Project Overview:

The objective of this project was to effectively design, build, and test a web-based application that enables instructors to detect plagiarism and academic dishonesty, or situations where two or more students submit similar solutions. We have demonstrated our understanding and application of classroom concepts as we worked across the semester on this tool.

Throughout the project, we have adopted the Waterfall SDLC as a methodology to guide our development. We determined that this would be the best software development methodology for our development. We did some collective research that suggested that the methodology typically prevails when there are constraints on the project, namely cost or time, and the requirements and scope are well understood or effectively provided.

To begin with, our team had a strong understanding of the requirements and scope based on the project overview and guidelines. As a result, the waterfall methodology provided our team with a distinct set of processes that were built on the principle of approval in each of the previous phases. In the first phase, we began by writing a set of use cases that describe the behavior of the system, as well as UI mockups that represent how the user would interact with the plagiarism tool. In the next phase, we created a UML diagram that helped our team visually represent the architecture and design of the system. In the next stage of the process, we began the actual development and testing process. We also determined that this methodology helped our team accurately define what we would be building in a detailed manner at the beginning of the process. In addition, since the project timeline was fixed and could not be moved, this enabled us to make sure that our deliverables met the deadlines that were set for us, and that we tracked progress based on these deadlines. We also found that since we had good insight into the product requirements and this was effectively documented and approved prior to the beginning of development in Phase B, we were mostly able to deliver a specific set of features which made our final product more consistent with the requirements.

This report further outlines the installation instructions and explores the system functionality, technologies, high-level architecture, plagiarism detector algorithm and testing.

Installation Instructions:

Setting up Node.js for Mac/Linux using HomeBrew

Homebrew makes the process of installation of Node a one-step process. By using Homebrew, we do not need to manually add the path of node executable.

1. From the terminal ***execute*** the command: $ brew install node
2. Node is installed on your system now.
3. ***Test*** the installed packages (Restart your computer)
4. ***Test Node:*** open command prompt and type node -v. You should see the downloaded version of node as “v7.3.0”
5. ***Test NPM:*** In command prompt, type npm -v. You should see the downloaded version of npm as “3.10.10”

Setting up Node.js for Windows

1. ***Download*** the windows installer for Node.js from the **Node.js** website
2. ***Run*** the installer (the .msi file downloaded in Step 1)
3. ***Accept*** the license agreement and all defaults and click install.
4. ***Test*** the installed packages (Restart your computer)
5. ***Test Node:*** open command prompt and type node -v . You should see the downloaded version of node as "v7.3.0" (the version you just downloaded in the previous step).
6. ***Test NPM:*** In command prompt, type npm -v. You should see the downloaded version of npm as "3.10.10" (the version you just downloaded in the previous step).

Setting up MySQL Workbench

1. ***Begin*** by installing **MySQL Workbench** on your local computer.
2. ***Start*** MySQL Workbench
3. Click the + button near MySQL connections
4. ***Click*** the 'Local instance MySQL' button
5. ***Click*** connect to begin the configuration process
6. ***Perform*** a mysqldump to recreate the login database in the same state as it was at the time of the dump by the team .../.../Team-06/Dump For CopyCat.sql

Project Installation

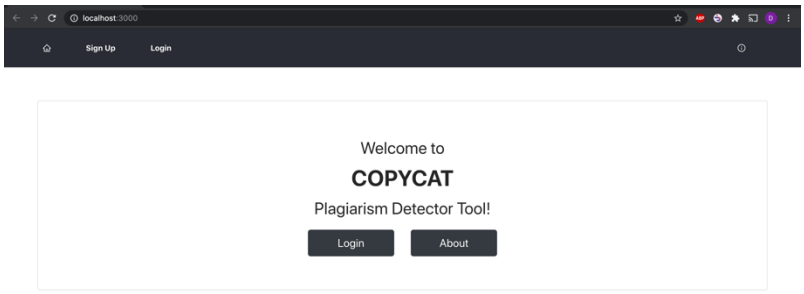
1. ***Clone*** the **team project repository** from **Github**
2. ***Navigate*** to .../.../Team-06/phaseC/CopyCat/server in the project repository on your local computer
3. ***Open*** the index.ts
4. ***Find*** the mysql.createConnection instance
5. ***Change*** the password to your mySQL workbench password for local host
6. ***Run*** `npm install` for both the client and server directories
7. ***Navigate*** to Team-06/phaseC/CopyCat/server and run `npm start` in the terminal
8. Simultaneously, ***navigate*** to Team-06/phaseC/CopyCat/client and run `npm start` in the terminal
9. The application client will ***launch*** on `http://localhost:3000`
10. The application server will ***start running*** on `http://localhost:3001`

Troubleshooting: If you have any challenges connecting the server to the database, you must run the following SQL query within the login system database on MySQL Workbench and replace userlocalhostpassword with your local password.

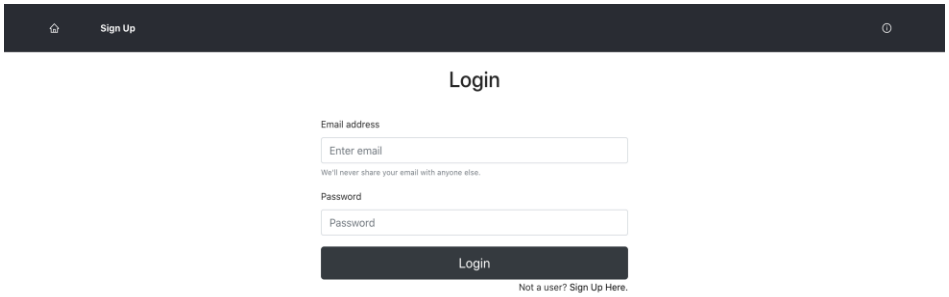
```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'userlocalhostpassword';

flush privileges;
```

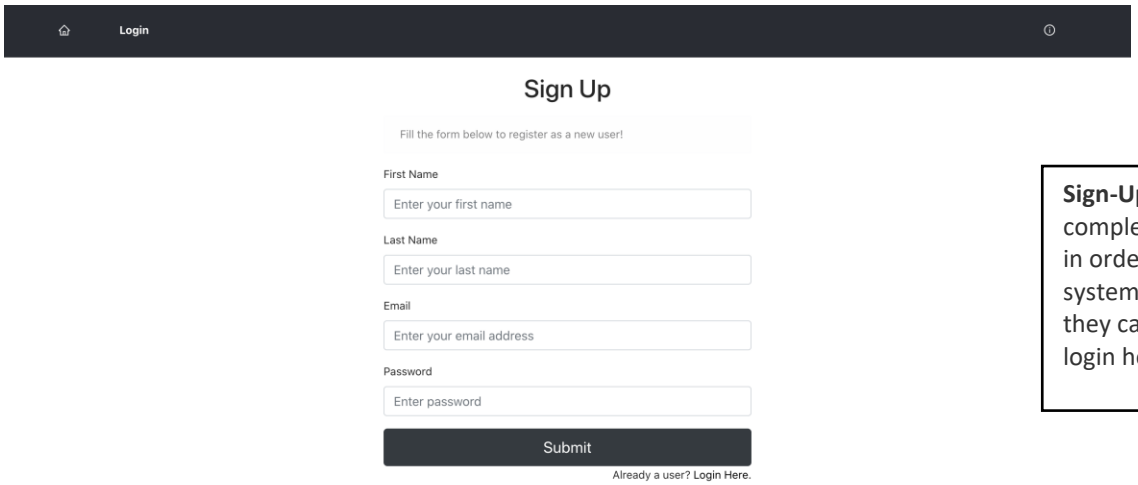
User Guide: To begin using the system, the user **must follow** the installation instructions above. Then, the user can navigate to **http://localhost:3000/** which will land on the homepage of the web-based plagiarism tool.



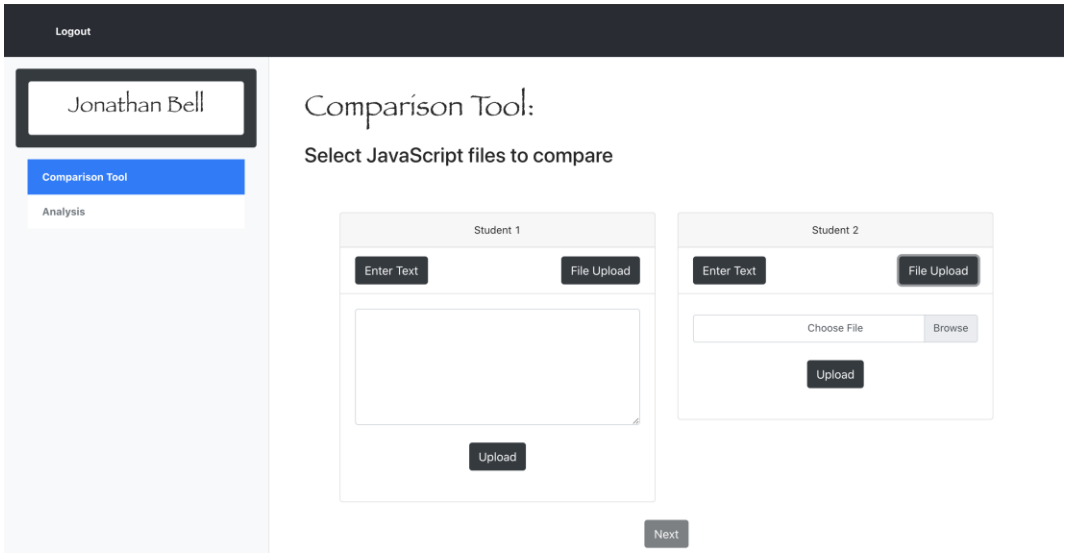
From the home page, the user may navigate to either the about section, and learn more about the system functionality, or login to the system.



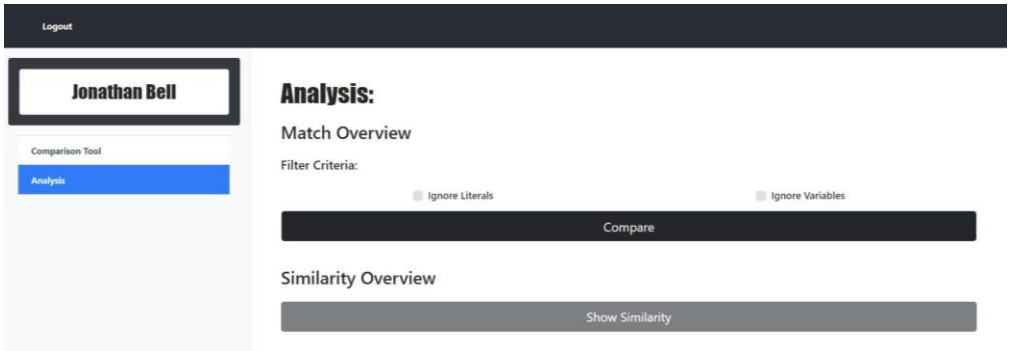
Login: On the login page, the user must enter their email address and password. If they are not a user of the system, they can click sign up here to create an account.



Sign-Up: The user must complete the sign-up below in order to login to the system. If already a user, they can sign in through the login here button.

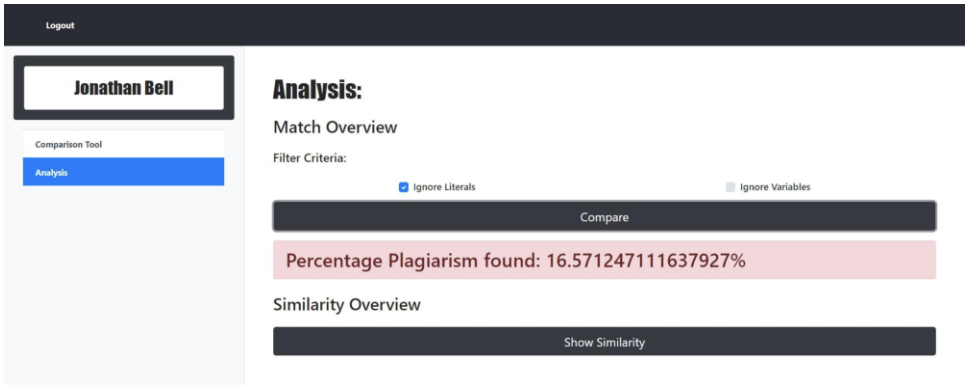


Once a user is logged in to the system, the user views the comparison tool. This tool enables users to add or upload single and multiple files. The tool supports two input methods, text input or file upload. Only once the system recognizes that files have been uploaded for **both** Student 1 and Student 2, the next button is enabled.

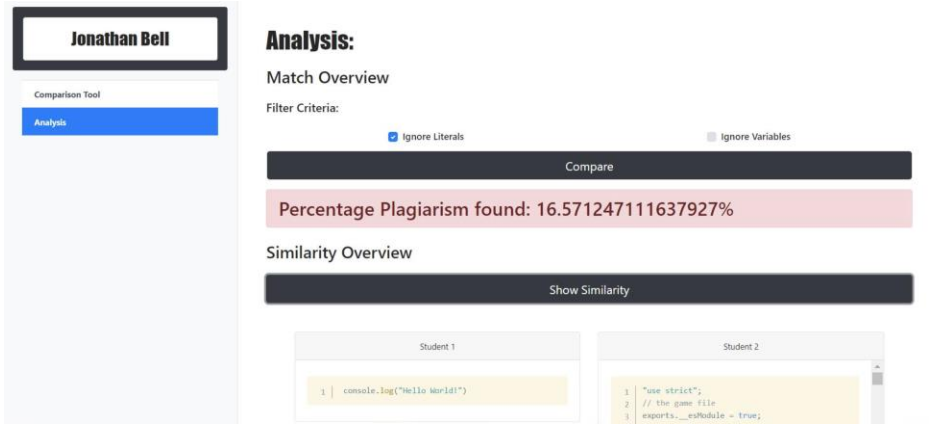


One a user clicks next; they view the analytics page. On the analytics tab, under filter criteria, the user can select to ignore literals or variables in the comparison, which will change the similarity percentage that is

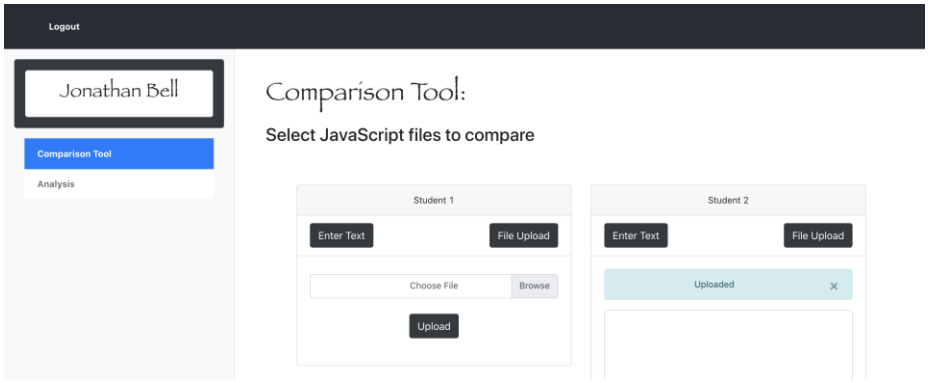
---



The user must click the compare button, to display the similarity percentage between Student 1 and Student 2. Show similarity is only displayed to the user after the comparison is done, and the results are displayed as a percentage, under **Match overview**.



Then, the user can navigate to the similarity overview section, where selected portions that match between the students are highlighted.



For security and privacy purposes, the user must log-out to use the comparison tool again and none of the student files will be stored in the system.

High Level Architecture & Overall Infrastructure:

We adopted a model that has a distributed application structure that partitions the key tasks between the server, and service requesters or client. In this client-server architecture, when the client sends a request for data to the server through the internet, ***specifically uploading and selecting student files that are being compared and scanned for similarity using our plagiarism detector***, the server accepts the requested process, ***runs the plagiarism and similarity algorithm*****.** Then, the server delivers the data,

more information about the code similarity, that was requested back to the client. Our application design also requires a database that stores the usernames and passwords of all users allowed to access the website, specifically instructors and teaching assistants.

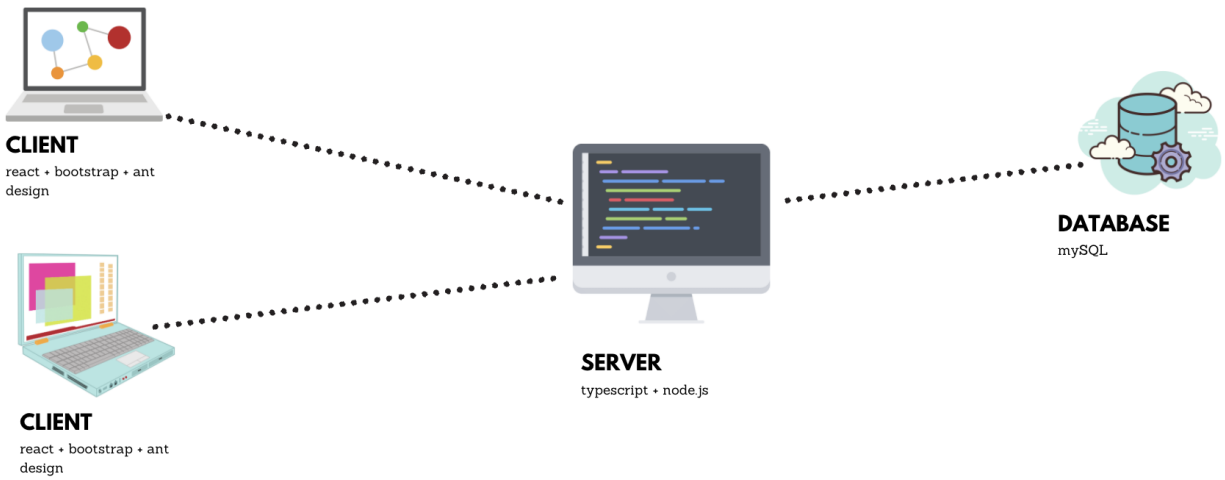
The UI design that we designed in our high-fidelity prototypes and implemented in development is simple, effective and intuitive for the user. On the information page, the user can find more information about algorithm to better understand how similarity and plagiarism is detected.

Technology:

Our front end has been developed in React as stated in the project specifications. More specifically, we wanted to outline and discuss some of the libraries and technologies that helped us achieve our dynamic UI functionality. We used Bootstrap, and Ant Design components as they are ideal for making responsive, fast React apps. We used Prism.js, specifically for displaying and highlighting code snippets in similarity overview. To perform HTTP requests, we used a very popular JavaScript library, Axios that works in both Browser and Node.js platforms. We also utilized react-router-dom for the client-side routing. MySQL is utilized in our project to create a DB connection with user information, and npm-file-system to create and read source code and submission files.

Our back end, and main logic for the program is developed in Typescript. Additionally, we used Express.js framework which is a Node js web application server to bind multiple pages of frontend to the backend. We did use existing libraries for basic infrastructure that we are going to outline. For parsing the code and building the AST, we use babel and we use string-similarity, which is based on Dice's Coefficient to help determine the similarities between the two strings.

High-Level Architecture Diagram:



Plagiarism Detection Algorithm:

Our string similarity comparison results created by the system represent the relative likelihood of how a comparison between two students contains plagiarized code; the higher the percentage, the higher the likelihood of plagiarism.

We begin by converting and pre-processing the source code to reduce the size of the source code within the repository and to enable a faster comparison between the student source-code submissions.

Each file is read as a string and parsed into an AST using the Babel parser (@babel/parser). Then, the AST is traversed to transform the JavaScript source code to *Four String Representations* of the code using the Babel Traverse (@babel/traverse). The four string representations include strings for file structure, identifiers, literals and tokens. These are outlined in more detail below:

Example code:

```
var a = 10

console.log("Hello World")
```

1. File structure: The first string is a line-oriented representation of the code dividing the source code into a series of tokens representative of the entire line. For the file structure, if the line contains more than one statement, only the first statement is detected and captured. For the example code above, this string would be -

   "ProgramVariableDeclarationExpressionStatement"

2. *Identifiers:* The second string is an alphabetically sorted list of all identifiers using hash-codes. While the source-file is tokenized, the parser stores all identifiers in a symbol table, with 32-bit hash codes derived from and assembled as a hexadecimal representation in a string. For the example code above, this string would be -

   "1f40fc92da241694750979ee6cf582f2d5d7d28e18335de05abc54d0560e0f5302860c652bf08d560252aa5e74210546f369fbbbce8c12cfc7957b2652fe9a75eed55db3ffa1983455e1c1b291f6d4d48009bbf43338657ac7a49631126140f126f0a95456b60535e3a7902acd712a62044b445972a8b6c1e79c7cbbb80bc01f873bcc37e512b7da86476367769c932009fc1bee59c929879f10ac89df541124db5010ae7297ec71db8f71a09adccd27c002f00fcfc93ca7e157105e7505f24d"

3. *Literals:* The next string, holds versions of literals that have been trimmed, to remove all spaces, tabs etc. For the example code above, this string would be -

   "10HelloWorld"

4. *Tokens:* The last string represents all the tokens of the source code and stored as the node type of each token. This is representative of the sequence of all the tokens as they appear on the source code. For the example code above, this string would be -

   "ProgramVariableDeclarationVariableDeclaratorIdentifierNumericLiteralExpressionStatementCallExpressionMemberExpressionIdentifierIdentifierStringLiteral"

Upon completion of pre-processing of multiple source codes, we begin to find the degree of similarity between two strings, based on Dice's Coefficient, using the 'string-similarity' npm package. The basis, the Dice Similarity Coefficient is a simple and effective way to calculate a measure of the similarity of two strings, by comparing the number of identical character pairs between the two strings. The outcome is a value bounded between zero and one that is then converted to a similarity percentage. More specifically, the algorithm works by finding the common tokens, and divides it by the total number of tokens present by combining both sets. The numerator is set as twice the intersection of two sets/strings in the formula, and the denominator is a simple combination of all tokens in both strings. The rationale for this is to ensure that if a token is present in both strings, its total count is obviously twice the intersection which discards duplicates.

$$S = \frac{S_f + S_i + 2 \cdot S_l + 10 \cdot S_t}{14}$$

To determine the overall similarities between the two source codes, we calculate a weighted average of the four similarities including file structure, identifiers, literals and tokens. This is the formula that is utilized to compute the overall similarity between two student submissions: S represents the total similarity as a weighted average of the four similarities: S_f (file structure), S_i (identifier), S_l (literals) and S_t (tokens).

We used a research paper to help us determine the best way to implement the plagiarism algorithm. This was published by D. Pawelczak in 2013: Online detection of source-code plagiarism in undergraduate programming courses.

Phase B to Phase C:

Our group tried to not make any significant changes in our project between Phase B and Phase C based on the software design methodology that we adopted. However, there were some limitations and we were unable to implement all of the functionality that we had outlined in our Phase B report. We are briefly going to discuss all of the changes, based on the categories outlined in Phase B, and then discuss any overall changes that were made.

Changes in Requirements and Functionality:

Our group faced significant challenges and but managed to mostly implement all the functional requirements iteratively throughout the process. We implemented all of the **must have** requirements, and prioritized these in our implementation, but were unable to implement some **should have** requirements. Below, we summarize and outlines the key changes that were made in the functionality of the plagiarism detector, and discusses which functionalities were achieved.

The functionalities that were achieved:

1. The tool must work on code written in at least one language of your choice: *This was achieved for student submissions that are **only** written in JavaScript*
2. The tool must take two programs: This was effectively achieved by our group and is represented as student 1 and student 2 in our UI interface.
3. The tool must go beyond textual "diff": *This was achieved through the implementation of pre-processing the code, building an AST and generating multiple string representations*

of the source code that we compare similarities between to analyze different components of the code.

4. The tool must handle multi-file programs (should be able to handle files that have been renamed): *This was achieved, and our UI component and backend functionality support single file upload and multi-file upload.*
5. The tool should detect strategies to avoid detection such as renaming variables, extracting code into functions, moving code, changing comments etc.: *This is effectively achieved through our pre-processing stage, where effectively convert the source code into 4 string representations by building an AST and then find the similarities between the 4 strings against each other.*
6. The tool must produce an interactive user interface: *This was effectively achieved through our client-server architecture and the numerous technologies that we have discussed in our overall architecture and design section.*
7. The tool must use a web-based interface: *This was achieved, and our tool is hosted on localhost:3000.*
8. The user interface should include kind of visual diff of the code portions: *This is achieved through selective line highlighting and display on our UI.*
9. The tool should have an Instructor Login: *This is effectively achieved; we also utilize a database to store the information for username and login information.*
10. The tool should be able to take two modes of input: *This is achieved as the UI supports both text upload and file uploads*
11. The tool should display the overall percentage of similarity between two programs as a high-level report: *This is achieved through our weighted average calculation of the overall string similarity between the multiple comparisons that occur in our detection algorithm.*
12. The tool should highlight code which are logically similar in both the programs in the same color- *We have **mostly** achieved this functionality but could refine the core logic that determines the code similarities more effectively.*
13. The tool should offer an option to Log Out: *This was achieved in our UI, and we also delete all of the student files that have been uploaded for privacy.*

The two functionalities that we were unable to achieve are outlined below:

14. The tool should be able to download the Plagiarism Analysis Report: *This is a functionality that we were unable to achieve during our process. This is because we could not determine whether this functionality should effectively be implemented on the server, as we were already displaying the information to the client, and we also struggled to determine how to effectively use the jsPDF library.*
15. The tool should allow a user to selectively examine and compare key portions of the program in the browser: *This is not achieved through selective line highlighting and display on our UI.*

UI Changes: The main changes that we made in our UI were on the analysis tab. Based on the plagiarism detection algorithm that we implemented; we were unable to support all of the filter functionalities that we had initially wanted based on the algorithm we chose. We also made some small UI changes, based on our groups limited experience and knowledge with building react apps, including removing our logo on every page.

UML Changes: We do not support all of this filter criteria, and we wanted to enable the user to have freedom and flexibility over the criteria that they select, and even allow them to input their own criteria, but we do not support that, based on our algorithm and the limitations of evaluating the similarities based on the code comparisons defined by the user. We also no longer support or require a user class, which we had previously outlined when we had created our UML diagram in Phase B. We also implemented a library, so we no longer needed all of the classes for the AST that we had initially outlined in our UML diagram.

Programming Language Selection:

In Phase B, we began by creating a plagiarism detection tool that would work with student files that had been written and uploaded in multiple languages including Java and Python. Over the course of a few weeks, as a team we decided to make the switch to developing a tool that would support plagiarism detection in only JavaScript which has good libraries for parsing, and which wouldn't require creating separate node classes. Upon doing so, we were able to implement and utilize Babel, a source-to-source compiler. We utilize the Babel parser to now generate an Abstract Syntax Tree according to Babel AST format. We also wanted to utilize this as an opportunity to familiarize ourselves with the numerous existing libraries that were available for JavaScript and React.

Software Engineering Processes:

Design Patterns:

For creating our plagiarism detector, we decided to implement the Factory Design pattern. The key advantage of utilizing this creational design pattern was that it promoted loose-coupling and eliminated the need to bind application-specific classes into the code. That means the code interacts solely with the resultant interface or abstract class, so that it will work with any classes that implement that interface or that extends that abstract class and enables us to scale our project to support plagiarism detection in multiple languages. The PlagiarismDetectorFactory creates instances of the detector.

IPlagiarismDetectorFactory (Creator) - The Creator class of the Factory Pattern. It creates a PlagiarismDetector which builds the AST, traverses it and build the 4-string representation.

PlagiarismDetectorFactory(Concrete Creator) - A class that implements the IPlagiarismDetectorFactory interface. This is the concrete creator of the Factory Pattern which creates the concrete product PlagiarismDetector.

IPlagiarismDetector (Product) - This is the product interface of the Factory Pattern. The detector builds an AST of the program and traverses it to build the 4-string representation.

PlagiarismDetector (Concrete Product) - This is the concrete product of the Factory Pattern. It implements the product interface IPlagiarismDetector. The detector builds an AST of the program and traverses it to build the 4-string representation.

Modularity & Encapsulation: We were able to achieve effective modularization and encapsulation, two important concepts in software development, in our project. While both of these concepts are not difficult to implement in most programming languages that we have previously used, we determined that implementing them in Typescript was not as intuitive and required our team to develop a more robust understanding of Typescript this semester. The key reason that we wanted to ensure that our code was modular, was that modularization provided an effective and efficient way to develop code that

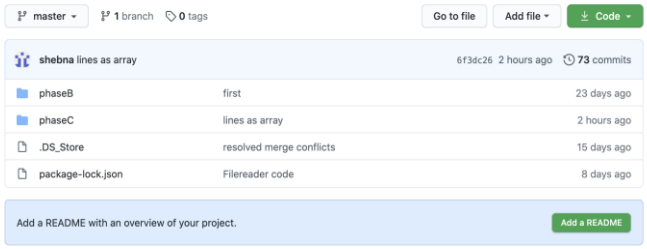
is more understandable and maintainable for the whole team. We all had different coding styles, and through this we ensured the bundling of our code. Through implementing effective code organization, we were able to mostly divide up functionality of our web-based plagiarism detector and provide encapsulation, which offers us both code reusability and code extendibility, if we choose to write code that would support more than one programming language as a source code, which we initially intended to do.

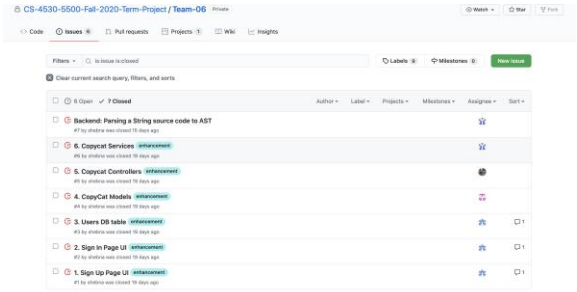
After understanding the importance of usability in the UI, we decided to ensure our UI had good error messages that informed the user of their errors. We additionally ensured that we used consistent colors, and formatting for buttons to mitigate any confusion that the user had. Another heuristic that we tried to implement is an aesthetic and minimalist design by removing any irrelevant information from the interface.

Development Process:

Version Control:

Our group used Git, a distributed version control system that allows for backing and sharing code. We also used GitHub, a web-based git repository hosting service, which offered our team distributed revision control and source code management. In addition, after our meetings, we would discuss and allocate tasks that would further enable us to assign issues to each other and manage our project progress.





Code Review:

In order to ensure that we found bugs early and that our code quality did not drop, we made sure to periodically review each other's code throughout the process of building the plagiarism detector.

We created a comprehensive checklist that helped us determine what we needed to accomplish in each of the code reviews. The first thing that we did was check and determine if the code was readable and understandable. As a result, we also focused on reviewing the comments, making sure that there was appropriate documentation, and the code was formatted correctly. The next priority was ensuring that we consistently tested our code frequently to check for accuracy in output and fix any bugs as and when we found them in the development process. This made sure that we maintained the quality of the code. Throughout the process, we also checked to see if the code was reusable and that our code was (DRY) by checking for any duplication that could be avoided. Lastly, when reviewing the code, we kept in mind the SOLID principles to make sure that our code was as efficient as possible.

Our code reviews were performed virtually through Microsoft Teams and Zoom. In these sessions, there were 1-2 members of the team discussing and reviewing the functionality of the project. Moreover, these discussions helped us resolve any inefficiencies and errors in our code on a periodic basis.

One of the interesting strategies that our team adopted was pair programming. This specifically enabled our team members to work on the same code together and thereby check each other's work and ask questions throughout the process. This also served as an effective way for our more senior and experienced team members to mentor the less experienced and novice programmers in the programming process.

Testing:

Throughout this semester, we have learned how software testing is a methodology that determines whether the software product matches expected requirements; and enables us to identify errors, gaps or missing requirements in contrast to actual requirements. Our project required us to adopt both **white box** and **black box** testing methodologies to ensure system accuracy. For the server, and more specifically the backend functionality, we decided to write a comprehensive collection of test cases, or test suites, for each of the classes. Here, we were exceptionally mindful of null values, empty strings and other user-defined inputs that were not directly supported by our program functionality.

In order to effectively test our user interface, and the appropriate error messages based on the user interactions, we performed manual testing and have attached a file in our repository that contains the screenshots of all the errors that we tested.

In order to effectively test our algorithm, we decided to implement a black box testing suite for single and multi-file uploads through the user interface and further enabled us to determine if we were achieving the desired results.

In order to test the effectiveness of the algorithm, we utilized sample test data from provided by the course instructors and our own written source code.

Future Scope:

With future iterations of this project, we wanted to outline some of the core functionality that we wanted to be able to achieve:

1. The system should be able to support comparisons between online source code and not only compare between students.
2. The system should support multiple students code comparisons and enable the instructor to cross-check the plagiarism and ensure academic honesty in the whole class.
3. The system can parse over source code that is written in Java and Python and also extend to cross-language comparison.
4. The system should enable the user to download a plagiarism report that highlights the key similarities between source code.
5. The system should have an admin or super-user who approves the sign up requests, this will make the sign up process exclusive to instructors and prevent students from signing up.

6. The system should be able to send out email notifications on sign up as well as for request approvals.

Works Cited:

"@Babel/Parser · Babel." Babel, babeljs.io/docs/en/babel-parser.

Kurdekar, Akash. "https://www.npmjs.com/Package/String-Similarity." Npm, Npm, Inc., www.npmjs.com/package/string-similarity.

Mark Otto, Jacob Thornton. "Introduction." · Bootstrap v5.0, getbootstrap.com/docs/5.0/getting-started/introduction/.

Parsons, Tim. "When to Use Waterfall vs. Agile." Macadamian, 17 May 2019, www.macadamian.com/learn/when-to-use-waterfall-vs-agile/.

Prasanna, Vijay. "JavaScript Object Oriented Patterns: Factory Pattern." DigitalOcean, DigitalOcean, 22 Nov. 2020, www.digitalocean.com/community/tutorials/js-factory-pattern.

Pawelczak, D.. "Online detection of source-code plagiarism in undergraduate programming courses." (2013).

Prism, prismjs.com/.

Publii Team. "Highlight Your Code Syntax with PrismJS." Publii, Publii Team, 20 Nov. 2020, getpublii.com/docs/highlight-your-code-syntax-with-prism-js.html.

Robinson, Scott. "Read Files with Node.js." Stack Abuse, Stack Abuse, stackabuse.com/read-files-with-node-js/.

"What Is Babel? · Babel." Babel, babeljs.io/docs/en/.