Phase A Report

Aim: Team One, over the course of three phases, will develop an interactive interface for professors, teaching assistants, and administrators to oversee the detection of plagiarism. In this summary, the reader will find the expected input for the project, an explanation of the project's interface, expected users of this project, and the underlying plagiarism detection methods that will be employed.

Constraints: While this project could be expanded in the future, the scope of this iteration is constrained as described below.

- This project is intended to be used to detect plagiarism in programs written in Java.
- Only single file programs will be considered for plagiarism at this time.
- Comments will not be taken into consideration when detecting plagiarism.

Logistics: Our plagiarism detection software will be written with a combination of Java, HTML, and JavaScript. The portion of the project that will detect plagiarism and manage underlying data storage and retrieval will be written in Java. The project's data will be stored in a SQLite database. Users of this project will interact with this project by means of a web interface that will require a secure logon.

Implementation Plan: This project has two intended types of users:

- **Instructors:**
  Instructors can create, modify, delete courses and assignments related to the course. The instructors of a course consist of both the Professors and the Teaching Assistants (TAs) of a course
  Instructors can upload the assignment submissions to the courses that they are associated. After uploading submissions, they can run plagiarism detection on the submissions. They can review the results of their courses' plagiarism detection. These actions can be performed by the Instructors as well.
- **Administrators (admin):**
  Admin users handle the creation and deletion of users of this application to restrict the access of the application

Our approach for detecting plagiarism between any two files borrows heavily from the Rojas Method[1] and consists of the three phases described below.

- Preprocessing: Removal of unnecessary components of a program, such as comments.
- Recording: Ordering functions based on character count.
- Running a comparison against the restructured submissions: Results in a score based on percent match between two given submissions. Submission pairs with a score above a given threshold will be flagged as potential instances of plagiarism.

JavaParser[2] toolset is used for processing and modification of submissions.

Schedule: Our project is scheduled to be fully implemented by the end of the Northeastern University's Fall 2017 semester. The project will be implemented in three phases: Phase A: design, Phase B: documentation, and Phase C: implementation. All deliverables will be stored in our GitHub repository[3].

1: Rojas, J. L. (2012). Detection of plagiarism in Java programming assignments in introductory computer science courses at Ifi (Unpublished master's thesis). University of Oslo. Retrieved from https://www.duo.uio.no/bitstream/handle/10852/34143/josek-master2012.pdf?sequence=1
2: http://javaparser.org
3: https://github.ccs.neu.edu/cs5500/team-1

Use Cases

Use Case:	Instructor/Admin Login
Primary Actors:	Instructor and Admin
Goal in Context:	The aim of the Instructor and Admin is to gain access to the plagiarism detector.
Preconditions:	1. The Instructor and Admin should have an account. 2. The Instructor and Admin should have access to assignments.
Trigger:	Opening the web address in a browser
Scenario:	1. The Instructor/Admin opens the web application. 2. When prompted for username and password, the Instructor/Admin provides a valid username and password. 3. Once login is successful, the Instructor/Admin is directed to the homepage. The homepage will have a facility that enables the Instructor to upload the assignments. For the Admin, the homepage would be a dashboard where user accounts can be created, modified, or deleted.
Exceptions:	The username and password does not match. The Instructor/Admin can try entering the login details again or cancel the operation.
Priority:	High, must be implemented.
When available:	Phase C
Channel to actor:	The Instructor/Admin interacts with the system through a web interface.
Secondary Actor:	N/A
Channels to Secondary Actors:	N/A
Open Issues:	1. Should we have a separate Admin user who will control the access to the detector? 2. Can we display any non-sensitive information related to the plagiarism detector/subject on the landing page such that login is not required? 3. Will there be any provisions to recreate password/username if the user has forgotten them? 4. Should any other fields be displayed along with the username and password fields? For example, a subject field.

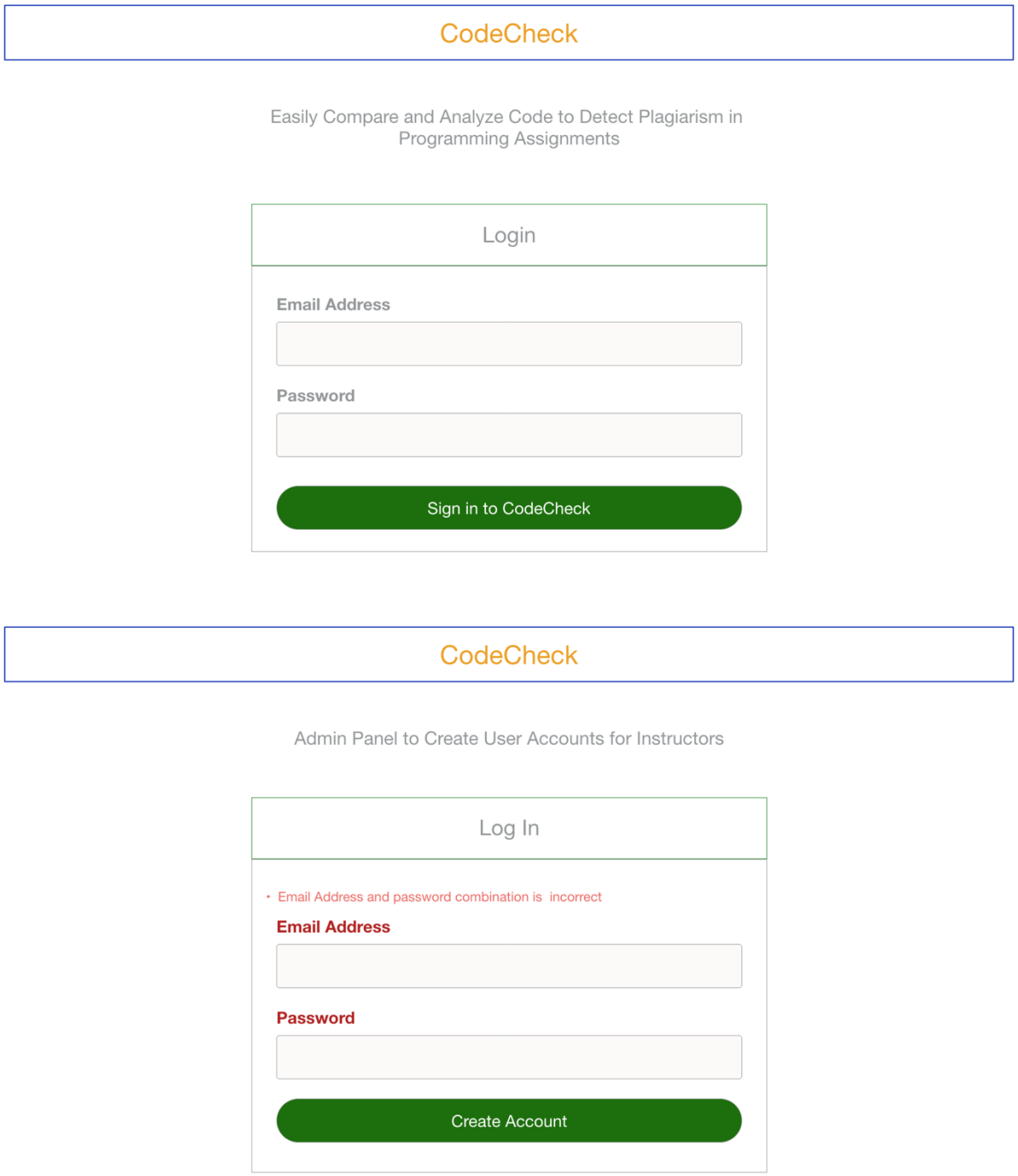
Use Case:	Admin Creates Account for Instructors
Primary Actor:	Admin
Goal in Context:	The aim of the Admin is to grant the Instructors access to the plagiarism detector.
Preconditions:	The Admin should be logged in.
Trigger:	The Admin clicks on Create Account tab.
Scenario:	1. The Admin clicks on Create Account tab. 2. The Admin is presented with a form to fill up. The form requests additional information regarding the Instructor. 3. The Admin clicks on Create Account button.
Exceptions:	1. The Instructor already has an account. The Admin cancels the create account operation or might replace the Instructor's current account details with the new account details. 2. Some fields are left blank. The Admin is prompted to refill the blank fields.
Priority:	Medium, not urgent
When available:	Phase C
Channel to actor:	The Admin and the system communicate using web interface
Secondary Actor:	Instructor
Channels to Secondary Actors:	The Instructor interacts with the system through the web application.
Open Issues:	1. What information do we need from the Instructor in the Create Account form?

Use Case:	Examine Saved Plagiarism Detection Report
Primary Actor:	Instructor
Goal in Context:	The aim of the Instructor is to review reports previously generated by the plagiarism detector.
Preconditions:	1. The Instructor should have an account. 2. The instructor should be logged in. 3. The Instructor should have access to assignments. 4. The Instructor should have already generated a plagiarism report for at least one assignment
Trigger:	The Instructor clicks on the Review button on homepage.
Scenario:	1. The Instructor clicks on the Review button. 2. The Instructor clicks on one of his courses. 3. The Instructor clicks on one the selected course's assignments. 4. The Instructor clicks on one of the assignment's previously generated reports. 5. The Instructor is presented with a list of instances of reported plagiarism. 6. The Instructor can click on an instance of plagiarism to review details.
Exceptions:	N/A
Priority:	Medium, not urgent
When Available:	Phase C
Channel to Actor:	The Instructor interacts with the system through a web interface.
Secondary Actor:	N/A
Channels to Secondary Actors:	N/A
Open Issues:	Should only courses and assignments with previously generated reports be navigable in the review section? If not, an option to generate a report for given course/assignment should be presented.

Use Case:	Upload assignment submissions
Primary Actor:	Instructor
Goal in Context:	The aim of the Instructor is to upload the submissions from students for an assignment to the plagiarism detector
Preconditions:	The Instructor should be logged in. The Instructor should be navigated to the 'Upload Submissions' page The Instructor should have the submission files on his/her computer
Trigger:	The Instructor clicks on the 'Upload Files' button and browses and selects the submission files from their local computer
Scenario:	1. The Instructor clicks on the 'Upload Files' button. 2. The Instructor is presented a file system popup to select files from his local file system 3. The Instructor browses through his local file system and selects the submission files for the assignment 4. The Instructor clicks on the 'Open' button in the file system popup
Exceptions:	1. No files are selected The Instructor is prompted to select files again.
Priority:	High, must be implemented
When available:	Phase C
Channel to actor:	The Instructor and the system communicate using web interface
Secondary Actor:	N/A
Channels to Secondary Actors:	N/A
Open Issues:	1. What types of files should acceptable? Eg. .txt, .pdf, .java

Use Case:	Check submissions for plagiarism
Primary Actor:	Instructor
Goal in Context:	The aim of the Instructor is to check all assignment submissions against each other to detect plagiarism and save the automatically generated plagiarism report
Preconditions:	The Instructor should be logged in. The Instructor should be navigated to the 'Upload Submissions' page The Instructor should have browsed and selected the submission files from his local system
Trigger:	The Instructor clicks on the 'Analyze Results' button
Scenario:	1. The Instructor clicks on the 'Upload Files' button
Exceptions:	1. Any file is larger than 25MB The file size is limited by the web server to 25MB, the instructor is given the option to skip over this file and try again 2. Any submission file is corrupted The instructor is given the option to skip over this file and try again
Priority:	High, must be implemented
When available:	Phase C
Channel to actor:	The Instructor and the system communicate using web interface
Secondary Actor:	N/A
Channels to Secondary Actors:	N/A
Open Issues:	1. What should be the plagiarism detection algorithm used in the back-end? 2. What should happen if an assignment submission file is corrupted? 3. Should the instructor be redirected to another page while the submissions are being processed? 4. Should the transformed files be displayed to the instructor in the plagiarism report?

Wireframes


CodeCheck
Easily Compare and Analyze Code to Detect Plagiarism in Programming Assignments
Login
Email Address
Password
Sign in to CodeCheck
CodeCheck
Admin Panel to Create User Accounts for Instructors
Log In
Email Address and password combination is incorrect
Email Address
Password
Create Account

CodeCheck

Admin Panel to Create User Accounts for Instructors

Create User

Email Address

Password (Use at least one letter, one numeral, eight characters)

Create Account

CodeCheck

Admin Panel to Create User Accounts for Instructors

Create User

- Email Address can't be blank
- Password can't be blank and is too short (minimum is 7 characters)

Email Address

Password (Use at least one letter, one numeral, eight characters) *

Create Account

CodeCheck Log Out

Browse and upload programming assignment submissions using the ‘Upload Files’ button

After selecting files, click on the ‘Analyze Results’ button to compare these submissions

+ Upload Files

Enter a programming assignment name

Enter a programming assignment name (Optional)

ANALYZE RESULTS

ANALYSIS HISTORY

CodeCheck Log Out

ANALYSIS HISTORY

October 7, 2017 at 6:00 PM	Programming Assignment 1	Topic A	Processing
October 7, 2017 at 6:00 PM	Programming Assignment 2	Topic X	Open
October 7, 2017 at 6:00 PM	Programming Assignment 3	Topic D	Open
October 7, 2017 at 6:00 PM	Programming Assignment 4	Topic E	Open
October 7, 2017 at 6:00 PM	Programming Assignment 5	Topic F	Open

< CodeCheck Log Out

RESULTS FOR PROGRAMMING ASSIGNMENT *x*

Rank 1 (side-by-side)	Student Name 1 submitted at dd/mm/yy mm:hh Student Name 2 submitted at dd/mm/yy mm:hh	Student Name 1, Student Name 2 Similarity Points: *x points* Similarity Percentage: *y%*
Rank 2 (side-by-side)	Student Name 1 submitted at dd/mm/yy mm:hh Student Name 2 submitted at dd/mm/yy mm:hh	Student Name 1, Student Name 2 Similarity Points: *x points* Similarity Percentage: *y%*
Rank 3 (side-by-side)	Student Name 1 submitted at dd/mm/yy mm:hh Student Name 2 submitted at dd/mm/yy mm:hh	Student Name 1, Student Name 2 Similarity Points: *x points* Similarity Percentage: *y%*

< CodeCheck Log Out

Side-by-side File Diff between two student submissions

Similarity Metric:
- Points: *x points*
- Percentage: *y%*

● Original Files ○ Transformed Files

Student Name, fileName1.java	*Student Name, fileName2.java*
---	----