**CS5500 Foundations of Software Engineering**

**Team 25** (Mikayla Werzanski, Will Cohen, Ture Carlson, Chris Martin)

# Term Project Report

## 0. Introduction

BooPlagiarismBad (BPB) is a web-based application intended to allow Computer Science instructors to compare student's code submissions to determine whether a given submission was likely plagiarized from another submission.

## 1. Installation Instructions

BPB may be installed and deployed in multiple ways depending on the destination environment. While the intended eventual deployment environment for BPB is a cloud platform with support for containerization, the below deployment options should suffice for auditing of the application's capabilities in a more localized environment.

**Vagrant:** Team 25 created a Vagrant development environment which may be used to launch the BPB application for development and evaluation purposes. Vagrant is a VirtualBox / VM management tool which allows the user to easily instantiate and provision a virtual machine using configuration-as-code.

a. Install VirtualBox and ensure the host system's BIOS settings match Vagrant documentation (see Appendix A: Vagrant Configuration). You may need to configure system-wide virtual machine support in order to allow VirtualBox to function correctly if you have not previously installed VirtualBox.

b. In the project directory, run the command **vagrant up**

c. After a delay, the BPB application should be accessible at **192.168.33.10:3000**

See Appendix A: Vagrant Configuration for more information on how to restart the server, change configuration options, and run tests in the VM environment.

**Manual Installation**: The BPB application may be installed manually by configuring the deployment environment.

a. Install MongoDB on the host machine. Installer binaries may be found here: https://www.mongodb.com/try/download/community?tck=docs_server

Additionally, if using Linux, you may install MongoDB using your package manager (e.g. sudo apt-get install mongodb)

b. Ensure all environment variables which are required are correctly set in the deployment environment. This step may require editing environment variables on your host system, if they are not already set. (See Appendix B: Required Environment Variables)

b. Navigate to the front-end app directory (bpb-front/) and run **npm install** and then **npm run start**

c. In a new terminal, navigate to the back-end app directory (bpb-back/) and run **npm install** and then **npm run start**

2. User Guide

Once the BPB application is launched, you are ready to create **new assignments**, make **submissions** to those assignments, and **compare** any submissions to determine whether there are common elements between those submissions.

If any of the steps below are unclear, please click "Help" in the BPB application for more details.

1. **Create an Assignment**

   a. On the main application page, click **Create Assignment** You will be shown a dialog which allows you to enter a name for the new assignment.
   b. Enter any name you wish, then click "Create Assignment" to confirm.
   c. The new assignment should appear in the list of assignments. (Note: you may need to refresh the page).

2. **Upload a Submission**

   a. On the home page, click any assignment you wish to submit to.
   b. Click **Upload a Submission**. You will be taken to the Submission creation screen, which allows you to enter a name for the new submission and attach any number of submission files.
   c. Enter a name for your submission, then drag-and-drop the files you wish to upload into the upload area.
   d. Once you are satisfied with the contents of the submission, click "Upload" to create the submission. The new submission should appear in the list of submissions.

3. **Compare Submissions**

   a. on the home page, click any assignment you wish to analyze, then select two submissions to compare by checking the appropriate boxes next to each submission.
   b. Once you have selected two submissions, click "Compare Submissions."
   c. Once comparison is complete, you will be shown a detailed breakdown of how the submission files compare to one another in terms of originality.

      a. **View Most Similar Files**: Review the match box at the top of the screen to see which files were most similar to one another, including their derived similarity percentage.

      b. **View Specific Files**: Click a file's name in either submission to show that file's content. When a file is selected, the list of matches will display those sections which match between the two files currently being compared.

      c. **View Matched Sections**: Click any match in the middle section to isolate and display the specific subsections of each file which matched

## 3. Project Architecture and Infrastructure

The BPB application was developed using a layered project architecture which separates the user-facing component of the application (front-end) from the underlying application services (back-end).

### bpb-front

The BPB front-end application, **bpb-front**, was developed in Typescript and React.js.

As the client, bpb-front has been designed to serve a user interface, delegating the manipulation, computation, analysis, and storage to server-side operations handled by bpb-back. By encapsulating these features, we were able to streamline bpb-front, and left open the possibility of designing an entirely different client without significant changes to the server.

bpb-front was constructed using an asynchronous approach that leverages Redux stores to reduce the number of outbound data requests to bpb-back. This also allows various components on a page to share the same information from the store. We leverage this in our comparison view, allowing multiple points of data visualization and manipulation without passing so many things back and forth.

We implemented a single-page-application approach that does not need to refresh with every transition: instead, refreshes only happen when manually triggered by the user or required by internal operations. This further limits the amount of outbound data requests, and increases the speed of local-only operations.

**bpb-back**

The BPB back-end application, bpb-back, was developed in Typescript and Node.js.

bpb-back leverages Express.js to expose a series of rich APIs which allow bpb-front to create, read, update, and delete both assignment and submission data.

bpb-back's Routers delegate valid incoming requests to its Managers, SubmissionManager and AssignmentManager. Managers manage execution of core application functionality on behalf of the requestor.

bpb-back ensures that all incoming requests are processed in a non-blocking, asynchronous fashion, ensuring that long-running CPU-intensive calculations (e.g. submission comparisons) run in separate threads from the main program in order to prevent blocking of the event loop.

bpb-back utilizes an intelligent cache design for incoming requests, which ensures that long-running and computationally expensive tasks are not executed repeatedly if such execution is not required. Repeated requests for individual documents and/or comparisons will be blocked if processing is ongoing, and requests for previously retrieved information will be retrieved from caches rather than from the application database.

bpb-back uses Mongoose ORM and MongoDB to ensure that assignment and submission data are persisted over time and between server restarts.

bpb-back follows configuration-as-code principles by abstracting out application configuration parameters as environment variables and storing default values for those variables in version control. The execution environment for bpb-back in the evaluation Vagrant deployment is managed via PM2 process manager, which injects the appropriate variable values based on the configured deployment environment.

See Appendix C: UML Diagrams for a detailed breakdown of the application's class-level architecture via UML diagram.

## 4. Plagiarism Detection Algorithm

BPB's file comparison analyses are powered by a plagiarism detection algorithm based on "DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones (2007)" by Jiang, Misherghi, Su, Glondu. This paper proposes a similarity analysis conducted by comparing all elements of one program's abstract syntax tree to that of a comparator program/file and determining the overall number of nodes which are considered to be similar to one another using a locality-sensitive hashing methodology (See Appendix D, Analysis Citations).

Locality-sensitive hashing prioritizes textually similar subtree elements while allowing for dissimilarity that might be introduced through the use of obfuscation techniques. When all subtree nodes have been compared, an overall percentage similarity between the two programs or files can then be determined by derivation using per-subtree similarity information.

$$Similarity(T_1, T_2) = \frac{2H}{2H + L + R}$$

*"...where H is the number of shared nodes in trees T1 and T2, L is the number of different nodes in T1, and R is the number of different nodes in T2." (DECKARD (2007) § 4.2, Jiang, Misherghi, Su, Glondu )*

BPB's similarity analysis operates at the file-to-file level, meaning that each pairing of files across two submissions is analyzed to determine its similarity. Submissions are effectively judged by the number of highly similar nodes in the comparator file's abstract syntax tree.

BPB uses the java-ast library to parse incoming Java-based programs into abstract syntax tree objects. BPB uses AnalysisResultEntryCollectorVisitor visitors, each of which traverses a given Java file's abstract syntax tree and generates a locality-sensitive hash value for each node using that node's subtree's textual content as input.

Hashing of individual subtree elements is performed using TrendMicro's TLSH library (See Appendix D, Analysis Citations). Hashed subtree elements, referred to as AnalysisResultEntries, are stored in the BPB application's database upon submission upload for all files within the specified submission.

When a comparison between two submissions is requested, BPB generates a similarity analysis for each file-to-file matching using the above methodologies. The results of the AST-level comparison are then used to derive similarity percentage scores based on the approach proposed in DECKARD This information is then displayed to the user.

## 5. Design Evolution and Refactoring

The design of the BPB application evolved in logical ways from the design which was initially proposed, mostly due to timeline and feasibility concerns which necessitated the development of a "minimum viable product" rather than a more fully-featured solution.

### 1. Evolution of design since Phase B

While the overall design of bpb-back roughly corresponds to the UML diagram submitted for Phase B (See Appendix E: Original UML Diagram), the design did evolve throughout Phase C.

#### a. Routes to Routers

Routes were broken out in an overly general way in the Phase B UML diagram (See Appendix E: Original UML Diagram), so Team 25 condensed the Route classes specified in the original design into two Router classes, AssignmentRouter and SubmissionRouter. These classes were made to extend AbstractRouter to initialize shared Express components.

b. Analysis Result Entries

At the  time of Phase B, Team 25 had not yet selected a specific comparison implementation, and thus had left the UML diagram blank in that area (See Appendix E: Original UML Diagram) . This area therefore needed to be filled in, and was an early focus of Phase C. When Team 25 selected java-ast and antlr4ts specifically and completed spike solution prototypes for usage of these libraries, the Team fully committed to abstracting the selected interfaces and created a more fully-featured implementation of the AnalysisResult class. Furthermore, since the Team determined that the similarity calculation would require restructuring the data model for analysis results, the AnalysisResultEntry class was created, and AnalysisResult was expanded to accommodate additional member properties required to store the comparison information required to calculate file-file similarity

c. Builders over Factories

At the time of Phase B, Team 25 had not fully completed development of either AssignmentDAO or SubmissionDAO. When implementation of database persistence required holding newly created Mongoose models somewhere, it became evident that these models should be stored on the model objects themselves and internally used to build Assignments and Submissions. This allowed each Assignment and Submission to innately have a unique database ID, among other improvements, and naturally lead to the development of the AssignmentBuilder and SubmissionBuilder respectively. This deprecated the corresponding AssignmentFactory and SubmissionFactory classes from our Phase B design.

2. How did the team settle on a final design?

When creating finalized Stories, Team 25 selected only those Stories for development which the Team thought should be considered core functionality of the tool. Given that this activity was roughly unanimous, not much settling was involved. While small components changed and evolved over the course of the semester, our overall design is fairly consistent with what was originally envisioned.

## 6. Software Engineering Practices

Team 25 took great pride in applying strict software engineering best practices while creating the BPB application. The following is a point-by-point breakdown of the ways in which these practices were applied during the development process.

### 1. Design Pattern Usage

MVC: BPB follows model-view-controller separation of concerns. bpb-front acts as the view layer, providing visualizations of data for the user. bpb-back acts as the controller and model, implementing capabilities via Router/Manager classes and modelling data via model classes and Mongoose/MongoDB.

Visitor: bpb-back uses an AnalysisResultEntryCollectorVisitor visitor to traverse individual file abstract syntax trees and create a flattened list of file subtrees for storage and later analysis.

Singleton: bpb-back uses the singleton pattern to ensure that only one AssignmentManager and one SubmissionManager may exist at any given time.

Builder: bpb-back's Assignment and Submission model classes both use Builder patterns to ensure that each model is correctly set up when initialized, since model setup is an involved process which requires the creation of database model objects.

Dependency Injection: The core Express app is passed to each Router via its constructor, which simplifies management of these dependencies.

Caching and Lazy Initialization: bpb-back utilizes multiple caches and doesn't automatically load all database objects on app reload. Requests for individual objects are performed individually and provided from the cache if possible. Repeated requests for objects are always fulfilled by the cache.

Composite: bpb-back's abstract syntax tree renderings are effectively composite data types (although they are inherited from the java-ast library), since each tree is composed of sub-structures of the same type as the tree itself.

Environment Abstraction: bpb-back and bpb-front utilize environment variable references to hold environment-specific configuration information, abstracting out environment and context-specific settings from application code.

Infrastructure-as-code: BPB's bundled vagrant environment and associated configuration (Vagrantfile, scripts/bootstrap, scripts/processes.json) define bpb's environmental configuration in a declarative fashion in versionable files, allowing the environment to be relatively portable.

## 2. Development Environment (Vagrant, scripts, Postman)

Team 25 created a Vagrant development environment to eliminate any environment-related confounding issues that might indirectly affect development. While Vagrant required some forethought to implement, testing in a VM environment quickly became critical to the team's development process, as virtualization allowed the team to effectively isolate environment-specific issues as well as to easily destroy and recreate the development environment when switching branch contexts.

When development of the environment has matured somewhat and the application was more stable, Team 25 developed a series of scripts which automated a series of routine operations (application reload, monitoring, and test execution). These scripts allowed for easier usage of the Vagrant environment and reduced overhead time spent on updating/refreshing/destroying the Vagrant environment.

When the team began integrating the front-end application (bpb-front) with the back-end application (bpb-back), the team additionally developed a Postman collection for testing the back-end application. This collection allowed for simpler reproduction of API-related issues and was also used to reproduce these issues without requiring front-end modifications or intervention.

## 3. Planning and  Issue Tracking Process

Team 25 used Atlassian Jira's Kanban board and "Kanplan" backlog functionality for Phase B and C planning activities. By the end of Phase B, the team had fully enumerated all initial requirements as Story issues with appropriate "testable" specifications (see "Testing Process" below for more details). The team continually refined the Backlog throughout Phase C to reflect current team priorities, expanded on and decomposed Stories which had grown too large, and ensured that the team's list of active Stories was continually accurate and reflected the current state of development activities.

Team 25 used Git in conjunction with Jira by naming all Git branches after Jira issues, allowing  for greater traceability of individual features and fixes and ensuring that Github Pull Requests could be easily associated with Jira issues without external integrations.

Team 25 developed a Jira workflow which mirrored the specification, version control, and code review processes. This workflow allowed the team to have strong insight into the current status of each feature in the pipeline. (See Appendix F: Team Jira Workflow and Issue Report)

## 4. Version Control Process

Team 25 enforced a strict Git workflow which required the following:

- Direct modifications to the master branch are disallowed
- All changes to the master branch are admitted only via Pull Request

- Any Pull Request to master must be a fast-forward merge (i.e., each branch must be in sync with master, have no merge conflicts, and thus replay onto master cleanly).
- Individual Story or Bug branches must be created for each Story or Bug implemented
- Each Story or Bug branch must be accompanied by appropriate test cases / coverage in order to be merged

Thus, for each discrete development issue, the team was required to:

- Pull or otherwise update the master branch
- Create a new branch named after the matching Jira issue
- Develop tests based on the documented specification on the Jira issue
- Develop implementation to satisfy the stated tests
- Create a Pull Request to request that the specified Story or Bug be merged.

## 5. Code Review Process

Team 25 required that each Story or Bug development item be reviewed prior to said item being merged into the master branch. This was achieved by requiring that a Pull Request be created for each development item.

For each item, Team 25 required that another team member who didn't participate in the active development of the item perform code review.

Code review consisted of two formal phases: code inspection and test execution. Code inspection was performed using GitHub Enterprise Pull Request comment functionality, with the reviewer requesting changes wherever required. Test execution was conducted by the reviewer in the reviewer's local development Vagrant environment. In a few cases, code walks were also required in order to explicate larger reviews, but this practice was not systematically applied to all code review activities due to lack of necessity.

In order for the reviewer to approve a given branch, the code on that branch must have been inspected and approved in a GitHub Pull Request, and must also have passed all appropriate tests when running in the reviewer's Vagrant environment.

See Appendix H: GitHub Pull Requests for a brief sample of the team's pull request naming scheme.

## 6. Testing Process and Test-Driven Development

Team 25 practiced test-driven development for developing bpb-back, requiring that each feature branch be accompanied by appropriate test cases prior to allowing said branch to be merged into master.

Team 25's specification process was conducted specifically with test-driven development in mind, as all specifications were written in such a way that they could be easily converted into Mocha / Jest specifications prior to development.

At the time of this writing, bpb-back has 100% branch and statement coverage. (See Appendix G: Test Coverage)

To test our front-end, we implemented a strict manual integration testing process, utilizing extensive checklists to confirm that operations proceeded as expected. To recreate TDD with these, we developed the checklists from the user stories we devised in Part B. (See Test.txt files in /bpb-front/src/test for checklists.)

We conducted these manual tests for each component as they were developed, with periodic full-suite E2E tests that increased in frequency towards the end of the project. We were unable to implement effective, non-breaking unit tests with our Redux architecture.

## 7. Appendices and References

### A: Vagrant Configuration

When using the Vagrant deployment, the following *.sh scripts can be executed from the repository directory (e.g. sh scripts/reload.sh while in bpb/)

- Run scripts/reload.sh to resync files and restart app components in Vagrant
- Run scripts/test.sh to execute all tests in the Vagrant environment
- Run scripts/test_back.sh to run back-end tests only
- Run scripts/test_front.sh to run front-end tests only
- Run scripts/test_integration.sh to run integration tests
- Run scripts/test_all.sh to run all tests
- Run scripts/monitor.sh to monitor running apps and view log

### B: Required Environment Variables

*APIPORT*

Specifies the port that bpb-back will serve API requests on. This value must match the port specified in REACT_APP_BPB_SRVADDR (see below)

Example Value: 8080

*DBCONNECTIONSTRING*

Specifies the location of the bpb MongoDB database

Example Value: "mongodb://127.0.0.1:27017/bpb"

*MAXFILEUPLOADSIZE*

Indicates the maximum allowable size for a single submission file upload (in bytes)

Example Value: 5000000

*COMPARISONTHRESHOLD*

Determines similarity sensitivity for individual subtree element comparisons

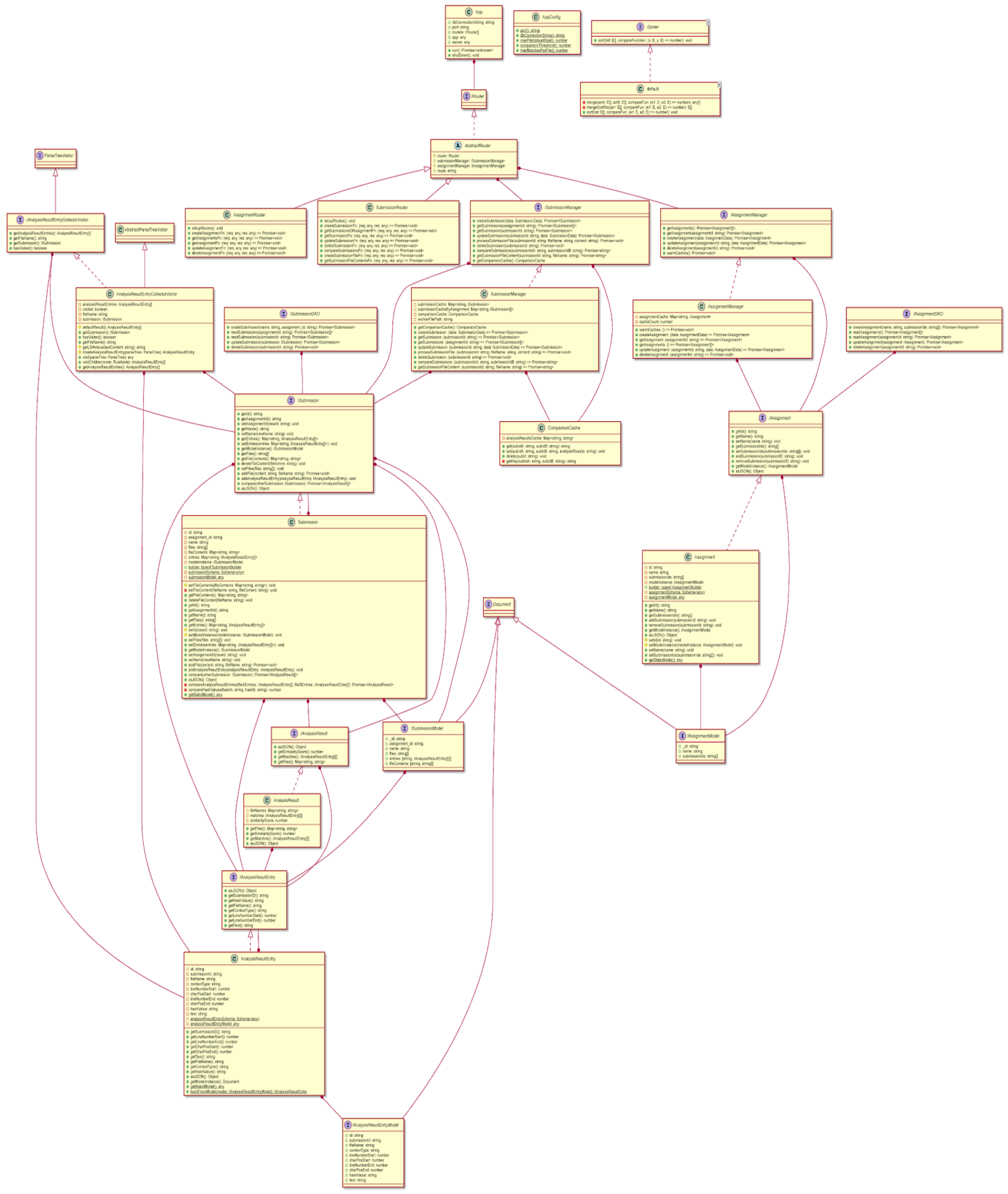
Example Value: 120 (default)

*REACT_APP_BPB_SRVADDR*

Determines location of the back-end server. Must point to where the back-end is hosted.

Example Value (when running the app components on localhost): http://127.0.0.1:8080/

## C: UML Diagram



*See /docs/ in repo for full res .svg/.png image*     *Generated with tplant (https://github.com/bafolts/tplant)*

D: Analysis Citations

*DECKARD: Scalable and Accurate Tree-based Detection of Code Clones ∗ Lingxiao Jiang Ghassan Misherghi Zhendong Su*

L. Jiang, G. Misherghi, Z. Su and S. Glondu, "DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones," 29th International Conference on Software Engineering (ICSE'07), Minneapolis, MN, 2007, pp. 96-105, doi: 10.1109/ICSE.2007.30.

IEEE Document Explorer: https://ieeexplore.ieee.org/document/4222572

*TLSH - A Locality Sensitive Hash Jonathan Oliver, Chun Cheng and Yanggui Chen*

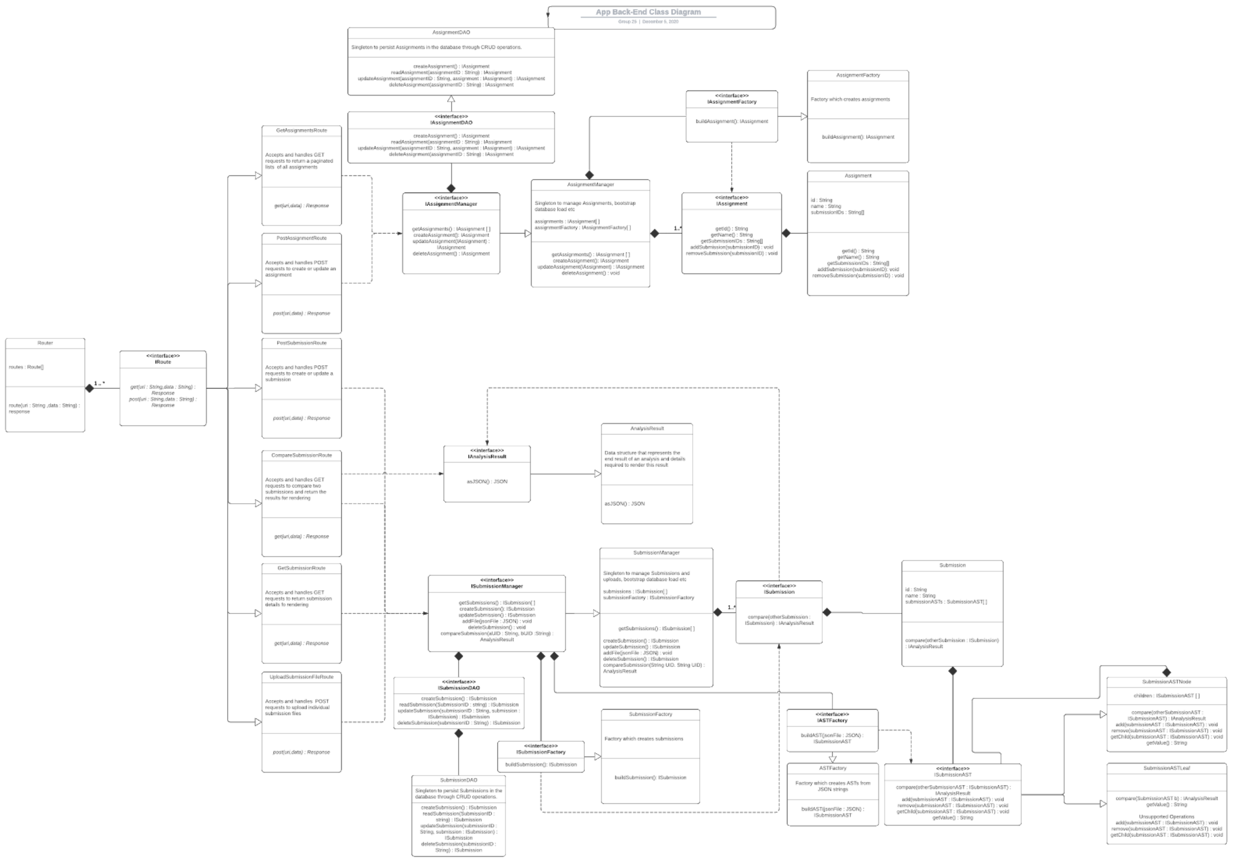
J. Oliver, C. Cheng and Y. Chen, "TLSH -- A Locality Sensitive Hash," 2013 Fourth Cybercrime and Trustworthy Computing Workshop, Sydney NSW, 2013, pp. 7-13, doi: 10.1109/CTC.2013.9.

IEEE Document Explorer: https://ieeexplore.ieee.org/document/6754635

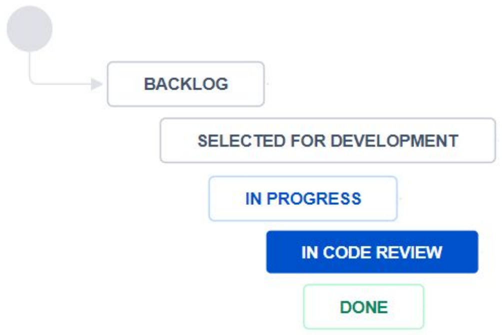
Repo: https://github.com/trendmicro/tlsh

Edits made for BPB implementation: https://github.com/turecarlson/tlsh/commits/master/js_ext

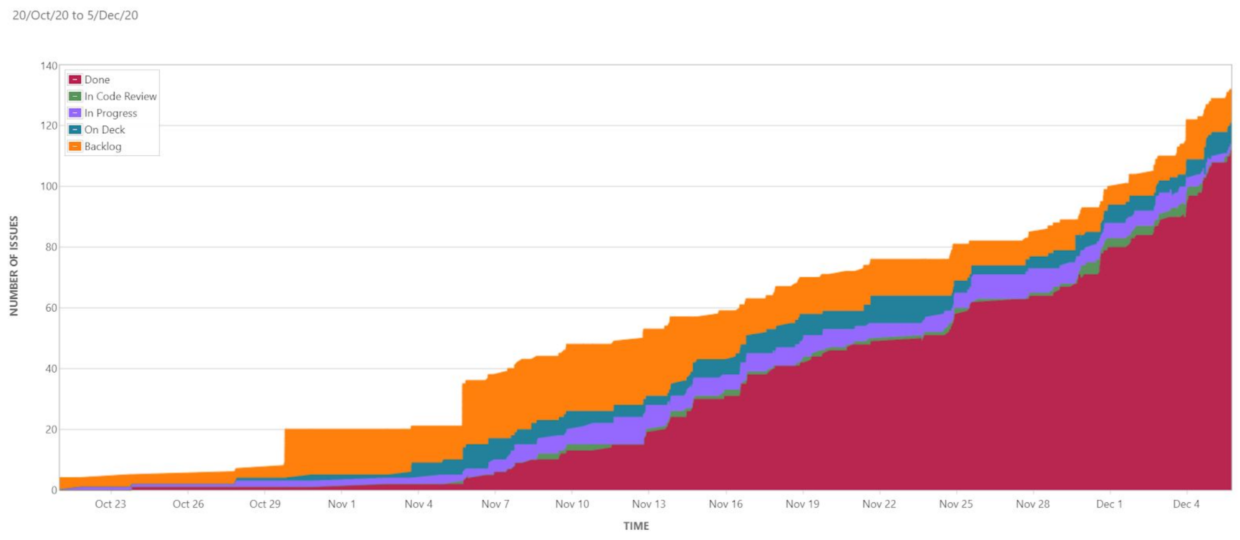
## E: Original UML Diagram (Phase B)


App Back-End Class Diagram
AssignmentDAO
<<interface>> IAssignmentDAO
<<interface>> IAssignmentFactory
AssignmentFactory
GetAssignmentsRoute
<<interface>> IAssignmentManager
AssignmentManager
<<interface>> IAssignment
Assignment
PostAssignmentRoute
Router
<<interface>> IRoute
PostSubmissionRoute
<<interface>> IAnalysisResult
AnalysisResult
CompareSubmissionRoute
SubmissionManager
<<interface>> ISubmissionManager
<<interface>> ISubmission
Submission
GetSubmissionRoute
<<interface>> ISubmissionDAO
SubmissionASTNode
UploadSubmissionFileRoute
<<interface>> ISubmissionFactory
SubmissionFactory
<<interface>> IASTFactory
ASTFactory
<<interface>> ISubmissionAST
SubmissionASTLeaf
SubmissionDAO
1..*

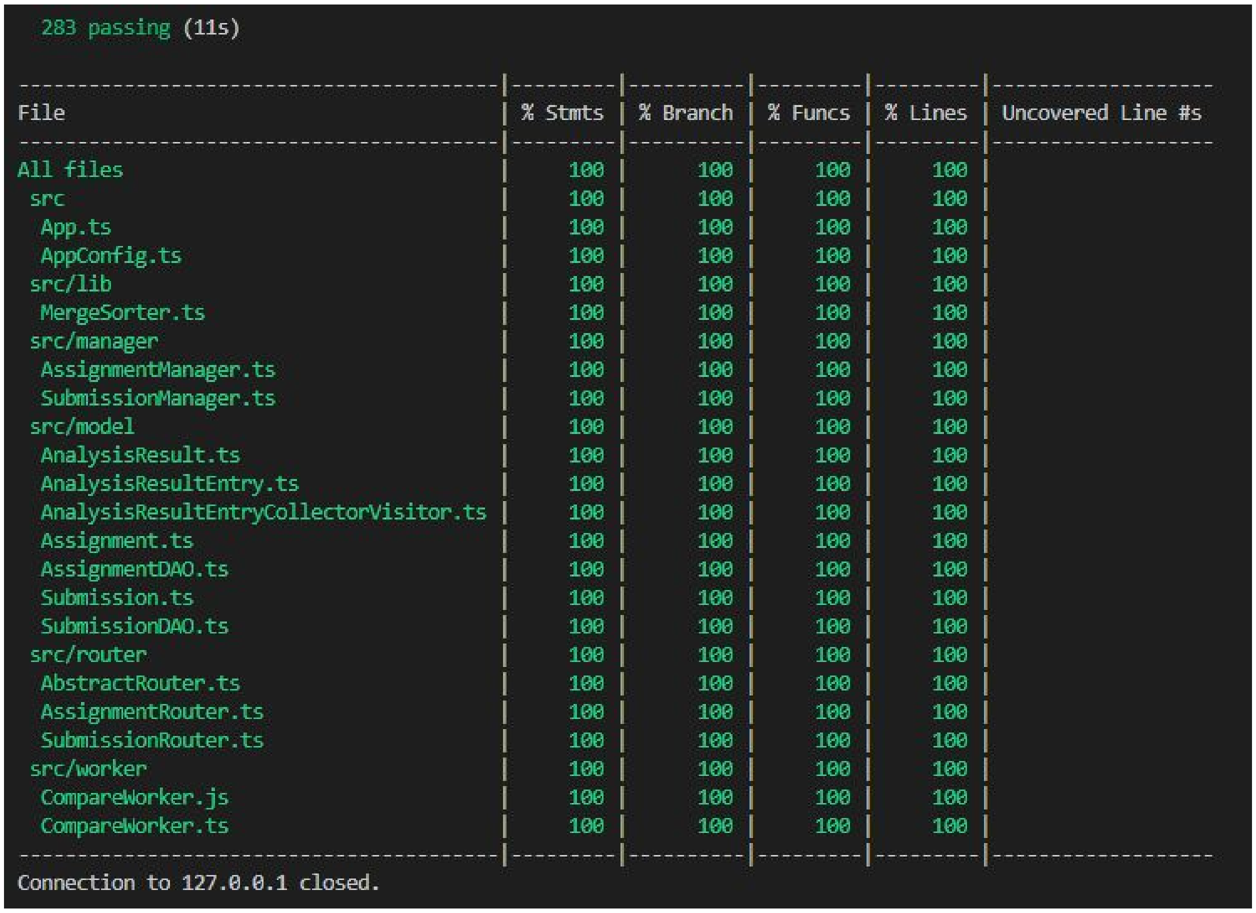
## F. Team Jira Workflow and Issue Report



Jira Issue Report (Dec. 6)



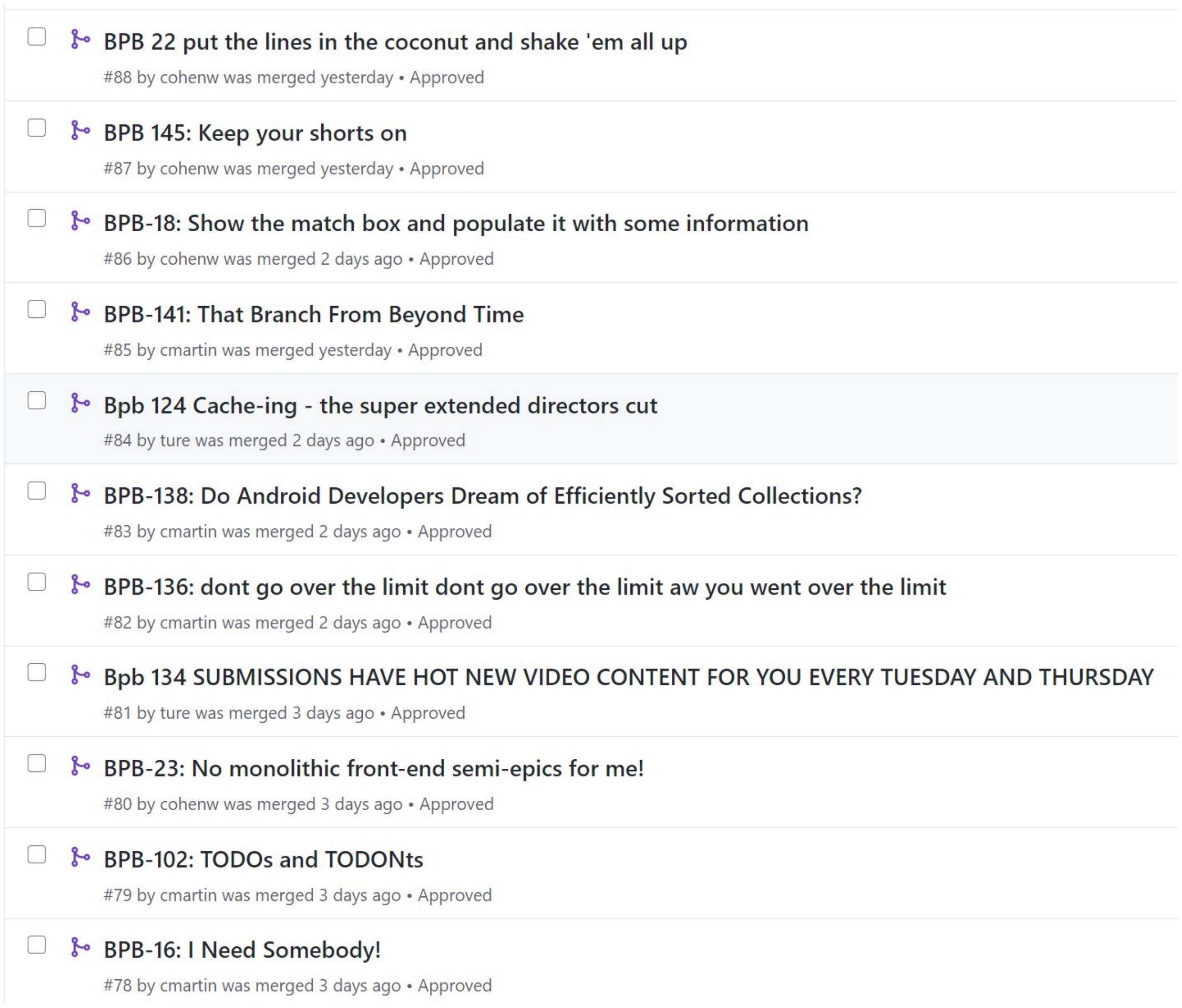
G. Test Coverage



```
  283 passing (11s)

-----------------------------------------|---------|----------|---------|---------|-------------------
File                                     | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------------------------------------|---------|----------|---------|---------|-------------------
All files                                |     100 |      100 |     100 |     100 |
 src                                     |     100 |      100 |     100 |     100 |
  App.ts                                 |     100 |      100 |     100 |     100 |
  AppConfig.ts                           |     100 |      100 |     100 |     100 |
 src/lib                                 |     100 |      100 |     100 |     100 |
  MergeSorter.ts                         |     100 |      100 |     100 |     100 |
 src/manager                             |     100 |      100 |     100 |     100 |
  AssignmentManager.ts                   |     100 |      100 |     100 |     100 |
  SubmissionManager.ts                   |     100 |      100 |     100 |     100 |
 src/model                               |     100 |      100 |     100 |     100 |
  AnalysisResult.ts                      |     100 |      100 |     100 |     100 |
  AnalysisResultEntry.ts                 |     100 |      100 |     100 |     100 |
  AnalysisResultEntryCollectorVisitor.ts |     100 |      100 |     100 |     100 |
  Assignment.ts                          |     100 |      100 |     100 |     100 |
  AssignmentDAO.ts                       |     100 |      100 |     100 |     100 |
  Submission.ts                          |     100 |      100 |     100 |     100 |
  SubmissionDAO.ts                       |     100 |      100 |     100 |     100 |
 src/router                              |     100 |      100 |     100 |     100 |
  AbstractRouter.ts                      |     100 |      100 |     100 |     100 |
  AssignmentRouter.ts                    |     100 |      100 |     100 |     100 |
  SubmissionRouter.ts                    |     100 |      100 |     100 |     100 |
 src/worker                              |     100 |      100 |     100 |     100 |
  CompareWorker.js                       |     100 |      100 |     100 |     100 |
  CompareWorker.ts                       |     100 |      100 |     100 |     100 |
-----------------------------------------|---------|----------|---------|---------|-------------------
Connection to 127.0.0.1 closed.
```

H: GitHub Pull Request Examples



BPB 22 put the lines in the coconut and shake 'em all up
#88 by cohenw was merged yesterday • Approved

BPB 145: Keep your shorts on
#87 by cohenw was merged yesterday • Approved

BPB-18: Show the match box and populate it with some information
#86 by cohenw was merged 2 days ago • Approved

BPB-141: That Branch From Beyond Time
#85 by cmartin was merged yesterday • Approved

Bpb 124 Cache-ing - the super extended directors cut
#84 by ture was merged 2 days ago • Approved

BPB-138: Do Android Developers Dream of Efficiently Sorted Collections?
#83 by cmartin was merged 2 days ago • Approved

BPB-136: dont go over the limit dont go over the limit aw you went over the limit
#82 by cmartin was merged 2 days ago • Approved

Bpb 134 SUBMISSIONS HAVE HOT NEW VIDEO CONTENT FOR YOU EVERY TUESDAY AND THURSDAY
#81 by ture was merged 3 days ago • Approved

BPB-23: No monolithic front-end semi-epics for me!
#80 by cohenw was merged 3 days ago • Approved

BPB-102: TODOs and TODONts
#79 by cmartin was merged 3 days ago • Approved

BPB-16: I Need Somebody!
#78 by cmartin was merged 3 days ago • Approved