

Assignment 2: Design Document

Project 3: Building an Autograding Question System using PrairieLearn

University of British Columbia Okanagan

COSC 499 - Summer 2022

Date: June 2nd, 2022

Contacts:

Emiel van der Poel (emielvdpobel@gmail.com)

Luis Lucio (llucio99@icloud.com)

Prajeet Didden (prajeetdidden@gmail.com)

Siqiao Yuan (siqiaoyuan@gmail.com)

1. Project Description

2. User Groups

2.1 User Groups

2.2 User Persona Examples

3. Use Case Diagrams

3.1 Student User Group Use Case

3.2 Instructor User Group Use Case

3.3 Admin User Group Use Case

4. System Architecture

4.1 System Architecture

4.2 Data Flow Diagram

5. UI Mockups

5.1 Log In Mockup

5.2 Rough Mockup For Answering Questions

5.3 Previous Frontend UI For Answering Questions

5.4 Previous Frontend UI For Question Feedback.

6. Tools To Build Software

7. Test Plan

7.1 Unit Testing

7.2 Regression Testing

7.3 Integration Testing

7.4 Component Testing

7.5 User Testing

7.6 Functionality Testing

8. References

1. Project Description

The purpose of this project is to modify an existing auto grading system whose purpose is to create an easy learning environment for students. The current system uses AutoEd as the backend to autograde questions which works well for UML questions but is built too specifically. With this project the goal is to shift the backend to an open source software called PrairieLearn, which utilises python scripts to autograde questions, allowing instructors to create any type of question and have it autograded.

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 randomised 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.

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.

2. User Groups

2.1 User Groups

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

2.2 User Persona Examples

Student:

1. Name: John Smith

Age: 21

Profession: 3rd year Computer Science Student.

Personal Interest: Various sports

Platform: Uses a 2020 macbook to do class assignments.

Target: Be successful in school

Bio: John has a Minor OCD, he likes things to be organised and neat.

Instructor:

2. Name: Jane Doe

Age: 46

Profession: Computer Professor at UBCO

Personal Interest: Playing guitar

Platform: Owns a Windows platform desktop at home and at work

Target: Helps her student to have better understanding of what she taught in class

Bio: Jane is a mother to three kids, doesn't have a lot of time after work.

3. Use Case Diagrams

3.1 Student User Group Use Case

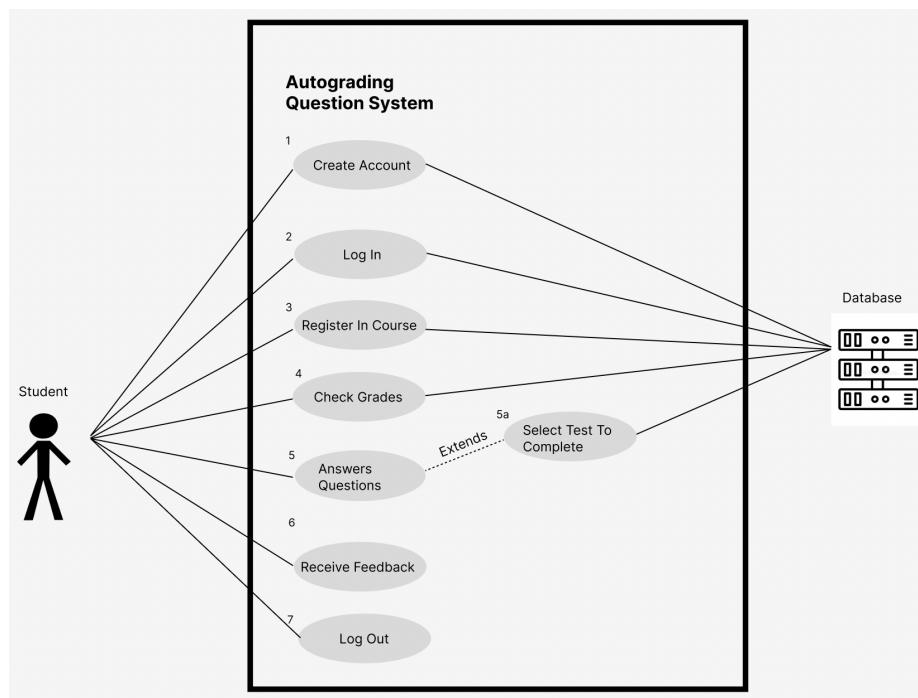


Figure 1

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 answered 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 internet 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.

3.2 Instructor User Group Use Case

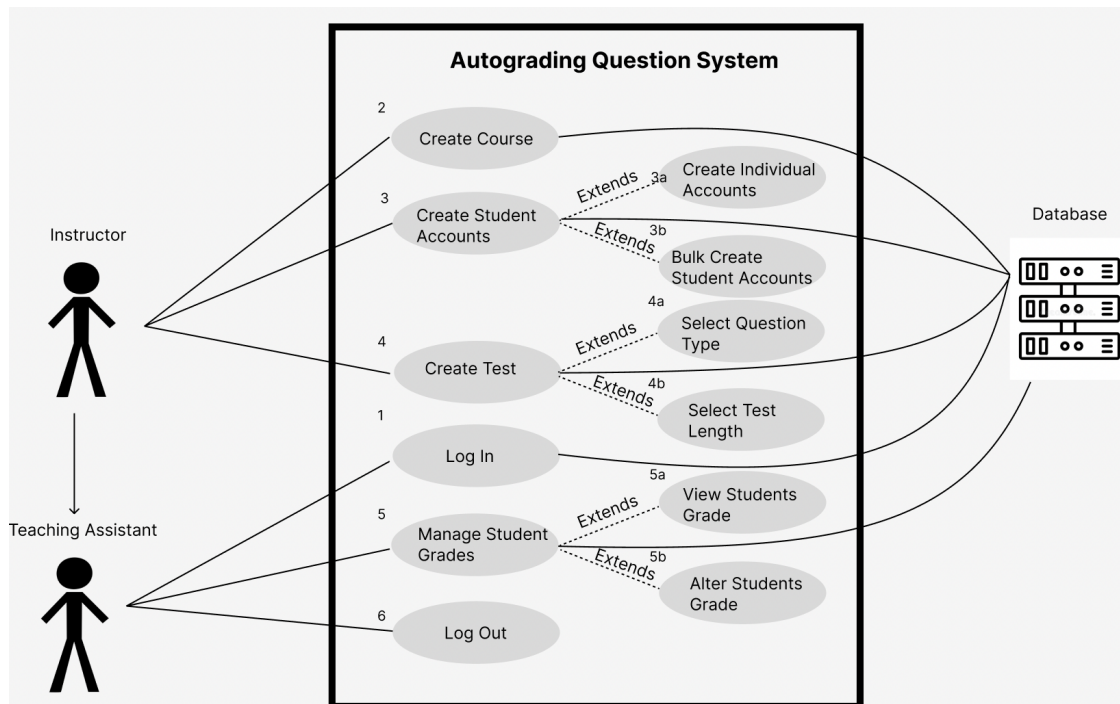


Figure 2

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 an 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 internet 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.

3.3 Admin User Group Case

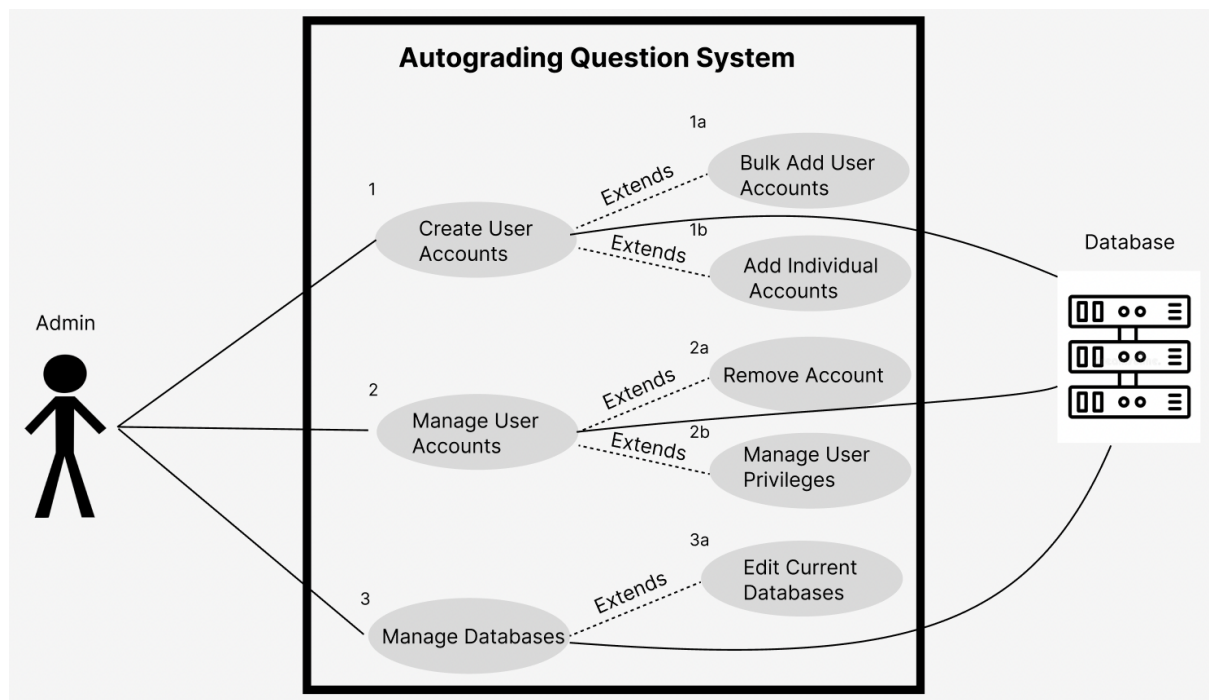


Figure 3

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.

4. System Architecture

4.1 System Architecture Diagram

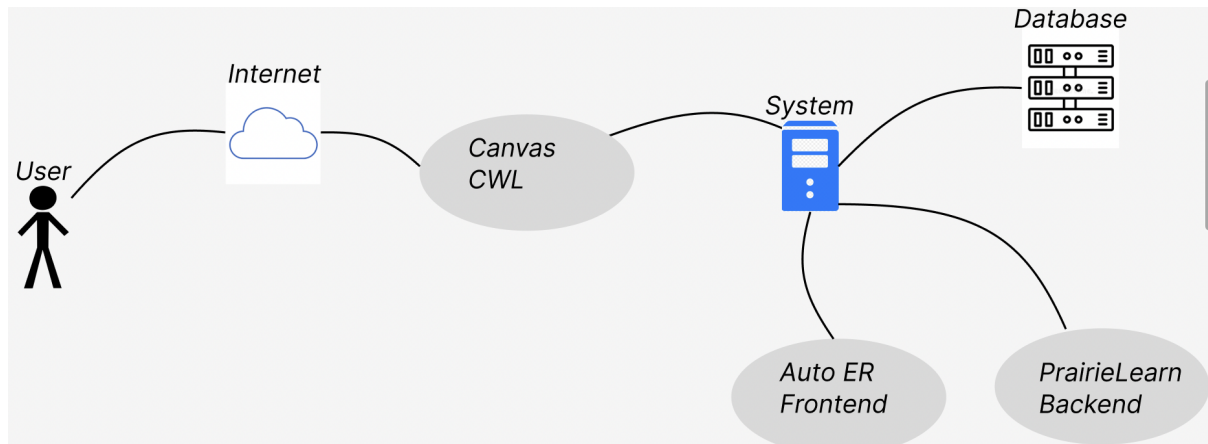


Figure 4: Here we have a high level architecture diagram. The user will connect through the internet to CWL. Through CWL they will be able to log into the system. The system will be using the AutoEr frontend and will be using PrairieLearn as a backend. The system is also connected to a postgres database for system storage.

4.2 Data Flow Diagram

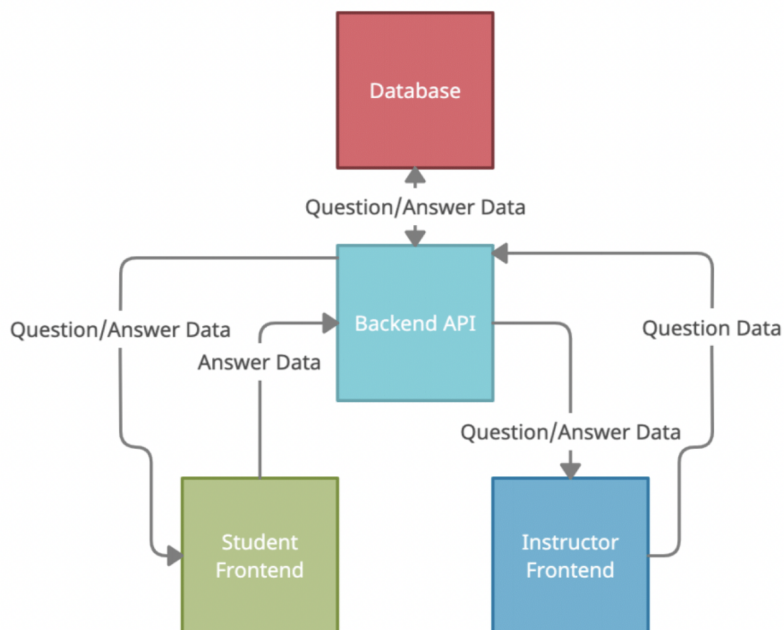
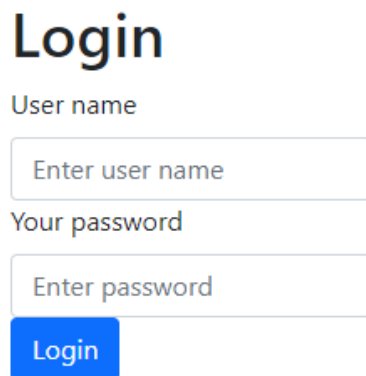


Figure 5: The instructor frontend will allow users with instructor privileges to enter question data and send that information to the backend API. The student frontend will give users questions generated by the instructor, which they will be able to answer through the use of the frontend interface in order to also send that data to the backend API. The backend API 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. This API will be responsible for both communicating with the database as well as sending/receiving all data to corresponding frontend interfaces. Reference: Foss, S., Urazova, T., & Lawrence, R. (n.d.). Automatic Generation and Marking of UML Database Design Diagrams.

5. UI Mockups

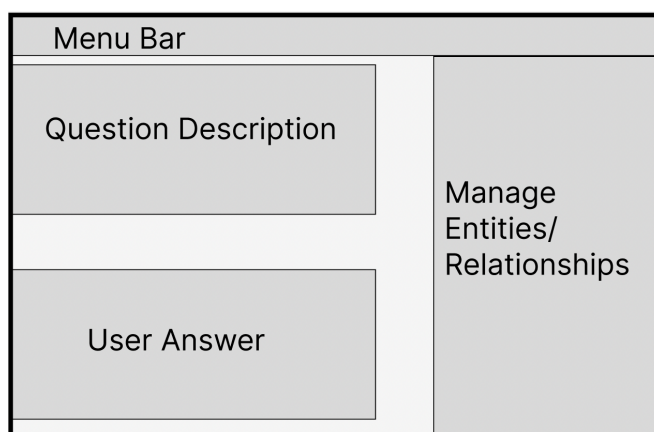
5.1 Log In Mockup



A login form mockup with the title "Login" in a large, bold, black font. Below the title, there are two input fields. The first is labeled "User name" and contains the placeholder text "Enter user name". The second is labeled "Your password" and contains the placeholder text "Enter password". Below these fields is a blue button with the text "Login" in white.

Figure 6: An example mockup of how our system will handle logins if the user is not using CWL login.

5.2 Rough Mockup For Answering Questions



A rough mockup for answering questions. It features a "Menu Bar" at the top. Below the menu bar, there are two main sections. On the left, there is a "Question Description" section and a "User Answer" section. On the right, there is a "Manage Entities/ Relationships" section.

Figure 7: This figure is a rough draft of how our question ui will be handled. This rough draft comes from the previous UI frontend in figure 8.

5.3 Previous Frontend UI For Answering Questions

Check Mark
Attempts: 3/Infinity
Manage Relationships
Manage Entities

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 identified for a particular **visit** by **date**.

At a **visit** zero or more **tests** are run each with a **cost** and an **outcome** identified for a particular **visit** by **name**.

Add entity
Add attribute

Hospital
Doctor
Locatedin
Patient
Visit
Test

Hospital
hospitalName [PK]
location

Doctor
medicalNum [PK]
name

Locatedin
officeNum
salary

Patient
healthId [PK]
name
gender

Visit
visitId [PK]
date

Test
testName [PPK]
cost
outcome

Figure 8: This figure showcases the way the student can interact with the question's interface in order to generate a UML diagram. The randomly generated question will have several keywords written in bold, which the student can click in order to add that word to the diagram either as a new entity, or an attribute of an already existing entity. The options shown in the top right corner will allow students to manage already existing entities by selecting certain attributes to be primary keys, or delete them in case they made a mistake. The Manage Relationships menu allows students to link entities to each other and form a relationship between them, where they will also be allowed to specify the cardinality of the relationship. After the students finish the question, they can click Check Mark in order to receive a grade on their submission. Reference: Foss, S., Urazova, T., & Lawrence, R. (n.d.). Automatic Generation and Marking of UML Database Design Diagrams.

5.4 Previous Frontend UI For Question Feedback

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

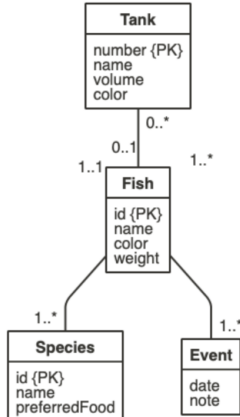
Construct a database design in UML for a fish store where:

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

Figure 9: This figure showcases the grading and feedback portion of the interface. After attempting the question, the students can Check Mark in order to receive marks for their answer and check their results. After confirming the marks awarded, students will also be able to look at the specific marking scheme by clicking the green highlighted grade and receive feedback in order to know exactly where they got their marks from and where they lost marks in the case they made a mistake.

6. Tools To Build Software

1. Frontend IDE - WebStorm.
2. Backend IDE - PyCharm.
3. Time Tracking - Toggl.
4. Task Tracking - Github Projects.
5. 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 optimise 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.

7.6 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).

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].