Project Plan

The Plagiarism detector project aims to help instructors detect situations where two or more students submit similar solutions to an assignment, in which one version derives from another version through one or more behavior-preserving transformations. Our plagiarism detector algorithm detects similarities between two programs written in Java. The application is built on top of Electron based API which calls a Java based wrapper program and parses the Java source code files. The application front-end is based on Angular.JS, Bootstrap, MongoDB for managing the user collections and Node.JS for server end utilities.

Project plan defines the following:

WEEK	PLAN
Week -> 1	The basic user-interface or the skeleton part of the application will be developed using Bootstrap for designing the UI, Angular.JS for the front-end and Node.JS for handling the backend server part.
Week -> 2	The database consisting of basic schema would be developed for handling user preferences. Uploading source code functionality would also be developed.
Week -> 3	The algorithm to compare the number of lines for both the files would be engineered.
Week -> 4	The algorithm logic will be extended to include the functionality for tokenizing the comment section and comparing the similarity in the comments. The algorithm should be able to compare both the source code files as whole documents.
Week-> 5	The algorithm will now be extended to detect variable renaming transformations in both the source code files
Week-> 6	The algorithm should be finally be able to detect if the functions names have been changed or the code is moved around the source files. Finally, the UI should be engineered to show similarity between the two source code files using histograms or through tabular form.

Implementation Languages:

Front-end: Angular.JS, Bootstrap, HTML, CSS, JavaScript

Middleware: Node.JS, Electron

Back-end: JAVA, MAVEN

Database: MongoDB

Project Use Cases

Use Case:	Count of lines
Primary Actor:	System
Goal in Context:	Check if the 2 input documents have same number of lines
Preconditions:	Input should be 2 .java files
Trigger:	From terminal: Java –jar plagiarismSoftware.jar File1.java File2.java From UI: Click Run after uploading the 2 input files
Scenario:	• Actor Triggers the Use Case • The program scans through both the files, keeps 2 counters to count the number of lines in both the files • Both the counters are compared and a Boolean is returned • True if both files have same number of lines • Prints the number of lines in both the files and mentions if they have same number of lines
Exceptions:	• If one of the source code files is empty/corrupt. User should upload the correct source code files • Both the uploaded source code files must be .java files
Priority:	High
When available:	10/30/2017
Channel to actor:	UI
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	None so far

Use Case:	Checking for similarities in the comments
Primary Actor:	System
Goal in Context:	Compute the similarity percentage of the comments of the uploaded documents.
Preconditions:	Input should be 2 .java files
Trigger:	Once the number of lines are compared for both the documents
Scenario:	• The main process extracts all the single-line and multi-line comments from the source code files and dumps them to two separate text files. • The comments are broken down into tokens and the percentage similarity between the two token sets is computed and returned.
Exceptions:	• If one of the source code files is empty/corrupt. User should upload the correct source code files • Both the uploaded source code files must be .java files
Priority:	High
When available:	10/30/2017
Channel to actor:	UI; executed after Count of Lines use case is completed
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	• Using Sequence matcher library in JAVA • Time Complexity to match 1 line from File1 with n lines from File2 and repeat n times

Use Case:	Check for similarities in code
Primary Actor:	Detector
Goal in Context:	Check if the word counts for data in the whole code (excluding comments) are similar and return the positions of sameness and the percentage of similarity between both the files
Preconditions:	Input should be 2 .java files
Trigger:	Once the number of lines are compared for both the documents
Scenario:	• Count of lines returns a Boolean value • Program goes through both the files simultaneously. A map is maintained keeping track of the word counts in both the files • Returns the percentage of words with same counts
Exceptions:	• If one of the source code files is empty/corrupt. User should upload the correct source code files • Both the uploaded source code files must be .java files
Priority:	High
When available:	10/31/2017
Channel to actor:	UI; executed after Count of Lines use case is completed
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	• Using Sequence matcher library in JAVA • Time Complexity to match 1 line from File1 with n lines from File2 and repeat n times

Use Case:	Renaming variables
Primary Actor:	System
Goal in Context:	Check if the input documents have renamed the variables and used the same coding style
Preconditions:	Input should be 2 .java files
Trigger:	From UI: Click Run after uploading the 2 input files
Scenario:	• The program scans through both the documents and compares the variables instances and their runtime values.
Exceptions:	• If one of the source code files is empty/corrupt. User should upload the correct source code files. • Both the uploaded source code files must be .java files.
Priority:	Medium-High
When available:	11/05/2017
Channel to actor:	UI; executed after Count of Lines use case is completed
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	None

Use Case:	Moving code in the files
Primary Actor:	System
Goal in Context:	Compare 2 files to see if parts of code are moved around using functions
Preconditions:	Input should be 2 .java files
Trigger:	From terminal: Java –jar plagiarismSoftware.jar File1.java File2.java From UI: • Click Run after uploading the 2 input files
Scenario:	• The program scans through both the files and constructs an AST based representation of the source code. • If the code is moved around, the AST similarity should be similar.
Exceptions:	• If one of the source code files is empty/corrupt. User should upload the correct source code files. • Both the uploaded source code files must be .java files.
Priority:	Low
When available:	11/05/2017
Channel to actor:	Terminal, UI; executed after Count of Lines use case is completed
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	None

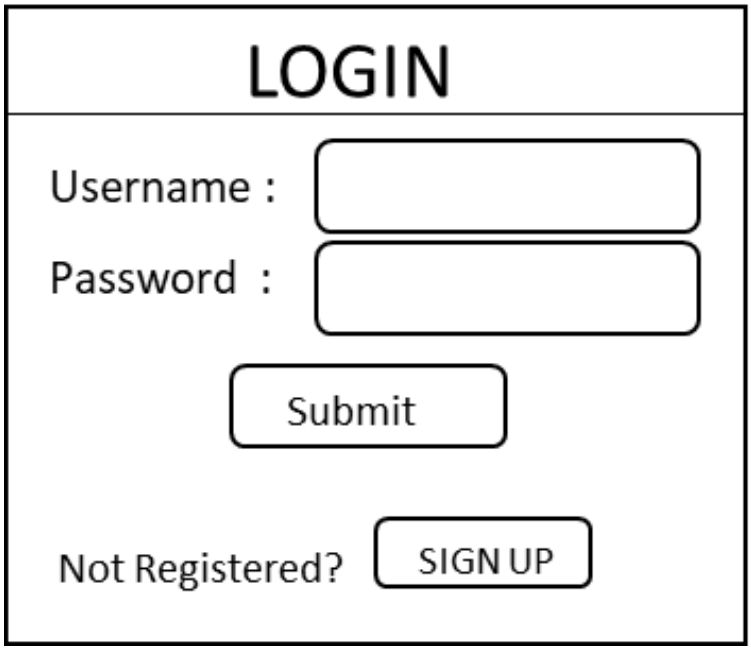
Use Case:	Comparing the object instances
Primary Actor:	System
Goal in Context:	Compare the hascode of different object instances for similarity.
Preconditions:	User should be logged in and the files should be uploaded to the application.
Trigger:	Once the number of lines are compared for both the documents
Scenario:	Computes the hashcode of all the different object instances in the program.If the hashcode of summation of all the different types of object instances is same for both the source files then the internal functionality is similar.
Exceptions:	• If one of the source code files is empty/corrupt. User should upload the correct source code files • Both the uploaded source code files must be .java files
Priority:	Medium
When available:	11/11/2017
Channel to actor:	None
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	None

Use Case:	Upload .java src files
Primary Actor:	Professor/TA
Goal in Context:	Upload 2 input files
Preconditions:	User should login, Files should be browsed in the local computer.
Trigger:	Click Upload button
Scenario:	• User is logged in • User browses his file system and choses 2 input files • User clicks upload • The two documents get populated on two 2 displays for overview
Exceptions:	• If one of the source code files is empty/corrupt. User should upload the correct source code files. • Both the uploaded source code files must be .java files.
Priority:	High
When available:	11/10/2017
Channel to actor:	GUI
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	None

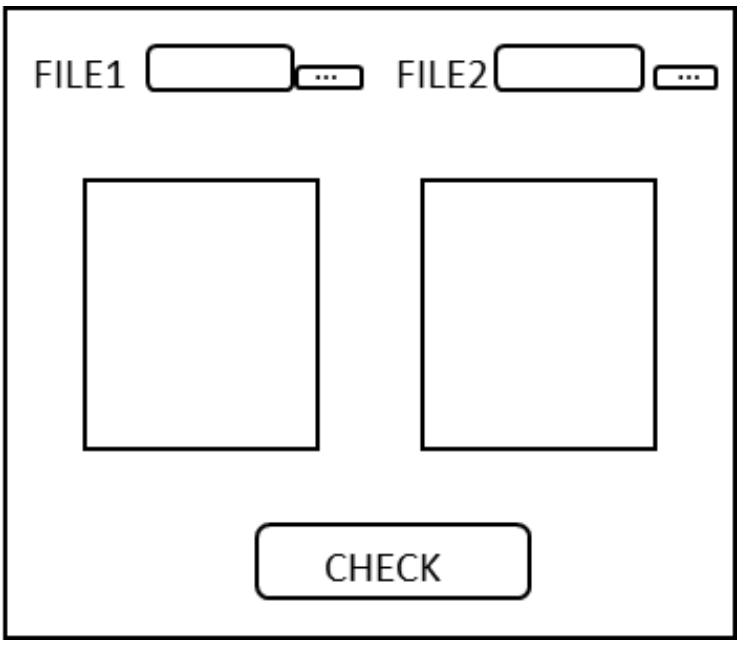
Use Case:	Login
Primary Actor:	Any User
Goal in Context:	Give access to the software to only the person who is authorized to use it
Preconditions:	User should login, Files should be uploaded in the local computer.
Trigger:	Open the App
Scenario:	• User opens the app • User enters his credentials • User clicks login button
Exceptions:	No exceptions
Priority:	High
When available:	TBA
Channel to actor:	GUI, Electron App
Secondary Actor:	None
Channels to Secondary Actors:	None
Open Issues:	None

MOCK-UP'S

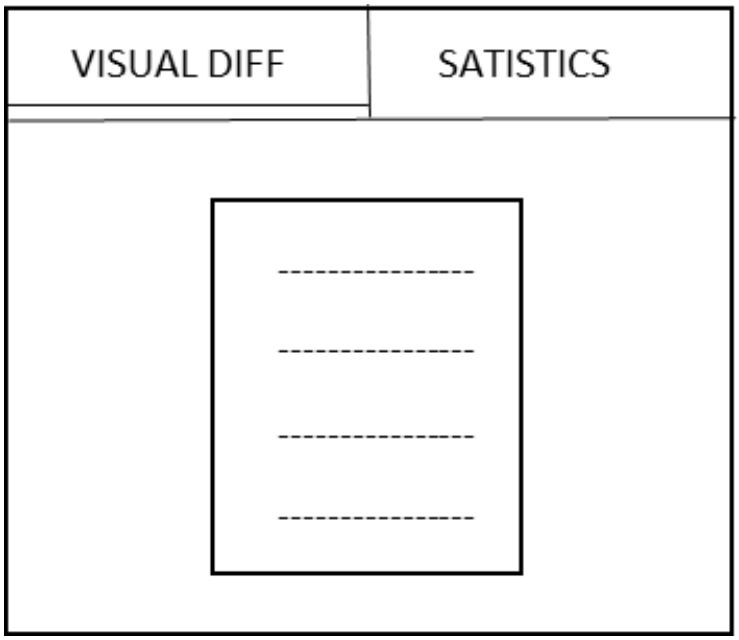
- User logs in to the system.



- User uploads two java files that he wants to check plagiarism against. User can browse the files, he wants to check plagiarism for. Once file is selected, the content of the files gets rendered on to the screen under the respective file name. Further, clicking on Check button, takes user to the evaluation screen.



- Here, the user would be visually able to figure out the textual similiarities between the two input files. This gives a fair amount of idea in predicting if the documents are plagiarized.



- The last screen would deal with the statistics, on how much is the similiarity between two given documents in different test scenarios listed in the use cases. Pictorial representation would be representing the numbers in form of High charts for user to easily visualize the statistics.