Spring 2017

Team 6

Version 1.4

Architecture Document

Code Plagiarism Detector

Table of Contents

[Change History 2](#_Toc496481286)

[Introduction 3](#_Toc496481287)

[Design Goals 3](#_Toc496481288)

[High Level Hierarchy 4](#_Toc496481289)

[Hierarchy Description 4](#_Toc496481290)

[Hierarchy Diagram 4](#_Toc496481291)

[Component Classification 5](#_Toc496481292)

[Presentation Tier 5](#_Toc496481293)

[Logic Tier 5](#_Toc496481294)

[Data Tier 6](#_Toc496481295)

[Process View 7](#_Toc496481296)

[Process View Description 7](#_Toc496481297)

[Application Thread 7](#_Toc496481298)

[Login Validation Thread 7](#_Toc496481299)

[Upload/Parse Thread 7](#_Toc496481300)

[Report Thread 7](#_Toc496481301)

# Change History

**Version**: <1.4>

**Modifier**: <Josh Gregory>

**Date**: 10/23/2017

**Description of Change**: Stopped Microsoft Word form restarting page numbers after break points.

**Version**: <1.3>

**Modifier**: <Jonathan Gregory>

**Date**: 10/22/2017

**Description of Change**: Change logs, and introduction.

**Version**: <1.2>

**Modifier**: <Josh Gregory>

**Date**: 10/20/2017

**Description of Change**: Component classification.

**Version**: <1.1>

**Modifier**: <Jonathan Gregory>

**Date**: 10/20/2017

**Description of Change**: Added the process views.

**Version**: <1.0>

**Modifier**: <Josh Gregory>

**Date**: 10/18/2017

**Description of Change**: Initial rough draft containing the title page, table of contents, design goals, high level hierarchy, and component classification.

# Introduction

This document describes the architecture and design for the Code Plagiarism Detector (CPD) application being developed for the University of Missouri—Kansas City (UMKC). The system will permit users to login with their GitHub account and upload files to compare for plagiarism. It will generate a report comparing the two uploaded files. The user may choose to review past comparison reports once generated.

The document contains an overall view of the system hierarchy, logical views of the system components, and a process view of the system’s communication.

# Design Goals

There is no absolute measure for distinguishing between good and bad design. The value of a design depends on stakeholder priorities. For example, depending on the circumstances, an efficient design might be better than a maintainable one, or vise versa. Therefore, before presenting a design it is good practice to state the design priorities. The design that is offered will be judged according to how well it satisfies the stated priorities.

For non-functional requirements, see the requirements document.

The design priorities for the Code Plagiarism Detector application are:

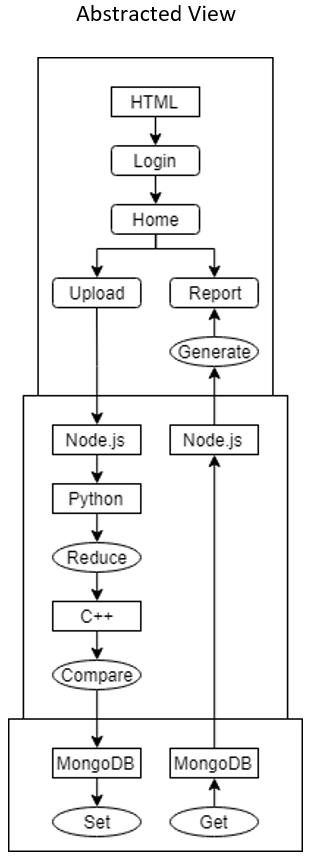
* The design should minimize effort above all else.
* The overall design should minimize complexity when possible.
* The system should not require extensive maintenance.
* The individual subsystems should be easy to integrate with the system as a whole.

# High Level Hierarchy

## Hierarchy Description

The architecture system for the CPD application is a 3-tier architecture. This architecture system is designed to allow for proper information hiding, modular components, and single system dependencies.

## Hierarchy Diagram

3-Tier Architecture

# Component Classification

## Presentation Tier

**Purpose**: The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

**Specific Nature**: This layer is responsible for presenting the login, home, upload file, and comparison reports pages. It is also responsible for running the corresponding code when the user clicks on a button. This layer will also be in charge of the making new threads.

**Login**: The login page will allow for users to login with a predetermined username and no password. It will then bring the user to the home page.

**Home**: The home page will have the option to either upload 2 files to be compared or view past comparisons.

**Upload** **File**: the upload file page will let the user upload 2 files to be compared, and then initiate the node.js script to begin the comparison process.

**Comparison** **reports**: This will initiate the node.js script to return a list of all past comparison reports. When an item in the list is selected, it will initiate the node.js script to show the pre-computed results from the code similarity algorithm, and display the lines of code that are deemed similar.

## Logic Tier

**Purpose**: This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

**Specific Nature:** This layer will be used to reduce the input codes (take out refactorable and common code) and find similarities between them. It also acts as a buffer between the presentation and data tier.

**Node.js**: This will act as the intermediary between the presentation, logic, and data tier. When the user uploads files to be compared, it will first run the python code, then the C++ code, then tell MongoDB to save the results. When the user selects reports, it will get a list of past reports and display them; when the user selects a report, it will get the actual report and send it up to the presentation tier.

**Python**: This will strip the code from both files of any common code or terminology, including but not limited to anything that can be refactored, such as variable names.

**C++**: This is the algorithm that will actually compare the measure of software similarity using hashing. It will generate a new file listing where similar code can be found.

## Data Tier

**Purpose**: Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

**Specific Nature:** This is where the comparison results and actual reduced code is stored. It can only be returned when a user requests to see the comparison reports.

**MongoDB**: This will actually store the files the user uploads as well as the comparison report generated from the C++ measure of software similarity algorithm.

# Process View

## Process View Description

The Process View is used to better illustrate the individual components and how it communicates with the rest of the application. The dataflow and storage can be optimized by ensuring thread safety through clear distinctions in the communication paths.

## Application Thread

This is the main application thread that is created at the initiation of the application. This is created by node.js. This thread handles the program logic between the user interactions of login, uploading code, and generating the report and the back-end interactions that will process those requests.

## Login Validation Thread

This thread is responsible for validating the user’s credentials after they have entered them. It will either hand off the user to the next part of the application if successful or return the user the login to try again if it is not successful. This thread will run as long as the user is signed into the application.

## Upload/Parse Thread

This is a user created thread that originates when the user uploads their code. It is responsible for delivering the users language input selection along with the code to parser and finally to the comparison analyzer storing the line by line results in the database. It finishes after it has confirmation of the data being stored.

## Report Thread

This is a user created thread that starts when the user requests a report on the similarities of the current or past uploaded code. It takes the users input and retrieves stored results of the comparison report. It is responsible for handling the report and delivering it to the user as a download. It last until the user selects a new report to generate, upload new code, or logs out.