Group 2 Plagiarism Detector
Phase C: Final Report

Authors: Shagun Bhardwaj
Kenji Fujita
Mason Leon
Prakash Tarun Kumar

Course: CS 5500 - Section 1
Foundations of Software Engineering
Dr. Frank Tip
Fall 2019

Khoury College of Computer Sciences
Northeastern University, Boston, MA

Background

Plagiarism is a violation of academic integrity and often limits both the personal and collective success of students in the academic environment. In order to reduce the impact of plagiarism within the computer science program at Northeastern University, the department's faculty has reached out for assistance in acquiring a software tools that can detect plagiarism in student submitted programming assignments. Detecting plagiarism in code can be different than detecting plagiarism in a traditional written work.

Language Choices

The plagiarism detector program will be written in Java and will support code written in Python and C++. These languages are chosen due to the fact that they are two of the most popular programming languages used in the world. They are core foundational programming languages that are taught to Northeastern students and are used in multiple classes. Due to this, many of the programming assignments turned into Northeastern will be written in Python or C++. This also means that there is a higher likelihood for potential plagiarism and academic misconduct by the students. Originally, we planned on choosing Python and C as the two languages to support. We made this initial choice because all of the group members had experience coding in Python and C, and we believed that it would be easier to implement a plagiarism detector on programming languages that we were familiar with. However, when we started developing the algorithm, we changed one language from C to C++. This change was made because C++ because C++ is a more object oriented programming language than C (it has classes), and the algorithm was similar between Python and C++. Therefore, we could use more abstraction in our algorithm to detect for plagiarism in code written in either programming language.

Algorithm

The program supports plagiarism detection between single Python or C++ files as well as multi-file packages. Our algorithm accounts for attempts to hide plagiarised work, such as code moved around within a file or code moved between files.

When we take in the files through our UI, we parse the files using ANTLR. The ANTLR classes create an abstract syntax tree for each file in both packages that we save. Our algorithm compares any two of the abstract syntax trees. The inspiration for our algorithm is taken from the Winnowing algorithm. The winnowing algorithm is an algorithm that functions as document fingerprinting. Document fingerprinting is a way to assign a document, or its contents, a unique value (“fingerprint”) that represents what is actually written. Document fingerprinting is done via hashing and can be used to compare two pieces of text to find similarities between them. This is useful for plagiarism detection as document fingerprinting can find similarities between two files of code, even if the code has been shuffled around in an attempt to hide this. Winnowing algorithm is typically used with traditional, written text (like articles and books) to detect similarities and potential plagiarism. It involves splitting the text into k-grams; a k-gram is a substring of length k. Each k-gram is then hashed, and a specific number of of k-grams is used to create the documents’ fingerprints.

Since our goal is to detect plagiarism between code files instead of traditional text, we altered this algorithm to better serve our purpose. Instead of splitting the code into k-grams, we split the code by each line. Splitting by a set string length would separate code in the same line, which would remove the logic of the code. This would make it very difficult to hash the code properly, as the code would be split at arbitrary places. Instead of having k-grams in our algorithm, we had line-grams, with each line being assigned to a gram. We then hash the line by retrieving it as a string and iterating through the line by each character. For each character, we add the ASCII value of that character to a running total for the line. This sum is stored in a HashMap, which maps the line number (int) to the hashed value of the line (also an int). We create this HashMap using a recursive function which iterates through the entire AST of a file. The recursive function starts at the head of the AST, and recursively iterates through the tree buy calling the same function on each of the node’s children. At each node, the algorithm calls the function to hash node’s text. For Python files, “Atom” is the only node that we hash because they contain the key information in each line of code. For C++ files, we hash every node because they are all pertinent for the algorithm. After all of the HashMaps are created, the algorithm compares every file from package 1 to every file from package 2. In each comparison, there is a nested loop to compare every line from the first file to every line from the second file. If the difference between the hash values for any two lines is less than 10, that line is flagged as a plagiarized line and added the final result. This final result is then outputted to the user.

This plagiarism detection algorithm has some advantages and disadvantages. One disadvantage is that the algorithm has a longer runtime, and is computationally intensive. This is because every file from one package is compared to every file from the other package, and every file has to be iterated to create the HashMap. The runtime for this algorithm is approximately O(n^3). An advantage is that the algorithm takes into account moving code between functions, moving code between files, and changing some basic syntax (ex. Changing type of loop: for-while). Since the algorithm compares every possible file combination, it can detect plagiarism from shifting around code.

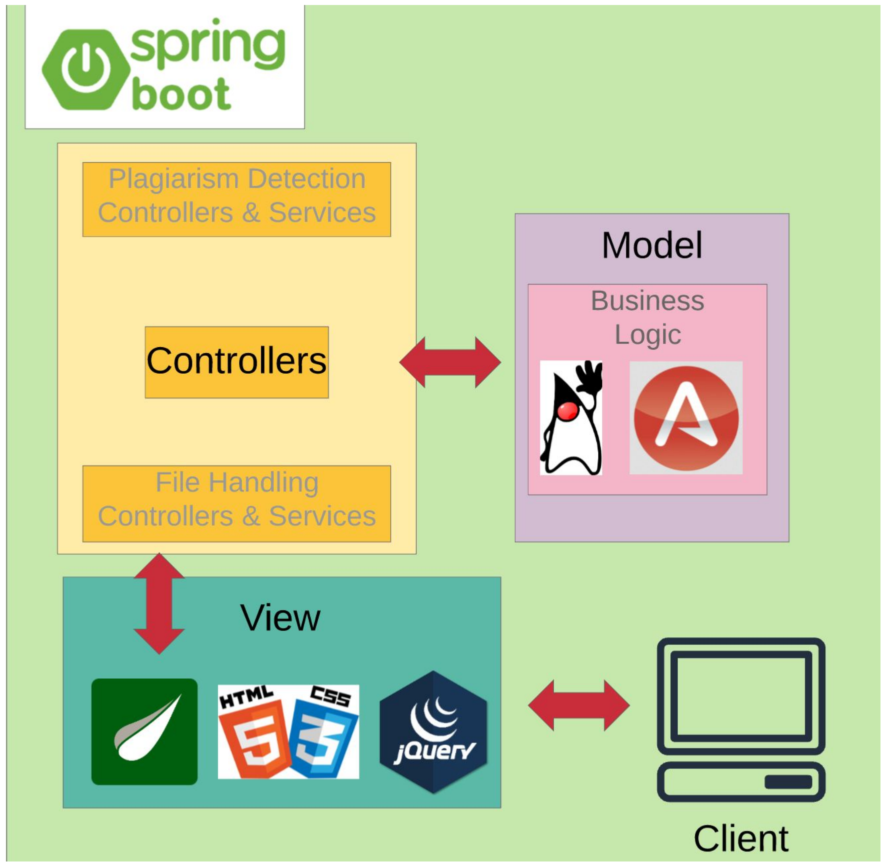
Functionality

The plagiarism detector works by finding common lines between two files. The program can detect common ways to hide plagiarism, such as: moving around code within a file, moving around code between files, renaming files, renaming functions whitespaces, and comments. The program does not account for renaming of variables right now. Since the algorithm hashes the contents of the nodes, it is not able to detect renaming variables.

Design

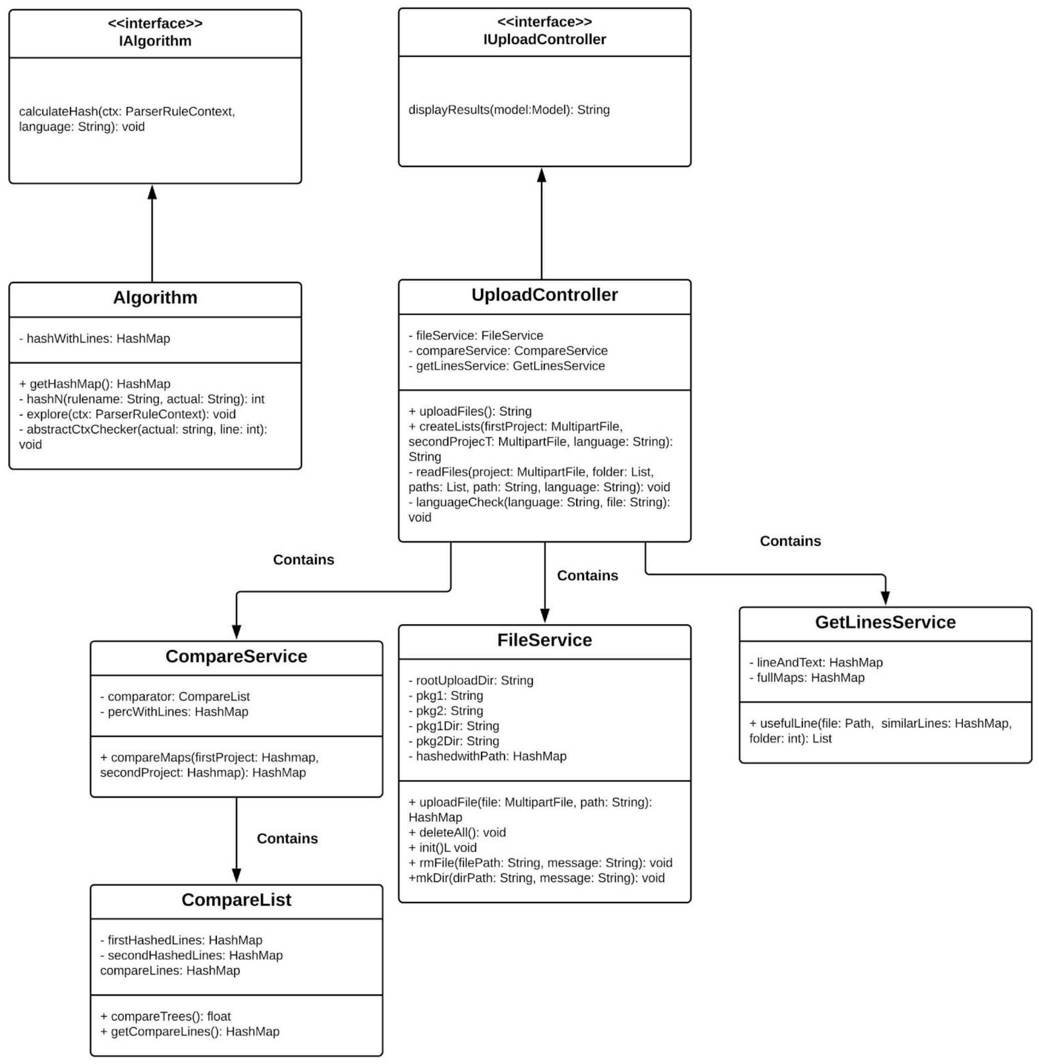
The program was written using the Spring MVC (Model-View-Controller) design pattern. We chose this design pattern because it keeps the three main parts (model, view, and controller) of the program modular. Any of the three parts can be swapped out very easily, allowing for easy and flexible changes in the future. Using Spring MVC, we mapped the components of the view with the controller. The controller then initializes service objects that take the data from the controller, pass the data to the model, receive results from the model, and then pass the results back to the controller. The controller then deals with this data and organizes it in a way that is usable by the view.

Architecture


spring boot
Plagiarism Detection Controllers & Services
Controllers
File Handling Controllers & Services
Model
Business Logic
View
HTML
CSS
jQuery
Client

The framework used for this project was Spring Boot. We initially chose Spring MVC as the framework, but changed to Spring Boot. The view was created using HTML and CSS, as well as JQuery. The model consists of the algorithm and how to compare files. The controller passes information from the model to the view and vice versa. The services help perform actuals such as opening files and using the data that the model holds.

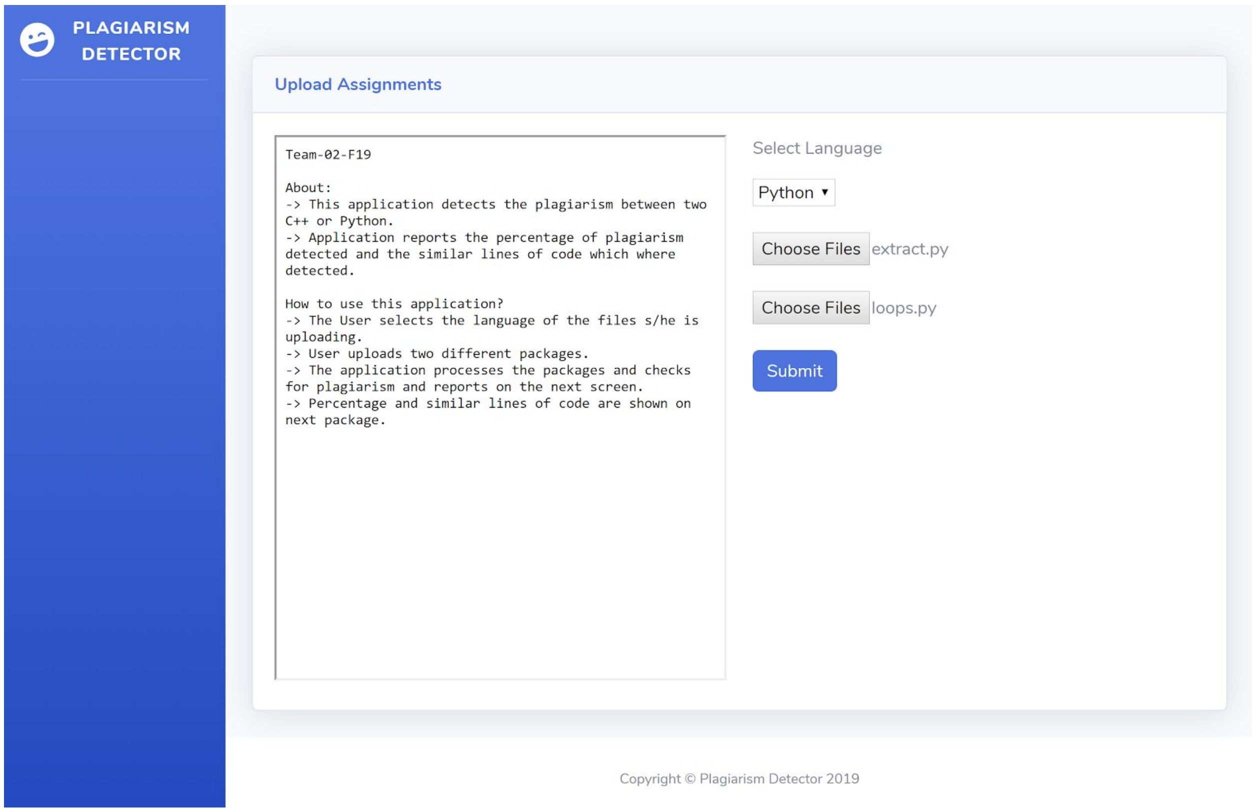
UML Class Diagram


<<interface>>
IAlgorithm
calculateHash(ctx: ParserRuleContext, language: String): void
<<interface>>
IUploadController
displayResults(model:Model): String
Algorithm
- hashWithLines: HashMap
+ getHashMap(): HashMap
- hashN(rulename: String, actual: String): int
- explore(ctx: ParserRuleContext): void
- abstractCtxChecker(actual: string, line: int): void
UploadController
- fileService: FileService
- compareService: CompareService
- getLinesService: GetLinesService
+ uploadFiles(): String
+ createLists(firstProject: MultipartFile, secondProjecT: MultipartFile, language: String): String
- readFiles(project: MultipartFile, folder: List, paths: List, path: String, language: String): void
- languageCheck(language: String, file: String): void
Contains
Contains
Contains
CompareService
- comparator: CompareList
- percWithLines: HashMap
+ compareMaps(firstProject: Hashmap, secondProject: Hashmap): HashMap
FileService
- rootUploadDir: String
- pkg1: String
- pkg2: String
- pkg1Dir: String
- pkg2Dir: String
- hashedwithPath: HashMap
+ uploadFile(file: MultipartFile, path: String): HashMap
+ deleteAll(): void
+ init()L void
+ rmFile(filePath: String, message: String): void
+mkDir(dirPath: String, message: String): void
GetLinesService
- lineAndText: HashMap
- fullMaps: HashMap
+ usefulLine(file: Path, similarLines: HashMap, folder: int): List
Contains
CompareList
- firstHashedLines: HashMap
- secondHashedLines: HashMap
compareLines: HashMap
+ compareTrees(): float
+ getCompareLines(): HashMap

We utilized a MVC design pattern for our project. The view was coded in HTML. The Controller contains three services that perform specific actions for the program. The algorithm is part of the model, as well as the class that compares the files.

The following images shows how our UI works for various file uploads, and how the results are displayed:

Single File Upload

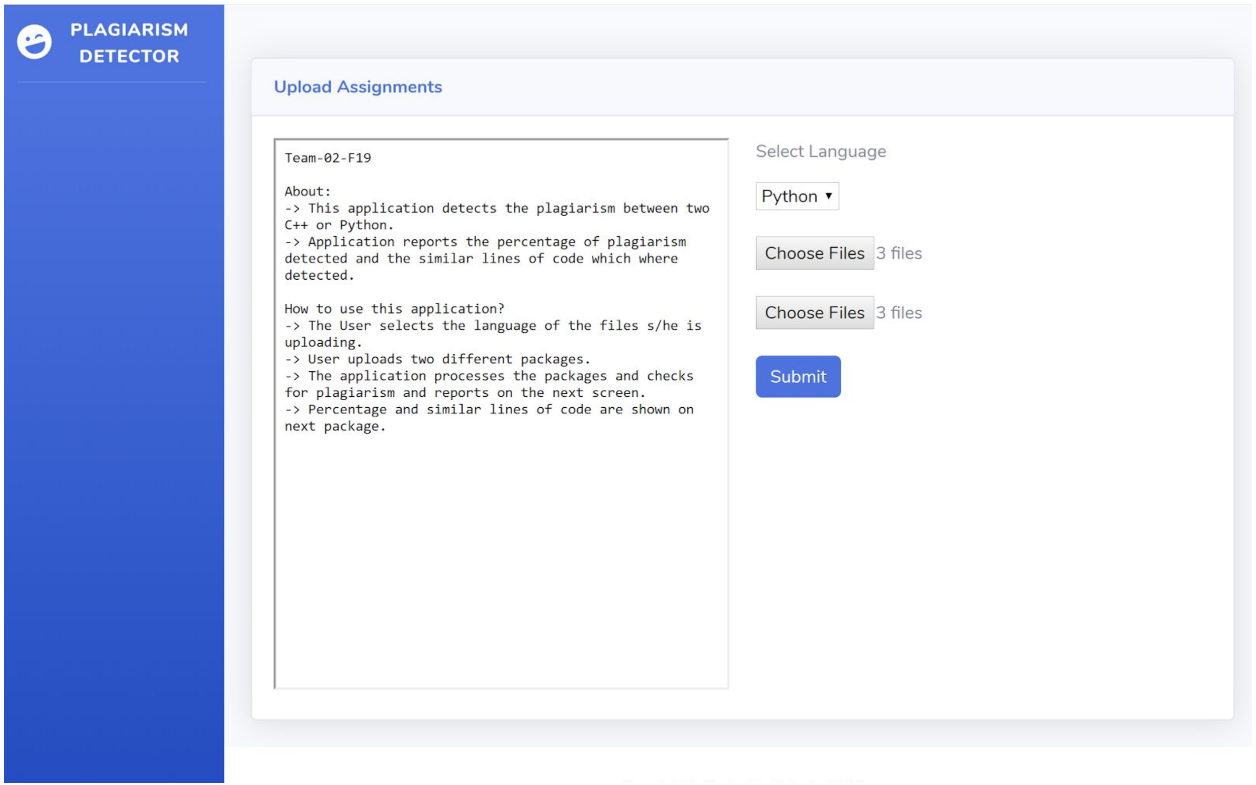


```
Team-02-F19

About:
-> This application detects the plagiarism between two
C++ or Python.
-> Application reports the percentage of plagiarism
detected and the similar lines of code which where
detected.

How to use this application?
-> The User selects the language of the files s/he is
uploading.
-> User uploads two different packages.
-> The application processes the packages and checks
for plagiarism and reports on the next screen.
-> Percentage and similar lines of code are shown on
next package.
```

Multiple Files Upload

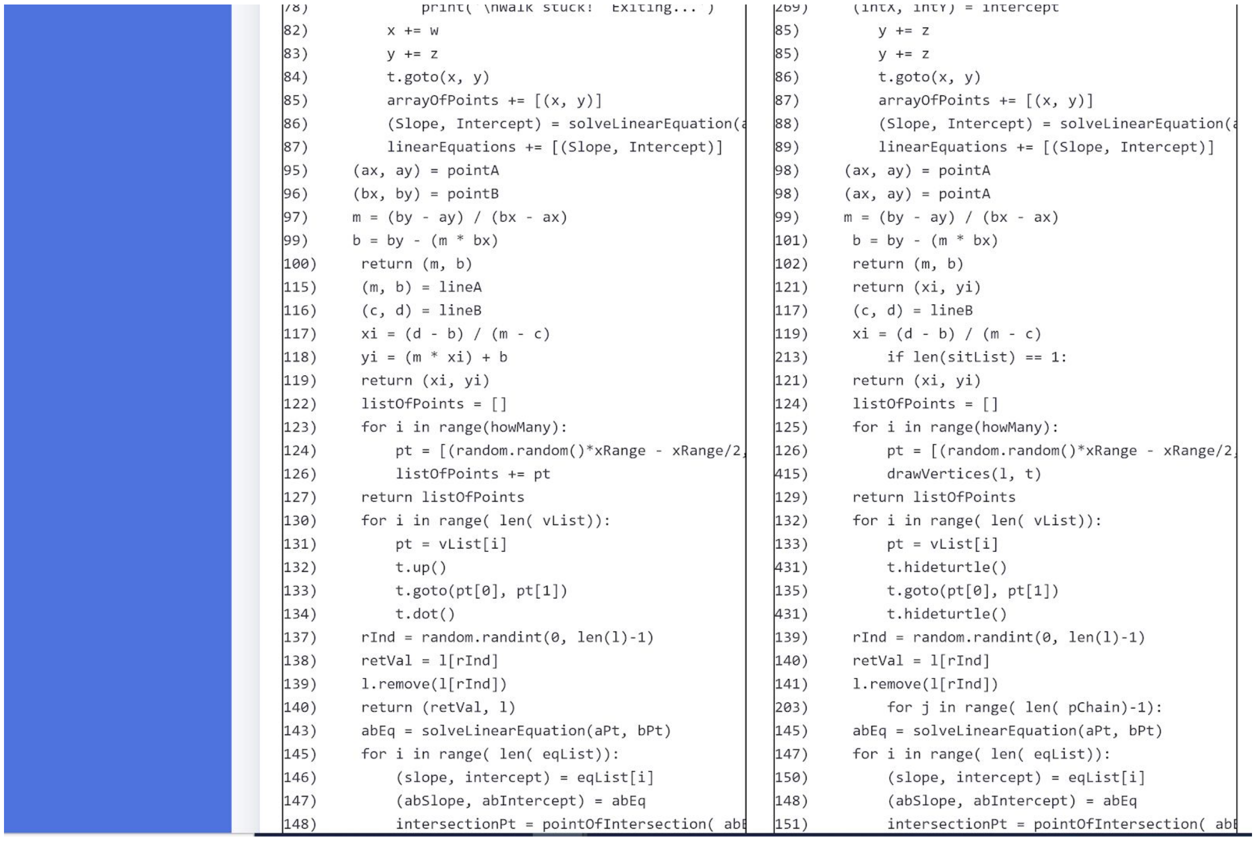


```
Team-02-F19

About:
-> This application detects the plagiarism between two
C++ or Python.
-> Application reports the percentage of plagiarism
detected and the similar lines of code which where
detected.

How to use this application?
-> The User selects the language of the files s/he is
uploading.
-> User uploads two different packages.
-> The application processes the packages and checks
for plagiarism and reports on the next screen.
-> Percentage and similar lines of code are shown on
next package.
```

Results


```
78)           print('\nWalk stuck!  Exiting...')      269)    (intX, intY) = intercept
82)         x += w                                   85)         y += z
83)         y += z                                   85)         y += z
84)         t.goto(x, y)                             86)         t.goto(x, y)
85)         arrayOfPoints += [(x, y)]                87)         arrayOfPoints += [(x, y)]
86)         (Slope, Intercept) = solveLinearEquation(a  88)         (Slope, Intercept) = solveLinearEquation(a
87)         linearEquations += [(Slope, Intercept)]  89)         linearEquations += [(Slope, Intercept)]
95)     (ax, ay) = pointA                            98)     (ax, ay) = pointA
96)     (bx, by) = pointB                            98)     (ax, ay) = pointA
97)     m = (by - ay) / (bx - ax)                    99)     m = (by - ay) / (bx - ax)
99)     b = by - (m * bx)                            101)    b = by - (m * bx)
100)    return (m, b)                                102)    return (m, b)
115)    (m, b) = lineA                               121)    return (xi, yi)
116)    (c, d) = lineB                               117)    (c, d) = lineB
117)    xi = (d - b) / (m - c)                       119)    xi = (d - b) / (m - c)
118)    yi = (m * xi) + b                            213)        if len(sitList) == 1:
119)    return (xi, yi)                              121)    return (xi, yi)
122)    listOfPoints = []                            124)    listOfPoints = []
123)    for i in range(howMany):                     125)    for i in range(howMany):
124)        pt = [(random.random()*xRange - xRange/2,  126)        pt = [(random.random()*xRange - xRange/2,
126)        listOfPoints += pt                       415)        drawVertices(l, t)
127)    return listOfPoints                          129)    return listOfPoints
130)    for i in range( len( vList)):                132)    for i in range( len( vList)):
131)        pt = vList[i]                            133)        pt = vList[i]
132)        t.up()                                   431)        t.hideturtle()
133)        t.goto(pt[0], pt[1])                     135)        t.goto(pt[0], pt[1])
134)        t.dot()                                  431)        t.hideturtle()
137)    rInd = random.randint(0, len(l)-1)           139)    rInd = random.randint(0, len(l)-1)
138)    retVal = l[rInd]                             140)    retVal = l[rInd]
139)    l.remove(l[rInd])                            141)    l.remove(l[rInd])
140)    return (retVal, l)                           203)        for j in range( len( pChain)-1):
143)    abEq = solveLinearEquation(aPt, bPt)         145)    abEq = solveLinearEquation(aPt, bPt)
145)    for i in range( len( eqList)):               147)    for i in range( len( eqList)):
146)        (slope, intercept) = eqList[i]           150)        (slope, intercept) = eqList[i]
147)        (abSlope, abIntercept) = abEq            148)        (abSlope, abIntercept) = abEq
148)        intersectionPt = pointOfIntersection( ab  151)        intersectionPt = pointOfIntersection( ab
```

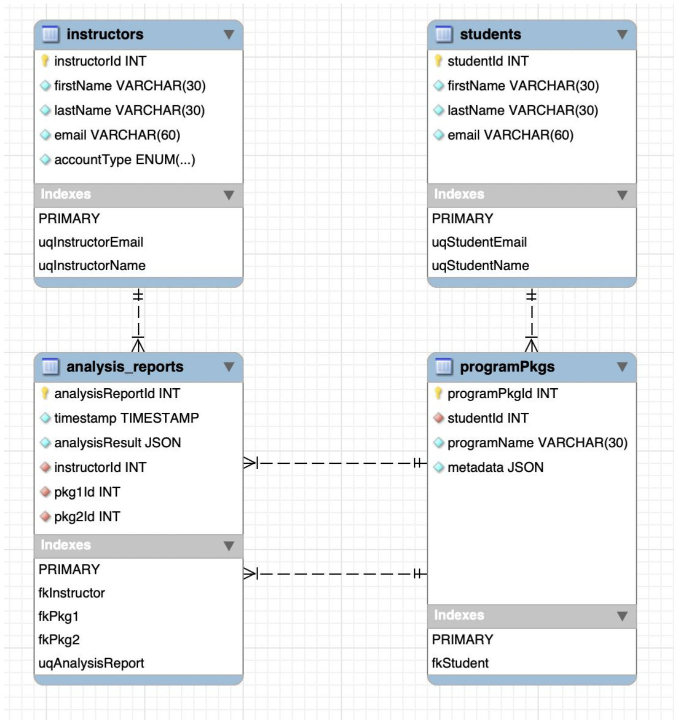
Positives and Challenges

While we had challenges with continuous integration like many of the other groups, most of our challenges came from the way we separated work and understood the architecture. We went with Spring Boot and tried to incorporate a database to allow persistence, however this became extremely difficult to test and map. We realized this very late and were scrambling to sprint at the end with a basic Spring controller. Once this was in place we had to do a lot of debugging and testing to ensure the mapping to the view worked.

One thing that we could have improved is our communication. There were times when we weren't too sure how our partners were doing because we were primarily focused on the individual responsibilities that we gave each other. It was hard to set up tests ahead of time because we weren't quite sure how our architecture would work and communicate until the very end.

Another challenge that we were having was connecting the database to our program. We had trouble understanding learning the mapping for Hibernate and JPA for our program. The learning curve was very steep as we had never worked with REST controllers or the Spring web framework beforehand. We decided not to have the database functionality with our program, and would implement it in a future version of our program.

The following shows the UML diagram for our proposed database:



What worked well for our group was pair programming. When we would get stuck on a part of the program, we would pair program and talk through our thought process. This worked well because each person had new insight into the problem. Pair programming allowed us to solve problems efficiently and allowed everyone to understand how the program works.

Our group also worked well with Git. We would always work on different branches, and use pull requests to make sure someone would review the code before merging into master. This allowed us to work on different parts of the project simultaneously, and integrate all the pieces together.

Conclusion

The plagiarism detector was a challenging project that required us to learn how to effectively communicate and collaborate on different parts. Our plagiarism detector accounts for changes such as moving around code within a file, moving around code between files, renaming of files or functions, whitespaces, and comments. However, our detector does not work as well with renaming variables. While we worked well at certain aspects such as pair programming, we could improve in the future on other aspects, such as communication. This project allowed us to learn how to plan, develop, and test a large program.