Satori: Open-source Course Management System

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Disha Jain
Spring, 2020

Technical Project Team Members
Andrew Lewis
Winston Liu
Austin Sullivan

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Signature Disha Jain Date 05/03/2020
Disha Jain

Approved Aaron Bloomfield Date 05/09/2020
Dr. Aaron Bloomfield, Department of Computer Science

Satori: Open-source Course Management System

Disha Jain
dishajain@virginia.edu

Andrew Lewis
ajl5yc@virginia.edu

Winston Liu
winston.liu@virginia.edu

Austin Sullivan
acs3ss@virginia.edu

Figure 1: The welcome page for the Satori application, 2020.

ABSTRACT

Satori (also a Japanese Buddhist term for awakening, "comprehension; understanding") is a new course tools system built to provide an open-source solution for managing any course with any number of students. It is designed primarily for computer science professors to provide a consistent approach to course management while offering the flexibility to customize each course as necessary. Satori aims to provide user-friendly technical solutions for course management, assignment submission and grading, office hours, and support tickets with streamlined and easy-to-use tools that are able to handle large numbers of users without issue. It incorporates useful features from various different course management tools that exist today, combining them into one all-inclusive solution.

CCS CONCEPTS

• **Software and its engineering → Software design engineering**.

ACM Reference Format:
Disha Jain, Andrew Lewis, Winston Liu, and Austin Sullivan. 2020. Satori: Open-source Course Management System.

1 INTRODUCTION

This project was inspired by CS 2150: Program and Data Representation at the University of Virginia. CS 2150 used a course management system built by Professor Aaron Bloomfield over a period of 13 years to suit the needs of the course. The original system contained an office hours queue, assignment submissions, assignment grading, a gradebook, regrade functionality, and support tickets. Although the system worked very well, we saw an opportunity to upgrade the system with new technologies and more features. Our goal was to create a system that is easier to maintain and is abstracted out in a way that it can be implemented at any school, for any course.

Our development team consisted of Disha Jain, Andrew Lewis, Winston Liu, and Austin Sullivan, and this system is our 4th Year Engineering Research Capstone project. Our system, Satori, was built using Django 2.2 and Python 3.7 and is currently being run on a server, Pegasus, within the UVA CS department domain.

2 RELATED WORK

Based on our research, we identified several pieces of existing software that address some of the individual challenges that must be solved in any unified course management solution, but we did not discover any application that solved every problem. The main issues that arise in these current partial solutions are a lack of API support for integration, paid features, and licensing in opposition to the open-source solution we wished to create.

2.1 CS 2150 course manager

The course management system used in CS 2150 has a basic office hours queue, assignment submission and grading flow, as well as functionality for support tickets and regrades. It is written in PHP

and has become increasingly difficult to maintain as features get tacked on as needed over the years, and as the language has lost popularity. The web application's assignment submission system is strictly built to handle coding assignments; it is not designed for grading written assignments, a common form of exam assessment.

2.2 TPEGS

TPEGS (Tablet-based Paper Electronic Grading System) is an application separate from the above course manager, built to reduce the time spent grading paper assessments [2] for the entire UVA Computer Science department. TPEGS is a system that allows digital grading of paper assignments, and requires grading to be done on a per page basis. It handles simultaneous grading and has desirable features such as background loading for graders and detailed statistics for professors [1]. Grade retrieval and user experience could both be improved. It similarly has the issue of using an older code base that is more difficult to maintain.

2.3 Gradescope

Gradescope is a paid submission grading service [3] that has many features that would improve on the Pedagogy and TPEGS submission and grading systems. Gradescope has a well-designed user experience, and it allows the option for grading to be done on a per question basis so that one grader can specialize and complete the grading process quicker. The concept of regrades are built into the grading system unlike the separate ticketing system that Pedagogy uses. The web app also includes autograding for programming assignments, an important feature we would like to include in our final solution. Gradescope's lack of an API makes it a challenge to integrate into our project, and using a paid service for a non-optional core feature would be counter-intuitive to our mission of making an open source application that is accessible to everyone.

2.4 Submitty

Submitty is an open source project that has extensive autograding capabilities for programming assignments [4]. It includes additional tools for analyzing networked and distributed programming challenges, database assignments, code coverage, and memory statistics. The software also includes plagiarism detection and allows for team submissions. Additionally, manual grading can be utilized to assess qualities of the code that cannot be unit tested. The grading for non-coding assessment lacks many of the features fleshed out in Gradescope. Submitty has a very basic first-in, first-out office hour queue system. The main issue identified with Submitty is its reliance on the PHP language for its codebase. The API for Submitty is still in development, so integration with this service may be possible in the future.

Our goal is to take inspiration from the best features of each existing system to develop a manageable application that meets the requirements of the CS 2150 staff and students as well as the foreseeable needs of any potential course, all while maintaining our commitment to open source design principles.

3 SYSTEM DESIGN

Satori is an application that can be separated into four primary systems: courses, permissions, queues, and rubrics. Careful consideration went into designing each of these systems.

3.1 Courses

The courses system was designed to very strongly mimic the general design of university courses. When a professor goes into the system to create a course, he or she first creates the overarching course (for example, CS 2150: Program and Data Representation), which is section agnostic. Then, after the course has been created and enrolled students' user profiles are linked to the course, the system assigns each student to the various sections they are in, whether it is a lab, discussion, or lecture.

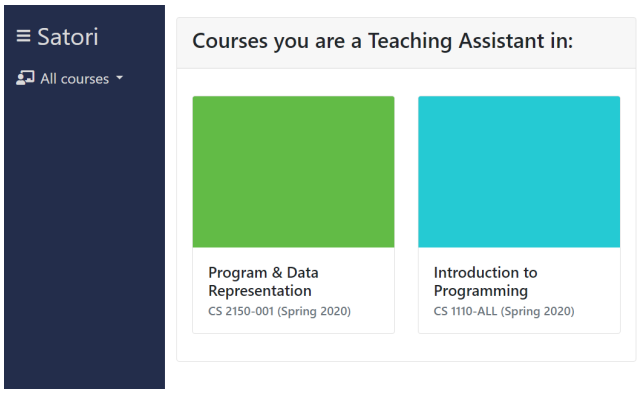

Figure 2: View of the Satori courses screen for a TA.

The courses system was an interesting one to design. The goal was to create a system that would work for any course, regardless of the structure, and to provide professors with a maximum amount of flexibility while still maintaining ease of use. The first iteration of the design was one where a professor simply created a course and added all the students in the course. The issue we faced with this was that there was no way for professors to set up their assignments, queue, and tickets for individual sections. We then transitioned to a system where the professor uploaded rosters for each section individually, and would have the option, on creation, to link multiple courses. This created a different issue, however. For courses with labs and lectures, for example, professors would need to enroll students in several different sections, and then students would need to be responsible for knowing which section to access at which time.

The current iteration of the course system tries to fix the failings of the two older designs. Because this system creates an overarching course first and then assigns students to sub-groups, the professor has the ability to set up their course tools either by section or by course, whichever they prefer.

An important part of the course creation system was roster parsing. This is the part of course creation where instructors upload a CSV file containing information about each of the members of the course (other instructors, the teaching assistants, the students, etc.). Roster parsing was built in as a separate library of functions

that are called by our app, and are all purely CSV parsing and string manipulation. It primarily handles input cleaning, dynamic role classification, and name parsing. The purpose of the roster parsing library is to standardize any roster input in a way that can be easily managed by Satori.

The most important consideration while we were developing this system was to ensure instructors will be able to use it however their course requires without sacrificing the user experience. This meant we often had to make difficult decisions regarding technical features to abstract away in exchange for a simpler user experience.

3.2 Permissions

In any course, there are multiple roles to consider: students, teaching assistants, professors, and perhaps more, depending on the course layout. Each role has different responsibilities and priorities, which a course management system must be able to handle. For example, only students should be able to submit tickets, while only teaching assistants and professors should be allowed to view or respond to tickets other than their own. Additionally, there should be clear separation between each course — that is, each course should be completely isolated from each other, and permissions should not be transferable between courses. Both of these constraints presented significant design challenges.

While Django has a highly robust permissions system centered around models, it does not yet provide a way to assign per-object permissions. Thus, while it is easy to give a user the permission to submit tickets for every course, it is much harder to give them the permission to submit tickets in one course while restricting that ability for another course they are enrolled in. To work around this issue, we integrated our project with django-guardian, a package that gave us the ability to assign permissions per-course. Unfortunately, while django-guardian solved the problem of user permissions, it did not provide a solution for encapsulated *groups*.

Like model permissions, groups are global in Django. Thus, we were forced to extend Django's concept of groups to better fit a paradigm in which a user can be in multiple groups at a time, each of which has its own unique set of permissions depending on the course and should not be discoverable outside of its respective course. We settled on an intermediary "role" model that would link courses and groups together. In this way, the role is able to store auxiliary information such as a human-readable display name while delegating the permission handling to unique groups behind the scenes.

Professors are able to create new roles as desired, allowing considerable flexibility in how a professor wants to run their course. We designed the permissions system to be full-featured without being too overwhelming; professors are able to fine-tune what each role should have access to through an easy-to-use checkbox system.

Occasionally, a situation arises where a professor needs to perform an action on behalf of a student. Satori handles this possibility by adding a special permission that allows users to transparently log in as someone else in the course while still retaining their own account permissions (impersonation). The controlling user is still able to view everything without being restricted by the permissions of the user that they are acting on behalf of, but they are only able to perform actions that the controlled user can also perform. Satori also implements proper access control that takes into account the multi-layered nature of courses — like every other action, impersonation is restricted by course, to prevent controlling users from seeing details of other courses.

3.3 Queue

One of the major features of our system is the office hours queue. A queue should be an efficient way to provide order to an office hours session - and nothing more. It needs to provide information quickly and accurately when you need it, and then get out of your way so you can get on with helping students. From a purely functional standpoint, making the queue fast and reliable is a top priority. Teaching assistants checking the queue during office hours require an immediate determination for who is the next student ready to be helped. This is especially important when office hours are busy and the queue grows to a considerable length; the more time that's spent waiting for a response from the queue, the less time the teaching assistants are actually helping students when they need it most. Extensive performance tests were conducted to ensure the queue performs acceptably, even under significant load. Various potential optimizations were tested, with the solution that best balanced simplicity and speed making its way into production.

An important consideration when designing an office hours queue is the order in which students will be helped. The most straightforward solution is to simply assign students based on the order in which they enter the queue, but one might imagine that a professor may want to order the queue in a different way, such as by the students seeking help on the earliest due assignment or prioritizing students who haven't recently been helped. A notion of priority is needed. We have provided a handful of suggested algorithms that can be chosen from by the professor upon creation of the queue.

As part of our larger goal of making this system easily adopted by other courses, an intuitive user interface was a priority in the development process. The queue system should ideally not require a deep technical background to operate. We designed the buttons to be clear and unambiguously labeled. The web page is easy to use from both desktop and mobile devices, to allow for teaching assistants to manage queues from their phones.

3.4 Rubric

The current Satori app includes the design of a basic submission system. It allows a gradebook item to be created by an instructor, specifying the title, category, instructions, file attachments, the files required for submission, and dates for when the assignment should be considered late or closed. A student can then submit the assignment, attaching the required files and including any comments, and the instructor will then be able to view, grade, and supply feedback on the most recent submission for each student.

While it is adequate for simple assignments, this initial iteration is not yet ready to replace the existing functionality of TPEGS that allows multiple TAs or instructors to grade exams simultaneously and efficiently as possible. To achieve this and the features we found to be useful within Gradescope, we need the additional concept of rubrics on the gradebook items. While this has not been fully

implemented, we have considered the design of the system and how it should function within the Satori application.

In order to facilitate the per-question grading model, each gradebook item would be able to have multiple questions that stored both the physical location of the question, common in every submission, as a rectangle object along with the question's rubrics. Knowing the question's location within the pdf document will allow Satori to focus on the same question for each student while navigating multiple submissions. Checkouts should be done at the question level. These question-level rubrics would have multiple options of point value and description pairs as well as a field to make a positive or negative point adjustment. The point adjustment and the value of the selected option will be used to calculate the raw score for the question. The accumulation of every question's raw score will compose the raw score for the assignment.

At this point, modifiers should be applied to the raw score to compute the effective score for the gradebook item; these modifiers should be able to add (bonus points), multiply (late penalty of a certain percentage), and override (cheating sets the score to 0). It should be made clear to the instructor in which order the modifiers are applied. Having the two separate scores, effective and raw, ensures that the history of the grading process is preserved.

4 PROCEDURE

Satori is designed to be very natural and intuitive in all cases. There are two aspects of Satori, however, uploading a roster and using the queue, that require a little more explaining!

4.1 Roster Uploading

In order for Satori to successfully parse an uploaded roster, there are some restrictions on the format and structure of the roster file. We require two different CSV files to be uploaded, one that is the entire roster and another that defines the groups each student is in. The more specific restrictions are included in the system documentation. For courses at the University of Virginia, any roster exported from Collab will be compatible with the system.

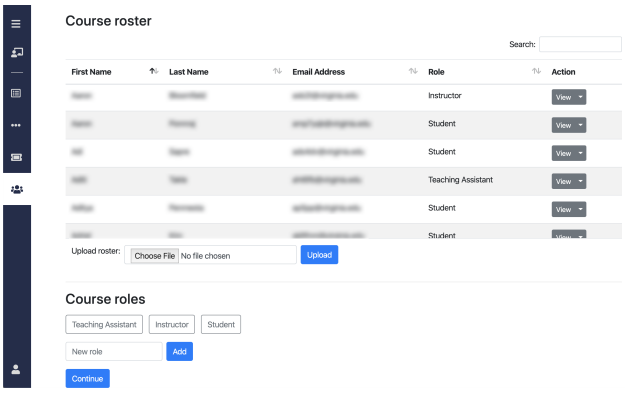

Figure 3: TA and instructor view of a sample course roster.

4.2 Using the Queue

Using the queue is meant to be very intuitive. First, a professor or a TA will go in and create a queue. Queues are meant to be specific to each section of the course. When creating the queue, they will define the algorithm with which students will be helped and the time the queue opens. Once the queue has been created and is open, they will see a dashboard that will list every student in the queue as they join, with a wide array of options. For example, they can take the student with the highest priority according to the queue algorithm, help specific students on the list, manually close/open the queue, add announcements, and so on. When helping a student, they are able to leave comments for other TAs to view and can requeue them if they feel another member of the course staff would be better suited to answering.

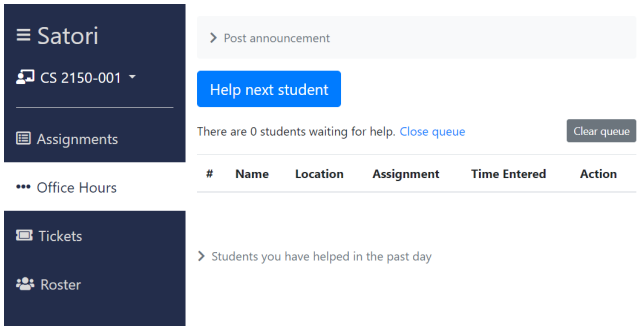

Figure 4: TA view of the Satori queue when it is empty.

For students, they are able to see the queue once it has been created and are able to join it if it is open. When joining, they will input their location and select the assignment they need help with (or none), and then wait until a TA helps them. If they no longer need help, they are also able to take themselves off of the queue.

5 RESULTS

Over the course of the Spring 2020 semester, the Satori queue was successfully used in two courses at the University of Virginia, representing a combined enrollment of over 1000 students. Whereas the old course tools would occasionally struggle when more than 50 students were on the queue at one time, Satori has thus far proven to be reliable under the same circumstances and has processed over 2000 queue entries within the span of two months.

From informal surveys given to students and teaching assistants, most respondents were pleased with the level of usability it offered and rated the system an average of 5.5/7. Specifically, TAs enjoyed the improved commenting feature in order to leave notes to other TAs, "the ability to re-queue students allows TAs to address the fact that another TA may be better suited to help a student without requiring that they physically...find the other TAs", and the estimated waiting time as a way to "[monitor] TA progress and [reallocate] teaching resources". Students considered Satori to be "an improvement from [Pedagogy]", providing a "more streamlined and modern" user experience.

6 CONCLUSION

As our 4th year capstone research project, we built a course tools system to replace the aging CS 2150 course management system and designed it to be able to be used at any school. We focused on four areas — course management, assignment submission and grading, office hours, and tickets. By the end of the school year, portions of it were already in use by over 1000 students and interest was being generated among other faculty due to its ease of use and its general-purpose nature.

7 FUTURE WORK

One of the most significant features from the original course management system that we did not implement was ticketing. Early in the semester we found a promising Django package which seemed to have all the features we needed, but we have yet to build this out.

Another area that needs work is course creation. As detailed in the "Courses" subsection, we changed how sections are handled partway through the development process. The design change was prompted by urgent demand for our office hours queue, but the addition of section-specific functionality across the system is still a work in progress. Our vision is to provide similar ability to manage each section of a course as for the course as a whole, but implementing these abstractions - while maintaining an intuitive view for the professor - will require more work.

Finally, there are a number of small improvements that could be made to the office hours queue. The most compelling of these is using web sockets to automatically update the contents of the queue page. Currently, the queue updates by refreshing the queue web page every fifteen seconds. Queue logs are another interesting area to explore. All actions on the queue are recorded in the database, which is ripe with data ready to be exploited. Statistics such as how long students wait on the queue, what percentage of the class attends office hours, or how long each TA spends helping each student may provide valuable insights to a professor. An easy-to-use interface exposing this data to professors could be a distinguishing benefit of our system.

REFERENCES

[1] A. Bloomfield. 2010. Evolution of a digital paper exam grading system. In *2010 IEEE Frontiers in Education Conference (FIE)*. IEEE, Arlington, VA, USA, T1G–1–T1G–6.

[2] Aaron Bloomfield and James F. Groves. 2008. A Tablet-Based Paper Exam Grading System. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education* (Madrid, Spain) *(ITiCSE '08)*. Association for Computing Machinery, New York, NY, USA, 83–87. https://doi.org/10.1145/1384271.1384295

[3] Gradescope. 2020. *Gradescope*. Turnitin. Retrieved May 3, 2020 from https://www.gradescope.com/

[4] Submitty. 2020. *Submitty*. Rensselaer Center for Open-Source. Retrieved May 3, 2020 from https://submitty.org/