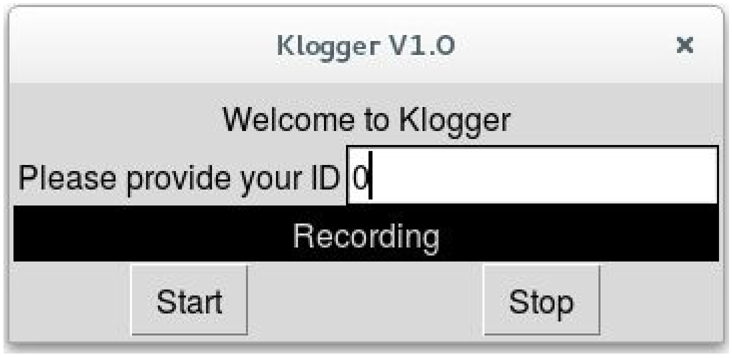
Student Engagement Monitoring Using Keyloggers

Author: Alasdair Smith (14061121)
Supervisor: Robert Cherry



Declaration

No part of this project has been submitted in support of an application for any other degree or qualification at this or any other institute of learning. Apart from those parts of the project containing citations to the work of others, this project is my own unaided work.

Signed: A. Smith

Abstract

Students at all levels in the academic system use USB pen drives to carry digital documents and other files with them throughout their day. USB pen drives have become a regular item in any student's everyday kit. This study looks to design a software solution installable onto a USB pen drive which will track the engagement of the student with their work.

It is common for academic establishments to have reduced insight into the working habits of their student populace. Often this can result in students feeling unmanaged and disconnected with their work, while academic staff are often left to consider how best they can support their students when they have have little insight into the root cause of the problem. Monitoring students' behaviour when they are working would provide valuable insight into the ways students approach work which may not be well supported by the current establishment.

This study demonstrates the use of keylogging software to monitor working behaviour of students while learning. The software shows promising potential for inferring the level of engagement shown by students whilst working with additional application of this metric towards the detection of plagiaristic behaviour. The resultant product of this study should be considered a working prototype. A scalable, commercial product would require considerably more development resources than were available to the researcher. Despite its prototypal design this study offers a working solution which demonstrates the potential capability that a highly developed version of this software might have on the academic ecosystem.

List Of Figures

List Of Tables

Abbreviations

AI: Artificial Intelligence.
DAO: Data Accessor Object.
GUI: Graphical User Interface.
HTTP: Hypertext Transfer Protocol.
HTTPS: Hypertext Transfer Protocol over Transport Layer Security.
JSON: JavaScript Object Notation.
KNN: *k*-Nearest Neighbours.
MITM: Man In The Middle.
ML: Machine Learning.
MVC: Model View Controller.
NDB: Next Data Base.
SDK: Software Development Kit.
SVM: Support Vector Machine.
SVN: Apache Subversion (a.k.a Subversion).
UI: User Interface.
URL: Uniform Resource Locator.
USB: Universal Serial Bus.
XML: Extensible Markup Language.

Introduction

Academic institutions frequently seek new ways to enable their students to engage with and retain the academic work they present for their courses and programs. The recent rise in use of technology in the academic ecosystem has resulted in a significant proportion of students having access to their course content at any time and in any location. Dissemination of content in this way offers more flexibility and efficiency for both staff and pupils alike. With this shift towards the use of computerised solutions comes a new set of challenges for both parties. Both parties can now operate remotely with less face to face interaction necessary. Thus the ability to monitor the progress and efforts of pupils has become a contemporary challenge. In previous times, when the majority of teaching would be under the watchful eye of a course tutor, there was an intuitive and contemporaneous knowledge of the state of the students' progress. Modern technology (while providing massive improvements to accessibility of content) has greatly reduced the inter-personal connection that was previously taken for granted.

This study looks to implement a solution to mitigate those problems stemming from the lack of interpersonal communication between academic staff and their students. A considerable push has been observed in the commercial world to make use of user-facing technology to return information about the habits of prospective clients. Information about their user populace has been used to great effect to improve customer experience. It is with this model in mind that this study aims to provide a solution which may have similar applications in the world of academia.

Staff often look for new ways to present the content of their classes in a format which better suits their pupils. Today's students will seek out electronic documents as sources of information in preference to going to a library to identify the same information in book form. This observation will be supported by anyone who has experienced the academic community within the last five years.

Nowadays technology is undoubtedly used by every student enrolled in the academic system, regardless of their course. With such a ubiquitous use of technology throughout the academic ecosystem, it would seem logical to implement a method of gathering the mass of potentially harvestable data about students' working behaviour. This could be implemented in a manner currently utilised in the ecommerce domain, as already mentioned.

Computerised solutions used in e-commerce commonly make use of tracking software to observe their user's behaviour. The data generated from this software is stored by the service provider and later analysed to see if the behaviour shows any areas where their service can be improved. The proposed solution described in this study looks to implement a similar method of collecting details of student behaviour by means of keylogging software. The logs generated from this software would contain values for metrics capable of describing the working behaviour of the student populace. These logs would then be collected, stored and analysed by the central server for later interpretation by academic staff.

Unfortunately the use of technology to disseminate academic material has also provided a means for acts of plagiarism. Internet accessibility has now made sources of information globally available which were previously localised to their respective immediate regions. For example, today a pupil in the United States can instantly access and read journals and research articles from academics in India.

This immediate international availability of content has brought about an increase in plagiarism in the academic system.

Previous digital efforts to identify the presence of plagiarism, have sought to compare submitted work with published research for commonalities. Works which show a large degree of similarity to already published studies are then flagged for further investigation. When these systems were tasked with the cross-comparison of sources within the immediately accessible environment of the resident students, they were functional. However the degree of modern accessibility of information has overwhelmed these systems with potential sources, and they can no-longer handle the volume of data to be analysed for cross-comparison.

This study looks to implement a means of monitoring students' engagement with their work through the use of data recorded using a keylogger. It is suggested that with detailed data representing the student's interactions with their keyboard and mouse, observations may be made which demonstrate the working behaviour of an individual. With this information, machine learning algorithms will then be applied to identify the level of engagement a student exhibits in the form of a percentage score. This study develops three software components which make up the final solution. A client with which the data is recorded (the keylogger). A server where the data is stored, analysed and interfaced with by staff. Finally, a machine learning model demonstrating the use of the recorded data to assign an engagement score. As an aside, this study demonstrates the potential for metrics collected for monitoring of student engagement to be applied to the problem of plagiarism detection.

Within the following pages of this document is a detailed exploration of previous research into the problem areas, a description of the proposed design for the software solution and a description of the steps taken to implement the design. The latter part of this document will contain an evaluation of the resultant solution and conclude with closing comment about observation made throughout the course of the study.

Below is a brief summary of the goals of the study:

- To explore existing research pertaining to problem areas with the specific aim of identifying any currently implemented technologies. As well as to identify the metrics used by these solutions and extrapolate any potential pitfalls that could be encountered in this study.
- Propose a software solution which addresses the problem domain.
- Design a software product which meets the determined functional requirements of the proposed solution.
- Implement the software product using relevant technologies and document the process of implementation.
- Test the solution to identify any potential improvements which could be made to future versions of the software.
- Evaluate the final product and discuss the solution.

Literature Review

Before opening the greater discussion pertaining to this literature review, a brief explanation of its purpose will be outlined. As part of forming the basis of a research paper, it is necessary for any researcher to conduct a review of existing studies in the field of interest. As such the content of this literature review relates to the use of, learning analytics, keyloggers, plagiarism and the various techniques employed to detect plagiarism.

There are three primary terms which need to be defined. From these key terms further additional terms can be derived.

Learning analytics is the process of collecting, measuring and analysing various measurements about learners and the context in which they learn. Using the insights gained from these analyses there is the potential to derive/develop actions or interventions which may be taken to improve the learning process, identify individuals who may be struggling as well as a whole host of other actions . Although this area of research is still in its relative infancy and little standardisation exists across researchers, the following review aims to identify key metrics which may be employed as a means to measure learners behaviour.

A keylogger or keylogging tool is defined as a piece of software used to record the keyboard input of a user. Typically this input, if from a keyboard, is termed as the keystrokes. For each key pressed, a record is made of this event, often in combination with additional details such as a timestamp or reference to the file/application to which the input was directed. Modern day keyloggers have advanced capabilities to record more varied types of input such as mouse pointer behaviour. Additional terms related to this might be, key-press, input, key event or similar terms. Broadly speaking these terms are synonymous in meaning.

Plagiarism is a term which is far harder to define. According to (Maurer et al., 2006) in their study entitled “Plagiarism - A Survey” the accepted definition can be found at Plagiarism.org. It defines plagiarism as “... In other words, plagiarism is an act of fraud. It involves both stealing someone else's work and lying about it afterward.”(What is Plagiarism?, n.d.). However, while this term may seem simple enough to grasp, Maurer et al. go on to further explain that not only can plagiarism be accidental or unintended but is is also possible to plagiarise one’s self. For the requirements of this review, plagiarism will be defined as seen above from Plagiarism.org

Review

Keylogging Software

The first question asked was, “What is a keylogger and how are they currently in use?”. An article entitled “Keyloggers” (Sagiroglu and Canbek, 2009) from the IEEE Technology and Society Magazine, offered a summative view of keylogging technology currently in operation in various forms. The article was a detailed description of the different classifications of the technology but still maintained a high-level perspective making it accessible to individuals with little experience in the field. Further, the article offered insight into the impact the technology may have both with legitimate use and malicious use, accompanying this discussion with statistics and observations relevant at the time of publication. Most useful from this piece of research was the insight given into the way different keyloggers operate. Generally, keyloggers fall into one of two categories, hardware-based keyloggers and software-based keyloggers. Having reviewed Sagiroglu and Canbek's explanation of hardware keyloggers, it was evident this was not the best route for a solution designed to detect plagiarism. Because a plagiarism detection solution should be fully visible to the user (for ethical and moral reasons) the solution software should not be hidden, which is one of the major features of hardware-based keyloggers. Further, a software-based keylogger offers additional data from which features indicative of the user's behaviour can be drawn. Something a hardware solution could not provide as well, if at all.

Learning Analytics

Having identified a technology which captures actions of users with a fine degree of granularity, it was then questioned which metrics may be applied to the recorded data to measure aspects of a user's learning behaviour. As previously mentioned, the study of learning analytics is in it's relative infancy. The value of internet/cloud based learning environments was not fully realised until the early to mid 2000s. The vast majority of research that exists on the topic has been published within the last 10 years. Many different e-learning environments have been created since the efficacy of the approach was shown in practice. The current ubiquity of cloud based learning environments has precipitated an increasing interest as to how business style analytic measurements might be emulated in academia.

"Academic Analytics: a new tool for a new era" (Campbell et al., 2007), observed the increasing shift towards computerization of the learning process and made some interesting suggestions about the use of the technology to predict the learning outcomes of students. The study proposed a process of five steps an academic analytics engine might take:

> "Academic analytics can be thought of as an engine to make decisions or guide actions. That engine consists of five steps: capture, report, predict, act, and refine." (Campbell et al., 2007)

In addition to isolating the steps which should be taken in an academic analytics engine, Campbell's study further breaks down each step identifying the potential pitfalls. Of particular interest is the discussion surrounding the issues associated with the capture process. The issues of storage, granularity and retention. The study explains that real-time collection of data requires significant storage space depending on the levels of granularity with which the information is recorded. Campbell et al offer the following explanation:

> "Systems can generate raw data or summarize it. For example, CMS data might be summarized by day or week rather than individual activity. Granularity is a balance between what the system provides, what questions the institution is attempting to answer, and the storage requirements of the data." (Campbell et al., 2007)

The research conducted by (Romero-Zaldivar et al., 2012) entitled “Monitoring student progress using virtual appliances: A case study” showed how a system could be implemented following the linear process as suggested by (Campbell et al., 2007). The system created by Romero-Zaldivar made use of virtual machines to create an environment inside of which, students would conduct their work while metrics could be calculated. Using virtual machines in this study, provided the added benefit of isolating the relevant information by restricting the software available. Romero-Zaldivar identified the trade-off between coverage and generality. Specifically, the desire for generality of the data collected can often have the impact of collecting vast swathes of data not specific to the learning process. Conversely, a highly specialized system collecting only specific details may not capture enough data to represent the context in it’s entirety, instead only capturing a snapshot of a narrow aspect of the learning process. Romero-Zaldivar used virtual machines to specifically create an environment which was neither too specific nor too general. By restricting the users’ access to only relevant software in the context of a more general environment, the system was able to collect relevant data with a more appropriate balance between generality and specificity.

As expressed by (Campbell et al., 2007) part of the data mining process involves reporting and prediction. (Larose, 2005) presented various machine learning algorithms which would be suitable for data mining as part of the learning analytics process. In particular, algorithms were identified which would be suitable for the process of classification, regression and segmentation. Decision Trees, Support Vector Machines and the Apriori algorithm were found suitable for the task.

Plagiarism Detection in Academia

The article entitled “Plagiarism - A Survey” (Maurer et al., 2006) provides a good top down perspective of the academic study of plagiarism and the effects on academia itself. Of interest was its extensive summary of existing plagiarism detection techniques and tools. Although the key-terms and definitions addressed in the article's opening paragraphs allowed for greater understanding of the subject domain, it was the assessment and evaluation of existing plagiarism detection methods which were suitable for the formation of a well-defined research proposal. The message taken from this article was that methods used by current automated plagiarism detection tools utilise word matching and text comparison. Overall, the tone of the article suggests that this current, widely adopted method, is indeed fallible for many forms of plagiarism that exist beyond the simple copy-and-paste level. Further, the article goes on to suggest that stylometric analysis (more akin to the research intended by the reader) is of greater effectiveness, however, it is still considered experimental. The implication is that these more experimental techniques are applicable to small tasks and infers that it is less scalable than existing cross-referencing based techniques.

The observations above present two major goals for future research. Firstly, to implement a method of plagiarism detection that makes better use of stylometric assessment techniques. Secondly, to be able to apply these more experimental technologies in a scalable way.

Finally, the article discusses more experimental techniques, in particular, the semantic equivalence of two documents, as a method for plagiarism detection. Quoted was a statement from another piece of work:

“To actually prove that two pieces of text are semantically equivalent one would require a complete understanding of natural language, something still quite elusive. However, we can consider a compromise: rather than allowing a full natural language we restrict our attention to a simplified grammar and to a particular domain for which an ontology (semantic network) is developed. Clearly, sufficiently restricting syntactic possibilities and terms to be used will allow one to actually prove the equivalence of pieces of text.”(Heinrich and Maurer, 2000).

Perhaps this is an indication as to the success of earlier research documented in a journal article titled “Computer Algorithms for Plagiarism Detection” (Parker and Hamblen, 1989)? Parker and Hamblen conducted an exploratory review of methods used at the time of publication which detected plagiarism in code text (e.g. C code, Pascal and FORTRAN programming languages). This study highlighted relevant metrics in algorithms used for the detection of plagiarism in program code and demonstrated the extraction of features for which these metrics could be calculated. Unfortunately, this paper predominantly discusses the detection of plagiarism in program code and not necessarily in submissions in essay form using natural language.

The study by Parker and Hamblen was attempting to detect plagiarism in user-written computer program code. Some of the approaches made use of compiler and interpreter technology to understand the code. This is tantamount to the semantic understanding of human, natural language. The primary difference is that it is extremely hard to apply concrete rule for natural language. It shows that, if a computerised system is capable of understanding the text it is parsing (in comparison to only picking up symbols and matching them with other symbols without any real understanding) then it would be entirely possible to make use of such a system to detect plagiarism with a high level of efficacy provided such rules could be established.

Machine Learning for Plagiarism Detection

Machine Learning (ML) as a subcategory of Artificial Intelligence (AI), is the study of designing computerised systems capable of learning from an input and inferring an observation without strict programming from a human being. The ability to make an intelligent observation is an essential feature for accurate plagiarism detection. Thus it was the goal of the reader to inquire about research which shows a use of ML in the inference of human behaviour.

Given that the identification of a user from biometric input requires some intuition by the computerised element, it was of interest to the reader to discover how this could be implemented with regards to keyloggers combined with ML. A journal article titled “Keylogger Keystroke Biometric System” (Tschinkel et al., 2011) offers an insightful overview of how biometric systems operate. The article lends the suggestion of implementing a data collection, conversion, feature extraction and classification process flow. Tschinkel highlights the need for the extraction of informative features from recorded data in order to classify an individual as deceitful or not. At every point in the suggested process flow outlined in the article, relevant and information-rich features are paramount to the success of classification. Without relevant and information-rich features taken at the initial data collection stage, the following conversion, analysis and comparison of the data will be irrelevant and achieving the goal of authentication becomes significantly reduced.

This invites the observation that the ability to create and implement a plagiarism detection method which uses biometric-like identification of a deceptive user is dependent, almost entirely, on the ability of the solution to collect and filter relevant metrics. This, in turn, raises the question of if such metrics even exist and, if they do, are they accessible through the user's keyboard and mouse interface alone? Would such a solution instead require a visual recording of the user operating the machine to detect deceptive behaviour? If so, then is that not the same as standing over the user's shoulder while they work to ensure no fraudulent activity takes place?

Research which may provide insight into an appropriate solution is the paper entitled "Web-based keystroke dynamics identity verification using neural network." (Cho et al., 2000). Despite the research having taken place roughly ten years before that of (Tschinkel et al., 2011) Cho et al. demonstrate that a method does exist which can reliably authenticate a legitimate user using ML without the use of recorded images. Cho et al. indicate that while more typical biometric systems using k-nearest neighbours algorithms have been extremely useful in the classification of authenticated users, more recent application of the k-nearest neighbour algorithm using keystrokes (as opposed to retinal data, or fingerprint data) to identify users by their typing patterns have been, practically speaking, unsuccessful. However, the paper goes on to suggest that through the use of MLP (Multilayer Perceptron, a method of ML utilising Neural Networks), the accuracy of detection is improved.

This research, implementing the idea of using the biometrics of keystrokes and typing behaviour to detect deceitful users, moves the possibility of using ML to detect plagiarism one step closer. The article indicates that it is indeed possible to use ML to infer deceit through features collected during the actual typing process rather than analysis of the finished product (i.e. the final submission text) or more typical biometric measurements (e.g. retinal data). The use of Neural Networks compared to a previous approach using the k-nearest neighbour algorithm showed a significant improvement from roughly 10% erroneous classifications to only 1%. It should be noted that, while this research did show positive improvements in its problem domain, the trial group used to test the solution was comparatively smaller than would be necessary to prove its efficacy.

Cho et al. invites the reader to believe that features attainable through a keyboard and mouse interface exist which are useable in identifying malicious users versus legitimate users. The tests conducted by Cho et al showed that even when both the imposter and the legitimate user were acutely aware of the password they had to type to achieve authentication, accurate detection of the malicious user could be made. Similarly, both an original writer and a plagiariser are fully aware what they intend to write. One seeks to copy existing works from another source, while the other will create their content fluidly from their experience and dynamic thoughts. Presuming that the findings in the study conducted by Cho et al. are accurate and scale with a larger test group, this may indicate the possibility of similar features and metrics being usable in an anti-plagiarism solution.

Finally, then, the reader sought to determine if the approach utilising ML to detect deceitful users had been applied to plagiarism. Many articles were found which implement ML to detect deceitful writing in complete text. However, few presented an approach whereby the system analyses the user's writing behaviour as it takes place. One conference paper which did present such an approach was entitled "Keystroke Patterns as Prosody in Digital Writings: A Case Study with Deceptive Reviews and Essays" (Banerjee et al., 2014). The paper suggests that a degree of mental burden occurs when someone attempts to write deceptively. As such, discrepancies can be seen between the typing/writing behaviour of an individual who is writing intuitively and one who is writing to deceive the reader. Similar to the study by Cho et al., this research made use of timing metrics taken when a user is typing as an additional feature to determine deception. Features such as time between words, the time taken to generate the entire document and average keystroke time were extracted from the data. These features were processed using Support Vector Machine classifiers (SVM) to then classify the data as deceptive or truthful.

The paper concluded that a noticeable difference could be observed between the time intervals of words and sentences of deceptive writers versus honest writers. The authors offer various psychological suggestions for the existence of such differences and provide empirical data to support their observations. Of note is the specific features extracted to detect deception or truthfulness seems to be domain specific. Deceptive writers show different characteristics when writing one type of document than another (e.g. their perspective on a news article or a review of a restaurant).

To establish a baseline for the investigation, researchers provided a SVM classifier with a feature set not including any temporal data at all. This was for the purpose of showing the difference in the percentage of correct classifications with and without the temporal aspect included. The baseline model was purely given a Bag of Words representation of the data. Even without the additional temporal information, the system had an average accuracy of 82.58% - 83.62%. This indicates that ML can indeed be useful in the classification of deceitful documents even without the use of more experimental features such as the temporal aspects of keystrokes.

Design

Solution Overview

The solution design discussed in further detail below details the functional requirements of a software solution to implement the monitoring of student engagement through the use of keylogging software. The solution is comprised of three parts: the client, the server and the machine learning component.

The client component of this design, is the software to be installed on the user's (student's) computer. It's goal is to be the mechanism by which the user's actions are recorded and monitored for later analysis.

The server component of this design, is the central location for interaction with the stored data. academic staff will interact with the server via the web application's user interface. Using this interface staff can observe measures of the student's working behaviour and request different forms of analysis to be made.

The machine learning component of this design, demonstrates the application of machine learning algorithms to assess a student's engagement with their work and to determine the likelihood that the work has been plagiarised. The machine learning algorithms are trained and tested using data collected from the client component with the ultimate goal of incorporating any successful algorithms into the design of the final web application.

Although the design of each component is considered independently of the others, the end goal for this design is to have all the components working as part of the overall solution.

Solution Functional Requirements

From a purely functional perspective the solution needs to achieve a few specific goals:

1. The solution must be capable of capturing user's input from the mouse and keyboard.
2. Captured input must be stored in a remotely accessible storage system in an organised manner.
3. The solution must be capable of handling multiple different users.
4. There must be a user interface which allows staff to monitor the users.

Taking a more detailed perspective the solution should also achieve the following:

1. User input should be categorised and tagged in a manner which allows for the separation of different users' input.
2. The storage solution used to retain all data collected should be secure to industry standards.
3. Raw recorded data (i.e. the literal input submitted by the client) should not be accessible.
4. Staff must be authenticated to industry standards when accessing the web interface.

To achieve these high-level requirements the solution will make use of carefully selected software libraries and environments which enable these features to be implemented.

Addressing items one in both lists above, the library pyhook (“JeffHoogland/pyxhook,” n.d.) will be used for the purpose of data collection on the client side. Pxyhook is capable of capturing both mouse and keyboard inputs as well as categorising which window the input is being directed to. For example, if the user types a web address into their web browser and clicks on a button to search for that resource, the library will record the input as it occurs and label its input window as the relevant browser software (e.g. Chrome, Windows IE, Firefox).

To achieve the requirements of two and three in the first list and item two in the second, the Google Cloud platform (*Google Cloud Computing, Hosting Services & APIs*, n.d.) and it’s Google App Engine (*App Engine - Build Scalable Web & Mobile Backends in Any Language*, n.d.) service neatly provide the appropriate functionality. Using Google Cloud, developers can quickly set up and run a web app in various environments. Google App Engine provides the means to generate a Python Flask (*Welcome | Flask (A Python Microframework)*, n.d.) app with which web components of the solution can be hosted remotely. Additionally, Google Cloud provides the means to store data online with fast read and write data access speeds necessary to keep up with the high solution’s high data flow.

Making use of these industry standard services will also offer the security required to ensure user’s data will be stored safely without worry of unauthorised access, addressing items four and two in the second list.

Finally, using the Flask web development framework makes the creation of web services on top of the Google Cloud environment more simplified. Flask provides a minimalist but extensible framework providing the developer with only the functionality necessary to implement their service, development of a web front-end user interface should be feasible within the solution time boundaries.

Resources Required

Below are the identified resources required to implement the solution:

- A USB flash drive with enough space to store the client program and the document to be submitted (estimated 8GB storage size).
- A cloud based service to host the web service, data storage and web front-end components. For ease of use and interoperability Google’s App Engine will be the chosen environment to host these elements.
- A fully portable and machine independent language with which to write the software. Python has been chosen as a suitable language for its compatibility across environments, growing popularity and large library code-base from which to implement the solution.
- Relevant libraries which will enable the solution to be implemented with relative ease. These include:
  - Flask, a web development framework (“Welcome | Flask (A Python Microframework),” n.d.).
  - Pyxhook the library which will be used to capture user interaction with the computer (“JeffHoogland/pyxhook,” n.d.).

- Built in JSON parsing and packaging libraries (json) included in the Python Standard Library (“18.2. json — JSON encoder and decoder — Python 2.7.13 documentation,” n.d.).
- Google Cloud SDK for integrating the Python app into the Google Cloud App Engine environment (“Google Cloud Platform/google-cloud-python,” n.d.) and (“Google Cloud SDK Documentation | Cloud SDK,” n.d.).
- Weka (*Weka 3 - Data Mining with Open Source Machine Learning Software in Java*, n.d.)
- Minitab (*Minitab Statistical Software - Minitab*, n.d.)

High-level Design Overview

Figure 1 shows a high-level overview of the entire solution design. Breaking this diagram down, it is composed of three major elements.

Firstly, as represented by the cloud shape taking up half of the diagram, is the Google Cloud environment. Inside of this environment resides the web application, web front-end and the storage component. The second major component is the App Engine Flask application (represented by the large rectangle within the cloud). This component will host most of the logic of the application and will be the central point of communication between the user's (both staff and students) and the data store.

Finally, the users themselves (as represented by the oval shapes at the top of the diagram). One oval represents the primary user (also referred to as the student) and the other the staff member with the responsibility of monitoring the student's behaviour. The key difference between the two user types is that the student will be interfacing with the solution transparently via the client application running on the USB flash drive, while the staff will interface directly with the web front-end, hosted by the Google Cloud platform.

To summarize, as shown in Figure 1, users will interface (as normal) with their machines during the writing process of their assignment. When plugging in the USB flash drive the solution will automatically submit their keystrokes, mouse clicks and other relevant data to the online web service. This data will be organised and stored for later analysis by the web app.

Independently of the client submitting data to the server, a staff member can log into the web front-end and observe multiple perspectives describing the students' behaviour. These perspectives will require the data stored on the Google Cloud storage to be processed and analysed by the App Engine Flask application and then presented to the staff member in a suitable format. For the purpose of this solution a standard MVC (Model View Controller) design pattern will be implemented to better separate the individual components of the web application.

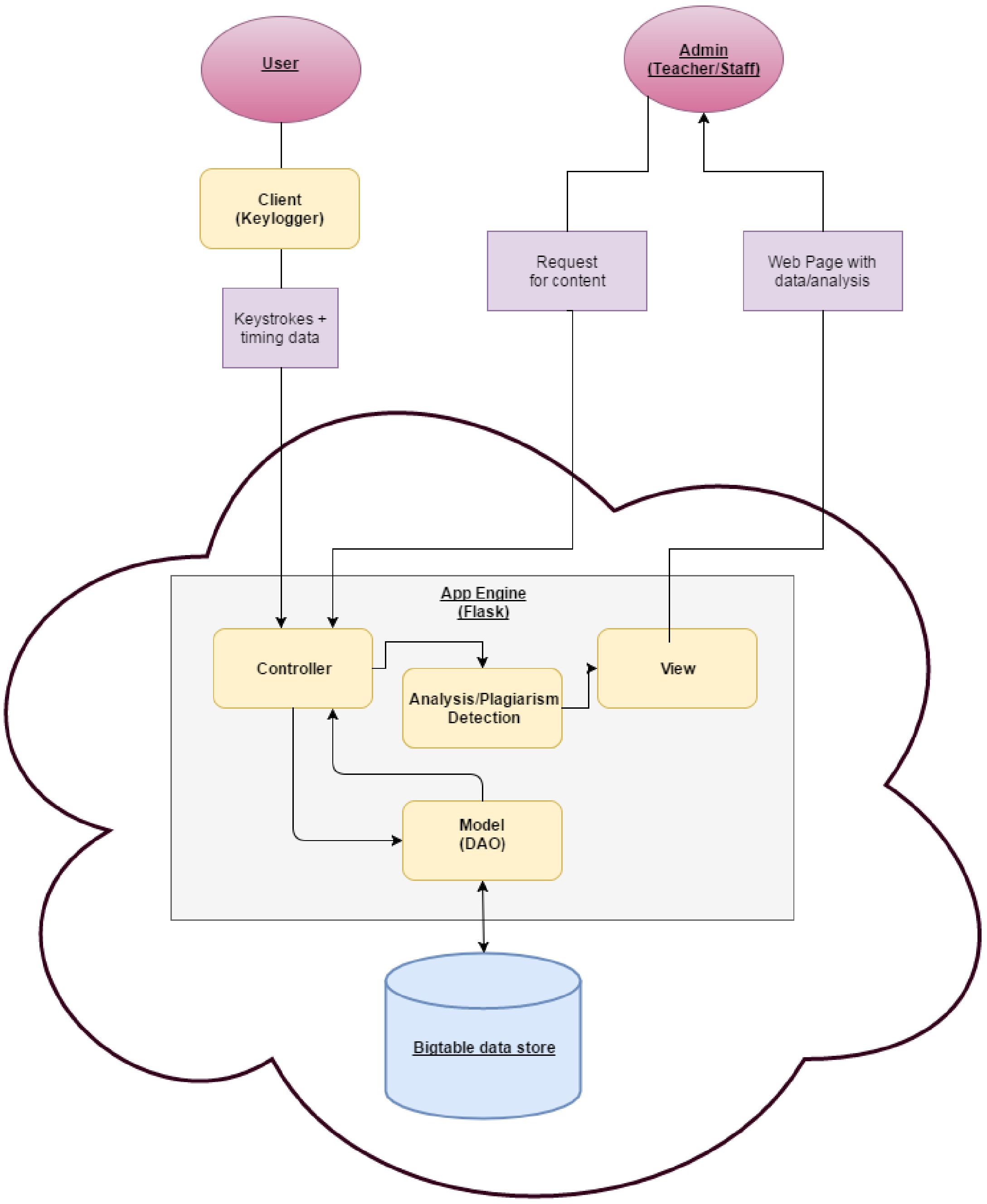


Figure 1: High-level overview.

Client Component

The client component of this system refers to the USB flash drive and its combined software, which will automate the monitoring process on the student's system. The client records the information which will be used by the rest of the solution and needs to achieve specific functional goals to enable the rest of the system to operate fluidly.

Component Functional Requirements

1. The client must be capable of collecting all keystroke and mouse input data in real time.
2. All data collected should be categorized and labelled appropriately to enable the back-end systems to achieve their goals.
3. If possible data collected should be limited to only the relevant data.
4. Data collected should be processed as much as possible before being sent to the back-end systems to relieve computational strain as much as possible. This requirement should be pursued up to the point it does not impact the real-time delivery of data.
5. The client should automatically start upon plugging in the USB drive.
6. Upon removal of the client USB flash drive the data collection process should cease to exist on the user's system.

Input Data

Data recorded by the client will be in the form of keystrokes and mouse behaviour. This collection of this data will be recorded in real-time. Using the Python library pyxhook, the software can be moved between systems without permanent installation of the client software. Further, using the window manager to receive the input allows for the data to be classified by which window/program it is being directed to. This will enable the categorisation process to have appropriately labelled data with which to classify each input. The decision to use a software level keylogger instead of hardware keylogger was informed by the article "Keyloggers" (Sagiroglu and Canbek, 2009) discussed in the Keylogging Software section of the literature review.

Design

As can be observed in Figure 2 the client component has two input types, keystrokes and mouse actions. This input, recorded by the client program stored on the USB flash drive is processed according to the functional requirements. Finally, the processed and packaged data is sent from the user's system to the web application on the Google Cloud platform (This is represented by the cloud image labelled "Web Service").

A few critical points should be noted when considering this diagram. Firstly, the addition of a user ID to all data collected (as shown on the user symbol). By giving each user an ID (either by labelling each USB drive and hard-coding it into the process or by asking the user for their academic identification number upon program execution) the system enables the storage process to differentiate between different users. Without this feature all data received by the cloud storage would be indistinguishable and thus become unusable.

Secondly, the note stating "This process begins from the moment the user plugs in the USB Drive, until the moment the user removes the drive." implies the system will discontinue to monitor activity after the USB drive has been removed. The operational status of the client software is therefore clearly manifest. This feature will be implemented using a basic "autorun" script stored on the USB drive.

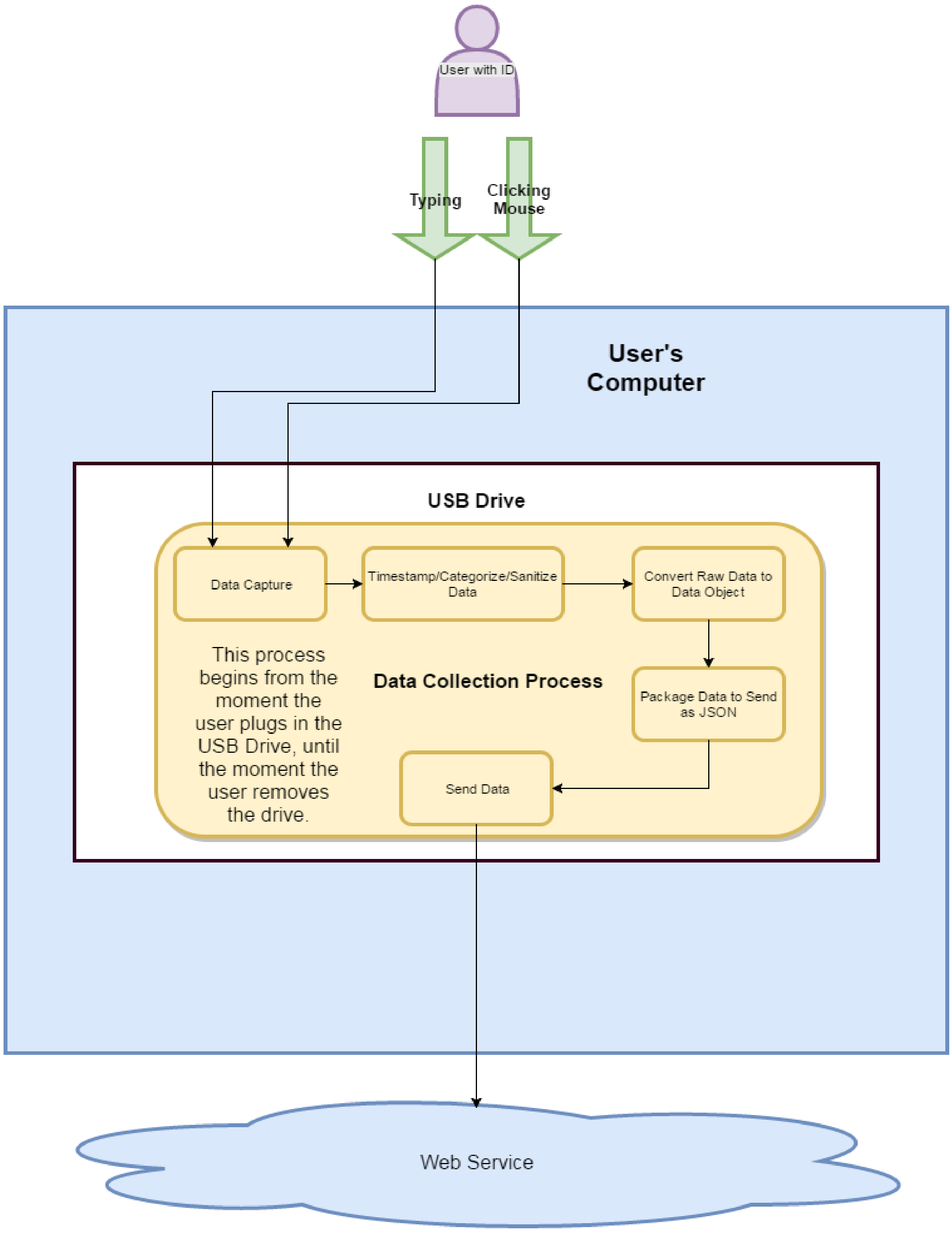

User with ID
Typing
Clicking Mouse
User's Computer
USB Drive
Data Capture
Timestamp/Categorize/Sanitize Data
Convert Raw Data to Data Object
This process begins from the moment the user plugs in the USB Drive, until the moment the user removes the drive.
Data Collection Process
Package Data to Send as JSON
Send Data
Web Service

Figure 2: Client design diagram

Output Data

During the "Data Collection Process" in Figure 2, the data is passed through multiple stages before it is sent to the web application. After the data is captured it will initially be timestamped, categorized by input type the window the input was directed to and sanitized so far as to remove unnecessary data.

From this point forward the data will be constructed into an appropriate data object. This new data object is required in order for the data to be in an organized manner to be processed by the following stages. Given the data is in an organized format, it can be encapsulated into a JSON object to be sent to the back-end components using Python's built-in JSON module. Packaged and processed, the data is then sent to the back-end components to be stored.

Central Server

The user interface for the web application should be a relatively simple layout with the necessary functionality readily accessible. It should be accessible only to staff members whose role it is to monitor student activity. However, given that it will be hosted remotely (potentially accessible to anyone with knowledge of its location) some serious consideration must be given to the security and authentication methods used to access this interface.

Component Functional Requirements

1. The web application will be hosted on the Google Cloud platform using the Flask (*Welcome | Flask (A Python Microframework)*, n.d.) web development framework.
2. The web application should only be accessible to staff members and use industry standard authentication and authorization methods as a means of implementing this feature.
3. The UI should be simple and functional in its presentation of the data.

Input Data

Data displayed by the web application UI will by default include a way to identify each user and statistics which demonstrate the behaviour of the user throughout the writing process. Data will be retrieved from the Google Cloud storage and analysed as it is requested. This analysis will be the basis from which staff can make reasonable assumptions about the user's working behaviour and as such should consist of metrics which should show the students' engagement with the work.

Design

The UI wireframe shown Figure 3 is a basic layout designed to achieve the functional requirements as explained above. At this stage it is yet unknown the exact process flow required to achieve these requirements and as such this UI layout will be further expanded upon as implementation occurs. Currently the design is purposefully sparse and simplified to allow for an expansion of scope as development takes place. It is certain that the design will be as simple as possible while still achieving the functional requirements as defined in this document. It is not the goal of this project to create visually appealing UIs and so it will be left to later iterations of the solution to address this element of the solution.

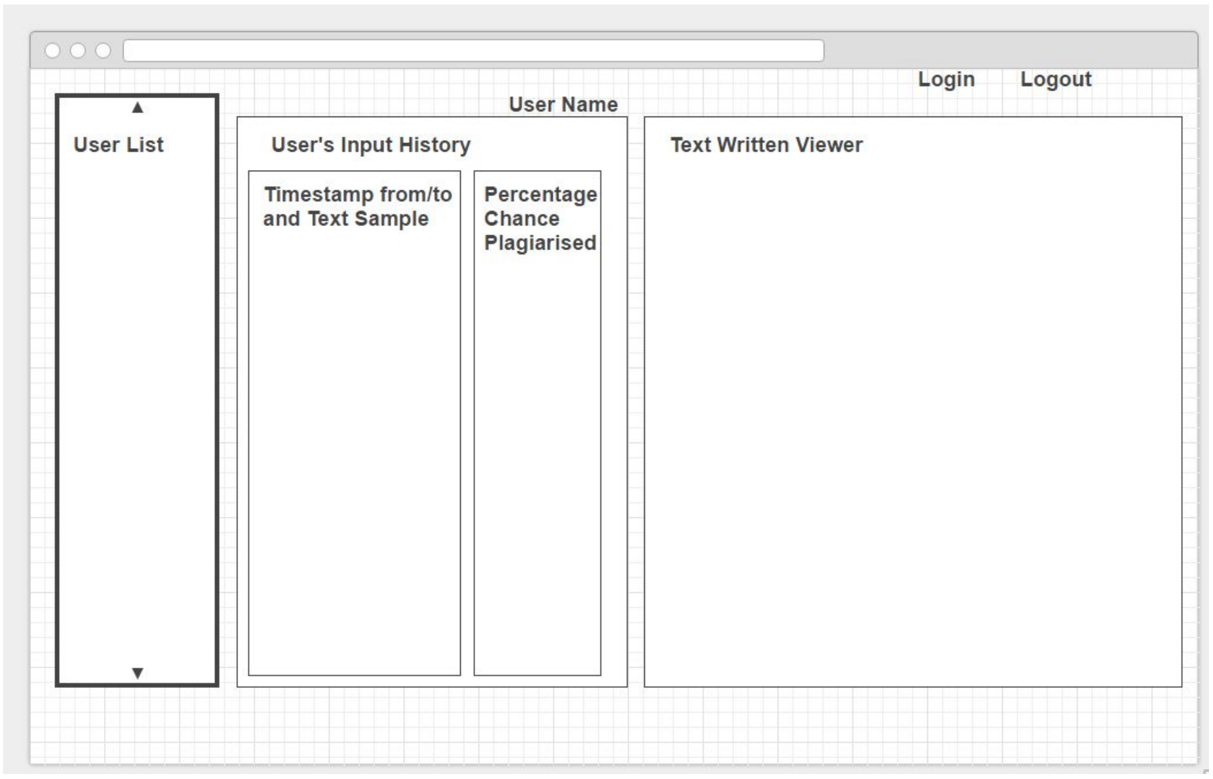


Figure 3: UI design wireframe.

From the diagram in Figure 4 showing the functional design of the web application, it can be seen that the primary objective of the central server is to act as an interface between the clients and the academic staff to the data storage offered by Google Cloud. Flask will be used as the framework for implementing the functionality of the web application. Methods will be created which handle the storing, retrieval, analysis and presentation of the data.

Flask will handle input from both the client and the users of the web application's UI. Data coming from the client will only be stored but never retrieved via the client. The transmission of data between the client and server will be one way. Data will only be fetched from the storage when requested via the UI. At no point will the data be modified after storage but it can be deleted entirely.

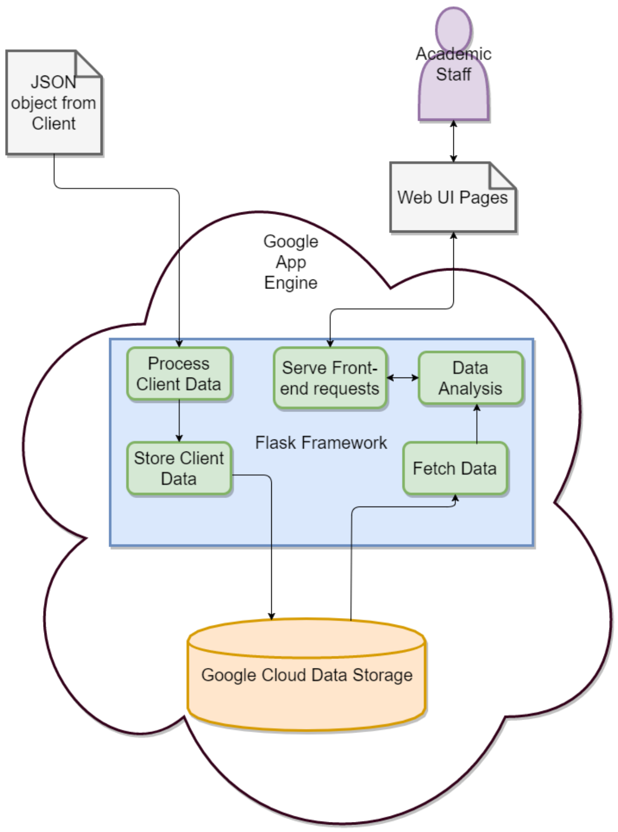


Figure 4: Web application design diagram.

Machine Learning Component

This project will include an application of Machine Learning algorithms in an attempt to identify features about the working behaviour of the student running the client. From the data recorded during student's sessions analysis will be made to determine if there are metrics which might be established to determine their level of engagement and the potential of plagiarised content.

Machine Learning and its application in data mining and detection of plagiarised content is a field which has been explored in detail by many researchers. It is clear from the review of previous studies, that one of the most difficult aspects of this part of the project will be the collection of representative data in large enough quantities to establish a reliable and accurate model. Given that, it is anticipated that any solution developed will be representative of further study in the area but not necessarily a suitable solution in its own right.

As previously stated, the collection of data which is representative will be a considerable challenge. It is expected that representative data will not be accessible using the Client Component proposed above. It is doubtful that enough data will be collected from the small sample of individuals with which the client will be tested neither will it be representative of an entire student populace. With this in mind it is the most logic option to pursue a process of generating the data. Generating the data will at the least mean that a large enough sample size can be created. Data generated based on a normal distribution would typically expect to represent a human population quite well. It is unlikely that data generated in this manner will well represent any outlier cases. Exceptional individuals and individuals who are plagiarising content are unlikely to be well modelled using a normal distribution.

Two machine learning models will be created for this section of the project. The first model will look at attempting to assign a score representing the effort or engagement a student shows based on their working behaviour. The second will attempt to assign a score to individuals showing the likelihood that they are plagiarising content during their work. For each model, a set of data will be generated with 5000 instances. Each model will be evaluated using 66/33% split of training instances to testing instances.

Component Functional Requirements

Below is a summary of the functional requirements of the Machine Learning Component to be created for this project.

- Two machine learning models. One demonstrating the application of regression to calculate the engagement of a student and one demonstrating the application of regression to establish potential for plagiarism.
- Both models should be trained and tested using data models with attributes which could be collected from the Client.
- The final model should be translatable to a form usable in the larger scope of the project (i.e. results can be calculated and displayed for users of the web application).

Required Resources

In order to conduct the study and generate the data for the solution two different tools will be used:

- Weka 3.8 (*Weka 3 - Data Mining with Open Source Machine Learning Software in Java*, n.d.).
- Minitab (*Minitab Statistical Software - Minitab*, n.d.).

Weka is a piece of software written in Java which provides users an easy means of running different machine learning algorithms with varying complexity and parameters. Due to it's flexibility and ease of use it is ideal for exploring the use of different algorithms without investing excessive time in their development.

Minitab is a statistical software suite which caters for the generation and analysis of data in graphical and purely mathematical forms. Minitab will be used to generate the necessary data sets and for the preprocessing of the set prior to their application to the ML algorithms.

Methodology

Before implementation of the ML algorithms can take place the data must be generated. As discussed above the data needed to conduct these tests will not be available in large enough quantities. As such the data will first be generated using Minitab.

For the purpose of this solution a scenario is offered to create some bounds by which the test adheres. This is described in the following paragraph.

Scenario

A selection of 5000 students have been given an assignment. The student have been informed that they have a week to complete this assignment. Each student has been told that they are allowed to work on the assignment between from day one to day seven. On day eight no more work must be done and the assignment must be submitted. Finally the students have been informed that they are to run the aforementioned Client whilst writing their assignment and that their actions will be recorded. Any resources can be used to write the assignment but they are to be referenced as per the academic requirements. It is anticipated that the assignment will take a total of 12 hours on average to complete and an average of 2000 words is expected to be written for each submission.

Data Generation

Minitab offers the ability to generate a random set of data based on a specified distribution. This features will be used to generate the data for the different attributes for each instance. In total 5000 instances will be generated for each model. Two data models will be created, one for the evaluation of student engagement and one for the prediction of plagiarism. Both models will make use of attributes which could be collected from the data recorded using the client. A summary of these attributes is shown below:

- Number of days worked during the seven day period.
- Average number of sessions worked each day.
- Average duration of each session.
- Average number of actions recorded for each session.
- Average number of keyboard actions recorded for each session.
- Average number of mouse actions recorded for each session.
- Total duration spent working on the project.
- Total number of actions recorded for the working duration.
- Total number of keyboard actions recorded for the working duration.
- Total number of mouse actions recorded for the working duration.

Both models will make use of these attributes as a base model. Each attribute will be generated using random values from a defined normal distribution.

Finally each model will require specific measurements to be created to enable them for use with regression base ML algorithms. Model one, concerned with the assignment of a percentage value indicating the engagement of the student, will need a measure of effort over the time period. This measure will then be calculated into a percentage value.

Model two will require a metric which shows the level of effort from the previous model in comparison to the final grade they received. This metric will be indicative of the balance between effort and result. From this value a percentage will be computed indicating the likelihood of plagiarised content.

Machine Learning Algorithms

The selected to implement the solution for this section of the project is the k-Nearest Neighbour algorithm. The k-Nearest Neighbour algorithm is a supervised learning algorithm capable of regression calculations. k-Nearest Neighbour was selected due to it's simplicity and the researcher's previous experience with implementing the algorithm.

kNN will be implemented using the Weka suite and will be ran with the configuration parameters tuned in different ways to identify the best performing configuration. Further discussion into the specific tuning of these parameters will be discussed during the Machine Learning Component section in the Implementation chapter.

After running the algorithm in its various configurations the best performing configuration for each data model will be identified based on its accuracy score. The configurations and their performance will be discussed further in the Implementation chapter.

Implementation

Due to the nature of this project, the development process was quite iterative and testing of individual parts of the solution took place during the creating process. It is hard to put the implementation process into a linear time-line description. In many cases the alteration of one aspect of the solution would require refactoring of code in another independant area.

Below, each major component of the project has been broken down and discussed independently to the others but with references to their interaction where necessary. Where possible code segments will be provided to improve the clarity of the description, however, this will be kept to a minimum as the size of the code-base which makes up these solutions is quite large. All code used to implement the following components can be found in the Appendices (Other) section at the end of this document.

Client

The client's primary task is to collect descriptive measures about a user's working behaviour. It is written in such a way that minimal interaction is required for the user to run the programs. The client is presented to the user on a pre-formatted USB pendrive. In order for the client to run the system needs the following requisite software installed:

System Requirements:

- A Linux-based operating system.
- Python 2.7 (*Welcome to Python.org*, n.d.) fully installed with the IDLE (*24.6. IDLE — Python 2.7.13 documentation*, n.d.) programming environment.
- An internet connection.

The client is written using Python 2.7 and requires the interpreter to be installed for the code to run. The IDLE programming environment is required for the GUI to run. Interestingly, on testing the client on multiple machines the software would only run on those with IDLE installed. It is still not known why this is necessary but it is suggested that the library/module used for the graphical front-end seems to have some requirement for this software. Finally, an internet connection is required so that when the user is finished using the software the relevant recorded information can be submitted to the central server as discussed in the following section.

Python 2.7 was selected as the language for developing this solution due to its light weight, ubiquity and interoperability between operating systems. Currently, the software will only operate on Linux-based operating systems, however this is due to the keylogging library, not Python.

A deeper look at the client will reveal that it conducts three critical processes. The client, has a GUI interface which is presented to the user to start the process. This GUI allows the user to provide a identifiable numeric value, start the program, stop the program and indicate to the user when the program is operating. This program uses the library appJar (*appJar*, n.d.) to design and implement the GUI. Below is an image showing the GUI presented to the user upon plugging in the USB pen drive.

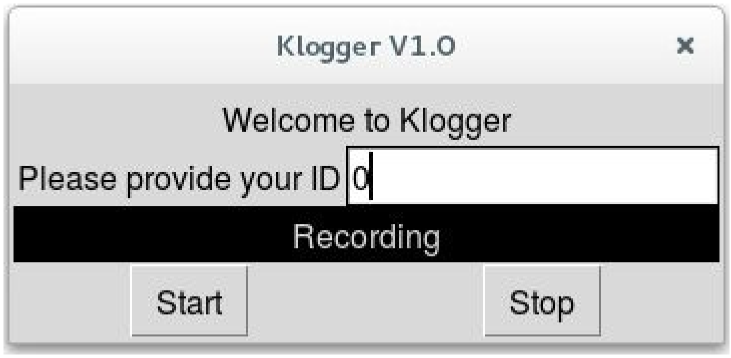


Figure 5: Klogger Client UI.

Gui.py is the script which describes the front-end interface using appJar. After entering the unique ID and clicking start the gui.py passes the integer value of the ID to kloggerClientHTTP.py which takes over execution and handles the collection and packaging/sending functionality.

Central to the functionality of the kloggerClientHTTP.py is the pyxhook library (*JeffHoogland/pyxhook*, n.d.), this library is a re-implementation of the pyHook (*pyHook / Wiki / Main_Page*, n.d.) library for Windows designed to work inside Linux with the X window manager (*X Window System Architecture Overview HOWTO*, n.d.). Pyxhook, implements a selection of callbacks which can be incorporated into a script to be notified when an event occurs through the use of a mouse or keyboard. Below is a basic example script provided by Jeff Hoogland to explain how to use these callbacks:

```
import pyxhook
import time


# This function is called every time a key is presssed
def kbevent(event):
    global running
    # print key info
    print(event)

    # If the ascii value matches spacebar, terminate the
while loop
    if event.Ascii == 32:
        running = False


# Create hookmanager
hookman = pyxhook.HookManager()
# Define our callback to fire when a key is pressed down
hookman.KeyDown = kbevent
# Hook the keyboard
hookman.HookKeyboard()
```

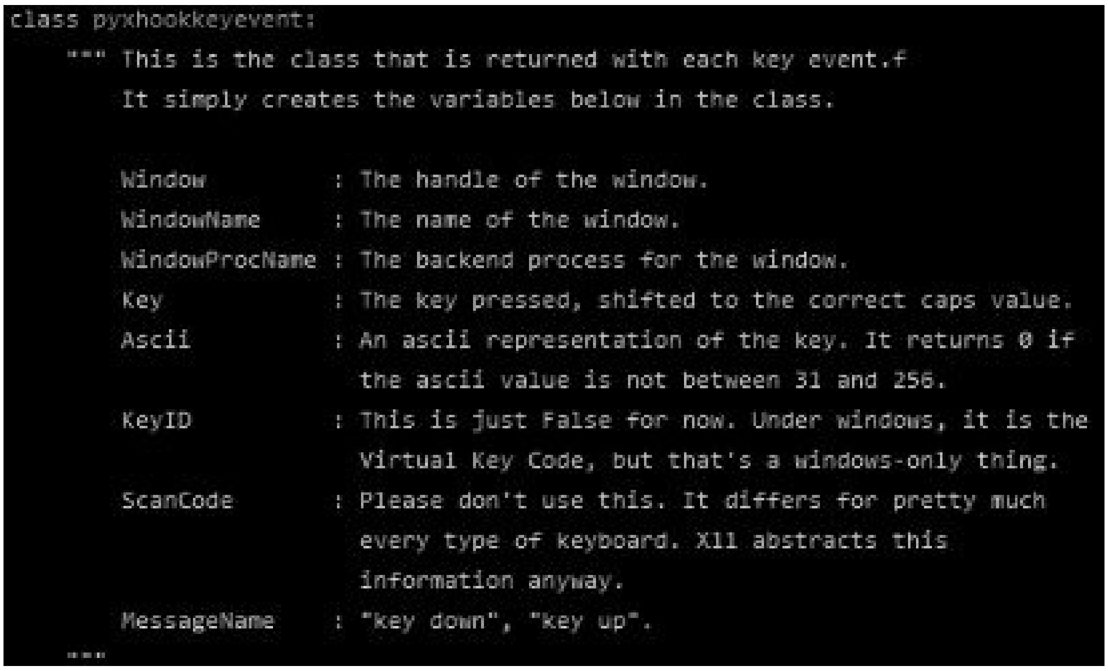
```
# Start our listener
hookman.start()

# Create a loop to keep the application running
running = True
while running:
    time.sleep(0.1)

# Close the listener when we are done
hookman.cancel()
```

Example.py *JeffHoogland/pyxhook* (n.d.) GitHub. [Online] [Accessed on 17th April 2017] https://github.com/JeffHoogland/pyxhook/blob/master/example.py.

kloggerClientHTTP.py implements the pyxhook hookmanager and relevant callbacks similar to the example code seen above. For further investigation, please see the kloggerClientHTTP.py script included in the Appendix.

kloggerClientHTTP.py retrieves the keyboard and mouse events using pyxhook. The events are cached until the user wishes to stop recording. When the uses clicks “stop” the recorded events are packaged into a JSON object and sent to the central server using the HTTPS network protocol.

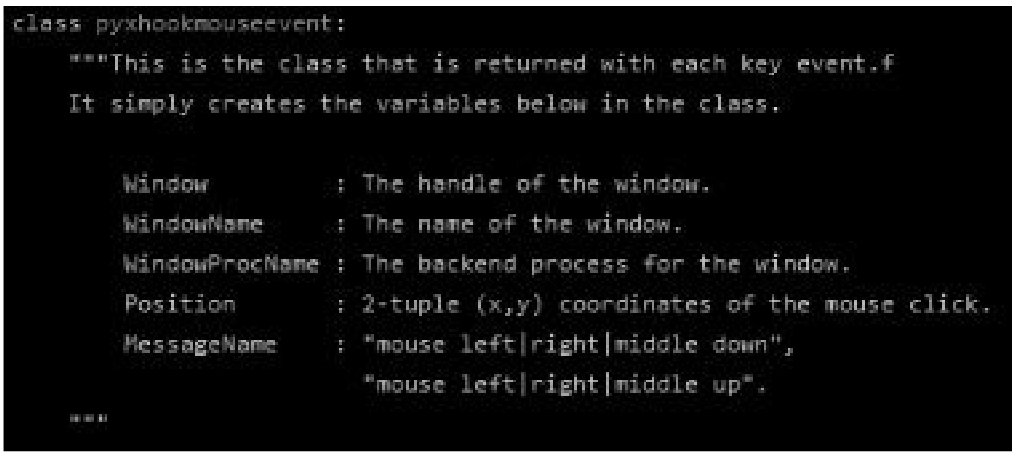
Pyxhook records information about both keyboard actions (keystrokes) and mouse actions (movement, clicks). More specifically an event object which is caught by the callbacks for a keyboard event is an object which is configured as below:



```
class pyxhookkeyevent:
    """ This is the class that is returned with each key event.f
        It simply creates the variables below in the class.

        Window         : The handle of the window.
        WindowName     : The name of the window.
        WindowProcName : The backend process for the window.
        Key            : The key pressed, shifted to the correct caps value.
        Ascii          : An ascii representation of the key. It returns 0 if
                         the ascii value is not between 31 and 256.
        KeyID          : This is just False for now. Under windows, it is the
                         Virtual Key Code, but that's a windows-only thing.
        ScanCode       : Please don't use this. It differs for pretty much
                         every type of keyboard. X11 abstracts this
                         information anyway.
        MessageName    : "key down", "key up".
    """
```

Figure 6: Pyxhookkeyevent class.

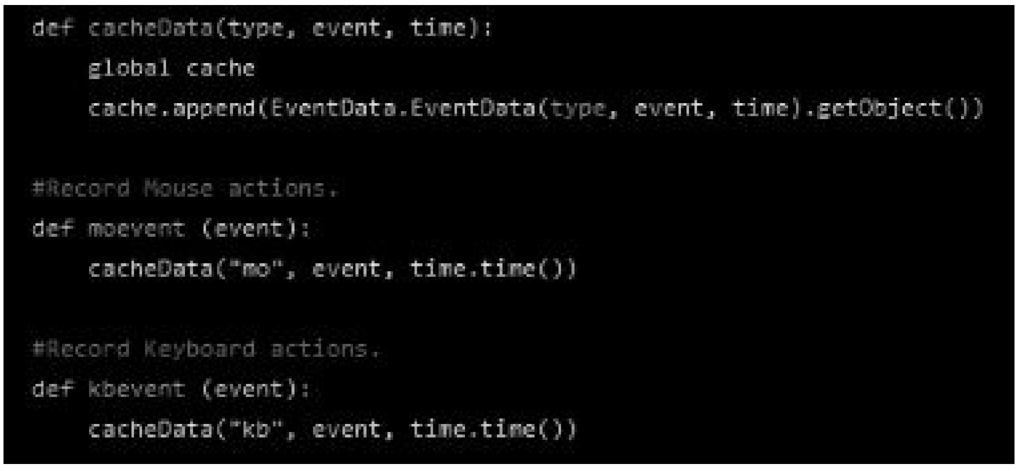
Similarly, mouse actions caught by the callback method are presented as an object configured as below:



```
class pyxhookmouseevent:
    """This is the class that is returned with each key event.f
    It simply creates the variables below in the class.

        Window          : The handle of the window.
        WindowName      : The name of the window.
        WindowProcName  : The backend process for the window.
        Position        : 2-tuple (x,y) coordinates of the mouse click.
        MessageName     : "mouse left|right|middle down",
                          "mouse left|right|middle up".
    """
```

Figure 7: Pyxhookmouseevent class.

As each callback is triggered, either due to a keyboard action or mouse action, the event object is stored along with a timestamp in a tuple "cache" for packaging at the end of the session.

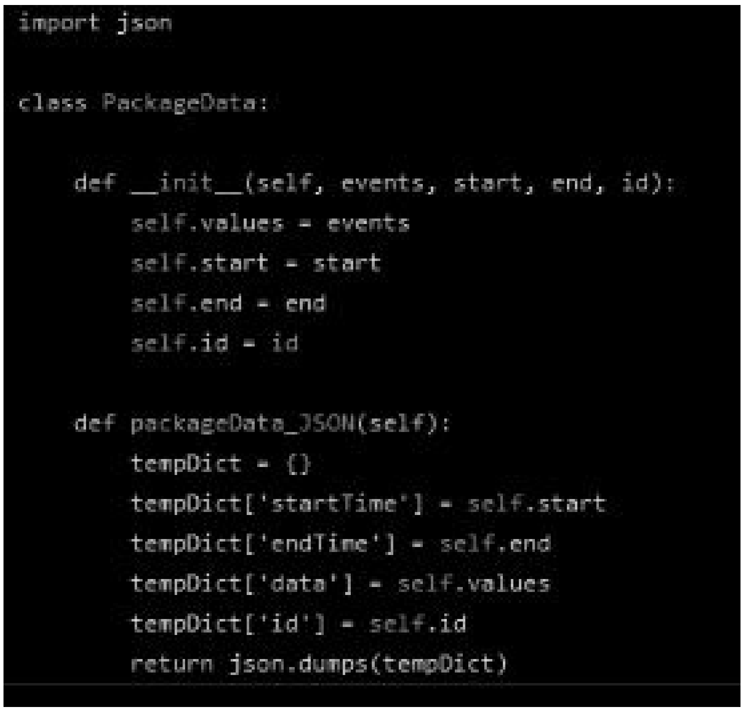


```
def cacheData(type, event, time):
    global cache
    cache.append(EventData.EventData(type, event, time).getObject())

#Record Mouse actions.
def moevent (event):
    cacheData("mo", event, time.time())

#Record Keyboard actions.
def kbevent (event):
    cacheData("kb", event, time.time())
```

Figure 8: kloggerClientHTTP.py callback methods.

Packaging of the recorded information is handle by the PackageData class. This class simple exposes a constructor which takes the tuple of events, a start timestamp, end timestamp and the integer ID for the user. This data is considered the session data, specific to one individual session for the user. Further session for a specific user can be identified via the unique identifier.

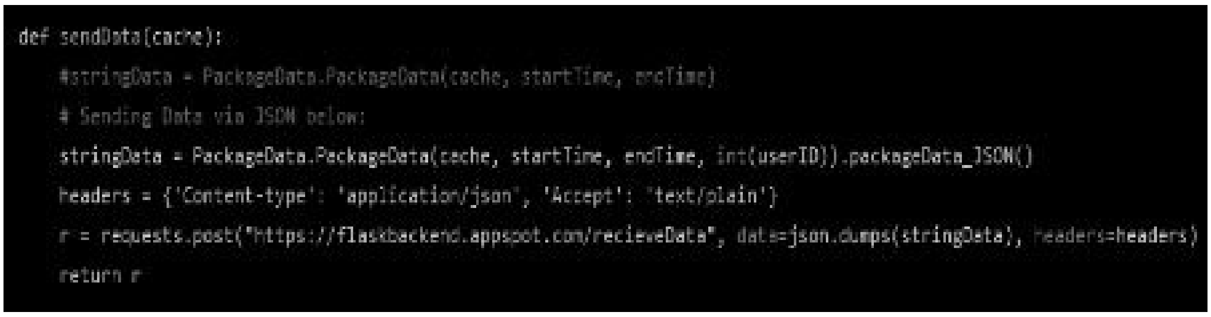
PackageData.py has only one public object method which converts the content of the object into a dictionary structure and then returns the dictionary as a JSON string. The processing of the dictionary to JSON string is handled by the json module included in the Python Standard Library (*18.2. json — JSON encoder and decoder — Python 2.7.13 documentation*, n.d.).



```python
import json

class PackageData:

    def __init__(self, events, start, end, id):
        self.values = events
        self.start = start
        self.end = end
        self.id = id

    def packageData_JSON(self):
        tempDict = {}
        tempDict['startTime'] = self.start
        tempDict['endTime'] = self.end
        tempDict['data'] = self.values
        tempDict['id'] = self.id
        return json.dumps(tempDict)
```

Figure 9: PackageData.py class.

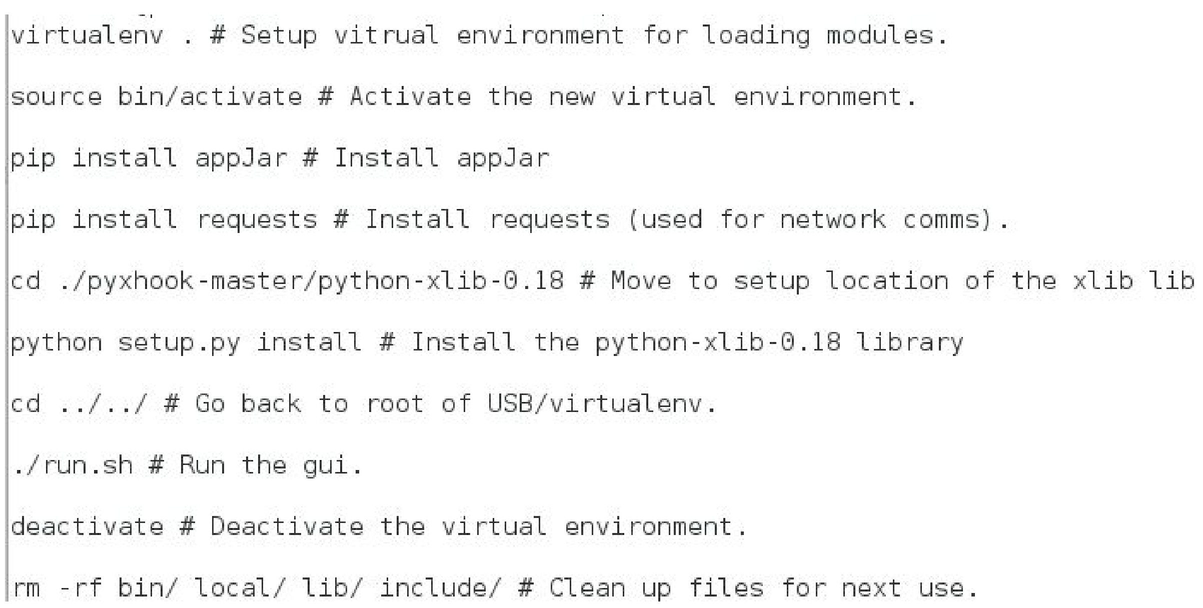
Once the cached data has been processed and formatted into a JSON object it is sent to the central server via a POST request and returns the status of the POST request back to the gui.py script.



```python
def sendData(cache):
    #stringData = PackageData.PackageData(cache, startTime, endTime)
    # Sending Data via JSON below:
    stringData = PackageData.PackageData(cache, startTime, endTime, int(userID)).packageData_JSON()
    headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
    r = requests.post("https://flaskbackend.appspot.com/recieveData", data=json.dumps(stringData), headers=headers)
    return r
```

Figure 10: kloggetClientHTTP.py sendData method.

All of the components described above are initially configured and started by and autorun.sh script stored on the USB pen drive containing the code. Autorun.sh creates a Virtual Environment (*Virtualenv — virtualenv 15.1.0 documentation*, n.d.) inside of which the code runs, independently of the other installations on the system. Because these scripts require specific non-standard libraries to run, a virtual environment must be configured so that these required libraries are not flagged by the system as not being installed. After creating the environment, autorun.sh start the program with the Python 2.7 interpreter and the GUI (gui.py) is loaded execution is then handled by the python scripts until the user stops the recording process.



```
virtualenv . # Setup vitrual environment for loading modules.

source bin/activate # Activate the new virtual environment.

pip install appJar # Install appJar

pip install requests # Install requests (used for network comms).

cd ./pyxhook-master/python-xlib-0.18 # Move to setup location of the xlib lib

python setup.py install # Install the python-xlib-0.18 library

cd ../../ # Go back to root of USB/virtualenv.

./run.sh # Run the gui.

deactivate # Deactivate the virtual environment.

rm -rf bin/ local/ lib/ include/ # Clean up files for next use.
```

Figure 11: Autorun.sh script.

Testing:

Testing of the client was conducted iteratively, throughout the development process by the researcher. However, to confirm the code was indeed operating correctly, additional user tests were conducted by 2 other individuals on systems configured for use in the University labs. These user tests were representative of the installation and execution of the process that a live user would be expected to take.

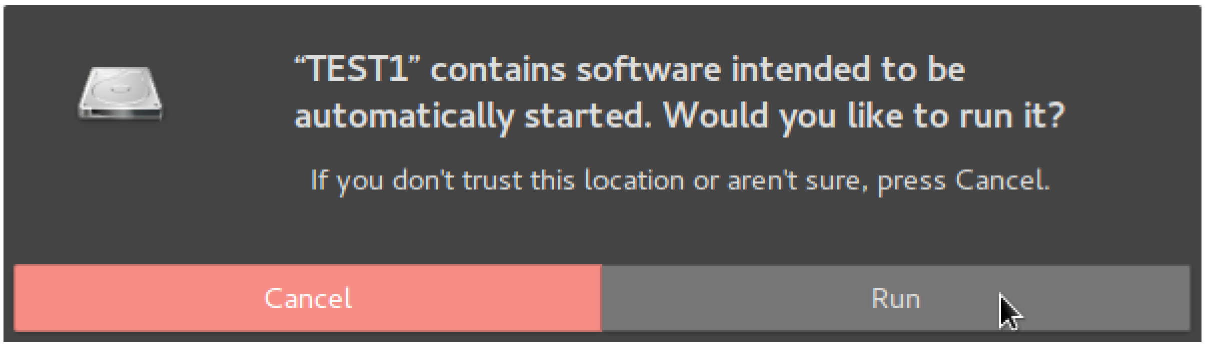
Both users were provided with the code-base pre-installed onto two separate USB pen drives. Each user was given an unique identifiable integer value to use during testing. The first user was given the ID 1 and the second ID 2. The developer used ID 0 for the tests conducted before and after these user tests took place. After inserting the dive, the code base was set up and the program started the GUI. Each user input their ID and clicked start. After clicking start the users were instructed to work as they normally would for anywhere between 10 minutes and an hour. Additionally, each participant was informed not to input any sensitive information during testing. This process was conducted 2-3 times by each user over the course of a week, in the university laboratories whilst doing their regular work.

Demonstration

The following section demonstrates the use of the client.

Step 1: Insert the USB

After inserting the USB the user is presented with a pop-up window asking if they would like to run the autorun.sh program.



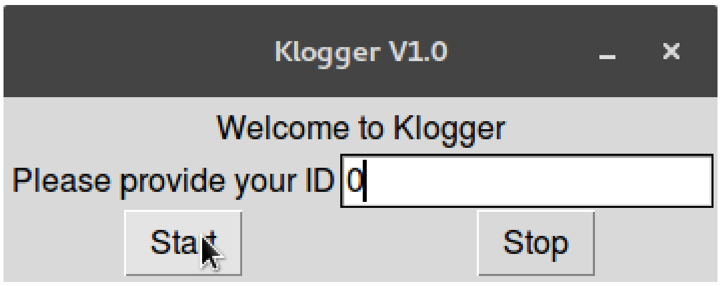
"TEST1" contains software intended to be automatically started. Would you like to run it?

If you don't trust this location or aren't sure, press Cancel.

Figure 12: Demonstration step 1.

Step 2: Provide ID and start

Once the user has selected to run the program Klogger's UI is presented. The can now provide their ID and click the "Start" button.



Klogger V1.0

Welcome to Klogger

Please provide your ID 0

Figure 13: Demonstration step 2.

Step 3: Recording

Whilst recording the UI displays a flashing "Recording" indicator.

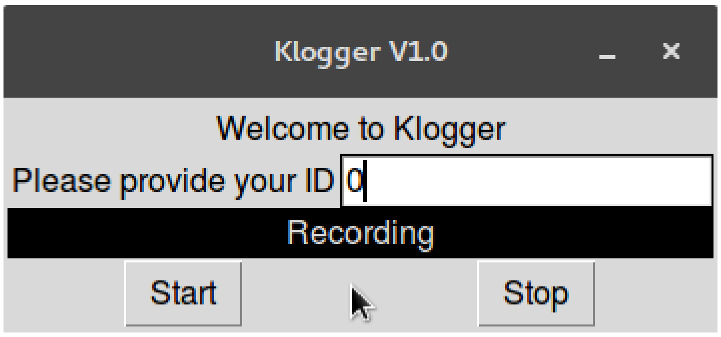


Figure 14: Demonstration step 3.

Step 4: Stop recording

When the user has finished working they can click the “Stop” button to finish the monitoring process and close Klogger.

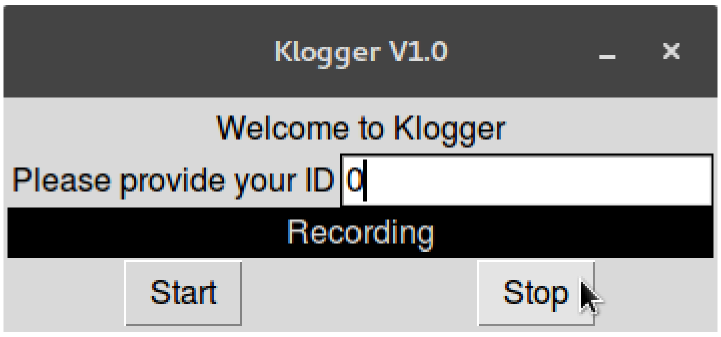
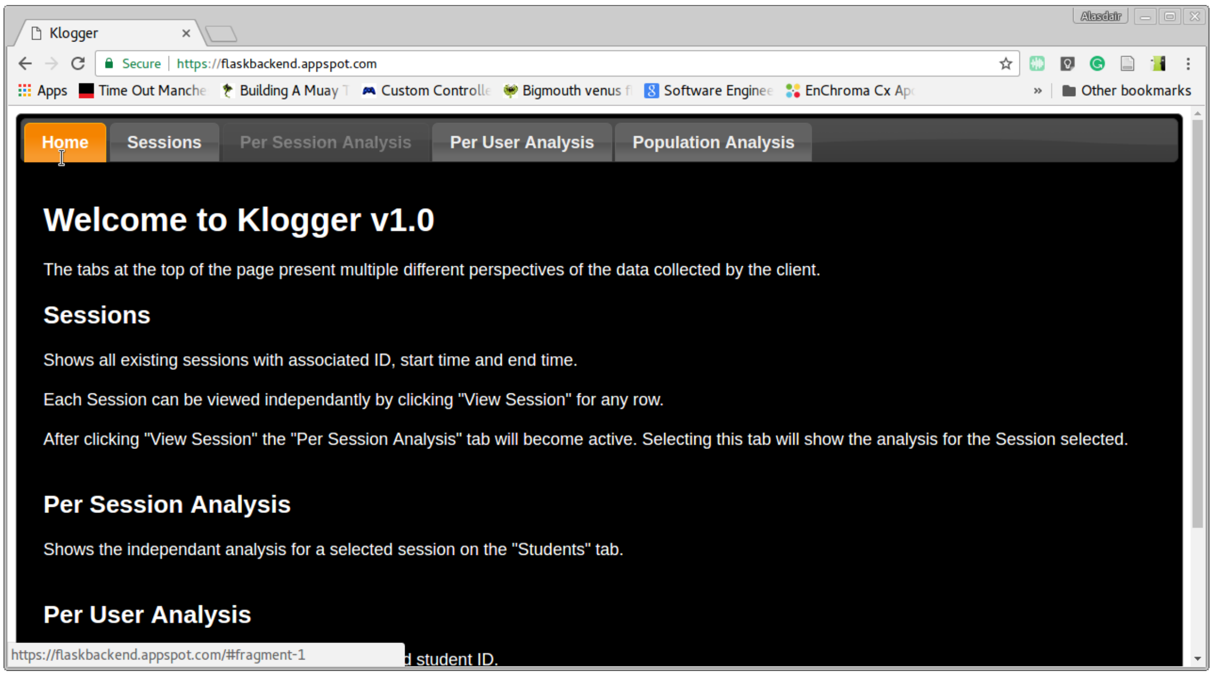


Figure 15: Demonstration step 4.

Step 5: Browse to the web application

Once Klogger closes it has finished submitted the session’s recorded data to the server. A staff member can now browse to https://flaskbackend.appspot.com to view the session.



Welcome to Klogger v1.0

The tabs at the top of the page present multiple different perspectives of the data collected by the client.

Sessions

Shows all existing sessions with associated ID, start time and end time.

Each Session can be viewed independantly by clicking "View Session" for any row.

After clicking "View Session" the "Per Session Analysis" tab will become active. Selecting this tab will show the analysis for the Session selected.

Per Session Analysis

Shows the independant analysis for a selected session on the "Students" tab.

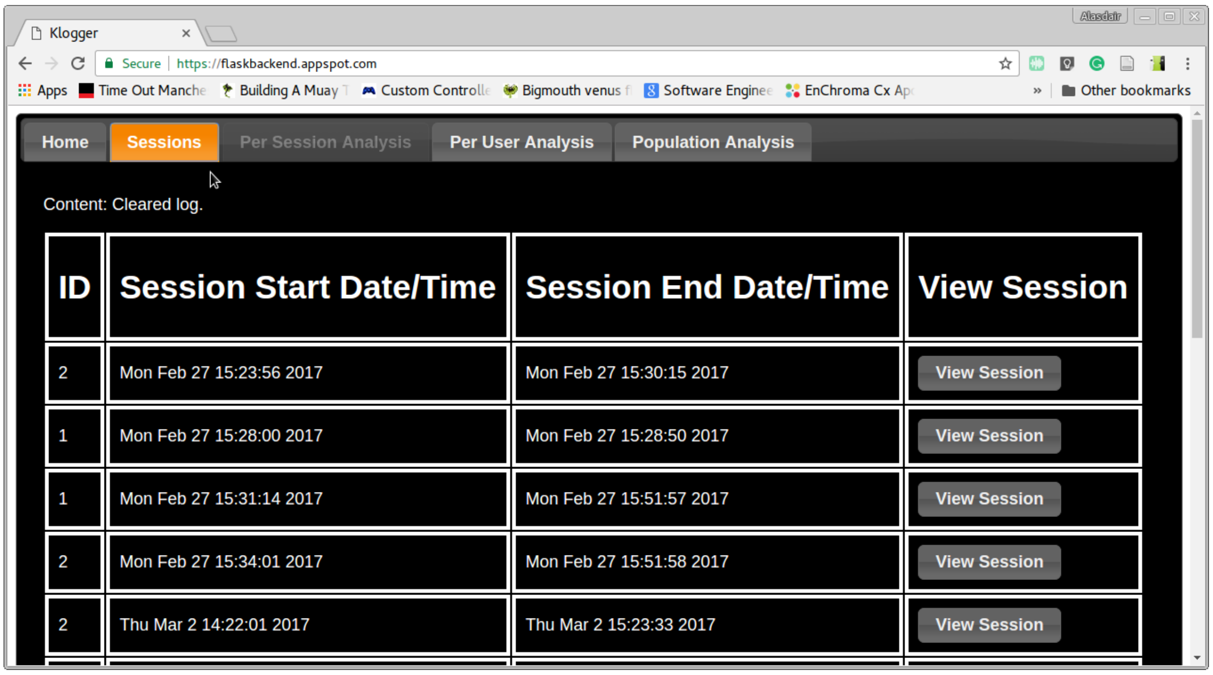
Per User Analysis

https://flaskbackend.appspot.com/#fragment-1 d student ID.

Figure 16: Demonstration step 5.

Step 6: View the sessions

To view all the session which have been monitored the staff member clicks on the "Sessions" tab.



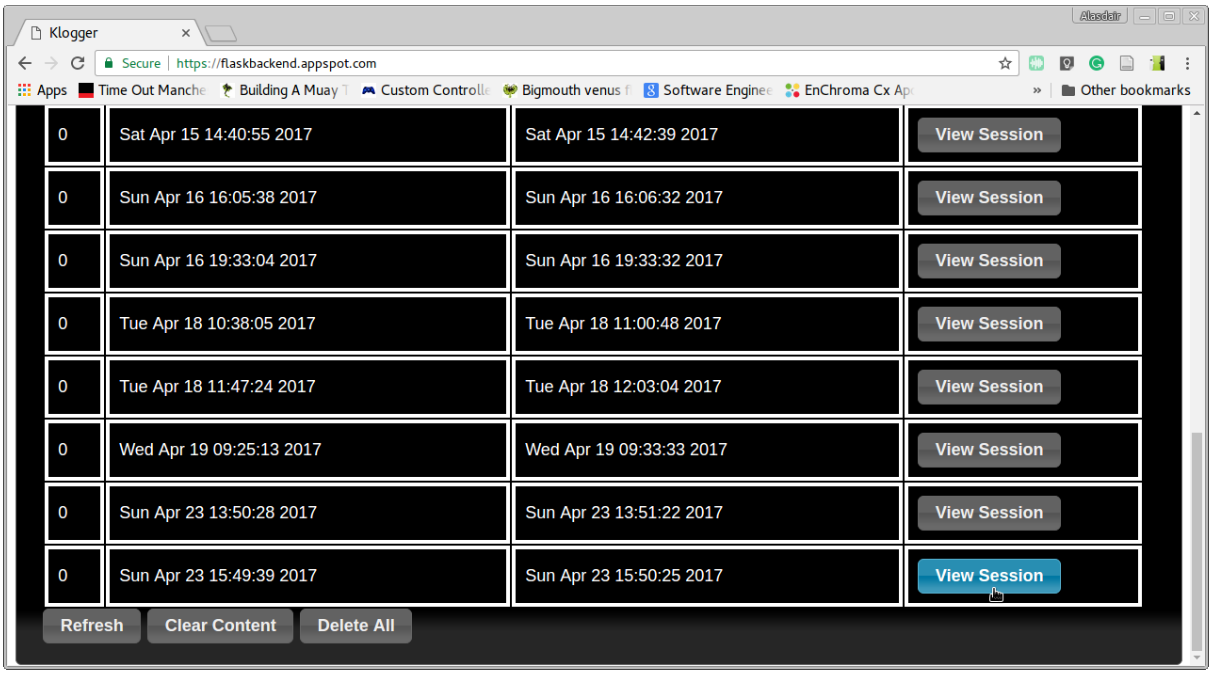
Content: Cleared log.

ID	Session Start Date/Time	Session End Date/Time	View Session
2	Mon Feb 27 15:23:56 2017	Mon Feb 27 15:30:15 2017	View Session
1	Mon Feb 27 15:28:00 2017	Mon Feb 27 15:28:50 2017	View Session
1	Mon Feb 27 15:31:14 2017	Mon Feb 27 15:51:57 2017	View Session
2	Mon Feb 27 15:34:01 2017	Mon Feb 27 15:51:58 2017	View Session
2	Thu Mar 2 14:22:01 2017	Thu Mar 2 15:23:33 2017	View Session

Figure 17: Demonstration step 6.

Step 7: Select the session

A list of all the sessions can be seen in chronological order with each student's ID on the left hand side. After identifiying the session to be reviewe, the staff member clicks the "View Session" button.

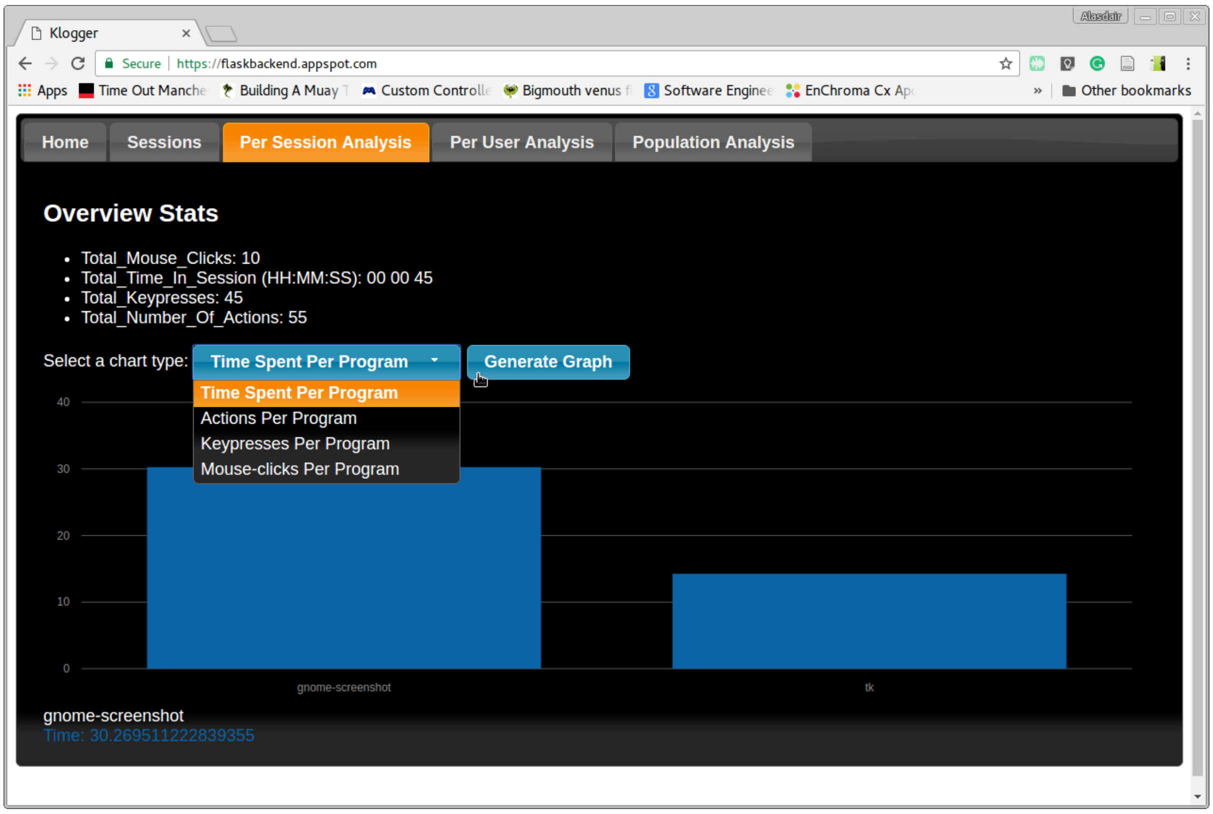


0	Sat Apr 15 14:40:55 2017	Sat Apr 15 14:42:39 2017	View Session
0	Sun Apr 16 16:05:38 2017	Sun Apr 16 16:06:32 2017	View Session
0	Sun Apr 16 19:33:04 2017	Sun Apr 16 19:33:32 2017	View Session
0	Tue Apr 18 10:38:05 2017	Tue Apr 18 11:00:48 2017	View Session
0	Tue Apr 18 11:47:24 2017	Tue Apr 18 12:03:04 2017	View Session
0	Wed Apr 19 09:25:13 2017	Wed Apr 19 09:33:33 2017	View Session
0	Sun Apr 23 13:50:28 2017	Sun Apr 23 13:51:22 2017	View Session
0	Sun Apr 23 15:49:39 2017	Sun Apr 23 15:50:25 2017	View Session

Figure 18: Demonstration step 7.

Step 8: View summary and generate chart

Clicking “View Session” in the Sessions tab will activate the Per Session Analysis tab and populate it with a summary of the session. Below this summary the staff member can select a graphical output for various different measurements. Selecting the desired graph from the dropdown menu and clicking “Generate Graph” will cause the application to display the graph requested.



Overview Stats

- Total_Mouse_Clicks: 10
- Total_Time_In_Session (HH:MM:SS): 00 00 45
- Total_Keypresses: 45
- Total_Number_Of_Actions: 55

Select a chart type: Time Spent Per Program

gnome-screenshot
Time: 30.269511222839355

Figure 19: Demonstration step 8.

Discovered Bugs:

The most notable difficulty which presented itself to both users was the significant time required to upload the cached data after each session. During this time the GUI would become unresponsive and uninformative. For shorter session of between 10-15 minutes the program would only become unresponsive for a short amount of time. Unfortunately, when the session duration was in excess of 20 minutes the program became unresponsive to the point of the users being unsure if the program had crashed. On later inspection, it became apparent that the program had not crashed but was taking a significant amount of time to process the large caches which had been generated during the session.

Despite this delay upon ending the session the program did not interfere with the user's work process. The client was left to run in the background and no noticeable change in system performance was detected. A minor issue which was identified, was the possibility of users stopping the recording process after they had finished working. On one occasion a user forgot that they had the program running after they had discontinued working. Luckily, the machine was not used any further after that point and no additional data had been recorded.

Potential Solutions:

A possible solution to aid with the slow speed of processing and sending the data may be to employ some form of compression. It is possible that JSON objects are not suitable for sending the data which is collected in such vast quantities. It may be preferable to store the data in an XML file which can then be compressed to reduce the overall size of the transmission.

Another alternative is to regularly send smaller JSON objects throughout the recording process. Python has proven to be difficult in this respect. Early versions of the client sent the cached data every 5 seconds. Unfortunately, due to the speed with which data is generated and because Python is not capable of true multi-threading, this would cause the program to stop recording new data whilst the program was sending the cache to the server. Furthermore, this would cause a vast amount of network traffic and potentially flood the network.

Git (*Git*, n.d.) or SVN (*Apache Subversion*, n.d.) could also be potentially applied to submit a log of events at the end of the session. Using Git or SVN would have the additional benefits of providing a historical record of each session which could be browsed in a more user friendly way. The structure offered by a versioning system is extremely appealing, however, the additional effort required to configure a versioning system on the central server is substantial.

Finally, there is the possibility that the slow transmission speed is in part due to the performance of the server rather than the client. In this case, the solution may require that additional processing power is allocated to the back-end to handle the data as it is received. Unfortunately, due to limited financial resources this was not possible.

Future Improvements:

One of the recurring issue which made designing and implementing the client difficult was the lack of true multi-threading with the Python language. The addition of multi-threading support would greatly improve the potential for the client to handle multiple processes at one time. The GUI, data collection and data transmission components could all be run on separate threads such that execution of one components would not impact the execution of another. This would better support the real-time nature of the code.

Currently, the client requires a Linux system in order to operate. Additional support for another keylogging library would need to be incorporated into the client in order for it to be fully cross-compatible between operating systems. Python was selected for its excellent cross-compatibility and its low claim on system resources. Unfortunately, the Python libraries for the data collection are specific to the window manager used on the system. The window managers used for Windows operating systems and Linux operating systems are significantly different enough to require separate libraries. It would be relatively trivial to implement the additional library pyhook (*pyHook / Wiki / Main_Page*, n.d.) to provide support for Windows. Given this, if the transition was made to use an alternative language other than Python (such as Java as mentioned above), it would be necessary to identify a suitable alternative library capable of recording similar properties as pyxhook and pyhook do.

In a publically distributed form of this client more consideration would need to be provided for the application of secure programming. A full analysis of the attack surface of the client would need to be conducted to identify areas of the code which are vulnerable to tampering/reverse engineering. Currently, a technically minded user would be capable of tampering with the cached data to alter the contents of their session (making it seem like they did more work than they really did for instance). Additionally, there is no mechanism to stop one user impersonating another. Proper authentication and authorization measures should be implemented in future versions. Many other potential attack vectors may exist in the current prototype (e.g. side-channel attacks, MITM vulnerabilities etc.). A full evaluation of the code is of priority in future versions of the code-base.

Server

Google's Cloud Platform offers many services for developers looking to acquire back-end hardware on-demand. Google App Engine (*App Engine - Build Scalable Web & Mobile Backends in Any Language*, n.d.) and Datastore (*Google Cloud Datastore Overview | Cloud Datastore Documentation*, n.d.) services were used in the creation of the central server for this project. App Engine was used as the virtual environment to host the web application. Datastore is a NoSQL service used for the storage of data collected from the client.

App Engine supports multiple languages compatible with their service. In order to build the web application with Python, Google's supporting documentation recommends using the Flask framework for the creating web applications compatible with App Engine. Python was selected as the preferred language for the server because of its use in the client. The development of the client took place before considering the server portion of this project and, as such, the familiarity gained from using Python for development of the client made it an appealing choice for the development of the server.

Flask is a web application framework written in Python. Getting a familiarity with Flask proved challenging as its design principles (although flexible) offered many different way of implementing the same structure. Previous experience creating web applications following the Model View Controller paradigm in Java, combined with the additional learning curve offered by a new framework made the process of constructing Flask application time-consuming and challenging. Once the basic operation of Flask was better understood, implementing the necessary features for the server became relatively fluid and quick. Flask proved to be instrumental to the ease with which the server was developed after the initial learning phase. It is of note that initial attempts to develop the server without the aid of a framework proved to be far more difficult.

Flask provides the core structure to create the functionality offered by the web application. The MVC model was broadly used as the design paradigm during development. The server's code-base was separated into three areas as defined when taking the MVC approach to web application development. The controller portion of the server manages the requests from both the web application users (academic staff) and the submission of data sent from the client (students). Any processing of the data for analytical purpose was the domain of the controller portion of code.

Data processing and storage in Google's Datastore service was handled by the model section of the MVC paradigm. All processing of the raw JSON objects coming from the client to be stored are handled by this group of code. Additionally the model handles the fetching of any data to be analysed by the controller and exposes a subset of methods to provide this functionality.

Data which has been requested by web application users (staff) is passed from the controller to the view components. The view is responsible for formatting the data such that it can be sent back to the web application users. The formatted data is sent to the web application front-end, developed with JavaScript/JQueryUI (jquery.org, n.d.), which handles the dynamic web pages for the user interface.

Any functionality offered by the web application front-end is handled using JavaScript. Specifically, JQueryUI was used for the ease with which the libraries widgets and UI components could be implemented. JQueryUI comes with the added benefit of providing the JQuery library as part of its base package. JQuery was used extensively to generate the asynchronous requests for the user to interface with the back-end Flask Controller.

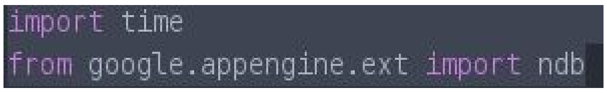
Graphical representations of the data are generated using the Morris.js (*morris.js*, n.d.) graphing library. In hindsight this library was not fully appropriate for the task. The variety of graphs available and the degree to which these graphs can be customized is not extensive. However, the majority of options available when identifying an appropriate library were either only trial versions or had licensing restrictions which made them inappropriate. Since completing the development of the project, new open source libraries have been made available which are potentially better suited.

Model

Below is a description of the components which make up the model of the MVC paradigm used in this project.

Model.py

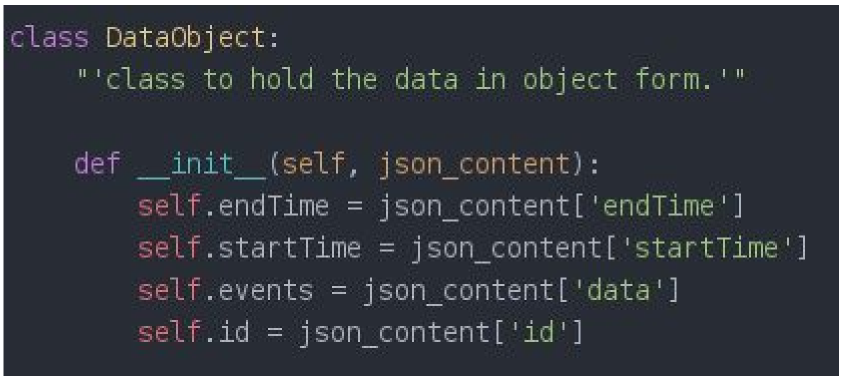
Model.py is the sole Python script responsible for the interactions with the Datastore used to record the data sent by the client. Modle.py uses the “ndb” module from the App Engine API provided by Google (*google.appengine.ext.ndb package | App Engine standard environment for Python*, n.d.). Python’s Standard Library also provides a module “time” which is used in Model.py for the processing of the timestamps sent in the data from the client.



```python
import time
from google.appengine.ext import ndb
```

Figure 20: Model.py imports.

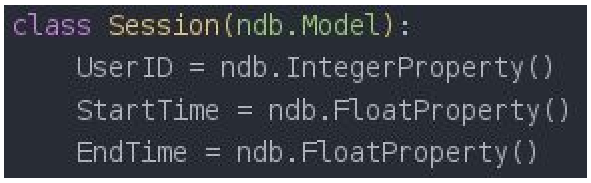
Data is passed to the Model.py script from the controller (main.py) in as a JSON object. This object is parsed and formed into a Python object, DataObject, defined in Model.py. DataObject objects are then used to better manipulate the data for storage in the Datastore.



```python
class DataObject:
    "'class to hold the data in object form.'"

    def __init__(self, json_content):
        self.endTime = json_content['endTime']
        self.startTime = json_content['startTime']
        self.events = json_content['data']
        self.id = json_content['id']
```

Figure 21: Model.py DataObject class.

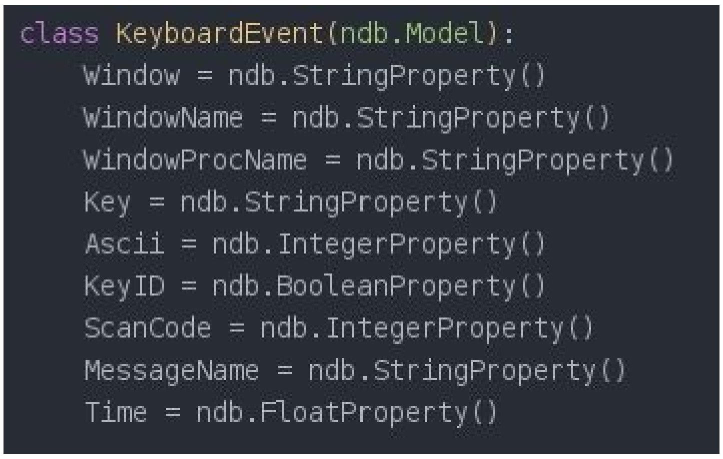
Each DataObject object is then processed and broken into three separate parts for storage in Datastore. The Session class is the ndb.Model for storing the information about a single session. The user’s ID, starting time of the session and ending time of the session is stored. This is the parent entity used to relate all subsequent KeyboardEvent/MouseEvent entities to the user's session. Thus, each action can be linked directly to a user and the session in which it occurred.



```
class Session(ndb.Model):
    UserID = ndb.IntegerProperty()
    StartTime = ndb.FloatProperty()
    EndTime = ndb.FloatProperty()
```

Figure 22: Model.py Session class NDB model.

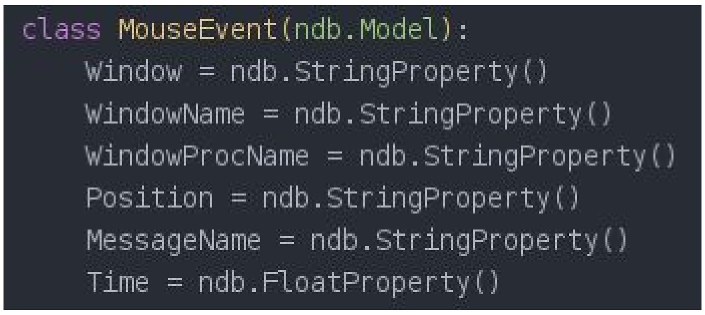
The KeyboardEvent class is an ndb.Model representing the KeyboardEvent entity. Each keyboard event which is generated by the client has associated data such as the "WindowName", "Key" and "Ascii" values. The structure of the KeyboardEvent entity directly relates to the event objects generated by the pyxhook library in the Client.



```
class KeyboardEvent(ndb.Model):
    Window = ndb.StringProperty()
    WindowName = ndb.StringProperty()
    WindowProcName = ndb.StringProperty()
    Key = ndb.StringProperty()
    Ascii = ndb.IntegerProperty()
    KeyID = ndb.BooleanProperty()
    ScanCode = ndb.IntegerProperty()
    MessageName = ndb.StringProperty()
    Time = ndb.FloatProperty()
```

Figure 23: Model.py KeyboardEvent class NDB model.

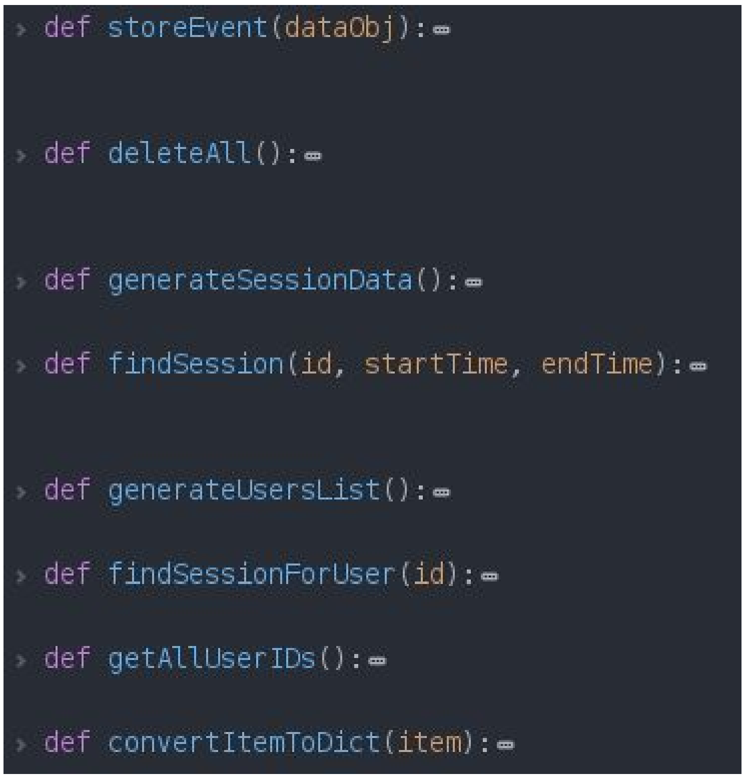
Similarly to the KeyboardEvent ndb.Model, the MouseEvent ndb.Model has a structure which directly related to its event objects generated by the pyxhook library used in the client.



```
class MouseEvent(ndb.Model):
    Window = ndb.StringProperty()
    WindowName = ndb.StringProperty()
    WindowProcName = ndb.StringProperty()
    Position = ndb.StringProperty()
    MessageName = ndb.StringProperty()
    Time = ndb.FloatProperty()
```

Figure 24: Model.py MouseEvent class NDB model.

Model.py exposes various function which can be used to interface with the Datastore. Each method is quite specific to it's function and so will not be explained in detail here. Further information about the internal operation of each function can be found in the Appendices or the GitHub repository. Below is a summary of the methods provided by Model.py.



```
def storeEvent(dataObj):

def deleteAll():

def generateSessionData():

def findSession(id, startTime, endTime):

def generateUsersList():

def findSessionForUser(id):

def getAllUserIDs():

def convertItemToDict(item):
```

Figure 25: Model.py methods.

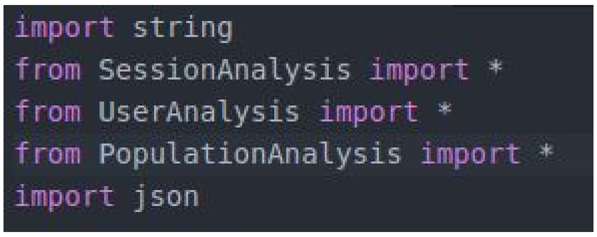
- **storeEvent**: takes a DataObject object and stores its contents in the Datastore.
- **deleteAll**: deletes all contents from the Datastore.
- **generateSessionData**: Returns a list of dictionaries. Each dictionary object contains the data for a Session entity in the Datastore.
- **findSession**: takes the basic information for a session (ID, starting time, end time) and searches the Datastore for the Session entity. If the Session exists it is returned, if not returns NULL.
- **generateUsersList**: returns a list of all user ID's stored per Session entity.
- **findSessionForUser**: takes a user's ID and returns all Session entities with the corresponding ID associated with it.
- **convertItemToDict**: takes any KeyboardEvent or MouseEvent object and converts it into a dictionary object. Exported for use by other scripts.

View

Below is a description of the components which make up the view of the MVC paradigm. The view group of Python scripts is the largest of the MVC components and is made up of multiple individual scripts for each type of analysis.

View.py

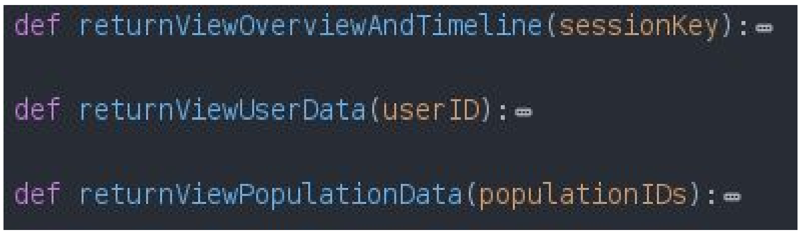
View.py is the primary point of entry to the view component. View.py provides the interface by which the controller sends data to be processed before it is sent back to be displayed on the front-end. View.py imports the supporting analysis Python scripts SessionAnalysis.py, UserAnalysis.py and PopulationAnalysis.py. Each of these scripts will be discussed below in further detail. Finally, the Python Standard Library module "json" is imported to convert the output to JSON objects the controller returns to the front-end.



```
import string
from SessionAnalysis import *
from UserAnalysis import *
from PopulationAnalysis import *
import json
```

Figure 26: View.py imports.

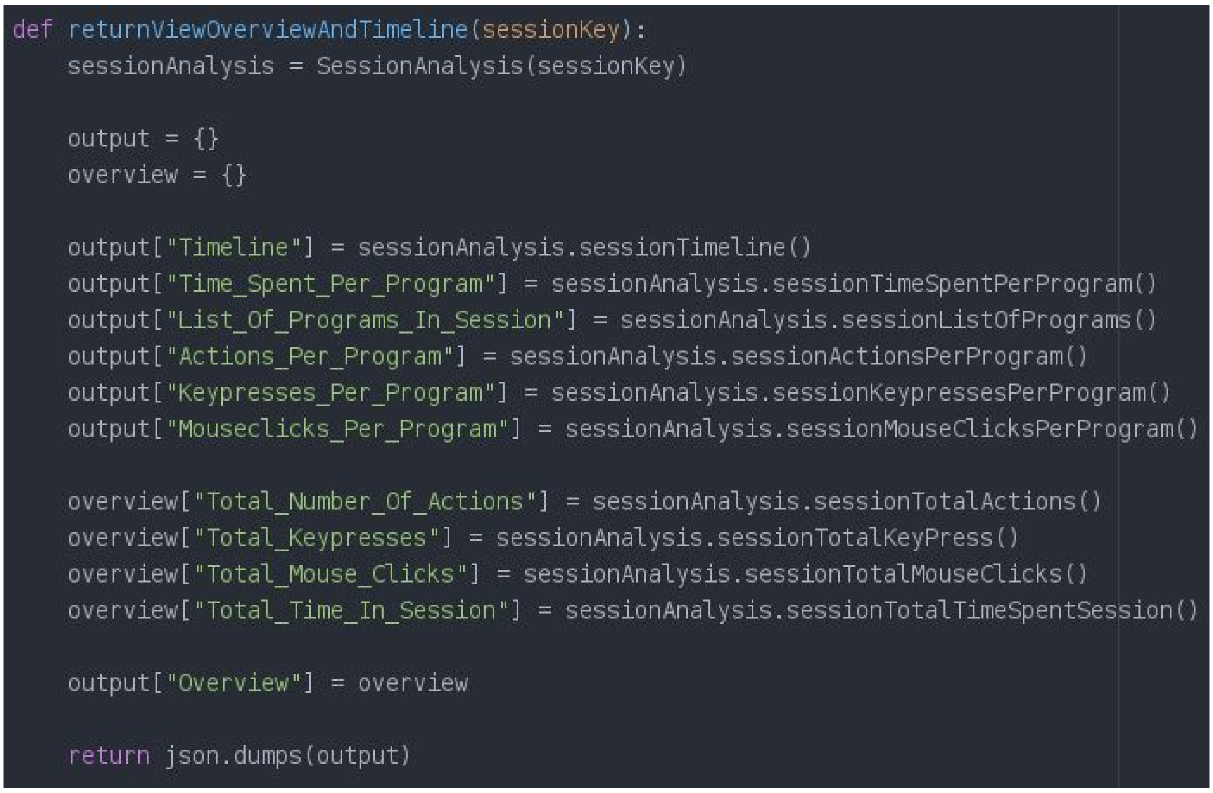
View.py offers three different functions. Each function conducts analysis on the Datastore data with a different resolution. "returnOverviewAndTimeLine" is used to generate the data for a given session. "returnViewUserData" is used to generate the data for a given user ID (i.e. a perspective of all the sessions for an individual user). "returnViewPopulationData" is used to generate the data for the entirety of the user populace, it gives a perspective of how all users compare to each other.



```
def returnViewOverviewAndTimeline(sessionKey):

def returnViewUserData(userID):

def returnViewPopulationData(populationIDs):
```

Figure 27: View.py methods.

Each of the above functions are used to package the data into a JSON object interpretable by the front-end JavaScript. Below is an example of what each function looks like. They are all similar in design. Each function constructs a dictionary object from the analysis conducted by the SessionAnalysis/UserAnalysis/PopulationAnalysis objects declared and instantiated at the beginning of each function. The resultant dictionary object is then processed by the "json" module into a JSON object string representation to be sent back by the controller.

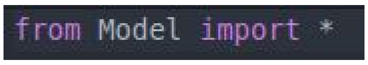


```python
def returnViewOverviewAndTimeline(sessionKey):
    sessionAnalysis = SessionAnalysis(sessionKey)

    output = {}
    overview = {}

    output["Timeline"] = sessionAnalysis.sessionTimeline()
    output["Time_Spent_Per_Program"] = sessionAnalysis.sessionTimeSpentPerProgram()
    output["List_Of_Programs_In_Session"] = sessionAnalysis.sessionListOfPrograms()
    output["Actions_Per_Program"] = sessionAnalysis.sessionActionsPerProgram()
    output["Keypresses_Per_Program"] = sessionAnalysis.sessionKeypressesPerProgram()
    output["Mouseclicks_Per_Program"] = sessionAnalysis.sessionMouseClicksPerProgram()

    overview["Total_Number_Of_Actions"] = sessionAnalysis.sessionTotalActions()
    overview["Total_Keypresses"] = sessionAnalysis.sessionTotalKeyPress()
    overview["Total_Mouse_Clicks"] = sessionAnalysis.sessionTotalMouseClicks()
    overview["Total_Time_In_Session"] = sessionAnalysis.sessionTotalTimeSpentSession()

    output["Overview"] = overview

    return json.dumps(output)
```

Figure 28: View.py method example.

SessionAnalysis.py

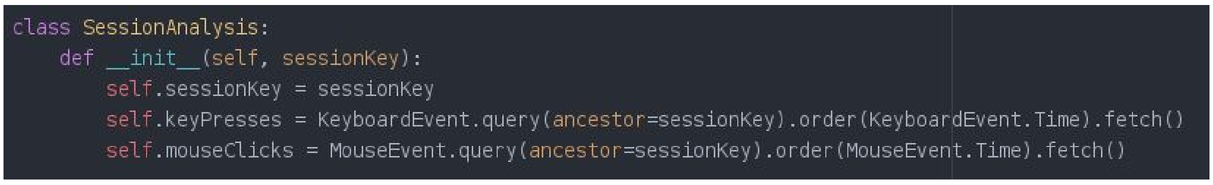
SessionAnalysis.py conducts all analysis for an individual Session entity. SessionAnalysis objects are constructed using the data from the Datastore for a given Session entity key. Model.py is imported to facilitate this process.



```python
from Model import *
```

Figure 29: SessionAnalysis.py imports.

Each SessionAnalysis object is instantiated with the key for the Session, the KeyboardEvents and the MouseEvents for the Session.

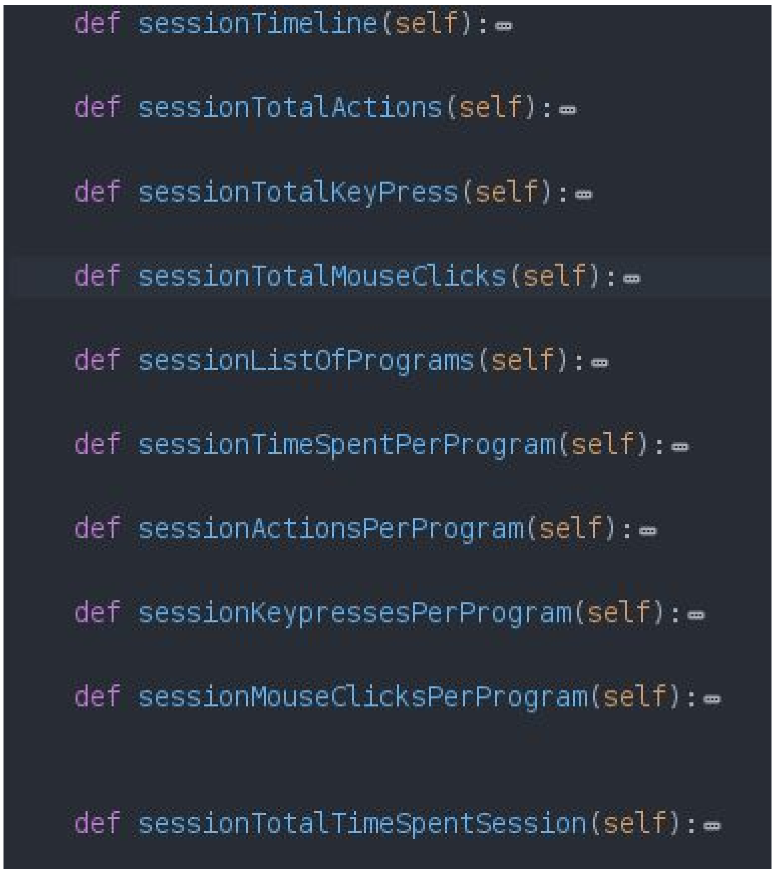


```python
class SessionAnalysis:
    def __init__(self, sessionKey):
        self.sessionKey = sessionKey
        self.keyPresses = KeyboardEvent.query(ancestor=sessionKey).order(KeyboardEvent.Time).fetch()
        self.mouseClicks = MouseEvent.query(ancestor=sessionKey).order(MouseEvent.Time).fetch()
```

Figure 30: SessionAnalysis.py class constructor.

SessionAnalysis objects provide multiple functions which extracts different features of the session. Below is a summary of each function and the information it extracts:

- sessionTimeline: returns the events which were recorded for the user’s session in chronological order.
- sessionTotalActions: returns the total number of actions (events) which were recorded during the session.
- sessionTotalKeyPress: returns the total number of KeyboardEvents (i.e. the total number of keys pressed) during the session.
- sessionTotalMouseClicks: returns the total number of MouseEvents (i.e. the number of mouse clicks) recorded during the session.
- sessionListOfPrograms: returns the names of each of the program interacted with during the session. Each event has an associated “WindowName” from which this can be determined.
- sessionTimeSpentPerProgram: returns the amount of time spent on each program as a sum of the starting and ending times of each action which is associated with that program.
- sessionActionsPerProgram: returns the sum of all the actions (both MouseEvents and KeyboardEvents) which are associated with each program.
- sessionKeypressesPerProgram: returns the sum of all the KeyboardEvent actions which are associated with each program.
- sessionMouseClicksPerProgram: returns the sum of all the MouseEvent actions which are associate with each program.
- sessionTotalTimeSpentSession: returns the total time spent in the session formatted into “HH:MM:SS” as a string.

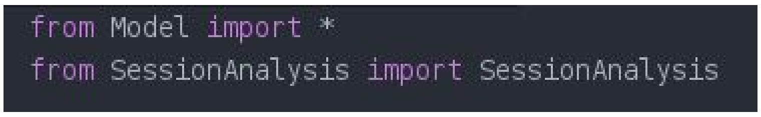


```
def sessionTimeline(self):

def sessionTotalActions(self):

def sessionTotalKeyPress(self):

def sessionTotalMouseClicks(self):

def sessionListOfPrograms(self):

def sessionTimeSpentPerProgram(self):

def sessionActionsPerProgram(self):

def sessionKeypressesPerProgram(self):

def sessionMouseClicksPerProgram(self):


def sessionTotalTimeSpentSession(self):
```

Figure 31: SessionAnalysis.py methods.

UserAnalysis.py

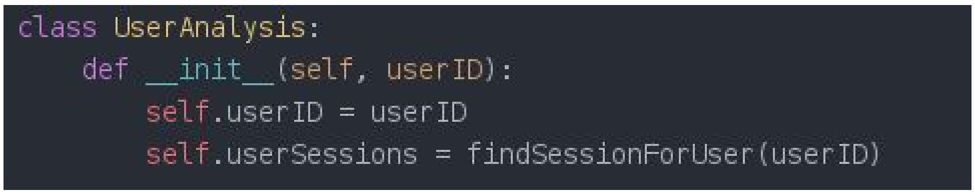
UserAnalysis.py conducts all analysis for events generated by a specific user from all of their sessions. To facilitate this, SessionAnalysis is imported to provide the analysis for each session associated with the user's ID. Model.py is imported again to provide access to the Datastore as needed.



```python
from Model import *
from SessionAnalysis import SessionAnalysis
```

Figure 32: UserAnalysis.py imports.

UserAnalysis objects are constructed and initialised with the user's ID value and all the Session entities associated with that ID. Below we can see the Model.py function "findSessionForUser" used to collect all the Session entities from the Datastore.



```python
class UserAnalysis:
    def __init__(self, userID):
        self.userID = userID
        self.userSessions = findSessionForUser(userID)
```

Figure 33: UserAnalysis.py class constructor.

UserAnalysis objects provide multiple functions to give an average value for many attributes calculated for a given user's session. Below is a summary of each functions:

- timeSpentPerSession: returns the time spent per session.
- timeSpentPerDay: returns the time spent per day from the very first session recorded for a user.
- timeSpentPerProgram: returns the time spent per program for all the programs seen to be used by a user across all of their sessions.
- numberOfKeystrokesPerSession: returns the sum of KeyboardEvents (i.e. key presses/ keystrokes) recorded per session.
- numberOfActionsPerSession: returns the sum of all actions recorded from each of the user's sessions (i.e. both MouseEvents and KeyboardEvents).
- numberOfClicksPerSession: returns the sum of all MouseEvent actions recorded from each of the user's sessions (i.e. mouse clicks).
- numberOfProgramsUsedPerSession: returns the sum of programs used as recorded from each of the user's sessions.

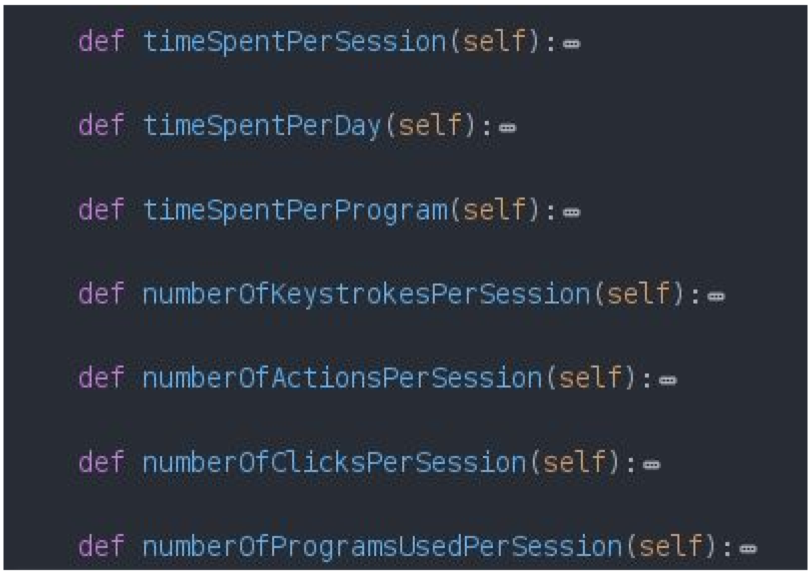

```
def timeSpentPerSession(self):

def timeSpentPerDay(self):

def timeSpentPerProgram(self):

def numberOfKeystrokesPerSession(self):

def numberOfActionsPerSession(self):

def numberOfClicksPerSession(self):

def numberOfProgramsUsedPerSession(self):
```

Figure 34: UserAnalysis.py methods.

PopulationAnalysis.py

PopulationAnalysis.py conducts all analysis for the entire population of users generating data using the client. Values generated by this script make use of the values generated by UserAnalysis for each user ID recorded in the Datastore. To facilitate this UserAnalysis is imported. Each user's sessions are averaged over all their sessions such that they can be compared between users.

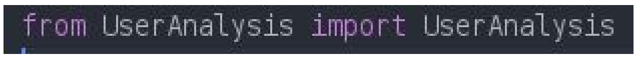

```
from UserAnalysis import UserAnalysis
```

Figure 35: PopulationAnalysis.py imports.

PopulationAnalysis objects are constructed and initialised with a list of all IDs which make up the current population of the Datastore.

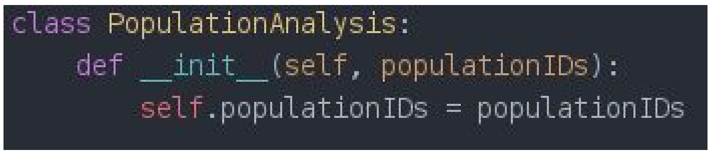
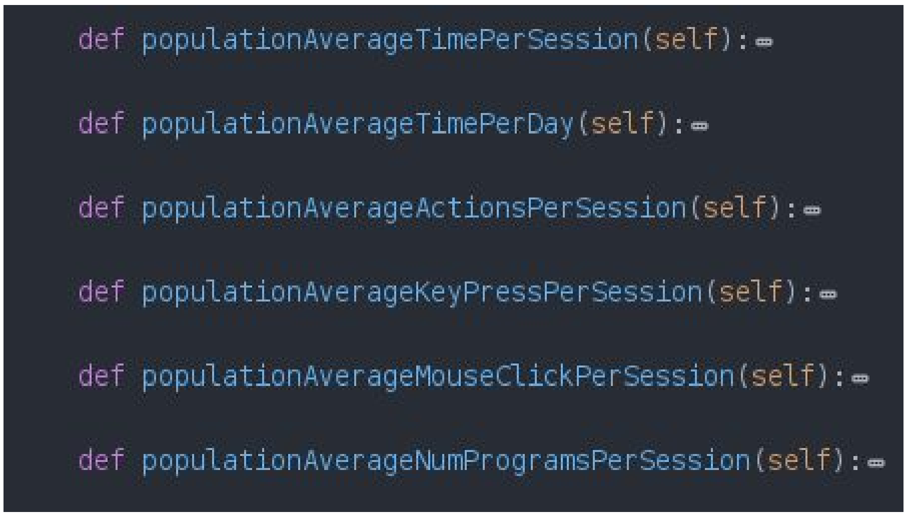

```
class PopulationAnalysis:
    def __init__(self, populationIDs):
        self.populationIDs = populationIDs
```

Figure 36: PopulationAnalysis.py class constructor.

PopulationAnalysis objects provide multiple functions for generating averaged values for each user in the population. Below is a summary of each method:

- populationAverageTimePerSession: returns a dictionary of users and their average time spent per session.
- populationAverageTimePerDay: returns a dictionary of users in the population and the average time spent per day for each user.
- populationAverageActionsPerSession: returns a dictionary of users in the population the average actions (MouseEvents or KeyboardEvents) recorded for each user's sessions.

- populationAverageKeyPressPerSession: returns a dictionary of users in the population and the average number of KeyboardEvent actions (keypresses) recorded for their sessions.
- populationAverageMouseClickPerSession: returns a dictionary of users in the population and the average number of MouseEvent actions (i.e. mouse clicks) recorded for their sessions.
- populationAverageNumProgramsPerSession: returns a dictionary of users in the population and the average number of programs used for their sessions.



```
def populationAverageTimePerSession(self):

def populationAverageTimePerDay(self):

def populationAverageActionsPerSession(self):

def populationAverageKeyPressPerSession(self):

def populationAverageMouseClickPerSession(self):

def populationAverageNumProgramsPerSession(self):
```

Figure 37: PopulationAnalysis.py methods.

Controller

Below is a description of main.py which is the script acting as the Controller in the MVC paradigm used in this project. All requests and responses handled by the web application are processed by main.py.

main.py

Main.py implements the functionality supplied by the Flask web application framework. Flask provides the means of allocating different methods to handle different request using decorators such as “@app.route(...)”. These decorators allow for the specification of what kind of HTTP methods are taken for a given URL (i.e. ‘POST’ or ‘GET’). Additionally, Flask uses templates as a means of structuring the pages to be rendered for the web application front-end. In order to implement this flask is added to the list if imports at the top of the Python script. Main.py also imports the Model.py and View.py scripts discussed above to interface with the other components in the web application. Finally, the Python Standard Library modules “json” and “requests” are imported to provide the functionality needed for processing incoming and outgoing requests/responses into the suitable forms for the rest of the application.

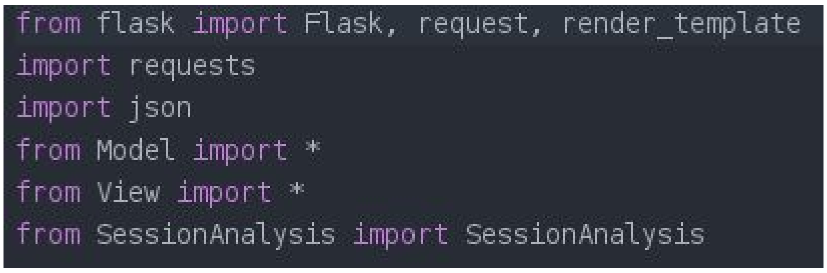

```
from flask import Flask, request, render_template
import requests
import json
from Model import *
from View import *
from SessionAnalysis import SessionAnalysis
```

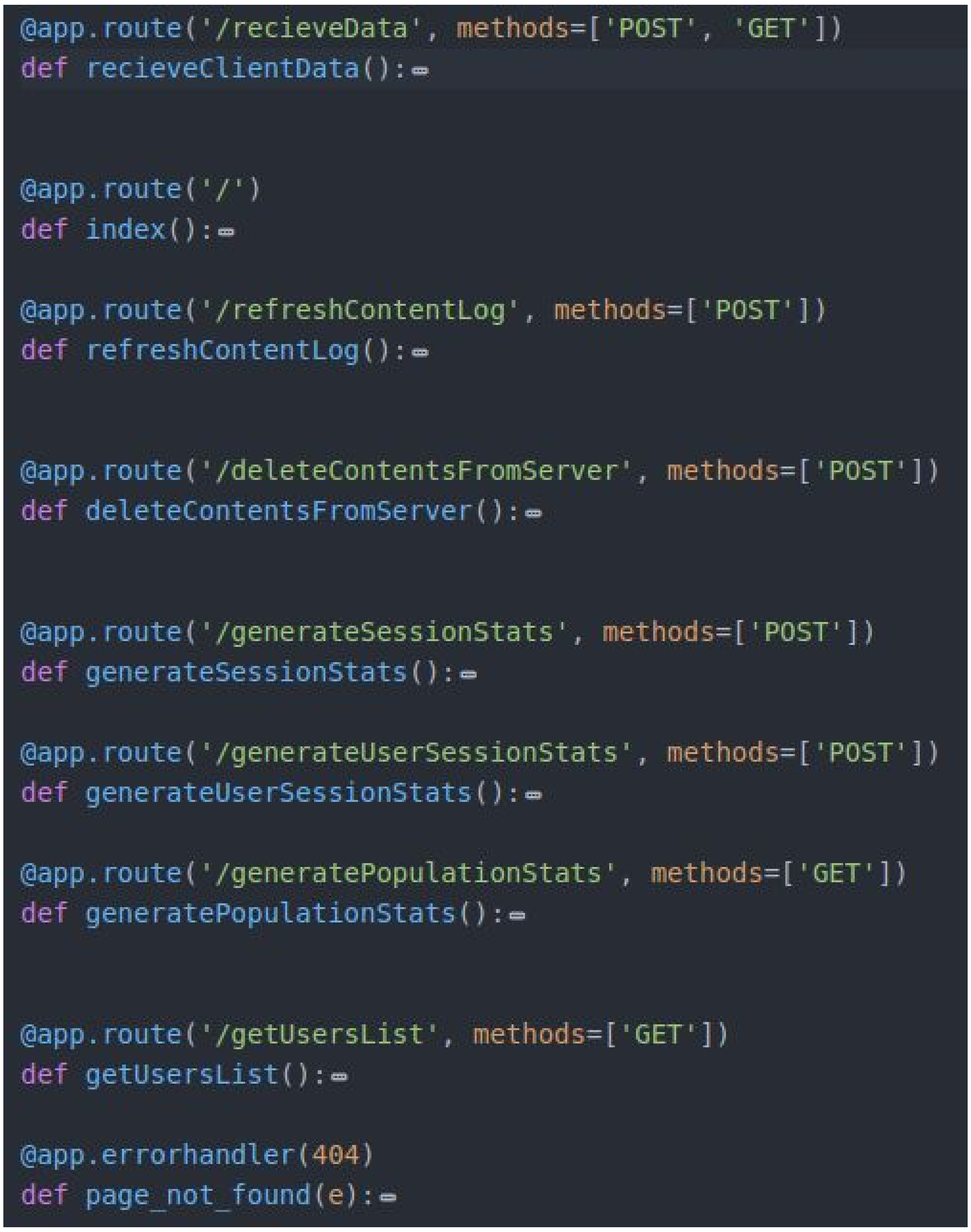
Figure 38: main.py imports.

Below is a summary of the methods defined to handle each of the requests accepted by the web application. Each of the methods names are self explanatory. The decorators used by each defines the URL suffix which the method handles and the HTTP methods which it will accept. For example:

```
app.route('/refreshContentLog', methods=['POST'])
Def refreshContentLog():
```

Decorates the refreshContentLog() function, specifying that it will handle requests to the URL suffix '/refreshContentLog' and will only accept HTTP POST methods. Similarly, the following decorator specifies the method to handle 404 errors.

```
@app.errorhandler(404)
def page_not_found(e):
```



```
@app.route('/recieveData', methods=['POST', 'GET'])
def recieveClientData():


@app.route('/')
def index():

@app.route('/refreshContentLog', methods=['POST'])
def refreshContentLog():


@app.route('/deleteContentsFromServer', methods=['POST'])
def deleteContentsFromServer():


@app.route('/generateSessionStats', methods=['POST'])
def generateSessionStats():

@app.route('/generateUserSessionStats', methods=['POST'])
def generateUserSessionStats():

@app.route('/generatePopulationStats', methods=['GET'])
def generatePopulationStats():


@app.route('/getUsersList', methods=['GET'])
def getUsersList():

@app.errorhandler(404)
def page_not_found(e):
```

Figure 39: main.py event handler methods.

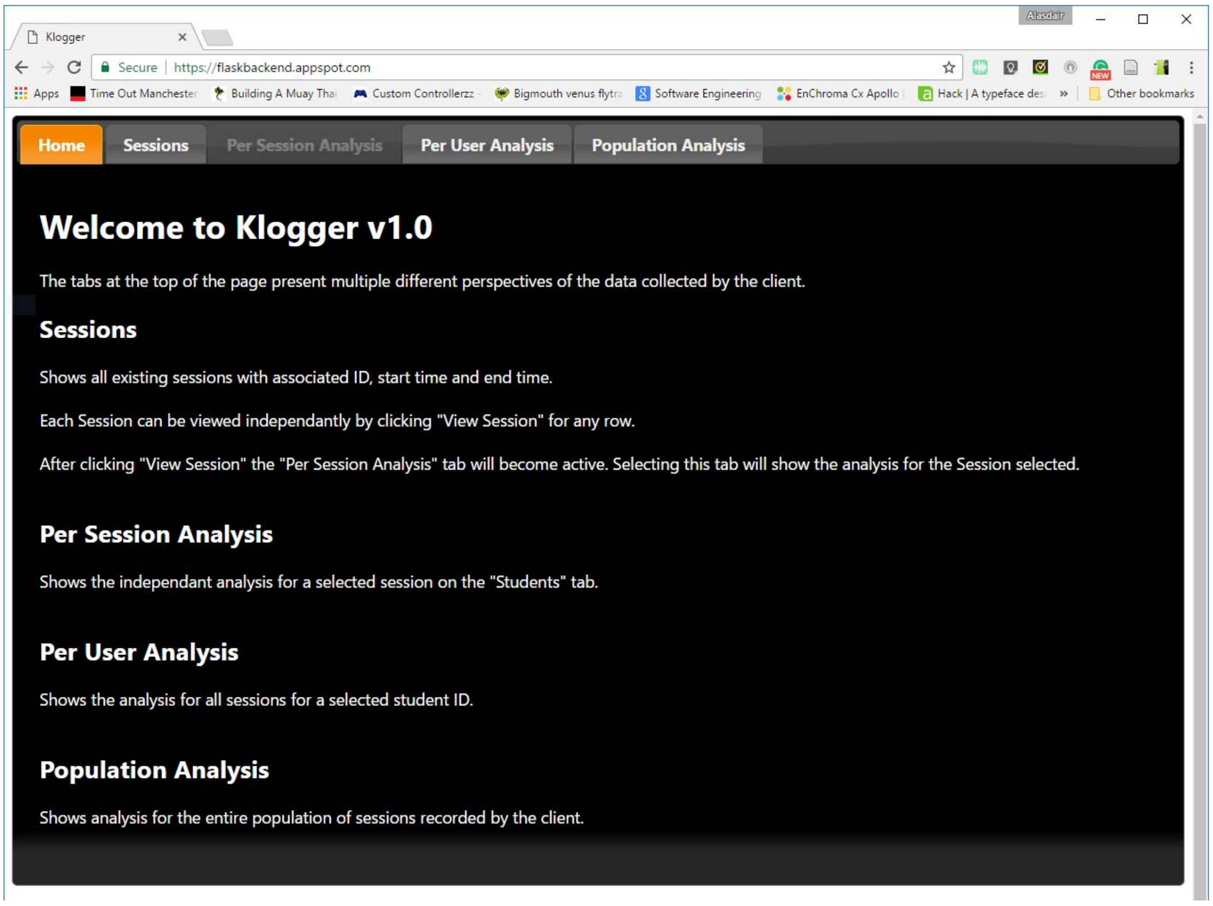
User Interface

In a following section is a description of the web application's user interface. The functionality of each aspect of the interface is discussed in each section along with a brief explanation of its potentially use by academic staff using the system.

Klogger's web application interface is collected into a single page with multiple tabs across the top of the page. Each tab shows a different perspective of the data collected by the client.

Home:

The Home tab is the first view seen by the user when requesting the viewing the root URL "https://flaskbackend.appspot.com". Home contains purely textual content explaining the purpose of each tab.



Welcome to Klogger v1.0

The tabs at the top of the page present multiple different perspectives of the data collected by the client.

Sessions

Shows all existing sessions with associated ID, start time and end time.

Each Session can be viewed independantly by clicking "View Session" for any row.

After clicking "View Session" the "Per Session Analysis" tab will become active. Selecting this tab will show the analysis for the Session selected.

Per Session Analysis

Shows the independant analysis for a selected session on the "Students" tab.

Per User Analysis

Shows the analysis for all sessions for a selected student ID.

Population Analysis

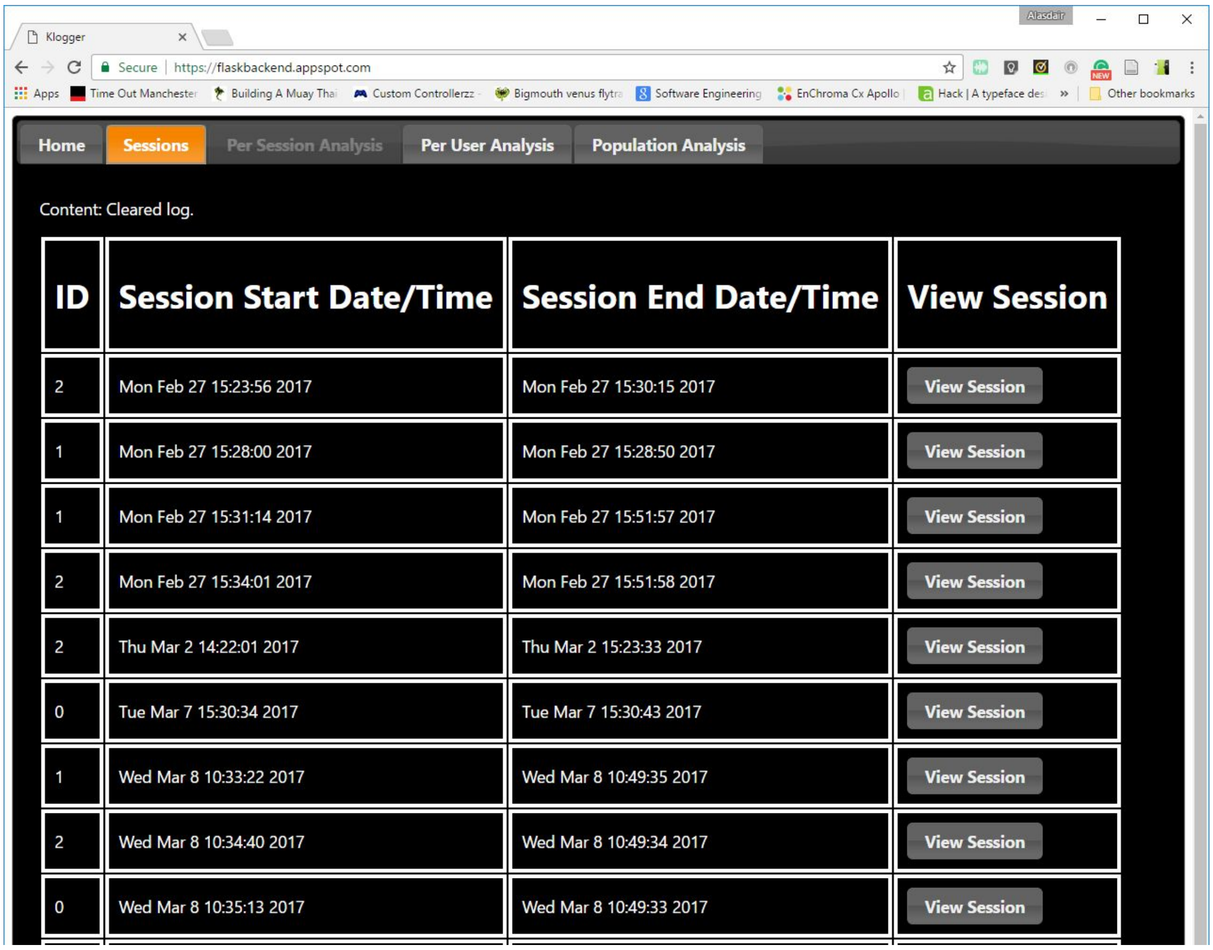
Shows analysis for the entire population of sessions recorded by the client.

Figure 40: Klogger web application Home tab.

Sessions:

The Sessions tab presents the user with a list of session. Each session has the associated user ID, start time and end time. Each session can be analysed independantly by clicking its corresponding "View

Session” button in the row. One clicked analysis will be conducted on the requested session’s recorded data and the Per Session Analysis tab will become active to browse.



Content: Cleared log.

ID	Session Start Date/Time	Session End Date/Time	View Session
2	Mon Feb 27 15:23:56 2017	Mon Feb 27 15:30:15 2017	View Session
1	Mon Feb 27 15:28:00 2017	Mon Feb 27 15:28:50 2017	View Session
1	Mon Feb 27 15:31:14 2017	Mon Feb 27 15:51:57 2017	View Session
2	Mon Feb 27 15:34:01 2017	Mon Feb 27 15:51:58 2017	View Session
2	Thu Mar 2 14:22:01 2017	Thu Mar 2 15:23:33 2017	View Session
0	Tue Mar 7 15:30:34 2017	Tue Mar 7 15:30:43 2017	View Session
1	Wed Mar 8 10:33:22 2017	Wed Mar 8 10:49:35 2017	View Session
2	Wed Mar 8 10:34:40 2017	Wed Mar 8 10:49:34 2017	View Session
0	Wed Mar 8 10:35:13 2017	Wed Mar 8 10:49:33 2017	View Session

Figure 41: Klogger web application Sessions tab.

Per Session Analysis:

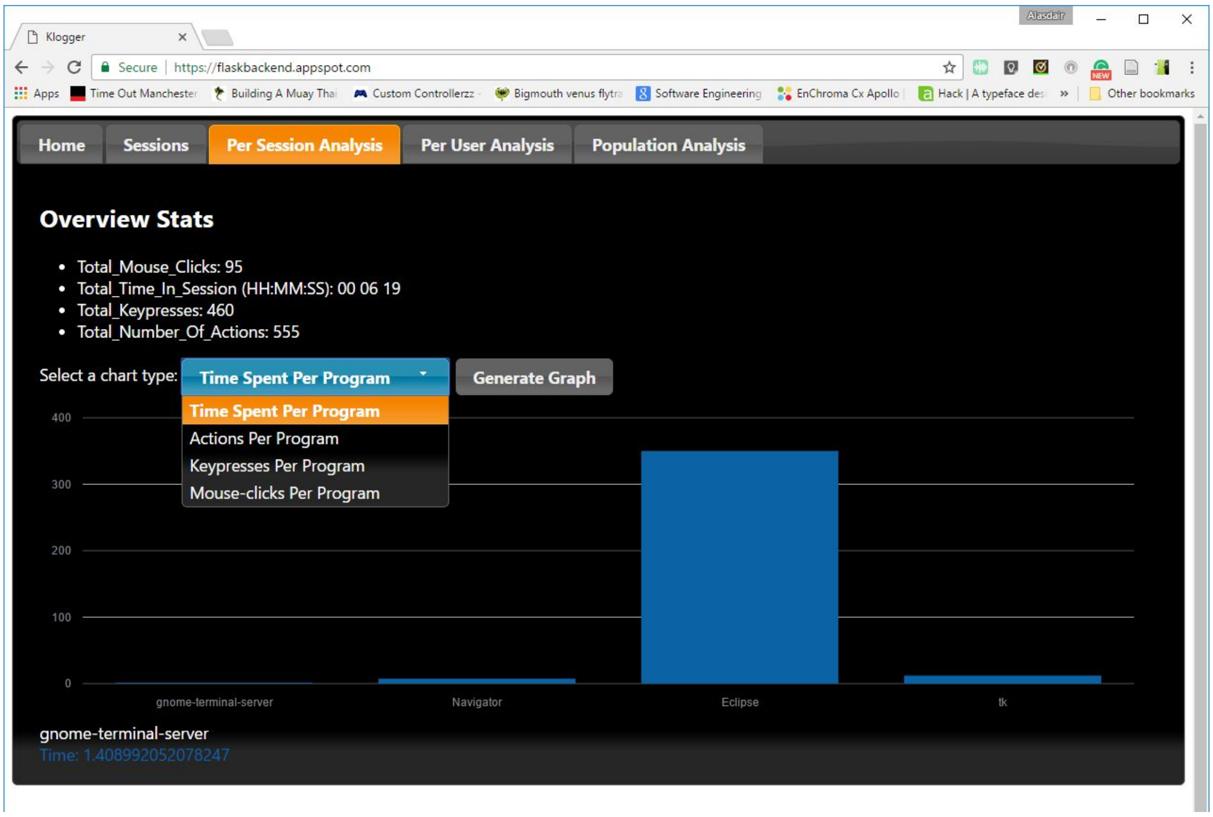
The Per Session Analysis tab will only become active once a session has been selected for viewing from the Sessions tab. Once a session has been selected its data will be analysed by the application and the results displayed in the Per Session Analysis tab content area.

Per Session Analysis gives the user a summary of the recorded data for that session and allows the user to represent the data graphically. By selecting a graph from the dropdown menu and clicking the “Generate Chart” button a bar chart will be generated from the session data and displayed beneath.

The following bar chart graphs are currently implemented:

- Time Spent Per Program.
- Actions Per Program.
- Keypresses Per Program.
- Mouse-clicks Per Program.

Each graph generated shows the different programs used during the session on the X-axis and the relevant attribute on the Y-axis (e.g. time spent, number of actions etc).



Overview Stats

- Total_Mouse_Clicks: 95
- Total_Time_In_Session (HH:MM:SS): 00 06 19
- Total_Keypresses: 460
- Total_Number_Of_Actions: 555

gnome-terminal-server
Time: 1.408992052078247

Figure 42: *Klogger web application Per Session Analysis tab.*

Per User Analysis:

The Per User Analysis tab offers the user the means to look at all the sessions associated with a specific client ID. The tab content area allows the user to select an ID from an ordered dropdown list. After selecting the ID of interest the user can choose the bar chart graph to display in the content area.

The following bar chart graphs are currently implemented:

- Time Spent Per Session
- Time Spent Per Day
- Time Spent Per program
- Number Of Keystrokes Per Session
- Number Of Actions Per Session
- Number Of Clicks Per Session
- Number Of Programs Used Per Session

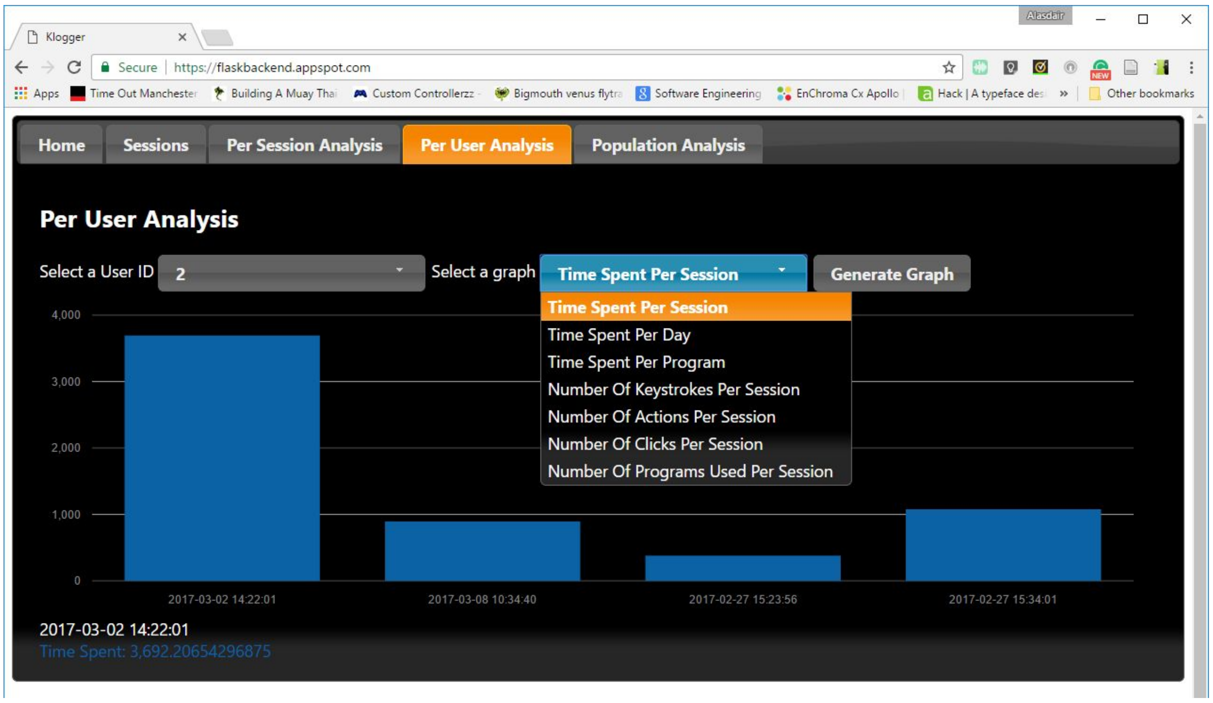


Figure 43: Klogger web application Per User Analysis tab.

Population Analysis:

The Population Analysis tab offers the a means to compare all the user ID's in the population. All values calculated for each ID in the Population Analysis tab are averaged values from all of the individual's sessions. The UI for the Population Analysis tab allows the user to select from a dropdown menu multiple different graphs. Below is a list of currently implemented graphs. Each graph is a line graph with the client ID along the x-axis and the corresponding value on the y-axis.

- Average TIme Per Session
- Average Time Per Day
- Average Actions Per Session
- Average Keystrokes Per Session
- Average Clicks Per Session
- Average Number Of Programs Used Per Session

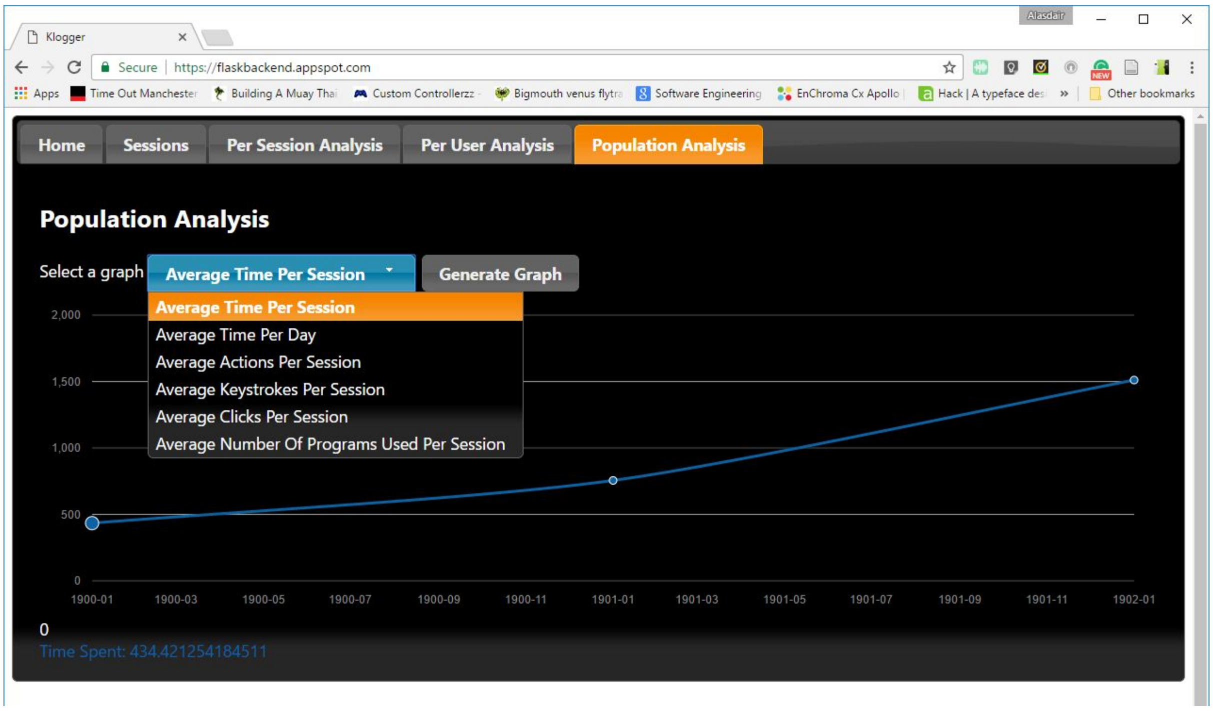


Figure 44: Klogger web application Population Analysis tab.

Testing

Testing for the server was a very iterative process. Testing occurred as and when new features were added to the solution. In truth, asides from the initial design, the development process was very fluid and dynamic more akin to a prototyping development model. As a result of this projects being developed by a single individual the testing/development cycle occurred iteratively in quick succession. Having a tight feedback loop meant that flaws/errors/bugs in the code did not get left long. Instead, as a new features was added to the code-base it was tested as a result of the development process itself. Further development would not take place until the last addition was seen to be in working order.

Despite this, it is well acknowledged that the solution requires additional testing to take place for it to be ready to go live as a public facing product. The researcher creating the solution does not consider himself to be a professional developer and as such would insist on a more structured development/testing process if the system were to be implemented in real-world conditions.

In addition, it is recognised that the development/testing process used for this solution is not ideal. Frequently during development a new feature would be added to the solution only to later discover that it does not work as expected/desired and will need to be remove/refactored. As a result the development was quite exhausting. The same solution could have been developed in a far more efficient manner with additional developers and a more concrete end-goal in mind. However, the nature of this project was such that it wasn't necessarily fully known what the resultant solution would look like. A degree of trial and error was required to assess the real functional requirements of the solution

Known Issues/Future Improvements:

- Input is not sanitised. In a final release of the software unit tests should be conducted on input processing to establish the validity of the data being submitted.
- Morris.js (the library used for the generation of chart output) is an unsuitable. The axes on graphs aren't always correct and the use of a better more fully functioning library would help resolve this issue.
- Datastore isn't necessarily the most appropriate method of storing the data. More exploration into the different storage options available should be conducted.
- No user login/logout functionality is implemented for the web application. This has security implications and is a critical feature to be implemented for the solution to be viable for public release.
- Client data stored in the Datastore is currently not scrambled/encrypted. Some form of reversible encoding should be implemented.
- The machine learning model explored in the following section is currently not incorporated into the web application due to resource constraints. Implementing this would provide a useful feature for staff.
- Storage and transmission of large JSON objects is resulting in unresponsive behaviour for the client/server communication. A form of compression should be implemented to reduce this processing overhead.
- Adherence to MVC model discussed above is loose. A full refactoring of the code-base should take place to better shape the code into a more logical structure.
- If the choice to implement the server using Google's Cloud platform is made further exploration of the services available would be recommended. The Google Cloud platform offers far more services than used in this solution, many of which may be better suited.
- Flask's use should be reviewed. Currently the framework is not used to it's full potential. Flask offers excellent opportunities for refactoring the structure of the code into a more organized layout.

Machine Learning Component

Applying machine learning to the problem domain proved to be challenging and not entirely representative of a solution applicable in the real world. It was by far the most experimental aspect of the project and required considerable assumptions to be made with regards to the scenario of the application. For these reasons, the resultant solution is considered representative of what could be achievable but not a final solution by any means.

Data Generation

As discussed in the Machine Learning Component section in the design chapter, before any algorithms could be used the data would need to be generated for a suitable sample size. Two sets of data were generated in total, one for each of the models. The first set of data would be used to test the ability of the selected algorithms to assign an engagement metric to a given student based on their recorded data. The second set of data would be used to train and test the ability of the algorithms to detect the potential for plagiarised content based off of a similar set of data. Both sets of data used the attributes discussed in the Data Generation design section as base-line measurements. For each model more specific metrics would need to be extracted for the training and testing process.

Model 1:

Model one required the extraction of a metric which represented the percentage of "engagement" achieved by the student during the working duration as specified in the Scenario. To establish this metric each parameter was plotted onto an Individual Value Plot.

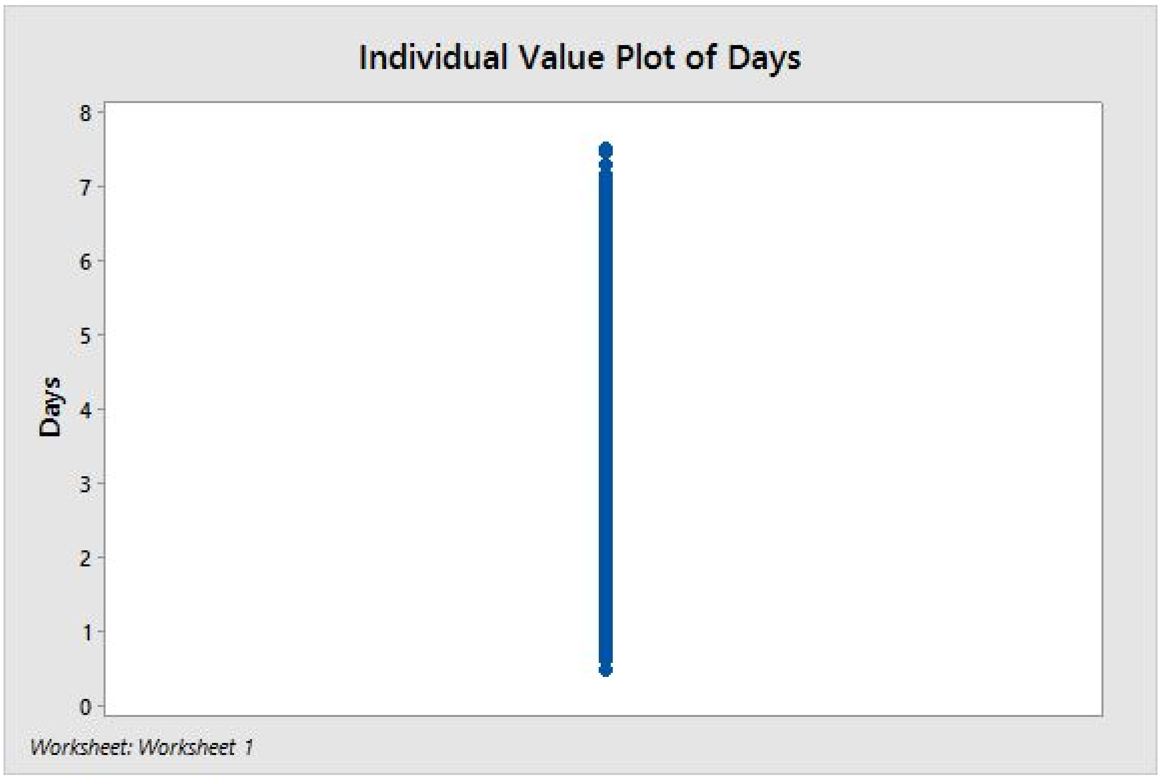
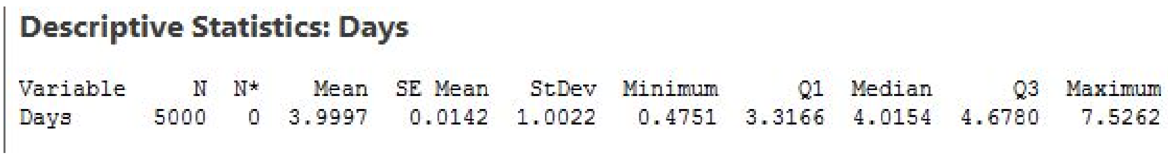


Figure 45: Individual Value Plot of the Days attribute.

Statistics were then generated for each attribute.

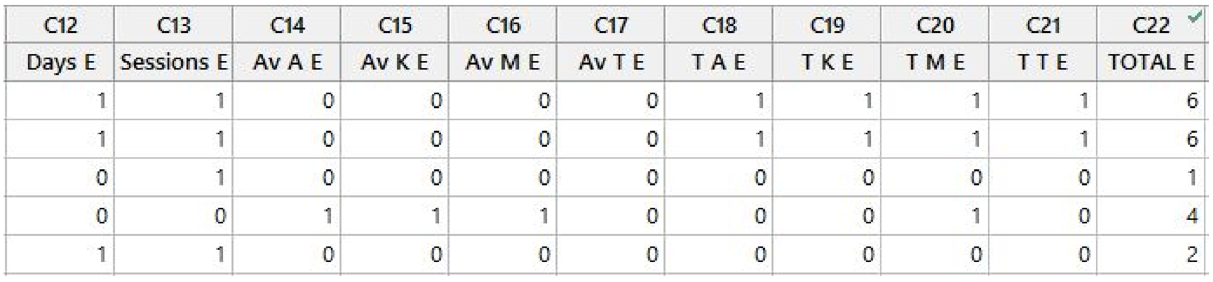


Descriptive Statistics: Days

```
Variable     N  N*    Mean  SE Mean   StDev  Minimum      Q1  Median      Q3  Maximum
Days      5000   0  3.9997   0.0142  1.0022   0.4751  3.3166  4.0154  4.6780   7.5262
```

Figure 46: Basic statistics for the Days attribute.

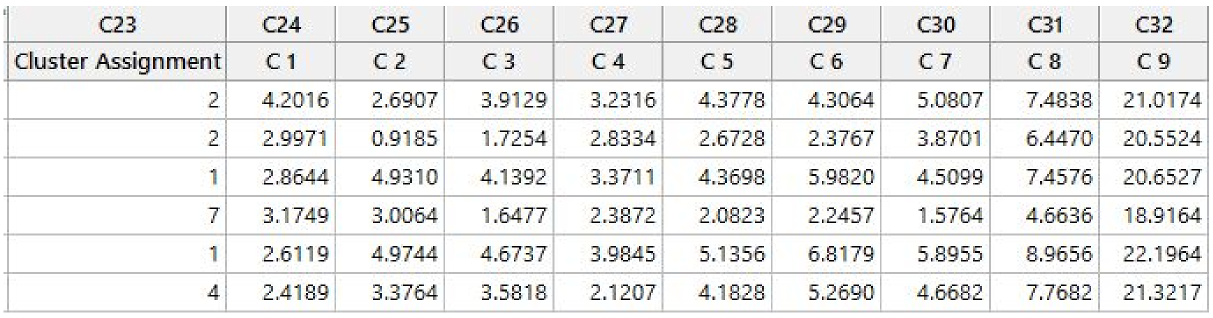
Based on the Mean value of each attribute, all instances which were below the mean were identified and given a ‘0’ value in a new column. All instances whose corresponding attribute value were higher than the mean were given a ‘1’ in the new column. This process was repeated for each Attribute until a table of binary values was generated. An additional column was then added which would contain the sum of all the values generated for each attribute, these column values are a representation of the effort of each instance.



C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22
Days E	Sessions E	Av A E	Av K E	Av M E	Av T E	T A E	T K E	T M E	T T E	TOTAL E
1	1	0	0	0	0	1	1	1	1	6
1	1	0	0	0	0	1	1	1	1	6
0	1	0	0	0	0	0	0	0	0	1
0	0	1	1	1	0	0	0	1	0	4
1	1	0	0	0	0	0	0	0	0	2

Figure 47: Weighted effort scores.

Using this ‘TOTAL E’ value (seen above) to represent the effort of each instance, the K-Means clustering algorithm was run against the set to generate nine clusters for the data (one for each potential effort value). The distance of each instance to each of these clusters was stored along with the final classification assigned by K-Means analysis.



C23	C24	C25	C26	C27	C28	C29	C30	C31	C32
Cluster Assignment	C 1	C 2	C 3	C 4	C 5	C 6	C 7	C 8	C 9
2	4.2016	2.6907	3.9129	3.2316	4.3778	4.3064	5.0807	7.4838	21.0174
2	2.9971	0.9185	1.7254	2.8334	2.6728	2.3767	3.8701	6.4470	20.5524
1	2.8644	4.9310	4.1392	3.3711	4.3698	5.9820	4.5099	7.4576	20.6527
7	3.1749	3.0064	1.6477	2.3872	2.0823	2.2457	1.5764	4.6636	18.9164
1	2.6119	4.9744	4.6737	3.9845	5.1356	6.8179	5.8955	8.9656	22.1964
4	2.4189	3.3764	3.5818	2.1207	4.1828	5.2690	4.6682	7.7682	21.3217

Figure 48: Cluster analysis.

Finally, a percentage score was generated using the following function for each instance. This would be the representation of the percentage engagement metric used later in the regression analysis for the k-NN and SVM regression algorithms. Below is the calculation used to determine the percentage engagement.

$$\{i \mid 0 < i \leq 5000\}$$

$$Engagement_i = \frac{E_i(D_iS_i)}{max(E(DS))} \times 100$$

E,D,and S represent the sets of values for all instances for total effort weight, days worked and average sessions per day, respectively.

The subscript *i* in this case represents an individual value from a set of E,D or S. The subscript *i* is any value in the range 1 to 5000 (i.e. the total number of instances in the set.

The percentage engagement is a score calculated from the individual's total effort weight multiplied by the number of days worked and the average number of sessions per day. Each instance is divided by the largest value E(DS) for the set of instances. This calculation aims to represent the effort of an individual over the duration of the working period and calculate a percentage in relation to the largest effort seen in all the instances. It is recognized that this formula is not necessarily the best representation of "engagement" but it was the best calculation achievable by the researcher given their limited mathematical knowledge.

Model 2:

The data for Model 2 was generated based off the resultant data from Model 1. In addition to the above features, Model 2 used a similar weighting system (as was used to determine effort in Model 1) to weight the likelihood of plagiarism.

To generates the weights for potentially plagiarised content, a scatter graph was created mapping the grade achieved against the total calculate effort weight. The grades were assigned randomly to each instance according to the normal distribution shown below.

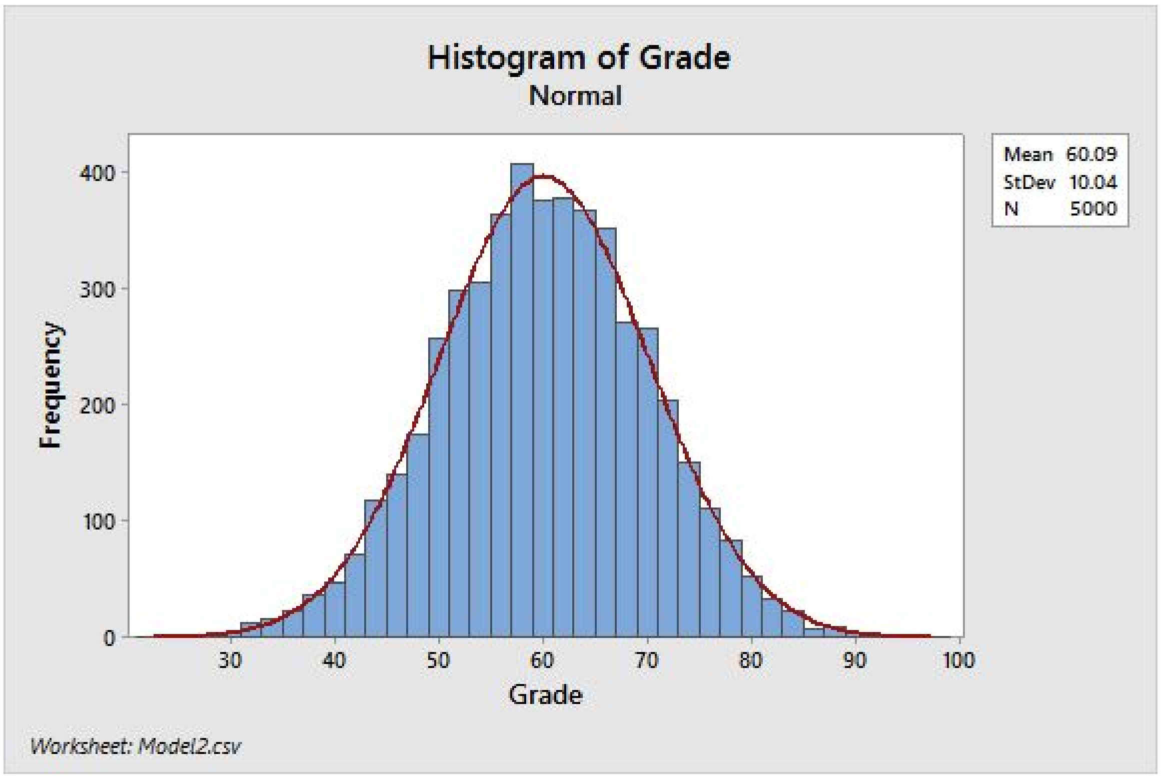


Figure 49: Distribution of Grade attribute.

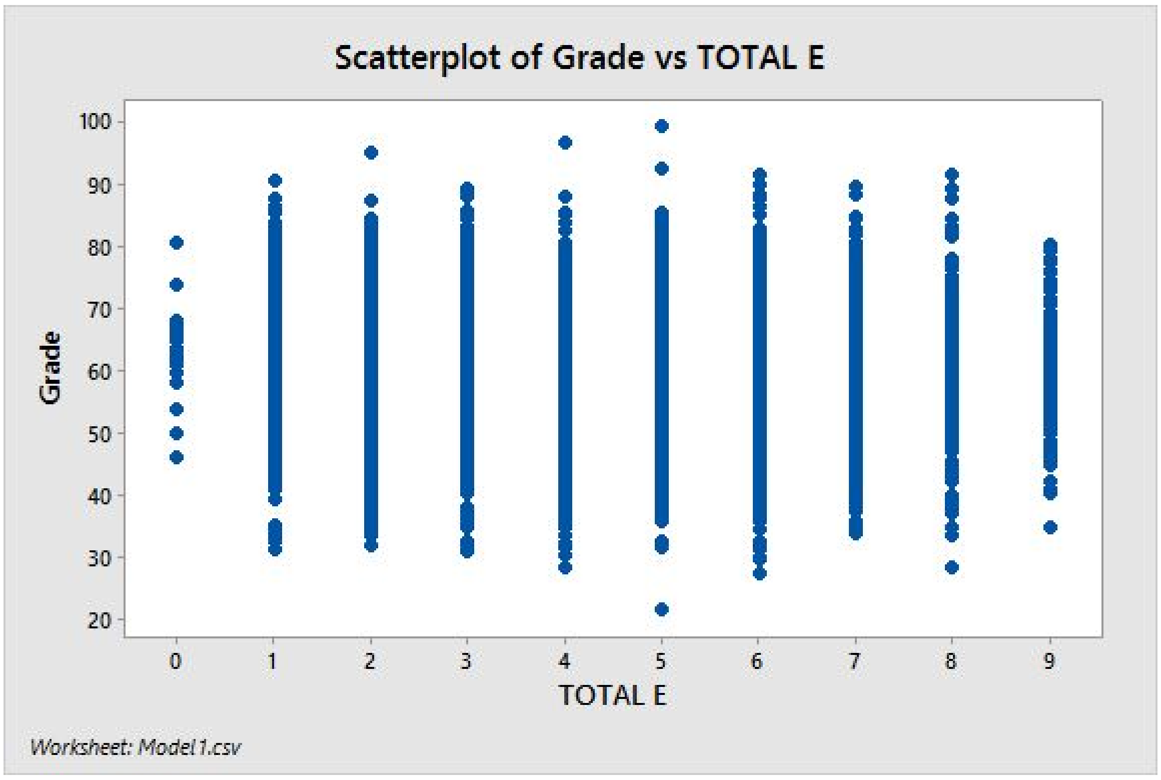


Figure 50: Scatter plot of Grade vs TOTAL E attributes.

The resulting scatter plot was divided into 5 areas. Each area was given a weighting. This weighting represented the degree to which plagiarism would be expected for an instance in the quadrant. For example, those instances which lie in the top left quadrant are expected to be more likely plagiarised than those in the bottom left because they show less effort for a higher grade. The instance will carry the weight of any categorie/s it fell into. For the other categories it would have a zero value. This was summarised in a table.

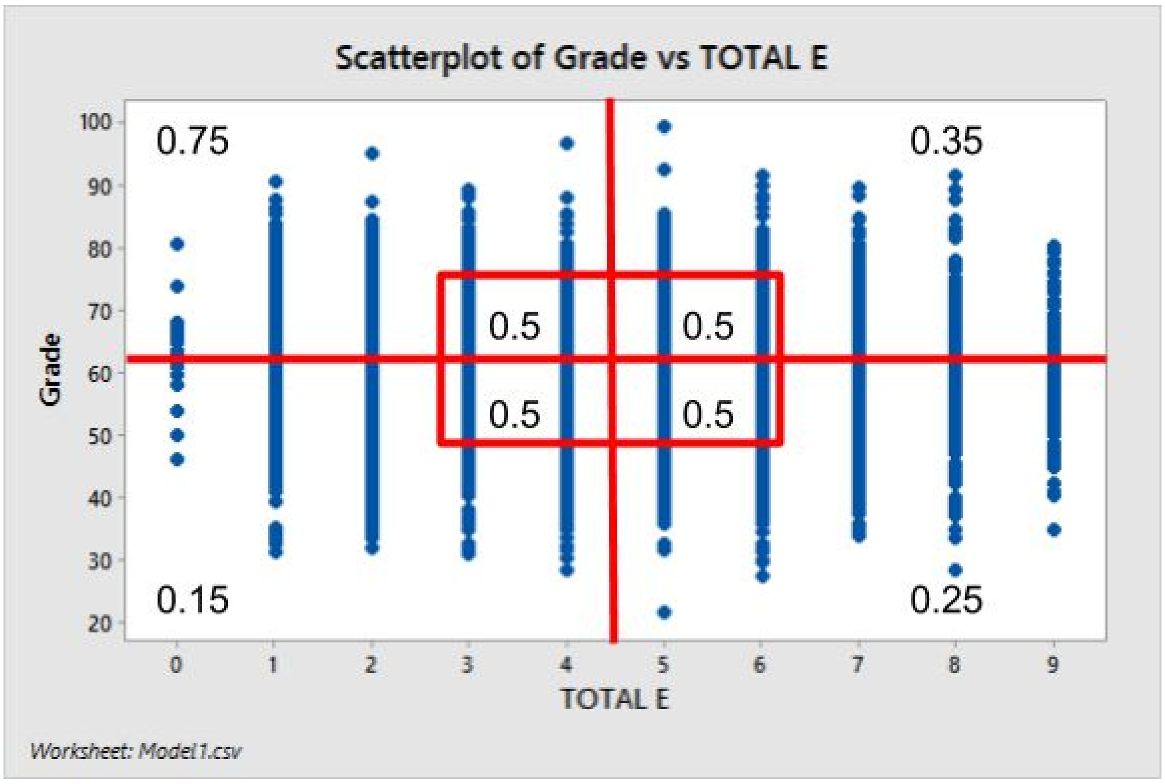
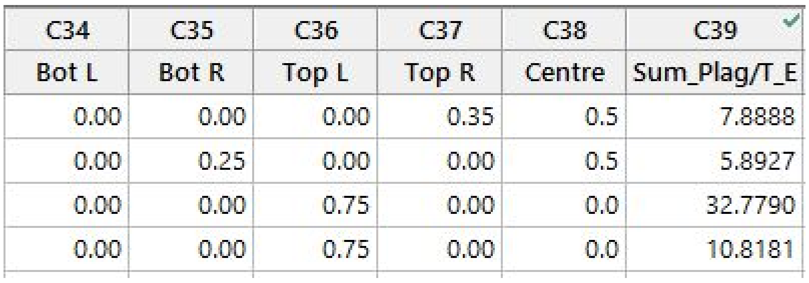


Figure 51: Demonstration of plagiarism weighting.



C34	C35	C36	C37	C38	C39
Bot L	Bot R	Top L	Top R	Centre	Sum_Plag/T_E
0.00	0.00	0.00	0.35	0.5	7.8888
0.00	0.25	0.00	0.00	0.5	5.8927
0.00	0.00	0.75	0.00	0.0	32.7790
0.00	0.00	0.75	0.00	0.0	10.8181

Figure 52: Plagiarism weights.

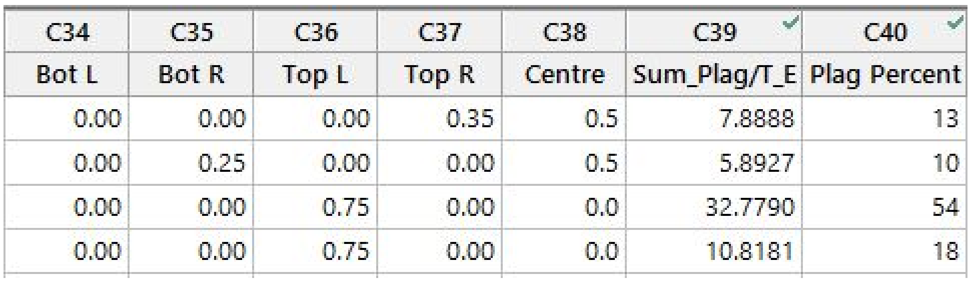
The total weight value in column C39 (above) is calculated using the following function:

$$\{i \mid 0 < i \leq 5000\}$$

$$W_i = (a_i + b_i + c_i + d_i + e_i)\frac{G_i}{E_i}$$

For the above function, G and E are the set of all grades assigned to each instance and the set of all total effort weights assigned to each instance (as per Model 1). In each case the subscript *i* refers to a single instance in the set where *i* is an value between 1 and 5000 (i.e. the number of instances).

The percentage likelihood of plagiarism is calculated using the weights in the calculation below and added to the final column.



C34	C35	C36	C37	C38	C39	C40
Bot L	Bot R	Top L	Top R	Centre	Sum_Plag/T_E	Plag Percent
0.00	0.00	0.00	0.35	0.5	7.8888	13
0.00	0.25	0.00	0.00	0.5	5.8927	10
0.00	0.00	0.75	0.00	0.0	32.7790	54
0.00	0.00	0.75	0.00	0.0	10.8181	18

Figure 53: Plagiarism percentage score.

$$\{i \mid 0 < i \leq 5000\}$$

$$Plagiarised_i = \frac{W_i \div E_i}{max(W \div E)} \times 100$$

In the function above W and E represent the set of weights calculated for each instance and the set of total effort weights calculated for each instance, accordingly. In each case the subscript *i* refers to a single instance in the set where *i* is a value between 1 and 5000 (i.e. the number of instances).

Both sets of data can be seen in full in the Appendices.

kNN

The machine learning algorithm selected for this solution was the k-Nearest Neighbour algorithm. kNN isimplemented and tested using Weka. Weka's implementation of kNN is called "IBk" (*IBk*, n.d.). These names are interchangeable for the purpose of this solution.

Testing

The parameters altered for the kNN algorithm were:

- The *k* number of nearest neighbours to use for regression.
- The distance weighting used.

The *k* value for the algorithm was established using the “crossValidate” option in Weka’s algorithm configurator. “crossValidate” will iterate through the values 1 to the *k* value supplied and identify the best performing value for the algorithm given the data set.

The distance weighting used was changed between the three available value “No distance weighting”, “Weight by 1/distance” and “Weight by 1-distance”. Once the optimum *k* value was established each of these value was used in turn to see if there was an noticeable improvement in performance. Below is a table summarizing the tests conducted for each data model:

Test Number	Data Model Tested	Identified Optimum Value of *k*	Distance Weighting Used	Time to build (s)	Time to evaluate (s)	Root Relative Squared Error %
1	Model 1	3	None	0	2.25	20.4464
2	Model 1	3	1/Distance	0	2.08	19.841
3	Model 1	3	1-Distance	0	2.09	20.3858
4	Model 2	2	None	0	2.76	16.6955
5	Model 2	2	1/Distance	0	2.83	15.6345
6	Model 2	2	1-Distance	0	2.66	16.6955

Table 1: Results for kNN algorithm for data Model 1 and data Model 2.

Results

From the above table it can be seen that the highest performing configuration of the algorithm for Model 1 was test two. kNN was configured with a *k* value of 3 using inverse distance weighting. With this configuration kNN achieved a Root Relative Squared Error of 19.841%. Applying kNN to Model 2 showed that a configuration using a *k* value of 2 and inverse distance weighting achieved a Root Relative Squared Error of 15.6345%. In both cases inverse distance weighting offered the most noticeable change in performance given that the value of *k* was constant for each model. It is interesting that for Model 1 the test two’s configuration offered both the fastest evaluation time and the lowest error. In Model 2 the same configuration offered the lowest error but took longer to evaluate than all the other tests. Unsurprisingly, in all tests the time taken to build the model was a constant 0. This is expected with the kNN algorithm as it has no specific training phase due to it’s instance-based learning approach.

Future Work

While the study conducted above was interesting and possibly representative of what may be achievable using the metrics identified, it is almost certainly not representative of the accuracy of the same classifiers on real-world data.

Future work would likely require a larger more representative set of data. Furthermore, in the case of identifying the percentage potential for plagiarism in Model 2, a representative set of samples of known plagiarised work would be required. This could be difficult to collect as an individual is unlikely to report if they have plagiarised especially when they are aware of the recording process.

Provided a better sample of data was available it would be interesting to see how a reinforcement learning algorithm would perform. The iterative process used in reinforcement learners would be extremely useful for creating a more flexible solution capable of making more general observations. The solution as implemented above is heavily scenario based and quite specific to this individual case. A reinforcement learner may be able to better generalise and give further insight into the working behaviour of student over a longer period of time.

It would be useful for the broader scope of the project if the process used above was incorporated into the web application for additional forms of output available to the academic staff upon review. Currently, the cost of the resources (server processing speed, storage for data etc.) was too high for it to be a viable option. With more financial support to acquisition the necessary resources this would a very useful feature to incorporate into the web application.

Finally, the metrics used for measuring the engagement of students and the percentage chance of plagiarism occurrence are not likely to be the most accurate measures. More research should be conducted into this area to find a better metric.

Evaluation

In the following section a balanced evaluation of the final solution proposed by this document will take place. Initially, a broad overview of the final solution will take place with an assessment of its achievements and failures with regard to the Solution Functional Requirements section proposed at the beginning of the document. After this a more detailed discussion of each subcomponent will take place. Again, for each component an assessment will be made in relation to their proposed requirements in the respective design chapter.

Overview

The final solution proposed and implemented in this document, as a means of monitoring student engagement, showed strong potential. The software developed by the researcher (while only a prototype) suggests that using technology for this purpose is a viable solution.

It is fully acknowledged by the researcher that the solution is far from finished. By no means can the software produced during this study be considered fit for purpose. Its design and implementation are not to industry standard and would require additional development and exploration. Furthermore, the application of machine learning to the problem domain is still a relatively new area of research in the academic community. Any research conducted here using machine learning as applied to learning analytics or plagiarism detection, only serves to support previous work and acts as a demonstration of what may be possible.

Of critical importance is the improvement of security across all components. In it's current state the recorded actions of users are not readable by anyone who does not have direct access to the Datastore control panel on the Google Cloud platform. Additionally, any data collected is stored securely to industry standards by Google and any information which was recorded during the course of this study was not sensitive. However, due to the requirements of the Data Protection Act (*Data Protection Act 1998*, n.d.), any formal release of this software would require a significant security audit and development of appropriate security mechanisms to be conducted.

Client

A recurring theme within this evaluation is that the software developed during this study is prototypal in nature. This holds true for the client as it does for the other components.

It is the opinion of the researcher that the client achieved the vast majority of the functional goals defined in the design chapter. Specifically, a solution was developed which was capable of capturing all keystrokes and mouse activity using keylogging software. The client requires little user interaction asides from the provision of a user ID and initiation of the software. Finally, the data recorded by the client is collected and transmitted to a central server for storage and further processing. In addition to this, the client is lightweight showing no noticeable interruption to the working environment of the student and has the potential for platform independent use.

Despite these achievements, the client is still only considered a working prototype for the reasons discussed below. Firstly, the biggest flaw observed during testing was that the client could become slow and unresponsive after the user clicks the button to stop the recording process and the transmission of the recorded data takes place. This behaviour has been identified as resulting from the client sending vast quantities of recorded data in uncompressed form. This problem is further compounded by the lack of multi-threading (a feature not fully incorporated into the Python language), the use of which would prove useful in creating a more responsive solution.

Other flaws were detected subsequent to the completion of the development and testing phases. The final version of this software proves problematic in a far less obvious way. Its recording process is very generalised. The client will record data about the working behaviour of students regardless of what they are working on. It is assumed and taken in good faith, that when the client is being used the student will only be working on academic content but this cannot be proved. In reality the data recorded during a session may not be representative of their working behaviour.

Solving this particular problem could take one of two forms. Firstly, additional development could take place to create a modified form of keylogging library capable of better identifying the content a student is working on. Alternatively a method could be developed by which the prescribed assignment is inherently bound to the use of the client. A possible solution for the latter choice might be to pre-load the USB stick with the final document which the student is required to submit. Manipulation of the document will be restricted to the USB. As a result, work on the assignment will be bound to the USB and thus the running of the client.

Regardless of any additional development to improve the functionality of the client, it is evident that a viable solution would require a full security audit. Currently, the client is susceptible to tampering and reverse engineering. Development of a fully functional solution would become entirely irrelevant whilst it is possible for the user to submit any arbitrary data. The current version of the software has not been fully tested with information security in mind. Efforts have been made to ensure the communication of data makes use of secure channels. Unfortunately, a full security audit of the final product was outside the scope of the development process, for what is ultimately a prototype.

Server

Upon embarking on the development of the server component, it was clear to the researcher that the primary goal was not to create an interface which was absolutely intuitive and designed solely with perfect user interaction in mind. The front-end web application was designed to be usable and functional and it achieved these goals. Of primary concern was the server's ability to execute the functional tasks necessary for storing, analysing and presenting the recorded data in a manner accessible to academic staff. Again, the core functionality of the server achieved these goals. A software solution has been created which stores the data in a suitable form. The solution can process and analyse the collected data to a standard which provides insight for academic staff and is scalable to provide other statistics as required. Finally the web application can present this data in a user friendly manner and although it is not an entirely good user interface, it is usable for the purpose of this prototype.

Many of the shortcomings of the web application are due to the formatting of the output and the design of the UI. These are minor issues and could be resolved with the use of better libraries and additional development resources.

Other pertinent and critical issues, ones which require additional research and exploration, exist due to the structure of the code-base and the storage of the data. Given the time available for development, it is acknowledged that the frameworks and platforms used to implement the server are not used to their full ability. With more time available for development the Flask framework and Datastore could have been used to achieve better structure and organization of the code-base and stored data.

Flask and Google's Cloud platform offer many different features which could have been used to greater effect for this solution. Of particular note, is the variety of services offered by the Google Cloud platform. For this project only two of the Google Cloud platform's services were used: Datastore and App Engine. Datastore offered a very simple way to create a fully functioning data storage solution using NoSQL. App Engine made the process of acquiring a server and hosting a web application effortless. The range of services offered by the Google Cloud platform and the ease with which these services could be scaled, makes it an ideal candidate for developing a web application of this nature. Indeed, the Google Cloud platform offers two entire subcategories of products dedicated to Machine Learning and Big Data. While these other products were considered during the design stage, the time and resources required to utilize them made them inaccessible for this study. If the solution described in this study was to be implemented for real academic use, optimally, further exploration of these products would take place. It is likely that a combination of these other services and products would result in a solution offering better functionality, scalability and structure.

Finally, the lack of authentication needed to access the web application has considerable impact on the security of the final product. No authentication or authorization process opens the web application front-end for viewing by any party with knowledge of the application URL. In its current state, very little could be done to alter or read the raw data. However, it would be possible to delete the recorded data entirely from the Datastore. This is a significant shortcoming which should be addressed in any future revision.

Machine Learning Component

An evaluation of the Machine Learning Component of this project has proven difficult to conduct. The data model and algorithms used for this aspect of the study were applied in a very experimental manner. This section of the project left the most unanswered questions and it is difficult to know how appropriate the solution was, based on the results observed.

It is the researcher's opinion that the application of machine learning to the problem domain as seen in this study is an example of what may be possible. However, due to the use of untested and unproven metrics, combined with the generated data model used for training/testing, many questions are left unanswered and many results are left unsupported. It is clear that to prove the potential for machine learning to be applied in this manner, live data would need to be collected.

Collecting the type of data necessary for this type of study would be a problem deserving of its own project. For instance, if a study was conducted with a primary focus of recording data representative of different levels of student engagement, how would one define the level of engagement exhibited by each instance? It would seem unscientific to enquire of the participant “how engaged did you feel?”. Similarly, if a study was conducted to collect data contrasting plagiaristic and non-plagiaristic behaviour, how does one prove plagiarism has taken place? If some of the participants are asked to plagiarise content and some not to plagiarise content, it would be unlikely the data would be truly representative as there would be no consequences resulting from their actions.

A possible solution may be to implement a reinforcement learning algorithm. For any given academic submission there will be a minority of individuals who have plagiarised. Because of the low incidence of plagiarism, it would take a considerable amount of time to collect sufficient data before a regular supervised learning algorithm could be trained. Using a reinforcement learning algorithm instead, the solution could continue learning over time. Each new instance of plagiaristic behaviour would improve the algorithm's ability to detect similar behavioural patterns. Similarly, a reinforcement learning algorithm could be an appropriate way of training a solution to calculate student engagement metrics. Over time the algorithm would refine the behavioural profiles associated with students across the range of levels of engagement.

The k-Nearest Neighbour algorithm used in this project showed a satisfactory performance when applied to the regression task for both data models but the researcher is disinclined to put much faith in the results. kNN has been shown to be susceptible to inaccuracy when not provided with enough training data. As an instance-based learning algorithm, kNN is reliant almost entirely on the training set containing a representative sample for all the attributes. Given the way the data models were synthesised it is unlikely that the models are fully representative. This is likely to affect the validity of the results.

Conclusion

This project demonstrated a viable solution for monitoring student engagement through the use of keylogging software. Data which can be recorded from the keyboard and mouse inputs was shown to be relevant in monitoring the level of student engagement.

Future versions of the software developed in this study would benefit greatly from the refactoring and reorganisation of the code-base.Currently the software is not of industry standard however, this was not the primary objective of the project. Future versions written to industry standards with regards to both design and security would be better suited for commercial use.

Machine learning algorithms showed promise for future work in the quantification of a student engagement score as well as generating an indicator of possible plagiaristic behaviour. While the application of machine learning demonstrated in this study held potential, future research would benefit from the identification of more appropriate metrics and the collection of better representative data. The collection of such data will likely prove to be the most difficult element of future studies.

References

18.2. json — JSON encoder and decoder — Python 2.7.13 documentation (n.d.). [Online] [Accessed on 14th December 2016] https://docs.python.org/2/library/json.html.

24.6. IDLE — Python 2.7.13 documentation (n.d.). [Online] [Accessed on 23rd April 2017] https://docs.python.org/2/library/idle.html.

Apache Subversion (n.d.). [Online] [Accessed on 23rd April 2017] https://subversion.apache.org/.

App Engine - Build Scalable Web & Mobile Backends in Any Language (n.d.) Google Cloud Platform. [Online] [Accessed on 23rd April 2017] https://cloud.google.com/appengine/.

appJar (n.d.). [Online] [Accessed on 17th April 2017] http://appjar.info/.

Banerjee, R., Feng, S., Kang, J. and Choi, Y. (2014) 'Keystroke Patterns as Prosody in Digital Writings: A Case Study with Deceptive Reviews and Essays.' *In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Doha, Qatar: Association for Computational Linguistics.

Campbell, J. P., Oblinger, D. G. and others (2007) 'Academic analytics.' *EDUCAUSE review*, 42(4) pp. 40–57.

Cho, S., Han, C., Han, D. H. and Kim, H.-I. (2000) 'Web-based keystroke dynamics identity verification using neural network.' *Journal of organizational computing and electronic commerce*, 10(4) pp. 295–307.

Data Protection Act 1998 (n.d.). [Online] [Accessed on 23rd April 2017] http://www.legislation.gov.uk/ukpga/1998/29/contents.

Git (n.d.). [Online] [Accessed on 23rd April 2017] https://git-scm.com/.

Google Cloud Computing, Hosting Services & APIs (n.d.) Google Cloud Platform. [Online] [Accessed on 23rd April 2017] https://cloud.google.com/.

Google Cloud Datastore Overview | Cloud Datastore Documentation (n.d.) Google Cloud Platform. [Online] [Accessed on 23rd April 2017] https://cloud.google.com/datastore/docs/concepts/overview.

Google Cloud SDK Documentation | Cloud SDK (n.d.) Google Cloud Platform. [Online] [Accessed on 14th December 2016] https://cloud.google.com/sdk/docs/.

google.appengine.ext.ndb package | App Engine standard environment for Python (n.d.) Google Cloud Platform. [Online] [Accessed on 18th April 2017] https://cloud.google.com/appengine/docs/standard/python/refdocs/google.appengine.ext.ndb.

GoogleCloudPlatform/google-cloud-python (n.d.) GitHub. [Online] [Accessed on 23rd April 2017] https://github.com/GoogleCloudPlatform/google-cloud-python.

Heinrich, E. and Maurer, H. A. (2000) 'Active documents: Concept, implementation and applications.' *Journal of Universal Computer Science*, 6(12) pp. 1197–1202.

IBk (n.d.). [Online] [Accessed on 21st April 2017] http://weka.sourceforge.net/doc.dev/weka/classifiers/lazy/IBk.html.

JeffHoogland/pyxhook (n.d.) GitHub. [Online] [Accessed on 19th November 2016a] https://github.com/JeffHoogland/pyxhook.

JeffHoogland/pyxhook (n.d.) GitHub. [Online] [Accessed on 17th April 2017b] https://github.com/JeffHoogland/pyxhook/blob/master/example.py.

JeffHoogland/pyxhook (n.d.) GitHub. [Online] [Accessed on 17th April 2017c] https://github.com/JeffHoogland/pyxhook/blob/master/pyxhook.py.

jquery.org, jQuery F.- (n.d.) 'jQuery UI.' [Online] [Accessed on 23rd April 2017] https://jqueryui.com/.

Larose, D. T. (2005) *Discovering knowledge in data: an introduction to data mining*. Hoboken, N.J: Wiley-Interscience.

Maurer, H. A., Kappe, F. and Zaka, B. (2006) 'Plagiarism-A Survey.' *J. UCS*, 12(8) pp. 1050–1084.

Minitab Statistical Software - Minitab (n.d.). [Online] [Accessed on 3rd April 2017] http://www.minitab.com/en-us/products/minitab/?WT.srch=1&WT.mc_id=SE3926&gclid=Cj0KEQjw5YfHBRDzjNnioYq3_swBEiQArj4pdPEuF3oB3C4cQkm76ihQMe00dRrQDO1l0OOnyDyFIuUaAmUU8P8HAQ.

morris.js (n.d.). [Online] [Accessed on 23rd April 2017] http://morrisjs.github.io/morris.js/.

Parker, A. and Hamblen, J. (1989) 'Computer Algorithms for Plagiarism Detection.' *IEEE TRANSACTIONS ON EDUCATION*, VOL. 32(NO. 2.).

pyHook - Browse Files at SourceForge.net (n.d.). [Online] [Accessed on 17th April 2017] https://sourceforge.net/projects/pyhook/files/?source=navbar.

pyHook / Wiki / Main_Page (n.d.). [Online] [Accessed on 17th April 2017] https://sourceforge.net/p/pyhook/wiki/Main_Page/.

Quickstart — Requests 2.12.1 documentation (n.d.). [Online] [Accessed on 19th November 2016] http://docs.python-requests.org/en/latest/user/quickstart/#passing-parameters-in-urls.

Romero-Zaldivar, V.-A., Pardo, A., Burgos, D. and Delgado Kloos, C. (2012) 'Monitoring student progress using virtual appliances: A case study.' *Computers & Education*, 58(4) pp. 1058–1067.

Sagiroglu, S. and Canbek, G. (2009) 'Keyloggers.' *IEEE Technology and Society Magazine*, 28(3) pp. 10–17.

Shalev-Shwartz, S. and Ben-David, S. (2014) *Understanding machine learning: from theory to algorithms*. New York, NY, USA: Cambridge University Press.

SMOreg (n.d.). [Online] [Accessed on 21st April 2017] http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/SMOreg.html.

Tschinkel, B., Esantsi, B., Iacovelli, D., Nagesar, P., Walz, R., Monaco, V. and Bakelman, N. (2011) 'Keylogger Keystroke Biometric System.' *Pace University, White Plains*.

Understanding k-Nearest Neighbour — OpenCV 3.0.0-dev documentation (n.d.). [Online] [Accessed on 11th April 2017] http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_ml/py_knn/py_knn_understanding/py_knn_understanding.html.

Virtualenv — virtualenv 15.1.0 documentation (n.d.). [Online] [Accessed on 17th April 2017] https://virtualenv.pypa.io/en/stable/.

Weka 3 - Data Mining with Open Source Machine Learning Software in Java (n.d.). [Online] [Accessed on 1st April 2017] http://www.cs.waikato.ac.nz/ml/weka/.

Welcome | Flask (A Python Microframework) (n.d.). [Online] [Accessed on 29th November 2016] http://flask.pocoo.org/.

Welcome to Python.org (n.d.) Python.org. [Online] [Accessed on 23rd April 2017] https://www.python.org/download/releases/2.7/.

What is Plagiarism? (n.d.) Plagiarism.org - Best Practices for Ensuring Originality in Written Work. [Online] [Accessed on 18th November 2016] http://www.plagiarism.org/plagiarism-101/what-is-plagiarism/.

X Window System Architecture Overview HOWTO (n.d.). [Online] [Accessed on 23rd April 2017] http://www.tldp.org/HOWTO/XWindow-Overview-HOWTO/index.html.

Bibliography

Chart.js | Open source HTML5 Charts for your website (n.d.). [Online] [Accessed on 17th April 2017] http://www.chartjs.org/.

How to autorun files and scripts in Ubuntu when inserting a USB stick like autorun.inf in Windows? - Ask Ubuntu (n.d.). [Online] [Accessed on 23rd April 2017] https://askubuntu.com/questions/642511/how-to-autorun-files-and-scripts-in-ubuntu-when-inserting-a-usb-stick-like-autor.

Raschka, S. (2015) *Python machine learning: unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics*. Birmingham: Packt Publishing (Community experience distilled).

Understanding k-Nearest Neighbour — OpenCV 3.0.0-dev documentation (n.d.). [Online] [Accessed on 11th April 2017] http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_ml/py_knn/py_knn_understanding/py_knn_understanding.html.

Shalev-Shwartz, S. and Ben-David, S. (2014) *Understanding machine learning: from theory to algorithms*. New York, NY, USA: Cambridge University Press.

Appendices

Appendix A:

TOR

Student Engagement Monitoring Using a Keylogger

Course Specific Learning Outcomes

As part of the project, it is the intention of the researcher to utilise standard database management software and database interrogation techniques to store and manipulate any data collected. Using existing DBMS software will provide an organised way of collecting the data needed for analysis as well as a standardised interface on which the project can be expanded at a later date. Furthermore, as a result of using standardised software, the solution will be more open and accessible to other researchers in the field.

In addition to the above, a degree of consideration must be given as per the elements of hardware to software interaction which will be necessary to complete the solution. The project requires that the collection of user input data is limited to the file system or applications stored on the USB stick on which the user's submission is written. This highlights the necessity for the solution to understand the distinction between software operating on files within its remit for collecting data and software which is operating outside of this boundary.

To transmit and store the data collected, a network connection will be created and used by implementing a basic client/server connection. This will allow academic staff to monitor the progress of the student remotely.

Finally, to ensure that all data collected is valid, a degree of attention will be paid to the security of the solution in its final form. For the solution to be considered successful, the data collected from the users must be reliable. As such, a level of tamper-proofing will be implemented to stop users from altering collected data from its true value. This final goal will require the project to be assessed from multiple attack vectors to determine its weak points and thus implement a suitable way to secure its design.

Project Background

Plagiarism has long been a problem since the academic world first blossomed on the face of the earth. Despite the academic environment now being in a full-fledged state, plagiarism still seems to plague the establishments. Many students seek to attain higher grades without putting in the required work, either taking to the internet in an attempt to surreptitiously copy another individual's work or to pay another to do the job for them. Efforts to detect this form of deception have had mixed results with many solutions having critical flaws which allow more industrious students to bypass the checks. Furthermore, with the number students attending academic establishments for further education reaching an all-time high, the amount of work to sift through to check for this deception manually has become insurmountable.

Current systems make use of large databases of the previous submissions and try to detect plagiarism by cross-checking existing works against the document submitted for checking. There are a couple of problems with this approach. Firstly, it is beatable. Secondly, it is fallible. Existing systems all tend towards a text analysis approach, taking the finished product, analysing it and comparing it via different means to existing works. Approaching the problem this way is becoming more and more difficult to do because the amount of work to cross-check against is increasing rapidly and because these techniques tend to work only in a subset of cases. For instance, some of these techniques work well for straightforward copy and paste style plagiarism. However, they do not function as well for plagiarism where the text is translated into another language but still passed off as work owned by the plagiariser.

This project attempts a new solution to detecting plagiarism. By making use of novel Machine Learning techniques, the solution will try to detect plagiarism as it is occurring instead of the current technique of trying to detect plagiarism after the fact. This will be achieved by keeping a log of what and when the writer is writing, in real time. This log will then be analysed by a program on a remote server using a Machine Learning based algorithm to determine the likelihood of plagiarism. Making use of a computer to analyse the work instead of a human will take considerably less time and allow academic staff to address more important issues instead of spending considerable time following up references and cross-checking material manually.

Aim

The aim of this project is to create a tool capable of detecting plagiarism in student submissions. The tool should be able to collect data on the way a student creates a piece of work and submit the data to a remote location for later inspection. From this data, inferences will be made using Machine Learning which allows academic staff to determine the likelihood that the work was either plagiarised or not plagiarised.

Objectives:

In the following section, a list of objectives are defined which the project should achieve to be considered successful:

1. Design a tool, resident on a USB drive, which can record the actions made by a user when writing a project/submission document.

2. Collect data using the aforementioned tool for later assessment.

3. Determine a suitable method by which the data can be securely sent across a network to a central server.

4. Assess data stored by the tool and determine relevant metrics by which plagiarism can be detected.

5. Develop a Machine Learning based algorithm which can observe metrics as defined in the previous objective to infer if plagiarism has occurred.

Problems:

Potential problems that are inherent to any project give the potential for the project to become stagnant at critical points during its course. The following section seeks to determine and outline these possible pitfalls for consideration.

1. Given the highly technical nature of this project, there is potential to allocate time disproportionately to certain areas and as a result, exceed the time-limit available for development. As a result, tremendous pressure would occur in the final stages of the project required for document preparation.

2. There is no guarantee that a Machine Learning based solution can be conceived from the data collected. Given the researcher's limited knowledge in the field of Machine Learning, at this time the investigator's skillset may not be finely tuned enough reach the desired outcome.

3. Machine Learning solutions require significant data to teach the solution what to look for or how to solve the problem for which it is designed. Given that the data the solution must operate on is unique to the design it is not absolute that the significant amount of data required can be collated within the time of the project.

4. There are ethical considerations to take account of before the project can take place. The observation of user interactions for the purpose of plagiarism detection is still an area unexplored by most existing solutions and may be considered a breach of personal privacy.

Required Resources

- Desktop computer running Linux/Windows/Mac OS X.
- Python.
- Necessary libraries for keylogging functionality.
- Necessary libraries for Machine Learning algorithms.

Timetable and Deliverables

Submit Terms of Reference	21/10/2016
Write Ethics Form	22/10/2016 - 26/10/2016 (Duration 4 days)
Submit Ethics Form	28/10/2016

Begin Literature Review	25/10/2016 - 04/11/2016 (Duration 9 days)
Write Literature Review	07/11/2016 - 11/11/2016 (Duration 5 days)
Submit Literature Review	21/11/2016
Development of Technical Solution	14/11/2016 - 30/21/2016 (Duration 35 days)
Write Product Design	21/11/2016 - 09/12/2016 (Duration 15 days)
Submit Project Design	12/12/2016
Majority of Technical Solution Completed	02/01/2017
Write Evaluation	03/01/2017 - 20/01/2017 (Duration 14 days)
Submit Evaluation	30/01/2017
Write Report Outline	06/02/2017 - 17/02/2017 (Duration 10 days)
Submit Report Outline	20/02/2017
Draft Slide Submission	06/03/2017
Practice Presentation	13/03/2017
Final Report/Product Submission	24/04/2017
Final Presentation	24/04/2017

The MANCHESTER METROPOLITAN UNIVERSITY

Faculty of Science and Engineering

RISK ASSESSMENT COVER SHEET

REFERENCE NUMBER: NPC / 171016 / JDE1.49

SCHOOL: Computing, Mathematics & Digital Technology

TITLE OF WORK: CMT Projects involving software development and acceptability testing

LOCATION OF WORK: John Dalton Building computing facilities, computers at student's own home etc.

INTENDED ACTIVITIES (attach methods sheets (e.g. standard operating practices) and work schedules to this form):

General use of computers to develop and test software, using other participants and feedback sheets / programs. Method sheets and work schedules not applicable.

PERSONS AT RISK (list names of all individuals (including status e.g. staff/student), and/or unit(s) / course(s) undertaking the activity. For students please indicate course and level, for staff give contact email / phone number):

Undergraduate students and the participants they recruit.

HAZARDS (provide a summary of the hazards anticipated and attach detailed assessments with appropriate risk control methods to this form):

Repetitive Strain Injury – work related upper limb disorder

Back injury resulting from improper posture

Eye strain

Fatigue Stress

Possible risk from 240v electrical mains supply

Normal dangers of moving about the University to attend testing sessions.

Are these hazards necessary in order to achieve the objectives of the activity?

Yes

Hazard Rating (delete as appropriate): **Low**

HAZARDOUS SUBSTANCES/MATERIALS USED AND HAZARD CLASSIFICATION

(appropriate COSHH data sheets / risk assessments must be attached to this form**):**

ALL CONTAINERS OF HAZARDOUS SUBSTANCES SHOULD BEAR CORRECT HAZARD WARNING LABELS.

NAME OF MATERIAL *Please provide also approximate quantity and concentration if applicable.*	**HAZARD CLASS**	**HAZARD LABEL**	**DISPOSAL** *Hazardous materials must not be removed from laboratories. List disposal arrangements for* all materials listed below *in the location where* the work will be

			carried out:

RISK CONTROL METHODS (provide a summary of the hazards anticipated and attach detailed assessments with appropriate risk control methods to this form):

The hazards identified above are controlled by:

Facilities review when laboratories are commissioned
Induction session on H&S given to students by Technical Services Manager
School H&S information given in Student handbook
Posters in laboratories
PAT testing of equipment after three years

Annual H&S inspections

The laboratory workstations, whilst not legally required to be DSE compliant, (the continuous usage is too low to present risk) are fully compliant with current legislation. Monitors and keyboards are adjustable, chairs are adjustable and the lighting designed for both computer usage and associated reading activity. In each laboratory, there is an adjustable desk, suitable for wheelchair users, usually located in the next to the door.

Hazard Rating with Control Methods (delete as appropriate): **Low**

Will any specific **training** *be required (if* YES *give details)?* N/A

Are there any specific **first aid** *issues (if* YES *give details)?* N/A

PROCEDURE FOR EMERGENCY SHUT-DOWN (if applicable):

In the event of fire, flood or other emergency, evacuation of the laboratory would take place and the technical staff would subsequently make an assessment of the necessity of switch-off. As overall system control is vested in a separate server room, there would be little physical harm to any device in directly cutting the power to the mains for each individual lab.

Re-start of the lab may present problems of a technical nature but would not affect the personal safety or health of any individual.

IF OFF-SITE INDICATE ANY OTHER ISSUES (e.g. associated with: individual's health and dietary requirements (obtain off-site health forms for all participating individuals and indicate where this information will be located); social activities, transportation, ID requirements; permissions for access and sampling).

Not applicable – this form applies only to the laboratories listed

	NAME	**STAFF/STUDENT No.**	**DATE**

Originator	Nicholas Costen	01900261	171016
Supervisor	N/A		
Technical Manager	David Higson	99901727	171016
Divisional / School **Health and Safety Coordinator (**p.p. HoS)		Digitally signed by Nicholas Costen DN: cn=Nicholas Costen, o=MMU, ou=SCMDT, email=n.costen@mmu.ac.uk, c=GB Date: 2016.10.18 12:51:08 +01'00'	

DATE TO BE REVIEWED BY: September 2017

Appendices (Other)

Klogger Client Code:

All code for the client component of this project can be found on GitHub repository with the URL below:

https://github.com/midmandle/Klogger

Klogger Server/Web application Code:

All code for the server/web application component of this project can be found on GitHub repository with the URL below:

https://github.com/midmandle/KloggerServer