Handover Documentation

Project 3: Building an Autograding Question System using PrairieLearn

University of British Columbia Okanagan

COSC 499 - Summer 2022

Date: August 17th, 2022

Contacts:

Emiel van der Poel (emielvdpoel@gmail.com)

Luis Lucio (llucio99@icloud.com)

Prajeet Didden (prajeetdidden@gmail.com)

Siqiao Yuan (siqiaoyuan@gmail.com)

1. Overview

1.1 Workflow

This project was created in an Agile environment using Github Projects as our Kanban board.

- Weekly Scrum meetings were held
- PR's were reviewed by the majority of the team.
- Weekly Client Meetings

Development began by breaking down the tasks into smaller tickets. Once tickets were broken down we would assign them based on expertise and availability. Once a ticket was assigned a branch off the main was created for that ticket and all work related to that ticket would be completed in the branch. Upon completion of the ticket a PR would be made to merge it into the master. Once this PR is created tests will be run on the branch and it requires an approval for the pull request to be merged in. Generally all team members would review the PR.

Tools used:

- Python (Dev Language) (version 3)
- JavaScript (Dev Language)
- Pycharm (IDE) (2022.1.1)
- Github Projects
- Docker (4.10.X)
- DroneIO
- Digital Ocean
- Linux Server
- Toggl (Time Tracking)

1.2 Testing

Various test suites were created to specifically test our new ER element. These test suites were written in Python using Pytest and test both the random generation and random grader modules. More information about our testing suite can be found on the PrairieLearn GitHub repository under the UML Python Testing markdown file. Instructions on how to actually run the tests can be found through a link in the markdown file mentioned above, or for simplicity: here (*Running the test suite*).

These tests were integrated into our CI/CD pipeline (DroneIO) and are automatically run everytime we want to merge something to the master branch.

These tests only cover a fraction of PrairieLearn's total functionality, including actually testing the buttons in our element as all of that is already tested and accounted for in PrairieLearn's already existing testing suite. They have over 8,600 tests that cover everything from database manipulation to website navigation. The majority of these tests are written in Javascript using Mocha and Chai as a testing framework, and instructions on how to run these Javascript tests can also be found in the Installing Local markdown file.

1.3 Deployment

Deployment for this project has been set up on an UBCO server (prairielearn.ok.ubc.ca). To redeploy or set up a new deployment instance, follow the instructions written by the team in PrairieLearn's documentation under: running in production. All the documentation can also be found in the **docs** folder in the root of the repository.

2. Where to Find Code

2.1 Dependencies

Docker must be installed to get the project running. The code for running in a production environment is available by forking the PrairieLearn Repository.

2.2 Deployment

Follow instructions in PrairieLearn's documentation for running in production.

Element code has been handed over to the client, with a course setup in the UBCO PrairieLearn courses organisation, in the pl-ubco-cosc304 repository.

Code pertaining directly to the capstone project for development can be found here. This code is strictly for development of the element and is not the recommended way for deployment.

3. User Documentation

Can be accessed via Google Drive: User Documentation.

PrairieLearn documentation can be found here: PrairieLearn Documentation.

4. Testing Not Implemented

- JavaScript integration testing as most of the code was provided and with nested functions dominating the code made it difficult and out of scope for our project as we did not create the code.
- UI testing as the UI is handled via PrairieLearn. Client acceptance testing was used for the UI.

5. Feature Not Implemented

Features that were originally intended for implementation that did not get fulfilled:

- CWL Integration. (as this was difficult to integrate due to IT restrictions on the UBC domain)
- Prototyping importing the Relax Editor for relational algebra questions (This was attempted but due to time restrictions we were unable to complete this task. Detailed description of what to do and what has been done has been given to client)

5.1 RelaX Relational Algebra Editor

Various approaches were taken to prototyping this editor into a question type inside PrairieLearn. Approaches that were explored were: bringing in the website's functionally through a javascript bundle and an entry point into the program, a REST API specifically made for this editor (relax-api), and using a repository to turn RelaX into a static site (react-static)

Running RelaX locally through their original repository worked, which might allow debugging the javascript bundle individually as PrairieLearn does not offer many debugging options.

The approaches mentioned above failed due to React not merging well with NodeJS (NodeJS is what PrairieLearn uses throughout its application)

The approaches recommended for future attempts at prototyping and implementing RelaX into PrairieLearn are to rebuild the editor and calculator from scratch, which can be aided by referencing the open-source code; this is the most promising approach, as even if the other approaches were expanded upon and a prototype become functional, the biggest problem with these other approaches is that these would bring the entire program of RelaX into PrairieLearn, not only the calculator (it would bring a landing page, alongside other pages and all their included navigational buttons).