CS5500 TEAM-6: Final Report

PROBLEM OVERVIEW

The project is designed and implemented as a web application so as to identify plagiarized solutions submitted by students. A solution is considered to be plagiarized if it is derivable from another solution, merely by renaming variables, changing the order of functions in the code, converting a sequence of statements into methods and so on. This application is for professors and other staffs to identify such plagiarised solutions and inhibit plagiarism amongst students.

The application runs the plagiarism check against source files written in the **Python** (version 2) programming language. The source files for student submissions will be uploaded in GitHub repositories. Users will have to provide URLs to repositories before the run. When run, our application starts a plagiarism check for every repository against every other repository. Since plagiarism is a relative concept, we enforce configurable 'match thresholds'. If the similarity of two submissions crosses border values, we mark them as different severity and show in the front-end. Each run will generate a report that will aggregate these similarities from three comparison strategies in the form of statistics for all submissions. Users will be able to see the report after the run.

The project comprises of three main components or layers: the front-end, the services/back-end layer and the database. The front-end layer is what is visible to the user and what the user interacts with. The goal of this layer is to facilitate easy, meaningful and smooth user interaction with the system. The front-end is implemented in React. The crux of the program lies in its back-end, implemented solely in Java as a Java web service. The front-end interacts with the back-end using REST services (APIs). All the logic for comparing submissions and detecting plagiarism resides in the back-end. The database layer provides a mechanism to store and retrieve persistent data. This is necessary for storing user objects, source files, generated reports, etc. The database is implemented using PostgreSQL, which is a relational database.

The entire development process implemented principles of the **Agile** model as it has an effective response to change and thus reduces the cost of operation. Our team had two scrum meets every week to discuss what we did, what obstacles we faced and what we plan to do in the coming days. Every alternate week (on weeks when a sprint review is not there), a team meeting was held to discuss how happy a team member is with his/her role and to peer evaluate each others' performances. The development team employed the "test as you build" philosophy throughout the project to maintain continuous code quality.

RESULT OVERVIEW

In sprint one, the development team mainly focused on the build of application infrastructure and some basic functionality expectations. We implemented: application infrastructure building such as: setting up Spring boot starter project to aws cloud(Jira ticket #CS106-1), integrating Postgress with springboot started project and deploy to AWS(#CS106-2), turning on story points in JIRA(#CS106-6), setting up jenkins with git integration(#CS106-7, #CS106-20), Slack integration for jenkins and GitHub(#CS106-8, #CS106-17). The front end is available for users to register and login as well(#CS106-9, #CS106-10, #CS106-15, #CS106-34, #CS106-36). AST generation (#CS106-11) and simple comparison strategies were finished as well(#CS106-13, #CS106-14).

There are 17 tickets created and closed in sprint one.

In sprint two, the development team mainly focused on detail and expansion of functionalities of the application. We implemented: System logs activity(#CS106-37), throwing a message when something bad happens(#CS106-38), three sophisticated comparison strategies and their test cases(#CS106-43, #CS106-49, #CS106-51, #CS106-53, #CS106-55), integration of front end's professor and admin dashboard with end points in back end(#CS106-32, #CS106-33, #CS106-41, #CS106-42, #CS106-46, #CS106-66, #CS106-68, #CS106-70, #CS106-71) and other necessary services.

There are 32 tickets created and closed in sprint two. Defects fixing started since this sprint.

In sprint three, our development team focused on the completion of the project, a totally new UI and most defect fixing work in this sprint. We implemented: a totally new UI(#CS106-72), an new admin dashboard(#CS106-73, #CS106-80, #CS106-86), filling in enough Java docs and test cases(#CS106-97, #CS106-99, #CS106-128) and many defects fixes.

There are 19 tickets created and closed in sprint three.

The defects fixing issue actually started after the sprint 2. We fixed many defects such as UI is not fully intuitive, lost comments and Javadocs in test cases and so on. There are less than twenty defects (after merging and closing) raised by the test team and ourselves. All defects were fixed properly.

According to requirements in each sprint from professor, all basic requirements and stretch expectations were achieved in our team's project. UI is intuitive enough and it is working very well.

TEAM'S DEVELOPMENT PROCESS OVERVIEW

The development progress is accumulated by achievements in sprint one, sprint two and sprint three. All basic and stretch expectations in these three sprints were completed.

As for the front end, our team spent plenty of time to make it intuitive. Users will have to register and then login to the system. During registration, users must complete all fields, including email, name, password and confirm password with correct format to be able to click the register button. In login, users must use the valid credentials. After a successful login, the user is guided to the dashboard.

In sprint one, our team mainly focused on the functionality of above-mentioned features. In sprint two, the user session management feature is added. By sprint three, a new UI designed and implemented.

The dashboard contains three parts: a navigation bar, a side bar showing run history and the body of report. In the navigation bar, the user is able to see the username and a logout button. The side bar will show all run history as buttons. When the user clicks the button, they are able to see student names and file names along with the similarity percentage and severity in the report body.

In sprint one, there are only three simple comparison strategies and the report service was not available. In sprint two, three simple comparison strategies were upgraded to complicated ones, while the run and report service was enabled. At that time, user would only see file names, similarity percentage score and separate scores of three strategies. In sprint three to the last moment, MOSS training and a feature showing difference and similarity between two files were enabled. Run and report service is complete.

As for the back end, user class was created for registering and login, including fields like user ID, email address, name and role. Report class was created for rendering a detailed report after each run, including fields like run ID, student names, file names, similarity percentage and difference between two files. We have implemented three sophisticated strategies, and all of them are based on the flattening of AST tree.

In sprint one, we focused on AST generation and a very simple report service. The report service was updated all the way to the end. We have made many design choices and we upgraded database in the backend. Thanks to a good job in AST generation, the three complicated comparison strategies worked well throughout the sprints. The final report service was elegant and nice. It worked perfectly with UI.

Integration is also an important part in our application. All front end user-input data is posted to back end using XHR HTTP request. We have made many end points for the post use, like registration, login, run plagiarism check, get reports and so on.

In conclusion, the development process is good and the final project is well-formed.

PROJECT RETROSPECTIVE

At the beginning of each phase or sprint, all four team members sat down and discussed together about how to design and implement the project. After that, we assigned tickets to each member in Jira. We really enjoyed the time working together, and the sense of creation and achievement when the project is done.

In this course we learned many useful and practical methods and processes in managing software development. We became familiar with some basic concepts in software development life cycle, saw many typical model in SDLC, learnt many design patterns in Java development, practiced important principles in user experience and user interface design and followed certain ways to do functional and structural testing.

In phase A, we focused on project design, system architecture, database schema and project setup; in phase B, we concentrated on drawing UML diagrams based on material we made in the previous phase; in sprint one, we set up Jenkins and AWS, connected GitHub repositories with Jenkins with Slack to get notifications and implemented user registration, authentication and navigation and AST generation; in sprint two, we enabled uploading submission source files, developing and implementing complicated plagiarism algorithms; in the last sprint and till the end, we generated meaningful reports off algorithm results and implemented all the functionalities.

As professor said in class, “teamwork is much harder than working alone.” Admittedly, we have faced some difficulties in teamwork. Sometimes we argued day and night about which design is better, sometimes we splitted workload in order to take care of the other course, sometimes we had to ask for explanation of a certain part of code from someone else, but all these experiences became the best memories.

But we got worried when our testers reported many defects after sprint three. Since the version of the application (the candidate branch) we provided is a quite old one, many defects existed in that branch (candidate) but was already fixed in the latest version of the application. Defects like “UI is not clean and intuitive enough”, our four testers were repeating the tickets again and again, even when we let them know that the defect has been fixed in the latest version of the application or the feature was removed. If there would be any advice from us, we will suggest that for reference, the testing team should read materials in phase A and B or discuss with dev team first then do the test.

Our team would like to show gratitude from the bottom of our hearts to Professor Mike. Thank you for showing us a great and useful class. We would like to say thank you to all teaching assistants who have graded us and answered our questions. Thank you all for making us more clear in this class and helping us solve many problems.