Coding Assignment Demo System

David Cooper

Introduction

Programming assignments are very challenging for students who are new to software development. Often, these programs can have concepts which are hard for students to understand. This can lead to confusion in a number of ways. First, students may not understand what the program they are trying to build is intended to do. Second, they may understand what it is trying to do but be unable to verify the results because the logic is not something which can be traced in any reasonable way. Dealing with these issues is not something that is necessarily a core competency that instructors are trying to teach to students. While there exist some solutions to these problems, such as autograders verifying accuracy of results, not all problems can be autograded and difficulties understanding the problem could still persist. As such, a system for demoing assignments to students will be developed.

This project seeks to have a system which allows students in a programming course to see or interact with completed programming assignments. A very simple example would be a factorial program. In this case, there would be interactable elements which would allow students to try different inputs and see the output. This would allow students to engage better with the material, understand the assignments better, and build confidence when their code is able to produce the same results as the demo.

Non-Functional Requirements

For this system, the following non-functional requirements must be met:

- The system can work on-line and off-line. This system is meant to be used by instructors and students alike. For instructors, there is no need to have an online version if they don't want students to interact with the system, so an offline version should be available. Additionally, if the hosting website is down for whatever reason, an offline version should be available as backup.
- The system should be able to meet requirements for translatability. Many international students will be more comfortable with their native language. As such, the system should be able to be translated manually or by machine without issue.
- The system should be lightweight if possible. Unless demoing an assignment which is highly resource intensive, the system should use minimal network and host resources. It should use no more than 100 MB of RAM and 25% of CPU.
- The code must run within Google Chrome, Firefox, and Edge. Different students use different browsers, so they should all be supported.
- The system should have documentation which supports the instructors, students, and IT personnel who use and maintain it.

Functional Requirements

The first and most important requirement is that students are able to see the results of code being run. For instance, if the assignment is to create a factorial program, there should be some way to input a number and see its factorial.

The second requirement is that students must be able to access and interact with the system. This will allow students to better engage with the assignment as well as be able to try text cases and make sure that their code matches the expected result.

The third requirement is that whether or not students have access to the interactive element is configurable. Instructors may wish to use it for in class demoing but do not want to give students access as they feel it may negatively impact learning. As such, being able to restrict student access is important.

The fourth requirement is that instructors must be able to manage visibility of assignments. Many assignments are built once and then used for many sections of a course over time. This means that the system could have all the assignment demos in the system at the beginning of a term. This is not ideal, so making the ones which are not yet published be hidden or unavailable is necessary.

The final requirement is that the system can integrate with the student's LMS, in this case Canvas. Because students already use a LMS, it makes the most sense to have any new system integrate well with it rather than have students use new technology which is unnecessary. Additionally, if configured to do so, students will be able to have the demo be integrated into the assignments page on Canvas for easy access.

Use Cases

Case 1 - Adding new assignment

Preconditions - Instructor must have admin access for the course and have written a solution for an assignment.
Main Flow - Instructor will access the configuration page and upload the corresponding files for the new assignment.
Subflows - There are no subflows for this use case.
This was not motivated by elicitation. This tends to be the sort of managerial task which is not thought of much, but still is a requirement. Also, the task was not specifically asked about since it is a relatively uncommon workflow.

Case 2 - In class demos

Preconditions - Instructor is presenting and has access to an assignment.

Main Flow - Instructor will open the assignment [S1], enter in some example inputs [S2], and the results will be displayed for students to see [S3].
Subflows -

1. From the main page, the instructor will click on the link for the corresponding assignment.
2. Exact implementation will depend on the assignment, but usually this will involve some sort of textbox input.
3. Exact implementation will also depend here, but there should be a text or virtual representation of the output produced.

Within the elicitation, it was mentioned that there should be an accurate representation of the assignment. This will help students to understand what is going on.

Case 3 - Student understanding assignment

Preconditions - Student has access to an assignment
Main flow - Student will open the assignment [S1], enter in some inputs, and see the results [S2].
Subflows -

1. From the main page, the student will click on the corresponding assignment.
2. As before, the interaction depends on the implementation. However, for the student case, the inputs are more random and the process will be repeated until the student is satisfied. If possible, the results should be somewhat annotated.

This case is motivated by elicitation responses which mention students being able to try inputs and students receiving explanations.

Case 4 - Student checking work

Preconditions - Student has access to an assignment, has completed a section of their assignment, and has some test cases they have run on their implementation.
Main flow - Student will open the assignment, enter in one of the inputs for their test cases [S1], and check that the results match what they got [S2].

1. Student will add input which matches one of their test cases.
2. Student will compare the results they got to the results from running the test case on their own code.

This case is motivated by elicitation results which indicate that it would be valuable if students were able to check their work against the instructor's code.

Case 5 - Nefarious Student

Preconditions - A student is struggling with the assignment and has access to the demo.
Main Flow - The student accesses the demo for an assignment [S1], then tries to access the source code [S2].
Subflows -

1. Student clicks on the link for the assignment from the system's homepage.
2. Student uses tools such as the developer console in Chrome to access the source code.

Alternative Flows -

1. Student fails to access source code.

This comes from elicitation responses which worry about students making their implementation too close to the demo. While it is hard to prevent something very close, it can at least make it harder for students. Ideally, the alternative flow would happen and students would not be able to access the source code.

User Stories

Story 1

As an instructor, I want to show students what the results of their assignment should look like so that they will better understand what I am looking for.

Acceptance criteria: Given that I have the code working myself, when I add it to the system then students will see the code in action during the lecture.

Estimated development time: 20 hours. In order to build this, the core of the system must be built. This will require about 20 hours of work to accomplish.

Story 2

As an instructor, I want students to only be able to see the published assignments so that they don't get confused when seeing assignments that haven't been explained in class yet.

Acceptance criteria: Given that an assignment is in the system, when it is published, then and only then will it be visible to students.

Estimated development time: 3 hours. To complete this, a configuration page will need to be added for instructors. Then, on the assignments page for students, some code will need to be added which hides unpublished assignments. Both of these tasks are relatively easy to accomplish.

Story 3

As a student, I want to see the results of the instructor's code so that I can get a better understanding of what I am trying to do in the assignment.

Acceptance criteria: When I access the assignment, then I am able to try many different inputs and see the outputs until I get a good idea of what is going on.

Estimated development time: 5 hours. This follows pretty easily from the existing system. If the instructor can do it, students should be able to as well. However, having students use it causes issues such as the need to obfuscate code so that students cannot copy what the instructor has done.

Story 4

As a student, I have completed a part of the assignment and I want to verify that I am getting the right answer so I can check for mistakes before moving to the next section.

Acceptance criteria: Given that I have tested my code and made some test cases, when I try those same test cases using the system, then I am able to see the results and check if they are the same.

Estimated development time: 0 hours. This task follows immediately from Story 3. While the intent of the student is different, the functionality is the same and would therefore require no additional development effort.