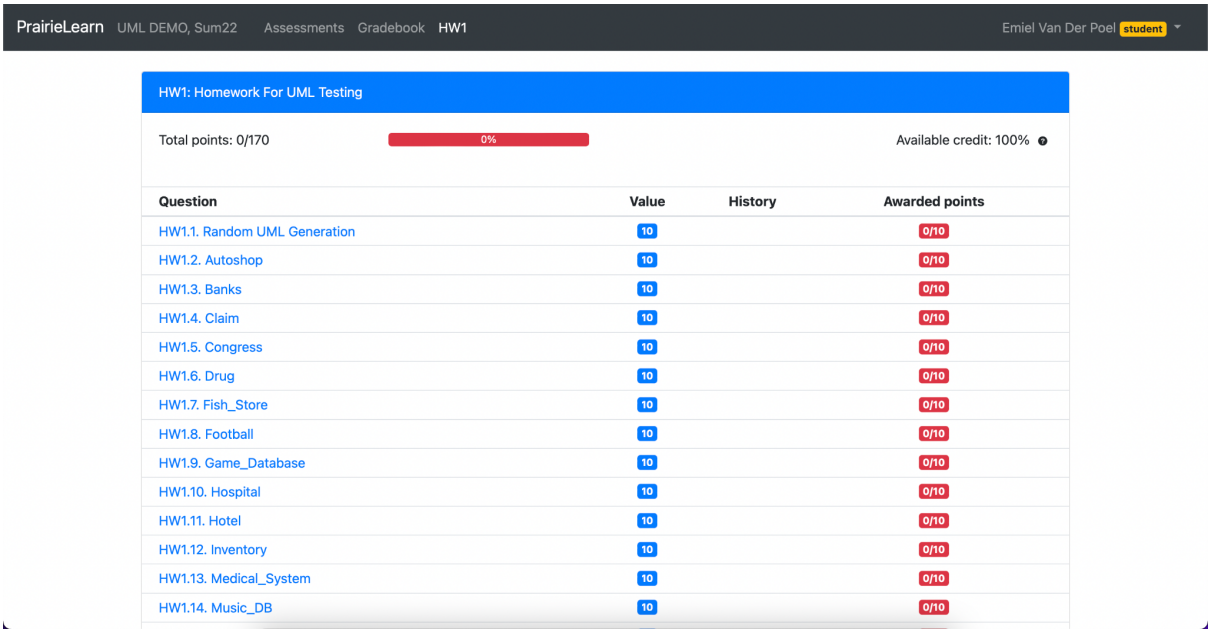

Figure 13: *Questions page where students can view the questions inside a homework assignment or exam.*

5.4 Instructor Course View

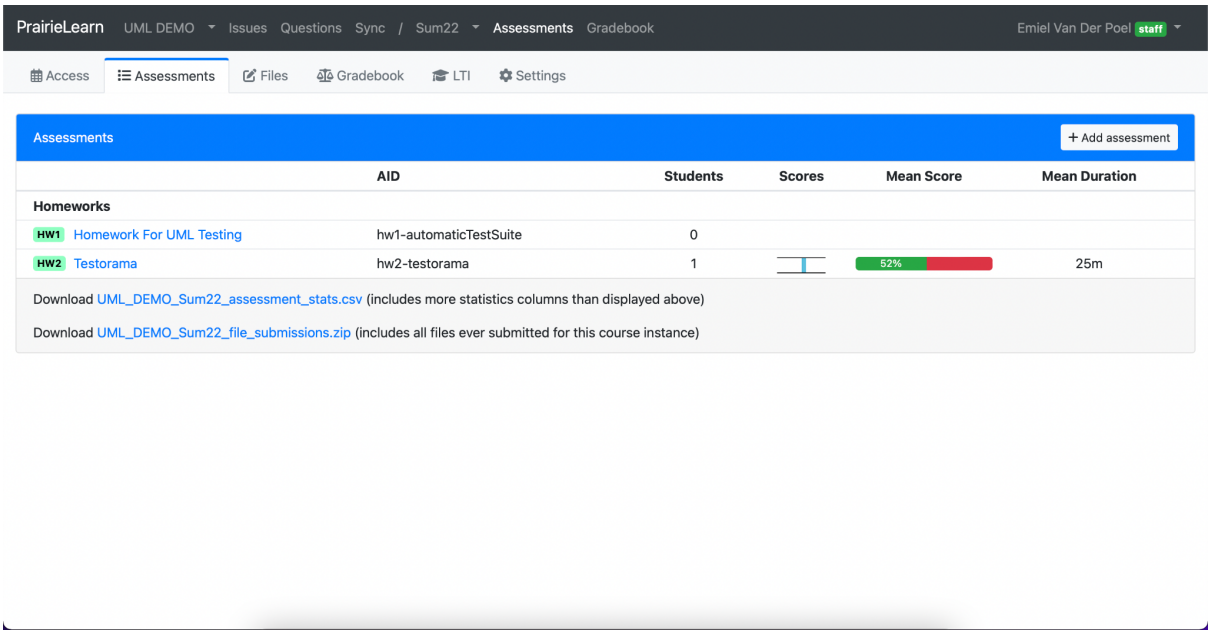

Figure 14: *Course page where the instructor can see assessments and relevant information in the course.*

5.5 Main Question UI

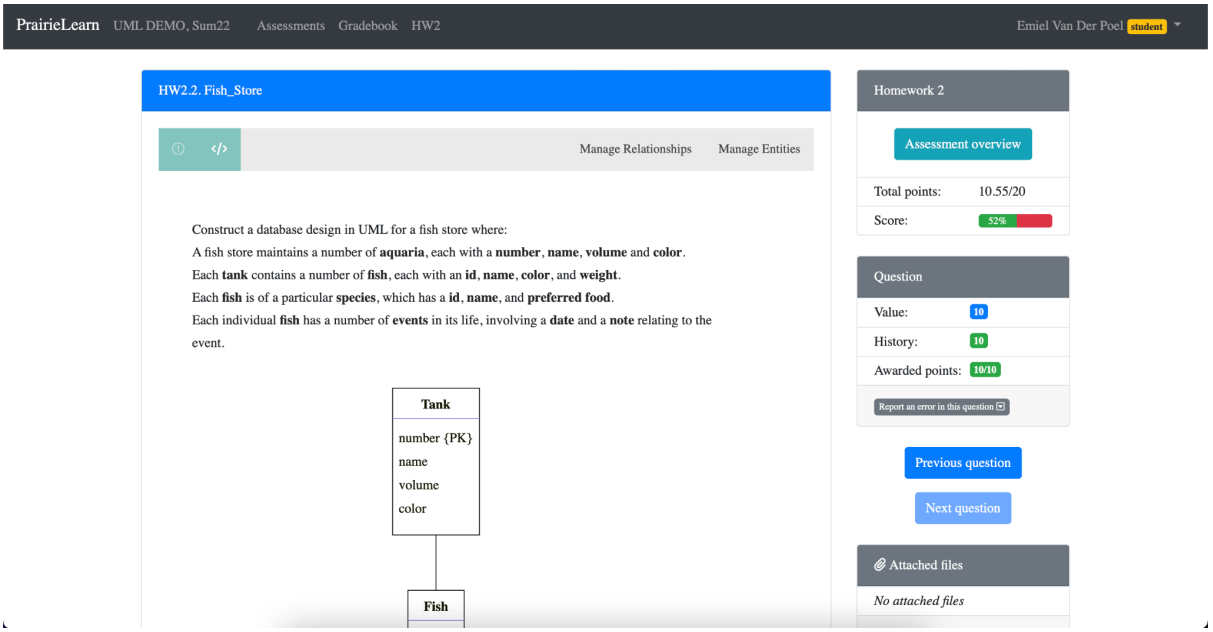

Figure 15: *ER question integrated into PrairieLearn.*

AutoER's question system has been integrated into PrairieLearn as an element, which allows instructors to create ER questions using the element. Students are then able to answer said questions which will give them a grade.

5.6 Question Feedback UI

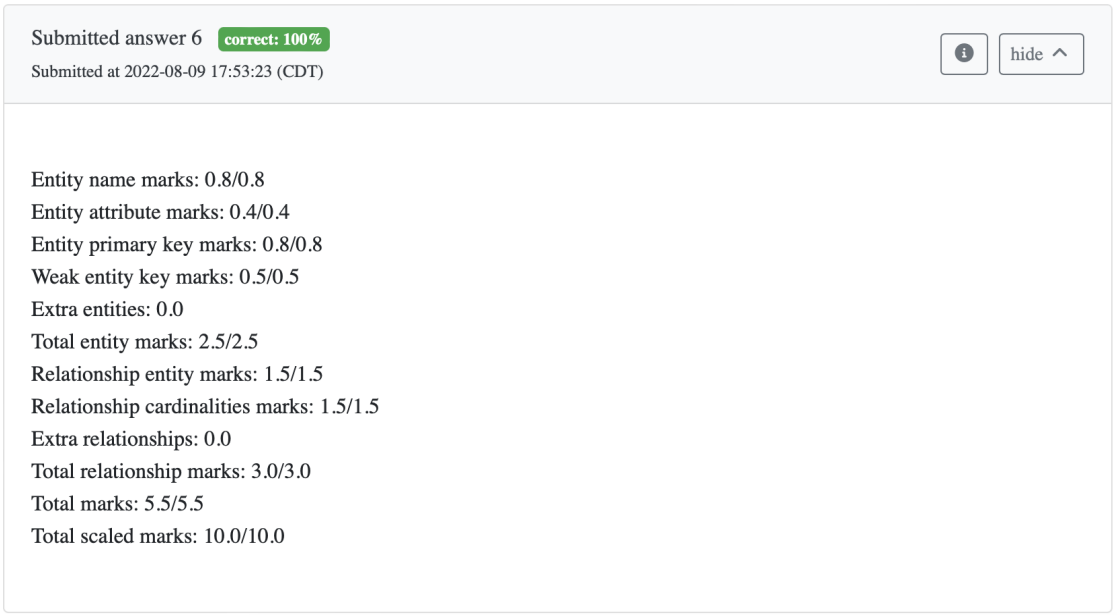

Figure 8: *Marks and feedback after submitting and grading a question.*

Instant marks and feedback is available for students without the need for an instructor or TA marker. This will allow students to practice questions efficiently and will decrease marking time on Instructors and teaching assistants.

6. Technical Specifications

1. Primary Language - Python.
2. Secondary Language - JavaScript.
3. Frontend IDE - WebStorm.
4. Backend IDE - PyCharm.
5. Unit Testing - PyTest.
6. Time Tracking - Toggl.
7. Task Tracking - Github Projects.
8. Continuous Integration - Travis CI.

7. Test Plan

7.1 Unit Testing

- Unit testing will be used to verify that functions results are as expected. Unit testing will be used throughout the whole project to help build up functions.

- Coverage testing should apply to all code in the codebase we have written. This will allow us to see which lines of code are being used which in turn can help optimize code.
- An example would be making sure that User Credentials are unique and valid when a user signs up.

7.2 Regression Testing

- Swapping the backend from Autoed to PrairieLearn will require regression testing, which can be easily implemented by using the tests written for Autoed with minor modifications. An example test could be checking whether the autograding script still works on the sample questions from the preceding project with the same output. Using the tests from the previous project and checking their output, then making sure the output of the modified system is the same will be the way to check if the system responds in the same way.

7.3 Integration Testing

- System integration testing will be written for connecting Autoer and PrairieLearn components. A test could be written to check whether data accessibility is happening correctly. An example would be testing whether the user credentials are authenticated correctly from the database.

7.4 Component Testing

- Functionality testing between components, to ensure that they work together. This will test over classes and different files. These tests can be written on a per component basis, and should fail before writing each component. An example would be testing the clicking of the login button and making sure the user has been logged in. Component testing should be done on individual components and tests should be built requiring the least amount of dependencies to keep components as modular as possible.

7.5 User Testing

- User creation will be tested for correctness of credentials, empty fields, incorrect credentials, and if the user is logged in successfully or not. This can be tested by creating tests with incorrect credentials, and tests with correct credentials.
- Logout will be tested for if the user is logged out once the logout button is clicked.
- User creation will be tested for field validation, so each field is filled out with the correct type. Testing can be done to check if the email field is only taking emails, Name field does not accept numbers and symbols, date fields only take numbers and months, and password field matches some form of security requirements.
- Uploading user credentials can be tested to see if the excel file is being accessed, user credentials are being pulled correctly, and if the table is populating correctly.
- The feedback for user testing will be through the form of surveys, SUS will be used as a survey template.

7.6 Functionality Testing

- The functionality of the system is based on whether or not the system can accurately grade ER diagram questions in the way it did before the shift to PrairieLearn. An example would be testing to check if the question is being graded, by checking if when entities are clicked, they are updated on the grading scheme. The grading scheme should align with the original system (AutoEd).

8. References

1. *Foss, S., Urazova, T., & Lawrence, R. (n.d.). Automatic Generation and Marking of UML Database Design Diagrams.*
2. *Urazova, T. (2022). Building a System for Automated Question Generation and Evaluation to Assist Students Learning UML Database Design. [Honours Thesis, University of British Columbia].*

Charter, Scope and Requirements Document

Project 3: Building an Autograding Question System using PrairieLearn

1. Project Objective

The objective of the project is to amend PrairieLearn, which is an open source software, to support a variety of questions regarding UML, SQL, and programming. The implementation will continue to use the previous front-end (AutoEr) which will be modified to use Mermaid JS for cleaner UML diagrams. Existing python question auto generation script will be imported into PrairieLearn which will replace the current existing backend (AutoEd). Investigations will be done for a possible integration of PrairieLearn into Canvas.

The current state of the system works with AutoEr functioning as the frontend, which lets students answer questions and see their grades. The backend works with AutoEd, which generates randomized questions from a python file. Documentation exists for the current system, and will need to be updated to incorporate the changes with the new system. The new backend system (PrairieLearn) will be deployed in a dockerized format, which will allow the professor full control over the system.

Thorough testing will be done on the generation and grading scripts to ensure accuracy. PrairieLearn supplied tests will be used for all tests related to user authentication, course, assessments, and question generation.

2. Stakeholder List

1. The University of British Columbia (Client Company).
2. Ramon Lawrence (Client).
3. Project Team.
4. Instructors (User).
5. Students (User).

3. Major Milestones

1. Finish documentation (August 13).
2. Design document (June 5).
3. Deliver design presentation (June 10).
4. Create early tests for test-driven design (June 18).
5. Update Nomnoml to Mermaid JS for rendering UML diagrams (June 25).
6. Import AutoEd back-end to PrairieLearn (July 1).
7. Create a working prototype / minimum viable product (July 8).
8. Import previous UML questions and design new SQL/programming questions to PrairieLearn (July 14).
9. Stretch goals, bug fixes, tweaks, security and visual updates. (August 13).

4. Requirements

4.1 Functional Requirements

1. Documentation will be created for deployment of PrairieLearn on docker.
2. The system will auto generate UML questions.
3. The system will auto grade UML questions.
4. The system will auto generate SQL questions.
5. The system will auto grade SQL questions.
6. The system will be explored for better front-end diagram rendering.
7. Front-end diagram rendering software will be implemented.
8. The system will have canvas integration.
9. Instructors will have the ability to bulk-sign up students.

4.2 Non-Functional Requirements

1. Deploy dockerized PrairieLearn.
2. Grading system that will mark correctly based on the grading scheme and instructor given answer.
3. Instructor can create a visual representation of an answer and convert that into the text solution.
4. Front-end diagram renders maintain high readability standards set by the client.
5. Compatibility with both Mermaid JS and NomNomL.
6. Use Python as the primary language for the back-end.
7. Extensibility and modifiability for future updates.

4.3 Technical Requirements

1. Questions will be randomly generated.
2. Correct autograded answer corresponds with the question.

4.4 User Requirements

4.4.1 Students User Requirements

1. Login to PrairieLearn system.
2. Users can view questions.
3. Users can answer questions.
4. Users can view feedback.
5. Users will receive a grade.
6. Users can self register.
7. Users can register via CWL login.

4.4.1 Instructors User Requirements

1. Login to PrairieLearn system.
2. Users can select specific question types.
3. Users can retrieve grades from student users.
4. Users can see a student's submission.
5. Users can set a time limit for taking the test.
6. Users can set the number of times the students can retake the test.
7. Users can bulk add accounts for students.

5. Assumptions

1. Developers will be able to program in Python 3.
2. Developers will be able to work with Docker.
3. Developers will provide weekly updates and progress.
4. Deployed instance of AutoEr with PrairieLearn on herokuapp.
5. Timely and weekly scheduled meetings with the client.
6. AutoEd autograder will be successfully imported into PrairieLearn.
7. Questions are generated automatically.
8. Questions are graded automatically.

6. High Level Risks

1. Team members getting sick for extended periods of time.
2. Team members dropping out of course.
3. Poor team synergy.
4. Not getting a prompt response from UBC IT.
5. Changes to PrairieLearn break functionality during development.
6. Unable to contact the client due to external circumstances.
7. Poor communication between developers.
8. Bad communication/relationship with client.

7. Methodology / Workflow

We will be following a test driven development cycle while also pursuing a Kanban workflow; the Kanban board will be visible and updated on GitHub Projects. As a team, we will have weekly 30min meetings (near the beginning of the week) to keep tabs on everyone's progress and to check what has been completed up to that point. We will also be holding 1 hour long meetings with our client every Thursday at 1:30pm which will allow the team to communicate progress and receive feedback on any prototyping completed during the week. Each member will use Toggle in order to keep track of how long they take on completing certain tasks, which we will later use to correct our WBS. The IDEs we selected are: Webstorm for front-end development and pyCharm for back-end development.

Tasks will be handled based on the WBS structure below (9). The best idea is to split off into two smaller teams and practice peer-programming. This will ensure that team members who are unfamiliar with certain work flows will be able to comfortably assimilate into the project. Testing will be written before each component and will be written to fail until said component is complete. This will ensure that the code that is written does exactly what it is supposed to do.

Canvas integration will be handled by first receiving a test course from the university, then trying the Canvas API library to possibly get a way to integrate into Canvas. If this step is deemed to take longer than anticipated and/or unfeasible, it will be documented and terms will be renegotiated with the client.

8. UML Diagrams

The process for an Auto-grader using PrairieLearn is as follows:

An Instructor will create a generic question, which will then be stored in the database. PrairieLearn will use this generic question to create variants for students to answer. A student will then use the AutoEr front-end to answer the question and submit their answer to which PrairieLearn will automark. The mark and feedback will then be released to the student which they will be able to view.

8.1 Student User Group Use Case

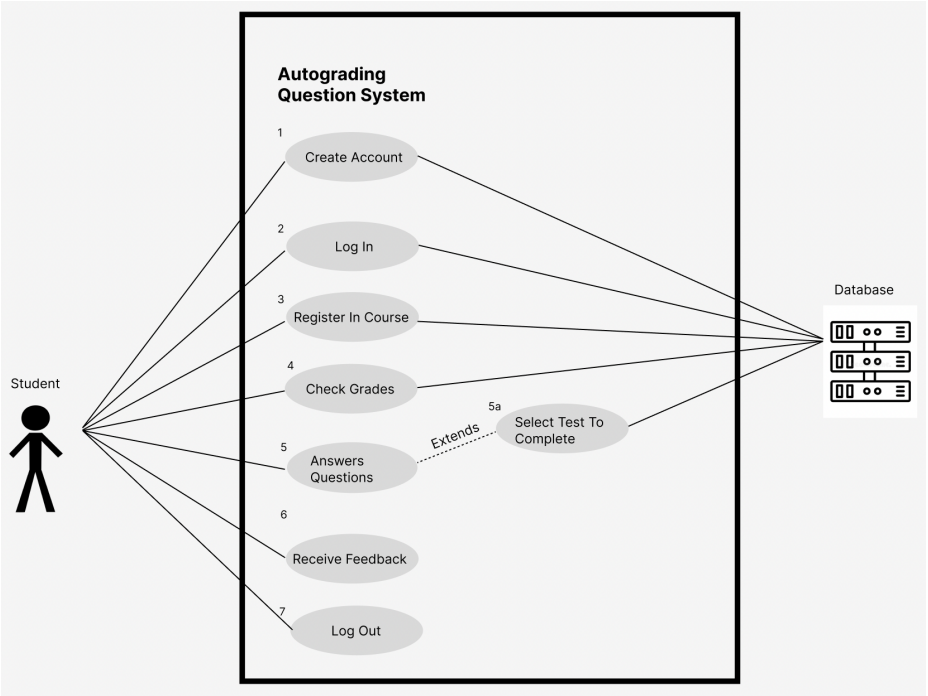

Figure 1

1- Create Account
Precondition- *To create an account user's are required to have a stable internet condition and a valid email address for signup. Server must be online to sign up.*
Description- *Users that are wanting to use this system are required to create or have an account. The Create account will be an event where users can create an account using a valid email and password set up.*
Postcondition- *Once a user has created an account they will be able to log into the system to start using the services. It is required that they maintain a stable internet connection to use the systems services. As well the new account will be stored in the database.*

2- Log In
Precondition- *Required to get into the log in state users are required to have a stable internet condition, and an account in the system. Server and database must be online. Users account must be in the database for log in with matching email and password.*
Description- *Users that are wanting to use the system must log into the system prior to use. This will be a basic authentication with the ability to use CWL log in credentials if the student is from UBC. Once logged in, students will be capable of checking their grades, answering questions or logging out.*
Postcondition- *Once users have entered this state they will be logged in and will be required to maintain a stable internet connection to stay logged in. Once logged in they will be able to access more features.*

3- Register in Course
Precondition- *Users are required to have a stable internet connection. The server and database must be online. There must be a course available to register. User must have a valid account in the system and must be logged in.*
Description- *Students will be able to register for courses if they were not registered into a course by an instructor.*
Postcondition- *Once a user has registered for a course the database will be updated to give access to the course. Server and database must be online for this.*

4- Check Grades
Precondition- *Users are required to have a stable internet connection. Users are also required to be logged in. Server and database must be online to check grades.*
Description- *This feature is used so the student will be able to check their grades on the questions they have completed.*
Postcondition- *They will be back in a logged in state. Server must be online and the user must have a stable and reliable connection*

5- Answer Questions
Precondition- *Users is required to maintain a stable internet connection. Users are required to be in a logged in state. The instructor of the course must have set the question type. Server and database must be online.*
Description- *This feature is for the students to answer the questions that the instructor has set as a test. Questions will be shown to the students on the PrairieLearn platform where the students can answer the questions. The student will be able to select a test from all available tests in the course.* ***(5a)****.*
Postcondition- *Once the student has anwered the question, the student solution will be sent to the autograding system where it will be checked for accuracy. The Server must remain online for this to occur and the user must have a stable internet connection. The database must be online to publish results from question.*

6- Receive Feedback
Precondition- *Server must be online to complete this feature. The system must have received a students answer to a question and the question must have been graded by the autograding system. The User must have a stable internet connection to receive feedback.*
Description- *This feature is designed to give a student feedback on the question they completed. If they got parts wrong the system will display where the user lost marks.*
Postcondition- *Once a user has received feedback the user must maintain a stable internet connection. The user will be able to preview there solution and read the feedback generated by the system.*

7- Log Out
Precondition- *User must maintain a stable internect connection to logout. As well the server must be online.*
Description- *The logout is for the users to be able to leave the system and lock their account until they log in again.*
Postcondition- *User is not connected to the system. Account is locked and can only be accessed once logged in again.*

8.2 Instructor User Group Use Case

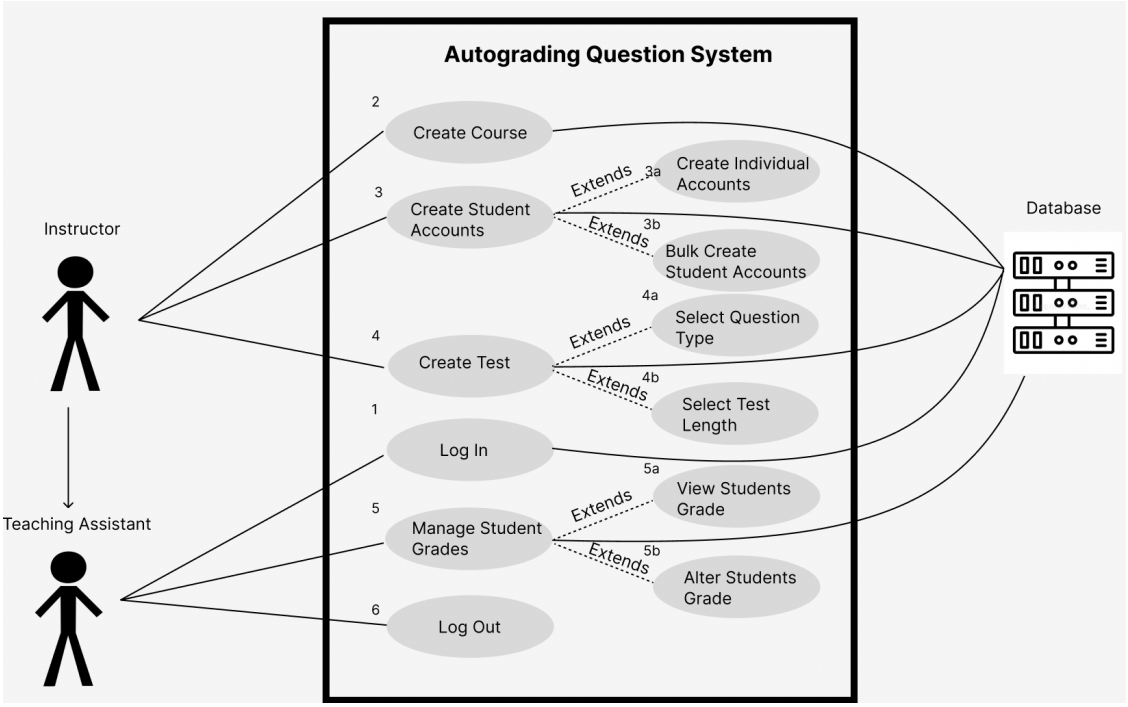

Figure 2

1- Log In
Precondition- *Required to get into the log in state users are required to have a stable internet condition, and an account in the system. Server must be online to log in. The database must also have the associated account registered in the system.*
Description- *To be able to use the system both teaching assistants and instructors will be required to log in.*
Postcondition- *Once users have entered this state they will be logged in and will be required to maintain a stable internet connection to stay logged in. Once logged in they will be able to access more features.*

2- Create Course
Precondition- *To create a course the user is required to have a stable internet connection, the server must be online, and they must be logged into their account with instructor privileges.*
Description- *To have students associated in a course an instructor must create a course. This will be a way for instructors to group students and create tests for those students respectively.*
Postcondition- *Once an Instructor has created a course, that course will be stored in the database, which requires the server to be online and the database to be online and functional.*

3- Create Student Accounts
Precondition- *To create student accounts an instructor is required to have a stable internet connection, the server must be online, and they must be logged into their account with instructor privileges*
Description- *An instructor will have the ability to create accounts for their students in there courses. Once an account is created it will be assigned to the course the student is registered in. Instructors will have the ability to create account for individual student or they can bulk add account by importing a excel file or csv.****(3a/3b)***
Postcondition- *Once an Instructor has created student account the student accounts will be stored in the database and the student accounts will be registered to a course. This requires the server and database to be online and functional.*

4- Create Tests
Precondition- *Users are required to have a stable internet connection to create a test. As well their account must be linked as an instructor to access these features. The User is also required to be in the logged in state. Server must be online for the user to select a question type. The instructor must be creating a test for a specific course that has been created.*
Description- *This feature is for the instructor of a course to create a test of random questions with a selected question type, such as UML or SQL and they will be able ot set the test length.* ***(4a/4b)****.*
Postcondition- *Once a user has created a test, it will be stored in the database and become available for all students in that course. Tests can be changed at any point in the future.*

5- Manage Student Grades
Precondition- *Users are required to have a stable internet connection. Users are also required to be logged in as an instructor or TA. Server and database must be online to check student grades.*
Description- *This feature is used for teaching assistants and instructors to see how students are doing in their courses. Both are able to view or alter the students grades in the event of an unfair question.* ***(5a/5b)****.*
Postcondition- *Any changes will be updated in the database, which requires the server and database to be online.*

6- Log Out
Precondition- *User must maintain a stable internect connection to logout.*
Description- *The logout is for the users to be able to leave the system and lock their account until they log in again.*
Postcondition- *User is not connected to the system. Account is locked and can only be accessed once logged in again.*

8.3 Admin User Group Case

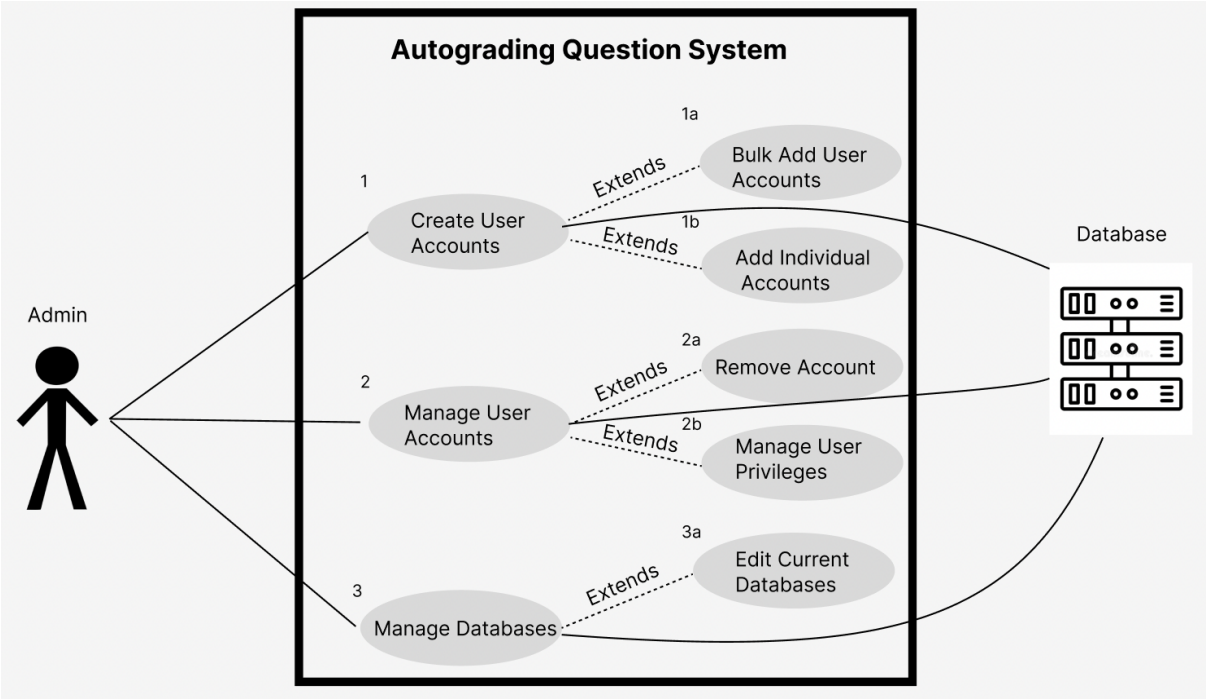

Figure 3

1- Create User Accounts
Precondition- *To create an account user's are required to have a stable internet connection. The user must be an admin account. Server and database must be online to sign up.*
Description- *The Admin will be able to create accounts for instructors or students by doing so individually or in a bulk fashion such as import an excel sheet or csv.* ***(1a/1b)***.
Postcondition- *Once user accounts have been created they will be stored in the database, which requires the server to be online and the database to be online.*

2- Manage User Accounts
Precondition- *To manage user accounts the admin must have a stable internet connection. The user must be an admin account. The server and database must be online to use this feature.*
Description- *Admin's will be able to handle different aspects of all users of the system. In the event that someone's privileges needs to be changed from student to teaching assistant this can be done. And an account were to become deactive than the admin would be able to remove these accounts from the database.* ***(2a/2b)***.
Postcondition- *After an account has been altered or removed this will update the database, which requires the server and the database to be online.*

3- Manage Databases
Precondition- *Users are required to have a stable internet connection. The user must be an admin account. Server and the database must be online.*
Description- *In the event that something needs to happen to the databases, the admin will have the capability to make the changes necessary.*
Postcondition- *All changes to the database can only occur if the database is online.*

9. Work Breakdown Structure

Complete Incomplete

Task List	Average Estimate Hours / Actual Hours				Average Estimate / Actual Estimate
	Prajeet	Emiel	Luis	Tony	
Meetings					
Team meetings	12 / 12	12/12	12 / 12	12/12	12
Client meetings	12 / 14	12/13	12 / 12	12/12	12
Familiarise with Current Setup					
Read AutoEr and AutoEd documents	2 / 2	2/2	2 / 2	2/2	2
Setup workspace for AutoEr and AutoEd	3 / 3	3/3	3 / 2	3/4	3
Analyze and prepare current code base for import into PrairieLearn	10 / 10	10/10	0 / 3	0/2	5
PrairieLearn Set Up					
Setup IDE and workspace	1 / 0.5	1/0.5	3 / 1	1/1.5	1.5
Read and familiarize with PrairieLearn	4 /20	4/15	4 / 7	4/10	4

Deploy PrairieLearn on Docker	2 / 25	3/27	2 / 0	2/6	2.25
Documentation					
Write PrairieLearn Docker Deployment document	0 / 14	3/0	0 / 0	0/0	0.75
Write Scope and Charter document	5 / 2	5/8	5 / 4	5/2	5
Write Instructor Guide document	4 / 0	5/4	4 / 0	4/0	4.25
Update and maintain Scope and Charter document	1 / 6	1/7	1 / 4	1/1	1
Kanban Board Maintenance	2 / 2	2/4	2 / 5	2/0	2
Testing					
Create mock database	0	0	1	0	0.25
Write unit tests for front-end	0	0	20 / 15	20	10
Write unit tests for back-end	20	20	0	0	10
Automate testing	3 / 2	3/4	0 / 4	0/1	12
Write unit tests for auto-grading	8/0	8/0	8 / 18	8/0	8

Write unit tests for auto-generation of question	8/0	8/0	8	8/28	8
Usability tests	2 / 2	2/3	2 / 2	2/3	2
Import UML Question to PrairieLearn					
Convert Frontend drawing from Nomnoml to Mermaid	0	0	8 / 3	8/0	4
Get current AutoEr to work with PrairieLearn backend	8 / 30	8 / 30	8 / 8	8/0	8
Get current autograding python file working with PrairieLearn	10 / 8	10 / 8	0 / 4	0/0	5
Create Relational Algebra Questions for PrairieLearn					
Create data structure to hold all desired Relational Algebra questions	2	2	0	0	1
Explore if Relax can be used for frontend in PrairieLearn	0 / 2	0/1	3 / 22	3/0	1.5
If not Create Frontend button layout to generate Relational Algebra Query	0	0	5	5	2.5
Create backend to pull Relational Algebra questions to properly display	7	7	2	2	4.5

Create backend to translate Relational Algebra to SQL	4	4	0	0	2
Create auto grading to compare questions expected output and queried output	5	5	0	0	2.5
Create SQL Questions for Prairie Learn					
Create data structure to hold all desired SQL questions	2	2	0	0	1
Create text box front end layout in PrairieLearn	0	0	4	4	2
Create backend to pull SQL questions to properly display	6	6	2	2	4
Create backend to query built database	6	6	0	0	3
Create auto grading to compare questions expected output and queried output					
Canvas Integration					
Contact UBC IT for help on Canvas Integration	0.5 / 2	0/0	0 / 0	0/0	0.125
PrairieLearn with AutoEr integration to Canvas	30 / 0	30	30	30	30

PrairieLearn gradebook exporting to Canvas	20 / 0	20	20	20	20
Code Review					
Github Merge requests	4 / 4	4/3	4 / 4	4/4	4
Peer code review	2 / 6	2/2	2 / 10	2/1	2
Github Repo Management	0.5 / 1	0.5/4	0.5 / 5	0.5/1	0.5
TOTAL	206	210.5	177.5	174.5	190.625
Weekly Average (12)	17.16666667	17.54166667	14.79166667	14.54166667	15.88541667

10. Approvals

Project Sponsor Signature _____

Project Manager Signature _____