Automating Database Question Generation and Marking with PrairieLearn

Design Presentation

Nishant Srinivasan: Project Manager
Matthew Obirek: Client Liaison
Skyler Alderson: Technical Lead
Andrei Zipis: Integration Lead

Project Description

The project is to automate the delivery and evaluation of relational algebra and SQL questions with PrairieLearn.
Client: Dr. Ramon Lawrence
Purpose: Improve the way COSC 304 is taught

As of right now, students in COSC 304 are reliant on multiple different platforms for their coursework - including time-sensitive evaluations such as midterms and final examinations. Current tools include :

- Canvas
- PrairieLearn
- RelaX
- GitHub
- SQuirreL SQL DBMS

Our objective is to integrate

- Labs 1-3 from COSC 304 into PrairieLearn
- Necessary tools for these labs such as RelaX and an SQL Editor
- Feature for students to verify their answers on PrairieLearn without submitting
- Automate randomized question generation + evaluation

Requirements

Functional Requirements

- Improve documentation for deploying docker
- Allow for relational algebra and DDL/SQL code to be entered
- Display visualizations of entered relational algebra statements prior to submission
- Show resulting tables of SQL queries prior to submission.
- Automatically mark relational algebra and DDL/SQL questions once submitted
- Show correct answer (if professor allows for it) after submission
- Professor will be able to see correct answer

Non-Functional Requirements

- Support all COSC 304 users simultaneously - about 200 students
- Prevent data loss upon submission
- System will display entered queries within 3 seconds *
- System will return automarked submissions within 5 seconds *
- User interface will matching existing COSC 304 software
- Software will be maintainable - modular code, appropriately commented

* At scale under optimal conditions

User Groups and Personas - Students

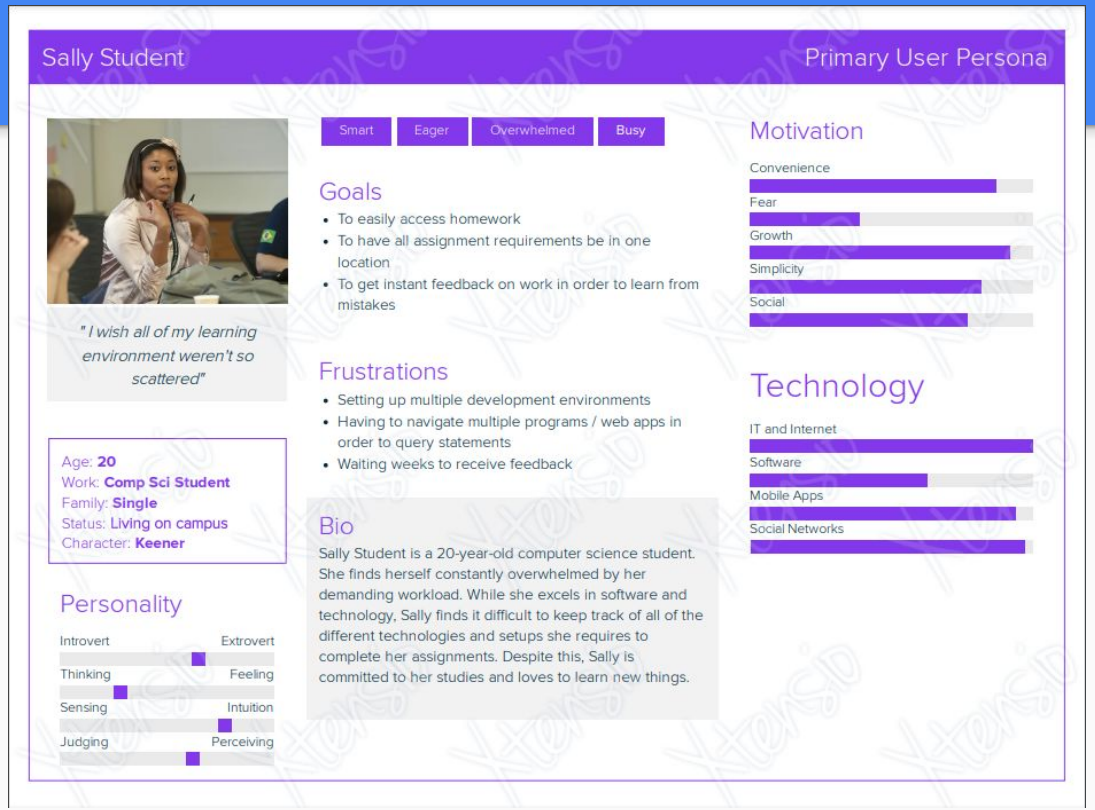
COSC 304 Students

Goals:

- Learn course material
- Be efficient with time

Issues:

- Setting up environments
- Integrating multiple programs
- Difficulties receiving feedback



User Groups and Personas - Professor

Dr. Ramon Lawrence
COSC 304 TA's

Goals:

- Streamline and simplify labs
- Provide instant feedback

Issues:

- Time spent helping students debug environments
- Learning is not provided at the pace of the student

PrairieLearn

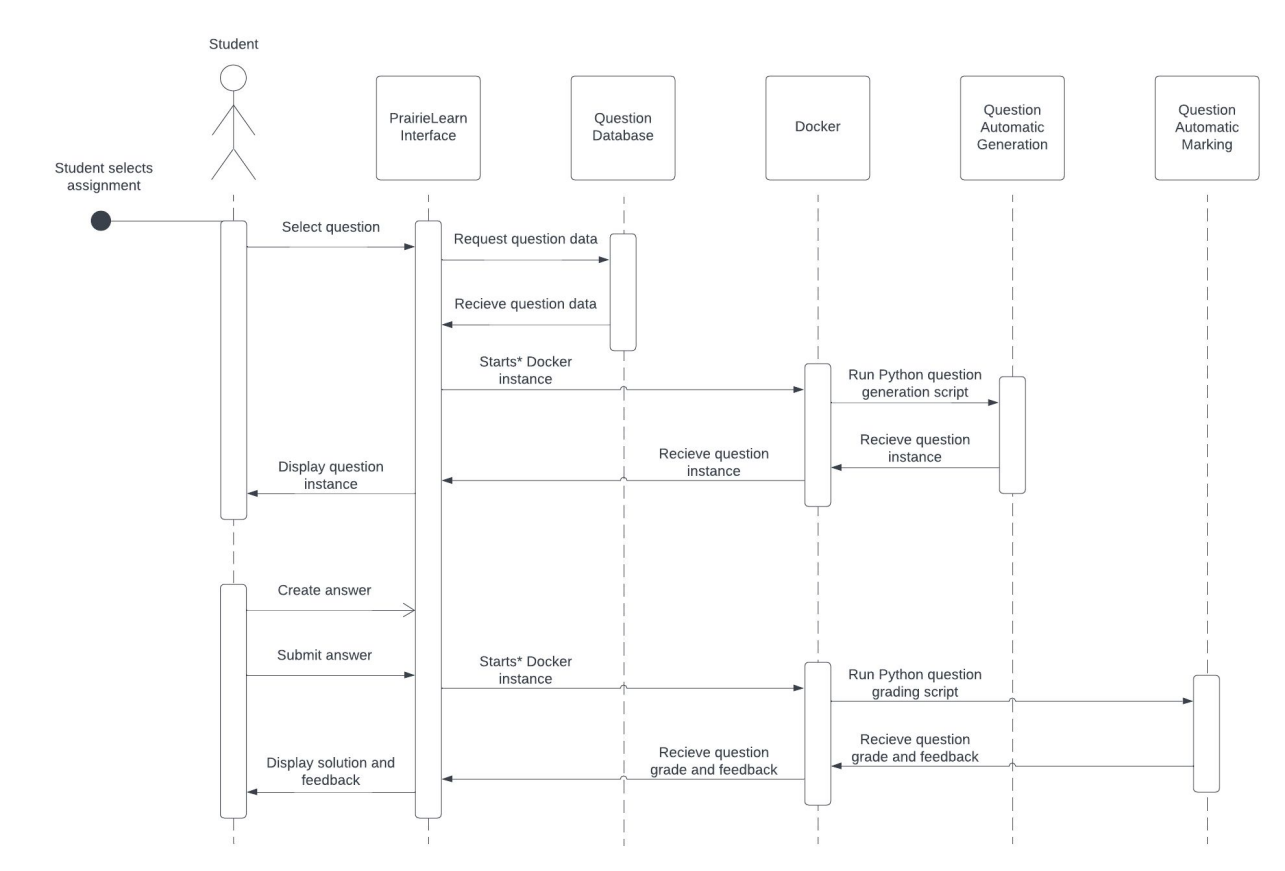


Figure 1: *After a student selects an assignment and a question, PrairieLearn retrieves the question's base data before using Docker to run Python to generate a question variant. After a student submits their answer, PrairieLearn uses a new Docker instance to run Python to grade the question and give feedback to the student.*

System Architecture - Level 0 DFD

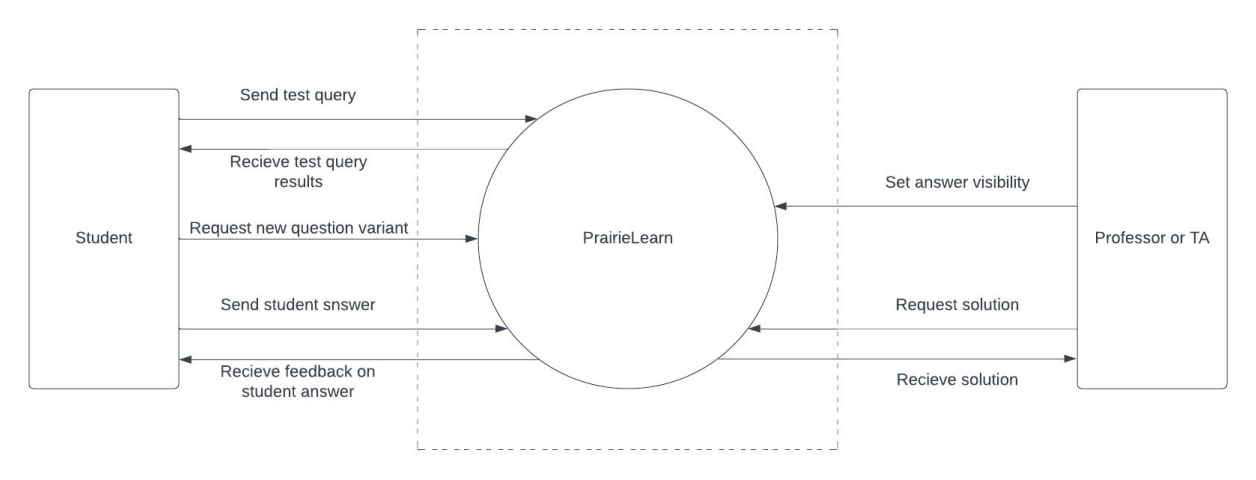
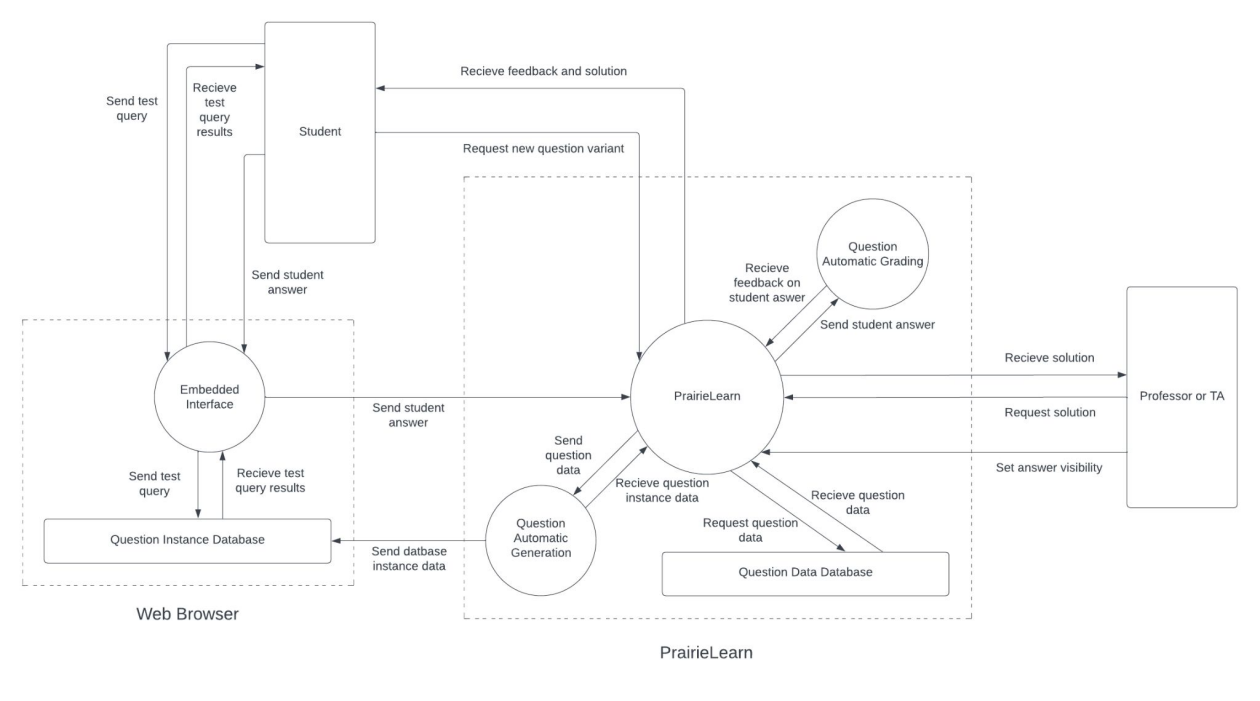


Figure 2: *The test query a student submits is sent to PrairieLearn, which returns the query's results. When a student submits their answer, PrairieLearn gives the student feedback. A student may request a new question variant. A professor or TA can show or hide the solution or set question visibility.*

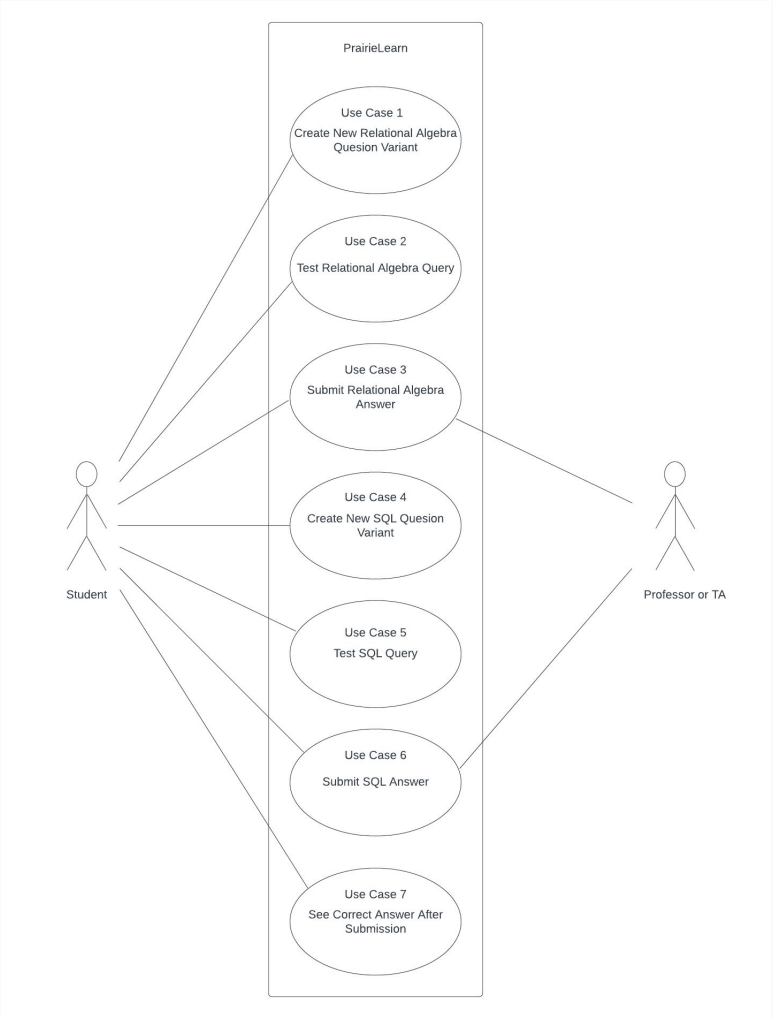
System Architecture - Level 1 DFD

Figure 3: *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.*



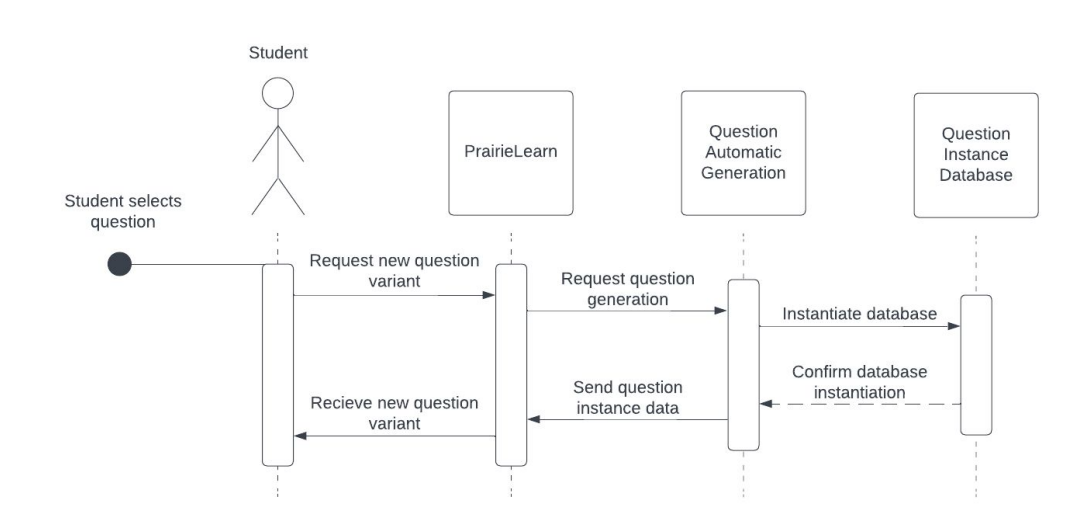
Use Cases: Student

Figure 4: *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.*



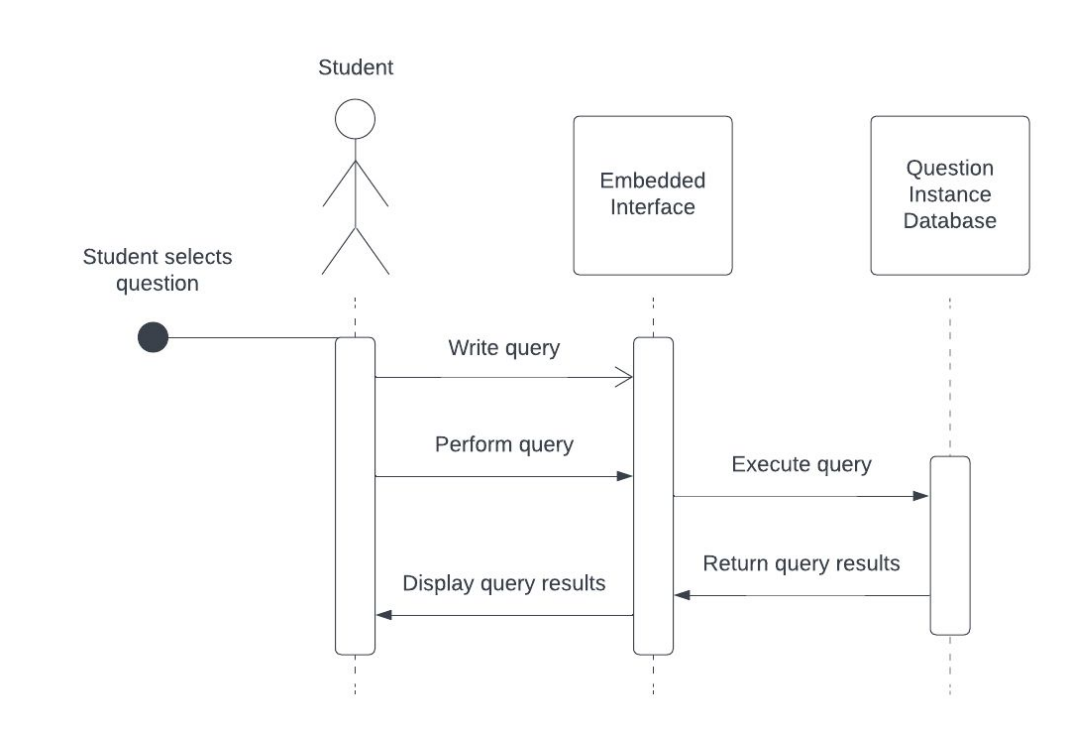
Use Cases 1 & 4

Figure 5: *After a student selects a question, they request a new question variant from the PrairieLearn system. PrairieLearn uses its automatic question generation to create a new question variant and in so doing instantiate a new database. The question instance data is returned to the student.*



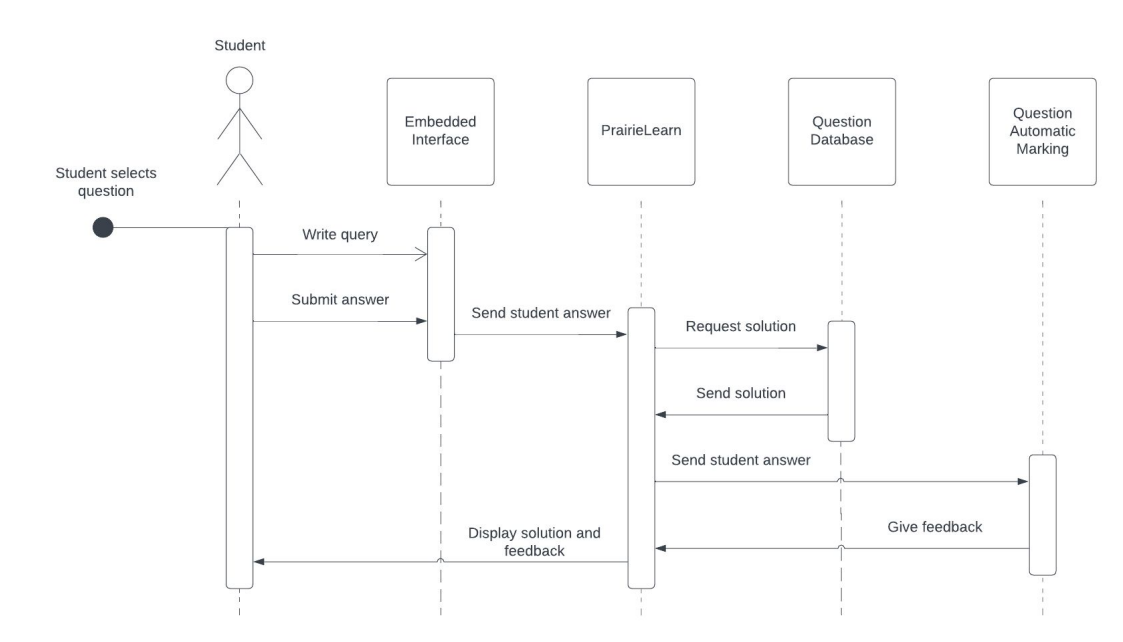
Use Cases 2 & 5

Figure 6: *After a student selects a question, they use the embedded editor to create a query. The student then tests the query, where PrairieLearn executes the query on a database instance. The results of the query are returned to the student.*



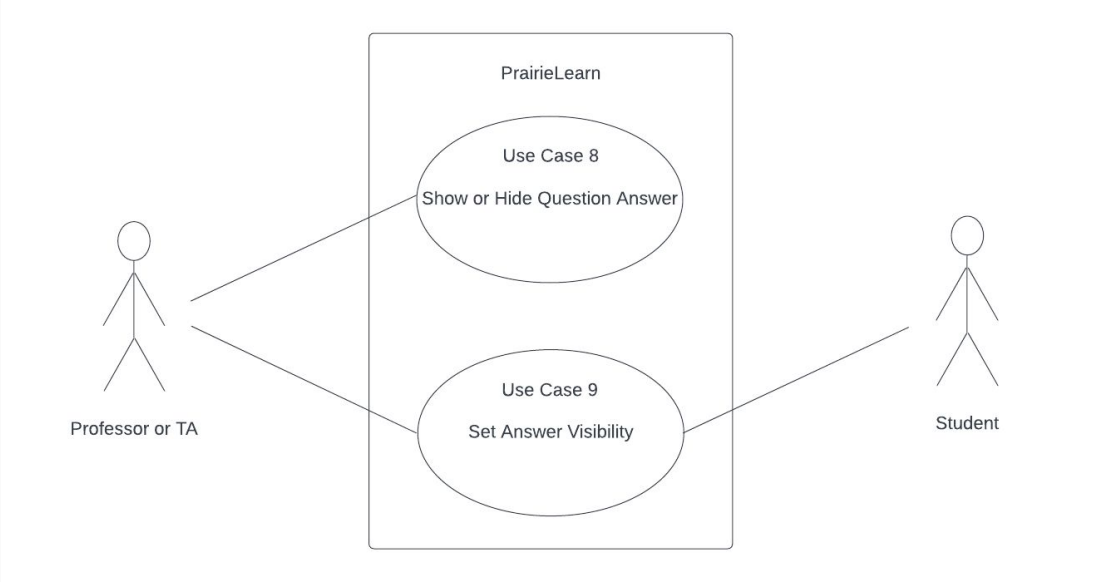
Use Cases 3, 6, & 7

Figure 7: *After a student selects a question, they use the embedded editor to create a query. The student then submits their answer, where PrairieLearn obtains the solution from the questions database and sends both the solution and the student's answer to the automatic grader. The feedback of the student's answer is returned to the student.*



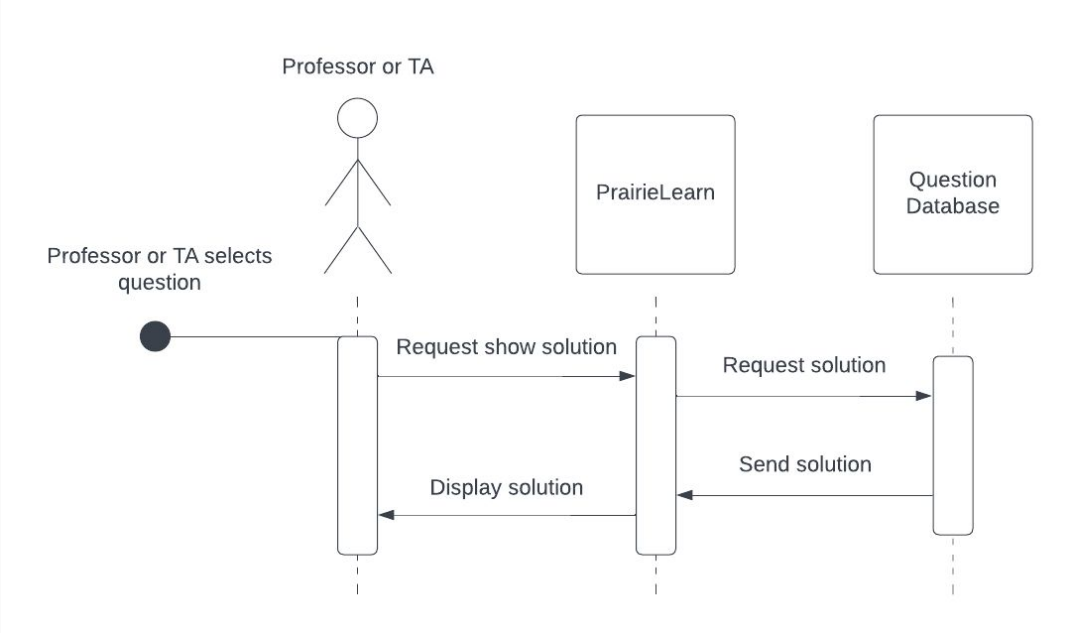
Use Cases: Professor

Figure 8: *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.*



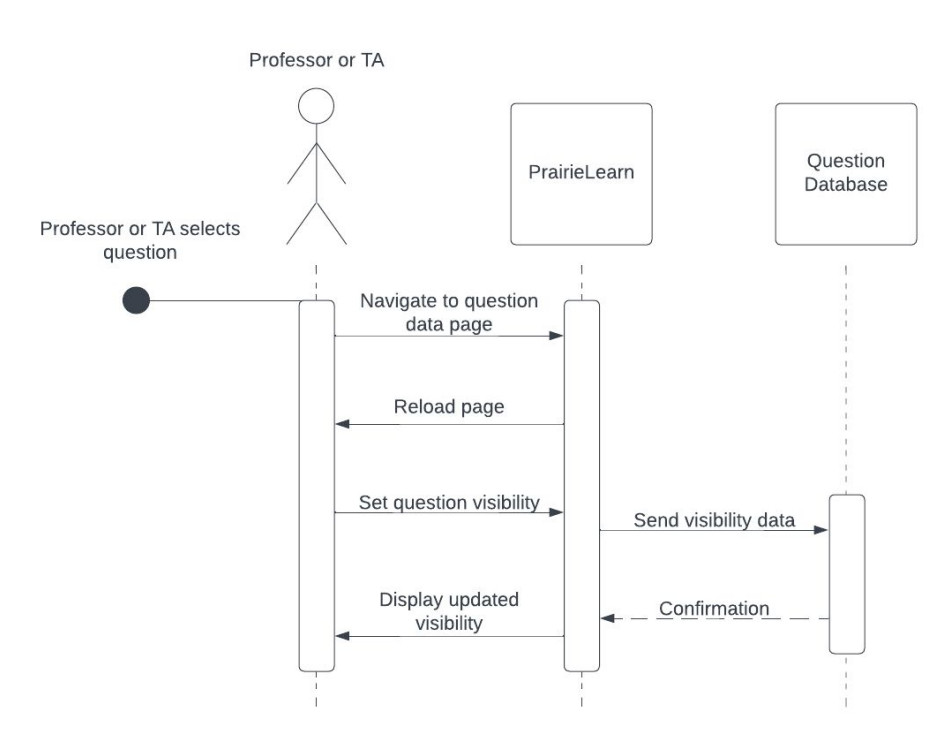
Use Case 8

Figure 9: *After a professor or a TA selects a question, they request to see the answer. PrairieLearn then obtains the solution from the questions database and shows it to the professor or TA.*



Use Case 9

Figure 10: *After a professor or a TA selects a question, they navigate to the files tab of PrairieLearn and selects the question data. The professor or TA then updates the solution's visibility, where PrairieLearn changes the visibility settings and displays the updated status.*



UI : RelaX

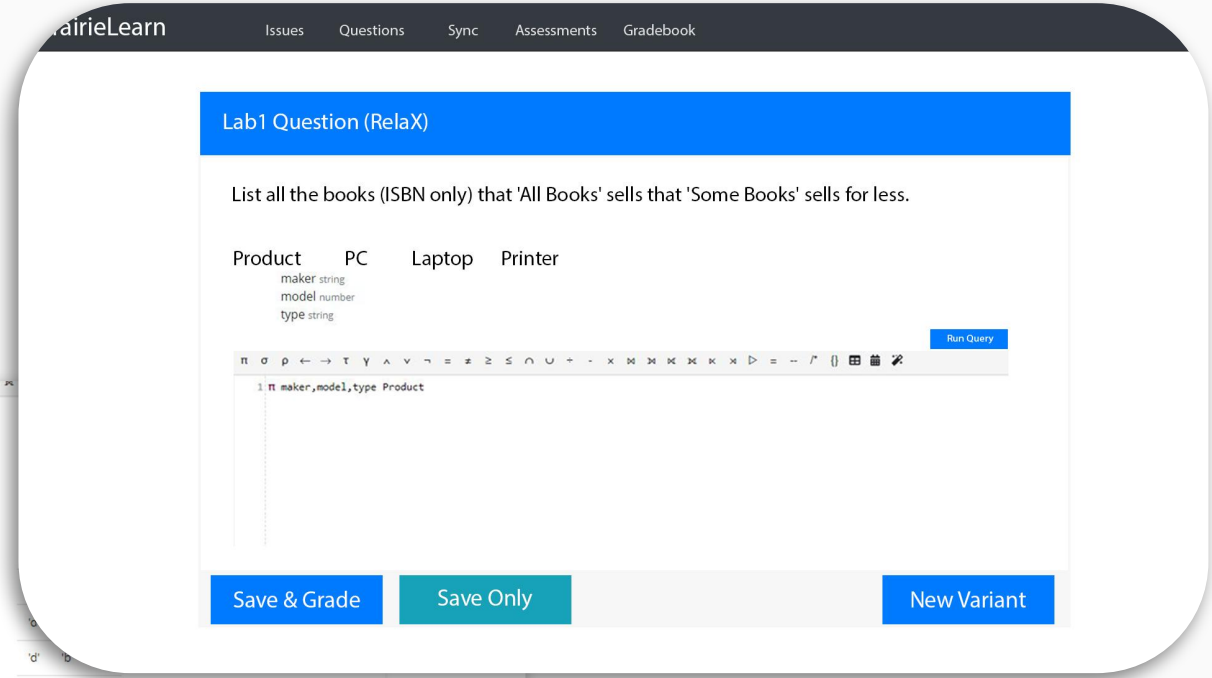


Figure 11: *A mock UI for the RelaX editor within PrairieLearn.*

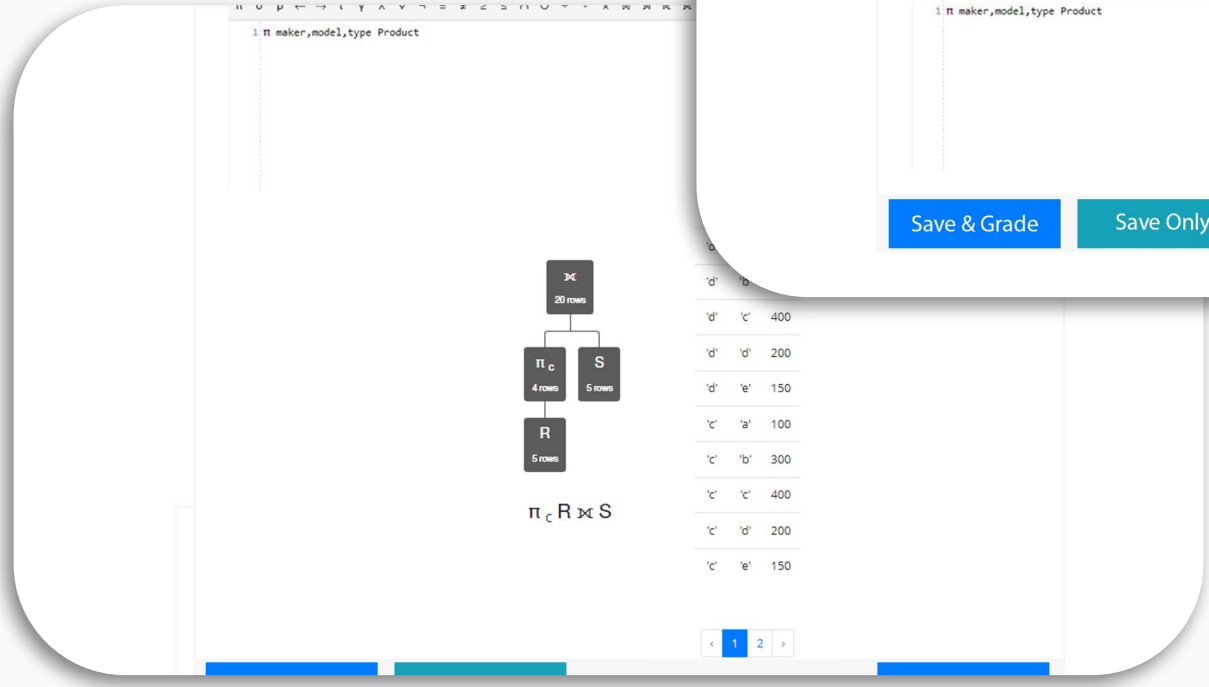


Figure 12: *A mock UI for the RelaX query results within PrairieLearn.*

UI: SQL/DDL

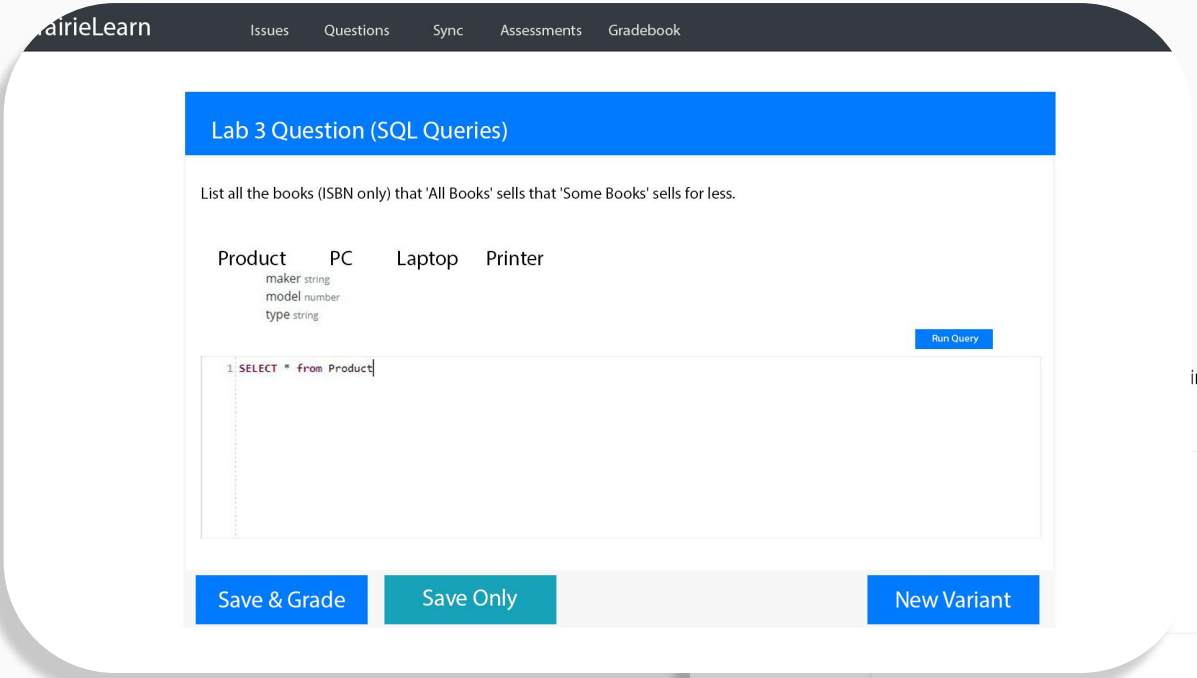
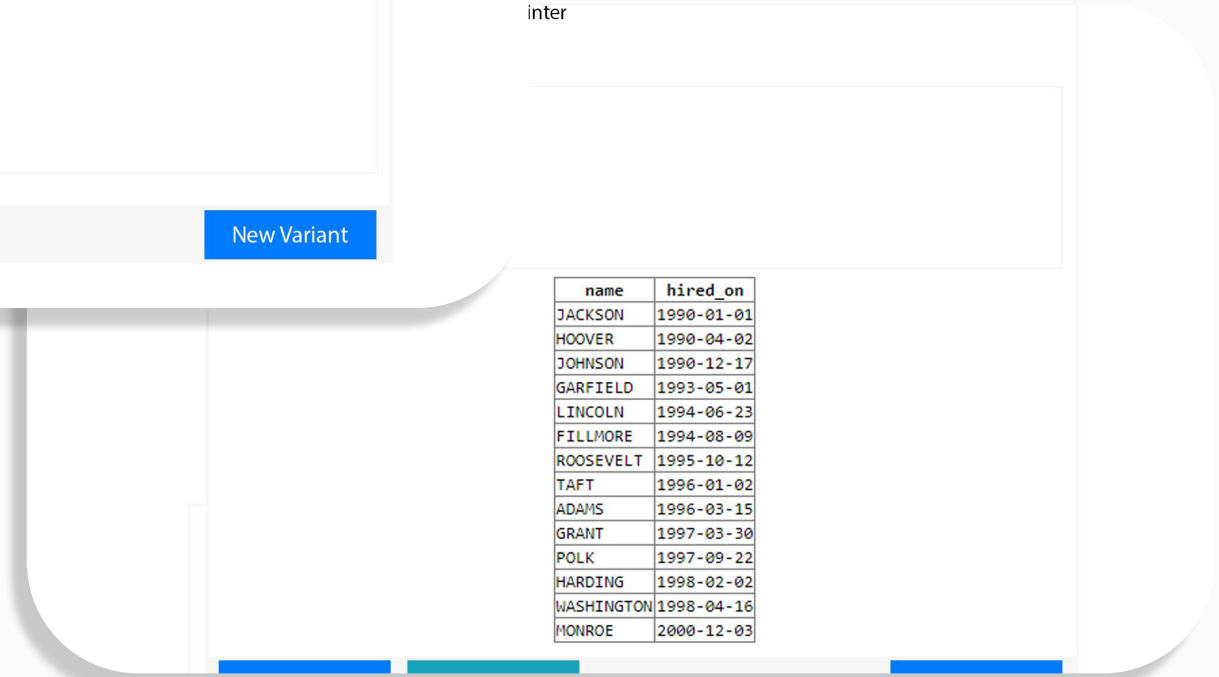


Figure 13: *A mock UI for the SQL editor within PrairieLearn.*



name	hired_on
JACKSON	1990-01-01
HOOVER	1990-04-02
JOHNSON	1990-12-17
GARFIELD	1993-05-01
LINCOLN	1994-06-23
FILLMORE	1994-08-09
ROOSEVELT	1995-10-12
TAFT	1996-01-02
ADAMS	1996-03-15
GRANT	1997-03-30
POLK	1997-09-22
HARDING	1998-02-02
WASHINGTON	1998-04-16
MONROE	2000-12-03

Figure 14: *A mock UI for the SQL query results within PrairieLearn.*

Frontend

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

Backend

Back-end development will continue to be in Python. PrairieLearn is built with Node.js but will not be directly used in our system's development.

Database

The data for the labs (questions and solutions) will be stored using PostgreSQL. This is also handled through the PrairieLearn system and will not be directly used in this system's development.

Tools

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

Testing

Python unittest framework
We will be using this for our Unit Tests, Integration Tests, and to automate our UI tests.

Selenium
To ensure that our UI components, both old and new are working as required, we will be using the Selenium WebDriver for our UI tests.

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

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.

We will be using unit testing, integration testing, performance testing, and UI testing 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.

Unit Testing

- White-box testing technique
- Function-by-function
- Coverage of cases
  - Question Generation for **RelaX**
  - Question Grading for **RelaX**
  - Question Generation for **SQL/DDL**
  - Question Grading for **SQL/DDL**

Performance Testing

- Essential to assess the system's ability to scale appropriately
- Stress tested for performance dips
- Determine bottlenecks and optimize resource allocation
  - Support all COSC 304 users simultaneously
  - Data integrity and preservation upon submission
  - Time to display entered queries
  - Time to return graded submissions

Integration Testing

- Black-box testing technique
- Integration and compatibility of multiple components and subsystems
  - Query Preview/Visualizations for **RelaX** Questions
  - Integration of **RelaX** Editor in PL
  - Query Preview/Visualizations for **SQL** Questions
  - Integration of **SQL** Editor in PL
  - Autograder + Auto Generator

UI Testing

- Ensures that various components of the software's UI, both new and old, are working as required
  - **RelaX** Questions
  - **RelaX** Query Visualizations
  - **RelaX** Editor
  - **SQL** Questions
  - **SQL** Query Visualizations
  - **SQL** Editor

That’s All Folks!

Questions?