DES2

ZINC – Online Assignment Submission & Automatic Grader System

FYP Final Report

By
CHEUNG, Daniel
CHUN, Hiu Sang
MAK, Ching Hang
WONG, Yuk Chun

DES2

Advised by
Dr. TSOI, Desmond Yau-Chat

Submitted in partial fulfillment
of the requirements for COMP4981/CPEG4901
in the
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
2019-2020
Date of submission: April 29, 2020

ABSTRACT

Facing the possible future HKUST Common Core program reform, the HKUST Computer Science and Engineering (CSE) department may hold a larger course involving programming. A way to automate the grading process is therefore needed and to fulfill this need, a tool would need to be produced such that courses with programming assignments and labs may use to increase teaching efficiency and may be used to provide instant feedback to students. This tool may also be paired with virtual classrooms as well as for general Massive Online Open Course (MOOC) services.

This project aims to develop such a system: an online programming assignment submission system and an automatic grading system, with a design exhibiting modularity, flexibility, and extensibility to facilitate the deployment, usage, and future development with ease.

The system is mainly developed using PHP, JavaScript, and Kotlin. In the research period, we have explored prior related solutions, such as Submitty, Auto-Grader, and Canvas. Throughout the project, we have implemented our automatic grading solutions using pipeline and containerization designs. The Submission Subsystem is a web interface designed for submitting programming assignments, reviewing grading reports, and managing the system in the administrative panel.

The system has passed preliminary tests with examples from previous assignments. A User Acceptance Test has been conducted.

We have also submitted ZINC to the HKUST President's Cup 2020 and were one of the 6 finalists.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Overview

Future reform in the Hong Kong University of Science and Technology (HKUST) Common Core curriculum would likely have the Department of Computer Science and Engineering (CSE) to host a larger Common Core course involving programming. Grading assignments manually in this course would be impractical, thus requiring the need for an automatic grading system. This itself is only a trigger to the more general problem of inefficient grading practices seen in some courses longing for a fix.

Although from a survey, some Teaching Assistants (TAs) do create their custom graders to automate grading in some ways, requiring all to implement custom graders is deemed to be unrealistic and consumes precious teaching resources when already scarce.

On the other hand, there is currently an understaffing of TAs, so one needs to oversee a large number of students in a tutorial or lab section. To ensure that students can still receive adequate support, many courses opt to limit student intake. The recent increase of interest in Computer Science among HKUST undergraduate students only intensifies the situation, further increasing the pressure on teaching quality. An automatic grader could help offset such overheads, as TAs will be able to divert time away from checking the work of students to helping those in need.

The CSE department currently offers the Course Assignment Submission System (CASS) [1], which allows students to submit assignments for COMP courses from various sources, and check submission history with some basic information. However, the current infrastructure of the system prevents the ability to attach an automatic grading module to the CASS. Although there are also existing open source and commercial solutions, they, unfortunately, have various shortcomings, which will be analyzed in the Literature Survey section.

This project presents ZINC (ZINC is not CASS), an online assignment submission system and automatic grader system, to solve the grading problem, and to cope with the new requirements and improve on the current design on the submission system based on feedback from the teaching staff and students of HKUST. Also, the system is designed with load balancing in mind to ensure that the system can handle a large number of connected clients near the deadline.

The project is split into two subsystems: the assignment submission system and the assignment automatic grading system, which we will be referring to as Submission Subsystem and Auto-Grader Subsystem thereon. We will be mainly using PHP and Node.js on the Submission side and Kotlin on the Auto-Grader side.

Division of Labor with Full-Time Staff Members

Since the scope of this project is substantial and the department sees potential into adopting the project into a production environment, our work is shared with one full-time and one part-time staff members, Kristopher and Chris in a bigger team named Online Assessment Platform (OAP), to hopefully complete the working project in one year.

1.2 Objectives

1.2.1 User Requirements

System Flow

The services delivered by the platform are typically used in this chronological order.

1. The system synchronizes student enrollment records with the department registry.
2. Teaching staff designs and manages assignments and related resources on our administrative panel.
3. Students submit assignments using the student user interface in their browser.
4. The automatic grader grades the submitted code in an isolated environment.
5. Scores and program execution results are composed into a readable report and saved.
6. Students and teaching staff receive email notifications on changes and generated reports and can view them on the user interface.

To fully quantify measurable and achievable objectives, we transform the interpretations of our system flow into a list of user requirements. The system is built for three various roles, and their requirements are listed below.

Requirements for Student Role

1. Log in to the system in a Student role.
2. Select from a list of enrolled courses that are using the system.
3. Select from a list of released assignments of the selected course.
4. Submit files or directory archives for the selected assignment.
5. View timelines of events regarding the assignment:
   - Submission history and status,
   - Assignment updates.
6. Receive emails regarding assignment updates.

Requirements for Teaching Staff Role

1. Log in to the administrative system in a Teaching Staff role.
2. Perform CRUD (Create, Read, Update, Delete) operations on assignments for managing courses of the user.
3. Perform CRUD operations on and generate test cases under assignments.
4. Perform CRUD operations on grading policy/strategy on assignments using GUI or script.

5. View all submitted source code of all versions.
6. Write remarks onto submissions.
7. Manual actions to re-run tests or alter score: (lenient adjustments/plagiarism).
8. Receive emails regarding updates.
9. Send platform emails to students under managing courses: (reminder/clarification).

Requirements for Administrator Role

1. Extend rights on top of the Teaching Staff role.
2. Perform CRUD operations on users, permissions, and roles.
3. Update associated courses of students and teaching staff.
4. Send platform emails to selected users.

1.2.2 User Acceptance Test

One or more user acceptance tests should be conducted to ensure the smoothness in product delivery.

1.3 Literature Survey

The literature survey covers existing submission and/or grading (automatic or not) systems available in HKUST or publicly, and several research papers on the topic of auto-grading programming assignments.

1.3.1 Existing Systems

The existing systems we studied are CASS of HKUST Department of Computer Science CS System [1], Submitty [2], Canvas [3], Autogradr [4]. Details of each system are shown in Literature Review Details on Existing Systems under Appendices. Common capabilities present in these four systems include:

- Manage users and enrollment: Synchronize or import from school systems
- Manage (CRUD) assignments
- Submit and resubmit assignments
- Download submitted assignment
- Keep submission history
- View submission statuses
- Retrieve submissions: via FTP or downloading

Their differences are listed in the table below, with our system ZINC in the last column to be compared against.

Feature / System	*CASS*	*Submitty*	*Canvas*	*Autogradr*	*ZINC*
Distribute course and assignment materials	🟥	🟩	🟩	🟥	🟩

Feature / System	*CASS*	*Submitty*	*Canvas*	*Autogradr*	*ZINC*
Control roles and permissions	🟥	🟩 By Administrator (Admin)	🟩 By Admin	🟥	🟩 Synchronized with the School system, and can be changed by Admin
List all assignment deadlines for a student	🟥	🟩	🟩	🟩	🟩
Control visibilities of assignments and grades	🟥	🟩	🟩	🟥	🟩
Send / Receive platform notifications	🟥	🟩	🟩	🟥	🟩
Send email notifications	🟩	🟩	🟩	🟥	🟩
Submit by various methods	🟩 Dropbox/ Google Drive/ OneDrive/ Upload	🟩 Upload/ Version Control System (VCS)	🟩 Text/ URL/ Google Drive/ Office 365/ Upload	🟥	🟩 Cloud storages/ Upload/ VCS (Potential feature)
Set late policy	🟩 Late days	🟩 Late days	🟥 Will just be marked late	🟩	🟩 Late days
Edit submission online	🟥	🟥	🟥	🟩 With IDE	🟨 Potential feature
Provides grouping within course for submitting and grading in groups	🟥	🟩	🟩	🟩	🟨 Potential feature
Keep past courses	🟥	🟩	🟩	🟩	🟩
Release assignment grades	🟥	🟩	🟩	🟩	🟩
Auto-grade in various languages and environments	🟥	🟩 Based on GNU/Linux, including network and database	🟥	🟩 Chosen in the technology stack including web, database	🟩

Feature / System	*CASS*	*Submitty*	*Canvas*	*Autogradr*	*ZINC*
Auto-grade with various methods	🟥	Including test cases, static analysis, code coverage, memory debugging	🟥	Mainly test cases with output matching and frameworks, and custom script	🟩
Show immediate feedback	🟥	🟩	🟥	🟩	Customizable in grading details
Show detailed grading results	🟥	🟩	🟥	🟩	🟩
Compute assignment scores and statistics	🟥	🟩	🟩	🟥	🟩
Check submission for file corruption	🟩	🟥	🟥	🟥	🟩
Handle deadline extension requests	🟥	Students can request, and instructor approve	🟥	🟥	Potential feature
Export grades as CSV	🟥	🟩	🟩	🟥	🟩
Comment on assignment by student and instructor	🟥	🟩	🟩	🟥	🟩
Allow appealing of assignments	🟥	The instructor can enable and set inquiry due date	No, but provides assignment-specific discussion thread	🟥	Potential feature
Allow grade overrides	🟥	🟩	As a regrade	🟥	🟩
Handle class exercises	🟥	🟩	As quizzes	No, but may be created as an assignment	As immediate grading feedback
Isolate grading environments	🟥	Work in progress	🟧	🟥	🟩
Extend grading methods	🟥	🟥	🟥	🟥	Teaching Staff can create new grading modules

Feature / System	*CASS*	*Submitty*	*Canvas*	*Autogradr*	*ZINC*
Provide load-balancing for high request demands	Unknown	🟥	🟥	🟥	🟩
Grade incomplete submissions	🟥	🟥	🟥	🟥	Questions in an assignment can be graded separately

Table 1 Comparison Table across existing systems and ZINC (Color Codes: Absent, Present, Other explanations)

From Table 1, one can observe that CASS and Canvas may be grouped as systems with basic submission functionalities, while Submitty and Autogradr may be grouped as systems with certain automated grading mechanisms built into them. Even so, such a dichotomy is also not too accurate, as Submitty and Canvas tend to share many features as a platform for students and teaching staff to perform course-level management tasks like releasing grades. In this light, containing most of the required features, Submitty emerges as the most suitable platform to possibly be adopted as a new system for our department.

Despite this, some key features missing from all the four systems are critical for the new online assessment platform. Submitty does not have native support for the secure execution of grading with the help of techniques like containerization. The ways of which Submitty is scalable are less configurable from the standpoint of an adopter, who is likely to have custom needs. Furthermore, the degree of customization and modulization are lacking in Submitty: though allowing to run on new modules, the modules combine in ways not apparent to a user.

ZINC inherits most of the features offered by the four systems and on top of which provides firstly a framework to scale the system using clusters configurable e.g. submission servers, and secondly a paradigm to combine and run customizable grading methods, i.e. a grading pipeline, which readily accepts new grading modules. This shows that ZINC is not simply an integration solution among the existing systems; it has designs that improve on them.

On the other hand, systems like Canvas are general-purpose and therefore are not focused on programming assignments. ZINC occupies a position for grading programming assignments and is equipped with the flexibility so that Teaching Staff is not forced to settle on specific grading techniques, and instead can use modules to enable their requirements.

1.3.2 Relevant Research Work

Research on automated grading of programming assignments explores a popular question: how we could do better than test case-based grading. Test cases are the standard for current practice (see Preliminary User Study on page 18), but costs labor-intensive design and specifications, and does not intrinsically shows correctness – it only captures falsehood in programs.

Automated test case generation may alleviate the problem and is indeed one future extension of our grading methods.

For techniques that are fundamentally different, Liu et al. [5] use formal semantics to free themselves from having to use test cases: by accepting a sample correct program, this technique compares execution paths between the sample and submission on some program input, to quantify the ways the execution paths are different between the programs semantically; it finally searches for inputs that may give rise to such a path derivation to show that a submission wrong. However, this method is not stable, and not trusted by teaching staff in general, who could not risk having grading mistakes. Due to the extensible design of our auto-grader, we could include this as a module, but for the moment we intend to employ test cases as the primary grading technique.

2. METHODOLOGY

2.1 Requirements and Design

We uphold several important principles in this project, including the following:

- Modularity
- Flexibility
- Extensibility

These basic software development principles are extremely important in allowing the project to be sustainable after deployment. And for such, we employ the Agile approach in software development to ensure the quality and stability of the code.

2.1.1 Design

Preliminary User Study

Interview with Teaching Associates

Upon presenting our proposed system to two CSE full-time teaching associates, who are mainly responsible for the grading work in COMP2011 and COMP2012, we learned about their existing grading workflow and preferences.

Because of the relatively high student intake every semester, COMP2011 and COMP2012 require automation in grading programming assignments. Currently, the TAs developed their grading pipeline (implemented in Bash/Makefile and Python respectively), which follows these steps:

1. Prepare the submitted .cpp files.
2. Replace the provided main function with one(s) containing test cases. If separate compilation is used, supply one or more sample main.cpp, each containing a main function; otherwise, replace the code directly.
3. Compile the source file(s) to executables.
4. If required, feed executables with test case inputs in the command line as arguments.
5. Check the return values of the tested functions against test case expected outputs in standard output.
6. Optionally check for memory leak, look up states of variables, replace some function implementations with correct implementations, and handle errors.
7. Compute scores of submissions based on the number of passed/failed test cases and other results.

Our grading pipeline, which will be elaborated in 2.3.1 Design in Auto-Grader Subsystem, is compatible with this workflow because these steps can be organized in a series of stages.

We have also found out that test case-based grading is indeed the dominant way of grading, and the TAs find designing test case inputs and outputs acceptable, though there can be more convenient ways for specifying them.

The current grading systems also standardize the grading procedures, as the TAs tend not to read individual code submissions to be fair.

Some known issues and challenges in grading programming assignments include:

- Students leaving debug messages in the code, resulting in extra, hence wrong output.
- Unwanted whitespaces in the code can cause programs to fail the automated grading.
- Integration testing for various parts of an assignment can be tricky since TAs need to specify which parts of the submitted code should be used to replace the sample solution for testing, similar to unit testing.
- Code with undefined behaviors.
- Submitted code should be run in sandboxed environments for security.

Some of these requirements are directly addressed in this system, while others can be added by extending the system with grading stages.

Interview with Instructor

As pointed out by the project supervisor, who himself is an instructor of several computer science courses, there are a few desirable and flexible features that would enhance his current grading practice and the learning experience of students:

- Immediate feedback of student submissions. Students can have the system perform sanity checks for them in order not to lose points due to carelessness.
- Partial grading of a submission. When students do not understand how to approach more difficult problems of an assignment, they often skip the sections and only complete portions they understand. Many existing auto-graders assume that all parts of the assignment are completed, causing the system to return a score of zero, hindering the student's ability to check whether their completed portions are correct. These students should still get grading results on the parts that they attempted. This can also be used as early submission feedback.

Survey for Instructors and TAs

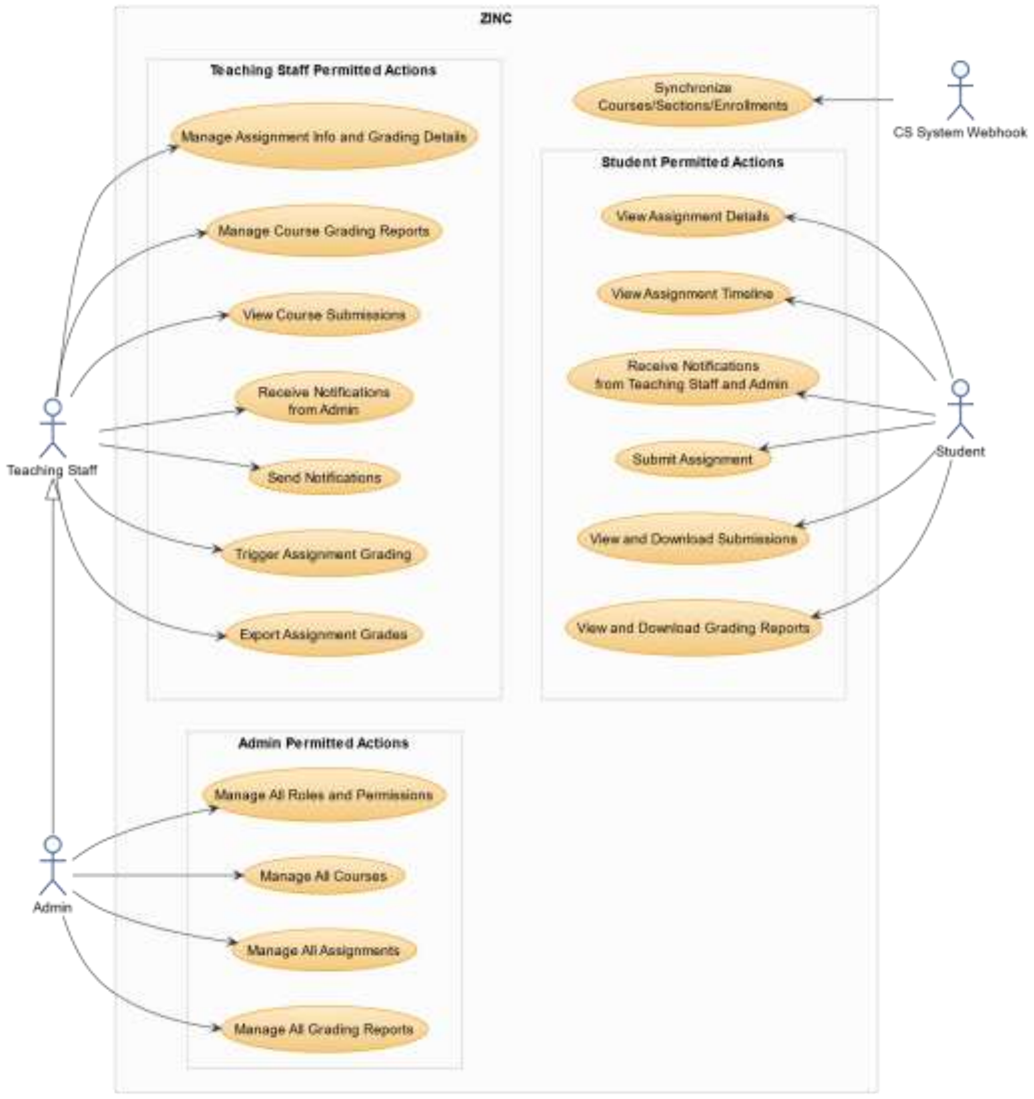
To verify the assumptions of the user requirements of this project, we designed a survey to ask for opinions on the features we will be providing. Questions include what the current practice of grading programming assignments is in courses that the surveyees teach regularly, and how helpful or wanted are features like immediate grading, sanity check, and test case editors. For the findings, refer to Survey Findings. For the questionnaire questions and responses, refer to the Questionnaire in the Appendices.

Functional Requirements

The requirements for users in this system are collected, documented, and explained below.

Use-Case Diagram

We have four actors in this system, including Student, Teaching Staff, Admin, and CS System Webhook.



ZINC
Teaching Staff Permitted Actions
Manage Assignment Info and Grading Details
Manage Course Grading Reports
View Course Submissions
Receive Notifications from Admin
Send Notifications
Trigger Assignment Grading
Export Assignment Grades
Teaching Staff
Synchronize Courses/Sections/Enrollments
CS System Webhook
Student Permitted Actions
View Assignment Details
View Assignment Timeline
Receive Notifications from Teaching Staff and Admin
Submit Assignment
View and Download Submissions
View and Download Grading Reports
Student
Admin Permitted Actions
Manage All Roles and Permissions
Manage All Courses
Manage All Assignments
Manage All Grading Reports
Admin

Figure 1 Use Case Diagram

Student-specific Requirements

Based on the courses a Student user is currently enrolled in, one can view the assignment timeline, consisting of upcoming deadlines of assignments from all the enrolled courses, and view the details for each assignment. A student can also receive notifications issued by Teaching Staff or Admin.

One can submit an assignment as a compressed file by creating a new submission. All previous submissions can be viewed in submission history and downloaded. Each graded submission has a detailed grading report with for example failed test cases, and any regrade will also generate a new report. All the grading reports are accessible by the student.

Teaching Staff-specific Requirements

The Teaching Staff user can manage assignments by configuring the assignment information and grading details, for instance, the grading pipeline. As students submit assignments, Teaching Staff can view the submissions of courses they are involved in and could trigger assignment grading immediately as opposed to the scheduled grading dates specified in assignment information. After grading, further manipulations to the grading reports can also be done, e.g. manually adjust grades.

Teaching Staff can send notifications to their students, and receive notifications announced by Admin.

Inputs	*Pre-Grading*	*Grading*	*Post-Grading*	*Outputs*
Decompressed submissions	File structure validation	Test cases	Score-weighting	Grading Report
Assignment configuration	Compilation	Custom scripts	Late submission score deduction	Logs
Assignment dates	Diff with skeleton	Coverage	Plagiarism detection	
		Static analysis	Memory leak detection	

Table 2 Grading stages (Color Code: Inputs and Outputs, Basic Features, Potential Features)

Table 2 illustrates the grading pipeline. A pipeline is composed of grading stages that a teaching staff chooses to use when grading a particular assignment. To execute the pipeline on a submission, it accepts 3 inputs: a decompressed submission, an assignment configuration specifying all the provided options, and a schedule of the assignment as dates. Teaching Staff then select all pre-grading, grading, and post-grading stages as they see fit, where each stage will perform a specific task. Stages are sequentially composed and can run in parallel in the pipeline. Finally, a grading report containing the detailed result and a system log is generated for review.

The pipeline can then be used for grading the assignment for all submissions, which may be updated after creation. The two main ways to run the pipeline are:

1. To give immediate feedback. This is for submissions received before the deadline. If enabled in the grading config, it will run the pre-grading stages and grading stages that are marked as “immediate feedback” (e.g. visible test cases) and publish results to the respective students.

2. To grade in full. This is for the usual grading work after the submission deadline, and also for any regrades needed. It would run the entire pipeline, from pre-grading, grading, to post-grading stages.

An illustrative example of constructing and using a pipeline is included in the Appendices in Grading Pipeline Example.

Administrator-specific Requirements

An Administrator (Admin) extends all features of the Teaching Staff user.

Additionally, an Admin is responsible to manage all the courses of the current semester, including creating, deleting, and archiving old courses. The roles and permissions of the courses are also fully managed by the Admin, where there are both general roles (Student and Teaching Staff) and course-level roles that only apply under a course. Also, all assignments and grading reports of all courses can be managed by the Admin.

Webhook

The CS System Webhook is an actor external to our system, providing course and enrollment information via synchronization. The Submission Subsystem can then access the most updated data.

Non-Functional Requirements

In addition to the aforementioned functional requirements, we also have a set of non-functional requirements we aim to achieve. These requirements concern the technical aspects of the system that users may not directly observe but would be critical to a quality user experience.

Security

Security is relevant in both the Submission and Auto-Grader Subsystems. The Submission Subsystem needs to make sure users are properly authenticated and roles are well managed. The Auto-Grader Subsystem needs to ensure that the student-submitted codes would not harm the execution environment, considering that there might be malicious code, intentional or not; the execution thus needs to be isolated in an easily discardable environment.

Performance

Performance concerns user traffic in the Submission Subsystem, especially for a course that has hundreds to a thousand students, and during assignment deadline periods. Load balancing and clustering are needed to serve the high number of requests in a scalable manner. The performance also concerns the speed of grading pipeline execution. While pipelines requiring quick feedback need to return results relatively fast, most grading works are done in the background after some submission deadlines and do not impose high-speed requirements. These deadline grading tasks shall be completed within a reasonable timeframe, targeting completion within a few hours.

Traceability

Traceability refers to the ability for users to retrieve submission and grading history, to use recorded facts to resolve confusion arising from various versions of an assignment and grading statues of an assignment (the feedbacks and rounds of grading with the possibility of regrades). On the semester and course levels, traceability is also required to distinguish between different offerings of a course in different semesters, and the different sections of a course, as well as an assignment submitted by various students. We devise to use a properly designed directory scheme in the file system and database tables to store the many versions arising from different students, assignments, and reports using their respective identifiers so that we can easily retrieve data in groups. These storage mediums are mainly append-only so the full record can be stored and tracked.

The suggestion of using Git VCS to keep track of the files and their changes is discarded after considering further that we choose not to incur extra maintenance cost of these changes just to conform to Git operations, and we are only using a limited subset of Git features so the benefit is negligible.

2.1.2 Implementation

Architecture

Architecture Diagram

Figure 2 Architecture Diagram shows an overview of our system architecture, showing the subsystems.

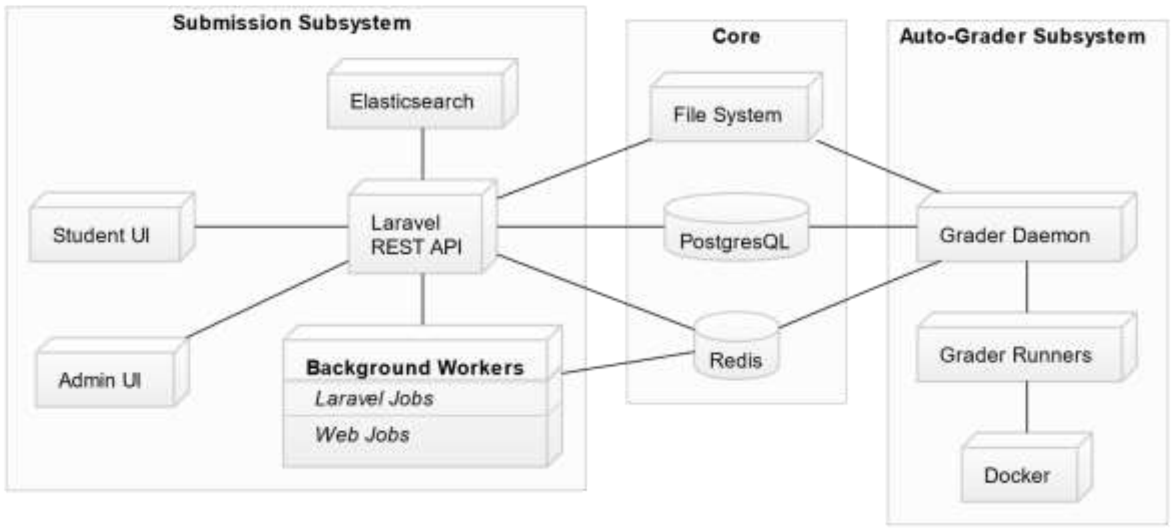


Figure 2 Architecture Diagram

Computing Resource Management

To facilitate scalability, we employ the technique of load balancing. Load balancing refers to the practice of mindfully using available computational resources to process requests, by routing clients to less busy servers so that the processing time would be more uniform across servers.

To achieve this, components are divided into stateless and stateful components. Stateless components do not store states in memory, in contrast to stateful components that do. Stateless components can be easily cloned and scaled up to serve more clients at a time or scaled down to release resources for other tasks. Stateful components are run in clusters to facilitate the processing of high numbers of requests.

This will be elaborated in more detail in the Technology Stack section.

Schema Design

The database records persistent memory. The following Class Diagram illustrates how the schema is implemented without the sight of necessary but visually cluttered columns like IDs. The schema is presented in Figure 3.

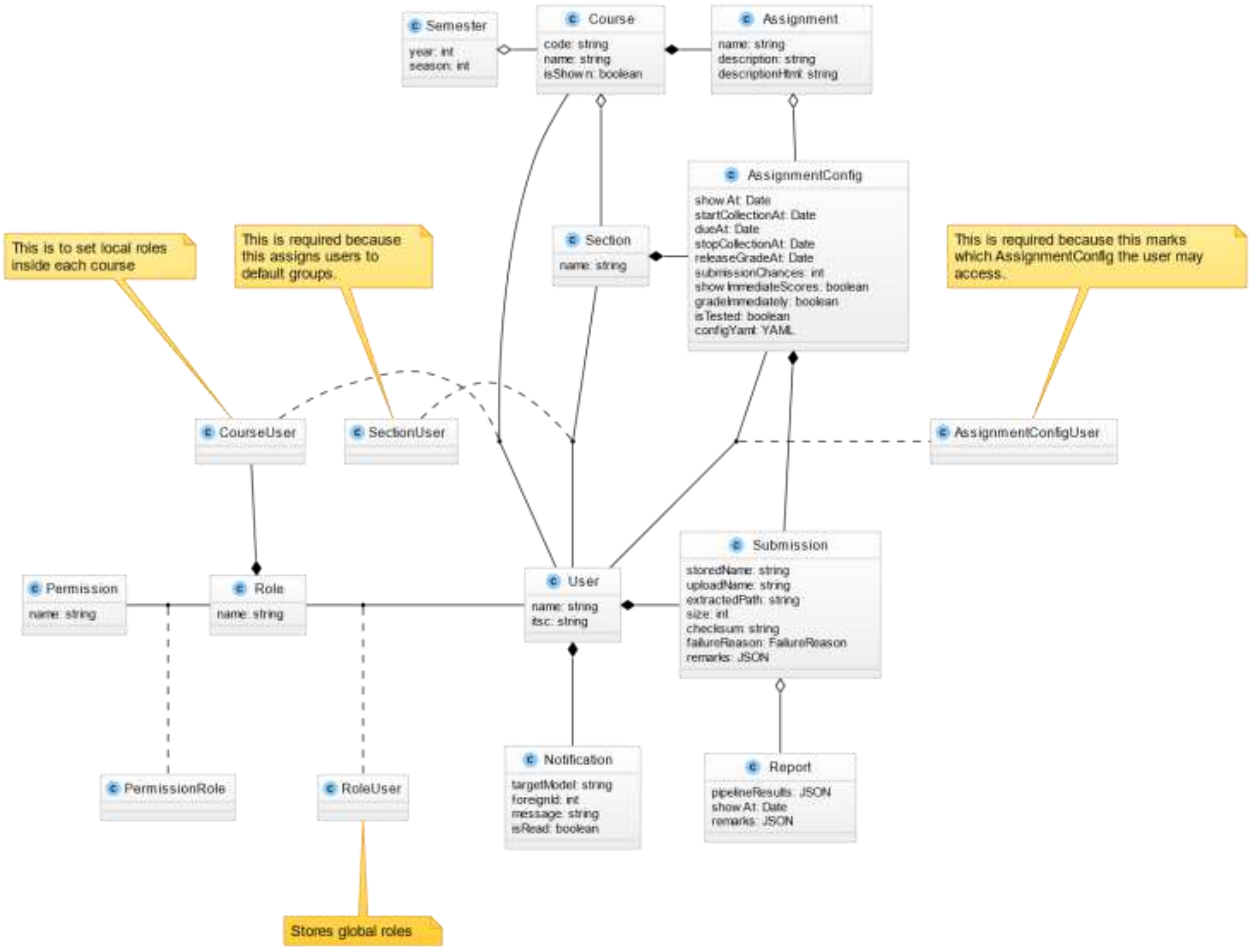


Figure 3 Database Schema Class Diagram

Inter-Process Communication (IPC) Design

The entire project consists of two subsystems which are required to communicate with each other to exchange information:

- web: The Submission Subsystem component.
- grader: The Grader Subsystem component.

web itself includes:

- The api server, handling client requests.
- A set of laravelJobWorker background workers executing jobs dispatched internally from api using Laravel's Job API (via Redis).
- A set of webJobWorker background workers executing jobs dispatched for web via Redis.

The communications among components and sub-components are organized in topics and messages in the style of a publisher/subscriber model. A more detailed implementation is given in Background Workers and Redis.

Comprehensive Graph

Figure 4 IPC Graph illustrates a comprehensive view of the IPC connections in a simplified graph.

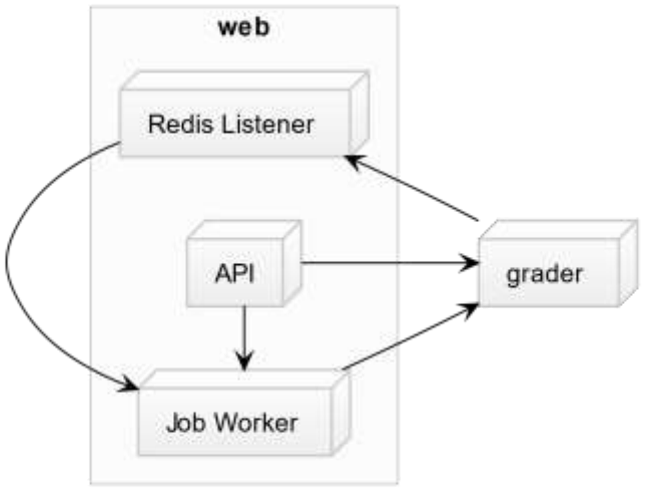


Figure 4 IPC Graph

Background Workers and Redis

The Submission Subsystem runs some PHP background workers independently in separate processes. Specifically, some are provided by the Laravel framework called Queue Workers, and some are custom written workers to process background tasks. These jobs are managed by queues on Redis, with data resilience and horizontal scaling in mind, to complete jobs issued by the API Server and the Grader Subsystem.

The algorithm for supplying and consuming jobs is designed with the premise that Redis crashes do not result in data loss and a full recovery is possible. Under this premise, job workers will reserve jobs they are immediately doing on a separate set on Redis. Each job is limited to run under a certain time constraint. When the job is completed by the worker, the worker removes the job from the reserved set and pops another task from the primary queue or sleep and continues polling for another available job. This repeats indefinitely.

If the worker crashes during the job, the job will time out and be cleaned out eventually, to be put into the primary queue again, until retry counts run out. In that case, the failed job is logged into the database for record and further investigation. This same queue mechanism is implemented in the Grader Subsystem as well, to process grading tasks and such, and lives in a coroutine instead of a separate process like in the PHP implementation.

IPC Jobs

There are 5 different IPC jobs in use currently, and they can be categorized into 2 types: queued jobs and immediate jobs. Queued jobs are resilient because a messenger of choice will keep them in a queue and make sure only after the job is successfully finished would it be removed from the queue. Immediate Jobs are fast but could be lossy because the messenger quickly sends it over and expects a quick response; this is usually a publisher-subscriber mechanism.

There is a grader queue that contains all queued jobs the grader shall process, and an API queue for the Submission API to consume, so that each subsystem can simply add a payload to the opposite queue to make requests and transfer results.

The jobs are:

1. gradingTask, doneGrading are queued jobs for immediate grading. gradingTask describes an upcoming grading task for an assignment, potentially with many submissions. doneGrading is the response job after a gradingTask is completed, and will be added to the API queue. They should be queued jobs because they are critical requests and their messages must not be lost.
2. configUpdated is a queued job for scheduled grading. It characterizes the event where a teaching staff updates an assignment configuration. This may or may not cause a change in scheduling, because assignment dates such as the “stop collection time” may be changed.
3. validateConfig and configValidated are immediate jobs for assignment configuration editing. They define a simple way for the Submission UI to determine whether a given assignment configuration is correct using the Auto-Grader as a backend. These jobs could be issued in large numbers, the workload is light compared to other jobs, and are in nature usually reissued by user actions so data loss is not a concern.

Each Job has a payload, which is a JSON object that specifies what data this particular job carries.

Queued jobs are implemented using custom Lua scripts embedded in Redis EVAL commands to provide atomic queue operations and maintain a reserved copy of each queue as a backup for crash recovery.

Immediate jobs are implemented using Redis SUBSCRIBE and PUBLISH commands on defined channel names.

Technology Stack

Database: PostgreSQL Cluster

PostgreSQL [6] is the second most popular database solution according to StackOverflow [7]. It is a type of relational database management system (RDBMS). Although NoSQL solutions like MongoDB [8] and Cassandra [9] are gaining popularity in recent years, we are choosing RDBMS over NoSQL because the data we have is structural, which can be represented with a traditional schema. PostgreSQL is chosen instead of the most popular MySQL [7] due to its feature to have schemas (containing tables) under a database, providing an easier solution to testing. But in conclusion, there is no compelling reason to choose one RDBMS over another in our case, considering that our data access rate would likely not show any differences in the various database solutions.

Since the database is a stateful component, it cannot be upscaled as easily as stateless components. PostgreSQL provides a clustering solution, so we use it to do load-balancing.

Redis

Redis [10] is an in-memory data structure store. It is a type of NoSQL solution. Although it can act as a database, we mainly use it for its cache, and message broker features.

Since the database lives on secondary storage, fetch time on PostgreSQL can be assumed to be greater than that on Redis, which resides on the main memory. For repetitive calculations, results will be cached for a limited time on Redis to help improve performance.

The message broker feature is used for communication between the Submission side, Auto-Grader side, and other microservices.

2.1.3 Testing

Since this system is aimed to be used in the long-term, extensive testing is a crucial part of the development of this system. Our team employ Agile Testing to ensure the correctness of the system. Agile Testing requires developers to write test cases as features are being implemented, incrementally building up the application until a milestone is reached. The benefit of using Agile Testing is that it allows for rapid prototyping during the initial MVP phase of development. This lets clients provide feedback to prototypes in a timely fashion, which prevents potentially major changes in design during later stages of development.

Integration Testing

Since this system consists of multiple isolated subsystems, our test suite also contains integration tests which aim to show that the different subsystems can be interfaced successfully and that all success and error outputs from one subsystem are properly handled in all other subsystems.

An integration test case consists of a mock client initiating a specific request to a subsystem, and the result of the request. The test succeeds if the result of the request matches the behavior as specified by our design.

User Acceptance Testing (UAT)

At the later stages of development, we shall conduct user acceptance testing to ensure that the end-user can operate the system with ease and collect feedback from end-users to improve the potential shortcomings of the current system iteration. The UAT will be conducted on professors, teaching assistants (TAs), and students to ensure all use cases of the system are tested against their respective actors.

2.2 Submission Subsystem

The Submission Subsystem deals with the user-facing portions of the entire system. It consists of the UI Servers, API Server, a search engine, and background workers. It is in charge of collecting submissions, displaying their results, managing assignments, and communications.

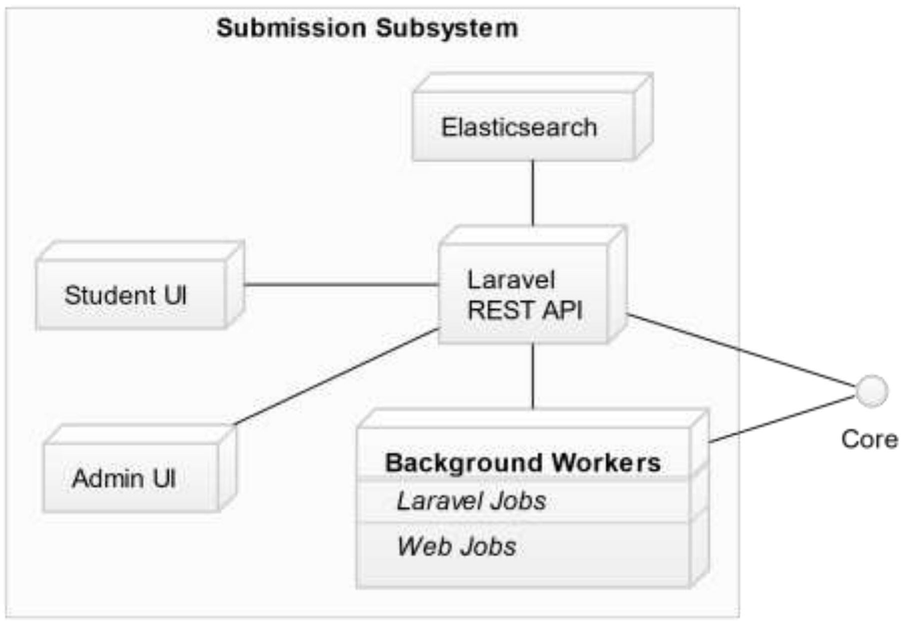


Figure 5 Submission Subsystem Architecture Diagram

2.2.1 Design

User Interface Design

The User Interface (UI) design consists of two views:

- Student, for students to submit assignments, and
- Administrative, for teaching staff and system admins to manage contents.

A clear layout difference separates the 2 sets of views. UI Details, Justifications, and Screenshots may be found on page 118.

Concepts of the Student View come from CASS, Canvas, and GitHub to allow for an easier transition to ZINC, including placements of navigation bars, buttons, and other elements of general layouts.

Dark Mode has been added for visual comfort. In the future, we may implement a high contrast mode, to improve the UX for visually impaired users.

Web Backend Services

The Submission Subsystem code should be designed to work with horizontal scaling, meaning that there should be no race conditions raised if the subsystem components were upscaled.

CAS Authentication

Central Authentication Service (CAS) [11] is a single sign-on (SSO) protocol used by the HKUST Information Technology Services Center (ITSC) to provide intranet services to infrastructures on the school premise. The school uses the Lightweight Directory Access Protocol (LDAP) [12] to maintain user directory services. This is the authentication provider used by CAS of the university.

CAS is used in our project as an external source of access delegation, as it provides a secure and easy way for internal users to access the web interface. There is also less security concern since our database does not store any user credentials. The full name of the user, which is provided in the list of attributes via CAS, is saved alongside their ITSC username into our database for account creation on our platform and communication purposes.

JWT and User Session Management

JSON Web Token (JWT) [13], is an open industry standard for representing claims securely, and a commonly used token-based authentication method across the Internet. JWT is used in the web interface portion to communicate between the API server, and the UI server as well as the client browser.

JWT was specifically chosen for the following reasons. Firstly, we have decoupled the UI and API aspects of the web services, so the UI servers will be required to request for resources on behalf of the user for pre-rendering, which will be mentioned later. Authentication via a token also allows for easy horizontal scaling of the UI servers and API server, as stateful information is only stored on the database and Redis.

Figure 6 is an example showing the interactions between the 4 participants: Client (user browser), UI Server, API Server, and the external CAS Server. In short, after the CAS authentication, the client browser is required to pass the CAS ticket to the API in exchange for a JWT token for resource access, for the client and the UI Servers on behalf of the client.

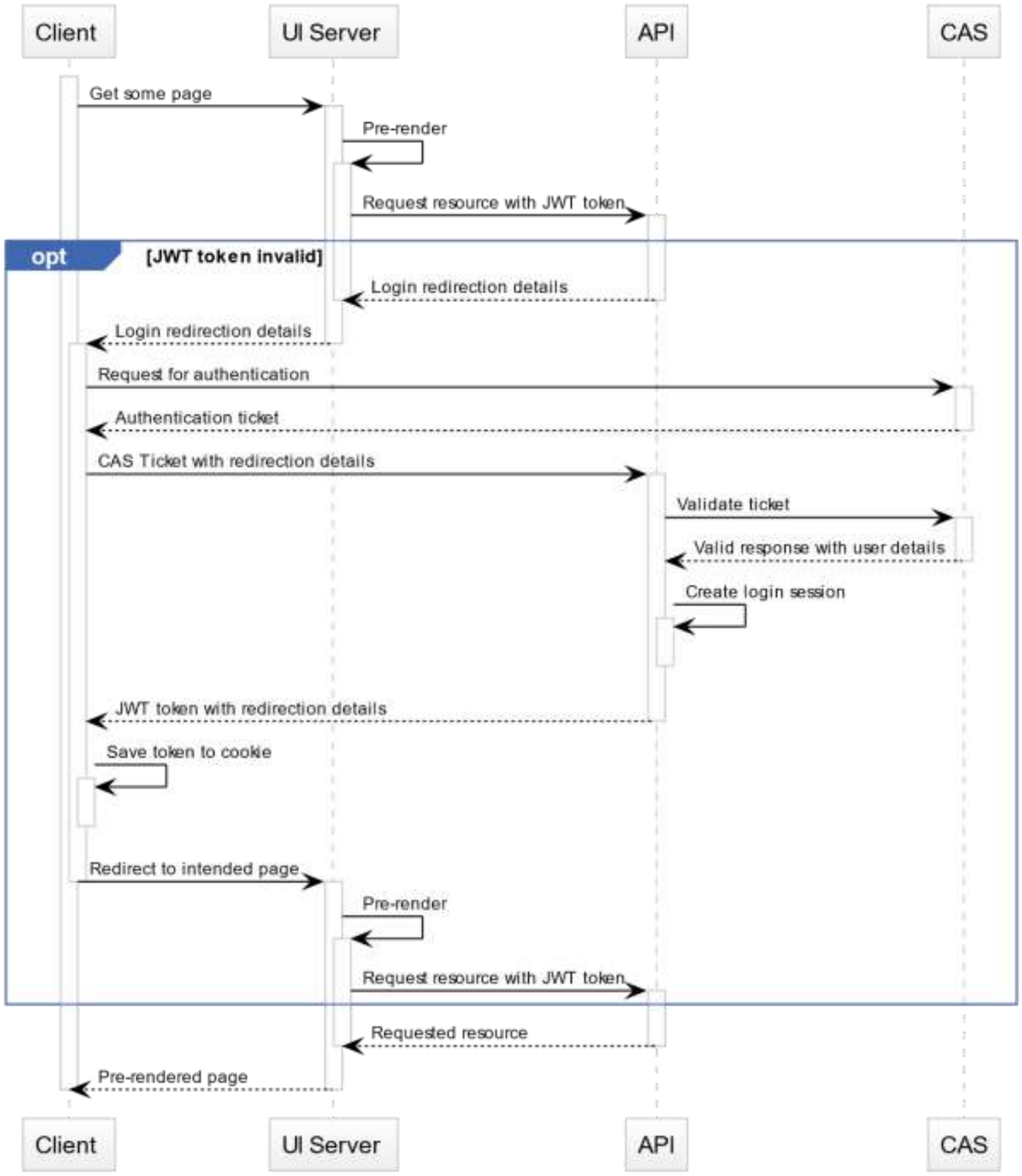

Client
UI Server
API
CAS
Get some page
Pre-render
Request resource with JWT token
opt
[JWT token invalid]
Login redirection details
Login redirection details
Request for authentication
Authentication ticket
CAS Ticket with redirection details
Validate ticket
Valid response with user details
Create login session
JWT token with redirection details
Save token to cookie
Redirect to intended page
Pre-render
Request resource with JWT token
Requested resource
Pre-rendered page
Client
UI Server
API
CAS

Figure 6 Resource Request with CAS Login Sequence Diagram

CS System Resource Synchronization

The Submission Subsystem is required to know the current enrollment status of students to be able to deliver accurate information to the users. The HKUST CSE CS System keeps enrollment details for CSE students. This module is required to retrieve a list of student enrollments from the CS System and insert and update relevant records in our database, such that our database reflects the current enrollments of the students.

Roles and Permissions

The API Server serves requests from the Admin and Student UI Servers as well as from browser clients indiscriminately. Users created via the CAS SSO are all created as normal users. Resources on the API Server are guarded by Role-Based Access Control (RBAC). There are two kinds of roles: global roles and course roles. “Admin” is a global role, granting the user permissions to execute all requests. “Teaching Staff” and “Student” are course roles, granting only some permissions to accommodate the intentions of the roles. Roles are associated with permissions, and requests are directly guarded by these permissions instead of by roles to increase flexibility. So, it is easier to add new roles in the future, like “Auditor,” to have read-only permissions for all courses as an example.

“Student” roles are synchronized automatically from the enrollment synchronization with CS System. “Teaching Staff” and “Admin” roles, on the other hand, requires system administrators to assign them to users manually.

Inter-Process Communication Services

To communicate within the Submission Subsystem across components, and with the Auto-Grader Subsystem, IPC utilities are required.

2.2.2 Implementation

The Submission System consists of web services and background workers. The subsystem uses PHP Laravel for the backend: API Server and background workers, whereas Node.js and React are used for the UI Servers.

Web User Interface

React, Next.js and User Interface Rendering

The Submission Subsystem UI is delivered by Next.js. Next is a popular React framework, providing a routing solution, delivery optimization, and server-side rendering [14]. The UI project is managed with Yarn Package Manager [15] using workspaces, a feature by Yarn to manage sub-project packages, and allows them to depend on each other. The Admin and Student-facing UI are two Next workspaces, with a shared package containing common components and themes.

The UI framework used for React is Theme UI [16], providing primitive UI components, CSS theming in JavaScript support, and convenient responsive design styling helpers.

Design and color choices are also taken seriously in the project, to deliver a professional and accessible UX. The layouts are inspired by Canvas and CASS, as we strive to create a similar UX to allow for an easy transition.

The programing style in React we have chosen is functional programming with Functional Components and React Hooks. Higher-Order Components (HOCs) are used to compose the render functions to augment their functionalities. For example, withResource is an HOC, collecting required resource requests and parallelly execute them to save time and allow for pre-rendering under the constraints of Next.

We used the server-side rendering capability of Next. Next will preliminarily render the React app for the specific page first. The DOM of the rendered React app is sent to the user, along with some initial page-specific React properties, which are serialized as JSON, and sent alongside the DOM to the browser client for rehydration. This accelerates page loading, by pre-populating the DOM with resources requested locally on the server-side. The user is also able to view the web page during web page download, as the React app readies seamlessly.

Quality User Experience

For providing better ease of use in UX, we have integrated TinyMCE [17], a WYSIWYG HTML5 editor, into the administrative panel, allowing teaching staff to design an elaborate assignment page with full HTML capability.

DataTables [18], a mature and trusted jQuery library, is used for every REST resource in conjunction with Elasticsearch, to create a database GUI browser similar experience for administrators and teaching staff to monitor and view resources in their browser. An extensive and fuzzy search experience allows for ease of searching.

Web Backend Services

CAS Authentication

We are using the Composer package subfission/cas for this feature. Most of the CAS interactions come built-in with the library.

JWT and User Session Management

We are using the Composer package tymon/jwt-auth in this feature. On top of this, we have implemented “Sliding Session” such that the client may renew their token before the expiration time or if their token is recently expired if they are still using the API services.

Roles and Permissions

We are using the Composer package spatie/laravel-permission in this feature to manage roles and permissions with course roles and permissions extending on top of this.

CS System Resource Synchronization

We have contacted CS System for an interface to synchronize student enrollments, and automatically set up and populate relevant courses. CS System has currently provided us a REST API solution to poll CS System periodically for updates.

However, it is to our best interest to implement a webhook instead, since it decreases work on both sides in dealing with the polling and provides a real-time solution to run a synchronization on change. We will communicate further with the CS System to negotiate this issue in the future.

Inter-Process Communication Services

The medium of IPC is fixed with Redis in the Submission System at this point. Redis is currently being used within Laravel to dispatch and run Laravel Jobs, cache API intermediate results, and communicate with the Auto-Grader Subsystem.

Three types of message worker services are hosted, and we give them the following names and functions.

- Laravel Jobs—provided by the Laravel framework to manage internal Job dispatched.
- ZINC Queue—works through queued up, non-time-sensitive messages from the Auto-Grader.
- ZINC Subscriber—immediately resolves time-sensitive messages from the Auto-Grader.

The first type of message worker is referred to as laravelJobWorker and the latter two by webJobWorker. webJobWorker may in turn dispatch other Jobs if required, and they may be processed in the current process or queued up in Laravel Jobs as internal jobs. For modularity and horizontal scalability, these are all run in separate processes in different Docker containers in our deployment.

Currently, dispatchable Jobs include decompression jobs, email dispatching, and message-receiving jobs. These are time-insensitive or long-running tasks and tasks involving IPC with the Auto-Grader.

Technology Stack

Web Server Language: PHP & JavaScript

JavaScript is the programming language of the web, and Node.js [19] has become quite ubiquitous as a server-side programming language in recent years. JavaScript is an interpreted language, therefore making it easy for quick deployment and updates.

Other languages were considered for web programming, namely: PHP [20] and Python [21]. PHP is still the most common web programming language. According to W3Techs, PHP makes up for 79.1% by 4 August 2018 [22]. PHP runs synchronously with each request as a process. Laravel [23], for example, is a widely used PHP framework, requiring to “boot up” every time a user arrives on the website. Node.js, on the other hand, keeps the server code running after startup, and serves users asynchronously, reducing overhead in the initialization of the server. However, the overhead of PHP can be reduced when disabling unneeded Laravel features.

Python, on the other hand, lacks the developer community in comparison to PHP and JavaScript [7]. Although some server frameworks may leverage better performance by running native code when possible, the language is not considered in the end based on its comparative lack of popularity in server-side programming.

In contrast, in recent years, JavaScript has gained traction as server-side programming as well, being the most popular language on StackOverflow for 7 continuous years [7].

We have decided to use PHP Laravel for our API server since it is tried and tested and has become the most widely used PHP framework [24]. It has a large and active development community. The framework has many security features with minimum setup, guarding against common network attacks. This is the reason why we believe it is best suited for our API server. We have previously opted to use Node.js for the API server, but development was slow because of the rather limited resources in comparison to Laravel's vast community-developed libraries, which could greatly accelerate development speed. One of the reasons that Node.js was previously chosen was to experiment with using GraphQL with the popular Apollo library. Since we have changed to REST instead, which Laravel excels in handling, there are even more compelling reasons for us to use PHP.

User Interface Framework: React & Next.js

React [25] is the most popular JavaScript MVC framework, alongside other popular ones like Vue [26], and Angular [27]. React, used by Facebook who also drives its development, is a well-tested and mature framework, used by the industry. We are using Next.js to provide server-side rendering.

We have previously opted for Vue.js and Nuxt.js [28]. However, development on Nuxt.js was quite slow due to its slow compilation time (due to the design of Vue), and its relatively small development community. Several critical questions were never solved during development with few open-source tools available. On the other hand, React, with its recent updates on the new Hook-related features, it has overcome previous design flaws, making it easier to communicate between parents and children components. Due to these circumstances, we have decided to use it to implement our user interface.

API Framework: PHP Laravel

There are mainly 2 popular API design architectures, namely: REST and GraphQL [29]. GraphQL is a newer architecture led by Facebook, designed to solve the problem of requiring multiple client requests for required information, i.e. *over-fetching* and *under-fetching* in REST API. Under-fetching occurs when the client attempts to fetch resources related to the results of previous requests. Over-fetching occurs when the response contains some useless information. These situations would result in a waste of network resources and time.

GraphQL solves these problems by exposing a single server endpoint which accepts any query, then collects the required information and returns one response to the client – reducing much of the overhead time of the multiple connections there used to be between the server and client when using REST. Apollo [30] is one implementation of GraphQL in JavaScript. Facebook improved much of its performance by using GraphQL internally [31]. The improvement was due to its previously poorly designed REST API calls and the vast number of such calls needing to be managed and optimized.

REST, on the other hand, is a very mature protocol, used by the industry since HTTP 1.1. If REST API calls were designed with optimization in mind, there should be no over-fetching and under-fetching concerns since each call is designed to only retrieve needed data. Additionally, since each resource requested in a GraphQL query is resolved statelessly, a well-maintained REST API should have an even smaller demand on computational resources than a GraphQL API. GraphQL frees networking bandwidth from the client but uses more local computational and networking resources. With our relatively small number of API resources to maintain, it is more advantageous for us to implement a REST API instead, thus we have decided to switch to using PHP Laravel with reasons stated.

Elasticsearch

Elasticsearch [32] is an open-source search engine implemented in Java. Elasticsearch is used in the Submission Subsystem to provide searching ability in the indexed models from the database. Elasticsearch provides a flexible, fast, schema-free, JSON HTTP search API, able to perform full-text searches with many features. One of the core features users will be using is its fuzziness search, being able to search from keywords with typos, to improve the UX.

2.2.3 Testing

PHP

The Submission Subsystem API utilizes PHPUnit [33] and Mockery [34] for unit testing and feature testing.

Feature testing includes controller tests that require a data source. For efficiency, we have opted to use an in-memory SQLite database. It is worth noting that SQLite and PostgreSQL have different language features. This may lead to the behavior of the production time of the software to differ from that in test time. However, this is worth sacrificing to gain a faster development cycle. To promote a version to the stable branch, a full integration test with PostgreSQL is required.

To accommodate the integration with the CAS of ITSC, we have created a mock CAS server, Fake CAS, to test out CAS behavior internally for development purposes. The Fake CAS component may be swapped in and out with the actual ITSC CAS endpoints.

Code Quality Tools

We are using the following tools to establish a standard of coding style in our source code.

- PHP_CodeSniffer. Our configuration is based on the PHP PSR2 standard.
- PHP CS Fixer

JavaScript

Code Quality Tools

ESLint is used with the Standard coding style enabled for JavaScript and TypeScript.

GitLab CI

GitLab CI is a continuous integration tool provided by GitLab. Coverage reports, unit tests, docker image builds, and coding style inspections are carried out with GitLab CI. Code can only be merged into the master branch when it has passed the various tests carried out in GitLab CI.

2.3 Auto-Grader Subsystem

The Auto-Grader Subsystem conducts automatic grading for submissions in a containerized environment. The subsystem consists of a daemon process that distributes tasks in runners, which in turn issues the actual grading tasks in Docker containers.

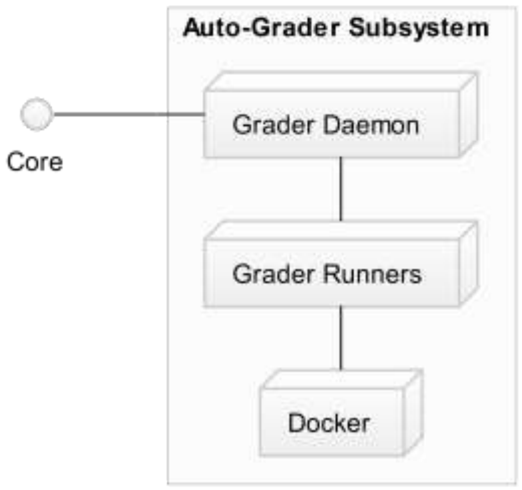


Figure 7 Auto-Grader Subsystem Architecture Diagram

2.3.1 Design

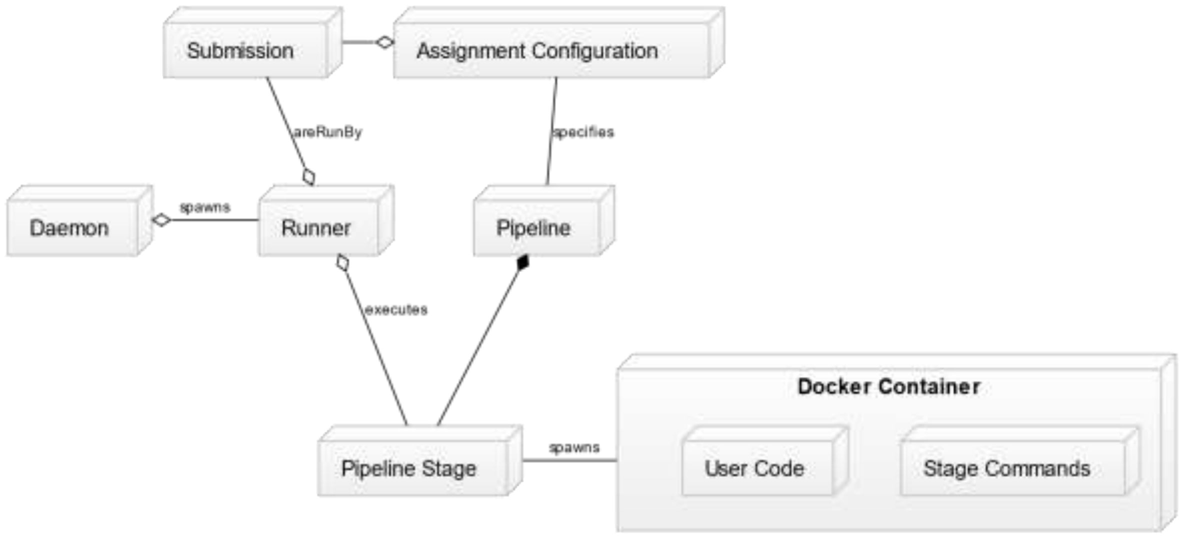


Figure 8 Grader Components and Relationships

For the Auto-Grader, we opted for a delegation-like design pattern, where a daemon will queue and dispatch grading tasks to a pool of grader runners. Each runner will execute a set of grading tasks in the form of a pipeline, as specified in an assignment configuration.

This design also upholds strong isolation and segregation of responsibilities. First, the daemon, runner and containers can exist on their own, so that for example if any grader instance crashes or errors during its execution, it will not crash the daemon process, and vice versa. Second, runners concern the setting up and cleaning up of grading environments, pipeline stages, and related containers only care about how to execute the particular stage logic, and finally, the daemon is just responsible for the orchestration and request handling.

Daemon Design

The grader daemon is the entry point of the Auto-Grader. It should trigger subsequent runners and pipelines to perform needed grading work. It should also coordinate with the Submission Subsystem as an interface to receive and send requests in the form of IPC Jobs.

A messenger is needed to implement IPC Job handling on the daemon. Currently, we fix this to be Redis, but other frameworks can be used as long as it extends the common interface.

As mentioned in configUpdated, Scheduled grading has to be handled by the daemon as well. There should be a mechanism for grading tasks to shelf and wait until a scheduled time to begin, meanwhile the data used should be fresh and latest, instead of what was there at the time of starting the schedule.

Grader rehydration is a direct consequence and requirement from scheduled grading because now the Auto-Grader is stateful and needs to store the scheduled grading tasks, which in turn needs to be able to recover from a system crash and continue to be scheduled.

Runner Design

Grader runners encapsulate grading tasks in the Auto-Grader. A runner should be responsible for managing an assignment configuration and a submission, initiating the corresponding grading pipeline, and forcing the pipeline to execute the pipeline stages by delegating to the methods specified in each stage. Finally, it aggregates any reported results from the stages to generate a report for this grading task.

Grader runners are heavily inspired by GitLab's Continuous Integration runners, where users define a CI pipeline for the runner to execute at predefined times (e.g. repository update, merge requests creation). Since we believe that both CI runners and grader runners share a lot of common properties, such a workflow representable in a linear fashion and the requirement for isolated environment, we have adapted aspects of the runner into our runner design.

Grader runners will operate in environments isolated from other runners and the host environment. Further details are explained in the section below. There are several benefits to using this approach.

Firstly, using runners to execute the grading procedure allows the grading process to be decoupled from the Daemon, meaning that if any grader instance crashes or errors during its execution, it will not crash the Daemon process.

Secondly, runners simplify some aspects of the grading pipeline. Since runners are designed to be single-use, the burden of setting up and cleaning up the grading environment falls onto the runner. Given that runners execute grading tasks in isolation from one another, this means that the setup and cleanup logic does not need to consider cases potentially affecting existing or newly created runners.

Finally, the isolation guarantee of runners means that the base environment which the grader executes can be modularized into images. This means that a new grading environment can be created in a short amount of time, reducing the overall time of one grading task. This also means that any malicious user code executed in the container will not propagate to the host operating system, which enforces a level of security between the grader runner and the other systems.

Pipeline Design

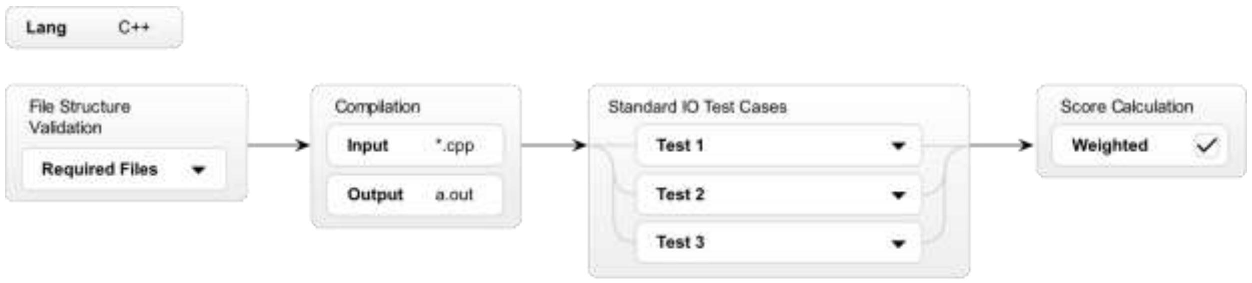


Figure 9 Example of a C++ assignment pipeline

Since one of the functional requirements of the Auto-Grader Subsystem is to allow for flexibility and extensibility, the grader pipeline is designed with those requirements in mind.

The grader pipeline consists of one or more stages, with each stage accepting an input and returning an output (Figure 8). The output of each stage will be provided to the input of the next stage. This design allows the user to specify any combination of stages to construct a grading pipeline while providing a mechanism to pass minimal information between the grading stages.

Each pipeline stage is associated with a Docker image, which acts as the template for all containers created in this stage (Figure 8). The benefit of creating and caching images on a per-stage basis is since pipeline stages are often reused across assignments, additional packages which may be required in a pipeline stage can be pre-installed into an image, reducing the startup time for all containers. If no such functionality is provided, the installation of dependencies will instead be handled when the container starts, meaning that all containers will have some delay before the actual task can start executing.

During the execution of each pipeline stage, the runner creates a Docker container which acts as the containerized environment and runs the task within the container. A timeout is set for each container, meaning that potential errors encountered during grading (e.g. infinite loops) will not lock up the entire Docker host. Pipeline stages also implement the ability to collect the execution result of itself, which is stored in the Grader runner for reporting purposes.

Since most grading tasks in programming assignments will involve some form of file manipulation, such as during compilation and standard I/O testing, the pipeline uses Docker volumes as scratch space, acting as temporary storage for intermediate files required by the grading task. Similar to Docker containers, the storage used by Docker volumes is isolated from the host system, enforcing separation between the grading environment and the host environment.

Figure 9 shows an illustration of a simple example pipeline that can be used to auto-grade C++ programming assignments using console input and output. This pipeline can also handle submission file structure validation to see if required files are properly submitted, compile the program, and calculate the score across test cases in an end-to-end fashion.

By adopting a pipeline design to programmatically represent the grader workflow, the teaching staff can freely compose the pipeline based on their previous grading workflow and the requirements of the assignment. Moreover, the composable nature of the pipeline also allows teaching staff to easily replace or extend existing workflows, achieving the flexibility design goal.

The stages are categorized into kinds: pre, grading, and post kinds, signaling where this stage is designed to be located along the pipeline.

Resource Management

On the Auto-Grader subsystem, since the most constraint resource is CPU time and memory for executing the grading tasks in Docker containers, three methods have been devised to enact resource management.

Firstly, containers by default enact a strict resource limit on CPU time, memory, and clock time. This limit prevents grading tasks from competing for computing resources, causing all tasks to slow down.

Secondly, a Docker job dispatcher is implemented for dispatching jobs to multiple Docker engines. This means that if there is a need to increase the throughput for grading assignments, additional physical machines can be added to the system to provide more computing power to the grader. Also, support for multiple Docker engines allows courses that require grading on a non-Linux environment to be supported.

Finally, the dispatcher also allows tasks to reserve more computing resources than the host system, allowing higher workload tasks to better utilize the computing resources that may be reserved by other jobs but left unused.

2.3.2 Implementation

Daemon Implementation

The daemon, as the always-running component of the Auto-Grader, is an event-driven system. At initialization, it has to establish a connection with the Database for later use by other Auto-Grader components.

Redis, the messenger in use, implements both queued jobs and immediate jobs, using the Java library Jedis for performing the various required Redis operations.

These two types of jobs are processed by issuing two blocking non-terminating calls to two functions, each continuously handling jobs of a type. Even though they are blocking, they can still be run in parallel because they are wrapped in Kotlin coroutines for lightweight threading.

As soon as a job arrived, its job name will be matched and its JSON object payload parsed. If parsing was successful, the respective job will be issued. Otherwise, the error will be reported in the logs, and the daemon will continue to process new jobs without crashing or being adversely affected.

In particular, gradingTask will spawn a grader runner for each submission in the payload and delegate the grading task to it. Each of these runners upon completion would generate a report detailing errors and pipeline results as described in Runner Design, which will be inserted to the Database for other components to access, followed by a doneGrading job being created and pushed to the API queue.

Also, the configUpdated job is implemented using Java Timer Tasks, which has the benefit of being able to launch a prescribed function independently when an assigned date and time is reached, using a background. This is chosen to be the “stop collection time” of the respective assignment configuration. In the prescribed function, gradingTask job handling can be reused since when the time has arrived, a scheduled grading task would be degenerated to an immediate grading task, with zero waiting time. Only by then would the grading task pull data from the Database, to ensure the latest records are used.

Grader Rehydration is implemented with the help of Redis since it is a more resilient service and can store serialized data easily and reliably. We are assuming a baseline that Redis should be working for the system to restart itself. For both scheduling and rehydration to work, the daemon has to keep two copies of the schedule: the first is an associative map data structure in Kotlin from an assignment configuration to a Timer Task for Kotlin to keep scheduled tasks alive, and the second is an associative map in Redis to store a serialized version for backup.

Runner Implementation

The main purpose of a runner is to coordinate the execution of a pipeline by converting configurations provided from the user into a container specification. Since the Grader runner forms a major part of the Grader, this section will go into more detail about its implementation.

When the runner executes a job, it first parses the assignment configuration YAML (detailed in Pipeline Implementation) attached to the runner, to build a pipeline containing the sequence of tasks to execute. To improve the runner’s performance, the properties of the pipeline cannot be changed after the pipeline is constructed, meaning that once the pipeline’s integrity is verified, the pipeline will remain valid during its execution by a runner.

A runner requires two inputs to start execution: the pipeline and a context containing data essential for running the pipeline in the environment. The context includes data such as the path of files required to run the submission and some options to customize the runner. This decoupling allows the same pipeline to be reused across different submissions by using different contexts.

Before a runner is created to run a grading pipeline, the integrity of the assignment configuration must be verified to ensure that a pipeline can be constructed. This process helps to reject invalid pipelines early on, allowing the teaching staff to receive early feedback on the issues of their assignment configurations.

Invalid assignment configurations may be passed to the grader if, for example, the YAML format is incorrect, or some keys are used incorrectly or are undefined. When that happens, a weaker version of the runner—a reporter, will be used instead to just report the involved syntax errors, called config errors, and instruct the teaching staff how to adjust for a later retry. In other words, the assignment configuration must be free of any obvious config errors to be executed in the first place.

Otherwise, a legal pipeline will be constructed based on the configuration. At run-time, each output-input pair of the pipeline stages will be further checked to make sure that the output of one stage can be provided to the input of the next stage. This will provide early warnings on mismatching types, which will allow for easier debugging of the pipeline itself. Note that this will trigger an exception that will be passed to the report as a pipeline error. This type of error is also shown in the report whenever a stage could not be executed or any other exceptions were thrown.

Notably, a runner extends a reporter to inherit all the reporting facilities and behaviors, because, during pipeline execution, the runner reports on the results of some pipeline stages and any config and pipeline errors occurred. Stages that report results to need to implement a Reportable interface as well, in which the runner will check and call its corresponding method to generate stage-specific report formats. After the runner finishes, with a method call, all the reporting information would be aggregated to a single report format.

One of the most important jobs of a runner is to track pipeline states that exist outside of a container. The runner uses temporary volumes (e.g. docker volumes) to store any files required to execute the assignment. These files are copied into the temporary volume via different pipeline stages opaque to the user to ensure that the submitted files cannot be changed after the fact.

When the pipeline stage is being executed, a container is created from the associated image, files and folders are mounted into the container as specified by the configuration, and the container script is executed. After the execution of the container, the container is removed, retaining any files which are written into the specific mount points. The use of container images in the pipeline means that the teaching staff could construct their grading workflow, as long as a valid pipeline can be constructed. This also allows the Teaching Staff to create their images, which is especially useful for custom grading schemes which are not provided by default.

In regards to how files are transferred between containers, we have opted to use the same volume wherever possible, reproducing a workspace-like environment when running a grading workflow. Also, reusing volumes can reduce the overhead of I/O operations between stages and therefore be more performant on large assignments with long grading pipelines. Other methods for file transferal, such as to always clone a volume whenever a stage is run, are also implemented opaquely to aid implementation for potential features.

Since a grading pipeline may create more than one volume during its lifetime, we have also included a mechanism to automatically remove the temporary volumes created by the runner after the pipeline finishes execution. Although this feature appears to be trivial, it is highly important as unused volumes can consume large amounts of disk space if not disposed of properly, which may cause a myriad of other issues.

One important objective stated for the design of the auto-grader is to maintain security and reproducibility. Therefore, one of the features we have implemented for the runner is to allow the teaching staff to specify whether network access is required for a grading pipeline. Disabling network access will further restrict the ability of foreign executables to damage the grading environment or the host system. At the same time, we understand the merits of allowing network access during the grading process, especially with workflows requiring the use of version control systems or dynamic dependency resolution. Therefore, this feature is opt-in by default.

As stated in Resource Management, runners must request a maximum amount of CPU time and memory resource to be reserved for a grading task, enforced by the containerization system. However, while testing an initial implementation of this feature, we realized that a naïve implementation will lead to resource underutilization, since grading tasks are unlikely to utilize all reserved resources. Therefore, we instead modified the resource reservation algorithm to allow resource overcommitment, which allows tasks to commit more resources than the system physically has, similar to implementations seen in virtual machine management systems. In addition, the resource reservation system also contains heuristics to always allocate new tasks onto the least-utilized system, keeping all grading tasks running at the highest efficiency while maintaining throughput.

To allow greater throughput, multiple runners can run simultaneously, especially useful in batch grading and lab grading. Although our original design mentioned capping the maximum number of graders at any instant, we have since revised the design to instead create as many runners as necessary, only blocking runner execution when the host system does not have enough resources to be reserved. This change allows better utilization of host resources, especially when the system is scaled up, while also preventing runners from consuming too much resources, slowing down other components of the host system.

Given that one of the major design goals of this project is to construct a system with high extensibility, the runner is no exception to this design paradigm. In fact, runners must be specialized based on the containerization framework used in the implementation, and the current implementation is specialized for Docker. However, since all runners are implemented under a common interface, migration to use another containerization framework (such as Vagrant) is relatively simple, only requiring the system maintainer to implement a small set of predefined interfaces. This design should allow the auto-grader to achieve greater extensibility as well as longevity. Also, certain simple computations could be done in local pipeline stages, for example score calculation. Since virtualizing the execution environment of runners is common but not mandatory under this design, this is also an option.

In order to fully exploit the computational power of modern systems, we have opted to use coroutines to implement runners and enable multitasking for performing different jobs at the same time. Unlike threads, coroutines use a multitasking paradigm called "cooperative multitasking", which means that individual tasks will not be allocated time-slices, instead requiring each task to voluntarily give up their computational time. The benefit of using this paradigm is that it allows for more performant multitasking, as less context switches are required between different tasks. In addition, because the scheduler does not involuntarily preempt tasks, the programmer is allowed more control over when tasks should prefer to keep running or preempt to allow other tasks to run.

Pipeline Implementation

The grader pipeline is a programmatic representation of a grading workflow. The implementation detail of the pipeline will be explained in this section.

Similar to the design of the pipeline, the assignment configuration file which defines the pipeline also takes heavy inspiration from GitLab's Continuous Integration feature. Defined by a YAML file, the pipeline configuration specifies the various properties of pipeline stages, such as the compilation flags and input/output files for a compilation stage. In addition, the configuration file also allows teaching staff to specify additional stages which may not be implemented natively in the system, making adding or using new and custom stages much easier.

The structure of a stage definition encompasses:

1. Config format specific to this stage
    a. The YAML object (as key-value pair) specification defining all options to configure this stage that are exposed to the user of the stage from the assignment configuration YAML. For example, the compile stage takes the input files and an output executable name.
    b. This is implemented as a Kotlin data class to be nested inside the stage class.
2. Parsing factory method
    a. This should define how the stage should be generated according to the provided config format from the assignment configuration

   b. This is implemented as a static factory method inside a Kotlin companion object implementing a ConfigParsable interface, in which this companion object is nested inside the config format data class (so it is nested twice).
   c. This is how a stage is created from the assignment configuration, and will be accessed by a configuration parser.
3. Stage class
   a. The name of this class is the key in the assignment configuration for specifying the use of this stage.
   b. The constructor of this stage should sufficiently initialize properties of the stage, including those that specify how to execute the stage.
4. Execute method of the stage class
   a. In general, this method is called when executing this stage, so all needed computations on the inputs can be defined here. For containerized stages, the execute method is mostly just wrapping around a command line script to execute, so those stages only have to specify the command to run. Currently we support Bash scripting.
   b. To aid command scripting and possibly reporting from container execution results, commands used in containers can be built using a simple domain-specific language we develop. This language encompasses the ability to automatically pipe and store the standard output, standard error, exit code of a given command to Bash variables and/or text files, among other common scripts, allowing us to retrieve these afterwards by parsing the execution logs to construct required reports of this stage.

As mentioned in Runner Implementation, a stage can optionally be Reportable, meaning that this stage upon completion will emit reporting information about the execution, to be collected by the guiding grader runner. Such a stage shall further specify:

5. Report format specific to this stage
   a. The JSON object specification of what this stage should emit. For example, the compile stage reports a “isSuccess” Boolean and all lines of the stderr.
   b. This is implemented as a Kotlin data class nested inside the stage class.
6. Report generation method
   a. This is similar to the parse method, but in the reverse direction: This method takes the state of the stage (mostly the results from the execution logs) and specifies how to generate the report JSON for this stage.
   b. This is an abstract method for the Reportable interface to enforce implementation.

To promote modularity, all of these definitions are encouraged to be specified in a single source file using nested classes, so that its sole inclusion can already enable the use of this stage in the assignment configuration.

The pipeline generation algorithm making use some of the above defined stage components is:

1. The Configuration Parser tries to read and parse the assignment configuration YAML, then interpret it by searching for a matching pipeline stage class name with the used YAML stage top-level key. Global settings in the YAML is also parsed to create a context so that the pipeline can be run using user-specified parameters. Any critical errors will be reported and this process will early return.
2. For each matched pipeline stage, the parse factory method is called on the config format for the stage to construct a lazy pipeline stage. Putting them in a list, they form a lazy pipeline. The reason for using laziness is that not all data required to initialize the pipeline stage instance are ready at parse-time, notably the submissionId, because an assignment configuration is independent of particular submissions. Later when this is ready and is patched to the partial context to produce the full context, the lazy pipeline will be transformed to a pipeline by applying the context to each of the lazy pipeline stage. Laziness is implemented using a lambda function, wrapping the recipe for the stage with a function parameter of the context.
3. Finally, the pipeline is ready to be processed by a runner.

To accommodate for new stages which are not provided by the system, the grader is able to dynamically resolve stage names when parsing the configuration. The parser achieves this by collecting a list of known stages at runtime, then using the list when resolving class names of the stages and their components in the configuration by reflection. The rationale for this is to allow additional stage definitions to be included in the stages search path, allowing system maintainers to add new stages without the need to recompile the grader application. This configuration design also allows a future implementation to further resolve stages by dynamically downloading images from a repository and creating stages on-the-fly.

Since pipelines make up a large part of the runner execution and often dictates the critical path of the runner, extra care has been taken to implement the pipeline to ensure the pipeline is executed in a performant manner. This is achieved via the use of containerized images, which are templates for a container. Container images are built when a grading stage is first run, allowing subsequent usages of the stage to reuse the image, drastically reducing the time required to setup a containerized environment. In addition, using images also decreases container startup time, because Docker containers use an overlay layer to track container filesystem changes, meaning that no copy operations are required for the startup of a container, and removal of the container incurs little to no overhead. Finally, our pipeline implementation makes heavy use of image tagging to track the images used by a specific assignment, allowing easy cleanup of obsolete images whenever an assignment results are finalized, as well as caching and reusing of images whenever the tag matches a previously built image.

Considering that the auto-grader is an always-running background process, fault-tolerance is an important aspect when implementing the pipeline, especially considering that we cannot predict what assignments or student submissions will be graded using our system. Therefore, our system implements error handling and recovery in various parts of the pipeline, with two specific features aiding the error recovery process.

Firstly, all pipeline stages implement the ability to recover from any failure. Since pipeline stages are backed by containers, common failures in a pipeline stage includes the container exiting with a non-zero exit code, and the container execution timing out. All failures received by the pipeline stage will be logged by the runner to allow for later inspection by other components of the Auto-Grader. While the next logical step would be to terminate the pipeline immediately, it is important to note that some failures are actually expected, such as a container exiting with non-zero code due to differences between the expected output and the output emitted by the student submission. Therefore, pipeline stages can optionally override the default behavior and instead ignore certain classes of failures, allowing the pipeline to keep running.

Secondly, there are some errors that cannot be handled solely within a pipeline stage, such as if the containerization engine disconnects from the grader and reconnection fails. In cases like these, the pipeline will propagate the error to the runner, and the runner will immediately terminate the current pipeline, reschedule the grading task to be dispatched, and create a new runner for the task to run on. In order to achieve this workflow, an associative map of assignments to pipeline definitions is used as a cache to rapidly create pipelines without the need to reparse the configuration. A side benefit to this approach is that normal submissions can also utilize the cache to retrieve pipeline definitions, speeding up the grading process for normal workflow.

The major implemented pipeline stages are briefly listed below:

- Pre-grading stages
    - FileStructureValidation (compare the submission with submission template)
    - DiffWithSkeleton (compare the submission with skeleton code for sanity check)
    - Compile: CppCompile, JavaCompile
- Grading stages
    - StdioTest (Standard Input/Output-based testing)
    - Valgrind (memory checking)
    - Make (GNU Make)
    - Copy stages: CopyHostToVolume, CopyVolumeToVolume, CopyVolumeToHost (copy files between Grader and pipeline stage volumes)
- Post-grading stages
    - Score (compute scores for this submission)

Technology Stack

Runner Containerization Engine: Docker

The Auto-Grader Subsystem is designed to spawn individual and isolated instances of grading processes called Runners. Therefore, we have opted to use Docker [35] to enforce the isolation of the various grading environment.

Compared to Virtual Machines (VMs), containers have a much faster spin up time due to the lack of a full Operating System. This means that the duration of a grading task is reduced. In addition, containers also tend to be much smaller in size, containing only the required components of the image. This will both decrease the disk space requirement of the grader task, which may mean it could be run on a faster media type, and further reduce the duration of the grading task.

We have specifically selected Docker due to several benefits it provides over its alternatives. Firstly, Docker is one of the most popular containerization solutions alongside LXC, meaning that community support for Docker is readily available. Secondly, Docker is supported on all major operating systems and architectures. This means that the grader can be extended to support assignments which require other execution environments, such as VBA scripts or ARM assembly. Finally, Docker provides built-in volume management functionality, meaning that instead of manually performing isolation of files between different containers, an out-of-the-box solution is already provided, reducing the risk of errors from a faulty isolation implementation.

It is important to note that although Docker is selected for the project, the grader's flexibility allows other containerization options to be provided in-place of Docker. This means that if other containerization or virtualization options are required in the future, it will only require minimal changes to enable the functionality.

Auto-Grader Language Ecosystem: Kotlin/JVM

Given the always-running nature of the Auto-Grader, the implementation language and accompanying frameworks must fulfill strict requirements, including high stability, strict correctness, reasonable performance, and ease of development. Kotlin/JVM [36], which runs on the Java Virtual Machine (JVM) [37], is chosen as the language and tool for implementing the Auto-Grader Subsystem, selected primarily because it balances the aforementioned requirements well.

Firstly, regarding stability, Kotlin/JVM mainly achieves this by leveraging its interoperability with Java to reap the rich and long-standing library ecosystem to its use, while also leans on the JVM to provide a production-tested system. By allowing Java libraries to be used drop-in into Kotlin code, it enjoys the benefit of using highly tested libraries within the implementation, improving both code reuse and stability. Moreover, this also contributes to the ease of development, as it allows existing developers familiar with JVM languages to use libraries which they are already familiar with, reducing the overhead of adapting to new frameworks.

However, it is also important to ensure that the code written by developers are also stable, which is where Kotlin's correctness requirement is advantageous. Firstly, as a statically typed language, Kotlin offers a wide range of compile-time type checks which eliminates potential errors often seen in weakly typed languages. In addition, Kotlin's type system explicitly differentiates nullable with non-nullable types, which eliminates the accidental use of null values and solving a class of errors commonly associated with C, C++, and Java.

While many languages trade speed for correctness, Kotlin maintains a good balance between the two. Although the Kotlin Compiler injects runtime checks into the bytecode to enforce various safety features mentioned in previous paragraphs, the compiler also optimizes these checks away when it can definitively deduce that the checks are unnecessary.

In addition, the JVM can also provide the required performance using the built-in functionality of on-the-fly bytecode optimization, as well as command-line parameters to tune JVM's responsiveness in various circumstances. These potential points of optimization allow JVM's performance to match most garbage-collected languages [38], only trailing behind compiled languages by a factor of 2. In this light, the performance offered by compiled languages would not be necessary, whereas the one order-of-magnitude slower performance of dynamically typed languages may result in an observable slowdown.

Finally, to further elaborate on the ease of development, Kotlin not only provides Java interoperability as stated above, but there are also libraries written specifically for Kotlin which attempt to bring the advantages of other languages into Kotlin. For example, Go [39] has Goroutines built into the language to offer a lightweight alternative to threads. Kotlin follows this trend by also offering KotlinX Coroutines [40] as an alternative to the Java Threading Library, allowing developers to choose between the backward-compatible approach of handling concurrency or a language-native approach.

It is important to note that candidates including C/C++, Rust [41], Go, Java, Scala [42], Python, JavaScript are equally capable of implementing the Auto-Grader Subsystem. However, none of the aforementioned languages and their respective ecosystem allows for the balance between stability and ease-of-use as offered by Kotlin.

2.3.3 Testing

As part of the test suite, the Auto-Grader subsystem contains unit tests to show the correctness of the subsystem's behavior. As stated in the testing overview, the test cases aim to assert whether the functional requirements of the subsystem are met with the implementation.

Kotlin

The Auto-Grader Subsystem utilizes JUnit [43] for unit testing. Framework-specific testing libraries are also used to complement the aforementioned testing frameworks.

2.4 Deployment

2.4.1 Docker

All services are implemented with containerization in mind, such that all services are built as Docker images and deployed in Docker containers. Doing so has many advantages, such as all required software packages at their tested versions and configurations come with the images, simplifying installation and maintenance. Using container deployment strategies like Docker Compose [44] or Kubernetes [45] allows for self-healing (restarting stopped containers) and horizontal scaling.

All services in the Submission Subsystem can be horizontally scaled if required by demand from networking traffic. The Admin UI Server and the Grader Daemon will not be scaled up, since the prior has low demand and the latter has built-in capability to scale up via the use of cooperative multitasking and dynamic dispatching of containerization engines.

2.4.2 Deployment Testing, UAT and Demonstration

Kris has provided us with a testing server. We are conducting deployment testing, UAT and demonstration on his server. A mocked CAS server is written and deployed onto the server to test out CAS integration and for quick swapping between different user profiles.

2.5 Evaluation

We use three methods to evaluate the completion of our system. Firstly, we use the Functional Requirements and Non-Functional Requirements to evaluate the degree of completion of the project. Secondly, we use UATs to ensure that our users are able to navigate and work with the system with ease. Finally, we reference the objectives stated in Objectives to determine whether the goals of the project are met.

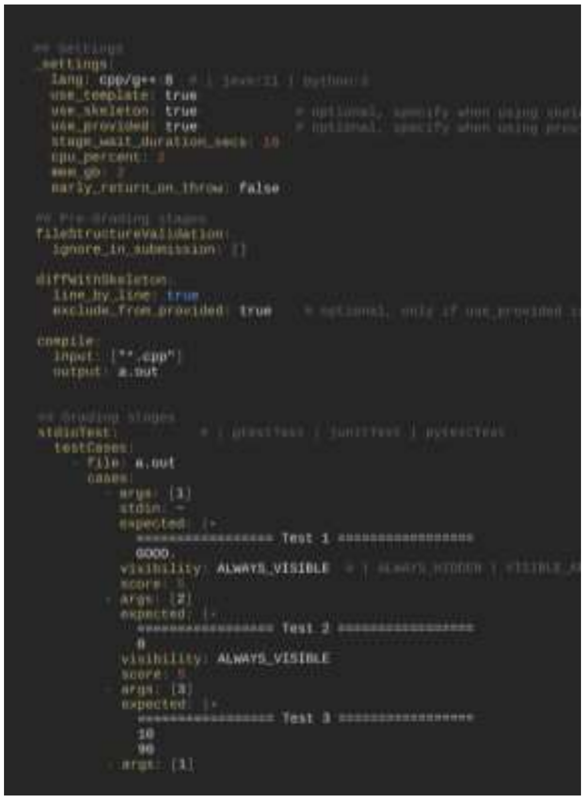
3. DISCUSSION

3.1 User Acceptance Test

To ensure project quality, we sought to use real inputs to test the system for faults and bad performance. We have requested for student submissions from COMP2012 course offerings from Prof. Desmond Tsoi. Under strict privacy consideration, we took all data from students serious and ensured none of their work were leaked to the public.

3.1.1 COMP2012 Fall 2019 PA3

We have experimented with student submissions of Fall 2019 COMP2012 programming assignment 3 during development. The reason why we have used this assignment specifically is that Wallace, the Teaching Associate in charge, has created a semi-automated grading program to assist in grading console-based assignments. However, he needed to manually edit the source code of his grading program numerous times during grading and tally up the scores on his side. By using student submissions of the same assignment, we can cross-check with actual grades awarded by Wallace and ensure our developed system works with a common grading situation.



```
## Settings
_settings:
  lang: cpp/g++:8  # | java:11 | python:3
  use_template: true
  use_skeleton: true                  # optional, specify when using skel
  use_provided: true                  # optional, specify when using prov
  stage_wait_duration_secs: 10
  cpu_percent: 2
  mem_gb: 2
  early_return_on_throw: false

## Pre-Grading stages
fileStructureValidation:
  ignore_in_submission: []

diffWithSkeleton:
  line_by_line: true
  exclude_from_provided: true    # optional, only if use_provided i

compile:
  input: ["*.cpp"]
  output: a.out

## Grading stages
stdioTest:             # | gtestTest | junitTest | pytestTest
  testCases:
    - file: a.out
      cases:
        - args: [1]
          stdin: ~
          expected: |-
            ================== Test 1 ==================
            GOOD.
          visibility: ALWAYS_VISIBLE  # | ALWAYS_HIDDEN | VISIBLE_A
          score: 5
        - args: [2]
          expected: |-
            ================== Test 2 ==================
            8
          visibility: ALWAYS_VISIBLE
          score: 5
        - args: [3]
          expected: |-
            ================== Test 3 ==================
            10
            90
        - args: [1]
```

Figure 10 Configuration Sample

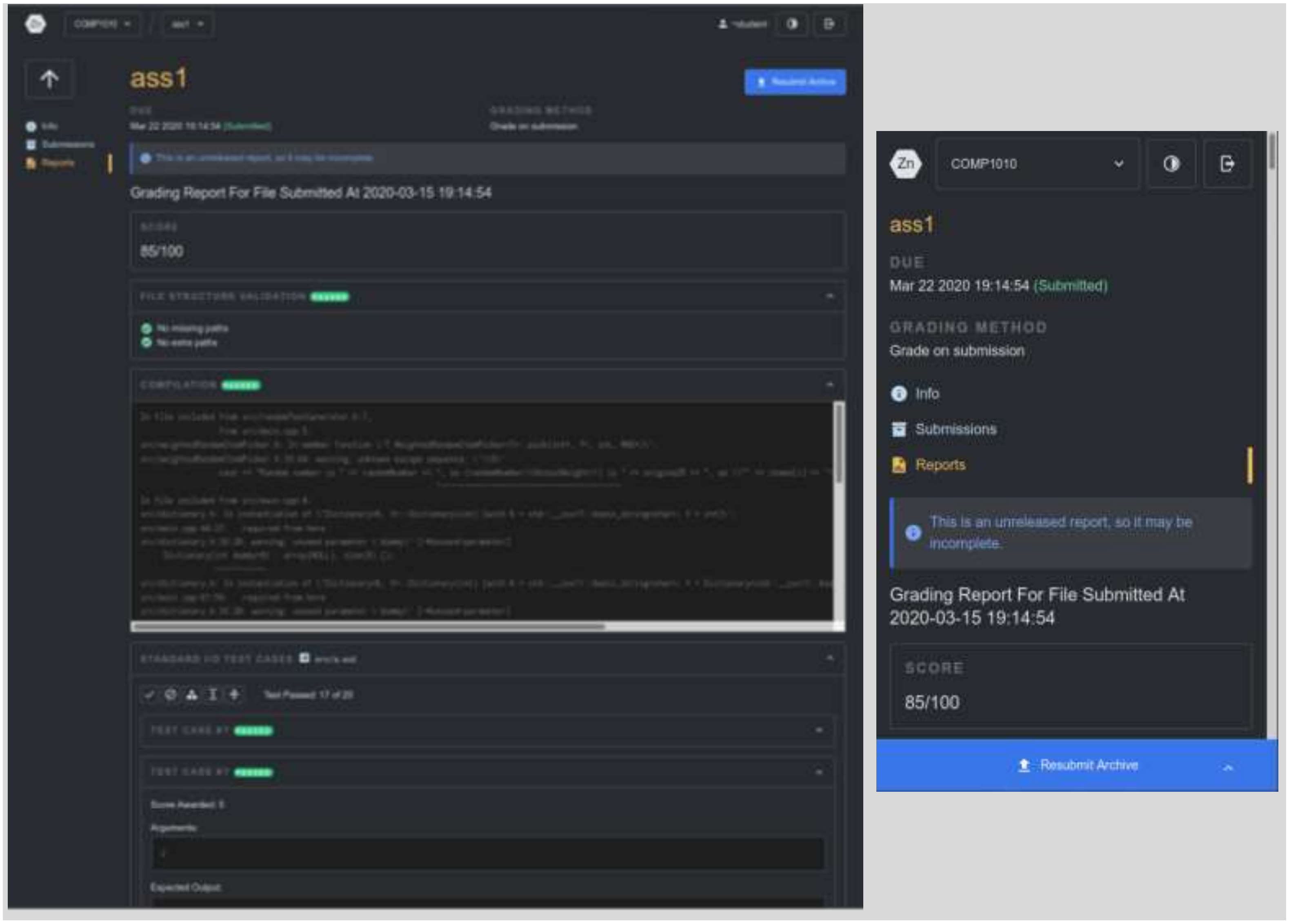
With the flexibility of our system design, our system is also compatible with Wallace's style of grading. With his grading package uploaded onto our system with an Assignment Configuration

(Figure 10), our system accurately runs standard I/O test cases specified in his assignment against student submissions.

We tested this assignment configuration against several cases: a normal submission, a submission that does not compile, and an erroneous submission that never exits. All of these can be properly handled by the Auto-Grader by limiting the execution within a timeout duration and checking for errors.

Submissions could also be executed in parallel. Currently, with our test environment, we can parallelize two submissions at the same time, each taking around a minute. We discovered that with our test environment, the bottleneck mainly lies in the available memory.

A generated report from this experiment viewed from the web interface is displayed below.



ass1

Grading Report For File Submitted At 2020-03-15 19:14:54

85/100

COMP1010

ass1

DUE

Mar 22 2020 19:14:54 (Submitted)

GRADING METHOD

Grade on submission

Info

Submissions

Reports

This is an unreleased report, so it may be incomplete.

Grading Report For File Submitted At 2020-03-15 19:14:54

SCORE

85/100

Resubmit Archive

Figure 11 Responsive View of a Grading Report in Dark Mode (L: Desktop Screen, R: Mobile Screen)

3.1.2 COMP2012 Spring 2020 PA1

A class-scale UAT involving live submissions was held in late April 2020 to test out the entire submission procedure across the Submission Subsystem and the Auto-Grader Subsystem. The submissions came from the 2012 spring offering programming assignment 1, where students have already been graded officially by course staff, so this exercise served as a test for our system, and students volunteered to help. A short UAT survey was conducted with optional submissions. Expected targets of the UAT are as follow:

- Catch unsmooth UX problems by observing and monitoring logs, as well as from survey feedback.
- Confirm the system runs fine across a large sample of different inputs.
- Ensure system performance when facing a sizable crowd of users.

Overall, feedback from the survey has been generally positive. A small fraction of survey respondents also filled in comments. Dissecting the general categories of comments, there are the following:

- Praises on the helpfulness of generated reports
- Critiques and confusion on report generation as well as the display of report details

We have discovered a fatal bug during the UAT relating to the IPC service in the Submission Subsystem. One of the generated reports occupied an unusually large size (>30MB) compared to other normal reports (<250KB), due to the incorrect source code implementation of the student. This caused the memory consumption of the PHP service to exceed the maximum allowed. The failing IPC Job, in turn, created an infinite loop, sending out a large number of repeated notifications incorrectly.

This issue was not considered during the design process and was immediately noted in our project issue tracker. A derivative problem of such an unusually large report also prompts the reconsideration of the report rendering in the UI, as currently, showing all details without truncating may pose threat to the performance of the client browser and networking resource of the REST API.

Two rating questions have been asked with 24 respondents:

- How would you rate your experience on ZINC?
- How much do you prefer using ZINC over CASS/Canvas for programming assignments?

For both questions, with the range between 1–5 stars, the average ratings were both 4 stars.

Overall, the UAT has been a success, by considering the fatal case was 1 out of 25 participated users or 1 out of 41 successful submissions.

3.2 Survey Findings

From the survey results, we can see there may be little impact ZINC can bring to these teaching staff for traditional assignment grading automation as they may have already spent effort into developing automation tools if possible.

ZINC must develop other enticing features to win over teaching staff, to establish a sustainable project with a pool of users of continuous development in the future.

Overall, judging from the responses to ZINC features, there seems to be demand on quality of life grading-related features that offload work from the teaching staff, as well as flexible submission collection schemes with immediate feedback that assists students in their assignments.

With some of these features already in development, the results of this survey have allowed us to reprioritize some of our existing works, to accommodate requirements from the department users.

3.3 Potential Additional Features

Several potential features to the system are listed below as a reference on the future direction of the project.

Online Appeal

Students upon receiving the grading report can initiate an online appeal of an assignment using his/her last submission. Our system can serve as an official communication channel between a student and a teaching staff of a specific course on a specific assignment, by providing mechanisms such as pinpointing a specific part of an assignment and allow the two parties to discuss on an online discussion thread and batch appeal management for teaching staff.

Assignment Grouping

Currently, we support single student submissions and require students to submit individually even if they formed a group in a course. By enabling group management and submission sharing among group members, managing group assignments can be made easier for teaching staff.

Extension Requests

If a student encounters special conditions that require an extension of the deadline for an assignment, our system could also support that by letting students submit requests and teaching staff accept/reject requests, as well as keeping deadline changes of individual students recorded.

More Grader Stages

Code coverage grading stages and Git integration to retrieve files from remote repositories, can all be added as options for Teaching Staff to compose the pipeline – and ideally, writing new custom stages and sharing them for reuse. Refer to Table 2 for a list of potential pipeline stages.

A scripting API may also be added to allow power users to create their custom pipeline stages, further extending workflows that are executable by the Auto-Grader.

3.4 Interview with TA

In mid-April, we had an opportunity to present ZINC to a Teaching Assistant of COMP2012 of Spring 2020, Kelvin. Unlike the Interview with Teaching Associates at the start of the project, this meeting allowed us to show our work and receive feedback on the current state of ZINC. Kelvin also provided first-person opinions of how ZINC compares with his current grading workflow.

The grading workflow he created and was using was a self-made Python script that calls system Bash commands as subroutines and is configurable by a custom JSON-based configuration file. Commonly used commands are built-in, for example, compilation, console-based testing, and optional Valgrind memory checking. Logs are automatically emitted into folders per submission. This is a local workflow on a command-line interface and is supposed to be just for his personal use.

We presented our Submission UI, explained our Auto-Grader assignment configuration structure and pipeline stage structure to him, and his feedback was that the feature sets and extensibility are sufficient to him, and would fulfill his needs. However, he preferred to also have a command-line interface of our system, especially for the Auto-Grader.

4. CONCLUSIONS

In this project, we have successfully implemented features required for the users, under our important design goals of flexibility, modularity, extensibility, and other non-functional requirements.

The Submission Subsystem code base comprises of the API service, Student UI, and Admin UI. The code is programmed horizontal scalability in mind, so each functional service is encapsulated in its container. The submission system can collect assignments from students, allow administrators and the teaching staff to manage assignments and related resources, synchronizing enrollment records with CS System, sending out notifications, as well as conducting IPC with the Auto-Grader.

The Auto-Grader Subsystem comprises of the daemon, runners, and pipelines. Strong isolation between these components allows them to specialize and separate implementation concerns. The daemon coordinates communication and handles IPC jobs, the runners are fully responsible for grading a single submission, whose execution details are specified in the corresponding pipeline. The pipelines are designed to be highly flexible and composable according to the needs of the particular assignment, making adding new features on top of this infrastructure easy.

Together the entire system fulfills the originally set objectives, including satisfying each user requirement and conducting the UAT.

For future work, since this project could become part of the new CSE assignment collection and auto-grading system, further integration and improvements would be required. Several interesting features could be added as the project may get continued by other students or staff.

We are grateful to have participated in this project, to contribute to a better education experience and a new system that potentially touches more than thousands of students. This project has always been a well-suited experience for training our project management skills and teamwork, as well as for us to learn deeper in software engineering and system design in a hands-on manner.

We would like to thank our advisor for giving this opportunity to us and supporting us throughout the process. We also would like to thank Kris, Chris, ITSC, and other people who have contributed to this project by accepting our interviews, completing surveys, and assisted in the setting up of the server and connection to relevant services.

REFERENCES

[1] HKUST CS System, "CS System - CASS," 8 Aug 2018. [Online]. Available: https://cssystem.cse.ust.hk/UGuides/cass/instructor.html. [Accessed 06 Aug 2019].

[2] A Rensselaer Center for Open-Source Project, 10 Sep 2019. [Online]. Available: https://submitty.org/. [Accessed 16 Sep 2019].

[3] Canvas, "Canvas LMS Community," [Online]. Available: https://community.canvaslms.com/community/answers/guides/canvas-guide.

[4] "Autogradr," [Online]. Available: https://www.autogradr.com.

[5] X. Liu, S. Wang, P. Wang and D. Wu, "Automatic Grading of Programming Assignments: An Approach Based on Formal Semantics," [Online]. Available: https://faculty.ist.psu.edu/wu/papers/autograder.pdf.

[6] https://www.postgresql.org/, "The PostgreSQL Global Development Group," 16 Sep 2019. [Online]. Available: https://www.postgresql.org/. [Accessed 16 Sep 2019].

[7] StackOverflow, "Developer Survey Results 2019," StackOverflow, [Online]. Available: https://insights.stackoverflow.com/survey/2019. [Accessed 2019].

[8] MongoDB, Inc., "The most popular database for modern apps | MongoDB," 2019. [Online]. Available: https://www.mongodb.com/. [Accessed 16 Sep 2019].

[9] The Apache Software Foundation, "Apache Cassandra," 16 Feb 2019. [Online]. Available: http://cassandra.apache.org/. [Accessed 16 Sep 2019].

[10] "Redis," 2019. [Online]. Available: Redis. [Accessed 16 Sep 2019].

[11] Apereo, "CAS," [Online]. Available: https://www.apereo.org/projects/cas.

[12] IETF, "RFC 4511 - Lightweight Directory Access Protocol (LDAP): The Protocol," [Online]. Available: https://tools.ietf.org/html/rfc4511.

[13] IETF, "RFC 7519 - JSON Web Token (JWT)," [Online]. Available: https://tools.ietf.org/html/rfc7519.

[14] ZEIT, "Next.js," [Online]. Available: https://nextjs.org/.

[15] Yarn, "Yarn Package Manager," [Online]. Available: https://yarnpkg.com/.

[16] Theme UI, "Theme UI," [Online]. Available: https://theme-ui.com/.

[17] Tiny Technologies Inc., [Online]. Available: https://www.tiny.cloud/.

[18] SpryMedia Ltd, "DataTables," [Online]. Available: https://datatables.net/. [Accessed 2020].

[19] Node.js Foundation, "Node.js," 16 Sep 2019. [Online]. Available: https://nodejs.org/. [Accessed 16 Sep 2019].

[20] The PHP Group, "PHP: Hypertext Processor," 16 Sep 2019. [Online]. Available: https://www.php.net. [Accessed 16 Sep 2019].

[21] Python Software Foundation, "Welcome to Python.org," 2019. [Online]. Available: https://www.python.org/. [Accessed 16 Sep 2019].

[22] W3Techs, "Usage of server-side programming languages for websites," 4 August 2018. [Online]. Available: https://w3techs.com/technologies/overview/programming_language/all.

[23] Laravel, "Laravel," [Online]. Available: https://laravel.com/.

[24] Google, "Google Trends - Laravel, Code Ignitor, CakePHP, Symfony and Yii," [Online]. Available: https://trends.google.com/trends/explore?geo=US&q=Laravel,Code%20Ignitor,CakePHP,Symfony,Yii.

[25] Facebook Inc., "React – A JavaScript library for building user interfaces," 2019. [Online]. Available: https://reactjs.org/. [Accessed 16 Sep 2019].

[26] E. You, "Vue.js," 14 Sep 2019. [Online]. Available: https://vuejs.org/. [Accessed 16 Sep 2019].

[27] Google, "Vue, React, Angular - Explore - Google Trends," Google, [Online]. Available: https://trends.google.com/trends/explore?cat=31&date=2015-07-04%202019-08-04&q=Vue,React,Angular. [Accessed 4 August 2019].

[28] "Nuxt.js - The Vue.js Framework," 14 Sep 2019. [Online]. Available: https://nuxtjs.org/. [Accessed 16 Sep 2019].

[29] The GraphQL Foundation, "GraphQL | A query language for your API," 15 Sep 2019. [Online]. Available: https://graphql.org/. [Accessed 16 Sep 2019].

[30] Meteor Development Group Inc., "Apollo GraphQL | Learn about the Apollo platform: Client, Engine, GraphQL servers, GraphQL support, and more.," 16 Sep 2019. [Online]. Available: https://www.apollographql.com/. [Accessed 16 Sep 2019].

[31] Facebook, "GraphQL - A data query language," [Online]. Available: https://engineering.fb.com/core-data/graphql-a-data-query-language/.

[32] Elastic NV, "Elasticsearch," [Online]. Available: https://www.elastic.co/elasticsearch.

[33] PHPUnit, "PHPUnit - The PHP Testing Framework," [Online]. Available: https://phpunit.de/.

[34] Mockery, "Mockery Docs," [Online]. Available: http://docs.mockery.io/en/latest/.

[35] Docker Inc., "Enterprise Container Platform | Docker," 10 Sep 2019. [Online]. Available: https://www.docker.com. [Accessed 16 Sep 2019].

[36] The Kotlin Foundation, "The Kotlin Foundation," 26 Aug 2019. [Online]. Available: https://kotlinlang.org/. [Accessed 16 Sep 2019].

[37] Oracle, "Java | Oracle," 27 Aug 2019. [Online]. Available: https://www.java.com/. [Accessed 16 Sep 2019].

[38] Benchmarks Game, "Which programs are fastest?," [Online]. Available: https://benchmarksgame-team.pages.debian.net/benchmarksgame/which-programs-are-fastest.html.

[39] Google Inc., "The Go Programming Language," 2019. [Online]. Available: https://golang.org/. [Accessed 16 Sep 2019].

[40] Kotlin, "Kotlin/kotlinx.coroutines: Library support for Kotlin coroutines," [Online]. Available: https://github.com/Kotlin/kotlinx.coroutines. [Accessed 2019].

[41] "Rust Programming Language," 2019. [Online]. Available: Rust Programming Language. [Accessed 16 Sep 2019].

[42] École Polytechnique Fédérale, "The Scala Programming Language," 16 Sep 2019. [Online]. Available: The Scala Programming Language. [Accessed 16 Sep 2019].

[43] The JUnit Team, "JUnit 5," 13 Sep 2019. [Online]. Available: https://junit.org/. [Accessed 16 Sep 2019].

[44] Docker, "Overview of Docker Compose," [Online]. Available: https://docs.docker.com/compose/.

[45] The Kubernetes Authors, "Kubernetes," [Online]. Available: https://kubernetes.io/.

[46] Rensselaer Center for Open-Source, "Submitty," 2019. [Online]. Available: https://submitty.org/.

[47] Google, "google/diff-match-patch: Diff Match Patch is a high-performance library in multiple languages that manipulates plain text.," [Online]. Available: https://github.com/google/diff-match-patch.

[48] Microsoft, "Monaco Editor," [Online]. Available: https://microsoft.github.io/monaco-editor/.

APPENDICES

Project Planning

Division of Labor

Tasks are divided between the team members as listed in Table 3, under these nicknames:

- Daniel: CHEUNG, Daniel
- Dipsy: WONG, Yuk Chan
- David: MAK, Ching Hang
- Bryan: CHUN, Hiu Sang
- Desmond: Dr. TSOI, Desmond Yau-Chat

Task	*Daniel*	*Dipsy*	*David*	*Bryan*	*Desmond*
Meeting minutes	🟦	🟦	🟦	🟨	🟥
Conduct user survey	🟦	🟦	🟦	🟨	🟥
Web platform UI	🟨	🟦	🟥	🟥	🟥
Webhook	🟦	🟨	🟥	🟥	🟥
Submissions API	🟦	🟨	🟥	🟥	🟥
Grader Daemon	🟥	🟥	🟨	🟦	🟥
Grader Runner	🟥	🟥	🟨	🟦	🟥
Grader Stages	🟥	🟥	🟨	🟨	🟥
Grader Configuration and Reporting	🟥	🟥	🟦	🟨	🟥
Grader Virtualization	🟥	🟥	🟨	🟦	🟥
Design Inter-Process Communication	🟨	🟦	🟦	🟨	🟥
Setup Redis	🟨	🟦	🟦	🟦	🟥
Setup Database	🟨	🟦	🟦	🟦	🟥
Setup File System	🟦	🟦	🟦	🟨	🟥
Setup Decompressor	🟦	🟦	🟦	🟨	🟥
Setup Mailer	🟦	🟨	🟦	🟦	🟥
Authentication	🟦	🟨	🟥	🟥	🟥
Roles and permissions	🟨	🟦	🟥	🟥	🟥
Continuous Integration	🟦	🟥	🟨	🟥	🟥
Deployment environment testing	🟦	🟦	🟦	🟨	🟥
User acceptance testing	🟦	🟨	🟦	🟦	🟥
Proposal	🟨	🟦	🟦	🟦	🟥
Monthly reports	🟦	🟦	🟦	🟨	🟥
Progress report	🟦	🟦	🟦	🟨	🟥
Final report	🟨	🟦	🟦	🟦	🟥
Short Video	🟨	🟦	🟦	🟦	🟥
Presentation	🟦	🟦	🟨	🟦	🟥

Table 3 Division of Labor (Color Code: 🟨 Accountable, 🟦 Responsible, 🟥 Consulted)

For every task, there is one person assigned to be primarily accountable for task completion, several people responsible for the contribution, and some others consulted for advice during the process.

Gantt Chart

The Gantt Chart Figure 12 is recorded in weeks until the Final Report deadline.

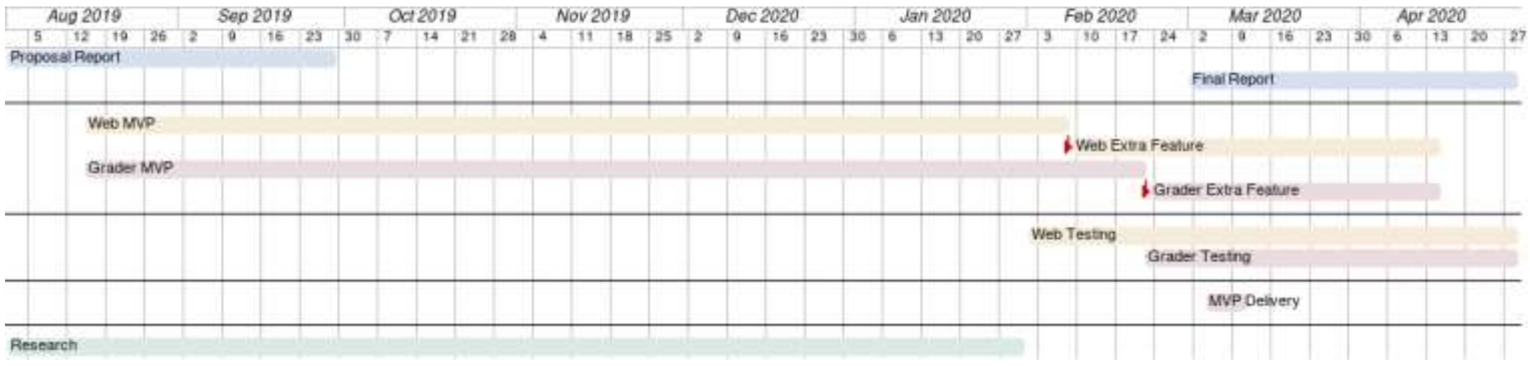


Figure 12 Gantt Chart (Color Code: Reports, Submission Subsystem, Auto-Grader Subsystem, Testing, and Research)

Interaction with the Submission Subsystem

Figure 13 displays the sequence flow of the relevant backend systems when a student submits an assignment to the system.

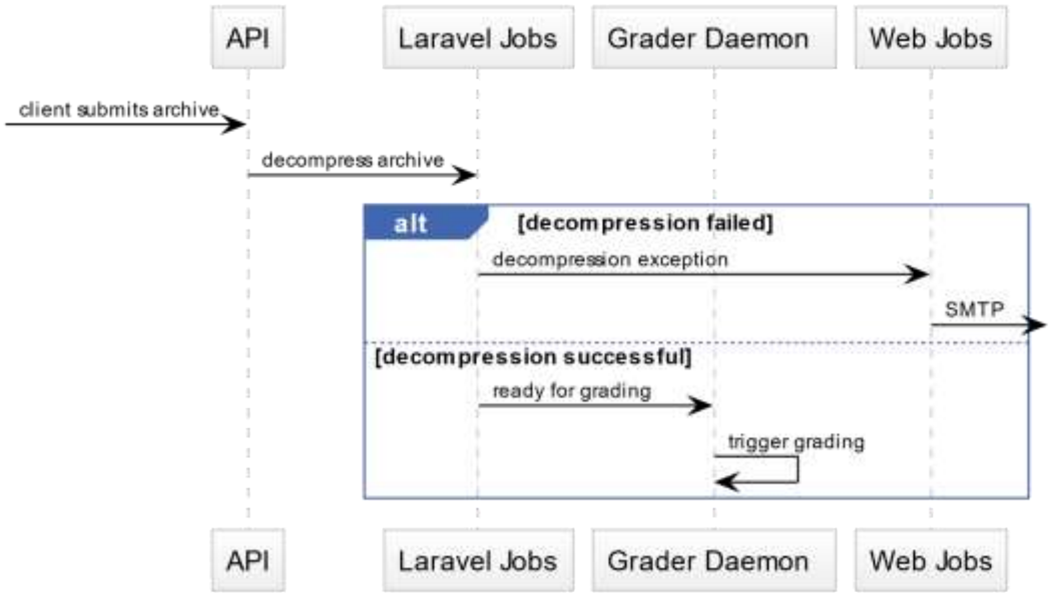


Figure 13 Client Submission Sequence Diagram

As seen from above, unless a failure occurs during the decompression stage, the Decompressor will publish a message to the Grader Daemon. On the other hand, if the decompression fails for any reason, the Mailer will send an email, using the Simple Mail Transfer Protocol (SMTP), to the student, informing them that the system cannot process their submission.

Figure 14 displays the flow when a grading task is started, either due to an assignment being just submitted to the system, or the grading task is scheduled to take place at the time.

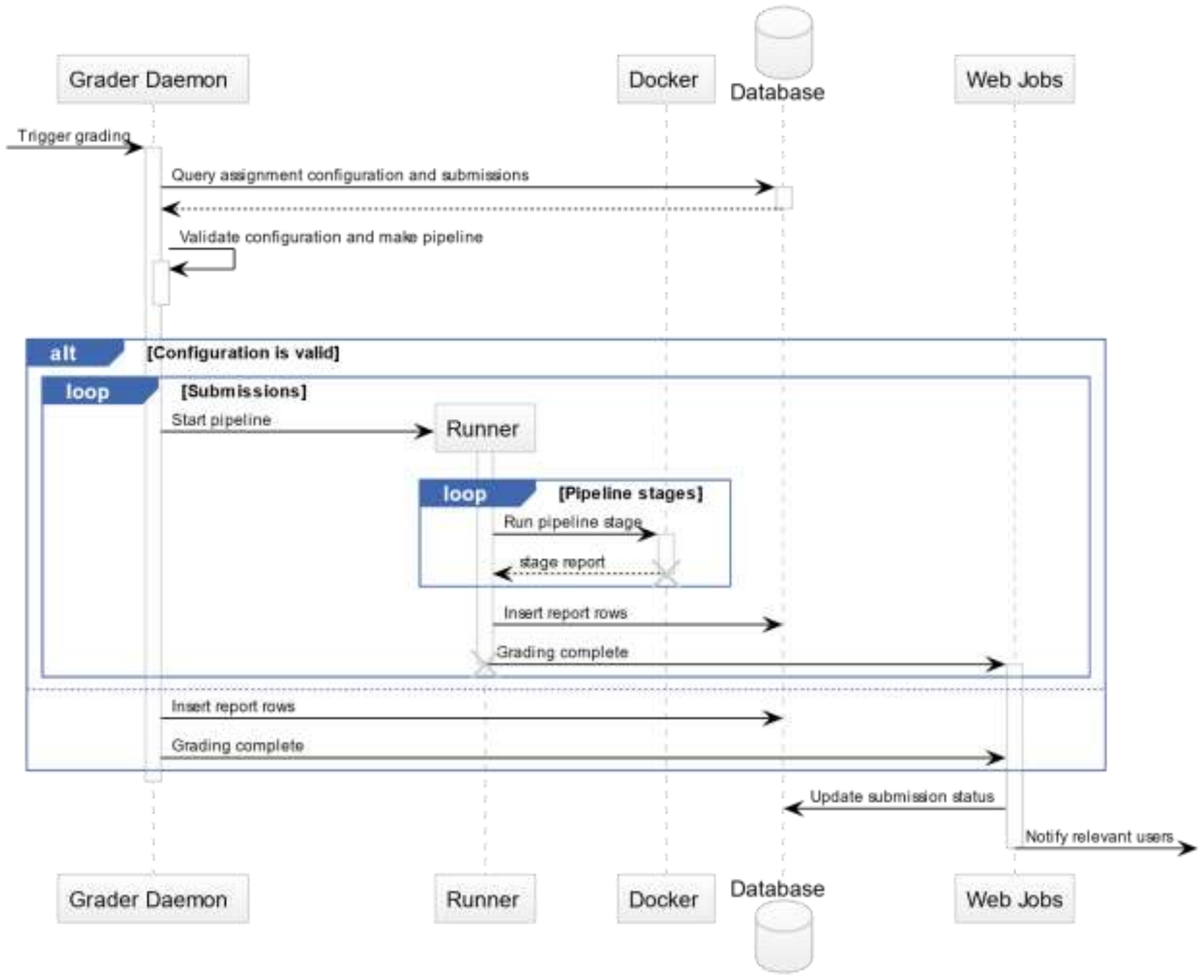


Figure 14 Grading trigger sequence diagram

Effectively, when the Daemon receives the message to start a grading task, the task will be allocated to a Grader Runner. The runner will then join all pre-grading and grading into a pipeline in that order and execute each stage sequentially.

In general, if any exceptions arise during the pre-grading or grading process, the entire process can be made stopped, and an email will notify the student that their submission has failed. Note that failed test cases are not treated as exceptions, as they do not affect subsequent grading tasks.

Grading Pipeline Example

Inputs
An COMP2012-L1 PA1 unzipped submission
Config: **- Enable immediate feedback** - Do not show hidden test cases
Sep 1 released Sep 14 due Sep 17 stop collection Sep 24 release grades

Pre-Grading	*Grading*		*Post-Grading*
File structure validation	Test cases		Score-weighting
	Visible	Hidden	
Compilation	Custom scripts		Late submission score deduction
Diff with skeleton	Coverage		Plagiarism detection
	Static analysis		Memory detection

Outputs
- Follows Structure - Compiled - Not skeleton - Passed and Failed Visible test case
Success

Table 4 An example of a grading pipeline, giving immediate feedback (red: stages included in this pipeline; dark red: stages being run; light red: stages not run currently; white: unselected stages)

In Table 4, as an example, instances of required inputs and outputs from the pipeline stages in Table 2 are supplied. As an example, let the current date be September 10 and a student has just submitted an assignment. In that case, immediate feedback will be given, meaning that they will receive immediate feedback of whether the submission file structure complies with what is required (specified in grading config), whether the program compiles successfully (in this case C++ programs can be compiled), and the result of diff (as in Unix diff utility between files) between the submission and the provided skeleton. Also, since immediate feedback is enabled, visible test cases will be run, and feedback will be shown.

Inputs
All last COMP2012-La PA1 submissions
Config: – Enable immediate feedback – **Do not show hidden test cases**
Sep 1 released Sep 14 due **Sep 17 stop collection** Sep 24 release grades

Pre-Grading	*Grading*		*Post-Grading*
File structure validation	Test cases		Score-weighting
	Visible	Hidden	
Compilation	Custom scripts		Late submission score deduction
Diff with skeleton	Coverage		Plagiarism detection
	Static analysis		Memory detection

Outputs
- Passed and Failed test cases - Scores over both visible and hidden test cases
All console history

Table 5 An example of a grading pipeline, grading in full (red: stages included in this pipeline; dark red: stages being run; white: unselected stages)

Further, after the date of submission deadline September 17, all last submissions are graded in a batch using standard I/O against test cases defined (in grading config for this stage), as Table 5 shows. This time, both visible and hidden test cases are run. An option controls the visibility of hidden test cases so that they would not be revealed to students. The test case results and scores are shipped back to the Submission Subsystem as part of the grading report for review.

File System Details

The file system as stated is part of the core components in Figure 2 Architecture Diagram is used to store submission archives, submission files, and other assignment-related files.

It is implemented as a Docker volume currently, such that containers can mount to this file system when they need to access these files.

The Submission subsystem is responsible for maintaining the directory structure to store files for different submissions and assignments, indexed by their ids on the Database, as well as uploading of the files. The Auto-Grader subsystem can then read these files and perform grading. The specification of paths is:

- submitted/<submission-id>/: submitted archive (e.g. .zip) of the submission
- extracted/<submission-id>/: the extracted files from submitted
- template/<assignment-config-id>/: template of the assignment for file structure validation
- skeleton/<assignment-config-id>/: skeleton files of the assignment
- provided/<assignment-config-id>/: provided files of the assignment for combining with the extracted files to form a full set of files for grading, if desired

Meeting Minutes

2019-05-31

Time: 4:45pm – 5:30pm

Place: Rm 3501

Present: Desmond, Daniel, Dipsy, David, Bryan

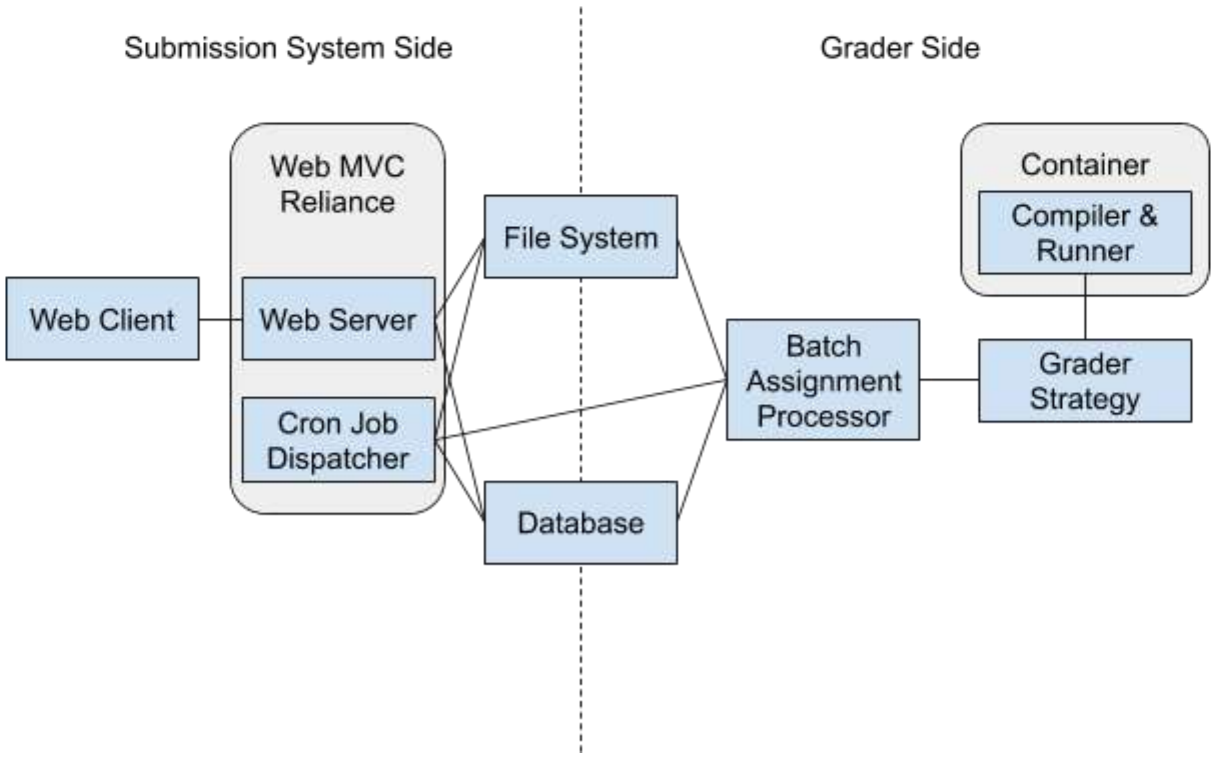
Absent: None

Recorder: Bryan

Report on progress: None

Discussed items:

1. Architecture components of the project



2. Component: Web Client and Web Server features

    a. Student view: Submit the assignment in various ways
    b. Student view: Siew results on graded assignments. Download files
    c. TA View: Create assignments
    d. TA View: Set project deadlines (soft: late will have a penalty or other strategy, hard: 0%)
    e. TA View: A set submission file structure for submission validation
    f. TA View: Grading strategies: Test case strategy, supply main.cpp, JUnit and Java Robot (may implement when time permits)
    g. TA View: Manual review individual submission test case
    h. Cron job creation and update to trigger assignment grading process.

3. Component: Cron job dispatcher
    a. Prepares the required files for grading (Unzipping, cloning from GitHub, etc.)
    b. Serialize required instructions from the database and save as the file to be passed onto the Batch Assignment Processor
    c. Called Batch Assignment Processor to begin the grading process
4. Component: Batch Assignment Processor
    a. Running as Daemon for faster grading
    b. Uses required grading strategy for the grading batch
    c. Updates results to the database
5. Planned technologies
    a. Node JS will be the programming language on the Submission System Side
    b. MariaDB will be the database software of choice
    c. JVM language: Kotlin/Java will be the language of choice on Grader Side
    d. Container technology undecided but will likely be Docker

    e. Server software likely to be Nginx

6. Perform data backup with a circular buffer

7. Useful links

    a. example system: https://github.com/Submitty/Submitty

    b. Gitlab VCS submission: https://courses.cs.washington.edu/courses/cse457/19sp/src/help.php#start

    c. JS ORM: https://github.com/typeorm/typeorm

Goals for the coming week:

1. Daniel - Research JavaScript MVC frameworks

2. How CRON job supply command to JVM Daemon

3. David - Research Hibernate for JVM database integration, use of Docker Daemons

4. Bryan - Research choice of backend language to work with web full-stack JavaScript

2019-06-22

Time: 3:00pm – 5:30pm

Place: Rm 4214A

Present: Daniel, Dipsy, David, Bryan

Absent: Desmond

Recorder: Bryan

Report on progress: None

Discussed items:

1. Containerise Tasks as Microservices
    a. Setup VM/Docker/Kubernetes
2. Refine the Services in the system (components)
    a. Redis: pub/sub communication, caching, session (cookies)
    b. Submission: admin panel, load-balancer by Nginx, backend
    c. Database: read-and-write databases
    d. File utilities: unzipper, structure verifier
    e. Email: queue, notify
    f. Grader manager as 1 daemon
    g. Clustering of submission mirror sites, read-database, file system, grader runner
3. Problems
    a. Semester transitions (add-drop, end of the semester)
    b. Race condition in writing File System

    c. PG course and more edge cases

4. Repository setup

    a. Zinc: Zinc is not CASS https://gitlab.com/.zinc

Goals for the coming week:

1. Find out configuration/bandwidth of CSD/UST server / CASS deployment, for cloud simulation

2. Design Database schema - https://hackmd.io/tfpGrcKsTgGHKv1V83ZG_A

3. Study Submitty and UW https://submitty.org/ https://summerofcode.withgoogle.com/archive/2018/organizations/5076076959105024/

4. How does Grader manager Daemon spawn Grader runner instances with Docker?

    a. dockerd? docker-java?

2019-07-23

Time: 8:00pm – 10:00pm

Place: Rm 3520

Present: Desmond, Daniel, Dipsy, David, Bryan

Absent: None

Recorder: Bryan

Report on progress:

1. Confirming system design https://gitlab.com/.zinc/zinc/wikis/System-Diagram
2. UI/UX design prototype https://xd.adobe.com/view/64156397-9a5e-4ef8-78b2-ea6eabda85ac-1224/

Discussed items:

1. Start drafting the proposal in August
    a. User role functionalities diagram
    b. Background: Several related work reference point for comparison in the table
    c. Tools literature survey and justification
2. Test the submission system with actual courses when ready
    a. COMP2012H homework packages
    b. COMP1021/1022P homework packages
3. Test the auto-grader system with teaching associates when ready, e.g. Wallace
4. Basic features
    a. Compare skeleton and submitted file (diff)
    b. Label for human review when auto-grading is impossible to run/continue

5. Good-to-have features
    a. Coursework release: code/zip/setup
    b. Coursework collection from lecturers for accreditation by RA
    c. Effort-based grading metrics: special case base scoring for compilation error homework
    d. Statistics data visualization

Goals for the coming week:

1. Cloud and infrastructure questions to CSD
    a. How to auto scale dockers?

2019-08-10

Time: 10:30pm – 12:00pm

Place: Rm 3526

Present: Desmond, Daniel, Dipsy, David, Bryan

Absent: None

Recorder: Bryan

Report on progress: None

Discussed items:

1. Technology stack
    a. File system: Git repository internal representation (nested, submodules)
    b. CSS framework: Bulma
    c. Markdown-based WYSIWYG rendering with TinyMCE
    d. In browser code editor with Monaco and CodeMirror
2. Feature: Archiving courses
    a. to ZINC's storage or CS system course folder
3. Feature: Permission and visibility
    a. Bit string flag
4. Architecture: Microservices
    a. Container-Container Communication through HTTP or One overarching container
5. Architecture: Avoiding direct control of some component to another
    a. Email shall not touch the database

    b. All needed data is piped to Auto-Grader via Redis as payload

6. Config file
    a. In YAML / JSON
    b. Ingest file to the database, flattened to store, regenerate to file when requested
7. Component: Upload module using OAuth and Git
8. Component: Course update API by CS System
    a. Using Webhook
9. Security measures
    a. Server-Server TLS certification, authorization, testing, local network restriction
10. System maintenance
    a. When choosing technology stack, consider longevity and employability
    b. Stability is the top priority of the Department
    c. Deploy fresh each semester/year or handled by CS System
11. Testing
    c. Schedule submission subsystem pilot test in COMP2012H, to upload both on CASS/Canvas and ZINC
    d. User Acceptance Tests with working TAs

Goals for the coming week:

1. Features and UI
    a. Add upload source dropdown element
    b. Improve features tree
    c. Make UI interaction flow diagram (when time permits)

    d. Seek review of features from Desmond

2. Try reflection library to analyze source code and return class methods (C++, Java)

3. Schema design task

2019-09-05

Time: 12:45pm – 1:15pm

Place: Rm 3530

Present: Daniel, Dipsy, David, Bryan

Absent: Desmond

Recorder: Bryan

Report on progress: None

Discussed items:

1. MVP of Submission Subsystem
    a. CASS except for email
2. MVP of Auto-Grader Subsystem
    a. A buildable and runnable grading pipeline
3. Feature Milestone (Second MVP)
    a. Auto-Grader: Pipeline extensible stages
    b. Auto-Grader: Custom image - Dockerfile generation for custom images - every stage will become an image
    c. IPC integration (Redis)

Goals for the coming week:

1. Finish the Proposal

2019-10-28

Time: 2:00pm – 3:00pm

Place: Rm 3526

Present: Daniel, Dipsy, David, Bryan, Desmond, Chris

Absent: Kris

Recorder: Bryan

Report on progress:

1. David presented a rudimentary grading pipeline
    a. Dynamically build a pipeline from config
    b. Parallelism: how to specify dependencies
2. Bryan presented a draft on Config

Discussed items:

1. DY stresses that lab grading is important for reducing TA manual checking workload
2. Daniel suggests to specify two designs of YAML for user and machine, and allow two-way to transform
3. David suggests requiring sample solution input from teaching staff to verify that the pipeline can be run first

Goals for the coming week:

1. Monthly Report for October
2. Discuss two kinds of config file

2019-11-25

Time: 12:00pm – 2:00pm

Place: Rm 3526

Present: Daniel, Dipsy, David, Bryan, Desmond, Chris, Kris

Absent: None

Recorder: Bryan

Report on progress:

1. Submission: none
2. Auto-grader: added manager abstraction

Discussed items:

1. Run-on past offering assignments: COMP2012 PA and COMP3021 PA
2. Assignment Submission of COMP2012/2012H available in mid-December
3. Progress Report to show it achieves existing submission/auto-grader capabilities and is robust under different assignment qualities
4. Ask expected Python assignments in Computational Thinking
    a. Sample https://cs61a.org/
    b. Turtle assignments are hard to grade
    c. Grading criteria: carefulness?
5. Set winter break period for all team members to intensive-work
    a. Starting from - David: Jan, Daniel, Dipsy, Bryan: Dec

Goals for the coming week:

1. Examine past grader (Wallace's)

2. Write config file for past assignment test cases

2019-12-16

Time: 3:00pm – 4:00pm

Place: CYTG001

Present: David, Bryan, Daniel, Dipsy, Kris, Chris

Absent: Desmond

Recorder: Bryan

Report on progress:

1. Dipsy: Rubberduck for VBA does not work as well

Discussed items:

1. Intermediate Representation for grading stage to post-grading stage (other stages)
    a. To defer till testing on COMP3111 JUnit and Jacoco
2. Post-grading to determine scores on conditions (not to be done in grading stage)
3. Database interaction from Auto-Grader
    a. ORM: Hibernate vs Exposed
4. Stages in Auto-Grader
    a. To use CLI: Process builder and Coroutines
    b. To use the library: git
    c. To use containers (mount - read-only input, copy to and write at output, persistence; remove in, container then pass): gradlew
    d. To use containers with Coroutines
5. Submission technologies
    a. REST API using Express.js

2020-1-23

Time: 1:30pm – 2:30pm

Place: CYT-G001

Present: Dipsy, Daniel, Bryan, David, Desmond, Chris

Absent: Kris

Recorder: Bryan

Report on progress:

1. Database schema/migration setup script pushed to zinc
2. Dockerized Grader
    a. The current approach is to build everything from scratch each time, including git source

Discussed items:

1. Survey
    a. Sanity check mockup: Student View
    b. Deliver through CSE Admin to Faculty (ask their TA), Full-time, Part-time TA
2. Deployment comparison with ttplanner
    a. After implementation, run stress tests with help from CS System?
    b. Test server would only be the one provided by Kris, no CSE/school provided servers until proven alright and logistics sorted out
3. Deployment repo
    a. The current approach is to git clone from repos and docker-compose from source, instead of using registry

   b. How to deploy from docker-compose to k8s (networking, TLS, security additional issues)

Goals for the coming week:

1. Short-term goal dates

   a. Grader Manager: Hibernate & Jedis

   b. Grader Manager: Runner & Coroutine

   c. Progress Report writing period

2020-2-14

Time: 2:00pm – 4:00pm

Place: Online (Zoom)

Present: Dipsy, Daniel, Bryan, David, Kris

Absent: Desmond, Chris

Recorder: Bryan

Report on progress:

1. Dipsy: Done unit testing of API
2. David: Merged runners with report branch, can parallelize runners in coroutines now; Further, fix resource-constraint bugs
3. Daniel: Investigated message queues in Redis to replace pub/sub
4. Bryan: Merged runners with report branch, can parallelize runners in coroutines now; Patched major enhancements of the pipeline/report/runner

Discussed items:

1. The meeting schedule for Spring semester
    a. Among ourselves: Mon 3 pm-5 pm, Fri 6 pm
    b. With Desmond: Wed 6 pm
2. kotlin-staging-area#54 Interface in the middle of PipelineStage and DockerPipelineStage
    a. EnvPipelineStage to go between PipelineStage and the specific choice of "environment technologies": DockerPipelineStage and VagrantPipelineStage
    b. Because Vagrant is more general albeit heavier than Docker and can be tasked with running in full VM environment including Windows

Goals for the coming week:

1. Progress Report: To be discussed on Monday’s meeting

2. wikis/Redis-IPC-Job Daniel to work on IPC: Redis message queues, referencing Laravel’s use of Lua scripts to store time in payload and recovery, to implement

3. Bryan to work on grader manager, to align with message queue spec from Daniel

4. David to work on resource-constraint, to merge with manager

5. David to work on remaining tasks as stated in GitLab boards

6. Dipsy to study for final exam

2020-3-25

Time: 6:00pm – 7:45pm

Place: Online (Zoom)

Present: Dipsy, Daniel, Bryan, David, Kris, Desmond

Absent: Chris

Recorder: Bryan

Report on progress:

1. Grader: David started Valgrind memory checking stage - need to parse its XML
2. Chris: Test case editor UI by reflection (in React.js)
3. [UAT] comp2012-spring2020-pa1 Mar 27

Discussed items:

1. President's Cup: Poster can be used to sell ZINC to CSE faculty
2. COMP2012/2012 grading used to standardize to eclipse; which can be replaced by std=c++11
3. Valgrind memory checking stage: default `-g` flag?
4. Grader: dynamic class loader
5. Feature-Completeness
    a. UI: Touch-up on Report rendering, Email event-handling, and rendering, UI notification? Assignment config editor.
    b. Grader: See Issue board

Goals for the coming week:

1. Desmond: Meeting with Henry to negotiate cloud spec

2020-4-15

Time: 6:00pm – 6:40pm

Place: Online (Zoom)

Present: Dipsy, Bryan, David, Desmond

Absent: Daniel, Kris, Chris

Recorder: Bryan

Report on progress:

1. Grader: Integrated Valgrind and Pipeline stage refactoring
2. UI: Report rendering of Test case results (diff), Completed assignment config editor

Discussed items:

1. Meeting with IA Kelvin to share automatic grading system
2. Last 1+ week of development to complete all essential features
    a. Submission:
        i. assignment config editor + config YAML GUI editor
        ii. report rendering
        iii. add stable server subdomain
    b. Grader:
        i. Scheduled grading
        ii. [ValgrindRun, GNUMake, Gradle, GTest, JUnit5] Config and Report Units
        iii. [Docker] volume tracker, multiple machine dispatching, tunneling of fsdata/socket
        iv. Stage grouping
        v. Refactoring

vi. Interfaces & Modularization

3. UATs
   a. comp2012-pa1: Grader to cross-check against teaching staff's grading script
   b. comp2012-pa2: Ask students to optionally submit for Valgrind only
   c. comp2012-pa3: After the presentation already
4. Server: Keep using Kris's server for FYP. Official server is ready by Henry but in centos, to be migrated later

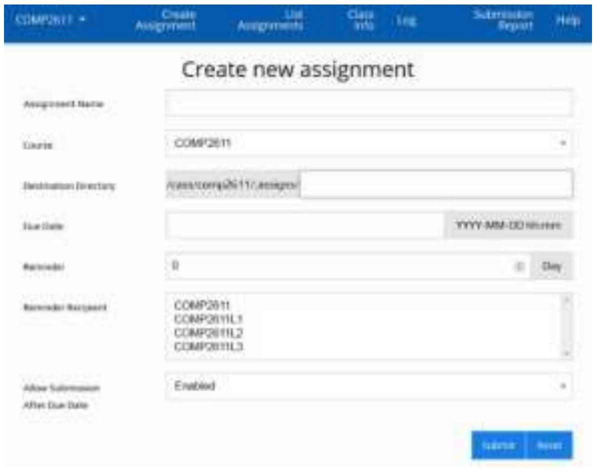
Goals for the coming week:

1. Grader: Debug uncompilable and timed-out submissions
2. Desmond: Confirm [UAT] comp2012-pa2 ready date
3. Chris: Test case editor UI by reflection (in React.js)

Literature Review Details on Existing Systems

CASS

CASS is the existing assignment collection system developed and maintained by CS System in HKUST. Teaching staff can log in to the system to create, view, modify, and delete assignments. In managing the assignments, CASS provides settings for assignment name, course, location in FTP, deadline, visibility, reminder policy, and audience, as well as a basic late day policy that can allow students to submit assignments within a period after deadline. Teaching staff can also view the students of a course section, submission status of the assignment, such as the submission status of each student, and the whole submission log. When retrieving the submissions of students, teaching staff need to login through FTP into the CS Lab2 machine. Meanwhile, the course list and system list are maintained by the system, not the teaching staff.



COMP2611
/cass/comp2611/.assigns/
0
COMP2611
COMP2611L1
COMP2611L2
COMP2611L3
Enabled

Figure 15 Interface to create assignment [1]

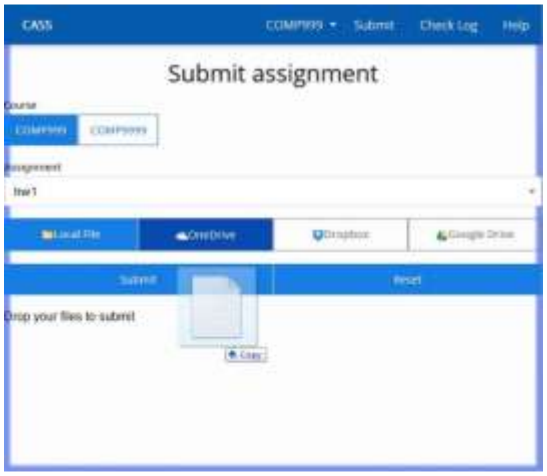


Figure 16 Interface to upload assignment [1]

On the other hand, students can submit assignments to the system, through file upload, Google Drive, Dropbox, or OneDrive. They can also view the checksum and check the log to ensure that they have really submitted the correct assignment without corruption, or whether have they submitted before. Assignments can be submitted multiple times and even with the same file name. The system will keep all the submissions, and file upload size is limited to 100MB. [1]

The limitations of CASS are that, for teaching staff, it does not provide auto-grading features, so they need to manually download the files to grade. As for students, they cannot download their uploaded assignments but can only check the submission from the file name in the log. CASS also does not manage the release of grades. [1]

Submitty

Submitty has the following Roles in the submission and auto-grader system:

- Student
- TA / Grader: Same as Student plus the permission to grade
- Instructor: for head TA as well
- System Administrator: provided with extensive documentation for maintenance and deployment
- Developer: provided with extensive documentation for understanding the project and further open-source development

It has system-wide capabilities of logging in and retrieving the associated course and assignment list for the user, secure testing of many programming languages (Python, C/C++, Java, Scheme, Prolog, SQL, SPIM, basically anything available on GNU/Linux), and notifications and email announcements. The system is scalable to multiple courses, thousands of students, multiple instructors, and TAs per course. Installation is available on hardware or a virtual machine, with detailed documentation. It is free, Open Source, and is contributed by the student community of Rensselaer Center for Open-Source of Rensselaer Polytechnic Institute.

Most code in the submission web platform is in PHP, while most in the auto-grader are scripted in Python.

At the per-course level, firstly there is a discussion forum for students and teaching staff to engage in course and assignment relevant discussion threads.

Students can (1) access current and past courses, and (2) access instructor-provided skeleton code (for VCS, clone and pull from school-hosted instructor-specified URL, and log in with student credentials). However, only school-hosted Git servers are currently supported, meaning that repositories hosted by other parties cannot be used in conjunction with Submitty.

Instructors can (1) host course materials; (2) set course settings: change course name, late days, version control system base URL, submission settings, grading settings; (3) manage users: the students and TAs involved, and can distribute students into equal sections for grading instead of enrollment sections; (4) manage late days: award extra late days to specific students, grant Excused Absence Extensions; (5) manage enrollment: by a university department, manual input, csv import; (6) manage files with a specified directory structure; (7) views logs: user access, site error, TA grading, auto-grading queue, auto-grading results, gradable configuration build, and finally, (8) create and edit a gradable (i.e. an object representing an assignment): Set unique id, title, 3 types of gradable (explained below), URL to instructions, due date, late days option, TA beta testing date, open to the submission date, TA grading date, grades release date, grader assignment methods (explained below).

In the submission level, an assignment is generally called a gradable. A gradable has a lifecycle: Future, Beta for TA testing, Open, Past Due (allowing late submissions), Closed for grading, Grades available – sorted in the order of a planned assignment to a graded assignment.

There are 3 types of gradable: (1) electronic submissions, includes homework/project/exercise code/pdf/images, can be auto-graded or manually graded or both. (2) Numeric, includes data entry of an array of numbers as in quizzes/exam scores, can be entered/edited with a spreadsheet-like interface, supports CSV import. (3) Checkpoints, for class exercises to demonstrate material understanding, marked as either full/half/no credit, can also be entered/edited with a spreadsheet-like interface. There are 4 Grading user groups for permission control: instructor, full access grader, limited access grader, student; and Grader assignment methods for choosing full/partial visibility of submissions to TAs, i.e. whether to allow a TA full access or that they can only access to assigned sections to grade.

Students upload a submission by drag-and-drop, zip upload, or version control (Git: commit and push with student credentials from git config, and then go to Submitty and press “Grade My Repository” to grade up to the current commit). Submitty will checkout the latest version and run any grading tests specified, finally return the testing and grading results to the web platform. An improvement to this is to let Teams work with Git, as we shall see later Submitty has strong support for team assignments.

All roles can view past submissions. Students can choose which submission as “Active” for a gradable before grading begins. Students can also view feedback, tests, and grading results later, on the same page.

Instructors have the flexibility to allow multiple submissions to correct mistakes. They can also batch upload various kinds of digital documents for submission purposes, including scanned pdf exams, QR code name matching, and PDF annotation.

Submitty supports team assignments. Students can create a team, view the team, and leave the team, seek members by exposing their name, and accept joining a team. Instructors can lockout teams, then the student can only view the team; set maximum team size, edit, and export team information.

In the grading level, the most basic parts are tests and grades.

Tests include file structure validation, compilation checks (if it applies), and other simple sanity checks. Students can receive the feedback and modify their submission, in which the immediate feedback is customizable, for example, hidden tests can be optionally visible to students.

Grades include the graded scores and feedback by the TAs / Instructor. Grade inquiry is a feature like appealing, where an official discussion thread is provided with the involved TA / Instructor. Instructors can enable/disable grade inquiry and set the inquiry due date; export spreadsheet of overall course grades; and manually adjust grades with features like “Silent edit grades” and performing Grade Override for makeup exams, plagiarism, one-off bonus, and penalty.

Advanced grading tools are also available for in-depth code analysis and subject-specialized assignments, these include static analysis (parsing tokens, nodes, function calls, and showing their visualizations), JUnit, code coverage, memory debuggers, networked and distributed assignments, screenshots/gifs of graphics assignments, and database assignments. There is also a user interface for manual grading by the instructor, TAs, or peer graders.

Late days policies are also in place. There are 3 of them: (1) setting allowed late dates per term, such that a student distributes the late days across assignments for their use; (2) setting allowed late days per assignment, where every assignment has some fixed number of late days and cannot be recycled to other assignments; (3) setting changing allowed per term, where late days of individual students can be tweaked by the instructor. Students can see their late days usage, so can the instructor monitor this for all students.

Students can also use Excused Absence Extensions in events of emergency, which they are not charged for late days. Instructors can manage and approve extensions.

Plagiarism detection is supported as part of the grading pipeline, by running their custom script.

Lastly, instructors have advanced control options: they can (1) trigger batch regrade by scripts; (2) view and configure “Rainbow Grades”: a color-coded table of grades, column: gradable, row: student; (3) have full access to various logs.

A drawback is that the grading environments were not containerized as of now, making the system vulnerable to malicious code, and the management of execution environments more complicated; the community is planning to implement containerization later.

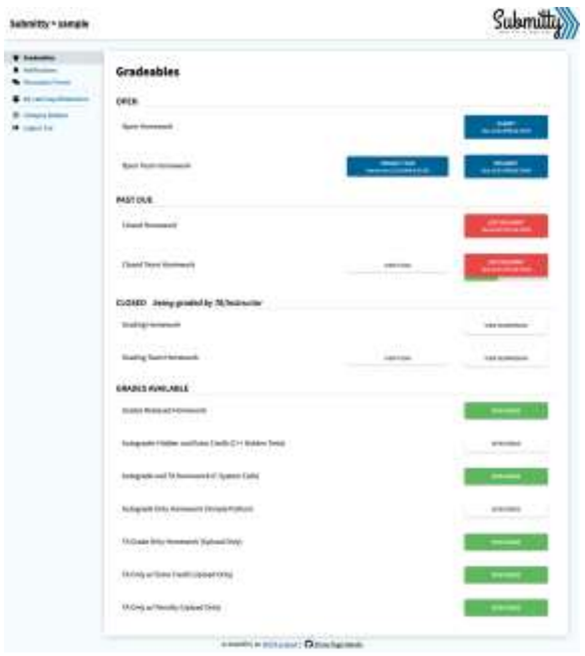


Figure 17 Student navigation page viewing assignment (“gradable”), where one can submit, late submit, view submission, view grade, view team, and manage team [46]

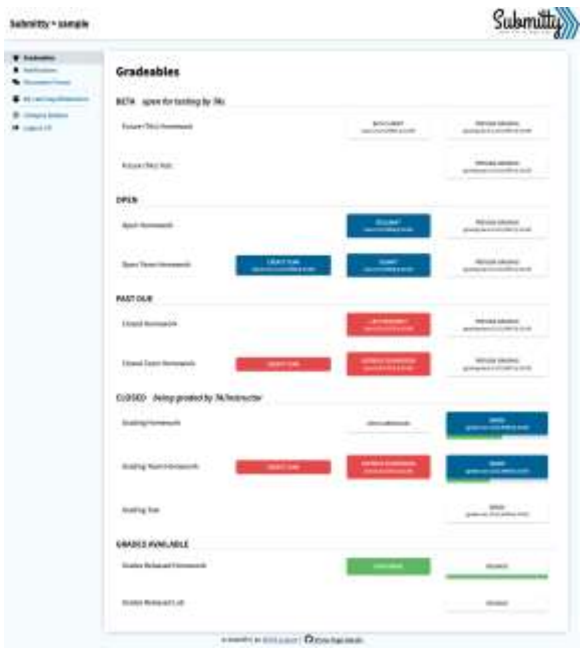
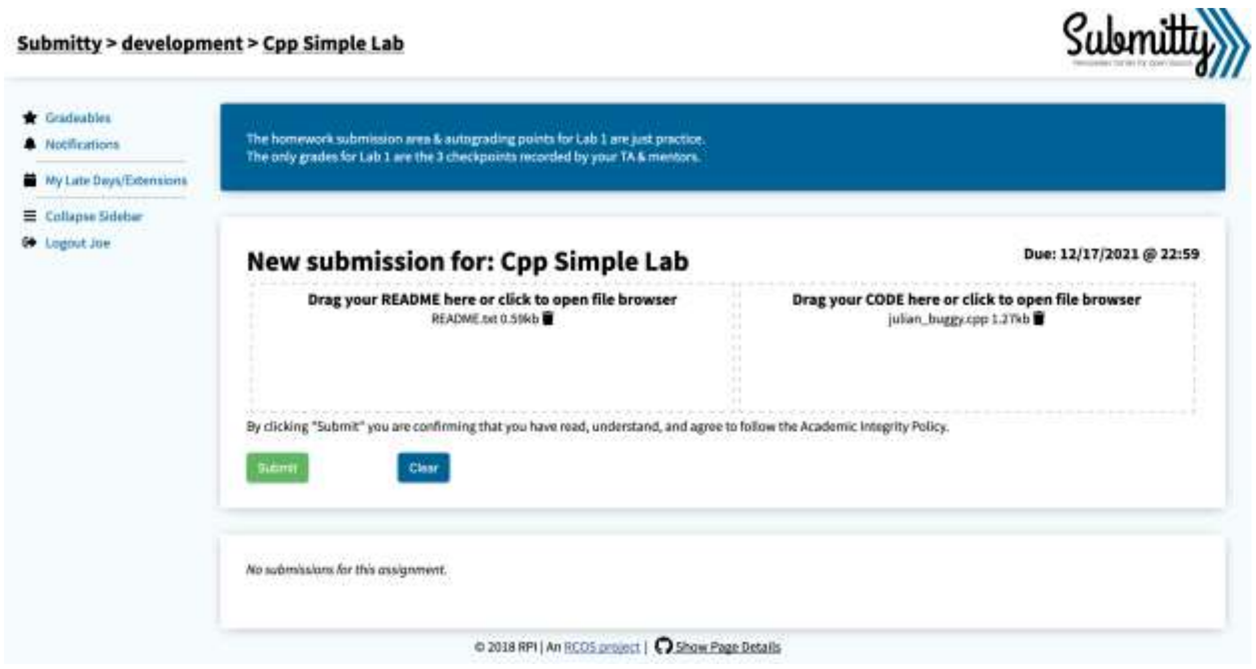

Submitty » sample
Gradeables
BETA open for testing by TAs
OPEN
PAST DUE
CLOSED being graded by TA/instructor
GRADES AVAILABLE

Figure 18 TA navigation page, with an additional column of grading-related buttons [46]



The homework submission area & autograding points for Lab 1 are just practice.
The only grades for Lab 1 are the 3 checkpoints recorded by your TA & mentors.

New submission for: Cpp Simple Lab **Due: 12/17/2021 @ 22:59**

Drag your README here or click to open file browser
README.txt 0.59kb

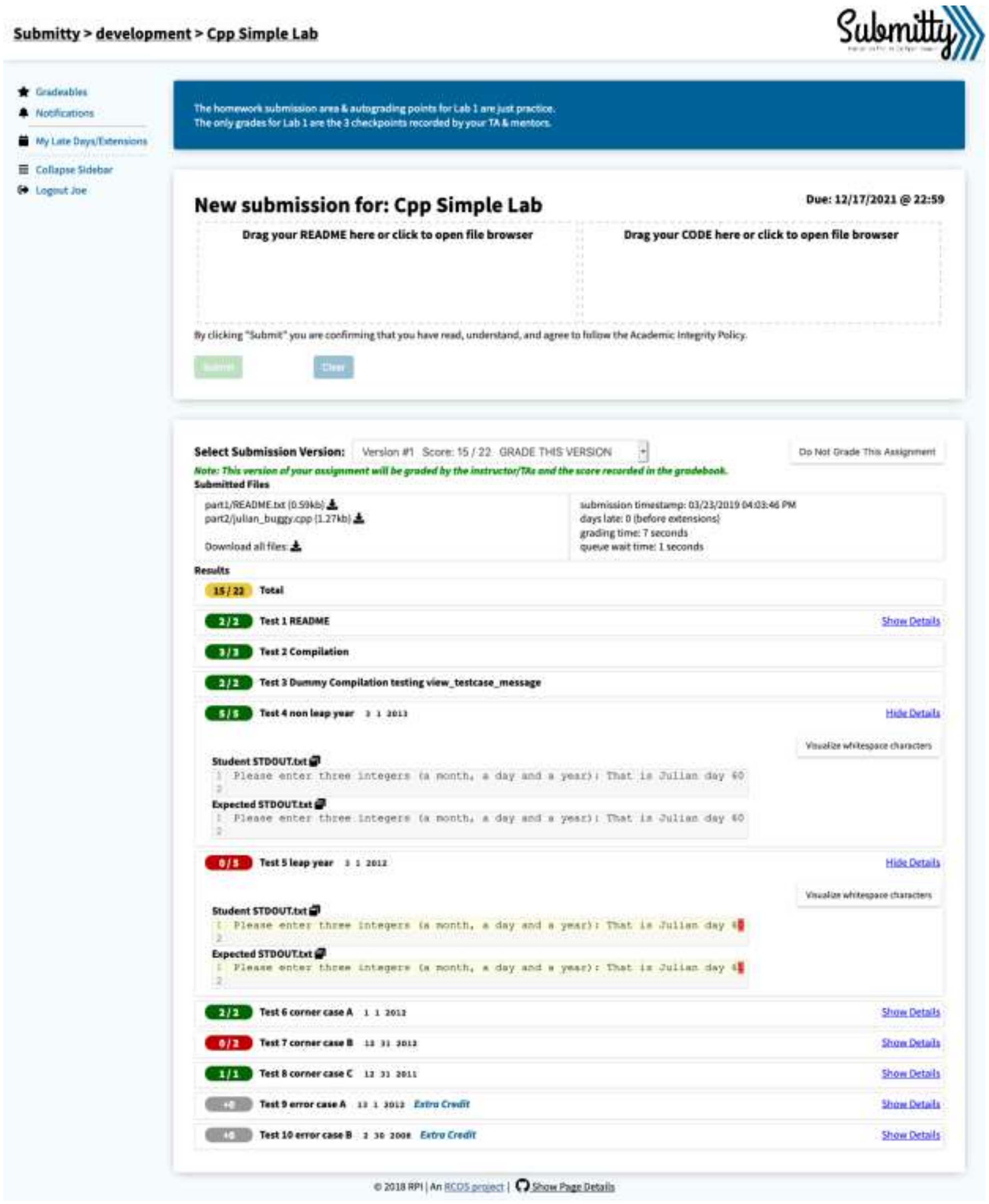
Drag your CODE here or click to open file browser
julian_buggy.cpp 1.27kb

By clicking "Submit" you are confirming that you have read, understand, and agree to follow the Academic Integrity Policy.

No submissions for this assignment.

© 2018 RPI | An RCOS project |

Figure 19 Student to upload a new submission for the assignment.


Submitty > development > Cpp Simple Lab

Gradeables
Notifications
My Late Days/Extensions
Collapse Sidebar
Logout Joe

The homework submission area & autograding points for Lab 1 are just practice.
The only grades for Lab 1 are the 3 checkpoints recorded by your TA & mentors.

New submission for: Cpp Simple Lab

Due: 12/17/2021 @ 22:59

Drag your README here or click to open file browser
Drag your CODE here or click to open file browser

By clicking "Submit" you are confirming that you have read, understand, and agree to follow the Academic Integrity Policy.

Select Submission Version: Version #1 Score: 15 / 22 GRADE THIS VERSION

Note: This version of your assignment will be graded by the instructor/TAs and the score recorded in the gradebook.

Submitted Files

part1/README.txt (0.59kb)
part2/julian_buggy.cpp (1.27kb)

Download all files:

submission timestamp: 03/23/2019 04:03:46 PM
days late: 0 (before extensions)
grading time: 7 seconds
queue wait time: 1 seconds

Results

15 / 22	**Total**	
2 / 2	**Test 1 README**	Show Details
3 / 3	**Test 2 Compilation**	
2 / 2	**Test 3 Dummy Compilation testing view_testcase_message**	
5 / 5	**Test 4 non leap year** `3 1 2011`	Hide Details

Student STDOUT.txt
```
Please enter three integers (a month, a day and a year): That is Julian day 60

```
Expected STDOUT.txt
```
Please enter three integers (a month, a day and a year): That is Julian day 60

```

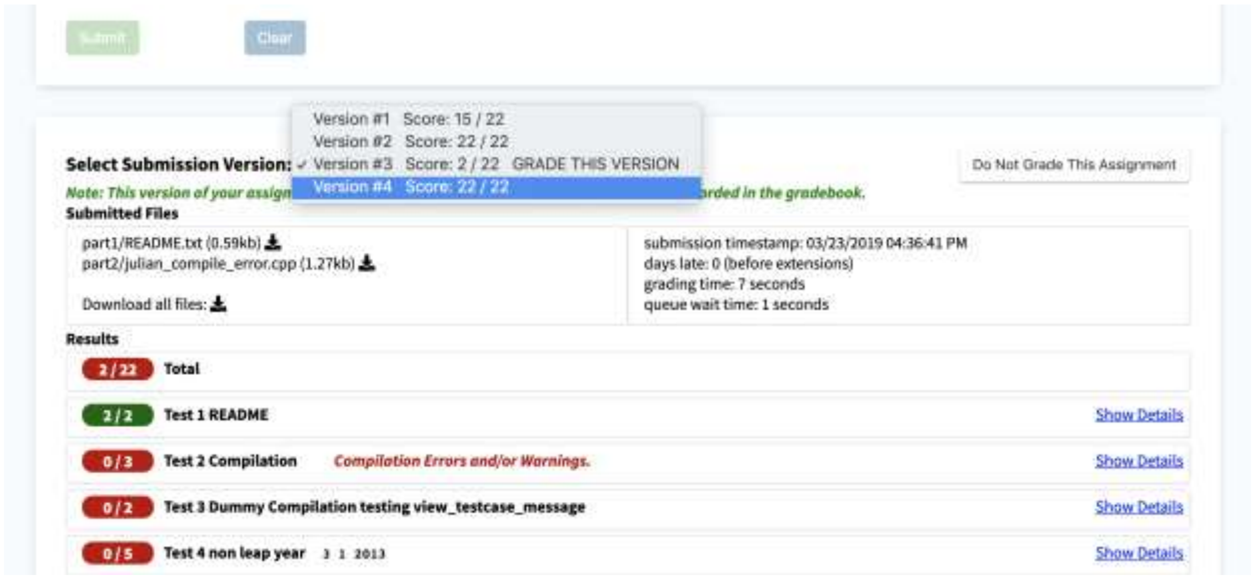
0 / 5	**Test 5 leap year** `3 1 2012`	Hide Details

Student STDOUT.txt
```
Please enter three integers (a month, a day and a year): That is Julian day 6[illegible]

```
Expected STDOUT.txt
```
Please enter three integers (a month, a day and a year): That is Julian day 6[illegible]

```

2 / 2	**Test 6 corner case A** `1 1 2012`	Show Details
0 / 2	**Test 7 corner case B** `12 31 2012`	Show Details
1 / 1	**Test 8 corner case C** `12 31 2011`	Show Details
+0	**Test 9 error case A** `13 1 2012` *Extra Credit*	Show Details
+0	**Test 10 error case B** `2 30 2008` *Extra Credit*	Show Details

© 2018 RPI | An RCOS project | Show Page Details

Figure 20 Student viewing grading results of automated tests, and can switch between submissions (versions) [46]


Submit
Clear
Version #1 Score: 15 / 22
Version #2 Score: 22 / 22
Version #3 Score: 2 / 22 GRADE THIS VERSION
Version #4 Score: 22 / 22
Do Not Grade This Assignment
Show Details

Select Submission Version:

Note: This version of your assign… …rded in the gradebook.

Submitted Files

part1/README.txt (0.59kb)
part2/julian_compile_error.cpp (1.27kb)

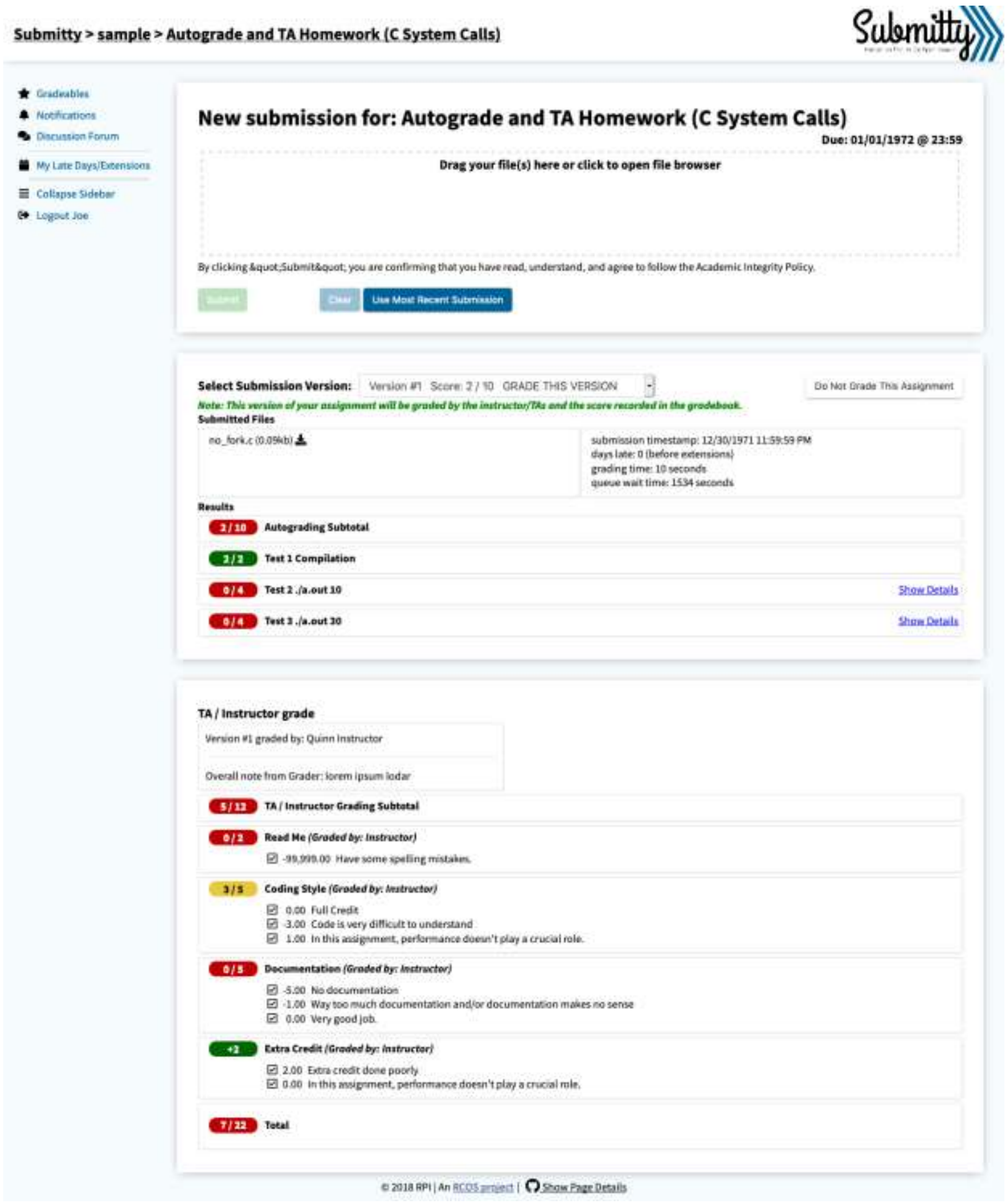
Download all files:

submission timestamp: 03/23/2019 04:36:41 PM
days late: 0 (before extensions)
grading time: 7 seconds
queue wait time: 1 seconds

Results

2 / 22 Total
2 / 2 Test 1 README
0 / 3 Test 2 Compilation Compilation Errors and/or Warnings.
0 / 2 Test 3 Dummy Compilation testing view_testcase_message
0 / 5 Test 4 non leap year 3 1 2013

Figure 21 Student selects which submission as the Active submission to be graded [46]


Submitty > sample > Autograde and TA Homework (C System Calls)

New submission for: Autograde and TA Homework (C System Calls)

Due: 01/01/1972 @ 23:59

Drag your file(s) here or click to open file browser

By clicking &quot;Submit&quot; you are confirming that you have read, understand, and agree to follow the Academic Integrity Policy.

Select Submission Version: Version #1 Score: 2 / 10 GRADE THIS VERSION

Note: This version of your assignment will be graded by the instructor/TAs and the score recorded in the gradebook.

Submitted Files

no_fork.c (0.09kb)

submission timestamp: 12/30/1971 11:59:59 PM
days late: 0 (before extensions)
grading time: 10 seconds
queue wait time: 1534 seconds

Results

2 / 10	Autograding Subtotal	
2 / 2	Test 1 Compilation	
0 / 4	Test 2 ./a.out 10	Show Details
0 / 4	Test 3 ./a.out 30	Show Details

TA / Instructor grade

Version #1 graded by: Quinn Instructor

Overall note from Grader: lorem ipsum lodar

5 / 12 TA / Instructor Grading Subtotal

0 / 2 Read Me *(Graded by: Instructor)*
- ☑ -99,999.00 Have some spelling mistakes.

3 / 5 Coding Style *(Graded by: Instructor)*
- ☑ 0.00 Full Credit
- ☑ -3.00 Code is very difficult to understand
- ☑ 1.00 In this assignment, performance doesn't play a crucial role.

0 / 5 Documentation *(Graded by: Instructor)*
- ☑ -5.00 No documentation
- ☑ -1.00 Way too much documentation and/or documentation makes no sense
- ☑ 0.00 Very good job.

+2 Extra Credit *(Graded by: Instructor)*
- ☑ 2.00 Extra credit done poorly
- ☑ 0.00 In this assignment, performance doesn't play a crucial role.

7 / 22 Total

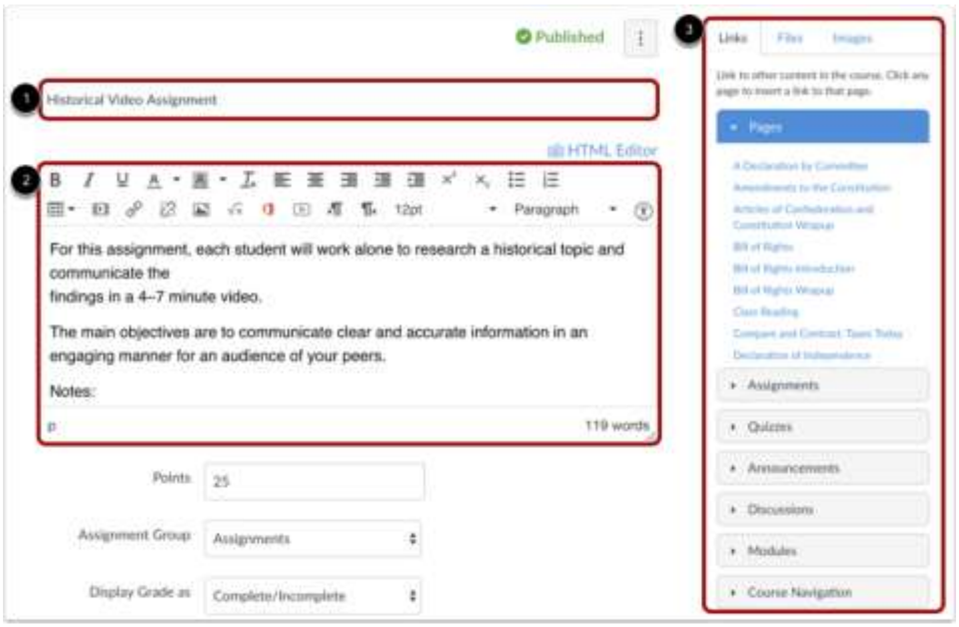
© 2018 RPI | An RCOS project | Show Page Details

Figure 22 Student view grades given by TA / Instructor [46]

Canvas

Canvas is a learning management platform developed by Instructure, which is currently used by various schools and departments worldwide including HKUST. The system includes four roles: instructor, student, admin, and observer.

Instructors can set up course materials, such as lecture notes, files, assignments, quizzes. Assignments and quizzes can be auto-graded or manually graded online, instructors can give comments to submissions and release grades. Moreover, Instructors can make announcements or control visibilities of details in grades, assignments, and other materials, and view coursework statistics of all students.



Historical Video Assignment

For this assignment, each student will work alone to research a historical topic and communicate the
findings in a 4–7 minute video.

The main objectives are to communicate clear and accurate information in an engaging manner for an audience of your peers.

Notes:

p 119 words

Points: 25

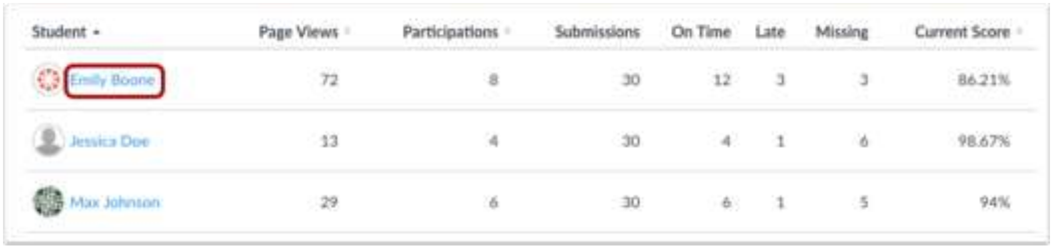
Assignment Group: Assignments

Display Grade as: Complete/Incomplete

Link to other content in the course. Click any page to insert a link to that page.

A Declaration by Committee
Amendments to the Constitution
Articles of Confederation and Constitution Wrapup
Bill of Rights
Bill of Rights Introduction
Bill of Rights Wrapup
Class Reading
Compare and Contrast: Taxes Today
Declaration of Independence

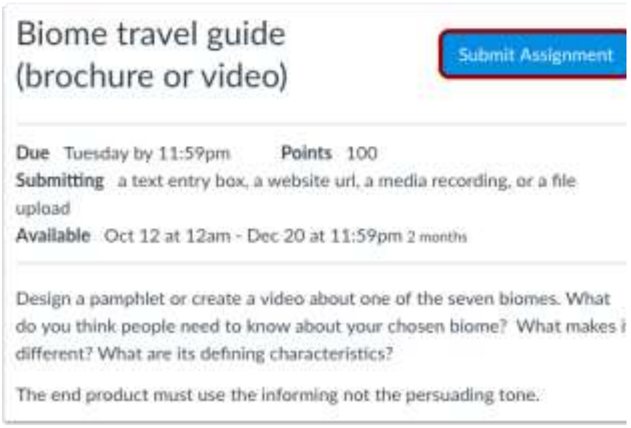
Figure 23 Interface for the instructor to create an assignment. 1: assignment title, 2. Assignment description, 3. Insert reference link to assignment description [3]



Student	Page Views	Participations	Submissions	On Time	Late	Missing	Current Score
Emily Boone	72	8	30	12	3	3	86.21%
Jessica Doe	13	4	30	4	1	6	98.67%
Max Johnson	29	6	30	6	1	5	94%

Figure 24 Interface for the instructor to analysis performance of students [3]

Students can download the materials, upload, and comment on assignments, review and download submitted assignments, check grades, do quizzes, and participate in discussions. Students can also check their total scores and assignment statistics, although the statistics do not exclude some extreme values such as empty submission, and the weighting of the total score is often wrong. Students also receive notifications by email when there are new assignments, description updates, grade release, or announcements from their instructors. Students can form groups on canvas for team projects and assignments, such that they can submit and be graded on a team basis. Canvas also shows a calendar that lists all the deadlines.


Biome travel guide (brochure or video)

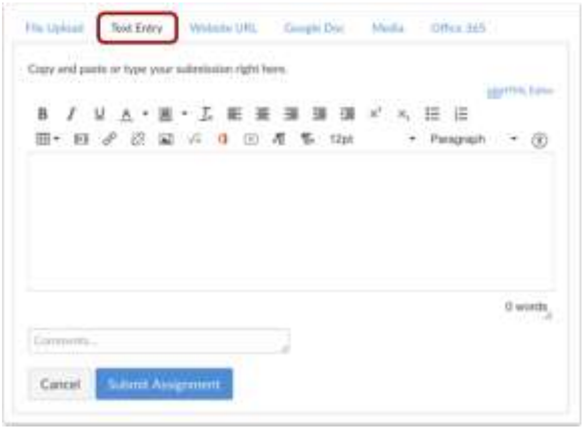
Submit Assignment

Due Tuesday by 11:59pm **Points** 100
Submitting a text entry box, a website url, a media recording, or a file upload
Available Oct 12 at 12am - Dec 20 at 11:59pm 2 months

Design a pamphlet or create a video about one of the seven biomes. What do you think people need to know about your chosen biome? What makes i different? What are its defining characteristics?

The end product must use the informing not the persuading tone.

Figure 25 Interface for students to submit an assignment [3]


File Upload | Text Entry | Website URL | Google Doc | Media | Office 365

Copy and paste or type your submission right here.

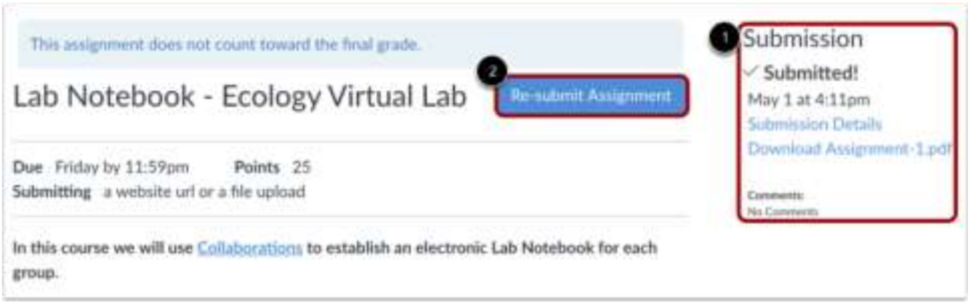
12pt Paragraph

0 words

Comments...

Cancel | Submit Assignment

Figure 26 Students can submit by file upload, text input, URL, google doc, media (recording video or audio using a webcam), o365 [3]


This assignment does not count toward the final grade.

Lab Notebook - Ecology Virtual Lab

Re-submit Assignment

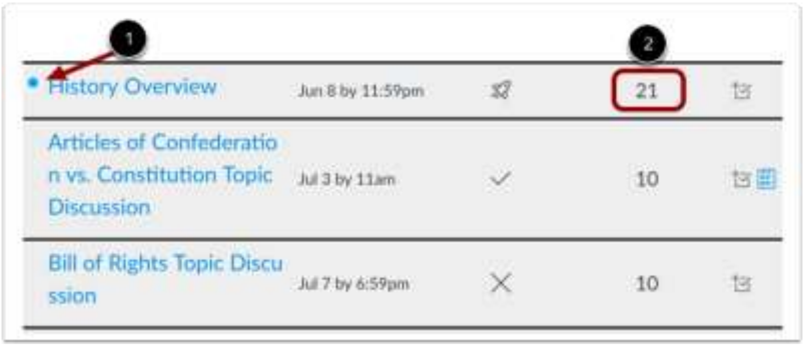
Due Friday by 11:59pm **Points** 25
Submitting a website url or a file upload

In this course we will use Collaborations to establish an electronic Lab Notebook for each group.

Submission
✓ Submitted!
May 1 at 4:11pm
Submission Details
Download Assignment-1.pdf

Comments:
No Comments

Figure 27 shows that students can 1. resubmit an assignment, 2. download submitted assignments and review comments from instructors [3]



● History Overview	Jun 8 by 11:59pm		21
Articles of Confederation vs. Constitution Topic Discussion	Jul 3 by 11am	✓	10
Bill of Rights Topic Discussion	Jul 7 by 6:59pm	✕	10

Figure 28 Interface for students to review their assignment status, including is submitted, is grading, score; assignment statistics. 1. The circle indicates a new change of score, 2. Score obtained. [3]

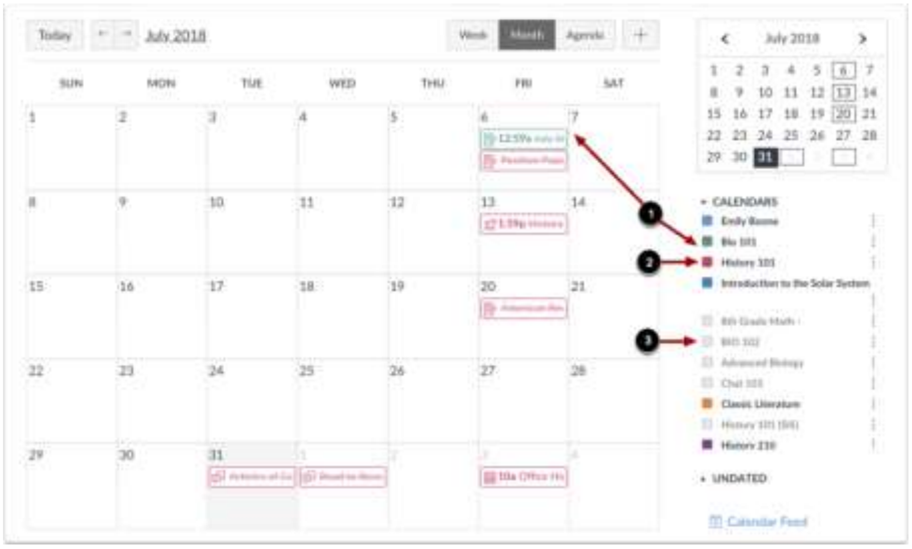
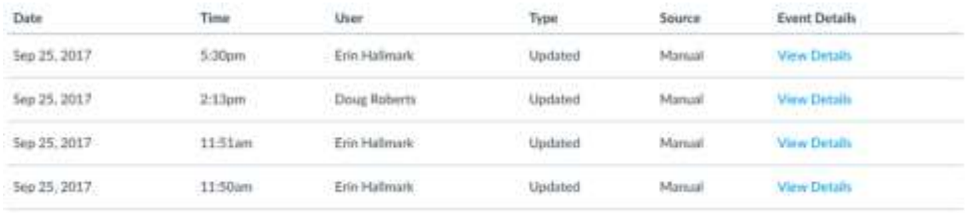


Figure 29 Calendar shows all assignments from courses, color indicates courses, the cross-line indicates the assignment is done. [3]

Admin controls the roles and permissions of all users and can assign new admins and manage data flow between SIS (student information system), such as importing users, courses, enrollments, etc. Admin can also review activities of all courses and users.



Date	Time	User	Type	Source	Event Details
Sep 25, 2017	5:30pm	Erin Hallmark	Updated	Manual	View Details
Sep 25, 2017	2:13pm	Doug Roberts	Updated	Manual	View Details
Sep 25, 2017	11:51am	Erin Hallmark	Updated	Manual	View Details
Sep 25, 2017	11:50am	Erin Hallmark	Updated	Manual	View Details

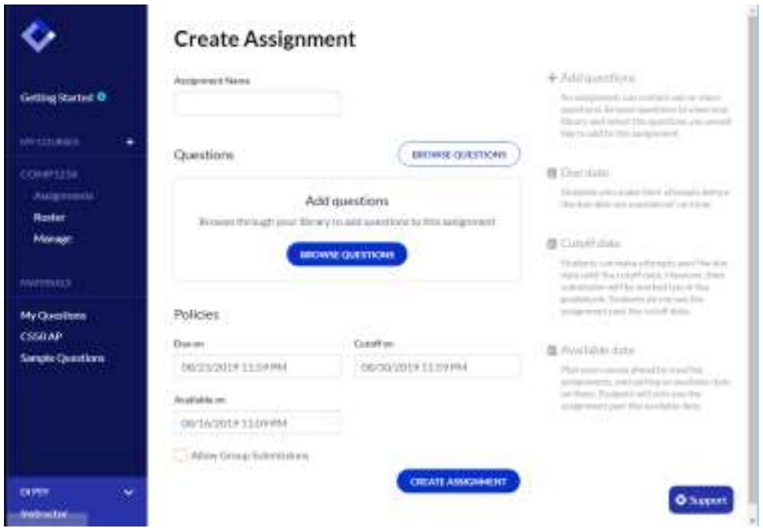
Figure 30 table lists all course activities [3]

The observer is a read-only user who can see all the course materials but not participating in the course activities, one of the usages is for the parent to observe the performance of their child.

Autogradr

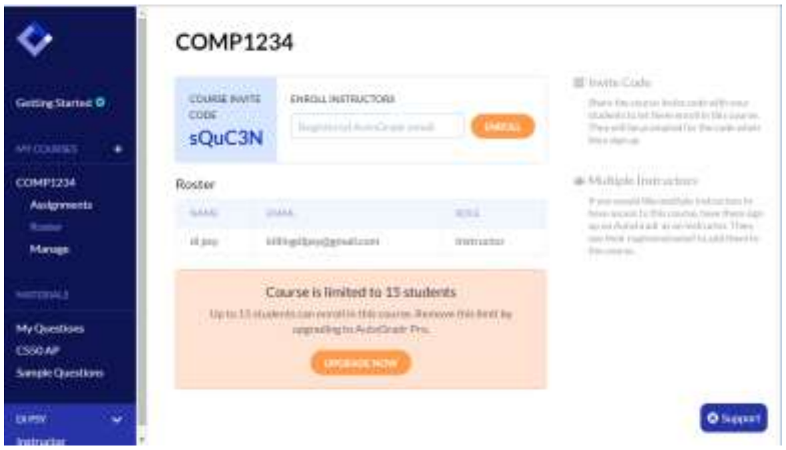
Autogradr is a free website aiming to provide automated grading for programming assignments, with advanced paid features like group assignments, download submissions, and hidden Test Questions [4]. Autogradr is being used by more than 90 institutes in 14 countries. It supports various programming languages, environments (CLI, Web, database), and testing frameworks. When preparing tests, instructors are not required to type any code. Users can register the website as instructor or student, the instructor can set up courses, questions, and test cases, students complete the question using the online IDE provided by the system, and they can receive feedback immediately after submission.

When an instructor creates an assignment for a course, the instructor first creates questions, which have test cases and specific stack, then the instructor creates an assignment and specifies which questions are for the assignment.



06/23/2019 11:59 PM
06/30/2019 11:59 PM
06/16/2019 11:09 PM

Figure 31 Create assignment page, can provide questions, due dates, cutoff date, and available date



sQuC3N

Course is limited to 15 students

Figure 32 Manage course students and co-instructor, they can join the course by invite code

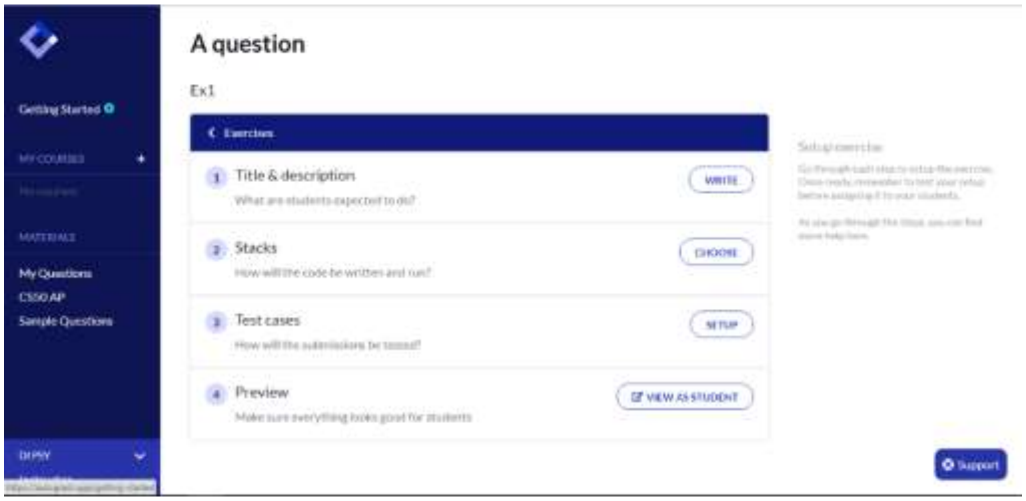


Figure 33 Create question page: specify descriptions, technology stack, test cases and try the question out

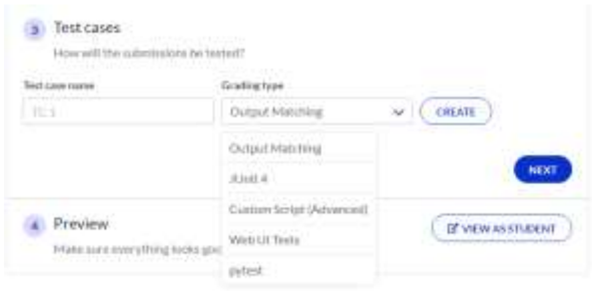


Figure 34 Different kinds of testing framework available, including a highly flexible custom script strategy

Figure 35 Preparation of test case, type or import standard input and output samples

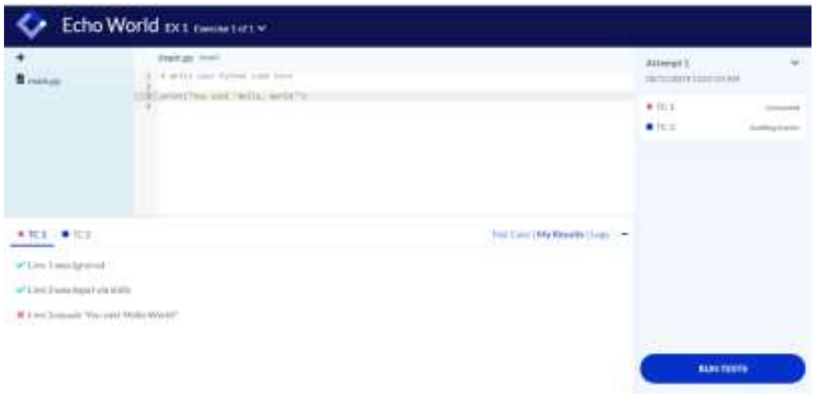
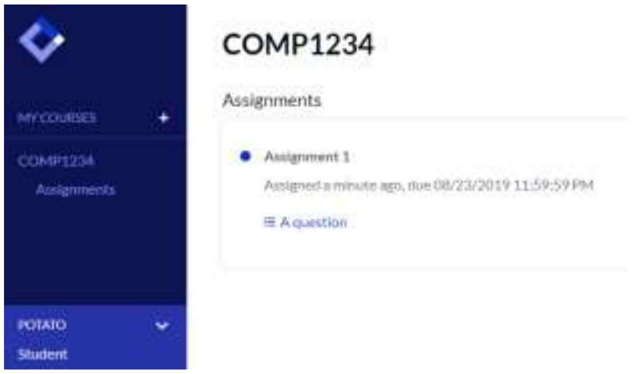


Figure 36 Online IDE with testing result and status of a test case: preparing, building docker, running, completed



COMP1234

Assignments

Assignment 1

Assigned a minute ago, due 08/23/2019 11:59:59 PM

A question

Figure 37 Student browsing assignments of course

Assignment 1

Figure 38 Instructor view all submissions by students and export the results

Questionnaire

A survey was conducted in February 2020 on the current assignment grading situations in CSE courses and opinions on ZINC features. The Google Form survey was released via email to all CSE faculty of HKUST. Out of 59 invitations, 7 responses were received.

Survey Form

In the following document, the ? symbol indicates a question. If there are predefined choices, ○ are used to indicate radio selections and ☐ checkboxes. If there are no choices under the question, the answer is a free textbox. Other texts are help texts and descriptions for the survey taker.

Background Information Section

? Do you regularly teach any courses involving programming?
- ☐ Yes
- ☐ No

? What courses do you teach?

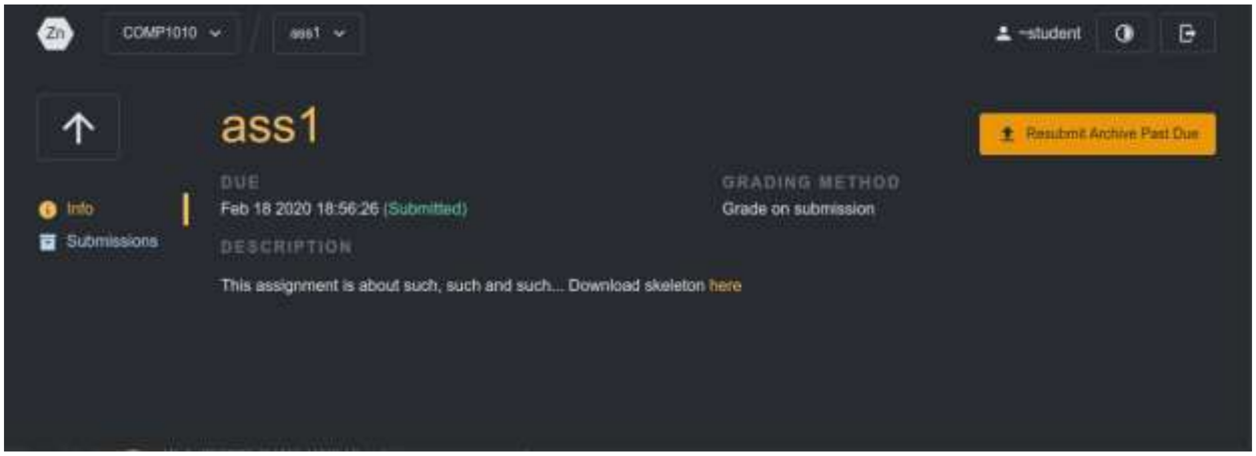
? What languages were involved?
- ☐ C/C++
- ☐ Python
- ☐ Java
- ☐ Matlab
- ☐ Other, please specify.

? What role do you assume in teaching the courses?
- ☐ Instructor
- ☐ Teaching Assistant

? How do you currently grade your programming assignments? (MC)
- ☐ With an automated script/program comparing test cases.
- ☐ Manually, but assignments could be graded automatically.
- ☐ Manually, since the assignments involve a visual or creative aspect.
- ☐ Other, please specify.

ZINC Features Section

ZINC will include an admin panel for teaching staff and a student facing website. Here is a working screenshot of the student web view of an assignment. This view can also display some publicly visible test cases which serve as sanity tests.



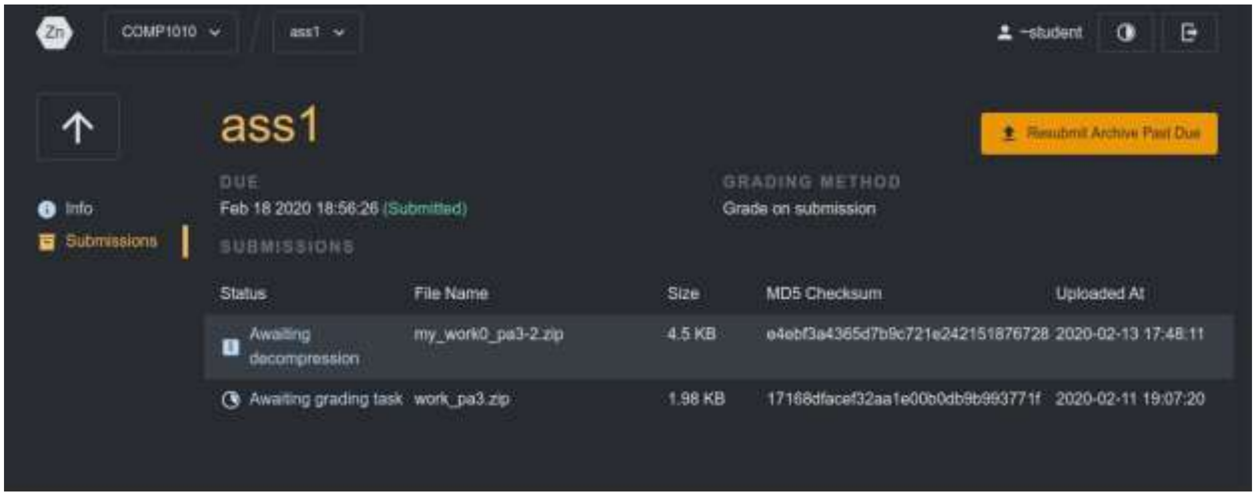
ass1

DUE
Feb 18 2020 18:56:26 (Submitted)

GRADING METHOD
Grade on submission

DESCRIPTION
This assignment is about such, such and such... Download skeleton here

Submissions view showing various stages of automation. Students and teachers will be able to view grading reports on each submission after they are graded.



ass1

DUE
Feb 18 2020 18:56:26 (Submitted)

GRADING METHOD
Grade on submission

SUBMISSIONS

Status	File Name	Size	MD5 Checksum	Uploaded At
Awaiting decompression	my_work0_pa3-2.zip	4.5 KB	e4ebf3a4365d7b9c721e242151876728	2020-02-13 17:48:11
Awaiting grading task	work_pa3.zip	1.98 KB	17168dfacef32aa1e00b0db9b993771f	2020-02-11 19:07:20

Assignment Modes

The flexibility of having an auto-grader allows it to tirelessly grade student submissions. This has inspired us to give the "immediate grading" option, allowing new submissions to be immediately sent to the grader to generate grading reports.

And with this idea in mind, we added the "submission chances" option, to restrict the number of times a student may submit an assignment for auto-grading if desired.

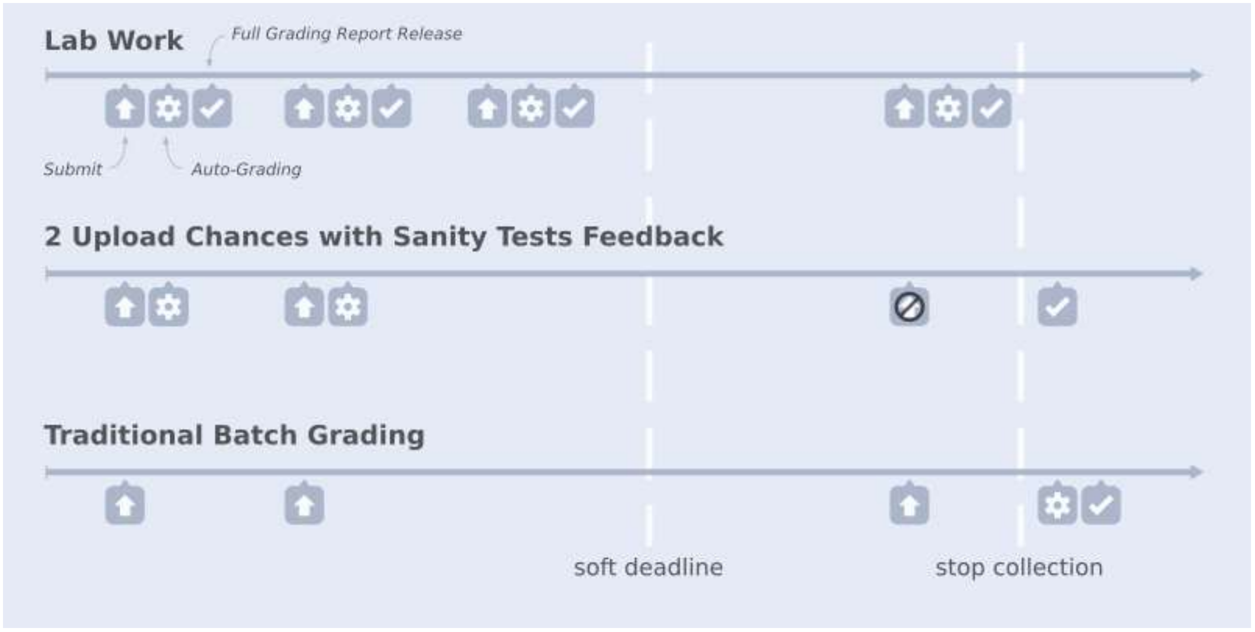
An assignment does not even have to enable auto-grading, allowing the system to still serve as an assignment collection system.

This has opened up many possibilities on new assignment/quiz designs:

- Lab tasks may now be "immediately graded" and "have unlimited submission chances".
- Machine learning tasks graded on accuracy thresholds may now be "immediately graded" and "have a certain number of submission chances".

- Of course, traditional assignments may still be auto-graded (or not) after the soft deadline and stop collection date.

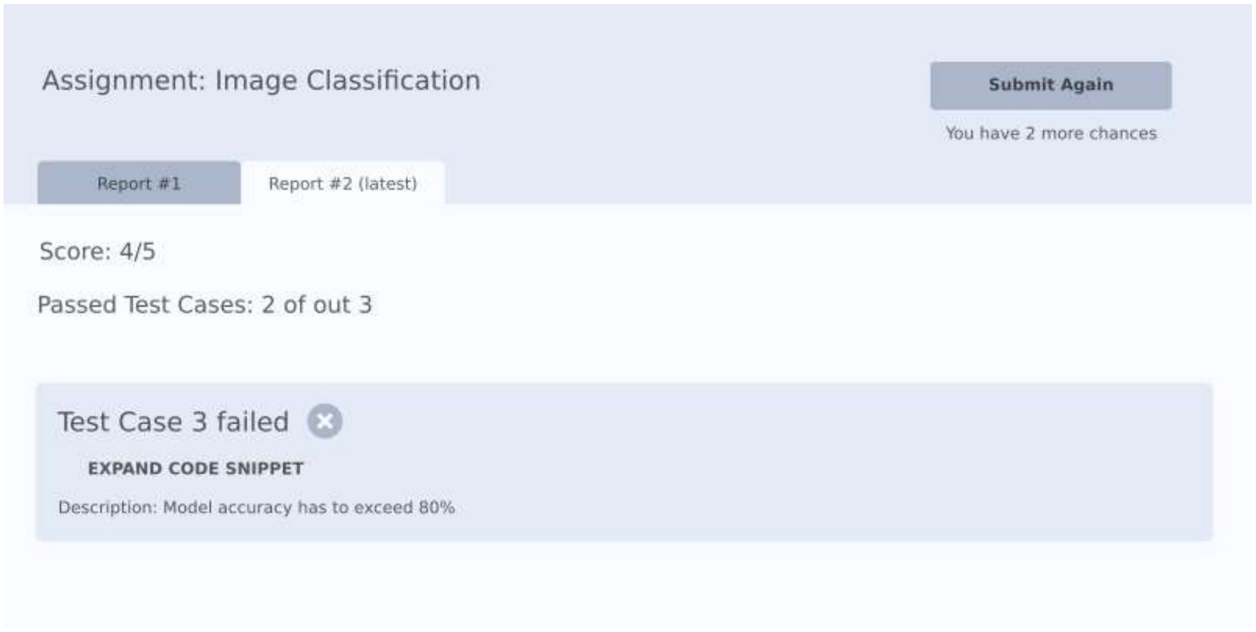
Illustration of various assignment mode presets. Details are configurable.


Lab Work
Full Grading Report Release
Submit
Auto-Grading
2 Upload Chances with Sanity Tests Feedback
Traditional Batch Grading
soft deadline
stop collection

? Immediately Graded Lab Tasks: How helpful do you think this feature would be in alleviating workload on the TAs?
Lab work may be set up to use this feature so that students can submit their tasks during lab time to the platform for an immediate and unlimited number of auto-grading.

- ○ Not Helpful 1
- ○ 2
- ○ 3
- ○ 4
- ○ Very Helpful 5

? Submission Chances: How helpful do you think this feature would be?
This is used alongside the "immediately grade" option. So that students may experiment with the submission, receive feedback from the report, and improve with their remaining chances. This feature is like some questions on the WebWork or Mastering Physics.



Assignment: Image Classification

Score: 4/5

Passed Test Cases: 2 of out 3

Test Case 3 failed

Description: Model accuracy has to exceed 80%

- O Not Helpful 1
- O 2
- O 3
- O 4
- O Very Helpful 5

Late Penalty

Assignments have 2 deadlines: soft deadline (due at) and the hard deadline (stop collection at). Students are still able to submit archives after the soft deadline.

This feature may be enabled for any assignment, for the system to conduct:

- A once-off discount of a fixed score for submissions submitted after the deadline, or
- Periodic grade deduction on a percentage per any custom time-scale

? Late Penalty: How helpful do you think this feature would be?

- O Not Helpful 1
- O 2
- O 3
- O 4
- O Very Helpful 5

Sanity Check

Sanity Check is formed using these features:

- A set of always visible console test cases, contrary to hidden ones that may or may not be displayed after the deadline is reached.
- File structure validation, to reject submissions lacking required files.
- Checking if the submission is merely the provided skeleton.
- Test compilation

? Sanity Check: How helpful do you think this feature would be?
The following is an illustration of screens the student may see after they have submitted their submission to an assignment which has enabled these features.

- O Not Helpful 1
- O 2
- O 3
- O 4
- O Very Helpful 5

Test Case-based Grading

An assignment may be graded using:

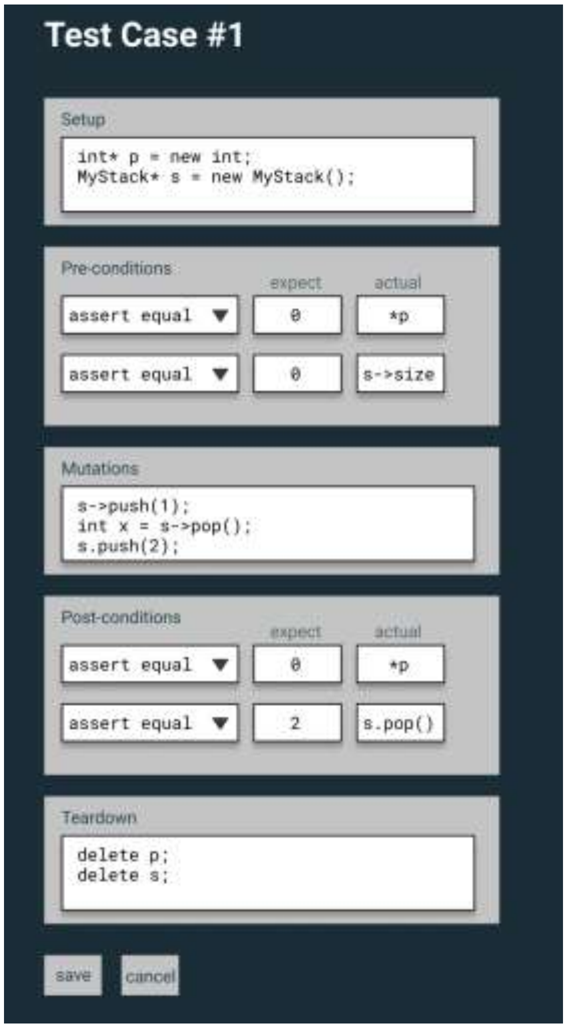
- Standard I/O (through the console), where each test case can specify its:
  - Target executable to run upon
  - Command-line arguments, standard input, expected standard output
  - Weighted score awarded for passing this test case, or equal scores for each test case
  - Visibility:
    - Visible (shown in Sanity Check)
    - Hidden (will never be shown)
    - Visible after grading (revealed only after grading)
    - Visible after grading if failed (revealed only if the test case was failed)
- A language-specific test suite: C++ (Google Test), Java (JUnit), Python (pytest)
  - Currently, we require the user to provide the test suite files

? Visibility of Test Cases: How helpful do you think this feature would be?
*The visibility of a test case in different situations: visible, hidden, visible after grading, visible after grading if failed. *Refer to the text above. You can set this up so that students may or may not be able to view the code used for testing their program.*

- O Not Helpful 1
- O 2

- 3
- 4
- Very Helpful 5

? Test Case Editor: How helpful do you think this feature would be?
We could provide a test case editor on the platform that generates the main files for stdio testing or test suite-specific code for you (prototyped in the image below), featuring test case templates and batch generation. However, it also requires you to understand how the language and/or the testing framework works.


Test Case #1

Setup
```
int* p = new int;
MyStack* s = new MyStack();
```

Pre-conditions

	expect	actual
assert equal ▼	0	*p
assert equal ▼	0	s->size

Mutations
```
s->push(1);
int x = s->pop();
s.push(2);
```

Post-conditions

	expect	actual
assert equal ▼	0	*p
assert equal ▼	2	s.pop()

Teardown
```
delete p;
delete s;
```

- Not Helpful 1
- 2
- 3
- 4
- Very Helpful 5

? Stdio Test Case-based Grading: How much does it cover as a grading strategy for all your assignments?

How much percentage of your assignments can be graded with Standard IO? For example, a computer graphics assignment can be difficult to grade in the console since the output may be required to be judged visually by a human. However, some assignments could be converted to a console-compatible format by the student, e.g. A jupyter notebook file can be exported into source code files to be compiled/executed.

- O 10% or less 1
- O 2
- O 3
- O 4
- O 5
- O 6
- O 7
- O 8
- O 9
- O 100% 10

? GUI Auto-Grading: How important is it to have a GUI auto-grading strategy for your assignments?

Do you need GUI testing automation for your GUI assignments? For example, JavaFX applications can be automated with JavaFX Robot. HTML5 applications may be tested with Puppeteer (headless Chromium)

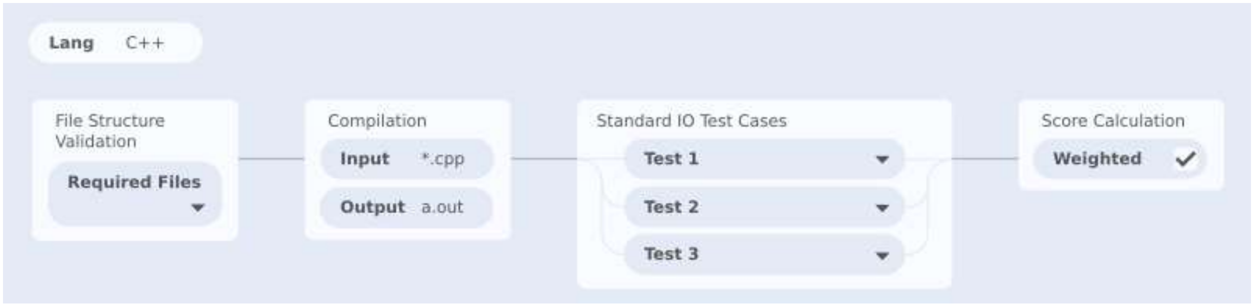
- O Not Important 1
- O 2
- O 3
- O 4
- O Very Important 5

We are using a stage-based design behind the auto-grading system

Each grading strategy has a corresponding "stage", and these stages are composed together to form a grading pipeline. Some stages may be enabled/disabled depending on the assignment, to allow for a flexible auto-grading environment.

Code coverage, similarity checking (plagiarism checking), unit testing, etc are stages that may be inserted into the pipeline.

Pipeline Illustration showing a basic C++ assignment grading design



- ? Customization: Would you be interested to make a custom grading stage that fits into our auto-grading system?
  For example, if you already have a custom auto-grading program that grades your course assignments, would you choose to integrate that into our platform to take advantage of our other features? Or if we do not currently provide a testing environment for a specific language/framework, would you try to write a custom grading "stage" on our platform? Of course, relevant development docs will be provided.
  - ○ Not Interested 1
  - ○ 2
  - ○ 3
  - ○ 4
  - ○ Very Interested 5
- ? Containerization: Grading on different OSes
  By default, we run user submitted code in Linux Docker containers. What other OS(es) do you need for grading assignments? By default, our containers are running on ***Debian Linux****. So if you just need a generic Linux environment, we are already supporting the latest stable Debian, so you may select none of these options.*
  - ☐ Windows 10 Version 1909 (Latest)
  - ☐ macOS 10.15 (Latest)
  - ☐ CentOS 8 (Latest)
  - ☐ Other, please specify.
- ? Do you have other features or grading strategies you wish us to implement?
- ? Any extra comments or suggestions?

Contact Information

- ? What is your preferred name? (We will not disclose personal information.)
- ? What is your ITSC email? (We may like to contact you for any follow-up questions.)

Survey Results

Due to the small sample size, **we may not assert the findings of the survey are conclusive**, but the statements below are what best describe the various questions, which may be interesting nevertheless.

These questions were originally set to find out the situations of various courses, so we may steer our development towards having more course coverage.

Some answers may have been reinterpreted for better visualization.

Language Usage

The following table illustrates the languages used by the teaching staff in their teaching per language answer.

Language Used in Courses	*Per Respondent Counts*
C/C++	6
Python	2
Java	1
Matlab	1
SML, Prolog	1
CUDA, MPI, Pthreads, OpenMP	1

Table 6 Language Used in Courses

From Table 6, we can see that 6 out of 7 respondents used C/C++ somewhere in their courses. This lines up with our hypothesis, as we planned to complete C/C++-specific grading strategies early in our development.

Current Assignment Grading Automation

Grading Automation Situation	*Per Respondent Checkboxes Counts*
Automated test cases check	5
Manual, but has automation potential	2
Manual, contains visual or creative aspects which cannot be automated	6

Table 7 Current Assignment Grading Automation

From Table 7, we can see there is only a little automation potential for some teaching staff for assignments.

Helpfulness of the Flexible Submission-Related Features

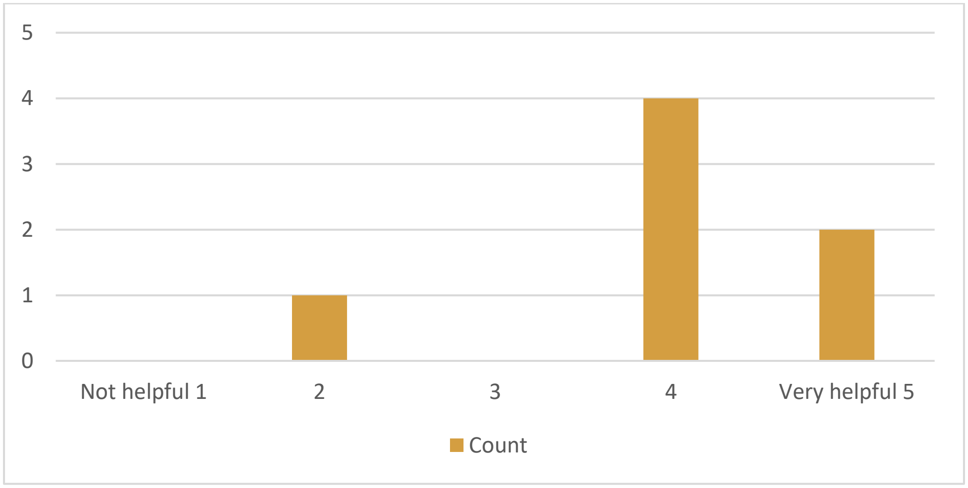


Figure 39 Helpfulness of Having Immediately Graded Lab Tasks

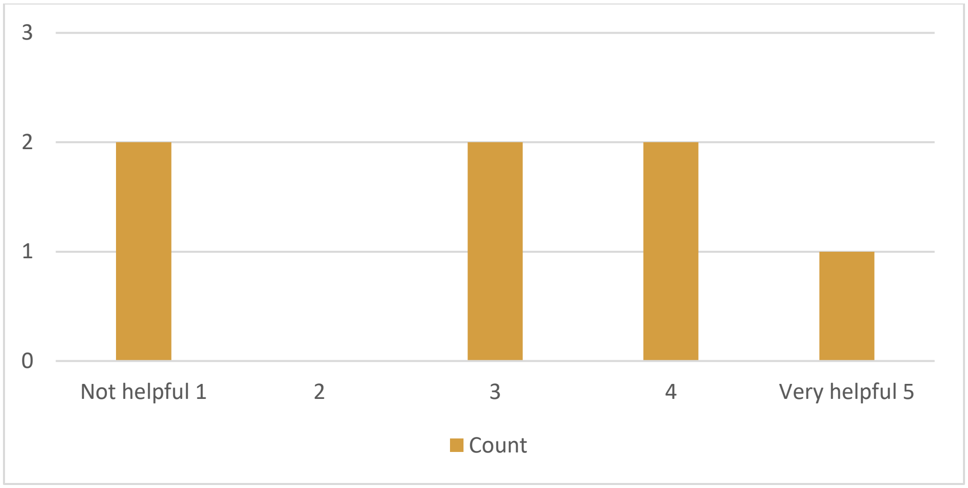


Figure 40 Helpfulness of Submission Chances

Most teaching staff are interested to see the Immediately Graded feature, but opinions are scattered for Submission Chances.

Helpfulness of the Flexible Grading-Related Features

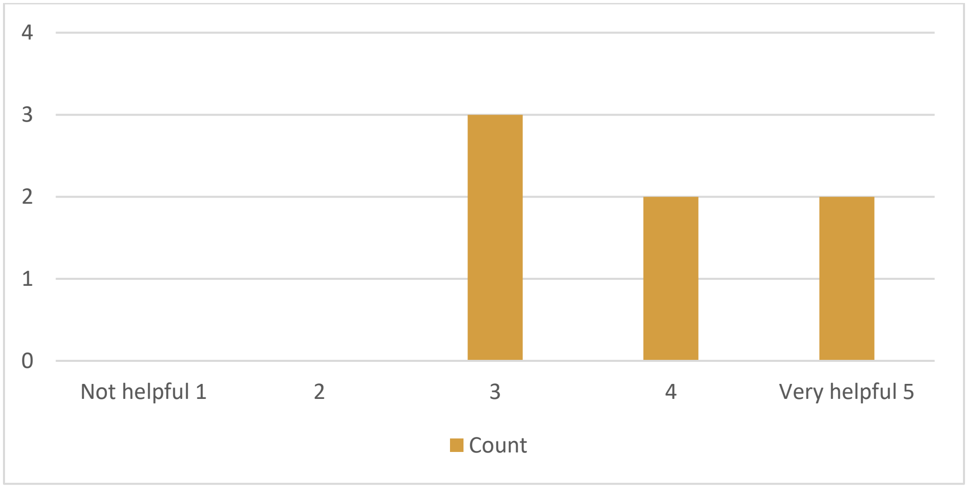


Figure 41 Helpfulness of Late Penalty

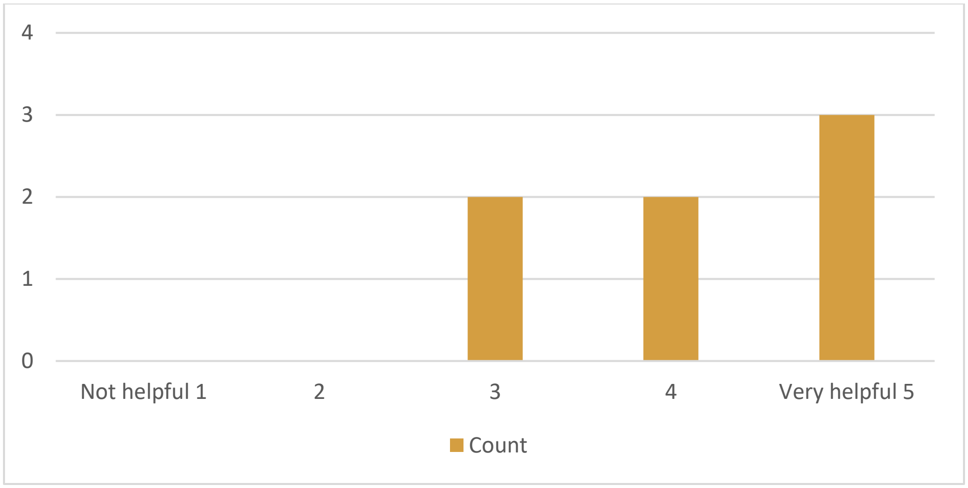


Figure 42 Helpfulness of Sanity Check

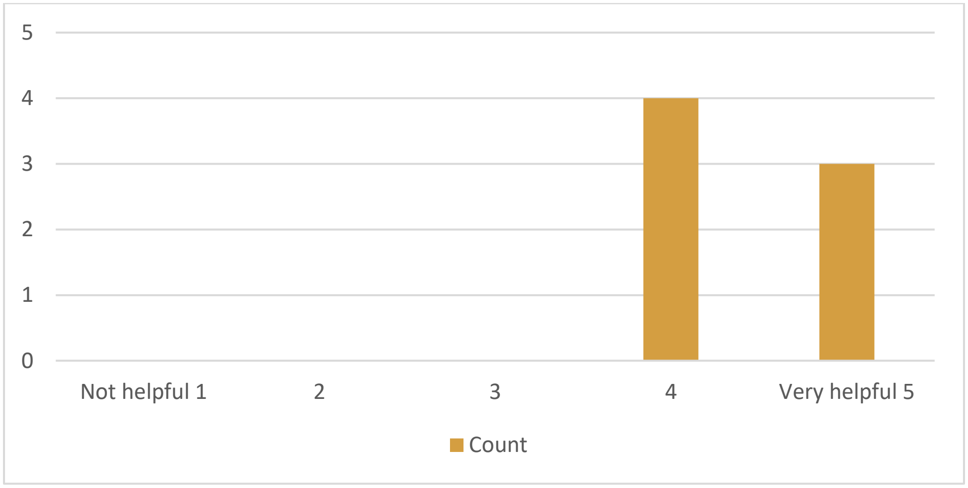


Figure 43 Helpfulness of Test Case-Based Grading

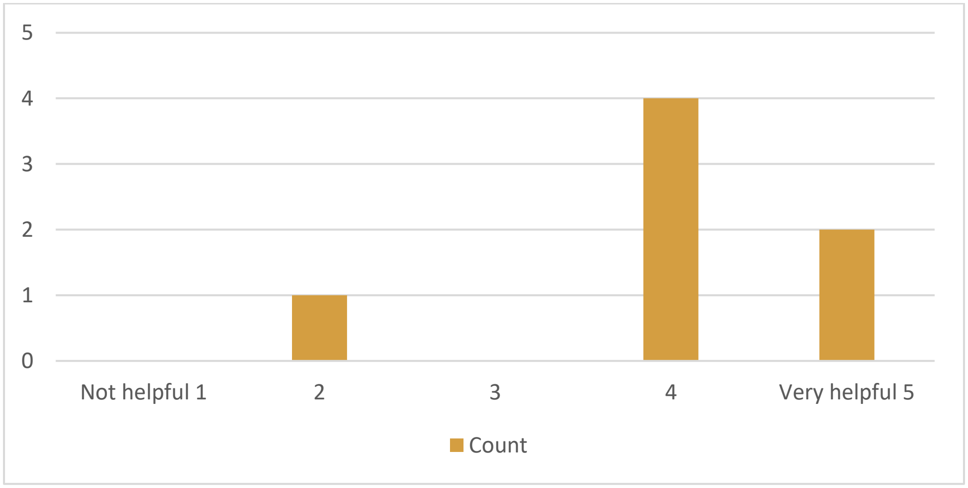


Figure 44 Helpfulness of Test Case Editor

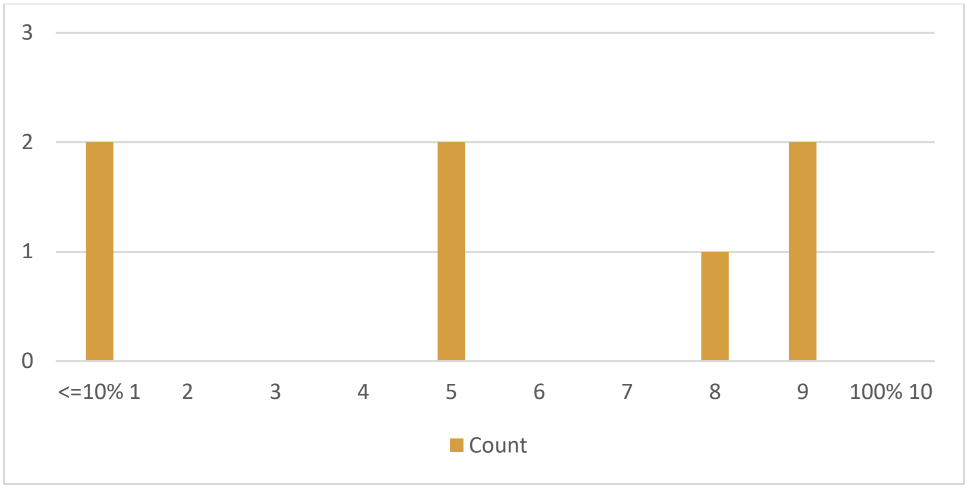


Figure 45 Coverage of Test Case-Based Grading In Teaching

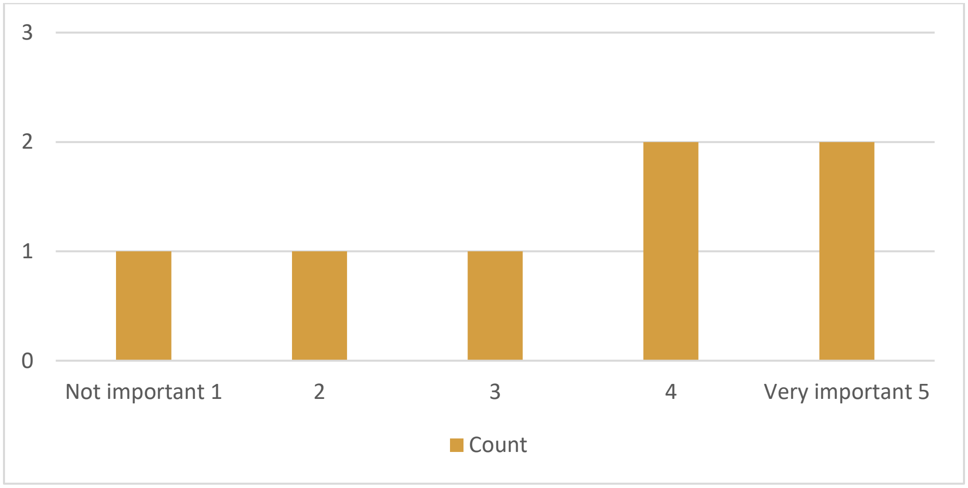


Figure 46 Importance of Having GUI Auto-Grading In Teaching

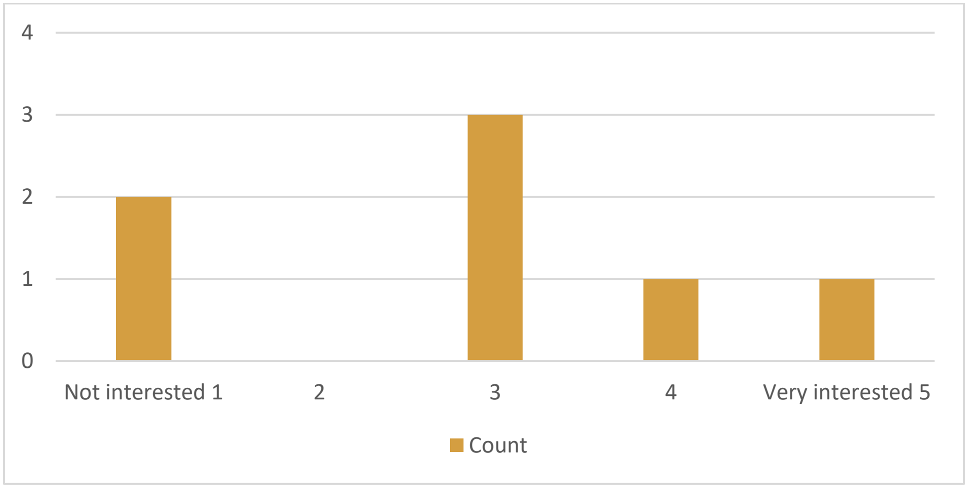


Figure 47 Interest of Respondents on Implementing Custom Grading Stages On ZINC

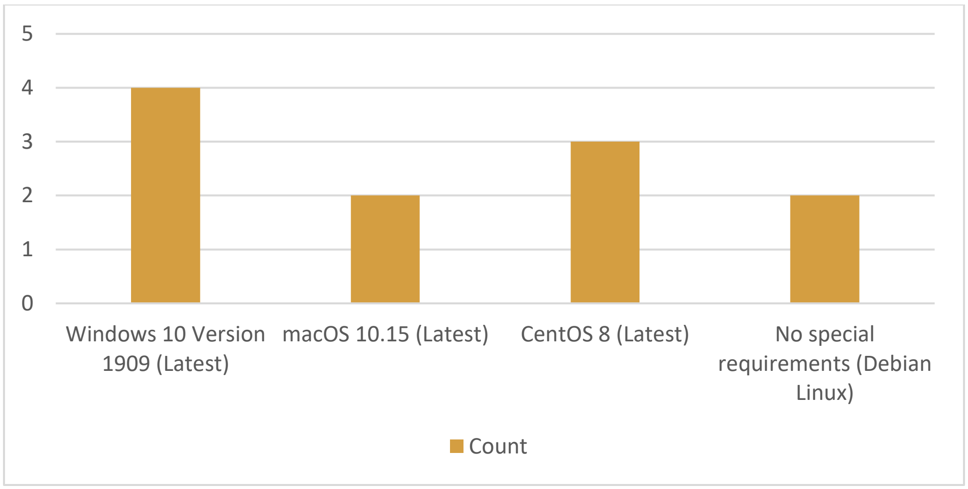


Figure 48 OS Requirements for Assignment Grading

Other Suggestions

- Compare scoring patterns among submissions to identify common weaknesses and detect possible plagiarism.
- Canvas integration to automatically grade submissions collected via Canvas.

UI Details, Justifications, and Screenshots

Student UI

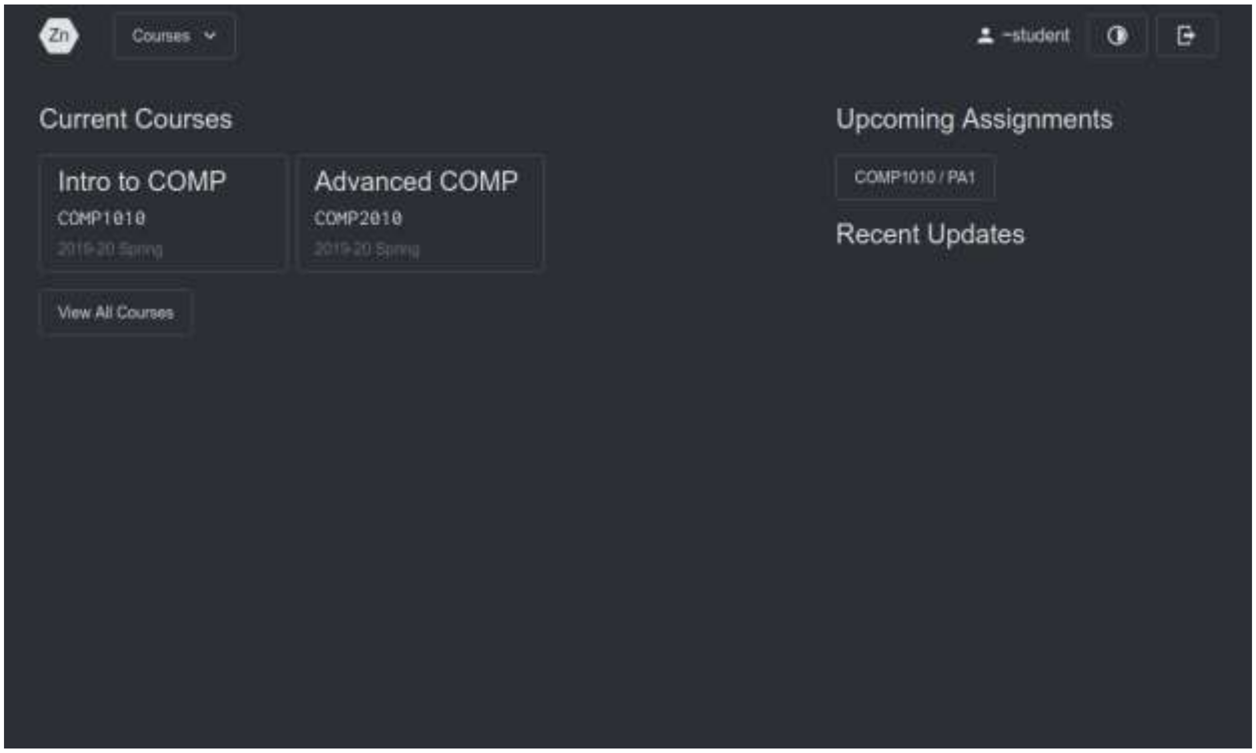
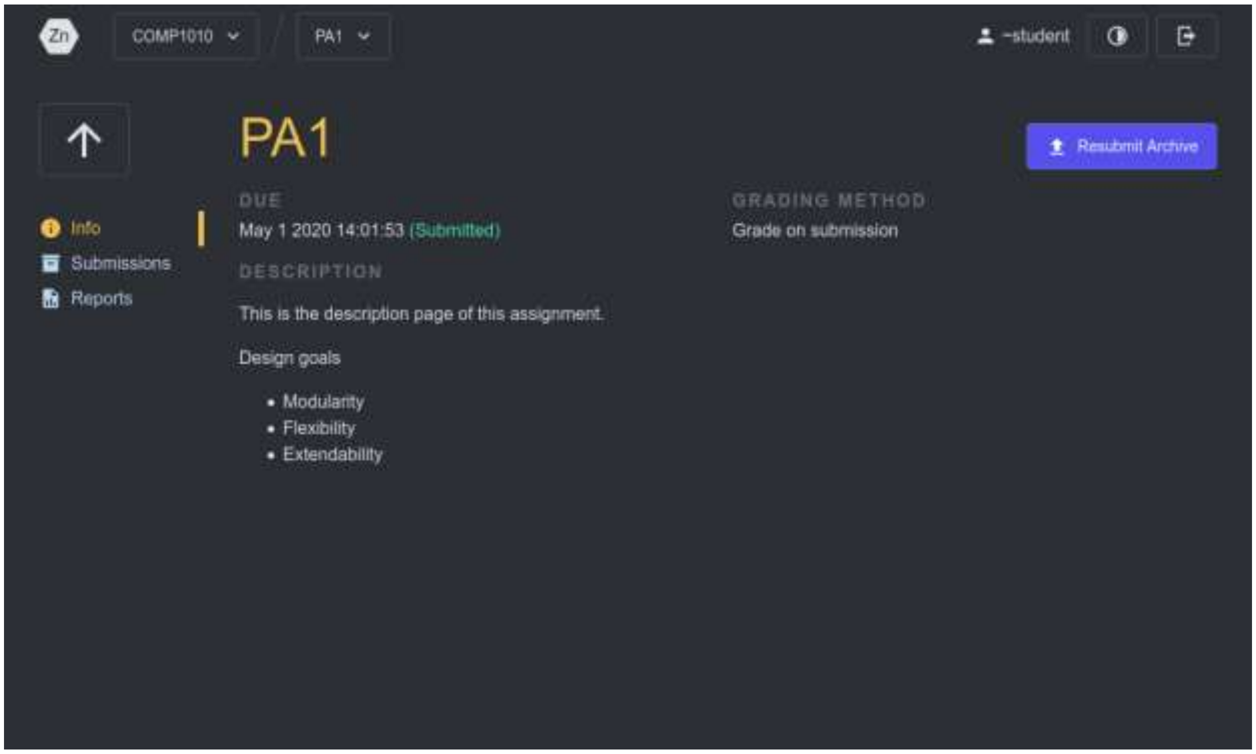


Figure 49 Student View Homepage Screenshot

The student is presented with a summary page of courses and upcoming assignments when they first enter the site. The student may navigate to their relevant courses in the current semester, upcoming assignments before due dates, and a page listing other courses they have taken.



PA1

DUE

May 1 2020 14:01:53 (Submitted)

GRADING METHOD

Grade on submission

DESCRIPTION

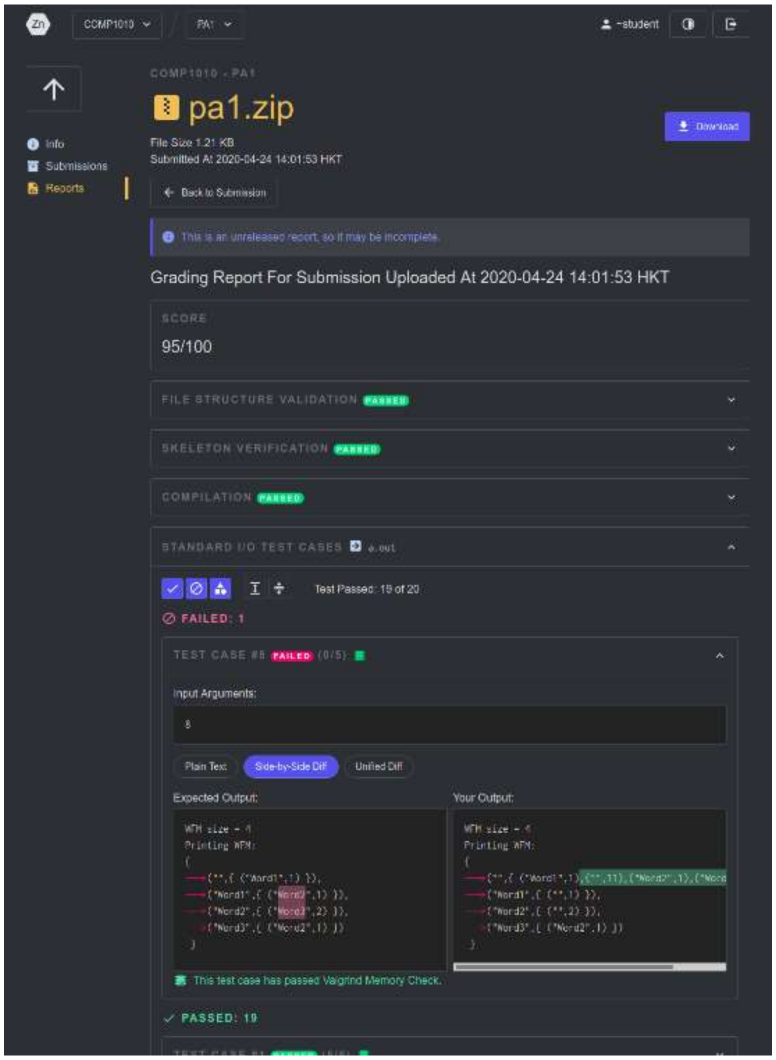
This is the description page of this assignment.

Design goals

- Modularity
- Flexibility
- Extendability

Figure 50 Student Assignment Page Screenshot

The assignment page allows students to view a descriptive page about the task, other relevant information relating to the assignment, and to easily navigate to other related pages like Submissions and Reports.



COMP1010 - PA1

pa1.zip

File Size 1.21 KB
Submitted At 2020-04-24 14:01:53 HKT

This is an unreleased report, so it may be incomplete.

Grading Report For Submission Uploaded At 2020-04-24 14:01:53 HKT

SCORE
95/100

FILE STRUCTURE VALIDATION PASSED

SKELETON VERIFICATION PASSED

COMPILATION PASSED

STANDARD I/O TEST CASES a.out

Test Passed: 19 of 20

FAILED: 1

TEST CASE #8 FAILED (0/5)

Input Arguments:

```
8
```

Expected Output:

```
WFM size = 4
Printing WFM:
{
  {"",{ {"Word1",1} }},
  {"Word1",{ {"Word2",1} }},
  {"Word2",{ {"Word3",2} }},
  {"Word3",{ {"Word2",1} }}
}
```

Your Output:

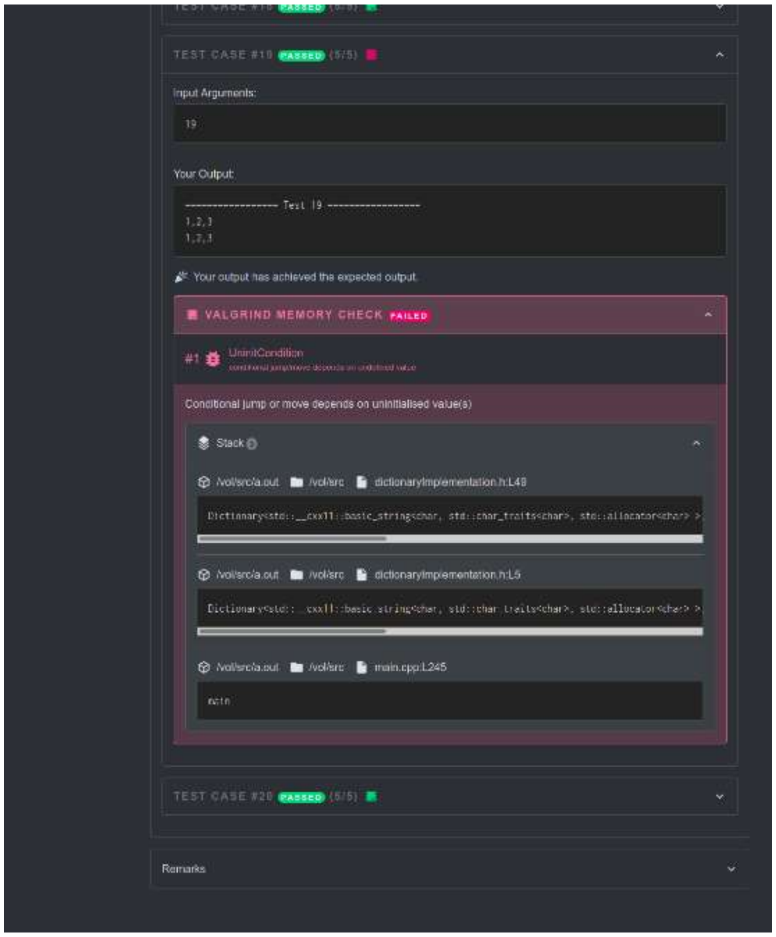
```
WFM size = 4
Printing WFM:
{
  {"",{ {"Word1",1},{"",11},{"Word2",1},{"Word
  {"Word1",{ {"",1} }},
  {"Word2",{ {"",2} }},
  {"Word3",{ {"Word2",1} }}
}
```

This test case has passed Valgrind Memory Check.

PASSED: 19

Figure 51 Student Report Screenshot A

This example report page showcases the design and layout of view of various stage reports from the generated report. Each stage is grouped into a collapsible tab. A place worth noting is the implementation of the test case result view. Students may selectively filter test cases to find the ones they wish to view. The pills are radio buttons showing different diff views, computed with Google Diff Match Patch [47] to form a readable diff. Invisible but important characters such as tabs, spaces, and newlines are rendered when required to show visual differences.


TEST CASE #19 PASSED (5/5) 🟥

Input Arguments:

```
19
```

Your Output:

```
----------------- Test 19 -----------------
1,2,3
1,2,3
```

Your output has achieved the expected output.

VALGRIND MEMORY CHECK FAILED

#1 UninitCondition
conditional jump/move depends on undefined value

Conditional jump or move depends on uninitialised value(s)

Stack

/vol/src/a.out /vol/src dictionaryImplementation.h:L48

```
Dictionary<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
```

/vol/src/a.out /vol/src dictionaryImplementation.h:L5

```
Dictionary<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >
```

/vol/src/a.out /vol/src main.cpp:L245

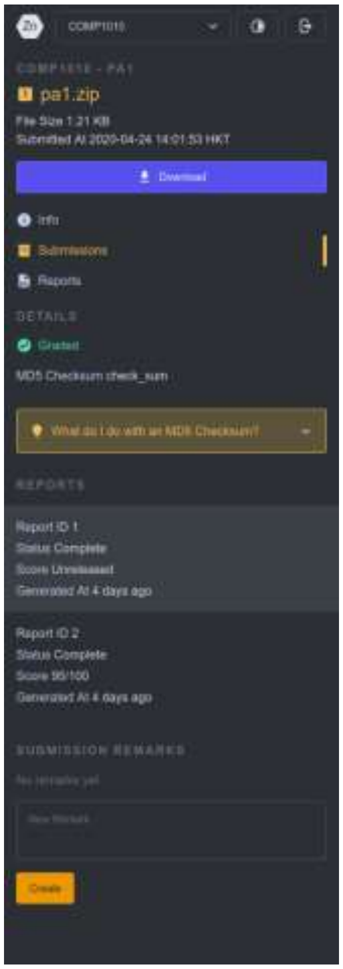
```
main
```

TEST CASE #20 PASSED (5/5) 🟩

Remarks

Figure 52 Student Report Screenshot B

This particular test case has failed the Valgrind memory check, required by the grading configuration. A separate report is generated for the student to view with relative ease. However, students may still be required to understand the function of Valgrind and its special error type names and meanings to fully understand the report.



COMP1010 - PA1

pa1.zip

File Size 1.21 KB
Submitted At 2020-04-26 14:01:53 HKT

DETAILS

Graded

MD5 Checksum check_sum

REPORTS

Report ID 1
Status Complete
Score Unreleased
Generated At 4 days ago

Report ID 2
Status Complete
Score 95/100
Generated At 4 days ago

SUBMISSION REMARKS

No remarks yet

New Remark

Figure 53 Student Submission Page Mobile View Screenshot

This submission page view showcases the responsive design of the UI. The layout of this page mirrors the assignment page in Figure 50 but adjusted under a small screen to allow for a comfortable viewing experience.

Administrative UI

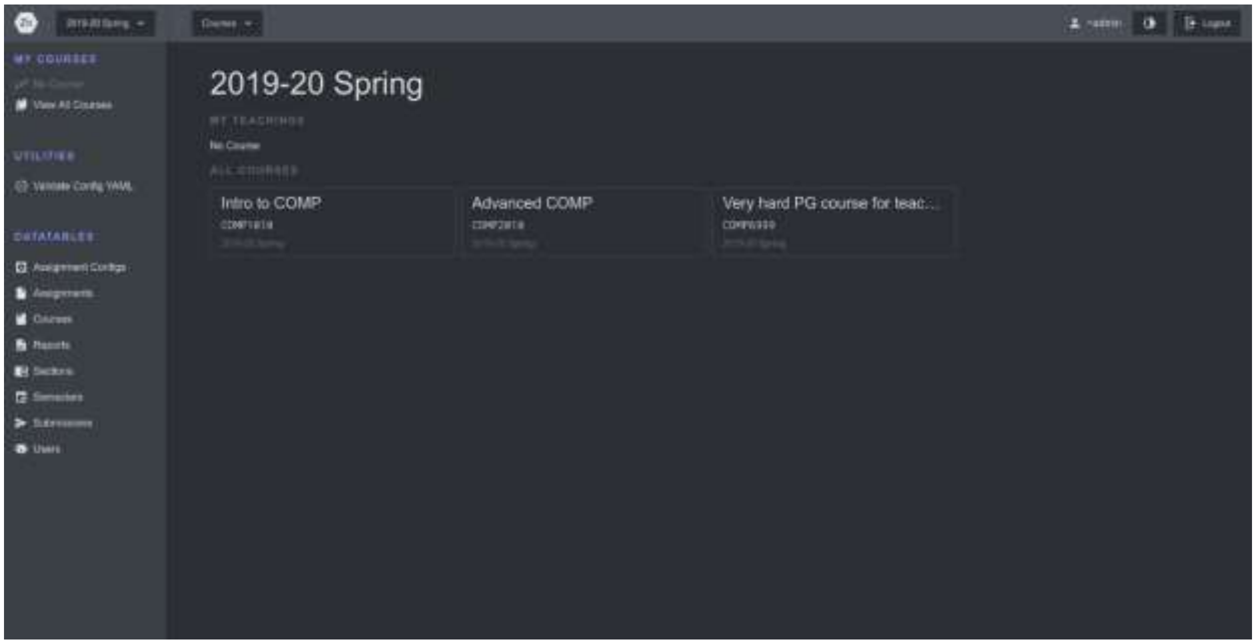
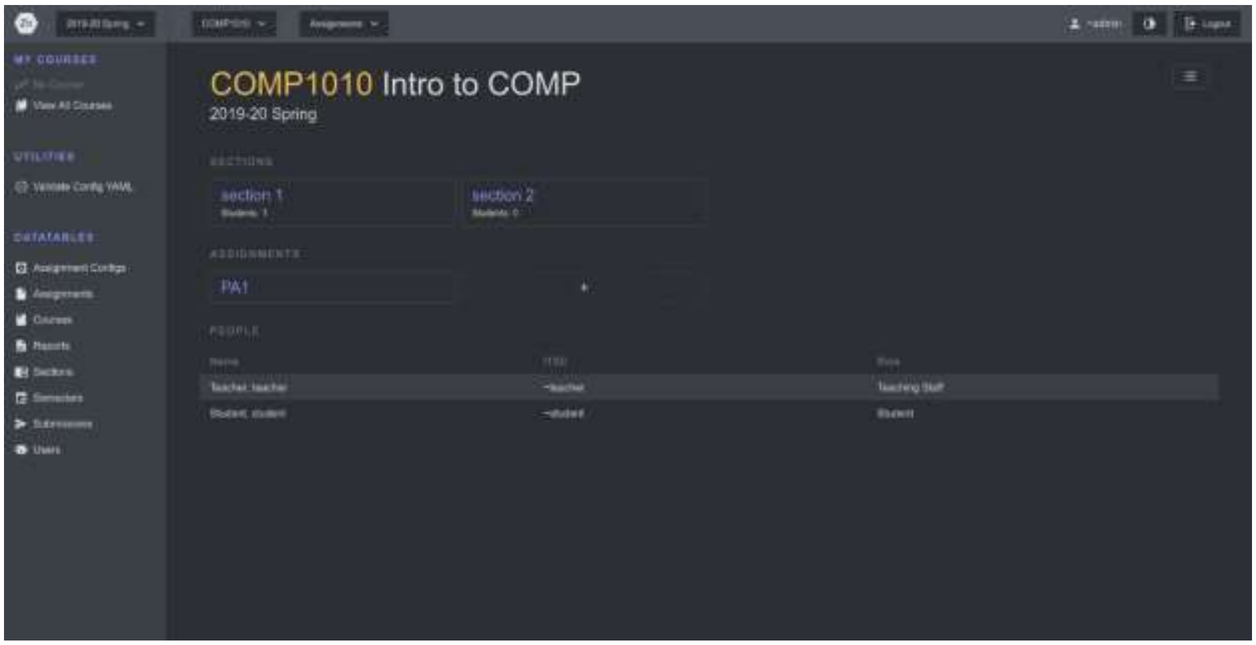


Figure 54 Administrative View Homepage Screenshot

This is the homepage of the administrative view. The layout is designed with intentional differences to differentiate the Admin UI and Student UI, as a platform user may simultaneously assume the role of a student in one course, and teaching assistant in another. Teaching Staff users would not be able to see as many items on the navigation bar as this administrative user in the screenshot.



COMP1010 Intro to COMP
2019-20 Spring

SECTIONS

section 1
Students: 1

section 2
Students: 0

ASSIGNMENTS

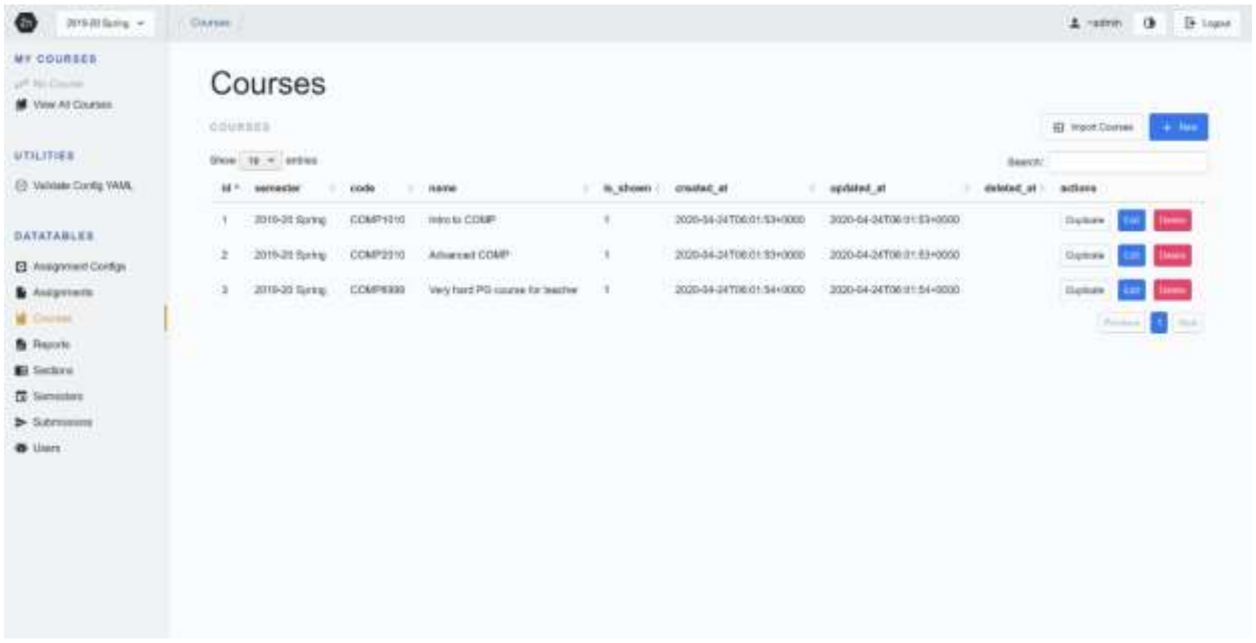
PA1

PEOPLE

Name	ITSC	Role
Teacher, teacher	~teacher	Teaching Staff
Student, student	~student	Student

Figure 55 Administrative Course Page Screenshot

This is the course page in the administrative view. Admin and Teaching Staff users can see sections, whereas this data construct is intentionally hidden in the Student UI. Assignments may be easily created from this page.



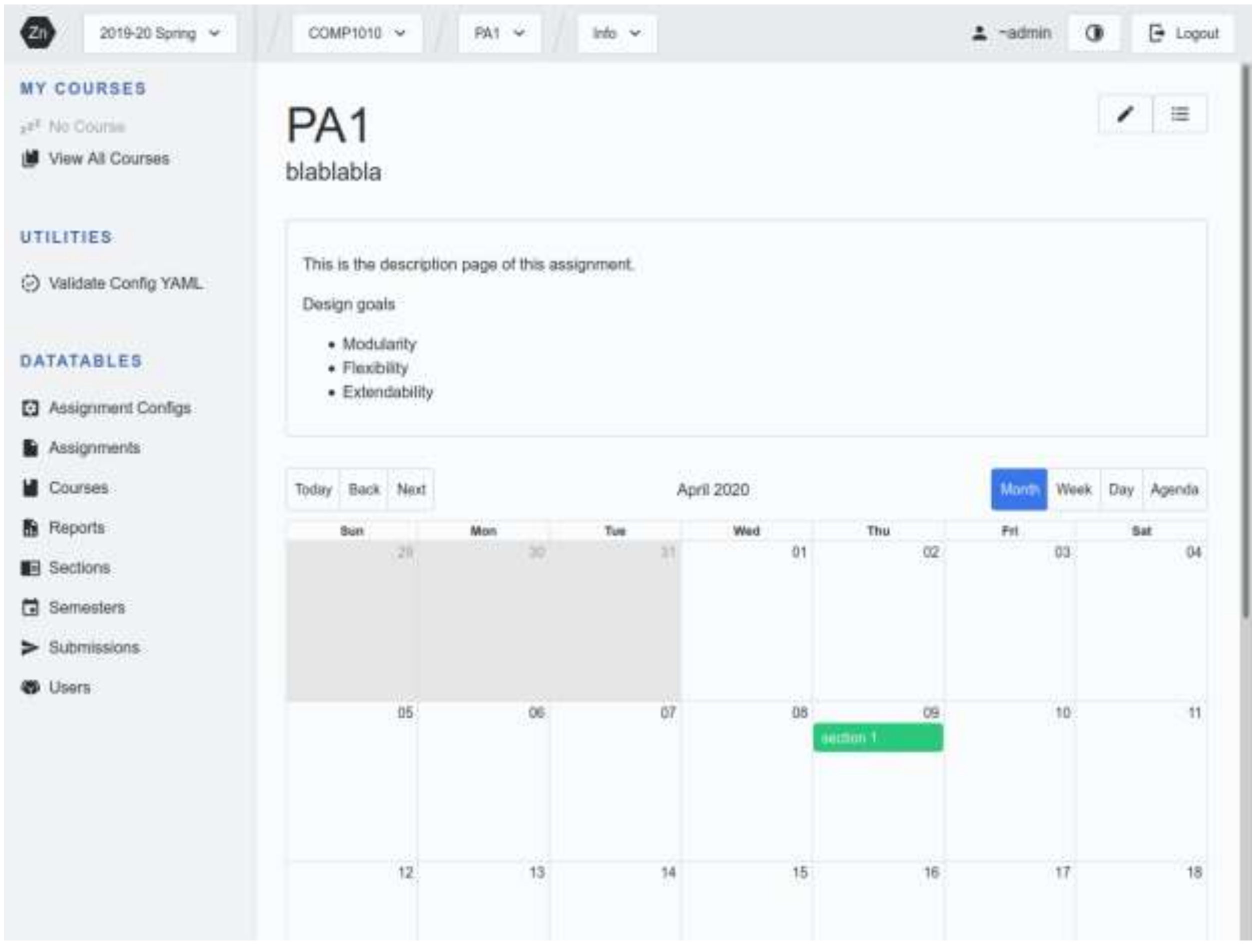
Courses

COURSES

id	semester	code	name	is_shown	created_at	updated_at	deleted_at	actions
1	2019-20 Spring	COMP1010	Intro to COMP	1	2020-04-24T06:01:53+0000	2020-04-24T06:01:53+0000		Duplicate Edit Delete
2	2019-20 Spring	COMP2310	Advanced COMP	1	2020-04-24T06:01:53+0000	2020-04-24T06:01:53+0000		Duplicate Edit Delete
3	2019-20 Spring	COMP6999	Very hard PG course for teacher	1	2020-04-24T06:01:54+0000	2020-04-24T06:01:54+0000		Duplicate Edit Delete

Figure 56 Administrative Course DataTables Page Screenshot

This is the DataTables Course page. This screenshot also showcases the alternative “light theme” of the site. The theme is automatically selective per user’s browser/OS setting, but may be manually toggled on the UI.



PA1

blablabla

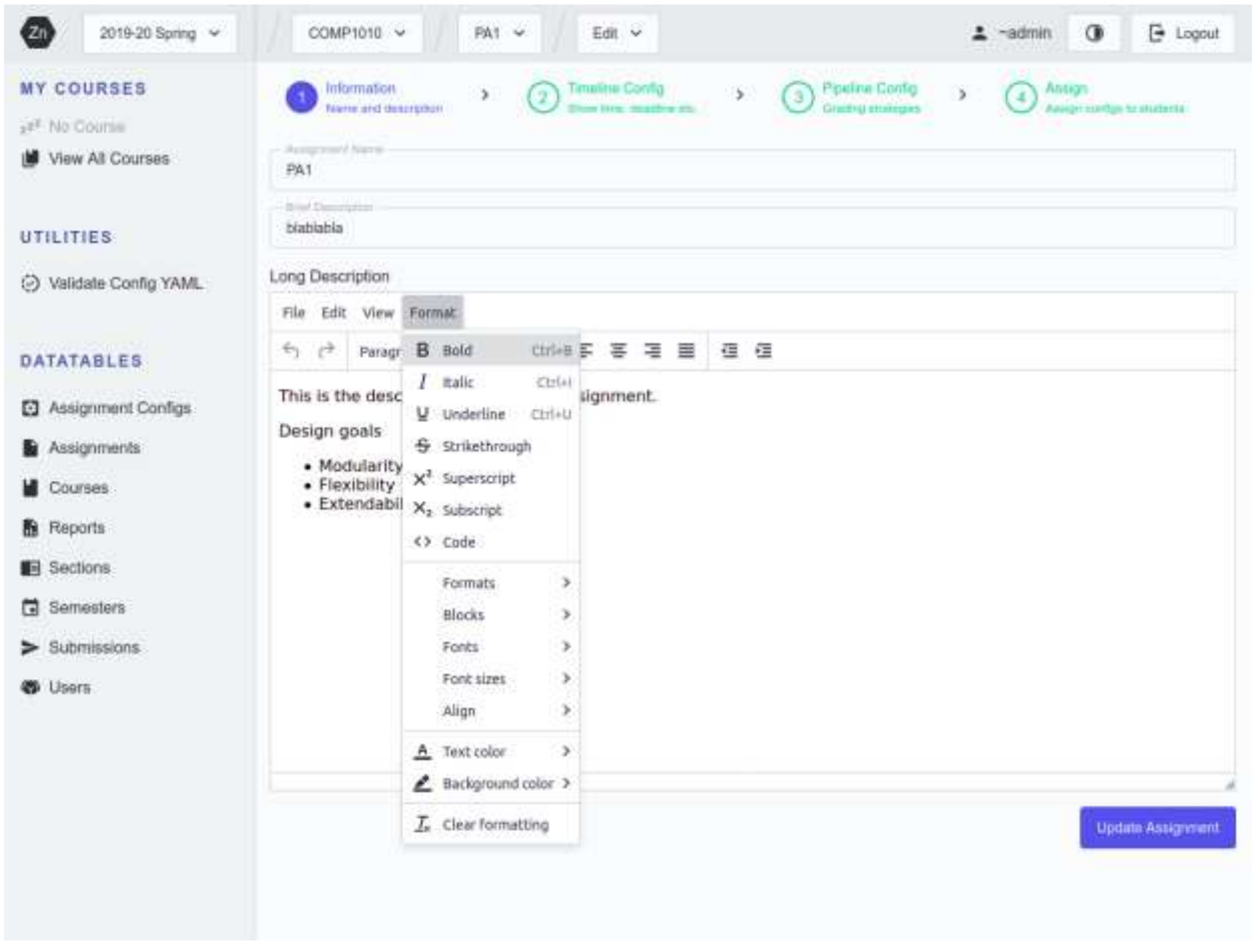
This is the description page of this assignment.

Design goals

- Modularity
- Flexibility
- Extendability

Figure 57 Assignment Page with Calendar Screenshots

This is the assignment page view, featuring a calendar view, allowing Admin and Teaching Staff users to see important dates of the assignment at a glance. The page also shows the description of the assignment.



Assignment Name
PA1

Brief Description
blablabla

Long Description

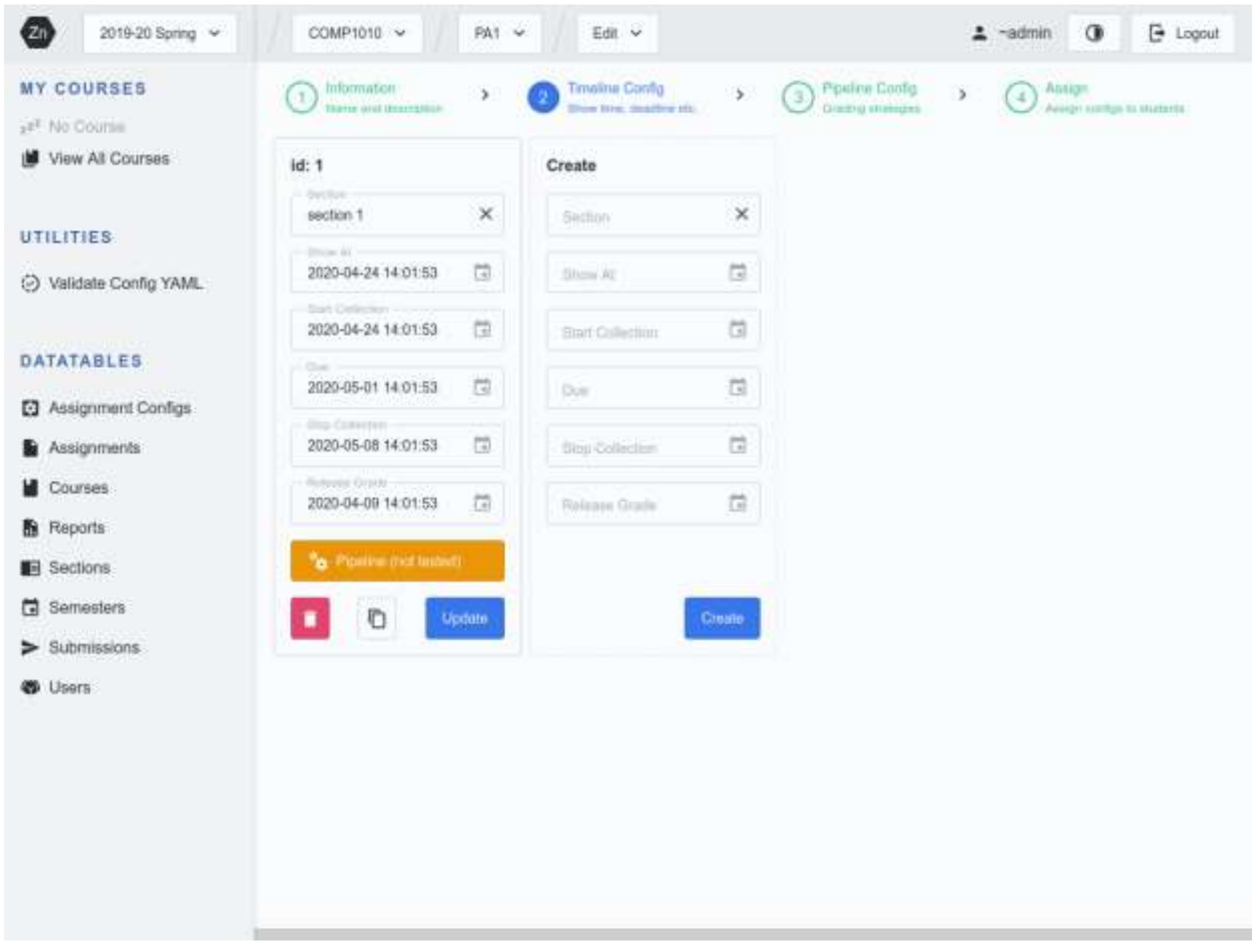
This is the desc … signment.

Design goals

- Modularity
- Flexibility
- Extendabil

Figure 58 Assignment Page Edit Page with TinyMCE Editor Screenshot

This is the edit page of the assignment. Steps taken from assignment creation are listed on top, where each step may still be configured. This page showcases the TinyMCE WYSIWYG editor integration in the UI.



id: 1

Section: section 1
Show At: 2020-04-24 14:01:53
Start Collection: 2020-04-24 14:01:53
Due: 2020-05-01 14:01:53
Stop Collection: 2020-05-08 14:01:53
Release Grade: 2020-04-09 14:01:53

Create

Section
Show At
Start Collection
Due
Stop Collection
Release Grade

Figure 59 Assignment Configuration Timeline Edit Page Screenshot

The Timeline describes the important dates of the assignment to different sections. Admin and Teaching Staff users may manage assignment configurations for sections on this page. Different sections may have a slightly different configuration, including grading strategy.

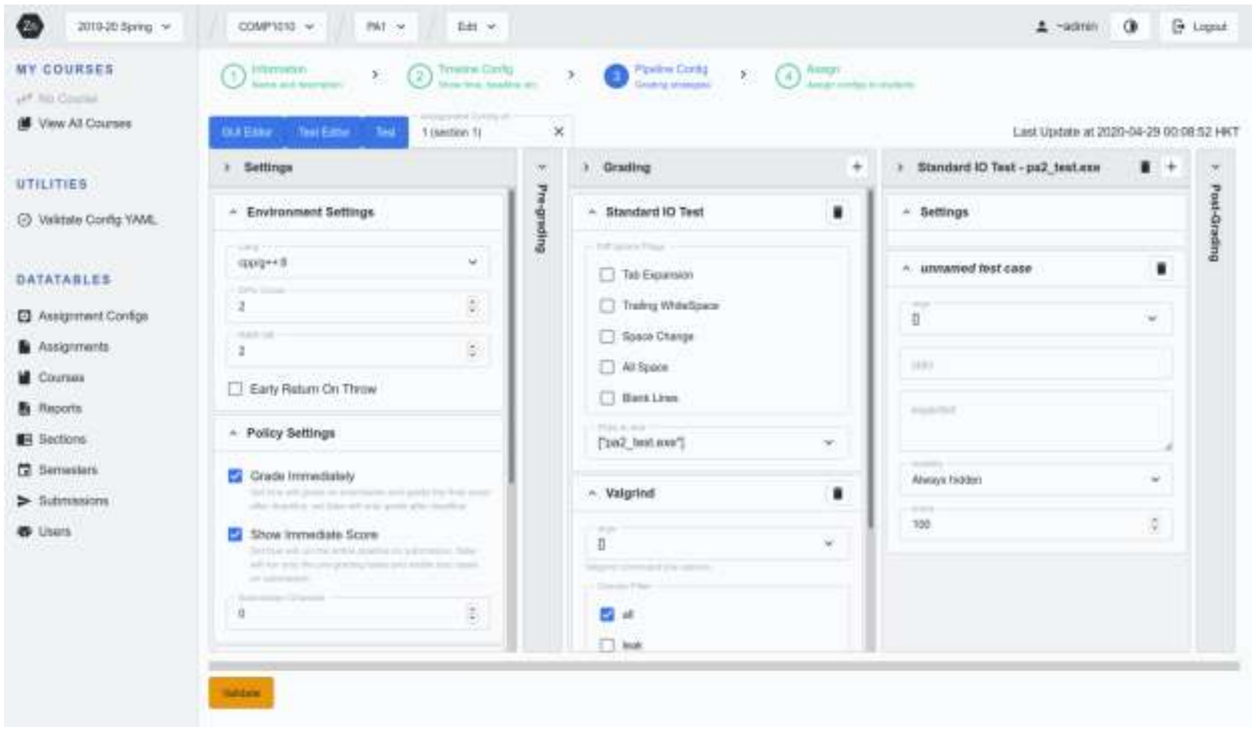


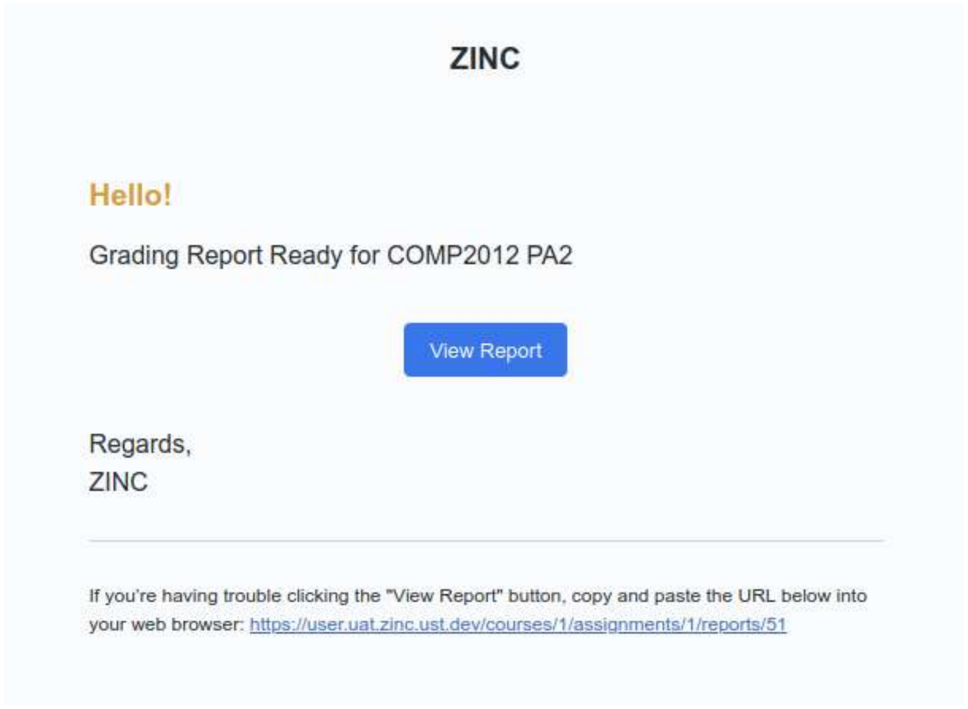
Figure 60 Assignment Configuration GUI Pipeline Editor Screenshot

This view shows the GUI Pipeline Editor, with the use of blocks and groups to represent the original YAML plain-text configuration. Admin and Teaching Staff users may choose to use this page or the text editor page, which features the Monaco Editor [48], the same editor used in Visual Studio Code, to edit the YAML configuration with some code assistance.

The user may validate the configuration with the Auto-Grader during modification.

Next to the Text Editor button is the Test button. The Submission Subsystem enforces that released assignments must pass the grading pipeline at least once, to ensure that student submissions and report generation runs smoothly. Unless the TA or instructor has tested the assignment, students are not allowed to upload assignments to the system to protect the interest of students. After each update to the assignment configuration which may affect the pipeline, the uploading of assignments is again frozen until the Teaching Staff test the pipeline.

Email View



ZINC

Hello!

Grading Report Ready for COMP2012 PA2

Regards,
ZINC

If you're having trouble clicking the "View Report" button, copy and paste the URL below into your web browser: https://user.uat.zinc.ust.dev/courses/1/assignments/1/reports/51

Figure 61 Grading Report Ready Email Screenshot

This screenshot is taken from Gmail. The email design mirrors the design language of the platform, to provide a smooth UX and assert product recognition. Various emails are sent to the email address of stakeholders of changes on the platform which are worth noting.