Building an Auto Grading System Using PrairieLearn

Luis Lucio
Emiel van der Poel
Prajeet Didden
Siqiao Yuan

What Is Our Project?

- Amend PrairieLearn, which is an open source software, to support a variety of questions regarding UML, SQL, and programming
- Migrate the existing system onto PrairieLearn for better scalability.
- Change UML diagram rendering library for cleaner diagrams.

Our Client- Dr. Ramon Lawrence

The problem

The existing system for generating questions and auto-grading is not scalable as it only supports the UML question type, with new questions types being difficult to create.

The solution

Implement an autograding system on an already existing learning platform; PrairieLearn

Requirements

Functional Requirements

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

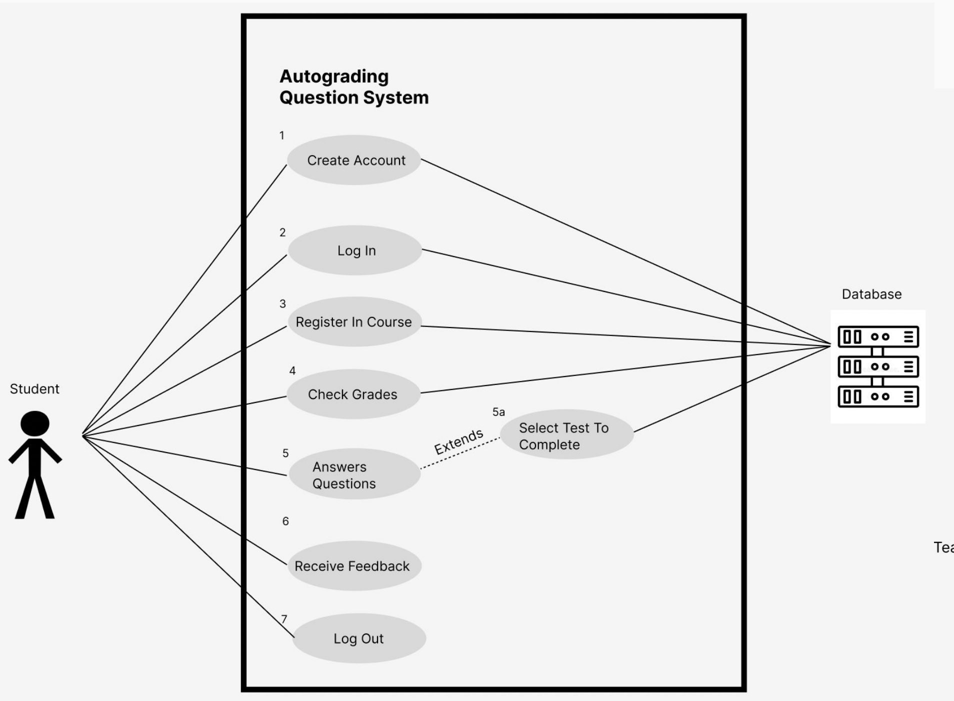
Non-Functional Requirements

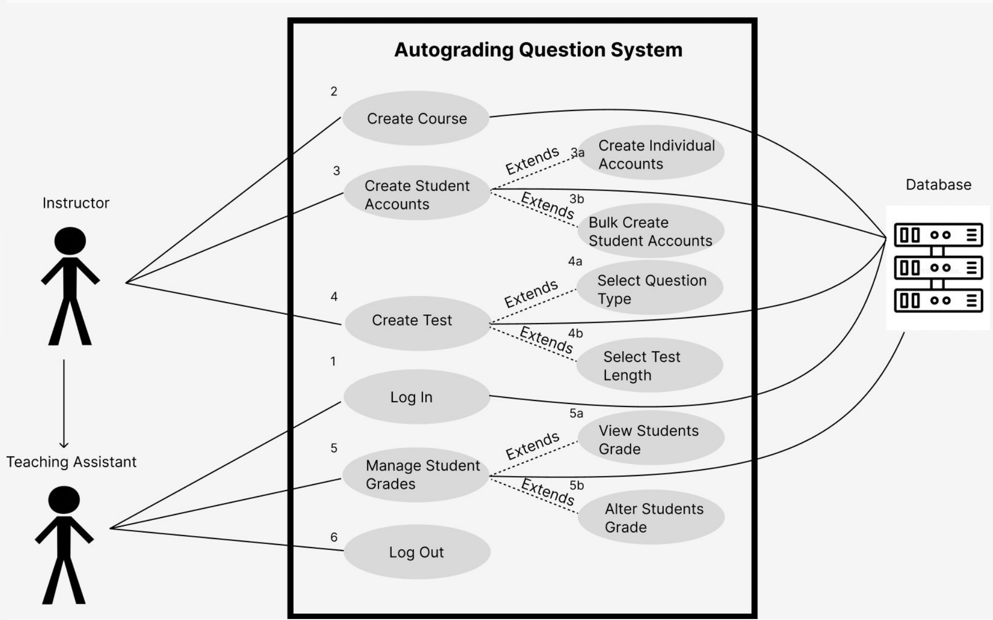
- Deploy dockerized PrairieLearn.
- Highly accurate grading system that will correctly follow set grading scheme.
- Front-end diagram renders maintain high readability standards set by the client.
- Compatible with Mermaid-js or nomnoml.
- Use Python as the primary language for the back-end.
- Extensibility and modifiability for future updates.
- Supporting capability.

User Groups

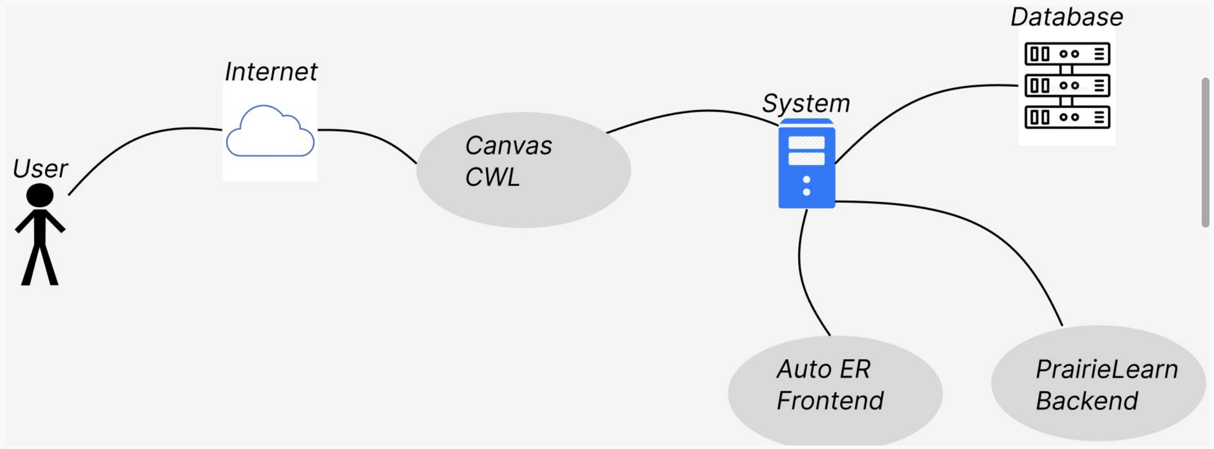
- Students
- Admin

- Instructors
  - Professors
  - Teaching Assistants



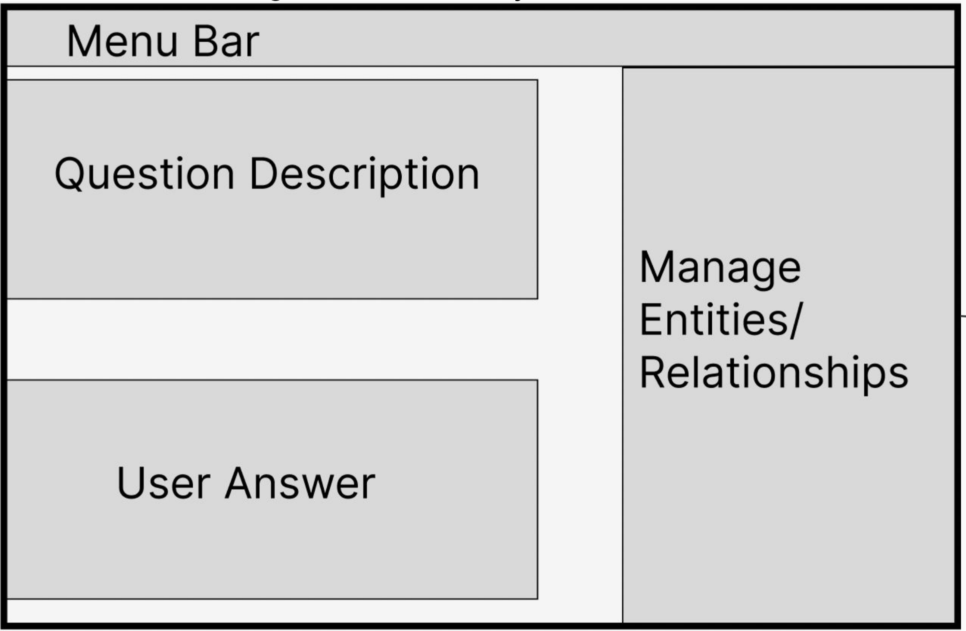


System Architecture

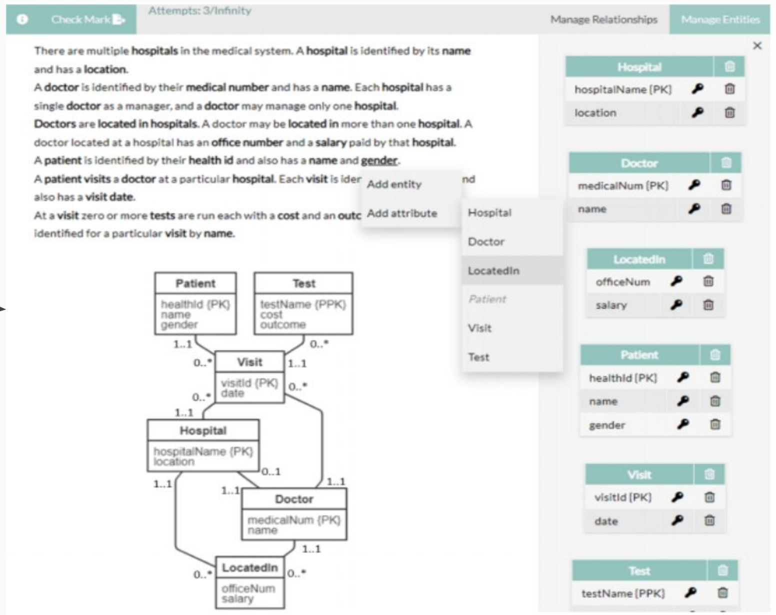


UI Mockups

Rough Question Layout



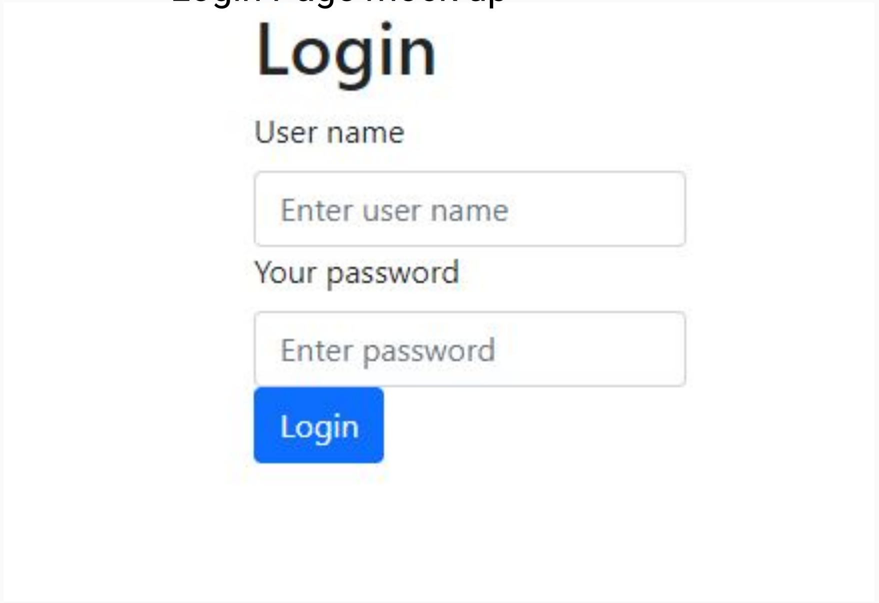
Previous Frontend layout



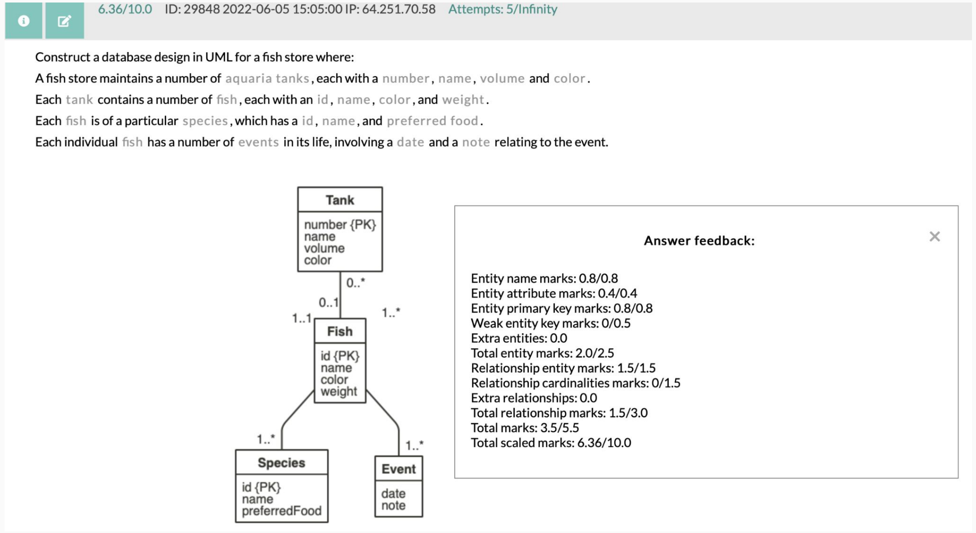
There are multiple **hospitals** in the medical system. A **hospital** is identified by its **name** and has a **location**.
A **doctor** is identified by their **medical number** and has a **name**. Each **hospital** has a single **doctor** as a manager, and a **doctor** may manage only one **hospital**.
Doctors are **located in hospitals**. A doctor may be **located in** more than one **hospital**. A doctor located at a hospital has an **office number** and a **salary** paid by that **hospital**.
A **patient** is identified by their **health id** and also has a **name** and **gender**.
A **patient visits** a **doctor** at a particular **hospital**. Each **visit** is ider nd also has a **visit date**.
At a **visit** zero or more **tests** are run each with a **cost** and an **outc** identified for a particular **visit** by **name**.

UI Mockups pt. 2

Login Page mock up



Receiving feedback mock up



6.36/10.0 ID: 29848 2022-06-05 15:05:00 IP: 64.251.70.58 Attempts: 5/Infinity

Construct a database design in UML for a fish store where:

A fish store maintains a number of aquaria tanks, each with a number, name, volume and color.

Each tank contains a number of fish, each with an id, name, color, and weight.

Each fish is of a particular species, which has a id, name, and preferred food.

Each individual fish has a number of events in its life, involving a date and a note relating to the event.

Answer feedback:

Entity name marks: 0.8/0.8
Entity attribute marks: 0.4/0.4
Entity primary key marks: 0.8/0.8
Weak entity key marks: 0/0.5
Extra entities: 0.0
Total entity marks: 2.0/2.5
Relationship entity marks: 1.5/1.5
Relationship cardinalities marks: 0/1.5
Extra relationships: 0.0
Total relationship marks: 1.5/3.0
Total marks: 3.5/5.5
Total scaled marks: 6.36/10.0

Test Plan

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 we have written. This will allow us to see which lines of code are being used which in turn can help optimise code.
- Will check statement, decision, branch, and condition coverage.

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.

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 when a user clicks the grade question box and a response is received from the server.

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.

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.

Functionality Testing

- The functionality of the system is based on whether or not the system can accurately grade UML 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).