



COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

Final Report: AtG 5e

Thomas Bullock
Will Culpepper
Ian Johnston
Finn Pfeifer

June 15, 2022



CSCI 370 Summer 2022

Prof. Kathleen Kelly

Table of Contents

I. Introduction	1
II. Functional Requirements	2
III. Non-Functional Requirements	2
IV. Risks	2
V. Definition of Done	3
VI. System Architecture	3
VII. Software Test and Quality	4
VIII. Technical Design	4
IX. Project Ethical Considerations	6
X. Project Completion Status and Results	6
XI. Future Work	7
XII. Lessons Learned	7
XIII. Team Profile	8
References	8
Appendix A – Revision History	9

I. Introduction

The AutoGrader version 5.0 is a completely revamped version of the current grading software used by the computer science department for its introduction classes. This includes classes that use a multitude of languages, including Python, C++, and RISC-V. The client is unsatisfied with the current version of the AutoGrader due to a variety of issues, including a lack of accessibility in the user interface as well as an outdated overall code structure that makes the program difficult to maintain and expand upon. The new version of the autograder alleviates these issues by redesigning the user interface from the ground up with accessibility in mind, moving the code from Ruby (a language that the client considers outdated and hard to read and maintain) to Python using the Flask server API.

II. Functional Requirements

- Flask Server API
 - Full suite of unit tests for the Flask API
 - Mock up test results being generated
- Vue.js front-end
 - Make program as a single page application, meaning the application dynamically changes itself on a single web page rather than loading separate web pages when links are clicked
- SQLite database integration
 - SQL calls must be written in generic SQL for future integration with a Postgresql database
 - SQL calls should be prepared queries, so that inputs are only read as text and not potential SQL code. This ensures that there is no possibility of SQL injection, which compromises the security of the project.

III. Non-Functional Requirements

The non-functional requirements are focused around accessibility of the student-facing UI. This includes a reworking of the current keyboard controls for the text editor such that a student is able to tab in and out of it and making the website fully screen reader compatible for the visually impaired. The current system has elements of the page that are not conducive to full accessibility. First, the code editor captures the tab key as part of editing text, so a user cannot tab back out of it. For users who do not have access to fine mouse control as a result of visual or motor impairment, to be accessible a website should be able to be navigated by keyboard alone. Second, the results are displayed in a colored bar that not only is difficult for colorblind users to comprehend, but also consists of HTML elements that cannot be accessed with tabs and are only accessible through use of a mouse. Since these accessibility features would not hinder normally-abled users, they must be enabled and ready to use by default or instructions to enable them are provided.

IV. Risks

There are several risks currently associated with the project. The most prevalent is code security. While the code students provide is sandboxed before it runs to prevent SQL code from being entered that accesses private information from or erases the database, the risk of a potential attack getting through is never zero. Another is an inability to work out any potential formatting issues in the actual database not exemplified by the dummy database provided, as the IEEE code of ethics (and the Mines policies on academic integrity) prevents us from being able to use the actual database without compromising student information. A third is that not all screen readers are satisfied with the format of the project, with some edits made to revolve errors for some of the readers tested resulting in new errors for others and vice versa, an issue that stems from a lack of standards in screen readers in general.

At the start of the project, the primary risk came from a lack of prior understanding of how to use both of the main components of the solution, the Flask API and the Vue.js front-end. Due to this, we ran the risk of creating something that is hard to maintain, too slow for reasonable use, or completely non-functional. All of these risks were eventually addressed and resolved as the project continued.

V. Definition of Done

For the project to be considered 'done,' our client required that the front end must be able to submit and export code to the back end, the back end must successfully create a dummy set of test results on the submitted code and return those results to the front end, and finally the front end must display the results to the user in a readily accessible manner. To ensure the system met these requirements, this functionality was tested in two parts, first being able to display the results to a user, then being able to communicate properly with the back end.

To make sure that our new version meets our accessibility requirements, we have made the website able to be navigated without fine mouse control, and even without any mouse control at all. In addition, online resources such as WAVE [1] can evaluate a website's accessibility. Having designed a website that meets those various accessibility requirements and does not have any accessibility issues identified by such programs allowed us to make a website that can be considered satisfyingly accessible.

The back end must be able to receive code from the front end, run test cases with the code, and send the results to the user, along with the expected result and whether the user's results were correct. This functionality was tested in two parts: first, our program must be able to run the code and compare results to expected results, then it must be able to receive code from and communicate results to the front end.

VI. System Architecture

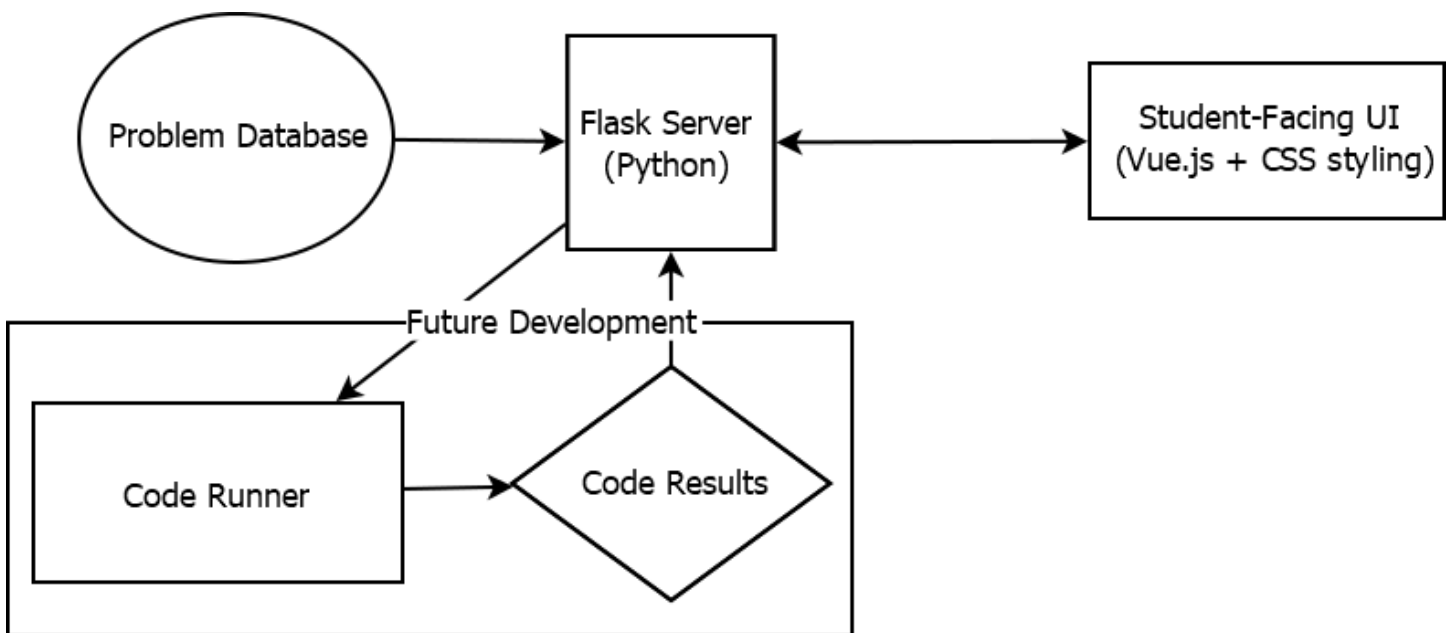


Figure 1: System Architecture

AutoGrader 5.0, the architecture of which is displayed in Figure 1, is built on a Python/Flask server with a Vue.js front-end. Both ends are relatively independent, and are run as separate processes. The Flask server

pulls from the AutoGrader problem database and sends the problem statement to the student’s in-browser Vue client. Because the actual database can’t be used, the client has sent an example of the database schema, the style of which has been fully accounted for in the project to allow the actual database to be swapped in place of the example without issue. Once the student submits their code to the server through the Vue-Flask pipeline, the results of the tests are then sent from the Flask server to the Vue client. These “results” are not true results from the code that’s been run, as we did not have time to reach our stretch goal of running it in the sandboxed Ruby environment. Instead, a list of dummy results are passed to the client, as a way of showing how the current user interface functions.

VII. Software Test and Quality

AutoGrader 5.0 was tested server-side on the Flask API with Python’s unit testing framework. Additionally, accessibility tests were performed client-side using the NVDA screen reader as well as the various testing services provided by WEB from webaim. The server-side tests were used to locate and correct programming mistakes, such as those that hinder the program’s effectiveness or leave it vulnerable to crashing due to bad input/routing. This has ensured that a user could access, navigate, and work on only those assignment problems they have permission to access in the final product. The accessibility tests were used to ensure that the website has an ADA compliance of at least AA, meaning almost all users can access it [2], by identifying situations where ADA regulations are violated and correcting them. Particular emphasis was placed on allowing screen readers to read any component of the webpage, especially within the editable text box, and designing the website such that it can be fully navigated using only a keyboard.

VIII. Technical Design

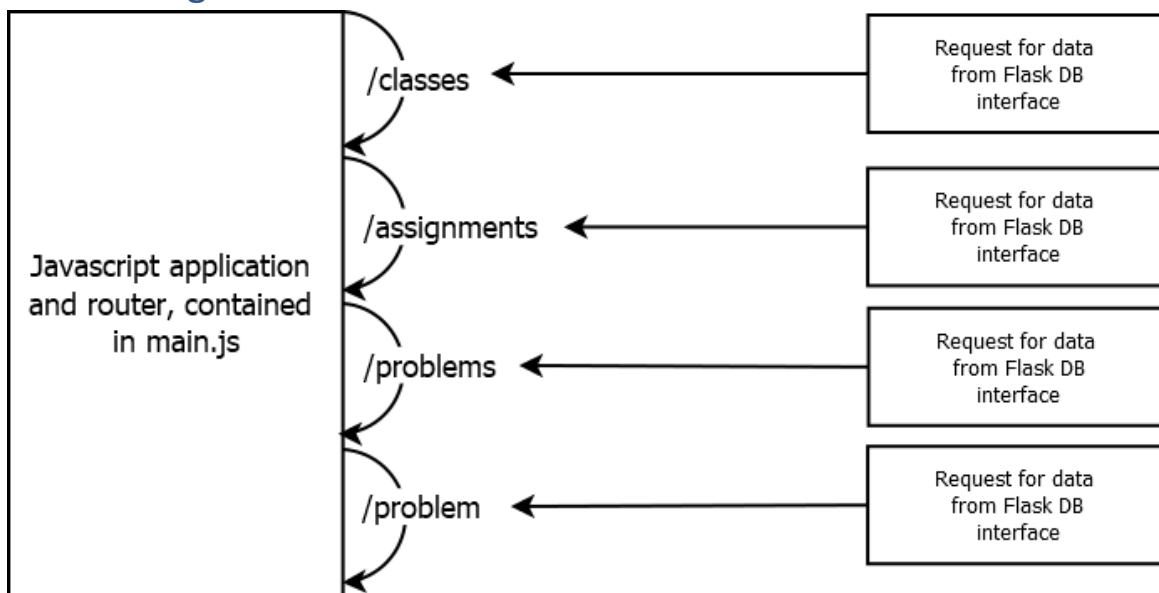


Figure 2: abstraction of the single-page application

One interesting aspect of the front-end Vue interface is that it’s a single-page application, which eliminates the requirement for any standalone HTML documents to be served up by the back-end. With each button click that navigates the site, the Javascript router (shown in Figure 2) pulls up a Vue component corresponding with the url. At each new destination, depending on what component is being served, a request to the Flask server is made telling it to send data from a certain database query—ending with the display of a user’s classes,

assignments for that class, problems for an assignment, or full problem content. With Vue.js, we've created a dynamic, blackboxed front-end interface that can theoretically be swapped out with some other framework (should the client or future field session team deem it necessary).

Another critical aspect of the project is the database integration with Flask. Since it's a Python framework, it was a relatively easy task to make calls to the sample SQLite database that was given to us by the client. Going back to the front-end diagram in Figure 2, when the page changes, the application sends an ID corresponding to a course, assignment, or problem to the Flask server to be used for prepared queries. The results of said query are passed into the data for the page that the user is navigating to, allowing the program to do things like showing a student's available classes to them. This is shown below in Figure 3, the Assignments page. The buttons you see are generated from a database query to get a class's assignments list. Figure 4 shows the result of clicking into a problem of the Python Tests assignment, with the title and requirements for it also queried from the database.



Figure 3: Assignments page

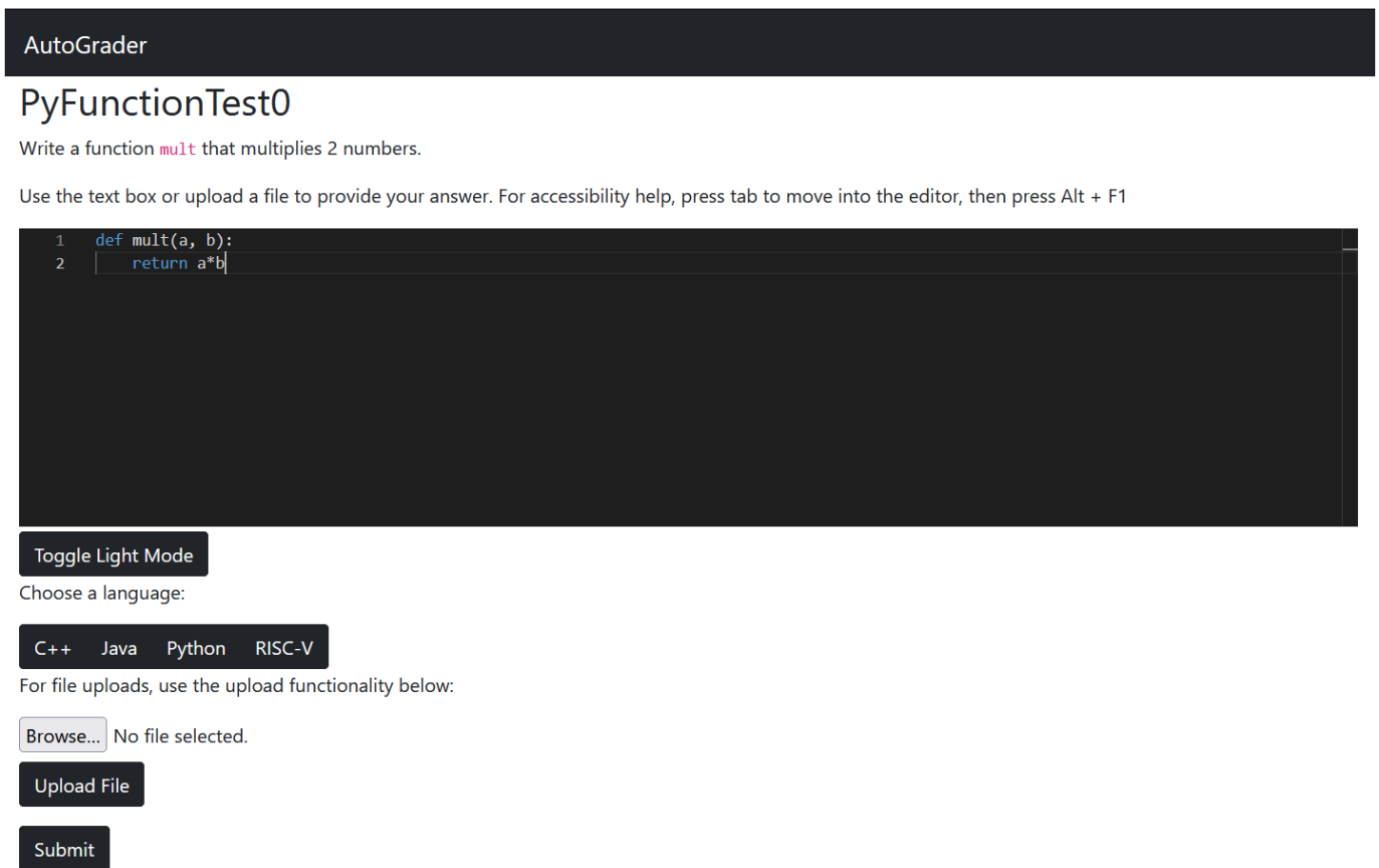


Figure 4: Problem page for a problem chosen from the Python Tests assignment.

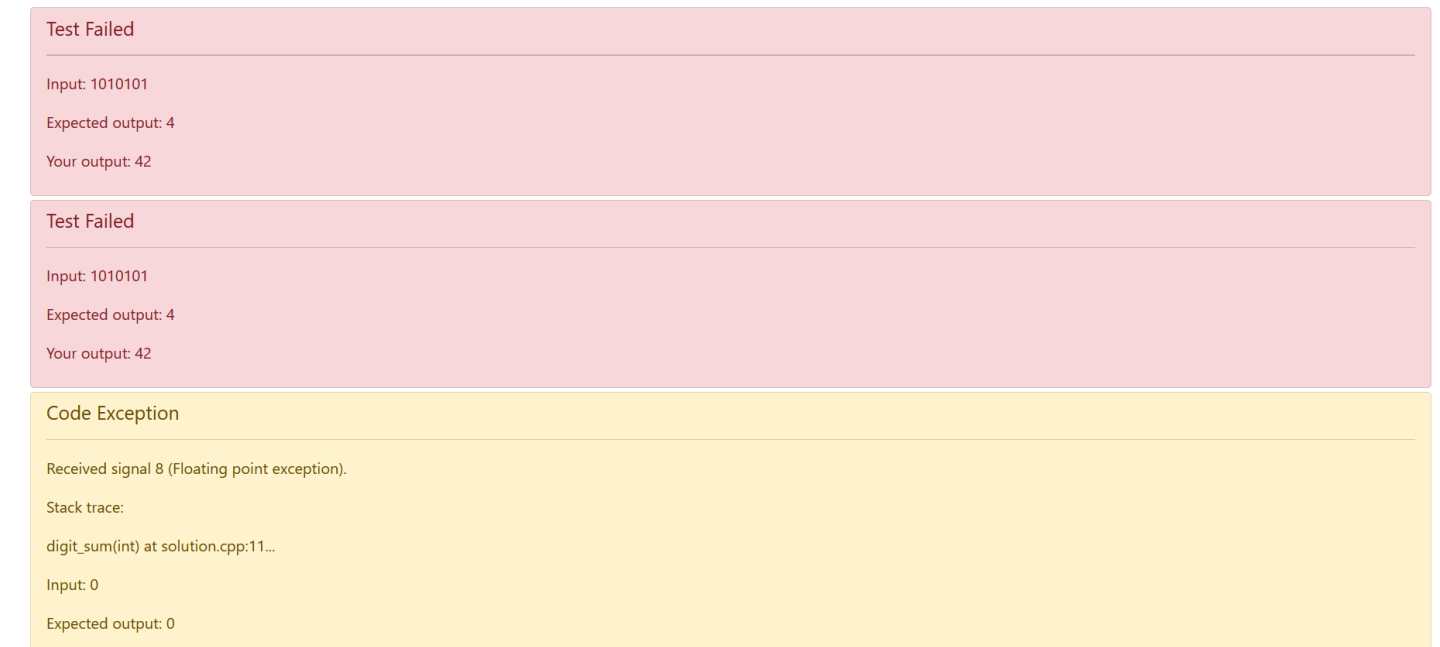
IX. Project Ethical Considerations

There are multiple IEEE ethical standards that have been taken into consideration in making AutoGrader 5.0, of which three in particular are emphasized. The first is Principle 1, Section 7, which requires that AutoGrader 5.0 accounts for users with disabilities that may otherwise prevent them from successfully using the project. This section had been given the highest priority because the most frequent problem students reported with the current version of AutoGrader was an inability to use the site due to a violation of this principle. Those who are blind, have motor function issues, or otherwise lack fine mouse control are unable to complete assignments. The current version relies on sight and mouse clicks to escape the text editor and view test results. To address these issues and meet this accessibility standard, the team has implemented tab-only navigation and screen reader support alongside other supportive measures—as detailed by the ADA website guidelines. The second is Principle 3, Section 12, which requires that the team do not infringe on the privacy of anyone who will use AutoGrader 5.0, as well as anyone who used previous versions and are still present on the database. This section is inherent to the AutoGrader as a concept, as students access the program through their registered school profiles. As such, to meet this standard, the team cannot see the actual database that holds important information such as usernames and passwords. Instead, we used an example database as a template to design necessary security measures and support with a final design that allows the actual database to be swapped in for the example without causing any compilation or runtime errors. The third is Principle 1, Section 3, which requires that AutoGrader 5.0 only be approved if all programming goals, safety measures, and testing are successfully achieved. Because the class is designed to meet this standard, with the client only providing their final approval once these requirements and others are met, the team met this requirement by following the weekly guidelines as well as regularly meeting with the client to discuss progress and expectations.

X. Project Completion Status and Results

As of June 9th, 2022, the project was approved by the client to have met the minimum requirements for our definition of done. AutoGrader 5.0 has achieved basic functionality, but until some future work is completed by either the teacher or another team, it is not ready to be implemented in the place of the previous version in time for the next semester. Our software has a Vue.js front-end working independently from a Flask back-end, adding functionality as a black-boxed abstraction that can have components swapped out with ease. Additionally, we have tested a free screen reader software (NVDA) for accessibility for visually impaired students, with complete success of accessing all parts of the website.

Results:



The screenshot displays the results of a code run in AutoGrader 5.0. It consists of three distinct colored sections. The first two sections are pink and each indicate a 'Test Failed' status. Both tests show an 'Input: 1010101', an 'Expected output: 4', and a 'Your output: 42'. The third section is yellow and is titled 'Code Exception'. It reports a 'Received signal 8 (Floating point exception)', includes a 'Stack trace' pointing to 'digit_sum(int) at solution.cpp:11...', and shows 'Input: 0' and 'Expected output: 0'.

Figure 5: results of run code

Shown above in Figure 5 is a sample of the results section in AutoGrader 5.0. Even though the results stubs are still color-coded, the text content in them meets the needs of our client as the text is easily readable from a screen reader and able to be navigated to using tab inputs. As the currently used Autograder requires mouse input in order to read its results, this was a much needed accessibility improvement.

XI. Future Work

There are a few stretch goals that we did not meet that have been earmarked as future work. First and foremost, we were unable to integrate the actual database at any point during the project, leaving it up to the client to do so. The current AutoGrader database contains user-sensitive information such as student CWIDs, as well as academically-sensitive information such as student submissions, grades, and results, which only Mines faculty such as the client should have administrative access to. This leads to another future goal, implementing administrator functionality to allow approved faculty to add problems, solutions, assignments, classes, class sections, and add students to classes. A third goal for future work is to redesign the Ruby sandbox environment for running the code, using an easier to understand language such as Python again in a similar fashion to the rest of the project. This was considered by the client as a stretch goal due to being more complicated than the time frame of the project allowed, and because the front-end and back-end were more immediate concerns.

XII. Lessons Learned

First and foremost, we should have asked for help with the project more often. Most of the first two work weeks were spent troubleshooting errors for hours on end while we learned how to use the required

components, and our overall productivity was far lower than it should have been. It became so much of a problem our client and advisor both expressed their concerns about the project not finishing on time, which motivated us to get back on track. With help from the client themselves and the people at World Wide Technology during their office hours, we were able to get our project complete on time, but were unable to touch most of our stretch goals because of this time crunch. We also learned how to *be* on a software engineering team. With a combination of pair programming and properly managing Git, we created a successful working environment. Additionally, we learned a lot about making web applications, from basic HTML programming, to CSS styling, to Javascript scripting. Furthermore, we learned a lot about making single-page applications. With our architecture and how we coded our front- and back-end, it is possible to swap out each component with relative ease.

XIII. Team Profile

Tommy Bullock - senior in CS-Business Focus from Fort Worth, TX. Experience as a software engineer at Zedasoft, Inc. Enjoys hiking, soccer, fighting games, and food tourism.

Will Culpepper - Rising senior in CS-General from Sugar Land, TX. Experience as a Teaching Assistant with the Computer Science department. Enjoys playing instruments, playing video games, and Dungeons & Dragons

Ian Johnston- Junior in CS-Data Science Focus from Fort Collins, Colorado. No prior work experience. Typically prefers walking or resting at home. Enjoys nothing in particular.

Finn Pfeifer - Senior in CS-Data Science Focus from Denver, Colorado. No computer science job experience, but currently working at Starbucks. Enjoys snowboarding, fencing, board games, and tabletop RPG games.

References

[1] <https://wave.webaim.org/>

[2] <https://www.testpros.com/articles/what-is-ada-compliance-and-what-does-it-mean-for-your-website/#:~:text=ADA%20compliance%20is%20short%20for,often%20confused%20with%20508%20compliance.>

Appendix A – Revision History

Revision	Date	Comments
New	5/19/2022	All members of the team met up to begin work on the final draft by drafting the Introduction, Functional Requirements, Non-Functional Requirements, Risks, and Definition of Done sections, after meeting with the client and the advisor for the first time. On the programming side, all members found and shared with each other initial sources and tutorials related to using Flask, Vue.js, and PyCharm, and began working on the front end of the project based on the knowledge gained.
Rev – 1	5/29/2022	In the week of the first revision, all members of the team met up to draft the System Architecture section, under the guidance of the advisor. On the programming side, following a meeting with the client to show previous progress and clarify some requirements, all members contributed to design a functioning vue front-end for a single problem using a Monaco Text Editor and began work on designing and integrating the corresponding flask back-end. Comments were also added, while coordinated teamwork gained greater emphasis due to a previous lack of communication resulting in several merging issues.
Rev – 2	6/3/2022	In the week of the second revision, all members of the team met up to draft the Software Test and Quality section, as well as prepare and give the first presentation on the project to a larger audience, both with the aid of the advisor. On the programming side, all members of the team worked together to send and receive information between the Flask and Vue components with the assistance of the client. Following the third meeting with the client, the team then began working on refactoring the project to better match coding standards, involving the removal of several html files, while work began on integrating support for SQL databases and dynamically creating web pages based on the results.
Rev – 3	6/12/2022	In the week of the third revision, all members of the team prepared and gave the second presentation on the project to a larger audience, taking into account feedback both individually and as a team from the first presentation, despite a quarantine leaving half of the team unable to attend in person. On the programming side, the remaining minimum requirements were achieved, with major goals including refactoring the program into a single-page application, completing the user interface navigation, automatically generating classes, assignments, and problems using the given database, appropriately grading student submissions, designing unit

		<p>tests for the Flask API, testing compatibility with multiple screen reader, and cleaning up the code afterwards. As a result, by the end of the week, the project was approved by the client as functional, with most of the remaining stretch goals to be implemented as future extensions by the client or another assigned team. The Project Ethical Considerations section was also drafted, following the discovery that it was meant to be a part of the previous revision.</p>
Rev – 4	6/15/2022	<p>In the week of the fourth revision, with the project completed, all members of the team worked on preparing to give the third and final presentation to a larger project as well as preparing the final draft of the report, using feedback from both students and faculty alike. On the programming side, the main branch was merged with the prototype branch to serve as the final definitive branch of the program, the readme was updated to provide assistance in creating, compiling, and running the project from a new system, and a few other minor editors were made to improve performance and clean up.</p>