Project Report

SYSTEM FUNCTIONALITY

Our application is based on a client server architecture. I have delivered the following functionalities in our system:

1. ***Uploading**, **Viewing** and **Deleting** multi file java projects*

- I can upload Java project folder on cloud based AWS S3 for storage and their respective metadata in AWS RDS as Database.
- All the uploaded Project's meta information can be fetched from Database and displayed to end user.
- A user can delete any project from UI from the projects information displayed to him. This will delete the Data from AWS S3 and metadata from AWS RDS.

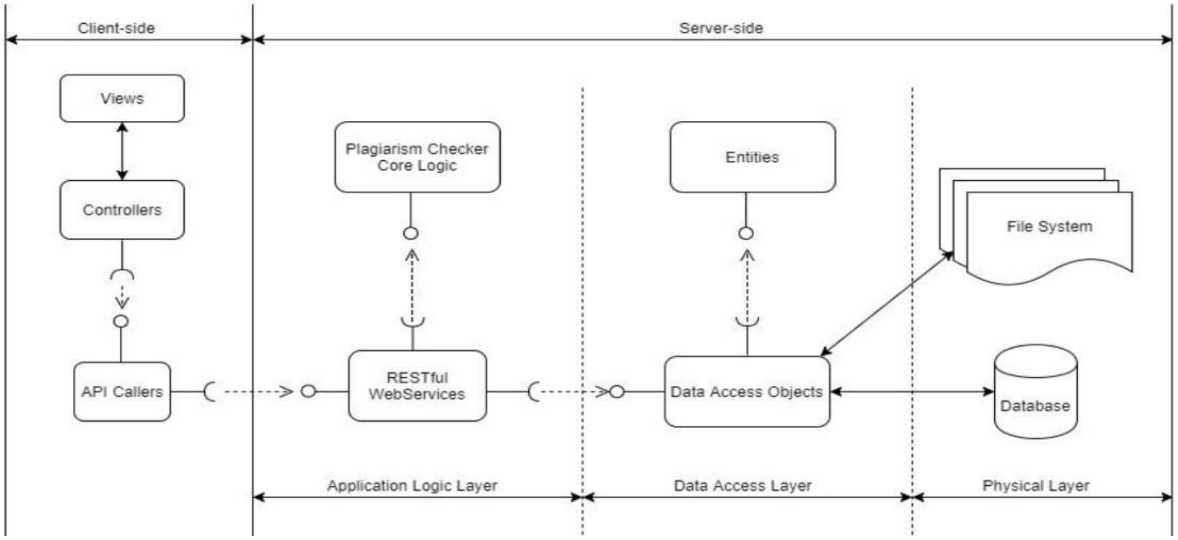
2. ***Plagiarism Check** between 2 different projects in the system*

- I can run the plagiarism engine (described in detail later) can be run on 2 different projects, and this returns a report summary describing the results of the comparison

3. ***Saving**, **Viewing** and **Deleting** the generated reports*

- The Report generated in last step can be saved to AWS S3 and its metadata to AWS RDS.
- I fetch all the generated Reports from AWS S3 and RDS and display it in a list.
- I can select a report from the list of reports to view its content and I can also delete any given report from AWS RDS and S3.

HIGH-LEVEL OVERVIEW OF YOUR DESIGN AND CHANGES



The application logic layer has the plagiarism engine which contains core logic for detecting cases of similarity between two uploaded java projects. POJO classes are used to store the data in memory and been mapped to the database using Hibernate ORM. The core engine for checking plagiarism has been written in Java. It utilizes JavaParser to parse a given string of code into an Abstract Syntax Tree. Also, I am comparing these generated trees using the tree edit distance algorithm. The RESTful web services integrate every component in the backend, they have been developed using Spring MVC. The server side application in its entirety is a Spring Boot Application, with an embedded Apache Tomcat server. It builds into a target jar file, which can execute on its own. Upon execution, it starts listening for any incoming requests from the client side. Our current design has not deviated much from the initial design submitted earlier. The only difference is in the client side, React has allowed us to combine all 3 client side modules to a single, Component based module which simplified the front end development.

ALGORITHMS USED FOR PLAGIARISM DETECTION

Plagiarism Detection between Multi-File Projects

Given two multi-file projects, the system iterates all possible pairs of files to check the similarity, no matter whether they have the same name or not. Only the file pairs with similarity score ***S*** higher than a threshold T_{score} will be considered as matched files. When a pair matches, the system further detects the suspicious similar parts of code, called line matching, and the system matches the rest pairs. After the step completes, a summary is generated for the two projects.

1. Line matching based on syntax parse tree

   The algorithm is based on syntax parse trees(refer to section 2.3). It is interested in matching node types, including class/interface, constructor, method, for/if block. The nodes from two trees are matched if they have same subtrees, and the line range for the node will be added to the summary. As for the method to match nodes efficiently, the system traverses the tree and generates the string representation for each node. Interested nodes and string representation are stored in a hash map. Using hash maps, the nodes with the same string representation are matched efficiently.

Plagiarism Detection between Java Files

The system checks 3 types of similarity between files, including MD5 checksum, text words and parse tree generated by the file. After the plagiarism engine checks the similarity between two files, it will generate a similarity score ***S*** within range of [0, 1].

1. MD5 checksum similarity S_{MD5}

   The algorithm checks whether the two files have exactly the same content. When two files have the same MD5 checksum, there is no need to do further comparison and $S = S_{MD5} = 1$. Otherwise, the system continues to check the rest two types of similarity.

2. Text words similarity S_{text}

   The algorithm checks whether the two files have similar text content, considering term frequency of each word. It ignores the order of the words, and helps to detect plagiarism if the most content of comments or variable names are similar. The similarity score S_{text} is given 0.2 weight in the final result. To calculate S_{text}, each file is transformed into a list of tokens, excluding java keywords. Then the tokens are represented by an array of integers, using the method of bag of words(BOW). The system takes cosine similarity between arrays as S_{text}. When all the words have same frequency, S_{text} equals to 1.

3. Parse trees similarity $S_{parseTree}$

   The algorithm is based on syntax parse tree, detecting whether the code have similar structure by replacing variable names, etc.

   At first, each file is converted to a syntax parse tree, with each node in the tree representing a block or a line. Tree edit distance(TED) and node type occurrence are considered to calculate $S_{parseTree}$. TED is the number of operations needed from the source tree to the destination tree. It is normalized by considering the size of two trees, and S_{TED}=1 means the trees are perfectly match. Node type occurrence considers the similarity between the frequency of the set of interested node types.

REPORT ON SOME EXPERIMENTS CONDUCTED WITH OUR TOOL

The system provides several results for the sample projects.

Sample Set	Percent Score	Our Verdict	Sample Set	Percent Score	Our Verdict
Set01	82.48 %	True Positive	Set06	55.72 %	False Negative
Set03	36.27 %	False Negative	Set09	40.00 %	True Negative
Set04	92.02 %	True Positive	Set11	97.42 %	True Positive
Set05	37.43 %	True Negative			

TECHNIQUES COVERED IN THE COURSE

DESIGN PATTERNS

- Visitor Design Pattern:
    1. NodeCountVisitior and SyntaxVisitor have been implemented using visitor design pattern. Using visitor pattern, I can walk over JavaParser AST and our own AST; and perform different operations like node count, check the type of node.
- Factory Design Pattern:
    1. While traversing the JavaParser nodes using visitor pattern, for each JavaParser Node; an appropriate node in our convention is created using CustomNodeFactory class. CustomNodeFactory has been implemented using factory design pattern.
    2. Creation of Entities object is achieved using Factory Design Pattern.
- Singleton Pattern:
    1. As I need only one instance of CustomNodeFactory, a single instance is created using Singleton Pattern.
    2. For creating Restful services like creating project request or report request, we have used singleton pattern.
    3. Accessing the Database is achieved using single instance of the database connection.
    4. All plagiarism engines have been implemented using Singleton pattern as I need only single connection per engine to run plagiarism and generate reports.

TESTING

I have written Unit Tests and Functional Tests for testing the system. Following are some stats of our Testing:

Packages	Statement Coverage	Branch Coverage	Comments
server. plagiarism.config	0	0	Not required as it is a config for springboot
server. plagiarism.controller	0	0	Tested in integration
server. plagiarism.dao	98	98	IO exceptions cannot be tested
server. plagiarism.engine	100	100	
server. plagiarism.entity	87	94	POJO classes, testing not required
server. plagiarism.exceptions	100	100	
server. plagiarism.interceptor	100	100	
server. plagiarism.requestbody	16	20	POJO classes, testing not required
server. plagiarism.service	100	100	
server. plagiarism.interceptor	92	93	IO exceptions cannot be tested

REFACTORING

- In first draft of system, the code for updating the Node Type count was in every visit method of NodeCountVisitor. In final version, it has been replaced with a routine (updateHashMap()) in every visit method of NodeCountVisitor
- In first draft of system, code for creating Expression and Statement was written in every visit of our visitor. In final version, it has been replaced with routines like makeExpression(...) and makeStatemet(...).
- PlagiarismEngine had method calculateSimilarity() initialized to have input as two projects and generate report for the projects with each engine. In final version, it is refactored to take two files as input and comparison between projects is implemented by PlagiarismSummary instead.