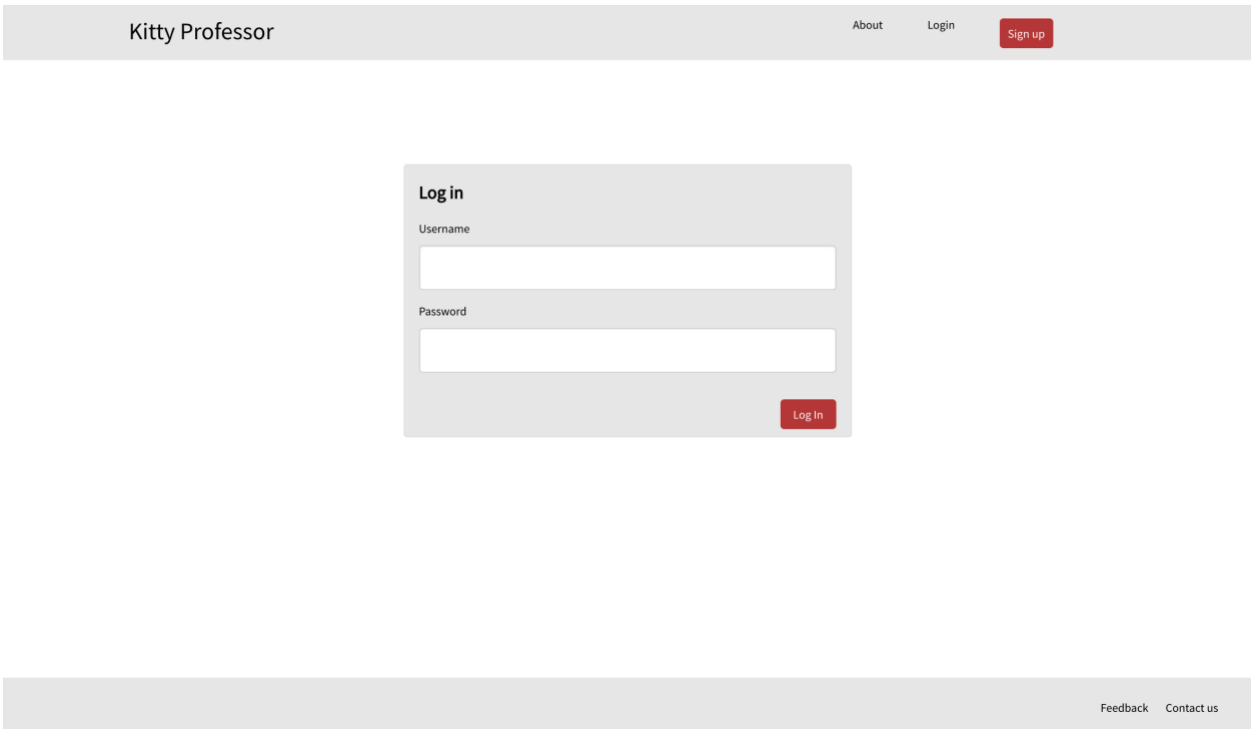
Managing Software Development – CS5500

Plagiarism Detection System



Team Members:

Balaraj Venkataramanappa: balaraj@ccs.neu.edu

Meng Tao: tao.me@husky.neu.edu

Nikitha Nagaraj: nagaraj.n@husky.neu.edu

Ravi Krishnan Iyer: iyer.rav@husky.neu.edu

Overview of the problem

This is a web application which focuses mainly on detecting plagiarism in assignments submitted by students. This application works as a single stop for students to submit their homeworks and for faculty to view these assignments and get a plagiarism report for the same. There are three roles of users that the system allows. The student role has the ability to add themselves to any of the existing courses by selecting a semester, a section under a professor and a course. Now, the student can check the application to see if the faculty has added any assignment under this course. If there is an assignment due, the student can select the assignment link and make a submission. The application allows the submission of homeworks as a zip folder or as a GIT link to make it more user friendly. One student can make multiple submissions to an assignment. The faculty role can create courses and assignments. When creating a course, the faculty can select a semester and add a course name to it. Now, if the faculty wants to add an assignment to this course, he can select this course and click on create assignment option. This will allow the faculty to enter a Name for the assignment, the due date before which the assignment needs to be submitted and a plagiarism threshold for the comparison algorithm. At this time, the faculty can also select the sections and semesters which need to be compared while detecting plagiarism. The faculty can also perform CRUD operations on the courses and the assignments created by him. The admin role can view the list of all the users in the system. This user can perform CRUD operations on all other users by changing their roles in the system. When the application loads, the login page shows up using which the user can log into the system. If the user does not have an account the system will prompt the user to first register for an account to use the system. Once registered, the user can now use the application to either submit assignments or view plagiarism reports. Once there are at least two students who have submitted to the same assignment, the system starts comparing their work and generates a plagiarism score. If this score or percentage of similarity is greater than the plagiarism threshold specified by the faculty while creating an assignment, an email will be sent to the faculty in charge with an attachment of this report. The faculty can view all the reports generated for an assignment by selecting the view reports link on the assignment block which he created. This will take him to a page which lists each pair of students along with their percentage of similarity next to it. The student pairs which

have a higher similarity value than the plagiarism threshold will be shown in red to grab the faculty's attention. The faculty can now view the similarity report. This report displays the percentage of similarity between the student. It also shows the files in comparison adjacent to each other along with the similar lines highlighted. The faculty can also send an email to the students involved in plagiarism.

Overview of the result

We set up tasks at the beginning of each sprint. During each sprint we progressed steadily by tasks number, and burnt down the tickets balanced in between team members.

Before sprint 3, we created a "candidate" branch for testers to test functionalities and hunt bugs based on sprint 2 requirements. After Homework 5, our testers raised more than 80 tickets as bugs. We then had a team meeting to carefully evaluate these tickets. At this time we found that 16 of these bugs were mostly to add new features and missing features or links. Out of the rest 19 of the bugs were sprint 3 requirements which we had already implemented by the end of sprint 3. After sprint 3 our focus was mainly on improving our project by implementing the stretch requirements, and solving the bugs raised. Till date we have 144 tickets on the jira board in total, including bugs and tasks. Among these 144 tickets, 1 of them is an open bug and 143 issues were solved by the end of the project submission. We have been able to successfully accomplish all the tasks provided for each of the sprints and fixed most of the bugs raised. Given below is a statistics of how our Jira board looks like.

Jira board:

	Sprint1	Sprint2	Sprint3
Tasks	17	22	25

Testing data:

In order to check the quality of our code in our Jenkins build pipeline at the end of each sprint, we setup SonarCube.

We are required to pass the quality stage every time the code is shipped on to master branch since the beginning of sprint 3. Our code is measured by reliability, security, maintainability, coverage, duplications, size, complexity and issues by SonarCube. By the end of the project, we have been graded A in reliability, security and maintainability. We have a code test coverage of 89.6% line coverage with no duplicated code.

To sum up, we have completed all functionality tasks that we set up on our Jira board for each sprint, fixed all bugs affecting the project reliability, and passed all code evaluations. Thus, we have accomplished a quality job after 3 sprints.

Development process

Initially we started off with gathering the requirements for the project. The team then started working on the different phases of the project. As a team our main agenda was to make sure that the work is divided equally among all the team members to make sure no one is overloaded. Prior to every phase, we would meet as a team and assign particular tasks among all of us on jira. We would keep a deadline a day prior to the actual deadline so that each of us could proof read the work of other team members and provide feedback. This process made sure that all of the team members were on the same page and were aware about how far we are in the development process.

Once we had a clear picture of the system architecture and the use cases, we started off with the implementation process. We initially discussed about the areas in which each of us were most comfortable with so that the work could be assigned accordingly. The implementation of the project was done in three sprints. At the beginning of each sprint all team members would meet, to understand the sprint expectations provided to us. We then made multiple tickets for the tasks listed. We added a jira task ticket for each of the features and assigned tickets to all the team members equally. In the first sprint we started off with assigning tickets based of the domain

each member was familiar with. However, as the Sprint 2 and Sprint 3 expectations came out we had to divide tasks from all sections in order to make the share of load even. Once we had tickets assigned, we would first add an estimate specifying how long the task is expected to take. We also created epic links for front-end and backend and assigned tickets under the respective tags in order to organize the Jira board to best suit our requirements. When we start working on a task, we would move the Jira ticket to the “in progress” section and when the task was completely integrated and ready to use with the entire system we would close the tickets using smart commits. In order to make sure all of the team members were aware of the progress of each task in the sprint we would provide an update to one another whenever we did not meet in person. This was done either on slack or on messages or email.

During the initial sprint, we were able to work individually and integrate our code later. However, as the tasks started to get more challenging we had to work together in person as there were more tasks for integration. Working at the same place together as a team rather that separately helped speed up our development process as all of us would work simultaneously on different tasks and once we wanted to integrate the front end with the backend we could do it with the least number of issues.

Retrospective of the project

Plagiarism detection system was a very interesting project to work on as we had an opportunity to learn a lot of technologies we were totally unaware of. We had a chance to learn how to use Spring Boot, Java, Thymeleaf , AWS , Jenkins, and other topics. Working on this project helped us understand the importance of team work and about how stressful it is to meeting deadlines. We were able to participate in the entire software development lifecycle and understand the different phases which a product goes through during its development. This helped us understand the actual importance of testing our application to make sure the quality of the system is maintained even when we add additional features to the application. We got to use tools like Jira to maintain our sprint tasks and slack to make sure every member of the team is aware of the development of the application. By working on this project, we learned the dynamic nature of software requirements.

However, there were a few challenges we faced during this process. When we were asked to start the project, the requirements provided to us were not very clear. So, until the first sprint we had a different set of requirements. These requirements kept changing and the expectations were clear only at the end of Sprint 2 which was very late in the process of development. As a result of which the TAs were themselves unclear about how to grade the different teams. For the Sprint 1 we had to write our own comparison strategy by generating AST trees. However, since the requirements changed after the sprint we had to totally discard this approach. We would have been able to work better and more efficiently if we had the requirements and expectations clear at the beginning of the first sprint as we could have planned our entire approach prior to the implementation.