PLAGIARISM DETECTOR **CS5500/ Group 9**

The widespread use of computers and the emergence of the internet has made it easier to plagiarize the work of other. All the data being easily available at the click of a button has made it more convenient for the evil do gooders to excel from the established work and efforts of others. Plagiarism is commonly found in academia, where documents are typically essays or reports. But it does not stop there, now days plagiarized code is a very common occurrence and a growing nuisance, which we plan to tackle with our system.

The general approach to plagiarism detection usually involves textual similarity analysis and there are a myriad of tools that exist and do that. However, for detection of similarity in a computer language source code, the textual similarity analysis tends to fail because it is easy to hide the plagiarism in a source code by changing the variable/function name, padding with comments or whitespace, alternating the sequence of code block, moving a block from a function to another function, etc.

In our project, we are using the agile style to develop our system, thus we have divided our plan in several iterative and incremental phases which will lead up to the development of an online tool to detect similarity between Java source code.

The first step to any detection process is to parse the uploaded source code. We also plan on creating an AST of this parsed code. The library we plan on using to convert the source code into an AST is org.eclipse.jdt.core.dom.ASTParser. This library is well documented here, http://www.eclipse.org/articles/article.php?file=Article-JavaCodeManipulation_AST/index.html#sec-parsing-a-source-file. A possible second step we thought of is applying a tree edit distance algorithm to the AST generated. This algorithm calculates the minimum number of node insertions, deletions and swapping that is required to transform one tree to the other.

If time permits, we also plan on using Machine learning classification algorithms that will help find the semantic similarity between the uploaded source codes.

Lastly, we will work on an online user interface that we would build using HTML, CSS, Javascript. Our backend code, that would be doing all the processing would be written in Java. This interface would make it easier for the user of the system to upload, detect and view the results generated by our system.

USE CASES:

Use Case 1:

Use Case:	Register User / Professor to the system
Primary Actor:	User / Professor
Goal in Context:	Successfully get the new user registered to the system.
Preconditions:	1. User is not registered to the system prior to this.
Trigger:	The user decides to log in the system
Scenario:	1. User clicks the register button 2. User fills his required details like username, password and email address. 3. User clicks submit button
Exceptions:	1. Incorrect credentials: If user inputs incorrect details which are not successfully validated. 2. Prior registration : If user is already registered in the system. 3. Username already in use: If the username inputted by the user is already taken by someone else.
Priority:	Essential and must be implemented
When available:	First increment
Channel to actor:	Via the register form interface
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	Should the register form interface display additional text messages?

Use Case 2:

Use Case:	User / Professor logs in the system
Primary Actor:	User / Professor
Goal in Context:	Get the user successfully logged in the system
Preconditions:	1. The user is successfully registered to the system.
Trigger:	User decides to log in and use the system
Scenario:	1. User clicks on login button 2. User enters the username and password 3. User clicks on submit
Exceptions:	1. Username/Password is incorrect: user re-enters the details. 2. User is not registered: user registers to the system
Priority:	Essential and must be implemented
When available:	First increment
Channel to actor:	Login form user interface
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	None

Use Case 3:

Use Case:	Logout
Primary Actor:	User/Professor
Goal in Context:	To log the user out of the system.
Preconditions:	User/Professor is logged in
Trigger:	User has finished using the system.
Scenario:	1. User clicks on logout button
Exceptions:	1. Unsuccessful logout: user re-clicks the logout button.
Priority:	Essential: must be implemented
When available:	First increment
Channel to actor:	Logout button
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	None

Use Case 4:

Use Case:	Upload src folders
Primary Actor:	Professor / User
Goal in Context:	To make the system upload 2 project/src folders to be compared with each other.
Preconditions:	User is logged in.
Trigger:	User decides to detect plagiarism in two project folders
Scenario:	1. Professor/User: Observes the system 2. Clicks to open a browse prompt 3. Selects a folder 4. Clicks on the detect button.
Exceptions:	1. The uploaded folder is empty: The Professor or User re-uploads the folder. 2. The uploaded folder has no .java files: the Professor or User re-uploads the folder.
Priority:	Essential: must be implemented
When available:	Second increment
Channel to actor:	Upload form interface (buttons, upload folder)
Secondary Actor:	Admin support
Channels to Secondary Actors:	Help page
Open Issues:	1. Should we allow access without log-in? 2. Should we allow access to the web public?

Use Case 5:

Use Case:	View generated report
Primary Actor:	Professor / User
Goal in Context:	To give a summarised report of the percentage similarity and the highlighted similar terms.
Preconditions:	1. User is logged in.
Trigger:	The user decides to view the summarized report, and clicks on the view report button.
Scenario:	1. User clicks view report button. 2. User is taken to report summary screen. 3. User goes through the report to find which lines are plagiarized.
Exceptions:	1. The systems fails to parse the uploaded files.
Priority:	Essential, must be implemented
When available:	Third increment
Channel to actor:	User: View report button click
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	None

Use Case 6:

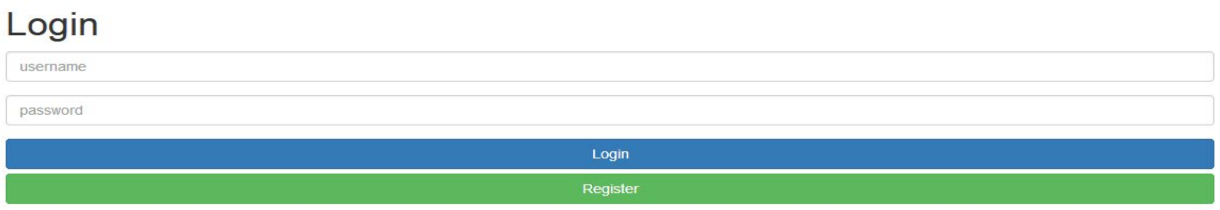
Use Case:	Download Report
Primary Actor:	Professor/User
Goal in Context:	To download the comparison report for the two uploaded source folders
Preconditions:	The plagiarism detection for the uploaded folders is complete or the user is viewing the report
Trigger:	User decides to download the comparison report
Scenario:	1. User: Observes comparison result screen or views report. 2. User: Clicks on the “Download Report” button. 3. User: Observes download prompt. 4. User: Clicks save button.
Exceptions:	1. Report not generated 2. Button not responding 3. No space on harddisk to store the report.
Priority:	Moderate Priority, to be implemented after basic functions.
When available:	Third increment
Channel to actor:	Download Report Button
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	Should the user be provided the option to choose the format in which the report has to be downloaded.

Use Case 7:

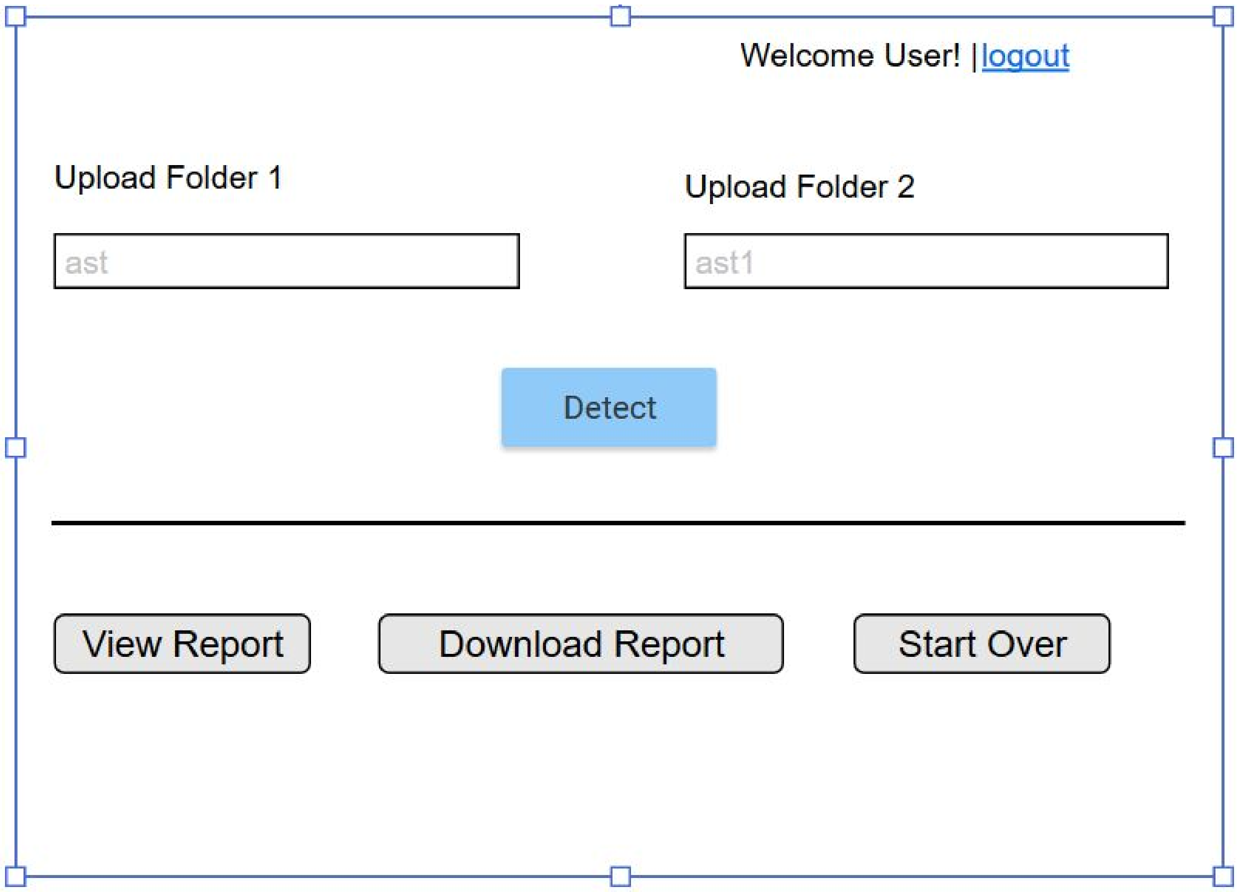
Use Case:	Start Over
Primary Actor:	Professor/User
Goal in Context:	To compare more folders for plagiarism
Preconditions:	The plagiarism detection for the uploaded folders is complete or the User is viewing the report
Trigger:	The user decides to compare another set of folders
Scenario:	User: Observes comparison result screen or views report User: Clicks on "Compare more folders" button
Exceptions:	1. Button not responding/ Broken link
Priority:	Low priority, to be implemented in the end
When available:	Third increment
Channel to actor:	Compare More Reports Button
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	None

UI MOCK UP:

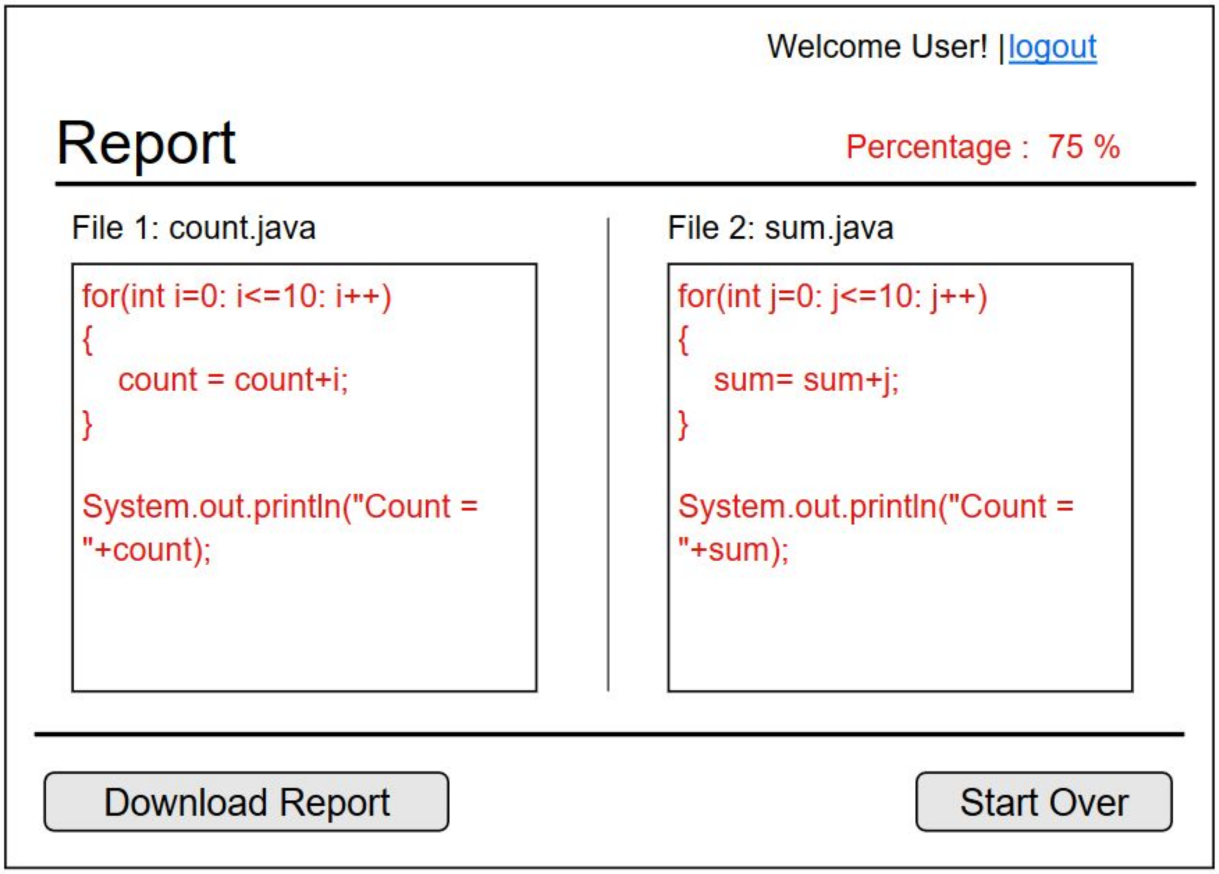
1. Login and Register UI



2. Upload folders and Detect Plagiarism



3) View Report UI



Welcome User! |logout

Report

Percentage : 75 %

File 1: count.java

```
for(int i=0: i<=10: i++)
{
   count = count+i;
}

System.out.println("Count =
"+count);
```

File 2: sum.java

```
for(int j=0: j<=10: j++)
{
   sum= sum+j;
}

System.out.println("Count =
"+sum);
```