A RESTFUL FRAMEWORK FOR WRITING, RUNNING, AND EVALUATING CODE IN MULTIPLE ACADEMIC SETTINGS

by

Christopher Ban

A PROJECT DEFENSE

Presented to the Faculty of

The School of Computing at the Southern Adventist University

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Rick Halterman

Collegedale, Tennessee

December, 2017

A RESTFUL FRAMEWORK FOR WRITING, RUNNING, AND EVALUATING CODE IN MULTIPLE ACADEMIC SETTINGS

Christopher Ban, M.S.

Southern Adventist University, 2017

Adviser: Rick Halterman

In academia, students and professors want a well-structured and implemented framework for writing and running code in both testing and learning environments. The current limitations of the paper and pencil medium have led to the creation of many different online grading systems. However, no known system provides all of the essential features our client is interested in. Our system, developed in conjunction with Doctor Halterman, offers the ability to build modules from flat files, allow code to be compiled and run in the browser, provide users with immediate feedback, support multiple languages, and offer a module designed specifically for an examination environment.

Contents

List of Figures

Chapter 1

Introduction

In academia, students and professors want a well-structured and implemented framework for writing and running code in both testing and learning environments. In a test or quiz setting, students prefer to type and have the ability to execute code. In an instructional medium such as a handout or online textbook, students would prefer to try out code snippets right in the browser without the need to install or switch to another program. Some on the other hand would like the ability to build exams and quizzes without multiple or repetitive steps, possibly leading to a decrease in the creation of these activities (which Cheang et al. [4] has shown can be crucial.) Finally, automated and near-immediate grading and feedback remains a huge advantage to both parties. Our client, Doctor Halterman of Southern Adventist University, has recognized this call for a framework that satisfies these needs.

Programming tests and quizzes today often are given through the paper and pencil medium. This medium is not desirable as it does not allow for immediate feedback, which can frustrate both teachers and students. Students must wait for results and teachers must invest large amounts of time in grading students'

answers. Another problem that arises due to this medium is the inability to type and format code. For students in a high-pressure environment such as a quiz or test there is not always time to neatly write code. Erasing or revising written code can further reduce readability. As a result, teachers and students are forced to read and work with messy answers. Finally, this medium does not allow for the possibility of compiling and running code. The ability to execute code allows students to fix small mistakes such as syntax errors and other problems that do not necessarily have any implication on the testing content.

Online textbooks and other related learning environments suffer from similar problems (i.e., no immediate feedback, and readability issues), with the addition of the need to switch to a different application completely. Using these media require students to leave the browser, have a compiler or IDE installed, and copy or write code snippets in order to try concepts out on their own.

These issues have led to the use and/or creation [5, 6, 7] of two separate products: Learning Management Systems (LMS), and automated evaluation systems. LMS' like Moodle [8, 6, 7] and Edomodo [9, 10, 11] both provide teachers with a web interface for building instructional articles, exams, and quizzes and provide students with a common interface to access those activities. An LMS can also serve as a secure environment for testing and quizzes, and offer features such as multiple attempts, instant feedback, and question randomization. LMSs also have the capability to integrate add-ons which can provide the ability to compile and run code written by students, such as Moodle's plug-in feature [12].

Modifying or building quizzes and exams via an LMS requires multiple, repetitive steps that are more time consuming than editing a simple flat file. In addition, current LMS do not not give statistics on test and quiz properties,

such as time spent per question. Existing add-ons for an LMS such as Moodle do not address this issue, and the compiler features do not contain some of the requested properties mentioned above (e.g., immediate evaluation and feedback).

As for automatic evaluation systems, students can type and execute code, run tests, and receive immediate feedback. However, widely used solutions simply support assignments and code challenges, excluding examination environments and general practice environments.

Therefore, based on our client's interests stated above, we believe no application exists which allows users to quickly build learning modules, exams, or quizzes via structured text files for students to use, that also give them the ability to write and run code and receive immediate feedback on submitted work.

This project addresses these issues by creating a framework which provides a medium with the following goals:

- Allow professors to quickly build activities via simple flat files
- Allow students to be able to type code in an environment that provides syntax highlighting
- Allow students to compile and run code in a controlled environment
- Allow for the ability to build activities specifically for an examination environment
- Allow students and professors to get immediate statistics (i.e., time spent per question/exam, student written code, and data for each test case run for teachers, and grading data for both)
- Allow for the support of multiple programming languages

Given this framework, professors can quickly and easily create quizzes, tests, and other learning environments, and students can edit and compile code and receive immediate feedback. We believe this will help professors limit time spent on building and grading tests and quizzes, and help students focus more on actual test content, rather than on formatting and cleaning up answers.

For the remainder of our paper, we first cover the background in Chapter 2. Chapter 3 and Chapter 4 will cover our project architecture/design and testing respectively. Finally, Chapter 5 presents our conclusion on the research found and our future work on the Master's Project.

Chapter 2

Background

Currently, there are virtually no systems in widespread use that provide automated feedback tailored for programming. However there are a limited number of automated grading systems (as shown in a survey paper by Kyrsti M et al. [13]) and to move from niche to mainstream it will be necessary for these automated grading systems to exist and incorporated in LMSs because schools are moving to such systems as a standard platform on their campus [10].

The most popular systems in use in academics today are Learning Management Systems (LMS). These products provide teachers and students with a familiar digital based medium in which to access assignments, grades, and even tests and quizzes. These systems also allow for modules to be added in order to provide new features developed by third parties. In addition to these systems there exists more focused projects that attempt to provide similar services, specifically tailored toward programming in education. In this section we explore the abilities of popular LMSs, and open source programming evaluation systems which attempt to provide similar functionality.

Our background analysis is broken into two different sections, Learning

LMS	Customers	Users
Edmodo	350,000	58,000,000
Moodle	70,569	89,237,532
SuccessFactors	6,000	45,000,000
Blackboard	16,000	24,000,000
Cornerstone	2,600	27,200,000
SkillSoft	6,700	19,000,000
Instructure	3,000	20,000,000
Saba Software	2,000	33,000,000
Schoology	2,000	20,000,000
Litmos	3,500	6,000,000
Latitude Learning	14,299	4,218,001
Edsby	10,800	1,910,000
Collaborize Classroom	57,600	420,000
Brightspace	2,000	15,000,000
WizIQ Inc	5,000	500,000
NEO LMS	8,700	970,000
Absorb LMS	720	6,200,000
TalentLMS	3,132	1,552,000
Educadium	9,300	75,000
DigitalChalk	4,090	908,440

Table 2.1: The top 20 most popular LMS packages, measured by a combination of their total number of customers, active users, and online presence (customer and user data shown in figure above) as of November 2017 [14].

Management Systems in Section 2.0.1 and automated programming evaluation systems in section 2.0.2, taking into consideration their strengths and weaknesses as follows:

- Test and quiz building
- Code formatting and compiling ability
- Answer evaluation and feedback

These three topics will provide a perspective on how the system functions and how well it addresses our goals.

2.0.1 Learning Management Systems

In the subsection below we evaluate the LMS based systems selected for analysis. Table 2.1 shows the most popular LMS software packages [14], from which we select two mature LMS platforms (Moodle [8, 6, 7] and Edmodo [9, 10, 11]) for analysis along with two LMS modules for Moodle called the Virtual Programming Lab [1], and Javabrat [2].

2.0.1.1 Analysis: Moodle

Moodle [8, 6, 7] is a free, open-source LMS for producing modular internet-based courses that support a modern social constructionist pedagogy. Because it is open-source, Moodle has a large community contributing many modules that extend its feature set.

Test and quiz building Moodle has a quiz activity module which allows the teacher to design and build quizzes consisting of a large variety of question types, including multiple choice, true-false, and short answer questions. To make a new quiz/test assignment in Moodle, the teacher may either import a quiz from a range of different flat file formats [15], or manually add a quiz module to the course. When creating a quiz manually, the teacher must first create the quiz activity. To add a new question to the quiz's question bank, the teacher must click 'Add' and then '+ a new question'. From the next screen, the teacher must choose the specific question type. Once the question form is filled in with the question and any necessary options are added the question can be saved and added to a question bank shown in Figure 2.1. The teacher must then add questions to the quiz from the question bank.

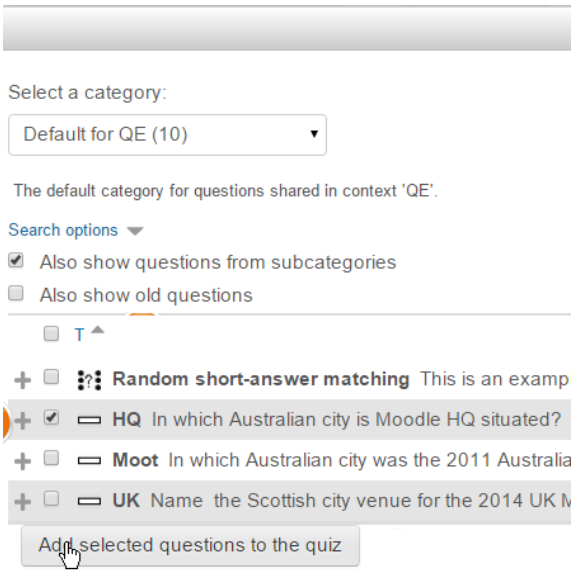

Figure 2.1: A simple Moodle quiz module.

With all of the steps in manually building quizzes, teachers must invest time in adding items to question banks on top of formulating the actual question content. While importing questions from flat files is useful in avoiding this overhead, the type of questions available to be asked are still limited by Moodle's standard questions types [16].

Code formatting and compiling Moodle quiz activity modules can accept multiple choice and text input. However, these input methods do not allow students to properly format code, or show syntax highlighting. Similarly, students do not have the ability to execute code and see results produced by the compiler which limits the types of questions that can be presented (e.g., we cannot ask a student to compile and debug based on the compiler output.)

Answer evaluation and feedback Moodle quiz activity modules provide the ability to evaluate and grade answers immediately and give students feedback,

which is another improvement on the pencil and paper medium. Unfortunately, due to question limitations mentioned above, there is no ability to run test cases on student defined code and grade submissions automatically.

2.0.1.2 Analysis: Edmodo

Edmodo [9] is a free social learning platform for schools. It provides teachers and students with a secure way to share classroom materials, and access homework, grades.

Test and quiz building Edmodo has a quiz activity that allows teachers to create online quizzes for students and receive instant feedback on grade results. Unlike Moodle, all Quizzes and Quiz questions must be created within Edmodo and cannot be imported [17]. The steps to create quizzes are comparable to Moodle's, and incurs similar overhead. To create a quiz in Edmodo, a teacher must create a quiz activity, and manually select a question type. Once the question form is filled in with all necessary information the question can be saved and added to a question bank.

Similarly with Moodle, Edmodo is limited by the building overhead and the types of questions available to be asked (Multiple choice, True/False, Short Answer, and Fill in the blank).

Code formatting and compiling Again, very similar to Moodle, Edmodo quiz activities can accept multiple choice and text input, which allow students to type code. However, these input methods do not allow students to properly format code, or show syntax highlighting. Nor do students have the ability to

execute code and see results produced by the compiler which limits the types of questions that can be presented, similar to Moodle.

Answer evaluation and feedback Edmodo quiz activities provide the ability to get feedback immediately. Due to question limitations mentioned above however, there is no ability to run test cases on student's code and grade submissions automatically in that manner.

2.0.1.3 Analysis: The Virtual Programming Lab

The Virtual Programming Lab (VPL) [1] is an activity module for Moodle that manages programming assignments and whose main features are:

- Students can edit program source code in the browser
- Students can run programs interactively in the browser
- Students and teachers can run tests to review the programs

Test and quiz building This activity module for Moodle is unfortunately tailored for programming assignments. Because of this, it is not possible to create an assignment in a test or quiz format.

Code formatting and compiling VPL allows files to be edited from the browser using the code editor component, as shown in Figure 2.2. It also has the ability to supply initial skeleton code. This added functionality allows students to run and evaluate programs without requiring them to install compilers or IDEs. The code editor component provides formatting as well as syntax highlighting, which is useful for students both reading and writing code.

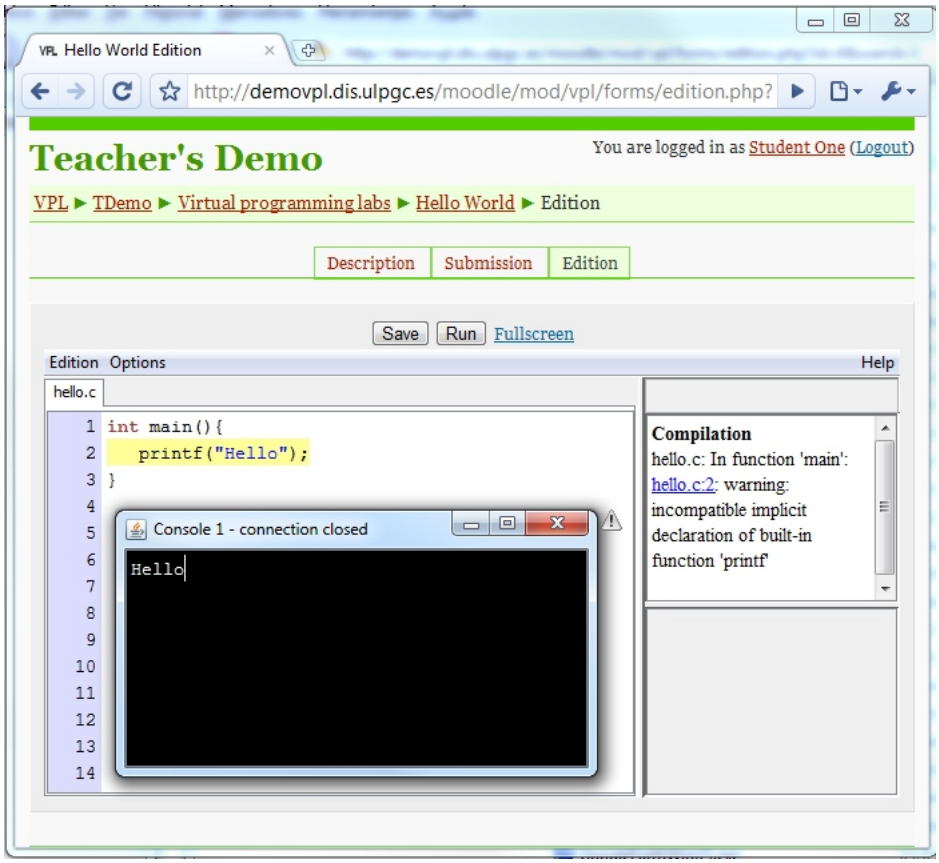

Figure 2.2: A simple VPL module programming assignment [1].

Answer evaluation and feedback VPL allows for scripts to be set to evaluate every submission, which enables students to receive feedback on their code based on test cases. However, this takes more time to set up than simply defining test cases and is geared more towards programming assignments instead of quizzes and tests.

2.0.1.4 Analysis: Javabrat

Javabrat [2] is a web-based grader useful for students learning Java and Scala languages. It is primarily in the form of a Moodle plugin that facilitates instructors to grade Java assignments.

- Students can edit program source code in the browser
- Students and teachers can add problems to be solved
- Students and teachers can run tests to review the programs and automatically grade them

Test and quiz building Similar to VPL, Javabrat is tailored towards programming assignments. Because of this, it is not possible to create an assignment in a test or quiz format.

Code formatting and compiling Javabrat allows for code to be typed directly into the brower, but does not support syntax highlighting, as shown in Figure 2.3. Similar to VPL, Javabrat has the ability to provide an initial skeleton code for students to build on. This setup allows students to type out code and evaluate results without having to leave the browser.

Answer evaluation and feedback With Javabrat, program results are evaluated immediately. When a student completes a solution, it can be evaluated in the browser by running pre-defined test cases. Once the evaluation is done, Javabrat sends back an HTML report file generated by the back-end as shown in the lower part of Figure 2.3.

2.0.2 Automated Programming Evaluation Systems

The following subsection will evaluate the automated programming evaluation systems selected for analysis. From our research we select three popular automated programming evaluation systems: Web-CAT [18, 19], Turing's Craft [20], and CourseMarker [5, 21].

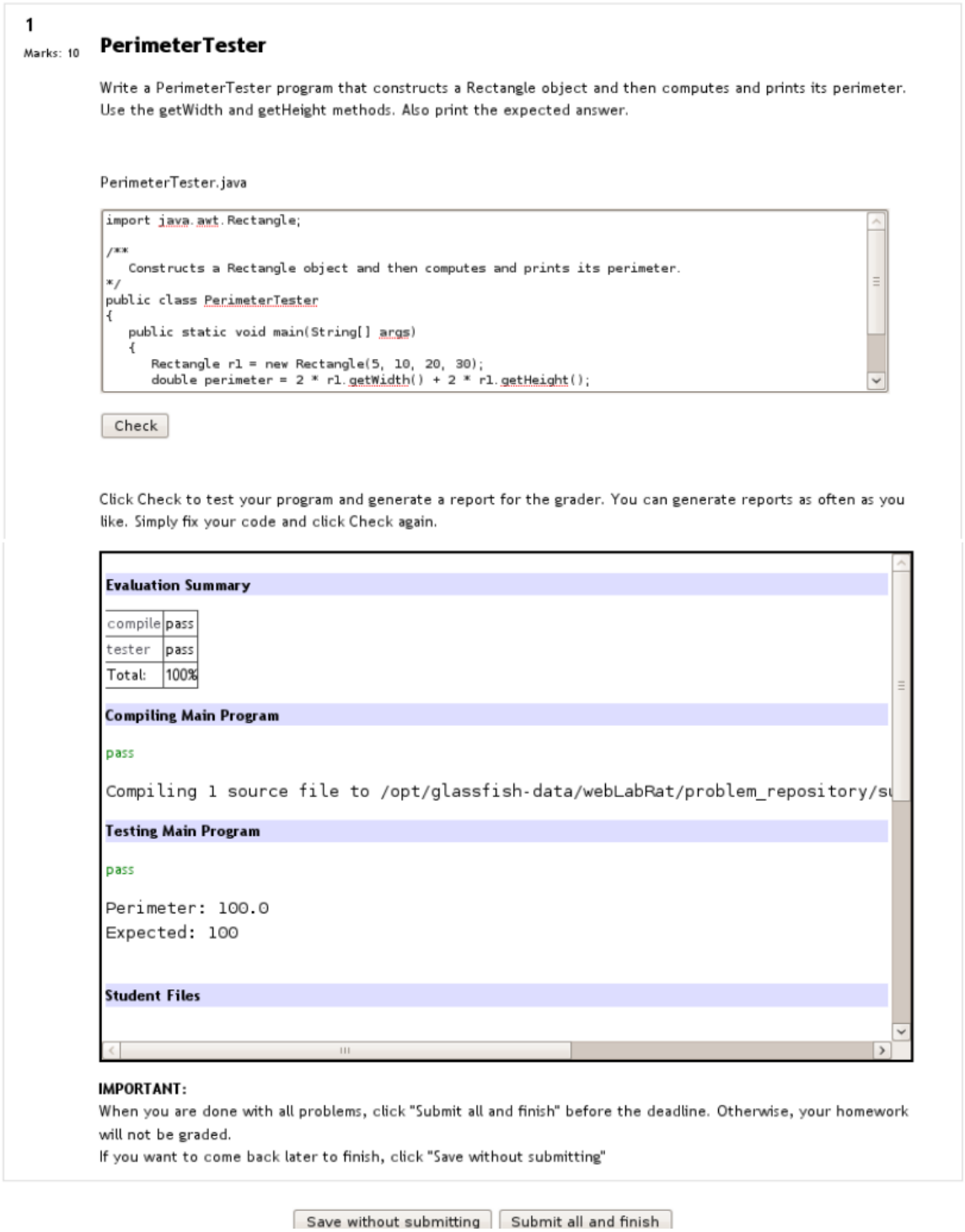

Figure 2.3: A basic Javabrat assignment example [2].

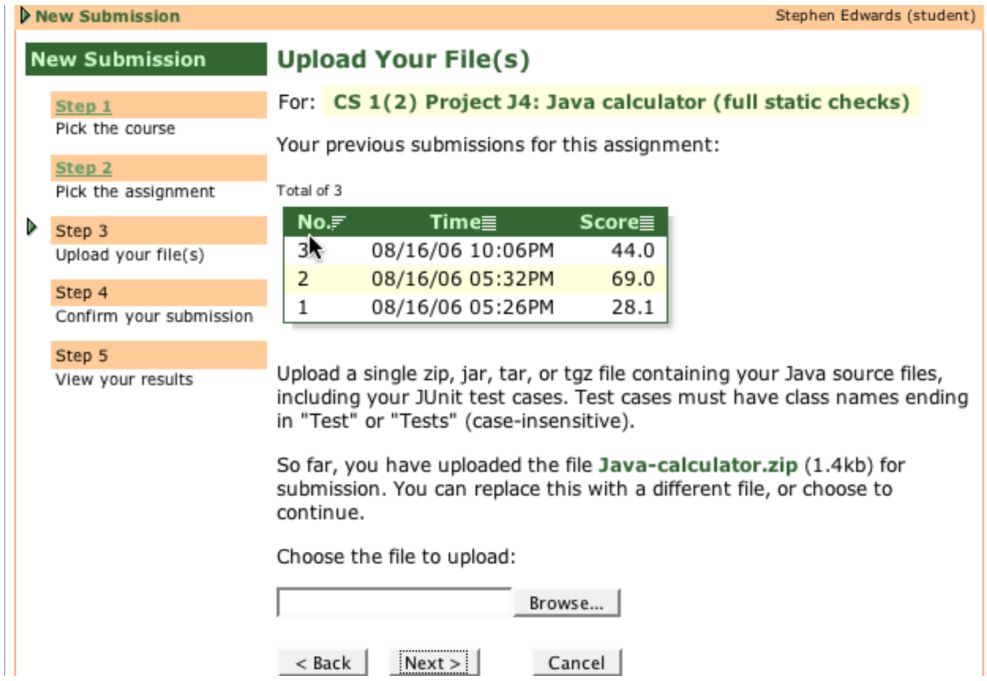

Figure 2.4: A simple Web-CAT module assignment [3].

2.0.2.1 Analysis: Web-based Center for Automated Testing

The Web-based Center for Automated Testing (Web-CAT) [18] is a flexible automated grading system designed to process computer programming assignments. Its main features are:

- Provide immediate feedback based on test-cases and code coverage
- Enable flexibility and extensibility by way of a plug-in based architecture
- Provide a secure and portable product

Test and quiz building Similar to VPL, Web-CAT is tailored for programming assignments. Because of this, it is not possible to create an assignment in a test or quiz format.

Code formatting and compiling Web-CAT does not allow files to be edited from the browser. It does however have the ability to supply initial skeleton code. Without this functionality, students must use their own compiler or IDE. In Web-CAT, students are required to upload their source code to the correct assignment as shown in Figure 2.4.

Answer evaluation and feedback Web-CAT uses a composite scoring system, which enables students to receive feedback on their code based on their code correctness, test completeness, and test validity. These three measures, taken as percentages, are then used to form a composite score [22]. This formula ensures that no aspect of the approach can be ignored, or the student's score will suffer. Code correctness is based solely on student written tests, unlike the other two scores. For test completeness, instructors must choose what level of coverage should be used for testing completeness, and for test validity, instructors must set specific reference tests to ensure the student's tests are accurate.

2.0.2.2 Analysis: Turing's Craft CodeLabs

Turing's Craft CodeLab [20] is the commercial equivalent to WebToTeach [23], developed to provide fully interactive, hands-on practice environments for learning computer programming. The supported topics start with the imperative programming core of Python, C, C++, and Java and then go on to address procedural and object-oriented programming. The exercises are targeted at a typical CS1 syllabus.

Its main features are:

- Students can edit and run source code in the browser

- Provide immediate feedback based on test-cases in the browser
- Provide teachers with a selection of exercises ranging from short and focused to more complicated problems intended to be used as teaching aids

Test and quiz building Because Turing's Craft is targeted towards programming assignments and challenges, it does not support quiz or exam formats.

Code formatting and compiling The Turing's Craft engine is hosted and supported by Turing's Craft and the CodeLab runs in the browser, so students are able to write and run code directly in the browser. Unfortunately, it does not support syntax highlighting.

Answer evaluation and feedback Turing's Craft CodeLab provides student with immediate feedback on submitted solutions. With each submission, the instructor's roster is automatically updated regarding solution correctness. As a teaching aid, Turing's Craft research found that CodeLab works best when required as 5-10% of a student's grade.

2.0.2.3 Analysis: CourseMarker

CourseMarker [5] was developed for the assessment of Java programming skills of CS students. CourseMarker has been tailored to student needs by teacher and student suggestions, building off of an older automated assessment tool called "Ceilidh" [21]. Its main feature is to allow students and teachers to run tests and receive in-depth metrics and assessments immediately. CourseMarker must be installed instead of accessing it through a browser.

	Exam environment	Code formatting	Immediate feedback	Multi-lang support
Moodle	X		X	
Edmodo	X		X	
VPL		X	X	X
Javabrat			X	
Web-CAT			X	X
Turing's Craft			X	X
CourseMarker			X	X

Table 2.2: A visualization of how many of this project's goals are met by each solution evaluated above.

Test and quiz building Similar to VPL [1] and Web-CAT [19, 18], CourseMarker is tailored for programming assignments. Because of this, it is not possible to create an assignment in a test or quiz format.

Code formatting and compiling CourseMarker does not allow files to be edited from the client, but it does have the ability to supply initial skeleton project files. Without this functionality, students must use their own compiler or IDE.

Answer evaluation and feedback CourseMarker provides grading results immediately once code solutions are submitted. Students receive feedback on typography, scalability, and test cases. With typography, CourseMarker checks the program layout, indentation, and usage of comments. Regarding scalability, CourseMarker attempts to use different sized datasets when possible. Lastly, CourseMarker evaluations solutions by checking for exercise dependent features defined by the teacher.

In summary, as seen in table 2.2, each of the popular solutions reviewed met different parts of our client's overall goals but none met all of them. While a few such as Web-CAT [19, 18] came close, many of the widely used solutions do not offer everything this project is attempting to accomplish.

Chapter 3

Project Implementation

Our system creates a RESTful web service that provides the following services:

- A quiz/test module generated from a quick and easy flat file which defines activity settings, question and answer values, grade-point values, and estimated difficulty levels, in a medium that allows for code formatting
- An interface to compile, execute code and automatically evaluate output against test cases

Seven external libaries were utilized in the development of this system, and the system contains over 2,000 lines of non-library code. In this chapter, we discuss the development approach, task delineation, system process workflow, and final deliverables for this system.

3.1 Development Approach

We implemented our system on an Ubuntu unix VM, using Node.js and jQuery in order to take advantage of modules such as ExpressJS [24] which provided

Module/Plugin	Purpose
CodeMirror	A text editor implemented in JavaScript used for styling and editing code
Express	An MVC-based web framework for Node.js used to handle client/server interactions
Body-parser	Node.js body parsing middleware used to parse JSON files
Connect-busboy	Connect middleware for busboy used to handle file uploads
Deasync	JavaScript wrappers of Node event loops used to keep from blocking threads
Fs	Simple wrappers of standard POSIX functions used to provide File I/O
Uglify-js	JavaScript parser/compressor used to minify scripts
directory-tree	Creates a JavaScript object used for displaying a directory tree.
child_process	A module which spawns a shell and executes commands within that shell

Table 3.1: Node.js modules and JS plugins used in this system

robust sets of frameworks and features for web applications. These requirements are detailed in Appendix A. While some synchronous functions were used for purposes such as writing data to file and handling compiler input/output, we generally utilized Node's asynchronous nature and Javascript's Promise objects to minimize blocking. We also made use of the Model-View-Controller design pattern in order to organize our code and system structure.

Since this system's intent focused on an exam environment, we considered security a high priority. With regards to client-side execution, since all scripts sent from the server to be executed client-side exist in a specific namespace, interaction through the console window cannot reach the code or any variables during normal execution and therefore data integrity is ensured. The design of the system ensures that no exam solutions ever touch the client side and that the system evaluates all submissions on the servver. Section 3.3.5 covers security in more detail.

We have utilized two internal Node.js modules (fs, child_process), five external modules, and two jQuery plugins in order to speed up development. All modules and plugins were licensed under the MIT license [25], with one module licensed under the BSD 2-Clause [26]. Table 3.1 lists each of these modules and their purpose in the system.

Task	Work Hours
System design and architecture	70
Front-end Examination module	85
Server-side Examination evaluation	80
Administration module	80
Server-side code execution and evaluation	120
Final changes, bug fixes, testing, and documentation	90
Project Completion	525

Table 3.2: A visualization of this project's goals and rough estimate of hours taken to complete

3.2 Task Delineation

Development was broken up into six major categories for testing and development. Table 3.2 shows an overview of major tasks, including work hours.

3.3 Architecture

Our system's architecture was distributed across both the server and client. The interactions are described in this section. The system design and interactions are illustrated in Figure 3.1, and the complete process flow is detailed in Section 3.4.

3.3.1 Server-side

Node and the ExpressJS [24] framework together which handled routing REST calls acted as the Controller in this Model-View-Controller (MVC) design. The majority of the Model's business logic provided functionality such as compiling code, building and evaluating exams, and formatting/serving data such as the core client facing business logic (minified [27] and sent to the client to execute).

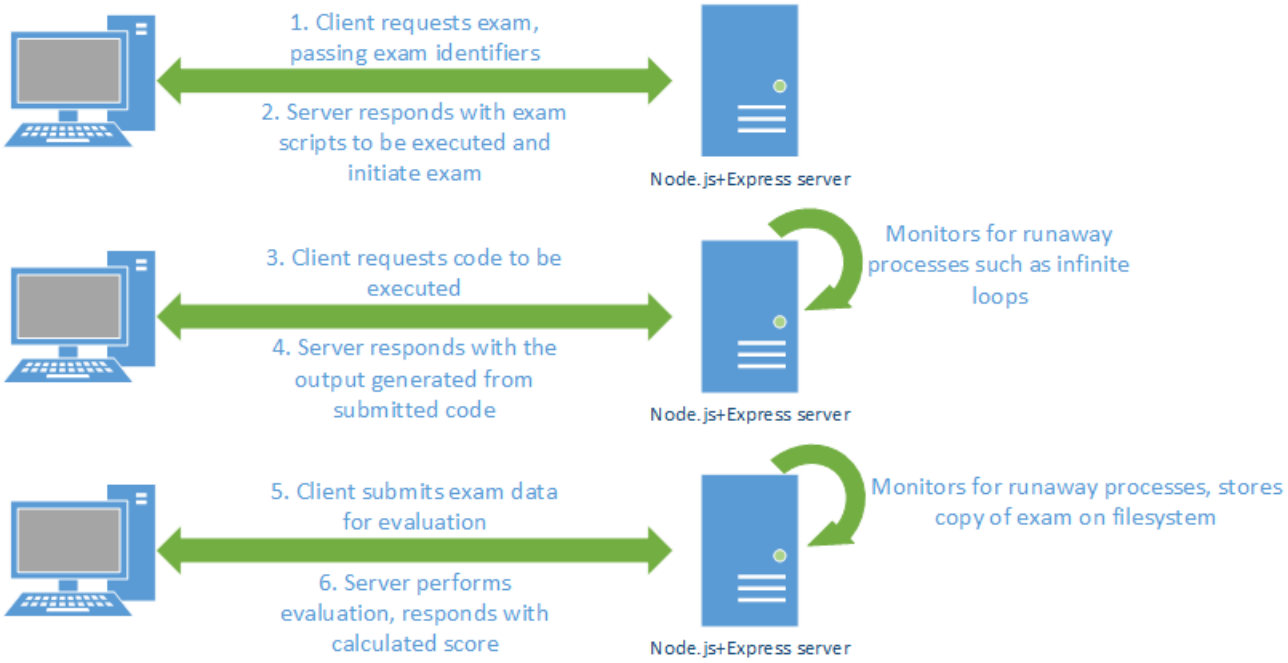

Figure 3.1: A diagram illustrating the high-level process flow.

3.3.2 Client-side

The View component utilized HTML and jQuery to populate the exam and provide all View related functionality. An AJAX request made by the client returns scripts to execute which initiate the rest of the exam. The Controller component processes these AJAX calls throughout the exam period, manipulating the Model and View accordingly. An example of this interaction is a simple compile, where the client requests the server to compile code and the response updates the View.

3.3.3 Datafile Design

We use a simple flat file to generate exam content, following the Javascript Object Notation (JSON) format. The flat data file is uploaded to the server, parsed to form a JSON object, and stored. This JSON object has the following structure:

```
{
  "0": {
      "questionType": "code",
      "language": "c++",
      "problem": "Write a C++ function called print that accepts
         an integer and prints it out.",
      "skeleton": ["#include <iostream>\nusing namespace std;\n\n%
         READ_IN%\nint main() {\n   string x = \"printed\";\n
         print(x);\n}\n", "%READ_IN%", "//Write your code here\
         nvoid print(string x){\n   cout << x;\n}"],
      "input": ["1"],
      "output": ["1"],
      "points": ["15"],
      "difficulty": ".5"
    },
    "prop": {
        "closeDate": "6-15-2018",
        "closeTime": "13:30",
        "allowMultiple": "false",
        "time": "15",
        "warn": ["10", "5"],
        "whitelist": ["0421291", "0421292"],
        "access": "true"
    }
}
```

Data-structure 3.1: Example JSON datafile object structure

Within this structure, "questionType" can hold one of two values, "code" or "mchoice". The value of "questionType" defines whether the question is a coding question or a multiple choice question with randomized answers. I.e., questionType determines what result is produced by a given code snippet. Each field serves the same function for either type of question, excluding the "input" and "output" fields. The "language" field defines the programming language used, "problem" defines the question, "skeleton" defines the initial skeleton code, "points" provides the point value, and "difficulty" defines a difficulty level which translates into minutes (value * 10 = suggested time) used to produce suggested completion times. The "skeleton" field also includes an optional usage which holds a "hidden skeleton", a replace token, and a "visible skeleton" for questions that may want to hide some details from the user, instead of just the standard skeleton. We have shown this optional format in line five of the example data structure above.

For coding questions, the "input" field utilizes an array that holds different test-cases. This input value can represent any input a keyboard can provide in a command-line setting. Each element in the "input" array is a different test case that will be run against student’s submissions. The "output" field represents the expected output produced after running each test case using inputs (if any) from the "input" field.

Multiple choice questions on the other hand, use the "input" field differently. With this question type, the "problem" field describes the overall instructions and the "skeleton" field describes the corresponding code snippet, identical to coding questions (except it is non-editable). However, "input" utilizes a multi-

dimensional array, with each internal array holding a sub-question and an array of options from which students can choose. The "output" field dictates the indices of the correct options.

The structure also contains a required "prop" field which can hold both required and optional information about the activity. The required fields "time" and "warn" are used to control how long an exam will last before automatically submitting. The optional fields "closeDate" and "closeTime" denote when an exam will restrict access, "allowMultiple" defines whether students can take an exam multiple times, and "whitelist" and "access" control access to the exam. Note that the server's internal time-zone may not match your own, which can cause date specific properties to work unexpectedly.

Since our system utilizes the JSON format, code cannot contain line breaks. Instead, line breaks must be replaced with a Line Feed ("\n") which can cause the creation of datafiles to become an arduous effort. To address this issue, skeleton code wrapped with predefined tokens (start: "_«_", end: "_»_") can instead be used with the server automatically converting line breaks to Line Feeds. An example of this format can be seen in Figure B.4.

3.3.4 Administrative Portal

Our system includes an administrative side where both datafiles and test results can be seen. These files are organized by course and exam. For datafiles, authenticated users can upload datafiles, edit them, and remove them. Result files can also be viewed, but not uploaded, edited, or removed. However, exams have the option to generate a report by walking through each of the result files and displaying it in a printable dialog window. These interfaces can be found in

Appendix B as Figures B.5, B.6, B.7.

3.3.5 Testing Security and Integrity

Security is important for any sort of testing module. Since the AJAX call's response calls the script containing the necessary logic for the client, it exists only in that namespace. Because of this namespace, interaction through the console window cannot reach the code or any variables during normal execution. In order to manipulate anything, a malicious user would have to use browser developer tools such as breakpoints, and even then it would not accomplish anything beyond using their allotted time and distort the presentation of questions. No exam solutions ever touch the client side and all submissions are evaluated server-side.

As an added layer of protection, any variables created are local variables and locked using the Object.freeze() [28] Javascript method. This method prevents malicious users from adding new properties to the frozen object; prevents existing properties from being removed; and prevents existing properties, or their enumerability, configurability, or writability, from changing. In essence, the object is effectively immutable. This method is mainly used to lock the answers given from the multiple-choice part of the exam once students begin the programming section in order to keep students from changing their answers in the multiple-choice section.

Regarding server-side code execution, students could attempt to take advantage of this system's architecture and submit malicious code. We decided on using both Unix filesystem privileges as well as process monitoring to help protect the system.

Lastly, we suggest that a proctor be used in addition to these features as we have no control over student collaboration and communication. Similarly, products such as Respondus' LockDown Browser [29] can also provide additional security by way of a custom browser that locks down the testing environment. Together, these layers of security limit the ways in which students were able to collaborate or cheat in these tests.

3.4 System Process Workflow

The system utilizes a Request-Response model, defined as a series of HTTP requests and responses between the client and server. This process is broken up into four main stages:

- Module selection
- Module execution
- Code compilation
- Final evaluation

We have shown these interactions in Figure 3.1.

3.4.1 Module Selection

This first step initializes the system. For an exam or quiz, the client will first post a request to the server's /getModuleSelector endpoint. This endpoint minifies and sends a script that provides the user with a dialog window and some additional logic that allows the user to request the specific quiz or exam. This information includes student ID, course ID, and exam ID.

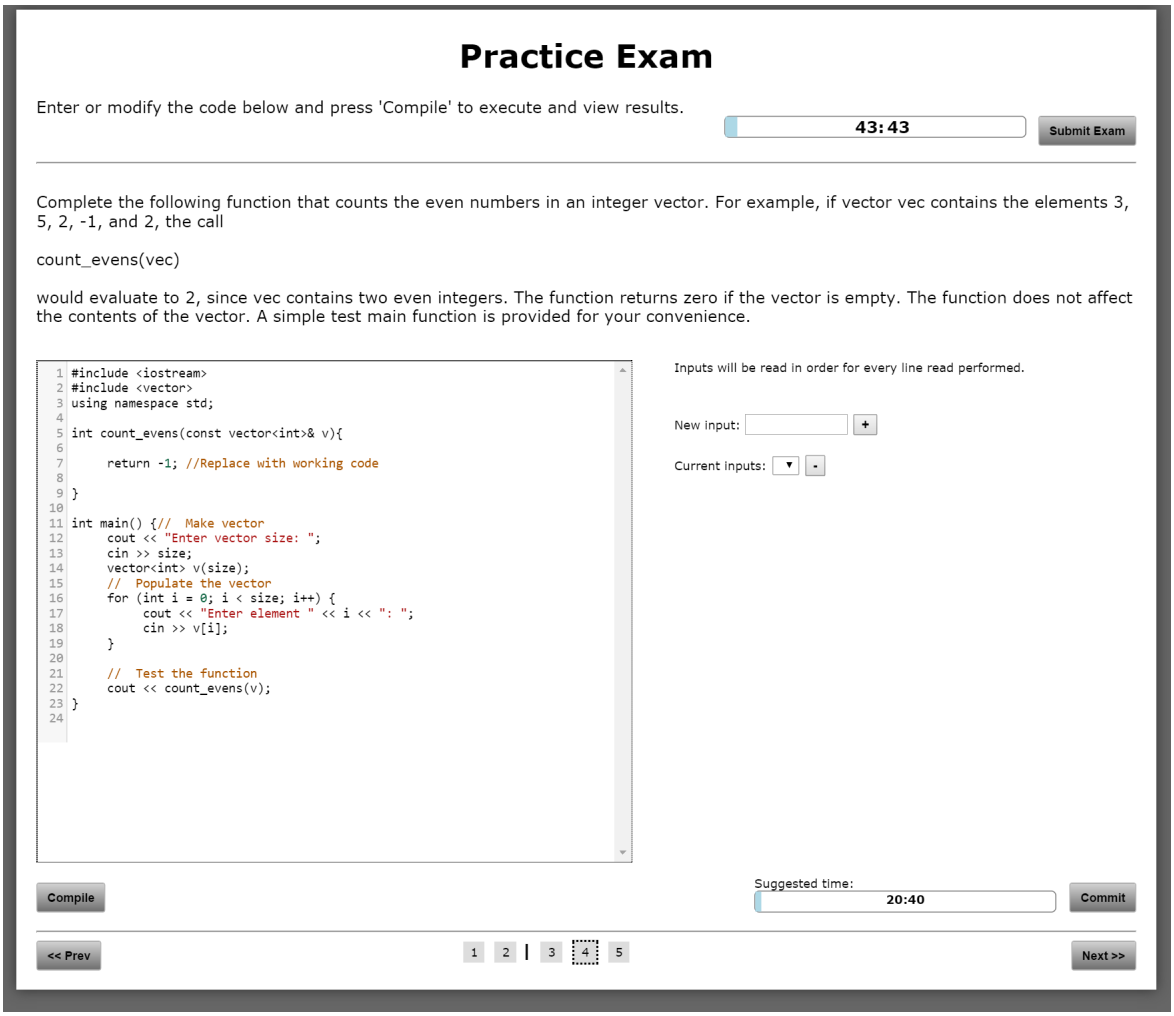

Figure 3.2: A screenshot displaying an exam coding question built from a datafile.

Once submitted, the server parses this information from the requested JSON datafile, inserts those values into Hyper Text Markup Language (HTML) templates, and sends the HTML code along with a minified script that holds the client facing business logic. This templating allows the system to build and serve a module quickly.

3.4.2 Module Execution

At this point in the system's workflow, the client will now have the full code which will execute and append the HTML code to the specified Document Object Model (DOM) element. Once the student enters the exam and selects 'Begin', the exam and timers will start. When the exam's timer elapses, the exam will then automatically submit all available data to the server for evaluation. Figure 3.2 shows an example exam built through this process.

3.4.3 Code Compilation

Once the exam or learning module has started, users can compile and run code. The response will send the language (currently we support C++11 and Python 3.X), code, and any inputs separated by newlines to be read in as standard input. It should be noted that for the quiz and exam modules, compilation can only occur once a student has completed the multiple-choice section. The system will record and freeze all answers from the multiple choice section once any compilation is initiated.

When compiling, the server creates a temporary directory by concatenating the current datetime and a random integer ranging from 0 to 9999. Once the server creates the directory, the code compiles (if necessary) and exectues, piping in the inputs (if any) as standard input. The response returned will hold the generated output or error.

3.4.4 Final Evaluation

This section of execution marks the final part in the system's workflow and only applies to the quiz and exam module. Two conditions can initiate this process:

reaching the exam's time limit or the user clicking the submit button. Once the server responds, a modal window displays the calculated score.

The server processes the submitted exam data and writes the results to a result file (identified by the student's ID). For grading multiple-choice questions, comparing the submitted answers' indices with values from the datafile indicate correctness. Similarly for code questions, comparing the outputs of each testcase to the expected outputs from the datafile determine correctness. For coding problems, all test cases must pass to award points.

3.5 Summary

This chapter described the project solution's architecture, design, and process flow. It detailed the business logic and syntax necessary to understand the overall application and create well-formed datafiles and therefore exams.

Chapter 4

Testing and Evaluation Results

This chapter reviews the software requirements, the evaluation process to test those requirements, and the results of that evaluation. Our evaluation relies on a simple pass/fail process.

We verified component requirements through user testing, integration testing and visual inspections of the system and show a check sheet that lists the requirements completion status. Figure 4.1 sums up the results of the testing performed.

4.1 Testing

Our testing approach consisted of both manual inspections and automated test cases via Selenium [30]. Of the more popular browsers, we selected Chrome [31], Safari [32], and Firefox [33] as our main platforms to test. With Selenium's ability to automate browser interactions, we were able to consistently verify both specific components and end to end processes successfully. These interactions included resource loading, error reporting, and code compilation.

We were able to consistently verify that modules were created and served as expected using Selenium to automate interacting with the activity. As Selenium walked through different sections of the test activity by simulating keystrokes and clicks, the question and answer values were verified as visible and error reporting was displayed when expected. However this only proved the expected elements and values existed in the DOM and were correct, therefore manual visual inspections were utilized to further confirm the appearance and validity of the generated modules.

With regards to compilation, Selenium was also used to test the functionality and correctness. With this approach, we were able to automatically select specific questions, enter predefined code and submit for execution. Results returned were then read and compared to expected results for verification. Compilation was also tested via HTTP requests posted directly to the compile endpoint in order to test that functionality directly.

Full integration testing was used in order to verify exam evaluation and reporting. This testing again utilized Selenium's automation to simulate the taking of an exam and comparing the final score received to the expected score. Observation of these processes assisted in verifying the results.

Coordinating with Dr. Halterman, Southern Adventist University's CPTR-124 students also used the system for both practice and actual exams. Through this user testing, the system evolved based on feedback and performance observations.

The following summary describes the testing results and feedback received:

- Practice exams: The feedback received was positive enough to warrant testing the system in an actual exam. No problems encountered.

- Actual exams: The feedback received was positive and very useful as users' experience revealed a few bugs such as runaway processes, and quality of life adjustments such as warnings and extra datafile customization.

- ✓ Modules are served correctly and consistently across platforms.
- ✓ Questions, sub-questions, and answers are all generated and displayed correctly.
- ✓ Server receives, compiles, and runs submitted code and reports results correctly.
- ✓ Submissions are received and results are reported successfully.
- ✓ All question specific test-cases run and evaluate successfully.
- ✓ Exam results are immediately reported to user and stored successfully.

Figure 4.1: Evaluation checklist.

4.2 Evaluation

User and integration testing and feedback paired with visual inspections show that our solution meets the requirements and goals set for this project. From creating the module datafile to grading submissions and waiting for test results, time spent was noticeably reduced. After implementing user testing feedback, performance evaluation showed that the system was able to handle loads with a negligable performance hit.

During the first testing session, an interaction between Node's child_process.exec timeout functionality and infinite loops caused timeouts to fail and the CPU us-

age to rise to 99% as these programs kept running. After the cause for these run-away processes was found, we implemented a monitoring solution that kept these programs from continuing infinitely. The peak performance impact after the fix was implemented is shown in Figure 4.2.

4.3 Final Deliverables

Once the project was completed, we delivered the following materials to the client:

- Application source code
- Access to project source control
- Documentation on setup, installation, and datafile formatting
- Final project report

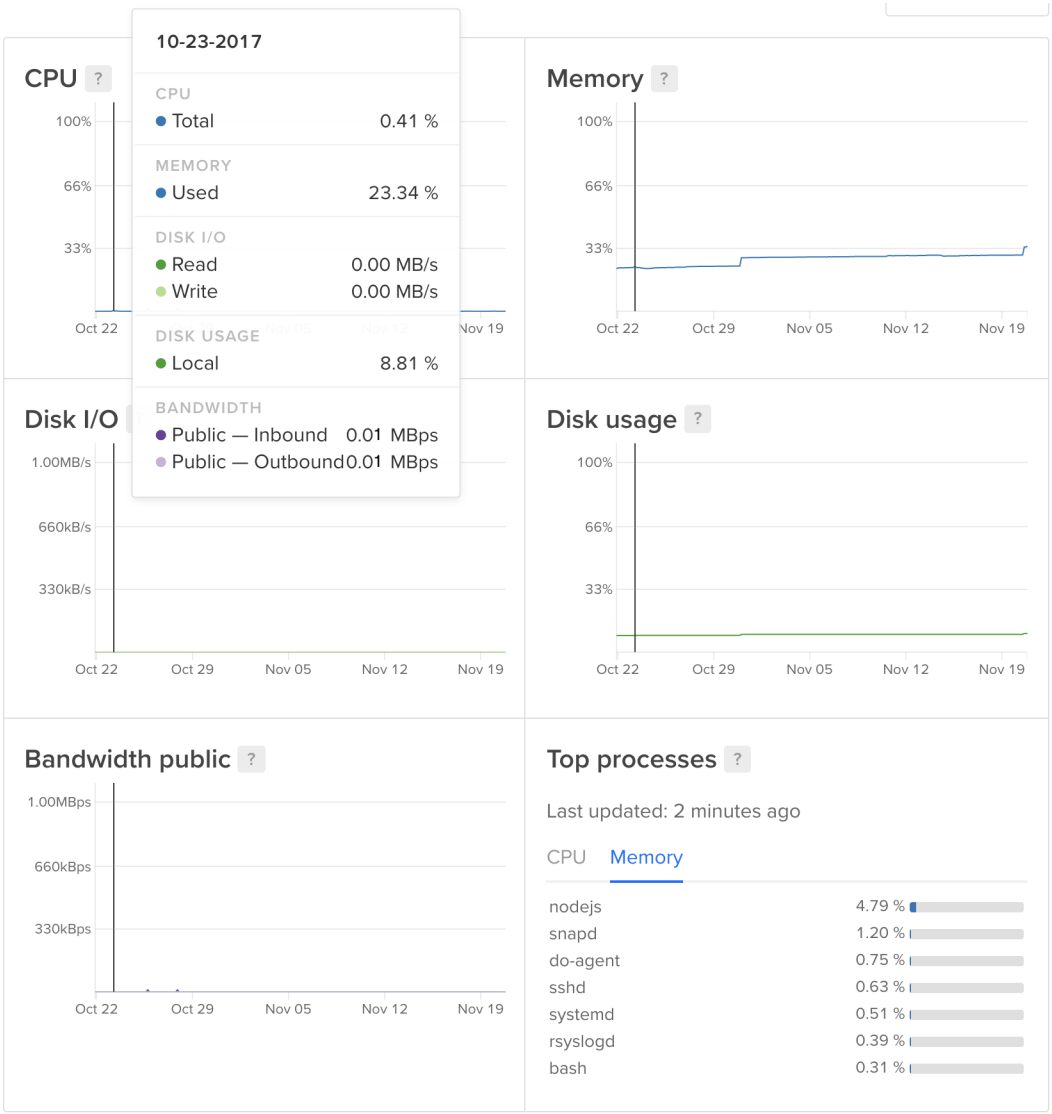

Figure 4.2: Graphs detailing the peak performance impact during user testing.

Chapter 5

Conclusion

Our client wanted a well-structured and implemented framework for writing and running code in both testing and learning environments. The current solutions reviewed in Table 2.2 cover most of the goals of this project in some form, however we did not find one that met all of the goals according to our client's specifications stated in Chapter 1.

In this thesis, we implemented a separate framework to remedy the above issues. Based on our research and testing results, our solution will help both students and teachers in many aspects of programming examinations as described in Chapter 1. The opportunity to type and format code, along with the ability to compile and execute code is helpful to students and allows them to focus on actual test content and non-trivial syntax or formatting issues. This solution also can help to dramatically cut down on the time it takes to build and grade exams due to the simple flat file format and automated grading. Therefore this solution will be of great benefit to the student as well as the professor, saving them both time and effort.

For future work we suggest adding support for other learning environment

modules such as online textbooks, and a more intuitive way of adding support for new languages. We also suggest further research into work done by Singh et al. [34] in order to create and provide more meaningful feedback such as potential corrections to failed test cases. In regards to furthering security, we suggest an approach similar to the algorithm created by Chen et al. [35], which used sandboxing and regular expressions to filter out malicious code.

Appendices

Appendix A

Environment Requirements

This project was developed on an Ubuntu 16.04.2 x64 Linux server. The following packages must be installed:

- Node (v8.9.1)
- Python (v3.5.2)
- g++-5
- Forever (Optional Node Package Manager (npm) package, used for ensuring that a given script runs continuously)

Appendix B

System Screenshots

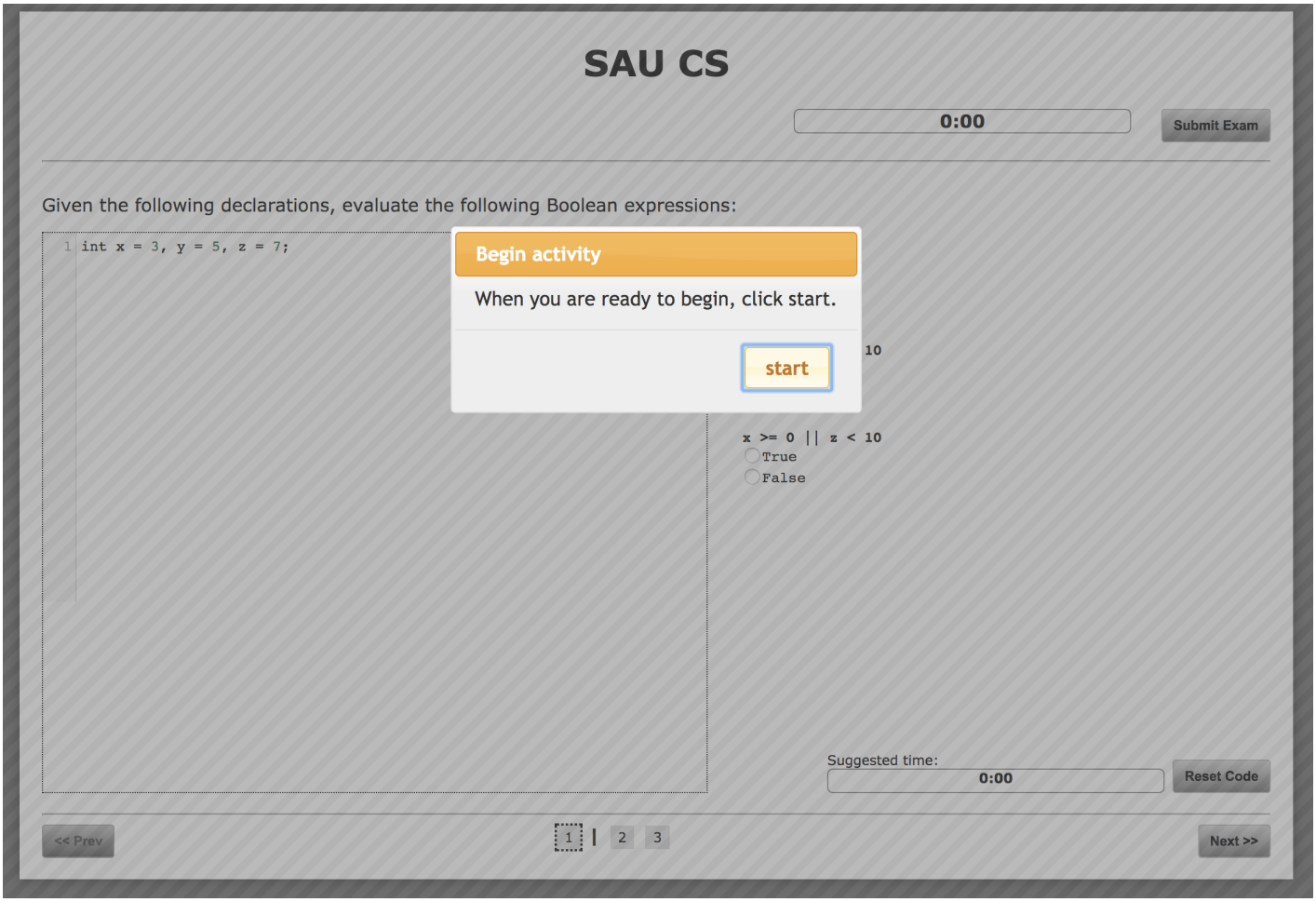

Figure B.1: The start screen of an exam.

Figure B.2: After using the compiler functionality, answers selected in the previous multiple-choice section cannot be changed.

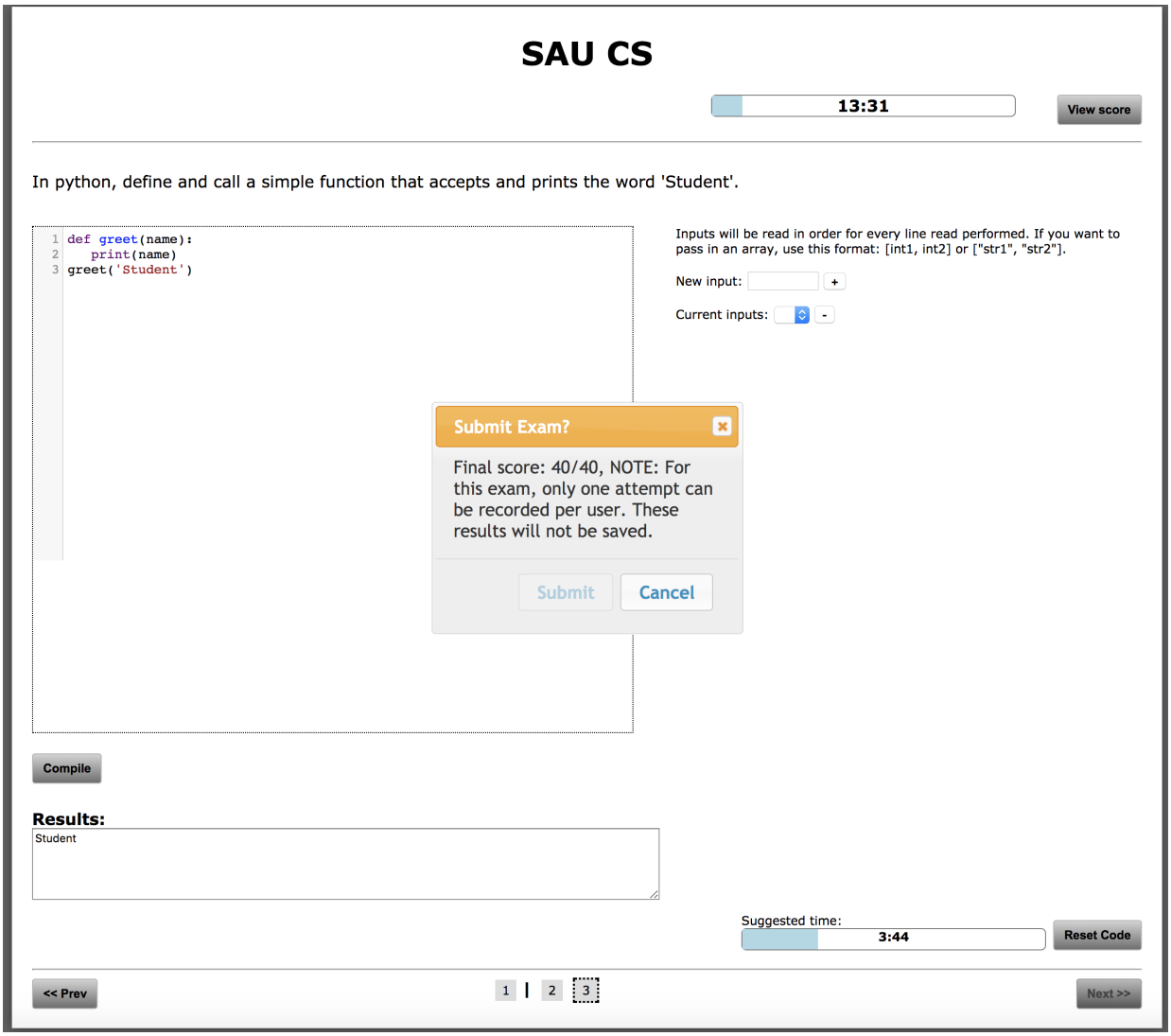

Figure B.3: Exam results reported back. In this case, the datafile specified not to allow multiple attempts, but did allow access. This is reported to the user in case of misunderstanding.

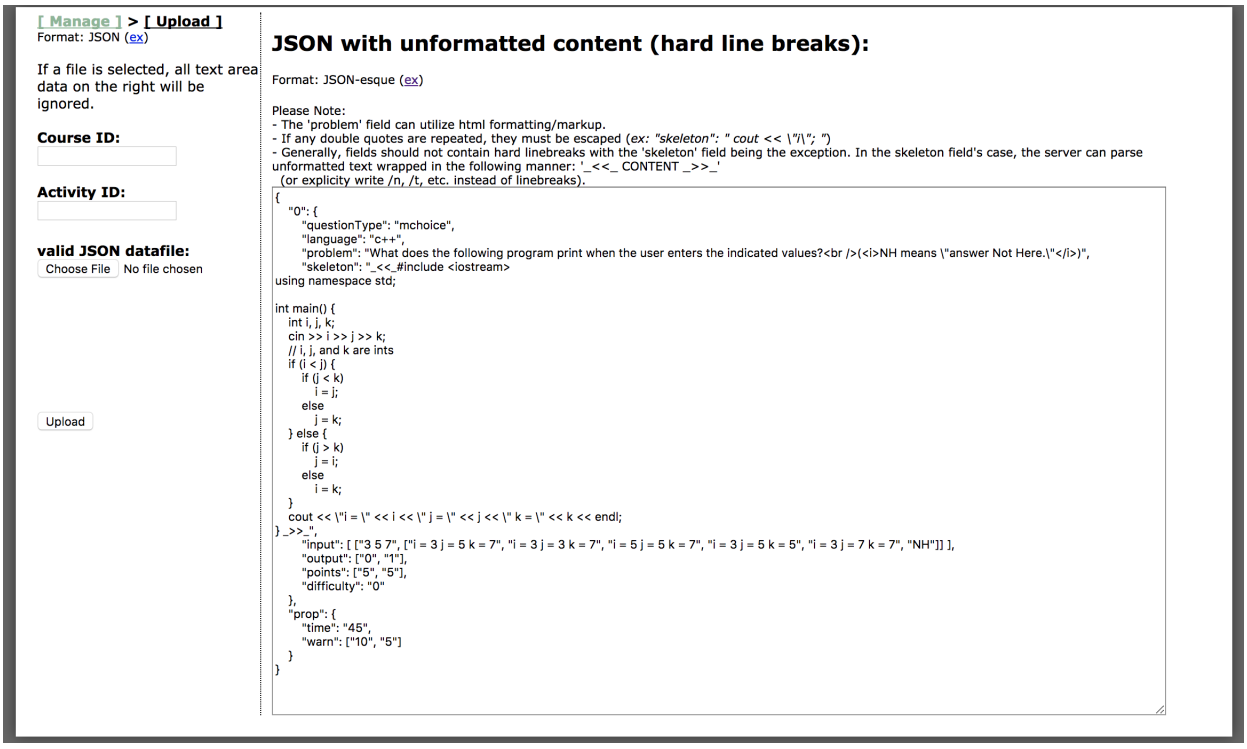

Figure B.4: The upload interface, showing the alternate datafile format which supports newlines.

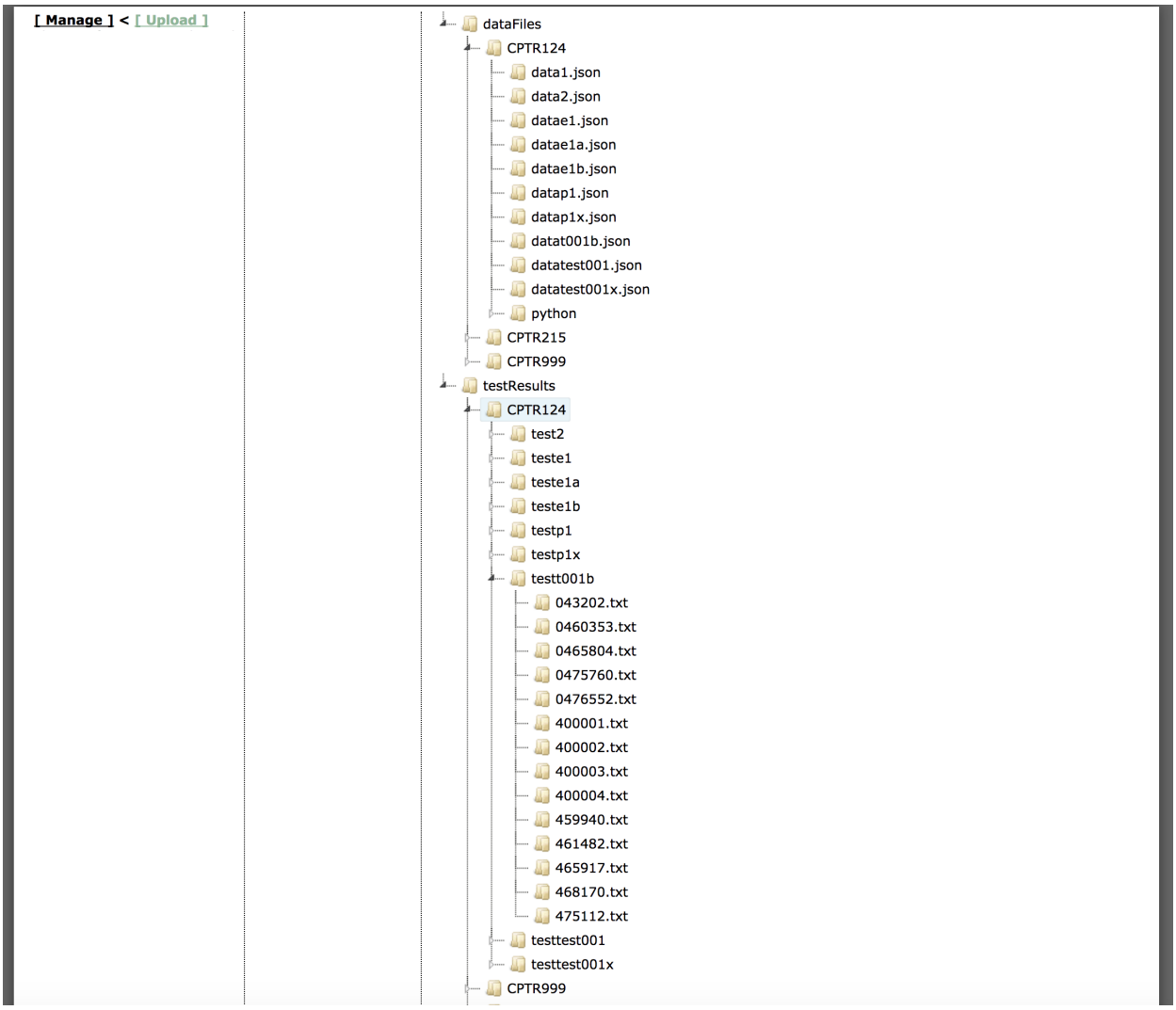

Figure B.5: A tree representing the file structure of both the exam datafiles and results, sorted by class and exam.

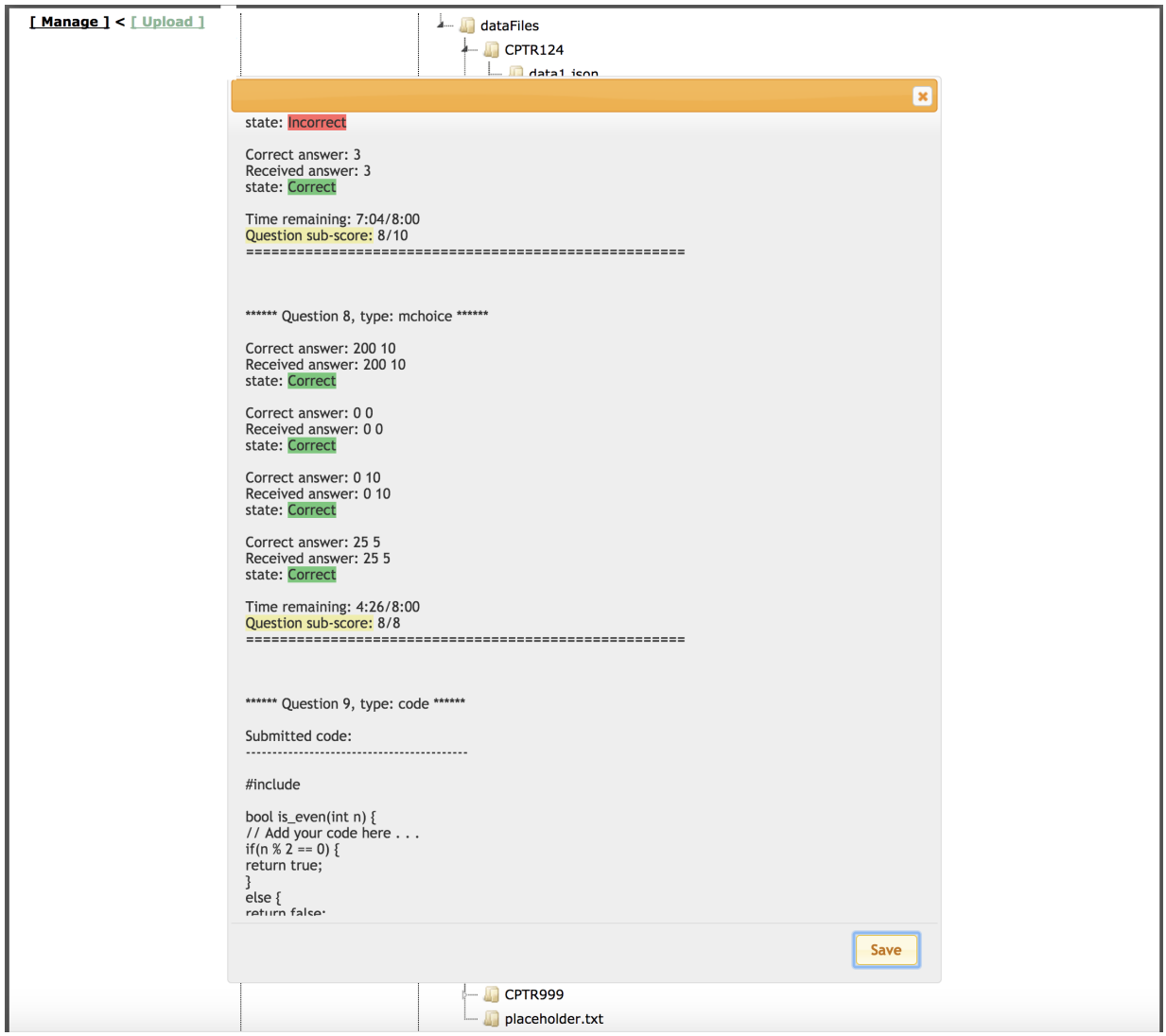

Figure B.6: A styled report of all submitted results for the selected exam.

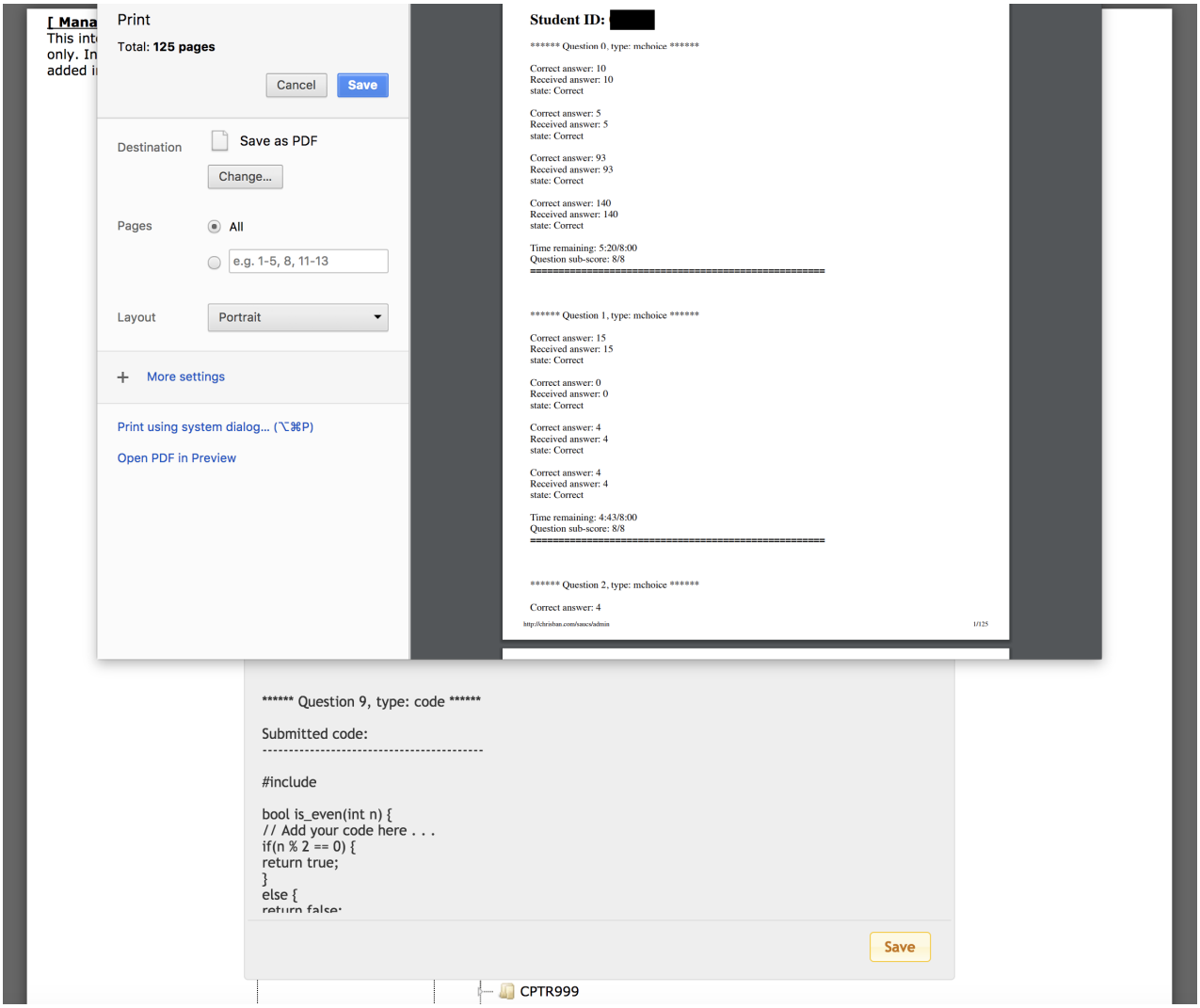

Figure B.7: A pintable exam report of all submitted results for the selected exam.

Bibliography

[1] D. Thiébaut, “Automatic evaluation of computer programs using moodle’s virtual programming lab (vpl) plug-in,” *J. Comput. Sci. Coll.*, vol. 30, no. 6, pp. 145–151, June 2015. [Online]. Available: http://dl.acm.org/citation.cfm?id=2753024.2753053

[2] A. Patil, “Automatic Grading of Programming Assignments,” Master’s thesis, San Jose State University, 2010. [Online]. Available: http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1050&context=etd_projects

[3] S. Edwards. (2006, April) The web-cat community: Resources for automated grading and testing. [Online]. Available: http://web-cat.org/group/web-cat

[4] B. Cheang, A. Kurnia, A. Lim, and W.-C. Oon, “On automated grading of programming assignments in an academic institution,” *Computers & Education*, vol. 41, no. 2, pp. 121–131, 2003.

[5] C. A. Higgins, G. Gray, P. Symeonidis, and A. Tsintsifas, “Automated assessment and experiences of teaching programming,” *J. Educ. Resour. Comput.*, vol. 5, no. 3, 2005. [Online]. Available: http://doi.acm.org/10.1145/1163405.1163410

[6] M. Dougiamas and P. Taylor, “Moodle: Using learning communities to create an open source course management system,” 2003.

[7] M. Dougiamas and P. C. Taylor, “Interpretive analysis of an internet-based course constructed using a new courseware tool called moodle,” in *2nd Conference of HERDSA (The Higher Education Research and Development Society of Australasia)*, 2002, pp. 7–10.

[8] Moodle. (2015) The moodle project. [Online]. Available: https://moodle.org/

[9] Edmodo. (2015) Edmodo: Connect with students and parents online. [Online]. Available: https://www.edmodo.com/

[10] K. Balasubramanian, V. Jaykumar, and L. N. Fukey, “A study on âĂIJstudent preference towards the use of edmodo as a learning platform to create responsible learning environmentâĂİ,” *Procedia-Social and Behavioral Sciences*, vol. 144, pp. 416–422, 2014.

[11] S.-S. Liaw, H.-M. Huang, Y.-T. A. Liaw, and Y.-H. A. Liaw, “Exploring learners attitudes toward a social e-learning system: A case study of the edmodo,” in *EdMedia: World Conference on Educational Media and Technology*. Association for the Advancement of Computing in Education (AACE), 2016, pp. 764–769.

[12] Moodle. (2015) Moodle: Installing add-ons. [Online]. Available: https://docs.moodle.org/25/en/Installing_add-ons

[13] K. M. Ala-Mutka, “A survey of automated assessment approaches for programming assignments,” *Computer science education*, vol. 15, no. 2, pp. 83–102, 2005.

[14] Capterra. (2017, November) Top lms software. Capterra Inc. [Online]. Available: https://www.capterra.com/learning-management-system-software/#infographic

[15] Moodle. (2015) Moodle: Question import formats. [Online]. Available: https://docs.moodle.org/25/en/Installing_add-ons

[16] Moodle. (2015) Moodle: Question types. [Online]. Available: https://docs.moodle.org/24/en/Question_types

[17] Edmodo. (2017) Edmodo: Quiz limitations. [Online]. Available: https://support.edmodo.com/hc/en-us/articles/205004854-Quiz-Limitations

[18] S. H. Edwards, “Work-in-progress: Program grading and feedback generation with web-cat,” in *Proceedings of the First ACM Conference on Learning @ Scale Conference*, ser. L@S ’14. New York, NY, USA: ACM, 2014, pp. 215–216. [Online]. Available: http://doi.acm.org/10.1145/2556325.2567888

[19] ——, “Teaching software testing: automatic grading meets test-first coding,” in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM, 2003, pp. 318–319.

[20] D. Arnow and O. Barshay. Turing’s craft. [Online]. Available: http://www.turingscraft.com/codelabs.php

[21] E. Foxley, C. Higgins, and A. Tsintsifas, “The ceilidh system: a general overview,” in *Second Annual Computer Assisted Assessment Conf*, 1996.

[22] S. Edwards. (2006, April) What is web-cat. [Online]. Available: http://wiki.web-cat.org/WCWiki/WhatIsWebCat

[23] D. Arnow and O. Barshay, “Webtoteach: An interactive focused programming exercise system,” in *Frontiers in Education Conference, 1999. FIE’99. 29th Annual*, vol. 1. IEEE, 1999, pp. 12A9–39.

[24] ExpressJS. (2017) Express - node.js web application framework. [Online]. Available: https://expressjs.com/

[25] SPDX. (2015) Mit license. [Online]. Available: https://spdx.org/licenses/MIT.html

[26] SPDX. (2015) Bsd 2-clause licence. [Online]. Available: https://spdx.org/licenses/BSD-2-Clause.html

[27] R. Anderson. (2012) Bundling and minification. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/performance/bundling-and-minification#minification

[28] M. web docs. (2018) The object.freeze() specification and documentation. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/freeze

[29] Respondus. (2015) Respondus lockdown browser. [Online]. Available: https://www.respondus.com/products/lockdown-browser/

[30] SeleniumHQ. (2017) Selenium - web browser automation. [Online]. Available: http://seleniumhq.org/

[31] Google. (2012) Chrome: One fast, simple, and secure browser for all your devices. [Online]. Available: https://www.google.com/chrome/browser/desktop/index.html

[32] Apple. (2018) Safari. the best way to see the sites. [Online]. Available: https://www.apple.com/safari/

[33] Mozilla. (2012) The new firefox. [Online]. Available: https://www.mozilla.org/en-US/firefox/

[34] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated feedback generation for introductory programming assignments," *ACM SIGPLAN Notices*, vol. 48, no. 6, pp. 15–26, 2013.

[35] M.-Y. Chen, J.-D. Wei, J.-H. Huang, and D. T. Lee, "Design and applications of an algorithm benchmark system in a computational problem solving environment," in *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ser. ITICSE '06. New York, NY, USA: ACM, 2006, pp. 123–127. [Online]. Available: http://doi.acm.org/10.1145/1140124.1140159