Project Name: Automating Database Question Generation and Marking with PrairieLearn

Design Documents

Version:
Draft 1 - May 28th, 2023

Contacts:
Matthew Obirek (matthewobirek@gmail.com),
Skyler Alderson (skyler@thealdersons.org),
Andrei Zipis (Andrei_Zipis@hotmail.com),
Nishant Srinivasan (nishant.srinivasan236@gmail.com)

Table of Contents

1. Project Description ✔️
2. User Groups and Personas - Andrei ✅
3. Use Cases
    - 3.1. Use Cases Diagram - Skyler ✅
    - 3.2. Use Case 1: Student Answers Relation Algebra Lab ✅
    - 3.3. Use Case 2: Student Answers SQL Lab ✅
    - 3.4. Use Case 3: Lecturer Adds New Question ✅
    - 3.5. Activity Diagram ~
    - 3.6. Component Diagram ~
    - 3.7. State Diagram - Skyler
4. System Architecture
    - 4.1. System Architecture Diagram
    - 4.2. Sequence Diagrams - Skyler ✅
        - 4.2.1. Student
        - 4.2.2. Lecturer
    - 4.3. Data flow Diagram - Skyler ✅
        - 4.3.1. Level 0 Data Flow Diagram
        - 4.3.2. Level 1 Data Flow Diagram
    - 4.4. Collaboration Diagram ~
5. UI Mockups - Nishant
    - 5.1. RelaX integration
        - 5.1.1. Input Field (mimic RelaX)
        - 5.1.2. Submit vs. Run buttons
        - 5.1.3. Returned results of running
        - 5.1.4. Question generation
    - 5.2. SQL/DDL integration
        - 5.2.1. Input Field
        - 5.2.2. Submit vs. Run buttons
        - 5.2.3. Returned results of running
        - 5.2.4. Question generation
6. Technical Specifications - Nishant
    - 6.1. Docker
    - 6.2. Database
    - 6.3. Tools to Build Software
7. Testing - Andrei
    - 7.1. Overview ✅
    - 7.2. Software/Technologies ✅
    - 7.3. Continuous Integration/Deployment

1. Project Description

The project - “Automating Database Question Generation and Marking with PrairieLearn”, is to successfully automate relational algebra and SQL question delivery and evaluation on PrairieLearn for students at UBC-O for our client, Dr. Lawrence.

As of right now, students in COSC 304 are reliant on using multiple different platforms for their coursework - including time-sensitive evaluations such as midterms and final examinations. These tools include Canvas, PrairieLearn, RelaX, GitHub, and a DBMS software to run and verify their queries locally.

Our objective is to integrate questions from specific, existing labs for COSC 304 from Canvas and GitHub into PrairieLearn, integrate the necessary tools for these labs such as RelaX, and automate randomized question generation as well as their evaluation process. In addition, the project requires us to implement a feature that allows students to verify their answers on PrairieLearn so that they need not rely on the DBMS software.

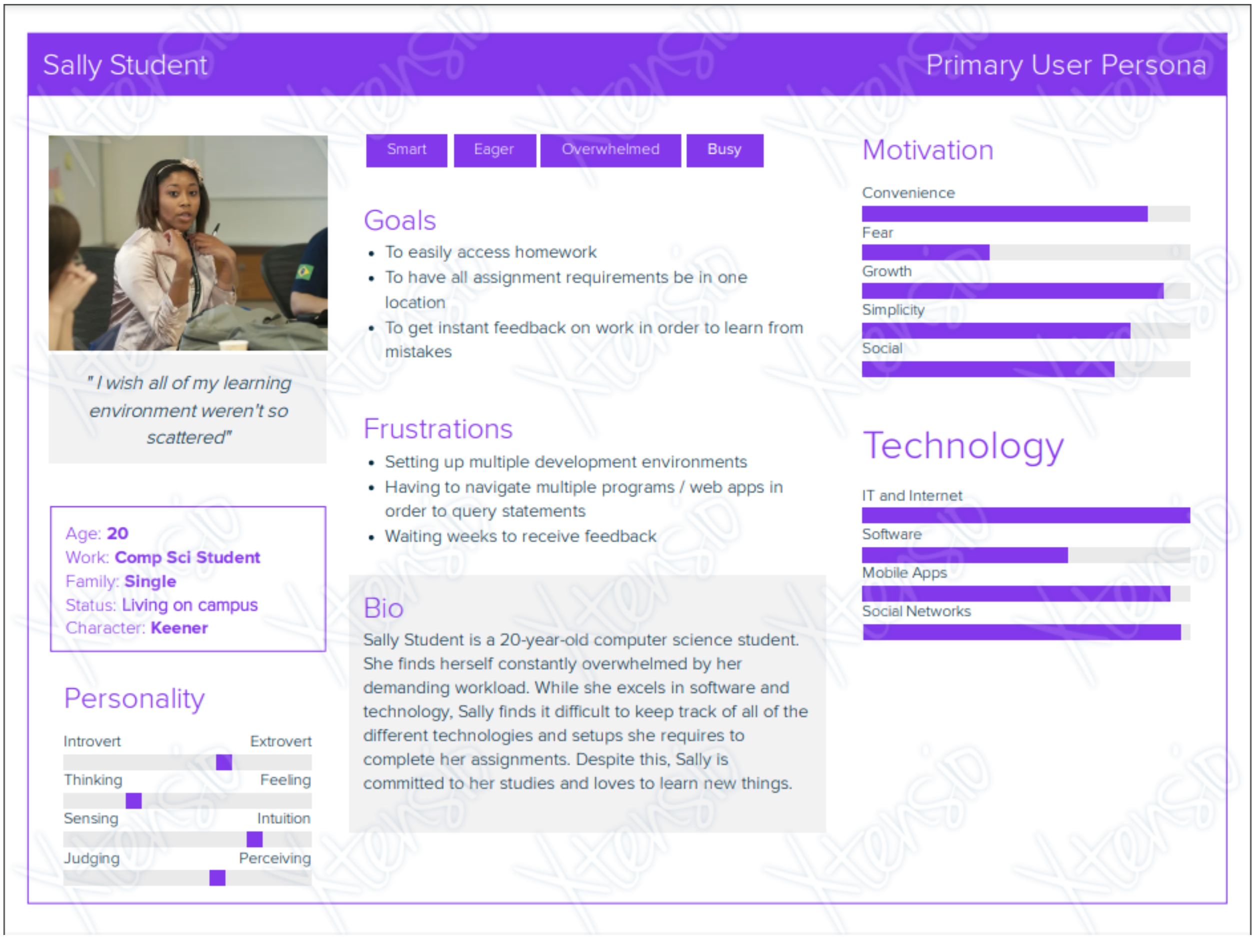
2. User Groups and Personas

2.1. Students

2.2. Administrators

2.2.1. Professors

2.2.2. Teaching Assistants



Sally Student

Primary User Persona

Smart | Eager | Overwhelmed | **Busy**

"I wish all of my learning environment weren't so scattered"

Age: **20**
Work: **Comp Sci Student**
Family: **Single**
Status: **Living on campus**
Character: **Keener**

Goals

- To easily access homework
- To have all assignment requirements be in one location
- To get instant feedback on work in order to learn from mistakes

Frustrations

- Setting up multiple development environments
- Having to navigate multiple programs / web apps in order to query statements
- Waiting weeks to receive feedback

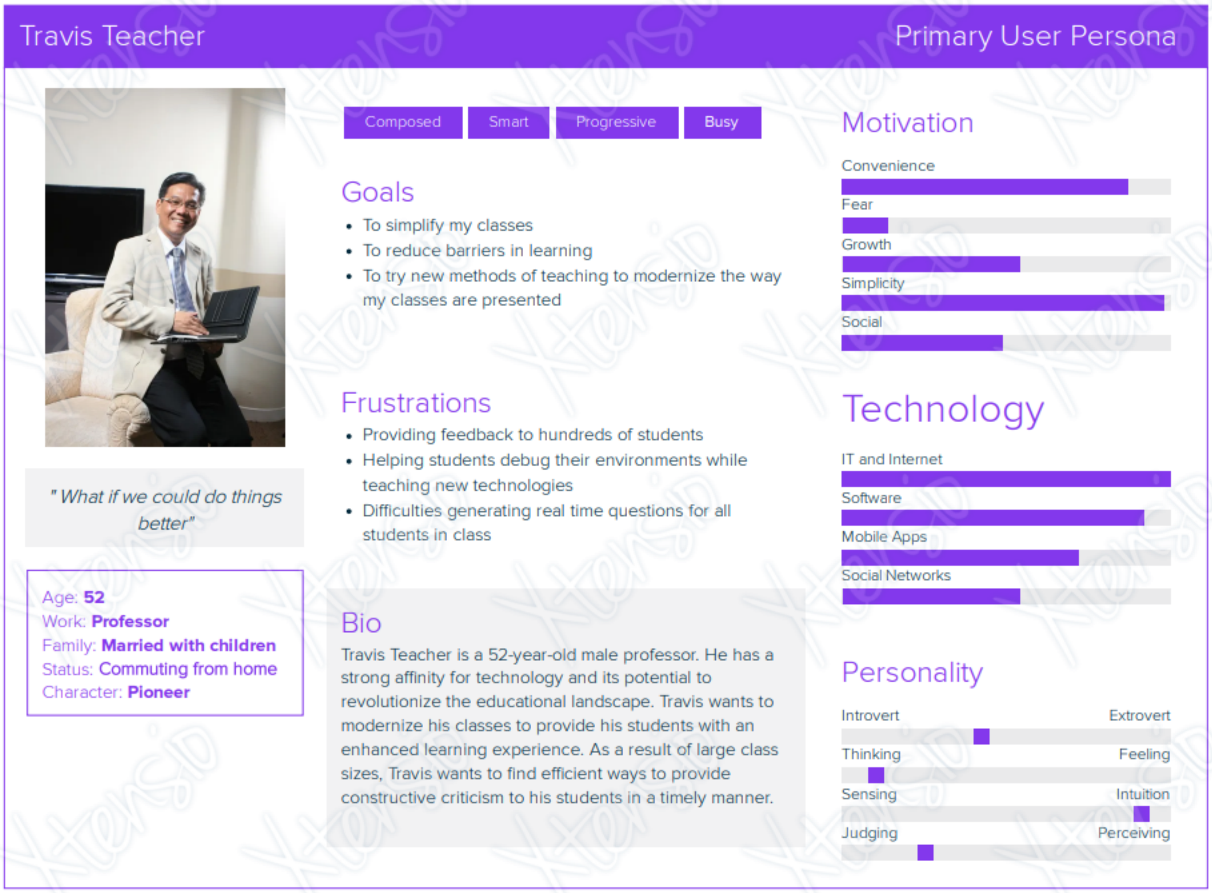
Bio

Sally Student is a 20-year-old computer science student. She finds herself constantly overwhelmed by her demanding workload. While she excels in software and technology, Sally finds it difficult to keep track of all of the different technologies and setups she requires to complete her assignments. Despite this, Sally is committed to her studies and loves to learn new things.

Personality

Motivation

Technology



Travis Teacher

Primary User Persona

Composed | Smart | Progressive | **Busy**

" What if we could do things better"

Age: **52**
Work: **Professor**
Family: **Married with children**
Status: **Commuting from home**
Character: **Pioneer**

Goals

- To simplify my classes
- To reduce barriers in learning
- To try new methods of teaching to modernize the way my classes are presented

Frustrations

- Providing feedback to hundreds of students
- Helping students debug their environments while teaching new technologies
- Difficulties generating real time questions for all students in class

Bio

Travis Teacher is a 52-year-old male professor. He has a strong affinity for technology and its potential to revolutionize the educational landscape. Travis wants to modernize his classes to provide his students with an enhanced learning experience. As a result of large class sizes, Travis wants to find efficient ways to provide constructive criticism to his students in a timely manner.

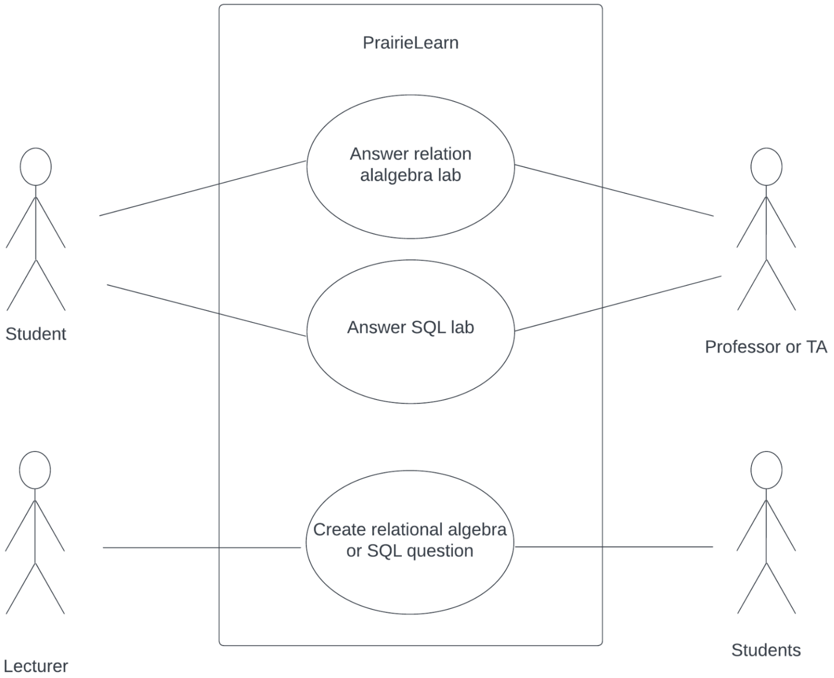
Motivation

Technology

Personality

3. Use Cases

3.1. Use Cases Diagram



PrairieLearn
Answer relation alalgebra lab
Answer SQL lab
Create relational algebra or SQL question
Student
Professor or TA
Lecturer
Students

Figure 3.1: *A student may interact with the system (PrairieLearn) either through the Answer Relation Algebra Lab or Answer SQL Lab. The Professor or TA are able to see the student's grades after they have submitted their answer. Additionally, a Lecturer may create a new relational algebra or SQL question, which the students may answer.*

3.2. Use Case 1: Student Answers Relation Algebra Lab

Primary actor: Student
Description: A student wishes to complete their lab on relation algebra
Precondition: Student has successfully logged in to PrairieLearn and selected the relation algebra question.
Post condition: If the student successfully accessed the lab and submitted the question, then all questions will have been marked and the student would have received feedback.

Main Scenario:

1. Student selects the question
2. Student enters their answer in the RelaX editor
3. Students tests their query without submission
4. The output of the query is displayed for the student
5. The student submits their answer
6. The question is automatically graded and the student receives appropriate feedback

Extensions:

3a. The student's answer includes one or more formatting errors
 3a1. System issues an error message, informing the lecturer of the error and prevents submission
4a. Student is unsatisfied by the results of their test query
 4a1. Student returns to step 2, entering a new answer
5a. Student's answer includes one or more formatting errors
 5a1. System issues an error message, informing the lecturer of the error and prevents submission

3.3. Use Case 2: Student Answers SQL Lab

Use case 2: Answer an SQL lab
Primary actor: Student
Description: A student wishes to complete their lab on SQL
Precondition: Student has successfully logged in to PrairieLearn and selected the relation algebra question.
Post condition: If the student successfully accessed the lab and submitted the question, then all questions will have been marked and the student would have received feedback.

Main Scenario:

1. Student selects the question

2. Student enters their answer in the SQL editor
3. Students tests their query without submission
4. The output of the query is displayed for the student
5. The student submits their answer
6. The question is automatically graded and the student receives appropriate feedback

Extensions:

- 3a. The student’s answer includes one or more formatting errors
  - 3a1. System issues an error message, informing the lecturer of the error and prevents submission
- 4a. Student is unsatisfied by the results of their test query
  - 4a1. Student returns to step 2, entering a new answer
- 5a. Student’s answer includes one or more formatting errors
  - 5a1. System issues an error message, informing the lecturer of the error and prevents submission

3.4. Use Case 3: Lecturer Adds New Question

Use case 3: Creating a new relation algebra or SQL question

Primary actors: Lecturer

Description: During a lecture, the lecturer wishes to quiz the class on a relation algebra or SQL topic, so they create a new question

Precondition: The lecturer and students are all able to access PrairieLearn

Postcondition: The students have received automatic feedback on their responses and the lecturer may see the class’ performance

Main scenario:

1. Lecturer selects question type as either relation algebra or SQL
2. Lecturer enters the question
3. Lecturer submits the question
4. Lecturer observes the students’ performance using PrairieLearn

Extensions:

- 3a. Lecturer’s question contains one or more errors
  - 3a1. System issues an error message, informing the lecturer of the error and prevents submission

4. System Architecture

4.1. Sequence Diagrams

4.1.1. Student

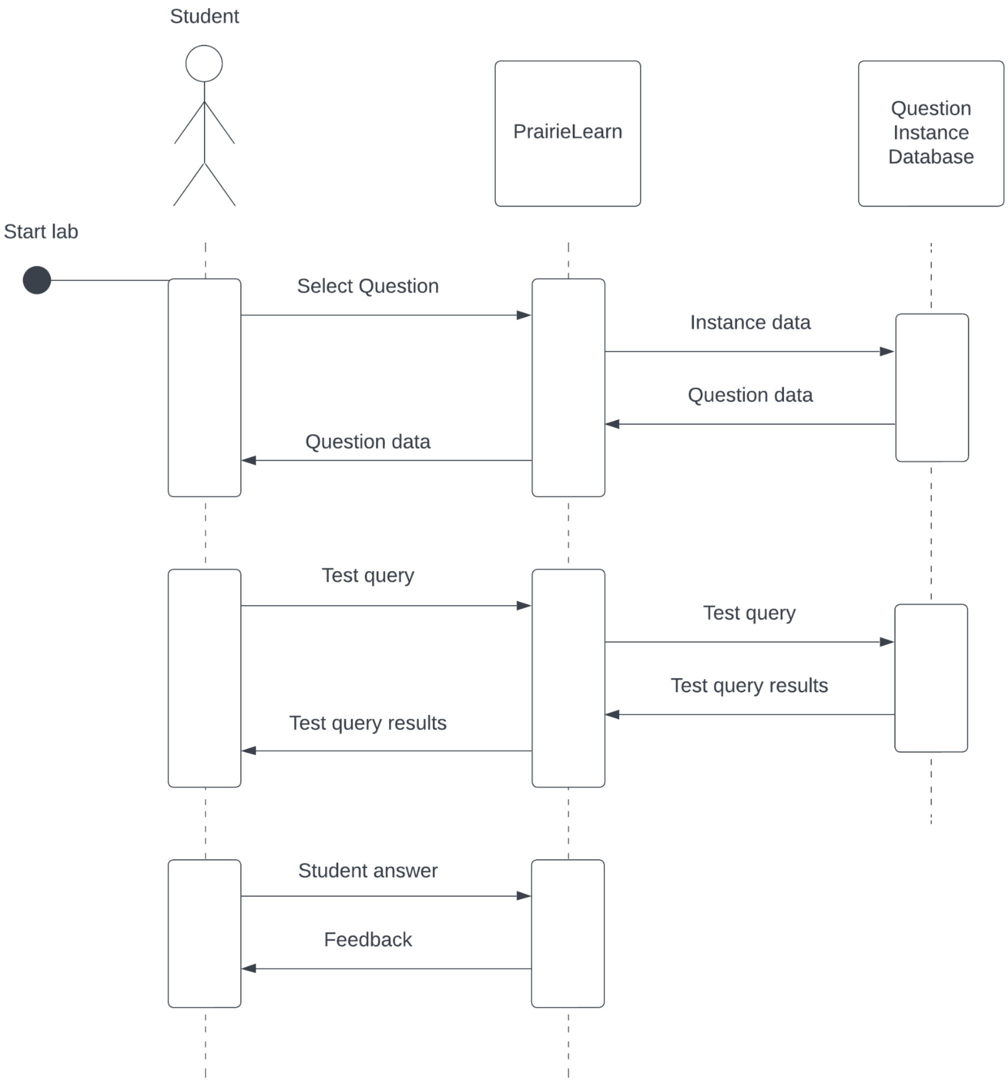


Figure 4.1.1: *A student begins their lab by selecting a question. PrairieLearn then generates the question data and instantiates an instance of a database for that question. The student then enters a query and tests it so PrairieLearn sends that query to the database instance to retrieve its results, which are then displayed to the student. The student ends by submitting their answer to PrairieLearn, which returns feedback for the Student.*

4.1.2. Lecturer

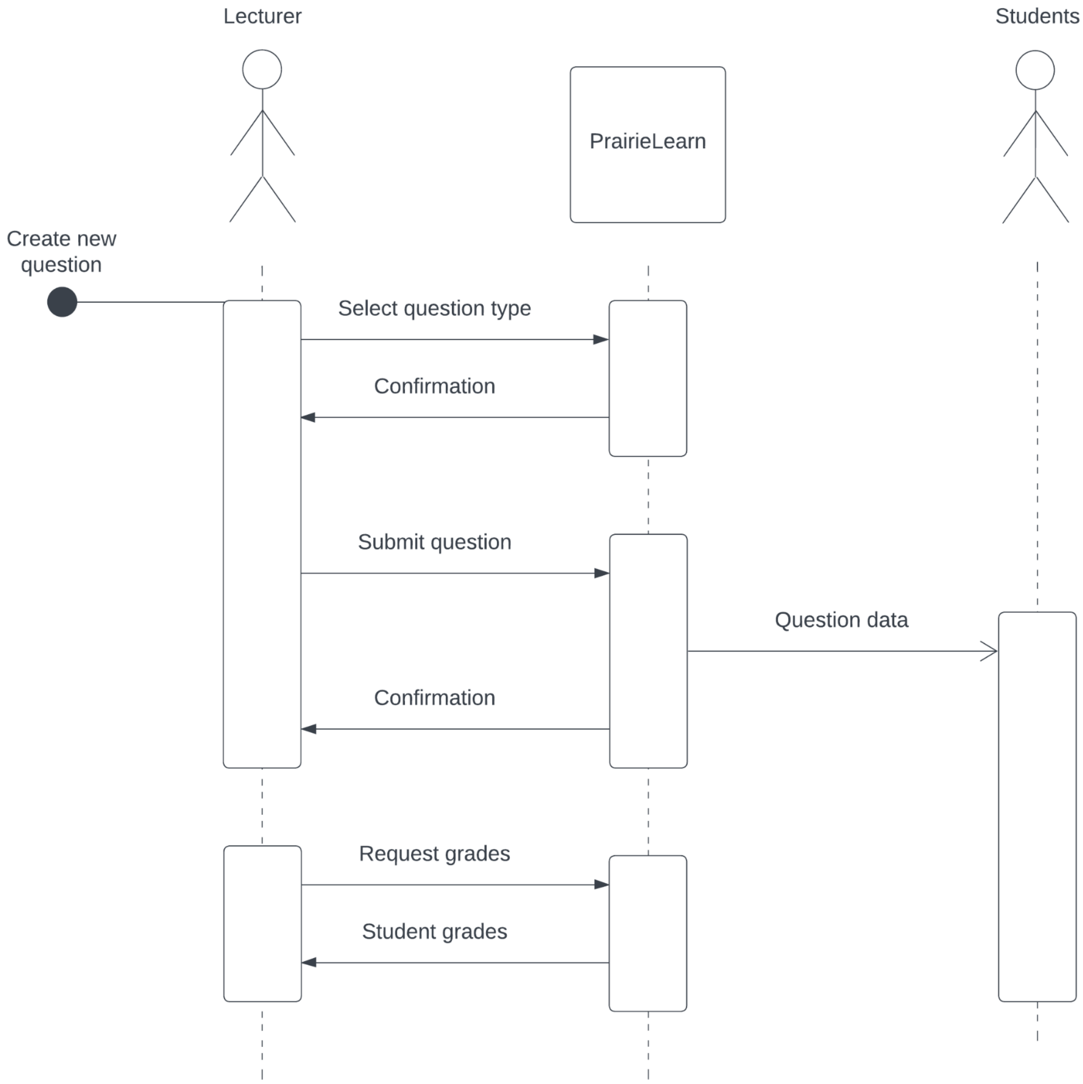


Figure 4.1.2: *A lecturer begins to create a new question by selecting the question type from either relation algebra or SQL. The lecturer then submits the question to PrairieLearn, which then makes the question available for students. The lecturer may then see the student's performance by checking PrairieLearn's student grade information.*

4.2. Data flow Diagram

4.2.1. Level 0 Data Flow Diagram

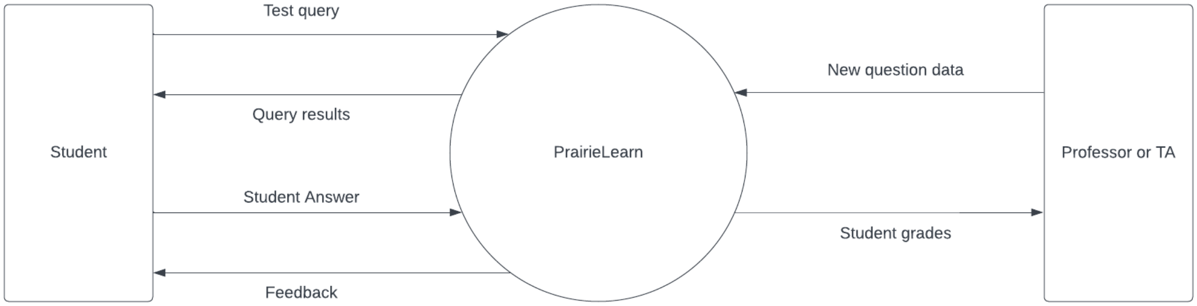


Figure 4.2.1: *The test query a student submits is sent to PrairieLearn, which returns the query's results. When a student submits their answer, PrairieLearn gives the student feedback. A Professor or TA may send question data in order to create a new question. A professor or TA may also receive student grades from PrairieLearn.*

4.2.2. Level 1 Data Flow Diagram

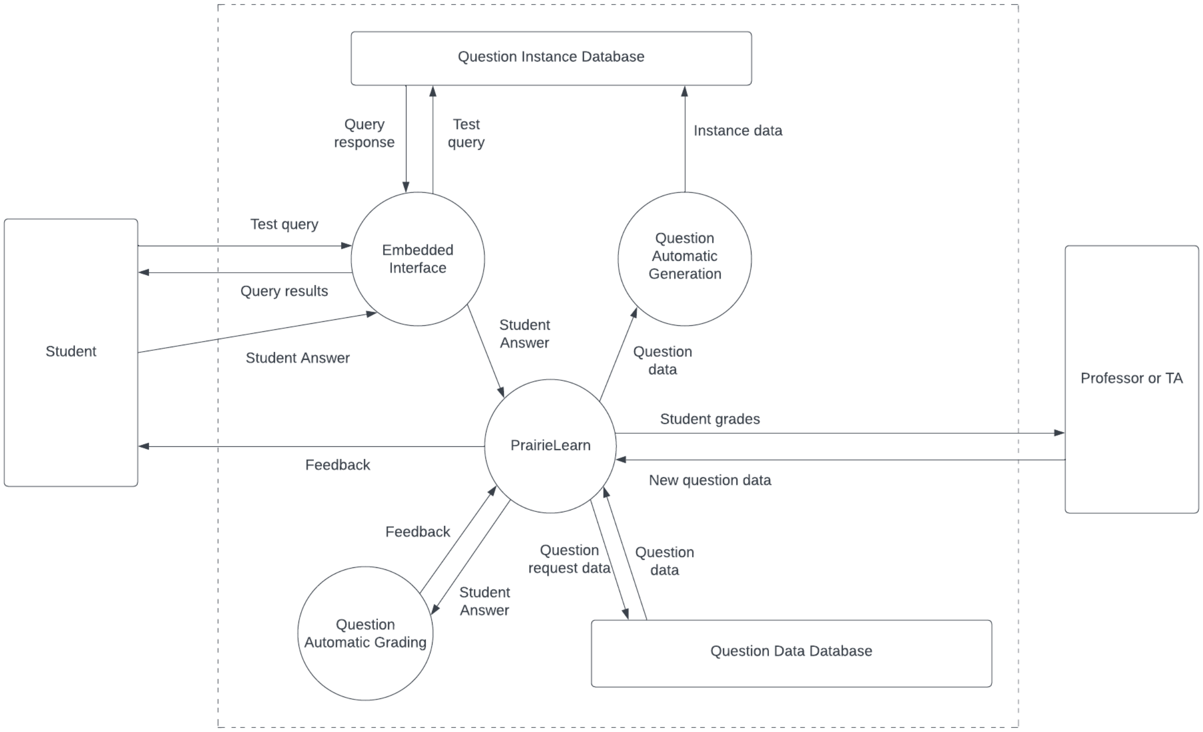


Figure 4.2.2: *This figure has the same information as figure 4.2.1 but has expanded upon the PrairieLearn system. The student sends their query to an embedded interface, which redirects it to a database instance; the database instance returns the query's results, which are displayed using the interface. A student submits their answer through the embedded interface and to PrairieLearn, where it is then sent to be automatically graded. The student's feedback is returned through PrairieLearn and then to the student. A professor or TA creates a new question, whereupon it is added to a questions database. When a student selects a question, the database sends the question's data through PrairieLearn to an automatic question generation, which populates a question instance database. When a professor or TA requests student grades, PrairieLearn returns the appropriate data.*

5. UI Mockups

5.1.

5.2.

6. Technical Specifications

6.1. Frontend

Integration and development of front-end development will continue to be in JavaScript, HTML, and CSS.

6.2. Backend

Back-end development will continue to be in Python and Node.js.

6.3. Docker

As per the client's request, there will be a docker container instantiated for each student to respond to questions in-class.

6.4. Database

The data for the labs (questions and solutions) will be stored using PostgreSQL.

6.5. Tools to Build Software

- IDE: Visual Studio Code
- Time Tracking: Clockify
- Task Management: GitHub Projects
- CI/CD: DroneCI

7. Testing

7.1. Overview

Testing is a crucial component of the software development process. It ensures that the system meets all of our specified requirements and functions as the client intends. In this design document, we explain the various types of testing that will be applied and the purpose that it serves. These include: unit testing, integration testing, performance testing, and functionality testing. As we are following a test-driven development structure, these tests will guide our development and reduce the amount of resources dedicated to bug fixes later in development.

7.2. Unit Testing (Structural)

Unit testing is a white-box testing technique. It involves designing tests for each function to ensure that it produces the expected output. At minimum one unit test will be designed prior to a function being created with additional unit tests designed during development. The goal is to achieve coverage testing to a degree to ensure that the majority of our code is tested. However, irrelevant tests will not be designed solely for the purpose of increasing our coverage. An example of a unit test would be an empty submission correctly being given a score of 0 ensuring that the scoring function works even without input.

7.3. Integration Testing

Integration testing is a black-box testing technique. This verifies whether multiple components being used together are working appropriately and producing the correct result. It examines the interaction between subsystems and identifies any issues that may arise from these interactions. Integration tests will be written to guide how features will be designed as a whole. An example of an integration test is appropriately handling user submitted strings in a textbox and ensuring that SQL code is then querying the database appropriately.

7.4. Performance Testing

Performance testing is essential to assess the system's ability to scale appropriately. One of the requirements of the system is to function concurrently between hundreds of students and a multitude of classes. As a result, the system will need to be stress tested to determine whether performance dips below acceptable values. We will progressively increase the submission load on the system and measure the response time. In doing so, we may be able to determine bottlenecks and optimize resource allocation.

7.5. Functionality Testing

Functional testing will determine whether the system is able to meet the requirements and fulfill its intended purpose. In our case, the system's primary objective is to allow for enrolled students to complete the first three labs in COSC 304. An example of functional testing would be to display the resulting tables of an SQL query in the web page when a student runs a query.

7.6. Software/Technologies

Python unittest framework

Python code for: the rendering of questions, automatic marking, and automatic question generation will be tested with python unittest framework.

Mocha

Any front-end results being displayed after a query is submitted will need to be written in Javascript. This will be tested using the Mocha framework for JS.

7.7. Continuous Integration/Deployment