Project Name: Automating Database Question Generation and Marking with PrairieLearn

Design Documents

Version:
Draft 1 - June 4th, 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
3. Use Cases
    3.1. Use Cases Diagram
    3.2. Use Case 1: Student Answers Relation Algebra Lab
    3.3. Use Case 2: Student Answers SQL Lab
4. System Architecture
    4.1. Sequence Diagram
    4.2. Data flow Diagrams
        4.2.1. Level 0 Data Flow Diagram
        4.2.2. Level 1 Data Flow Diagram
5. UI Mockups
    5.1. RelaX Integration
    5.2. SQL/DDL Integration
6. Technical Specifications
    6.1. Docker
    6.2. Database
    6.3. Tools to Build Software
7. Testing
    7.1. Overview
    7.2. Software/Technologies
    7.3. Unit
    7.4. Integration
    7.5. Performance
    7.6. UI
    7.7. Functionality
    7.8. Continuous Integration/Deployment

1. Project Description

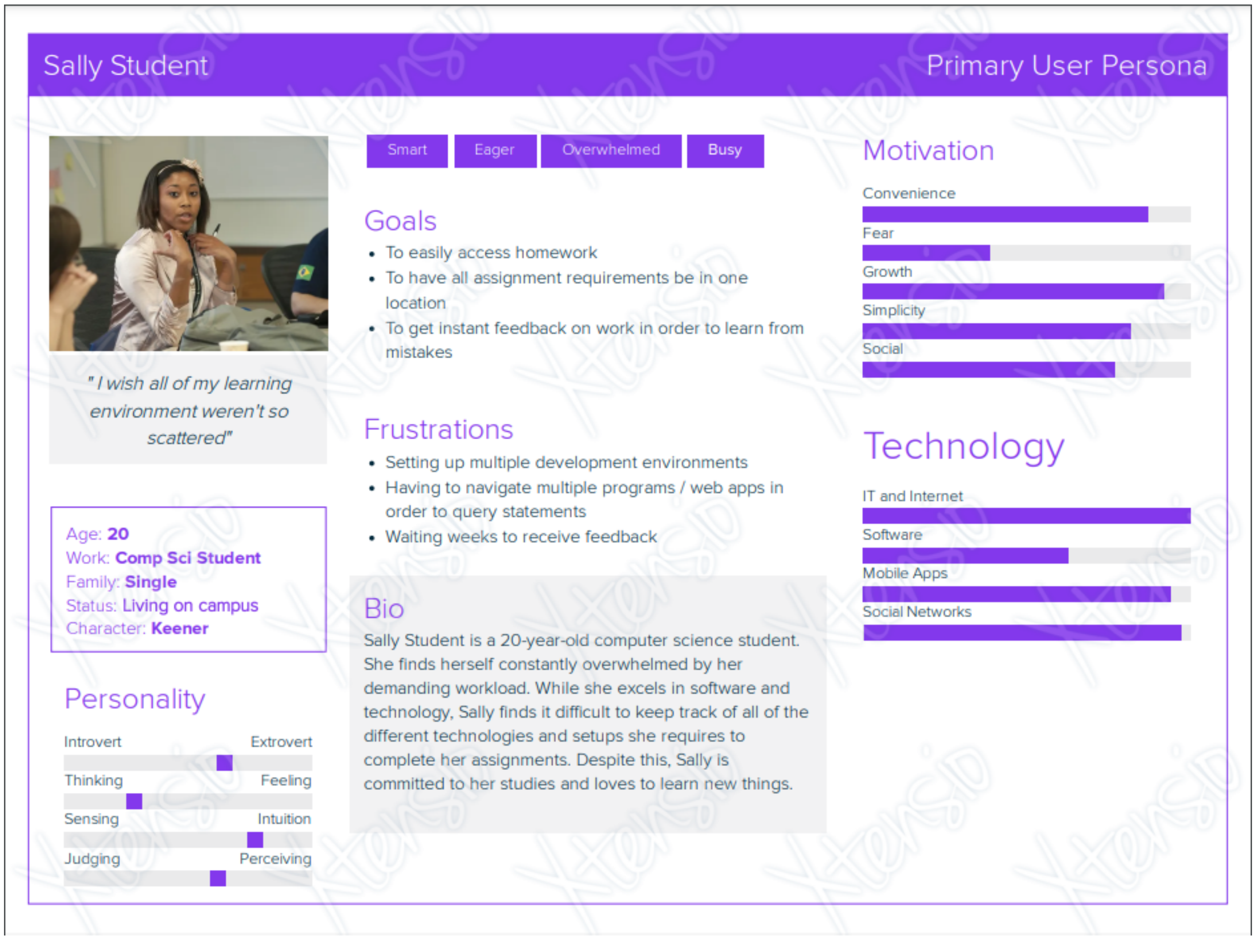
The project - “Automating Database Question Generation and Marking with PrairieLearn”, is to successfully automate relational algebra and SQL question delivery and evaluation on PrairieLearn for students at UBC-O for our client, Dr. Lawrence.

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



Sally Student

Primary User Persona

"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**

Personality

Introvert
Extrovert
Thinking
Feeling
Sensing
Intuition
Judging
Perceiving

Smart | Eager | Overwhelmed | Busy

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

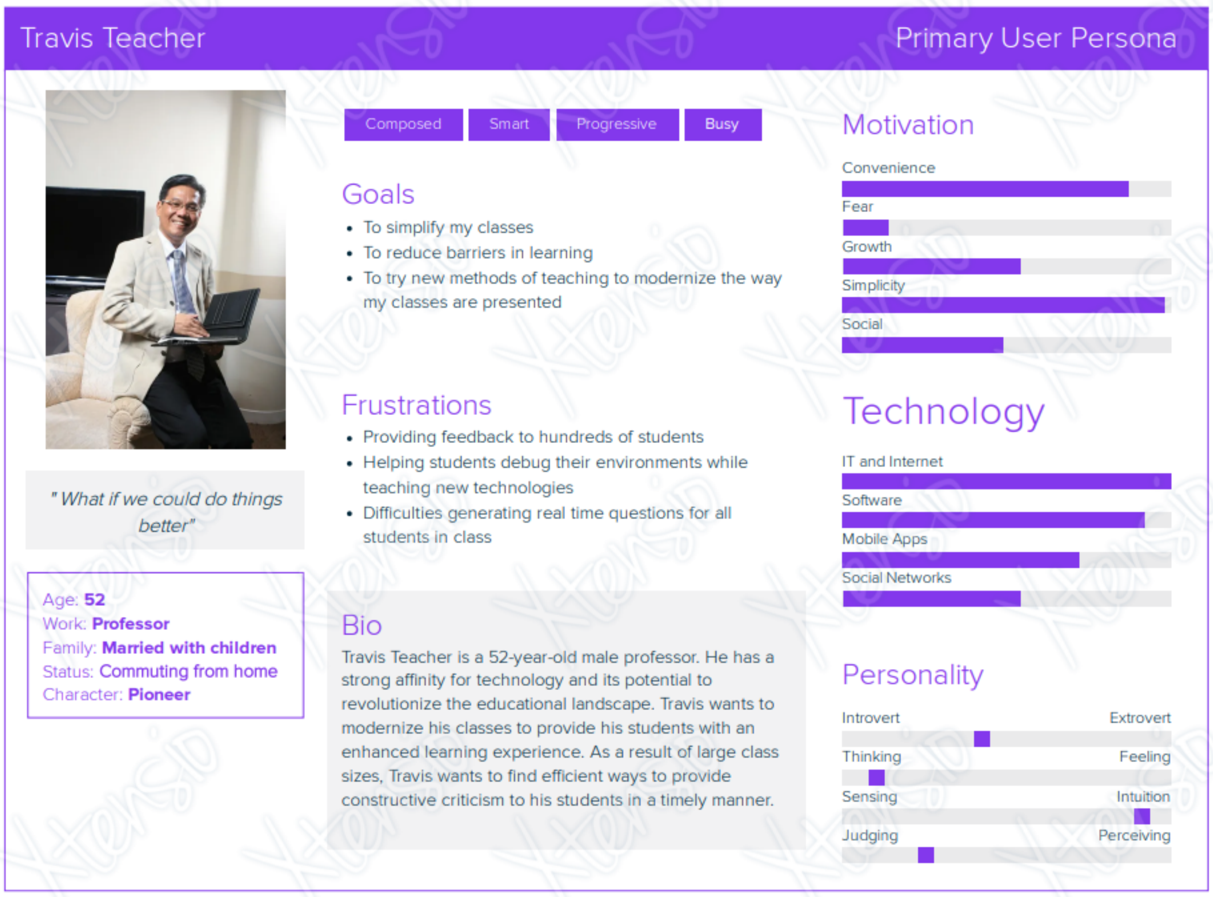
Sally Student is a 20-year-old computer science student. She finds herself constantly overwhelmed by her demanding workload. While she excels in software and technology, Sally finds it difficult to keep track of all of the different technologies and setups she requires to complete her assignments. Despite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

Motivation

Technology

Figure 2.1: *A persona for a student user profile named Sally Student.*

2.2. Professor


Travis Teacher

Primary User Persona

"What if we could do things better"

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

Composed | Smart | Progressive | Busy

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.

Figure 2.2: *A persona for a professor user profile named Travis Teacher.*

3. Use Cases

4. Use Cases

These use cases build upon existing PrairieLearn documentation and only include items modified by this project. They should not be taken as a complete set of design documents.

4.1. Student Use Cases Diagram

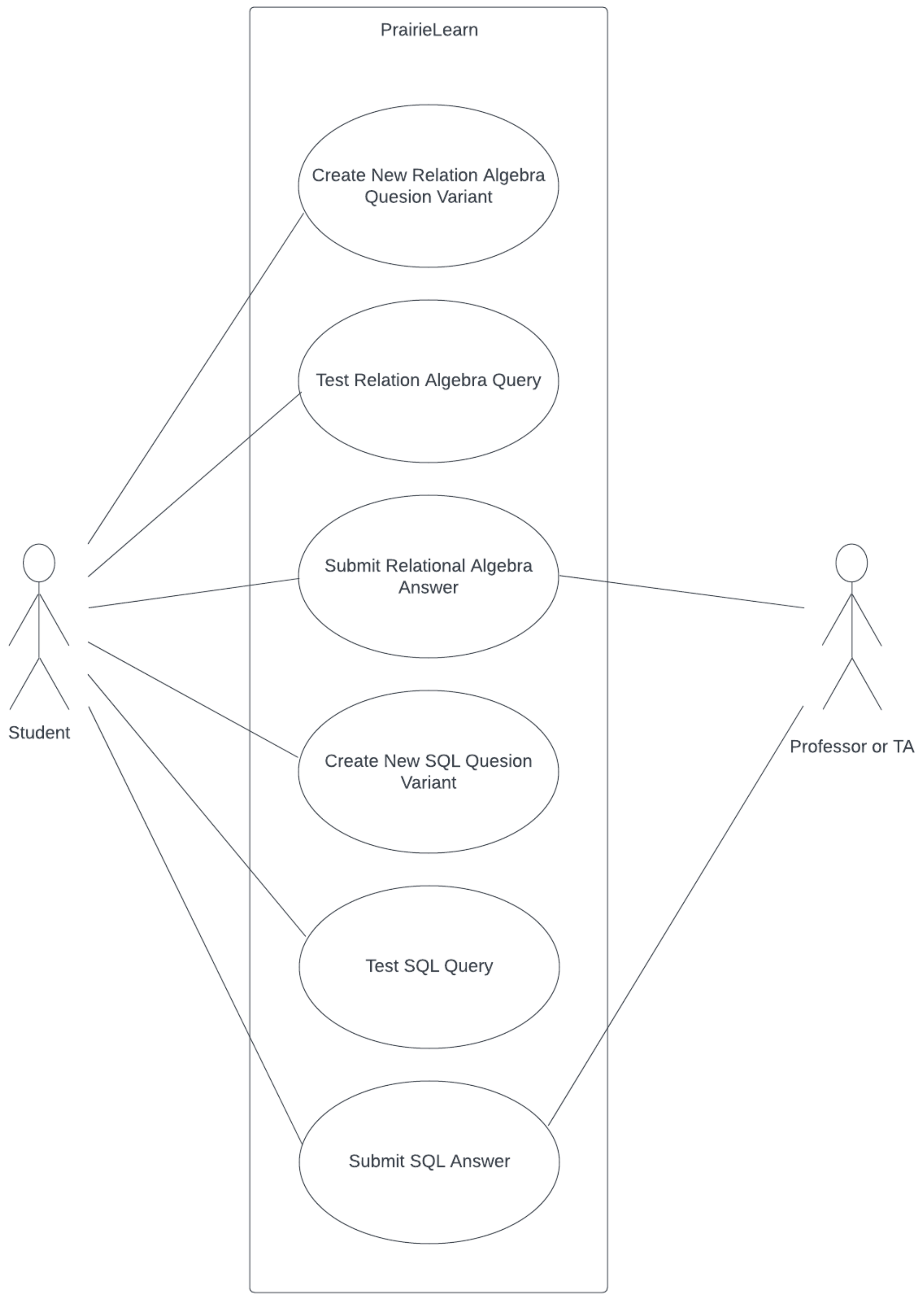

PrairieLearn
Create New Relation Algebra Quesion Variant
Test Relation Algebra Query
Submit Relational Algebra Answer
Create New SQL Quesion Variant
Test SQL Query
Submit SQL Answer
Student
Professor or TA

Figure 16.1: *A student may interact with the system (PrairieLearn) either through a Relation Algebra Lab or Answer SQL Lab. In either lab, the student may generate new question variants, test queries, and submit an answer. The professor or TA are able to see the student's grades after they have submitted their answer.*

4.2. Use Case 1: Create New Relational Algebra Variant

ID: 1
Primary actor: Student
Description: A student wishes to obtain a new question variant
Precondition: Student has successfully logged in to PrairieLearn and selected a relation algebra question.
Postcondition: If the student has successfully accessed the question and generates a new question variant, the student sees a new variant of the relational algebra question.

Main Scenario:

1. Student selects the relation algebra question.
2. Student selects “Generate New Variant”.

Extensions:

None

4.3. Use Case 2: Test Relational Algebra Query

ID: 2
Primary actor: Student
Description: A student wishes to receive feedback on their answer prior to submission.
Precondition: Student has successfully logged in to PrairieLearn and selected the relation algebra question.
Postcondition: If the student successfully accessed the lab and formats their question, the student will see the results of their query.

Main Scenario:

1. Student selects the relation algebra 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.

Extensions:

3a. The student’s answer includes one or more formatting errors.
  3a1. System issues an error message, informing the student of the error and prevents the query from being run.

4.4. Use Case 3: Submit Relational Algebra Answer

ID: 3
Primary actor: Student
Description: A student wishes to submit their answer for grading.

Precondition: Student has successfully logged in to PrairieLearn and selected the relation algebra question.
Postcondition: If the student successfully accessed the lab and formats their question, the student will receive feedback on their answer.

Main Scenario:

1. Student selects the relation algebra question.
2. Student enters their answer in the RelaX editor.
3. Student submits their answer.
4. Student receives feedback on their answer.

Extensions:

3a. The student's answer includes one or more formatting errors.
  3a1. System issues an error message, informing the student of the error and prevents the query from being run.
3b. The student's answer is incorrect and the student has remaining attempts.
  3b1. The student is informed their answer is incorrect and is prompted to try again.
3c. The student's answer is incorrect and the student has no remaining attempts.
  3c1. The student is informed their answer is incorrect.

4.5. Use Case 4: Create New SQL Variant

ID: 4
Primary actor: Student
Description: A student wishes to obtain a new question variant
Precondition: Student has successfully logged in to PrairieLearn and selected an SQL question.
Postcondition: If the student has successfully accessed the question and generates a new question variant, the student sees a new variant of the SQL question.

Main Scenario:

1. Student selects the SQL question.
2. Student selects "Generate New Variant".

Extensions:

None

4.6. Use Case 5: Test SQL Query

ID: 5
Primary actor: Student
Description: A student wishes to receive feedback on their answer prior to submission.
Precondition: Student has successfully logged in to PrairieLearn and selected the SQL question.

Postcondition: If the student successfully accessed the lab and formats their question, the student will see the results of their query.

Main Scenario:

1. Student selects the SQL 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.

Extensions:

- 3a. The student's answer includes one or more formatting errors.
  - 3a1. System issues an error message, informing the student of the error and prevents the query from being run.

4.7. Use Case 6: Submit SQL Answer

ID: 6
Primary actor: Student
Description: A student wishes to submit their answer for grading.
Precondition: Student has successfully logged in to PrairieLearn and selected the SQL question.
Postcondition: If the student successfully accessed the lab and formats their question, the student will receive feedback on their answer.

Main Scenario:

1. Student selects the SQL question.
2. Student enters their answer in the SQL editor.
3. Student submits their answer.
4. Student receives feedback on their answer.

Extensions:

- 3a. The student's answer includes one or more formatting errors.
  - 3a1. System issues an error message, informing the student of the error and prevents the query from being run.
- 3b. The student's answer is incorrect and the student has remaining attempts.
  - 3b1. The student is informed their answer is incorrect and is prompted to try again.
- 3c. The student's answer is incorrect and the student has no remaining attempts.
  - 3c1. The student is informed their answer is incorrect.

4.8. Use Case 7: See Correct Answer After Submission

ID 7:
Primary actor: Student
Description: A student wishes to complete their question to see if it is correct.

Precondition: Student has successfully logged in to PrairieLearn and selected the desired question.
Post condition: If the student successfully accessed the lab, submitted the question, and the professor has set the question's solution to be visible, then the question will be marked and the student will see the correct answer.

Main Scenario:

1. Student selects the desired question.
2. Student enters their answer.
3. Student submits their answer.
4. Student receives feedback on their answer and is able to see the correct solution.

Extensions:

3a. The student's answer includes one or more formatting errors.
    3a1. System issues an error message, informing the student of the error and prevents the query from being run.
3b. The student's answer is incorrect and the student has remaining attempts.
    3b1. The student is informed their answer is incorrect and is prompted to try again.
3c. The student's answer is incorrect and the student has no remaining attempts.
    3c1. The student is informed their answer is incorrect.

4.9. Professor Use Case Diagram

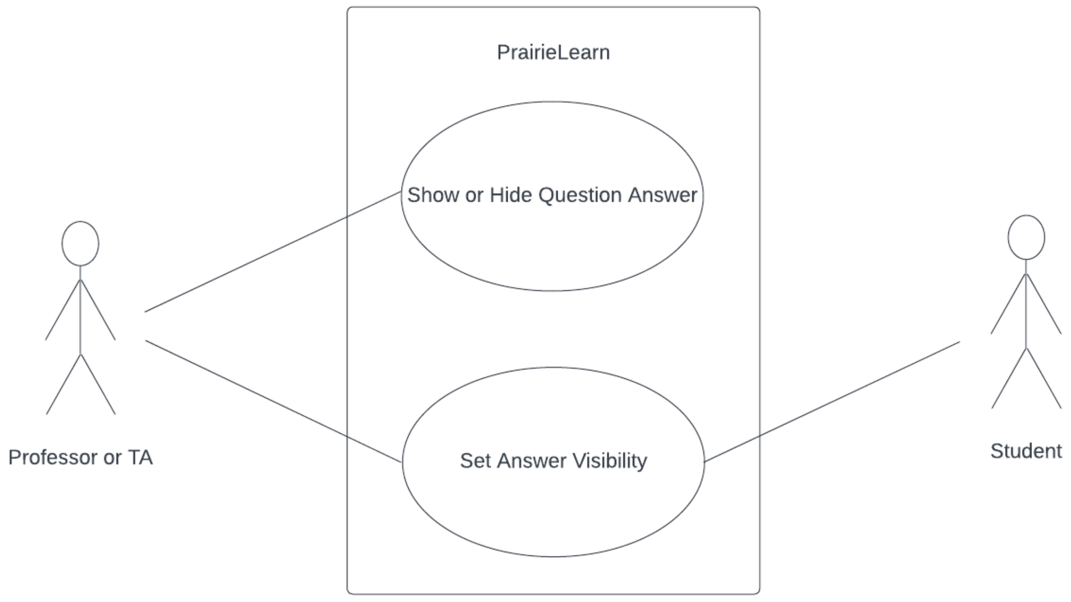


Figure 16.x: *A professor or TA is able to interact with the system (PrairieLearn) either by viewing a question's answer or by setting the visibility of a solution after the student submits an answer.*

4.10. Use Case 8: Show or Hide Answer

ID: 8
Primary actor: Professor
Description: A professor wishes to view the correct answer to a desired question.
Precondition: Professor has successfully logged into PrairieLearn and selected the desired question.
Post Condition: If the professor successfully accessed the lab, then the professor sees the question solution.

Main Scenario:
1. Professor selects the desired question.
2. Professor selects “Show/Hide Answer”.

Extensions:
None

4.11. Use Case 9: Set Answer Visibility

ID: 9
Primary actor: Professor
Description: A professor wishes to change whether the answer is viewable after the question is answered by a student.
Precondition: Professor has successfully logged into PrairieLearn and selected the desired question.
Post Condition: If the professor has successfully logged into PrairieLearn, then the question visibility is set.

Main Scenario:
1. Professor selects the desired question.
2. Professor navigates to the “Files” tab.
3. Professor selects “info.json”.
4. Professor sets desired visibility.

Extensions:
None

5. System Architecture

5.1. Sequence Diagram

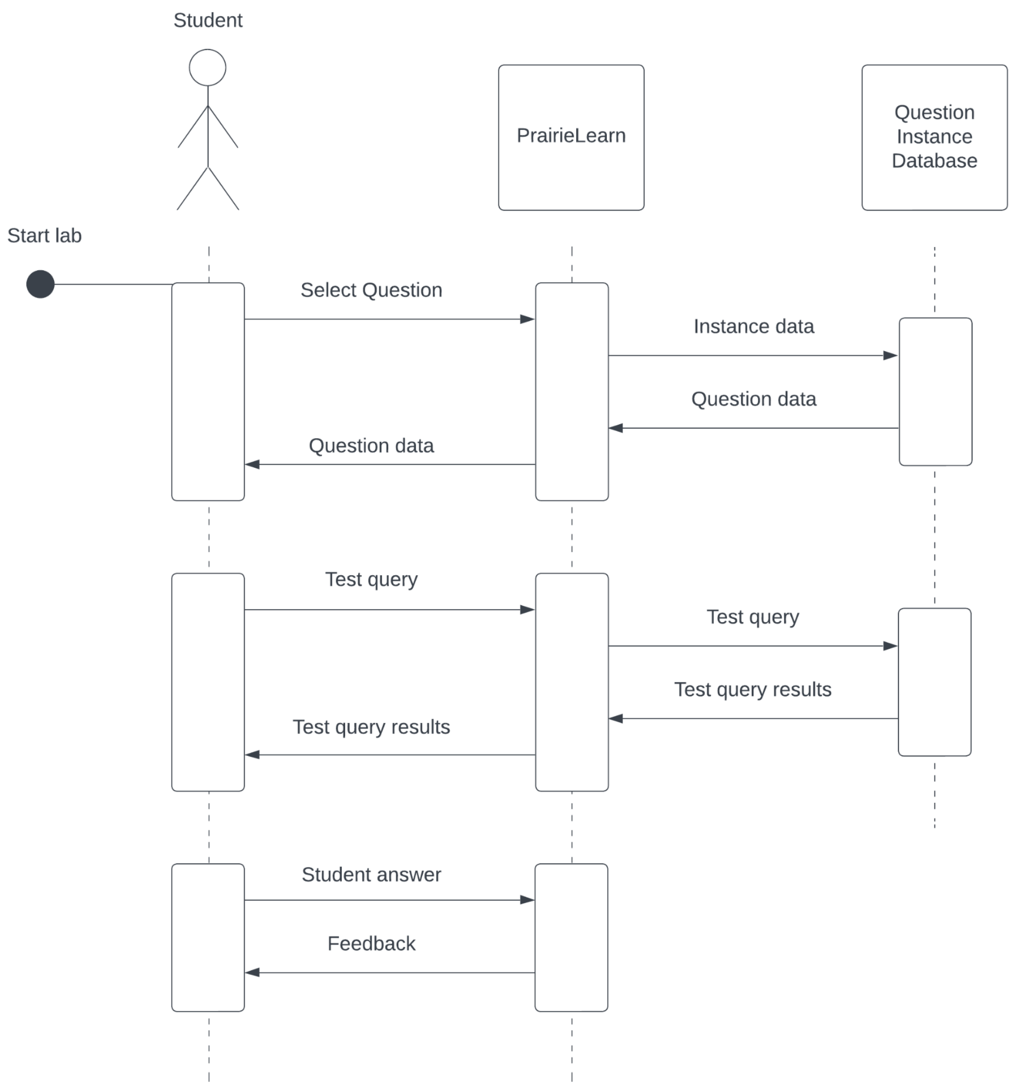


Figure 4.1: *A student begins their lab by selecting a question. PrairieLearn then generates the question data and instantiates an instance of a database for that question. The student then enters a query and tests it so PrairieLearn sends that query to the database instance to retrieve its results, which are then displayed to the student. The student ends by submitting their answer to PrairieLearn, which returns feedback for the Student.*

5.2. Data flow Diagram

5.2.1. Level 0 Data Flow Diagram

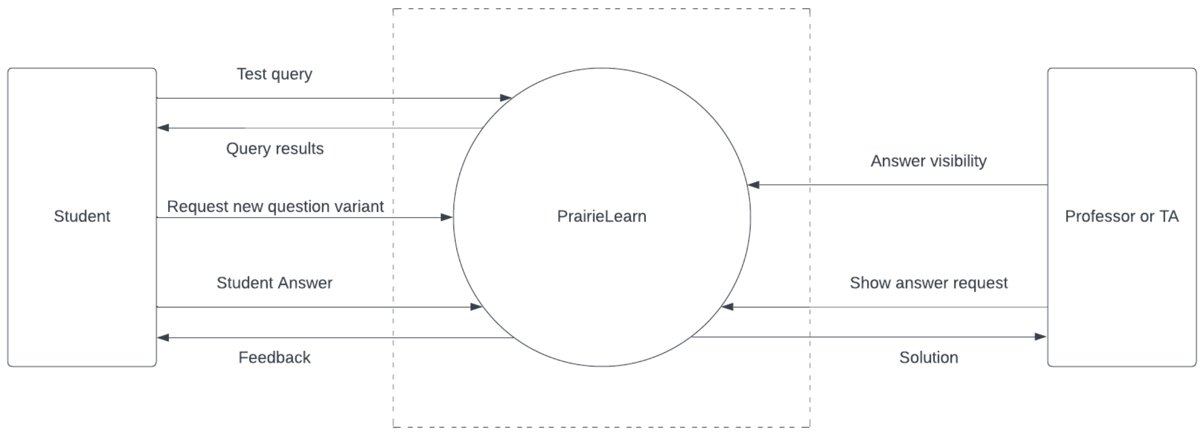


Figure 4.2.1: *The test query a student submits is sent to PrairieLearn, which returns the query's results. When a student submits their answer, PrairieLearn gives the student feedback. A student may request a new question variant. A professor or TA can show or hide the solution or set question visibility.*

5.2.2. Level 1 Data Flow Diagram

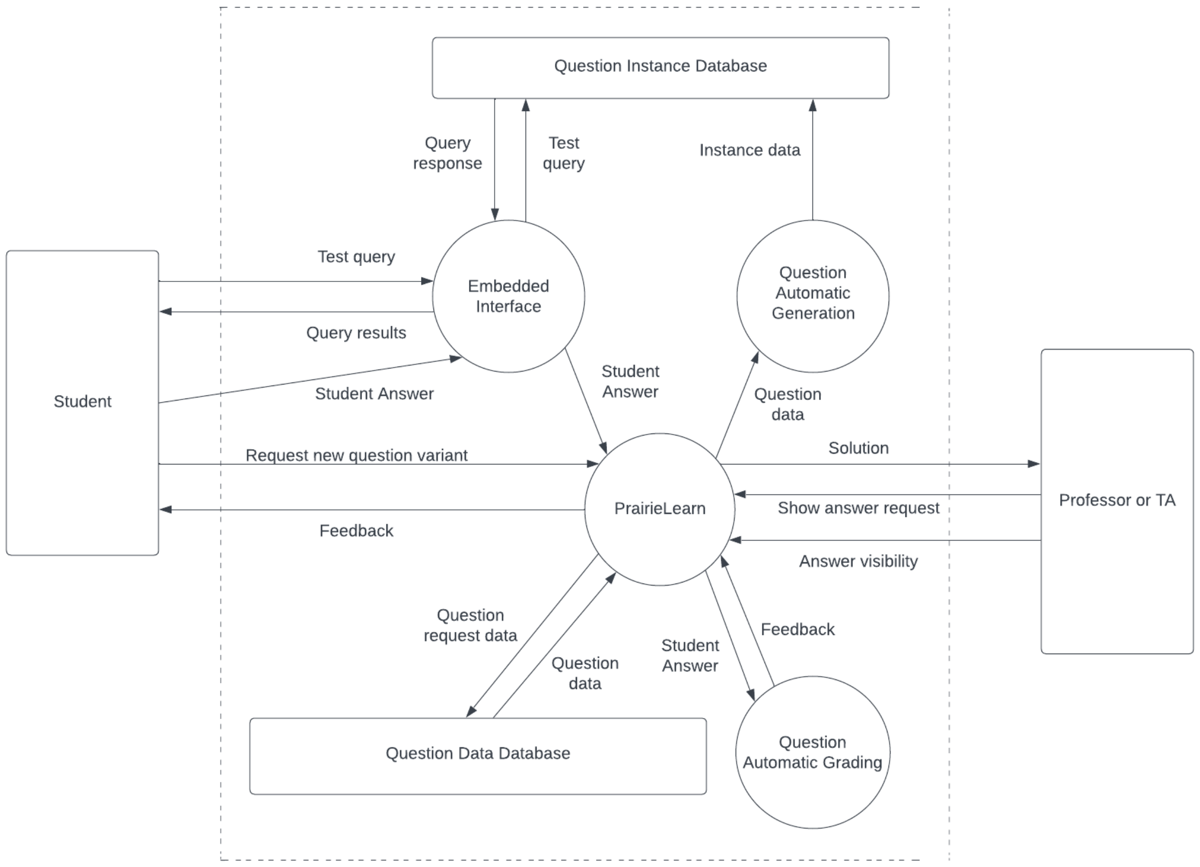
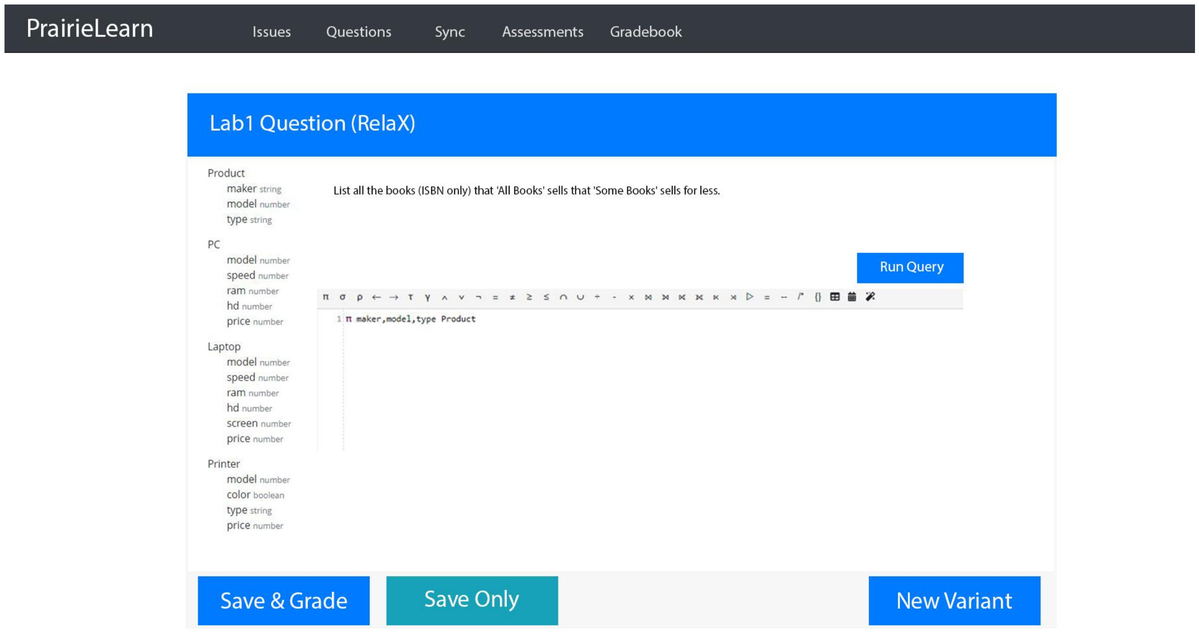


Figure 4.2.2: *This figure has the same information as figure 4.2.1 but has expanded upon the PrairieLearn system. The student sends their query to an embedded interface, which redirects it to a database instance; the database instance returns the query's results, which are displayed using the interface. 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.*

6. UI Mockups

6.1. RelaX Integration



Product
- maker string
- model number
- type string

PC
- model number
- speed number
- ram number
- hd number
- price number

Laptop
- model number
- speed number
- ram number
- hd number
- screen number
- price number

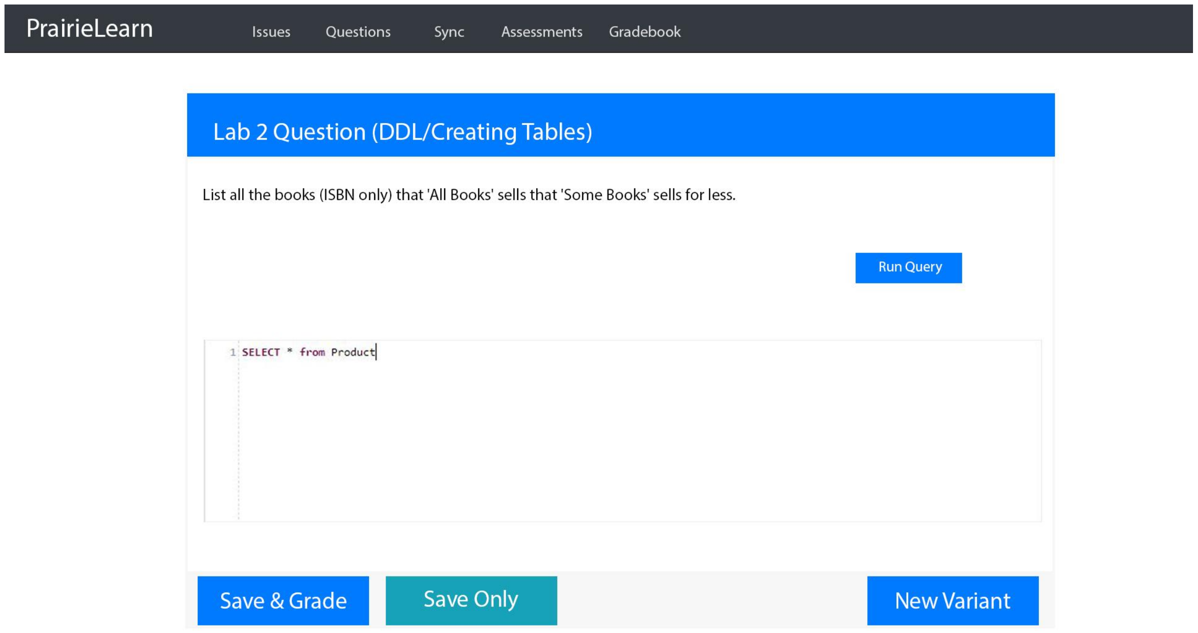
Printer
- model number
- color boolean
- type string
- price number

List all the books (ISBN only) that 'All Books' sells that 'Some Books' sells for less.

```
π maker,model,type Product
```

Figure 5.1: *A mock UI for the RelaX editor within PrairieLearn.*

6.2. SQL/DDL Integration



List all the books (ISBN only) that 'All Books' sells that 'Some Books' sells for less.

```
SELECT * from Product
```

Figure 5.2: *A mock UI for the SQL editor within PrairieLearn.*

7. Technical Specifications

7.1. Frontend

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

7.2. Backend

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

7.3. Database

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

7.4. Tools to Build Software

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

8. Testing

8.1. 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.

8.2. 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.

8.3. 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.

8.4. 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.

8.5. UI Testing

UI testing allows us to ensure that various components of the software's UI, both new and old, are working as required. In this project some of the new UI components will be the button for running queries without submitting, displaying results of those queries in tables, an input field for Relational Algebra, and an input field for SQL/DDL.

8.6. 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 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.

8.7. 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.

8.8. Continuous Integration/Deployment

Our continuous integration will be completed through DroneCI. All tests will be run and passed before merging and again when integrating new features to the main branch

Requirements	**Type of Testing**	**Status**
Functional		
System will allow for relational algebra statements to be entered.	Unit Testing Integration Testing	Fail Fail
System will show visualizations of the resulting entered statement prior to submission.	Unit Testing UI Testing	Fail Fail
System will automatically mark the relational algebra questions once submitted.	Unit Testing	Fail
System will allow for DDL/SQL code to be entered. System will show resulting tables of queries prior to submission.	Unit Testing Integration Testing UI Testing	Fail Fail Fail
System will automatically mark the DDL/SQL questions once submitted.	Unit Testing	Fail
Student will be able to see the correct answer if the professor has allowed for the correct answer to be displayed after the question is submitted.	Unit Testing Integration Testing UI Testing	Fail Fail Fail
Professor will be able to set whether the correct answer will be displayed after the question is submitted.	Unit Testing UI Testing	Fail Fail

Professor will be able to see the correct answer.	UI Testing	Fail
Non-Functional		
The system will support all COSC 304 users simultaneously – about 200 students.	Performance Testing	Fail
The system will ensure data integrity and preservation so that no data is lost upon submission.	Performance Testing	Fail
The system will display entered queries within 3 seconds at scale and under optimal conditions.	Performance Testing	Fail
The system will return automarked submissions within 5 seconds at scale and under optimal conditions.	Performance Testing	Fail
The user interface will match existing software used for COSC 304.	UI Testing	Fail
Technical Requirements		
Rebuild RelaX editor and calculator into PrairieLearn	Unit Testing UI Testing Integration Testing	Fail Fail Fail
Frontend: JavaScript, HTML, CSS	UI Testing	Fail
Backend: Python, Node.JS	Unit Testing	Fail
Write JavaScript code that takes in SQL/DDL statements	Unit Testing Integration Testing	Fail Fail

and displays appropriate table results		
Write Python code that automatically marks submitted data and returns the students grade	Unit Testing	Fail