

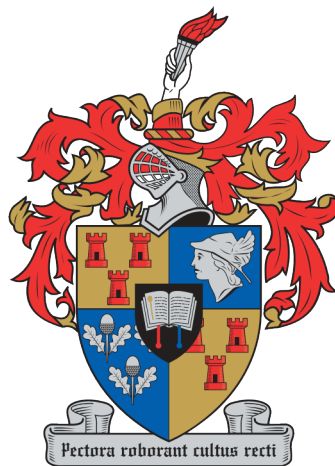
UNIVERSITY OF STELLENBOSCH

Assessment Management System for Maties CS

Author:
M. J. SHEPHERD

Supervisor:
W. H. K. BESTER

November 10, 2019



Contents

1	Introduction	3
1.1	Submission System	3
1.2	Assessment Environment	3
2	Overview	4
2.1	Web app	5
2.2	Command line interface	5
2.3	Environment	6
2.4	Gitlab	6
2.5	Server	6
2.6	Data	7
2.7	Microservices	7
2.7.1	Marking	7
2.7.2	Email	7
3	Design and Implementation	7
3.1	Server	7
3.1.1	Gitlab	8
3.1.2	Microservices	9
3.1.3	Marking Microservice	9
3.1.4	Email Microservices	10
3.2	Web app	10
3.2.1	Create Assignment	11
3.2.2	Mark Assignment	12
3.3	Command Line Interface	13
3.4	Environment	13
3.4.1	Virtual Machine	13
3.4.2	Firewall	14
3.5	Data	14
3.5.1	User	15
3.5.2	Module	15
3.5.3	Assignment	15
3.6	Security	15
3.7	Project Overview	15
4	Testing	16
4.1	Load Testing	16
4.2	Monkey Testing	17
4.3	User Testing	17
4.3.1	Manual User Testing	17
4.3.2	Real World User Testing	18

5	Discussion	20
5.1	Comparison	20
5.2	Regrets	20
6	Future Work	21
6.1	Gitlab Continuous Integration	21
6.2	Assessment Environment	21
6.3	Server	21
7	References	21

1 Introduction

At Stellenbosch University, Computer Science students spend a lot of their time doing practical work. Managing this is a large task, and the existing solutions are frustrating and have led to a number of unofficial solutions being made by different people. Maintaining and using the current solutions is time consuming for lecturers and markers that need to manage and mark large numbers of submissions that come in many different forms.

The most used solution in the Computer Science Division is the SunLearn platform. SunLearn is an effective system, but can over-complicate simple actions by offering too many options, or requiring a user to switch between multiple pages for simple actions. One example of such an over-complications would be submissions by a student. After moving between multiple pages to make or edit a submission, it is sometimes not clear whether a submission has been made or whether a successful edit has been made. Another example is the assignment creation by a lecturer or marker. SunLearn offers a large variety of options for assignments, but often these options do not fit the needs of lecturers in the Computer Science Division, such as allowing Gitlab to be an option for submissions. This can also make finding options that are needed more difficult than it would be if all of the unnecessary options were to be omitted.

In addition, there is a lack of an isolated and configurable dedicated assessment environment at Stellenbosch University. For tests and assignments that require a fresh and restricted environment, we are able to use Ubuntu test accounts on the university lab computers. This solution is effective for separating students from their own accounts, but does not offer any extra features such as internet restrictions, adding directories with documentation, or allowing students to save work done in the environment for future reference. The current solution also does not have any way of ensuring that a student is using the environment.

1.1 Submission System

One of the aims of this project was to solve the problem of over complicated submission systems by creating a submission platform that can be easily used from both the submission and submission handling ends. This submission system aims to be an all in one solution for making assignments available to students and submissions available to lecturers and markers. The most important goal of this submission system was to make it intuitive, so as to improve user experience of assignments for lecturers, and students of the Computer Science Division at Stellenbosch university.

The system that was created allows a lecturer to create an assignment from either the web app front end or the command line interface application that communicates with the server through http requests.

After an assignment has been made available, the submission system keeps track of all of the different interactions that students have with the submission system. The lecturers and markers are then able to view the logs of these interactions and are able to download specific submissions or all submissions for the assignment as required.

1.2 Assessment Environment

Another aim of this project was to solve the problem of the lack of a customisable and effective assessment environment for the Stellenbosch University Computer Science Division. To solve this problem, a scriptable virtual environment was made.

The assessment environment solution that was created works in conjunction with the submission system by being an option for the base of an assignment. This allows a lecturer set up a unique custom

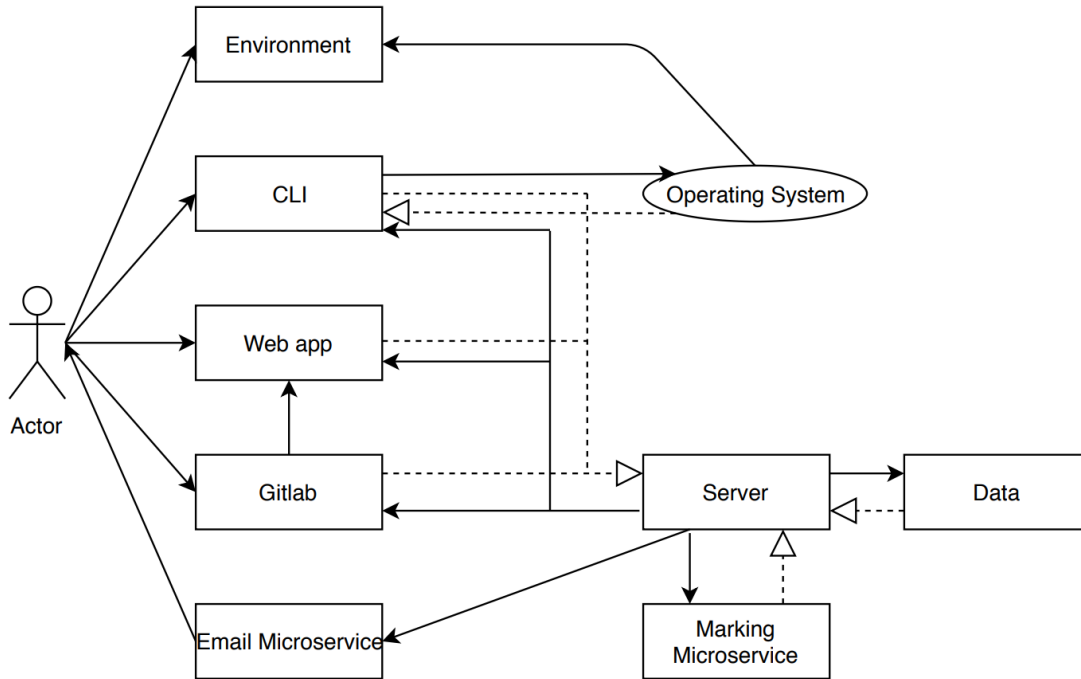


Figure 1: This is the basic system diagram for the Assessment Management System

environment for a specific assignment. A lecturer can specify configurations for an environment such as internet restrictions, computing power restrictions or whether a directory containing documentation for the assignment must be included. This assessment environment also allows lecturers and markers to see when a student has entered or left the environment and whether their submissions came from within the environment.

The environment is easy to use for a student, as it only requires one command in the Ubuntu terminal to instruct the Assessment Management System command line interface application to import and start the assessment environment. Additionally, the operating system that available as an option is the same version of Ubuntu that is currently installed on the Stellenbosch University lab computers, which puts a student in a familiar environment.

2 Overview

This section will give a brief high level overview of the project and all of its components, as is illustrated in Figure 1. A more detailed discussion of the implementation and architectural decisions will be provided in the next section.

2.1 Web app

The web app is the primary user interface for the entire Assessment Management System. The web app allows lecturers, markers and students to all interact with the system in different ways after logging in through the Stellenbosch University Single Sign-on system. The server assigns each user a title and allows access to pages according to that title. The pages that are available to all users are the *Home* page, the *Module Assignments* page and the *All Assignments* page. The *Assignment Report*¹ page is only available to lectures and markers. The pages that are exclusively available to lecturers are the *Manage Modules* page and *Automated Marking* page.

Lecturers are able to use the web app to manage their modules and assignments. Managing modules includes adding and removing students and markers for the module and creating and tracking assignments. After an assignment has been created by a lecturer, all of the students and markers that have been signed up for the module that the assignment was made under are able to see it. Creating an assignment will also open up options for that specific assignment which includes at least some of the following options, depending on the assignments specifications, as specified when it was created.

- *Toggle Late Submissions* to allow the assignment to be opened or closed after the deadline has passed.
- *Mark Assignment* to allow all of the submissions to be marked with a marking script that is provided by a lecturer.
- *Download Marking Logs* to download all of the logs from the automated marking. This option is also available to markers.
- *Assignment Report* to display assignment logs² and allow student specific submissions to be downloaded by lecturers and markers. This option is also available to markers.
- *Download Submissions* to download all submissions that have been made for the assignment as well as the files associated with the assignment. This option is also available to markers.
- *Download Assignment Files* to download the files that a lecturer has made available for the assignment.

Students are only able to use the web app to view assignments and make submissions. After a lecturer creates an assignment, the students will immediately be able to see the time frame and description/instructions of the assignment, but will only be allowed to download assignment files and make submissions at times that a lecturer has allowed. Once a student has made a submission, they are emailed a confirmation and are given the ability to download their own submission. Students are able to replace their submission with a re-submission if a lecturer has allowed re-submissions.

2.2 Command line interface

The Command Line Interface is the secondary user interface for interacting with the Assessment Management System. It comes in the form of an executable python script that provides a simple way for users to log in through the Stellenbosch University Single Sign-on system and make requests to the server. Just like with the web app, the actions that a user may take are limited by their title, which is assigned

¹There is no general report page, only pages specific to each assignment

²Logs include but are not limited to submission counts, submission times, environment logs and Gitlab pipeline statuses.

based their SUID. The actions available are slightly more limited than than the web app for the sake of simplicity.

A lecturer is able to fetch the files required to create an assignment, create an assignment, download all of the submissions for an assignment and add a list of students to one of their modules from a file. A marker is only able to download all of the submissions for an assignment. A student is able to get the files for an assignment, make a submission for an assignment and start the environment associated with an assignment.

The most important function of the CLI application is the fact that it handles the creation and state monitoring of the assessment environments. This means that when a student makes a request to start the environment, the application begins communicating with the server to find out how it needs to configure the environment. The application then ensures that the environment is behaving as expected and updates the server whenever anything notable happens.

2.3 Environment

The the assessment environment is a virtual machine that is configured from a pre-made virtual machine. This allows it to be configurable in terms of computing power and time limits. The configurations of the environment also allow for various different internet restrictions³, such as full internet blocking, only blocking Inetkey or only blocking/allowing specific websites. A lecturer also has the option to allow a shared folder to be enabled while the environment is running, which allows files to be made available to the students while keeping them in an isolated environment. The shared folder also allows work that is done in the environment to be saved and persist on the host file system once the environment has been torn down by the CLI application.

2.4 Gitlab

The Assessment Management System allows a lecturer to create Gitlab projects for an assignment. This would create a subgroup within the Gitlab module group for the assignment and creates a project for every student that is signed up for the module. Every marker associated with the module is given reporter permission on the group, so that they are able to report on each users project.

When creating a Git assignment, a lecturer also has an option to enable continuous integration tests. This includes the option to include a style checker and/or a script that will be run in the specified language. The status of the pipelines from the projects are also visible and linked in the Assignment Report page so that a lecturer or marker can see the status without having to move to Gitlab to have an overview of the pipelines.

2.5 Server

The server is the functional tier of this project. This means that it connects all of the different components and allows them to communicate with each other. The main function of this functional tier is to perform almost all of the logic required for the system so that the data tier and user interface tier can communicate effectively. The server works as a web API that can be used with HTTP requests. To protect the server and ensure that only Stellenbosch University personnel can interact with the system as a whole, the API is only available if a user has signed in to their Stellenbosch University user account with the university

³All restrictions will still allow the web app to be accessed.

Single Sign-on system. This not only allows the server to know that a user is authorised, but also allows it to use the users SUID as a unique identifier to assign a title in their session.

2.6 Data

The data tier of the Assessment Management System contains all of the information about users and assignments as well as submissions and templates for email responses and assignment files that the server uses to create assignments. The user data allows the server to decide what permissions to give and which modules and assignments to take into account when a user logs in.

To ensure that the data is secure, it is stored on the server. Only the server can access any of the data and as was stated in section 2.5, the server is protected by the Stellenbosch University Single Sign-on system.

2.7 Microservices

This project makes use of microservices so that tasks that might hinder the main server can be outsourced to a different service. These microservices are served as RESTful APIs and are exposed locally to the server and protected by an API token. This will be elaborated on in section 3.

2.7.1 Marking

This microservice enables a lecturer to safely run a marking script on every submission from an assignment. The microservice requires the marking script and the assignment code⁴. After the marking process is complete, the logs are available for download by a lecturer or marker from the Assignment Report page.

2.7.2 Email

This microservice allows the server to send emails to users at their Stellenbosch University email accounts. The microservice only requires the SUID of a user and will send emails from pre-created templates according to the endpoint specified.

3 Design and Implementation

The Assessment Management System is a combination of many very different components. The final design of the project is a form of three-tier architecture. Three-tier architecture is characterised by having a presentation tier, an application tier and a data tier [6]. The tiers of this project are illustrated in Figure 2. This style of architecture allows for each of the different components to be updated or changed without needing to update any of the other components, which increases the maintainability of the project. This section will discuss the implementations of all of the different components.

3.1 Server

The server is a set of functions that implements most of the functionality of the Assessment Management system and is Python web application that was build using the lightweight WSGI web application framework, Flask. Flask was chosen so that the server could be kept maintainable. Using a lightweight

⁴The assignment code is a unique identifier for each assignment

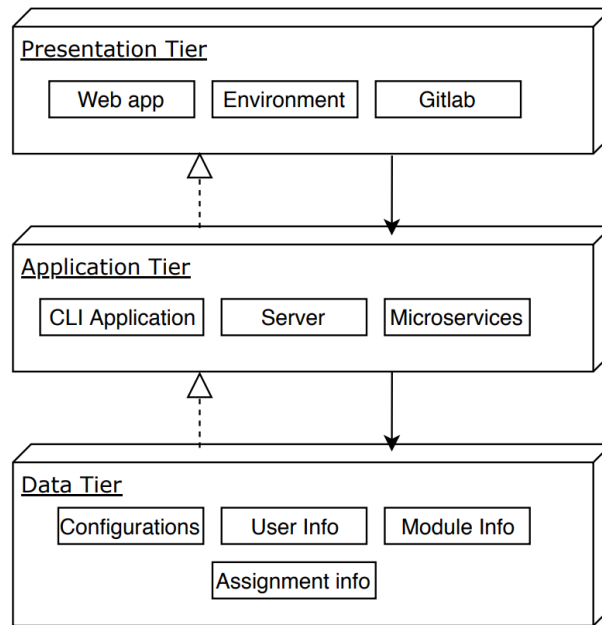


Figure 2: The three-tier architecture design.

framework like Flask means that it is easy to keep the implementation simple, which makes modifying and extending the functionality of the server easier [13].

The server does serve the web app front end directly, but also exposes endpoints for all of the other presentation tier components to interact with it. The server functions as the main link between the presentation tier and the data tier, as it is the only part of the application tier that has access to the data tier. This makes the data more secure, which will be discussed in more detail in section 3.6.

3.1.1 Gitlab

The Gitlab integration for the Assessment Management System is done by communicating with the Stellenbosch University Gitlab server through the Gitlab API. There is an *AssessmentManagementSystem* group on the server and every module that is added to the system receives its own sub group within that group. When a lecturer creates an assignment and requests that Git projects be made, a subgroup within the module subgroup is created for the new assignment. All lecturers and markers that are assigned to the module are given reporter permission for the assignment group. Once the group has been created, a new project is created for every student that is signed up for the specified module. If the lecturer chooses to include continuous integration tests in the projects, then shared runners and jobs are enabled for the projects on creation. To ensure that the continuous integration pipelines run correctly, the configuration Yaml file is generated by the server to fit the options chosen by a lecturer. This configuration file is then included in each project, along with all of the files and scripts that are required for the tests to run. To keep track of project activity, a hook is added to each project that notifies the server of every commit that occurs. The server logs this activity and displays it on the *Assignment Report* page.

The web app of this project also allows markers to track the status of the pipelines of each student's

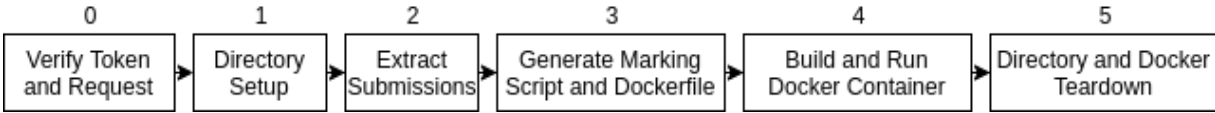


Figure 3: The automated marking pipeline.

project from the *Assignment Report* page. Students are shown a link to the project from their *Assignments* page so that they can easily find their Gitlab project for the specific assignment.

3.1.2 Microservices

Microservices are a collection of services that are highly maintainable and testable, loosely coupled, independently deployable and organised by the purpose of their functionality [10]. The decision to use microservices to serve the main server came as a solution to the problem of multi-threading in Python. Although creating threads to carry out the tasks would have been possible, this would put extra unnecessary computational pressure on the server that needs to be able to handle multiple users at the same time. The use of microservices also makes the server more modular, which increases its maintainability.

The microservices in the project are in the form of RESTful APIs that have been built with the Flask-RESTful extension of the Python Flask framework. This also means that the microservices can be run in their own threaded mode, which allows multiple users to be making use of any of the microservices without blocking the server or the microservice. The APIs are protected by secret API tokens that only the server and services have access to.

3.1.3 Marking Microservice

The marking microservice is a service that is able to run marking scripts on all of the submissions that are being stored on the server for a particular assignment. The service then makes the logs from the marking available to lecturers or markers. The service takes an assignment code, a language, marking files and an API token as data parameters. The service works as a pipeline that is illustrated in Figure 3. This pipeline allows for assignments to be marked easily and safely, as it executes all code inside a Docker container and enforces a timeout on marking processes that are taking too long. Executing the code in a Docker container means that the server is protected from any malicious code [7].

Stage 0: Verify Token and Request

The API token that is received is verified against the one stored in the data tier to ensure that the source of the request has sufficient permissions to be requesting a marking job. The request is then analysed to see if the assignment for which the marking has been ordered is a valid assignment and if a marking script has been included.

Stage 1: Directory Setup

A temporary directory to contain the extracted submissions is created for the specified assignment. This directory will be included in the Docker container in Stage 3. A logs directory is also created for the specified assignment to store the logs of the marking script. The logs directory is labeled with a timestamp so that an assignment can be marked multiple times without losing the previous logs.

Stage 2: Extract Submissions

All of the submissions for the specified assignment are taken from the server and stored in the temporary directory created in Stage 1. Since the Assessment Management System only allows for compressed submissions, these submissions must be extracted into the temporary directory so that the marking script can be run on it. As students tend to compress entire directories that contain their submissions, this stage also checks to see if this is the case and moves all of the contents of that directory one directory up so that the marking script can find the files that it expects to find.

Stage 3: Generate Marking Script and Dockerfile

A script is created that runs the provided marking script on each of the different submissions and pipes the outputs into log files, which are then stored in the logs directory, which is bound to the Docker container so that they persist once the image has been deleted. The script enforces a time limit on the marking process for each submission so that it does not get caught in an infinite loop. A Dockerfile is created that uses the specified language as a base and includes the generated script. This Dockerfile describes an image that contains the temporary directory that was created in Stage 1 and executes the generated script.

Stage 4: Build and Run Docker Container

The Docker image is built using the Dockerfile that was generated in Stage 3. Once the image has been built without issue, the container can be run. The container is run with the `-rm` flag used, so that it will perform its own tear-down once it has completed the marking process. The `-v` flag is used to bind the logs directory to the container as a volume.

Stage 5: Directory and Docker Tear-down

The Docker tear-down is not explicitly done by the marking service, but by Docker itself through the `-rm` flag being used in Stage 4. To complete the tear-down, the temporary directory that contains all of the extracted submissions is removed completely. The only remnants of the marking process is a time stamped directory that contains the marking logs.

3.1.4 Email Microservices

The email microservice is a service that allows the server to send multiple emails at a time without blocking the main server thread. Currently, it is only used to send emails to students to confirm that they have made a submission. The service takes a SUID, assignment code and API token as data parameters. Once the API token has been verified, a pre-written email template is completed with the assignment code and SUID and sent to the Stellenbosch University email account associated with the SUID.

3.2 Web app

The web app is intended to be as intuitive and uncomplicated to use as possible, while still providing an adequate level of functionality. The application is a Flask application that is served by the server. It uses the Jinja template engine [14] to generate dynamic web pages, based on the variables that it receives from the server. From the web app, a lecturer has access to the following actions that require work to be done by the server:

3.2.1 Create Assignment

A lecturer is able to use the *Assignment Creation* page to choose the configurations for a new assignment. An assignment has the options of a submission base, a Git base or an environment base. Any of the bases can either exist on their own or can be combined with the other bases in any way. Any assignment has the compulsory requirements of a module, a title, start time and end time. The rest of the available options are specific to the bases that a lecturer has chosen.

Submission

- *Allow late submissions.* This can be toggled at any point after the assignment has been created.
- *Allow re-submissions.* A lecturer can specify whether a student is allowed no re-submissions, infinite re-submissions or a specific amount of re-submissions.
- *Make files available.* This option allows a lecturer make files available to the students for this assignment. There is also an option to only allow the files to be downloaded between the start and end time (With late submissions being allowed extending this time as well).

Environment

- *Internet Restrictions.* This gives a lecturer multiple options for internet restrictions which will be discussed further in Section 3.4.2.
- *Make use of a virtual environment.* This gives a lecturer the ability to configure an environment to suit the needs of the assignment. This includes setting a time limit on the environment, selecting the operating system, specifying the amount of CPUs and RAM that will be made available to the environment and whether the files attached to the submission base will be made available inside the virtual environment through a shared folder.

Gitlab Repositories

- *CI language* allows a lecturer to set the language for their continuous integration tests.
- *CI style-checker* and *CI tests* allows the lecturer to specify whether they would like to make use of continuous integration or not. Selecting CI tests will prompt a lecturer to upload a script to be executed as the continuous integration test.

These options were chosen to give a lecturer as much control over the configuration of an assignment as possible, while maintaining an uncomplicated and uncluttered user interface. Once a lecturer has chosen the configuration for the assignment, the form is sent to the server as a POST request. This allows the server to parse the form and create two Json files that define an assignment. The Json files are *sub.json* and *env.json*.

The *sub.json* file contains all of the submission and Gitlab configurations of the assignment. This is then used to configure the git projects, display the assignment on the *Assignments* pages as well as provide the server with the parameters required to decide if submissions are valid.

the *env.json* file contains the configuration parameters required for the assessment environment. This file is used in conjunction with the command line interface to configure and create the virtual machine that is needed for the assignment. This will be elaborated on in Section 3.4.

Scientific Computing 372: Comprehensive Test - wb372

Assignment code - wb3720

✗ You have not yet made a successful submission

Duration:
Fri 01 Nov 2019 - 09:00 — Fri 01 Nov 2019 - 12:00

Submissions Available:
Unlimited

NOTE: Only files with a .tar.bz2 or .zip extension will be accepted. It is your responsibility to compress your submissions correctly.

[UPLOAD SUBMISSION](#)

☐ I declare that I have read the SU Policy on Plagiarism, and that I have abided by the rules.

[Submit](#) ➤

[FILES FOR DOWNLOAD](#)

Figure 4: An example of an assignment card.

When the assignment has been created on the server, a log file is also generated. This log file is populated with empty logs for every student that is signed up for the assignment's module and is updated whenever a student interacts with the assignment in a meaningful way⁵

3.2.2 Mark Assignment

A lecturer is able to use the *Automated Marking* page to trigger the marking service that was discussed in Section 3.1.3. This action requires a lecturer to choose the assignment to be marked, the language that their marking script is written in and to upload the marking script. These options are then sent to the marking service, which begins the marking process and makes the logs available for download after the process is complete. This action is designed to time with marking for both lecturers and markers.

A student or a marker does not have sufficient permissions to trigger any significant work to be done by the server. A student is able to view all of the modules that they are signed up for on the Assessment Management System, as well as all of the assignments for each of those modules. A student can view and make submissions for an assignment from the *Assignments* page, which is able to show all available assignments or only the available assignments for a chosen module. The *Assignments* page uses the *sub.json* files for each of the assignments to create a display card for each. This card is designed to get all of the information of the assignment to a student as succinctly and effectively as possible, as is illustrated in the screenshot in Figure 4.

⁵Meaningful meaning making a submission, pushing to Gitlab or using the assessment environment.

When a lecturer or a marker is viewing an assignment they also see a card, but the section where a student can make a submission for the assignment is replaced with buttons to either download all of the submissions for the assignment or to go to the *Assignment Report* page for that specific assignment. A lecturer also has the additional option to toggle whether late submissions are allowed or not.

3.3 Command Line Interface

The CLI is a Python application that allows users to easily communicate with the server from the command line, as was discussed in Section 2.2. The largest barrier to communicating with the server from the command line is the fact that the server is protected by the Stellenbosch University Single Sign-on system. Python was chosen for this application, because we were able to overcome this barrier by using the Requests [9] library for Python, which allows us to create a session in which a user can be logged in to through the Single Sign-on system and make requests to the Assessment Management System. The application needs to be downloaded and installed for it to be used.

The most important function of the CLI is to coordinate assessment environments. It does this by fetching the environment configurations from the server and using VBoxManage to import and configure the environment from a pre-determined source on the Stellenbosch University computer lab system. This process will be elaborated on in Section 3.4. The CLI application keeps track of the actions of the environment through system commands and updates the server when anything of note occurs.

3.4 Environment

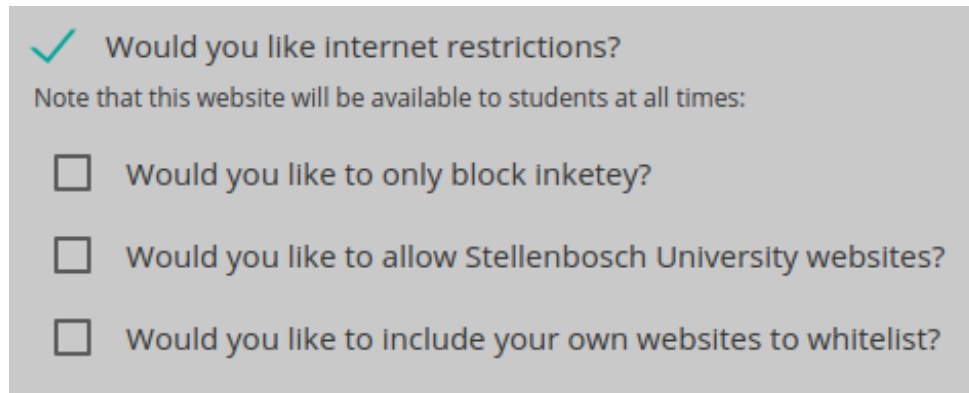
The assessment environment is an isolated environment in which a student is able to complete an assignment in an operating system that closely mimics the Stellenbosch University computer lab computers. An environment is configured according to the options selected by a lecturer in the assignment creation phase discussed in Section 3.2. This is achieved using VirtualBox [2] virtual machines and Uncomplicated Firewall (UFW) [16].

3.4.1 Virtual Machine

The isolated assessment environment is a Virtual Box virtual machine (vm), with the specified operating system installed⁶. The vm is imported from a pre-configured open virtualisation file (OVF). This OVF contains an operating system that is locked in full screen mode. This is achieved by disabling any control that the guest system has, such as options from within the guest and the host key that would allow the user to switch between the host and guest systems. The vm includes a student account and an admin account. The student account has protected scripts that run on start up to ensure that if a shared folder has been included, that it is mounted correctly and made available in the home directory of the student.

The environment is completely imported, started, controlled and monitored by the CLI application discussed in 3.3. This is done with the VBoxManage command line interface that Virtual Box provides. This allows the CLI application to take an existing OVF, import it, alter the configurations of the vm, start it, monitor its status and shut it down if the time limit of the environment has been reached. The configurations are fetched from the server with a Get request and are returned in the form of the *env.json* file of the assignment. The CLI application then informs the server when the environment starts and stops and the IP address of the machine that it is being run on.

⁶Ubuntu 16 is currently the only available operating system, to match the version installed on the Stellenbosch University computer lab computers

A screenshot of a web form titled "Would you like internet restrictions?" with a green checkmark icon. Below the title is a note: "Note that this website will be available to students at all times:". There are three checkboxes, all of which are currently unchecked. The first checkbox is labeled "Would you like to only block inketey?". The second checkbox is labeled "Would you like to allow Stellenbosch University websites?". The third checkbox is labeled "Would you like to include your own websites to whitelist?".

✓ Would you like internet restrictions?

Note that this website will be available to students at all times:

☐ Would you like to only block inketey?

☐ Would you like to allow Stellenbosch University websites?

☐ Would you like to include your own websites to whitelist?

Figure 5: All of the options for internet restrictions.

The `env.json` file contains all of the configurations for both the vm and the firewall, which will be discussed in section 3.4.2. The vm configurations include the allowed computing power in the form of number of CPUs and amount of memory to be made available, the time limit of the environment⁷, whether a directory needs to be linked to the environment as a shared folder⁷ and all the details of the files that need to be included in this directory. If a lecturer chooses to make a shared directory available, this means that documentation for the assignment can be made available in the environment and a student is able to save the work that they have completed in this assignment for future reference. If a lecturer does not want to allow a student to save their work beyond the assignment, then the assignment documentation can be made available through only the web app, which is always available from within the assessment environment.

In the unlikely case of a student escaping from the environment, the CLI application will detect that the environment has been exited and will shut it down and notify the server. This means that a lecturer can keep track of students who try to get around the system. The vm also uses cookies in all of its browsers to notify the server that a submission has come from within an environment. This also allows lecturers to see if a student is trying to get around the system.

3.4.2 Firewall

The firewall, which is controlled with UFW, allows the CLI to enforce internet restrictions on the host system while the assessment environment is running. The internet restrictions available to a lecturer are illustrated in Figure 5, which is part of a screenshot from the *Assignment Creation* page of the web app. If a lecturer selects the option to include internet restrictions, but does not select any of the other options, then all internet access will be blocked in the environment.

3.5 Data

The storage of data for the Assessment Management system has been kept simple to avoid infrastructure for the sake of infrastructure. The data that needs to be stored, is user data, module data and assignment data. Any files that could be accessed by different threads of the server are protected with the Python

⁷These configuration options are optional.

Filelock library [12]. This allows simple files to be used to store our data, which avoids the overhead of a database.

3.5.1 User

User data is stored in the *users.csv* file, which is stored in the resources directory of the server. This stores the SUID, title and modules that a user is signed up for and is used to create a User object so that the server is able to easily gather and display the correct pages for the user through the web app.

3.5.2 Module

Module data includes information about the module and all of the students who are registered for the module. Both of these are stored in the resources directory on the server. Module information for all of the modules is stored in *modules.csv*, and stores the module code and the Gitlab group for that project. The students that are registered for the module are stored in *<module code>.csv*.

3.5.3 Assignment

When an assignment is created, a directory is created to contain all of the files associated. The directories created are:

- *modules/<module code>/<assignment code>*,
- *modules/<module code>/<assignment code>/assignment_files*, and
- *modules/<module code>/<assignment code>/submissions*.

The submissions that are received from students for an assignment are stored in the newly created *submissions/<SUID>* directory. The assignment files directory stores the previously discussed *sub.json* and *env.json* as well as the log file and the files that are available for download for the assignment.

3.6 Security

To keep the server secure, all routes except for the Gitlab push notification route are protected by the Stellenbosch University Single Sign-on system. This required the server to link the Stellenbosch CAS server as the login server for the Flask-CAS library [15]. This allowed the *@login_required* annotation to be used to force the server to require a valid token from the Stellenbosch CAS server before routes can be accessed. The title and SUID of the user are stored as Flask session variables so that they can be seen by both the server and the web app. Flask sessions are secure from tampering as long as a good secret key is used [8]. The secret key that is used by the server is a 24 character String of randomly generated characters, generated with Python's *os.urandom* function. According to the Python language documentation, this is suitable for cryptographic purposes [3].

3.7 Project Overview

To count the lines of code written for this project, CLOC [1] was used. This allowed for different components to be easily analysed.

Language	Files	Comments	Source
Python	8	196	1366
HTML	9	84	984
JavaScript	4	1	157
Bourne Shell	2	0	3
	34	281	2510

Table 1: Server and Web App

Language	Files	Comments	Source
Python	6	28	217
Bourne Shell	2	4	21
INI	2	0	8
Dockerfile	1	25	6
	11	57	252

Table 2: Marking and Email Microservices

Language	Files	Comments	Source
Python	1	55	288

Table 3: CLI Application

4 Testing

The fact that most of the server and web application are protected behind the Stellenbosch University Single Sign-on system makes automated testing extremely difficult. To get around that problem, a testing Git branch was created where the log in system was replaced by spoofing a user. This allows the system to be manually tested with different levels of permission.

4.1 Load Testing

Load testing was used to test the limitations of the Assessment Management System when multiple users are trying to use the system for similar functions. The component that needed to be tested due to its potential weakness was the submission system. This component has the potential to fail under serious stress, because of the fact that it is updating a log file, rather than a database and is saving files to the server file system. The server handles potential issues by using a file lock system and swap files to ensure that the logs version is always the correct version and to ensure that no two user threads will cause each other to be blocked. To test this, the submission was tested with 10, 100 and 1000 simultaneous submissions respectively. The system was able to save all of the submissions and correctly log the timing of the submission for all three of the cases. This shows that the system can handle a far higher load than can realistically be expected of it.

4.2 Monkey Testing

Monkey testing is testing that deals with random inputs [11]. To use this form of testing, the free online tool Monkey Test It [5] was used. This tool clicks every link on a page submitted searching for errors. The tool also notice if there's any broken links, JavaScript issues, missing images, or other commonly missed mistakes [5]. Every page of the web app was given to this tool with every different permission. The only issue that was found was from within the Materialize CSS library that is used.

4.3 User Testing

User testing is the process of using real human interaction with a product to evaluate if it behaves as expected [4]. Two types of user testing were used to test the different components of the Assessment Management System. We were able to use the system in a real life assignment for Scientific Computing 272, which allowed us to monitor how the system handles multiple users that are representative of the target group of this project. Some of the components could not be tested by these users under the constraints of the assignment, which meant that some of the components were tested by manually running through all of the possible interactions with the system.

4.3.1 Manual User Testing

This section explains how different components were tested by manually using all of the features of the Assessment Management System.

Git integration

The Git integration was tested using dummy accounts on the Stellenbosch University Gitlab Server. The users were signed up for a module on the system with permissions of lecturer, marker and student. A Git assignment was the created with continuous integration enabled. This tested part of the *Assignment Creation* page, and all of the Gitlab integration. It was observed that projects were created for every one of the students and the pipelines in those projects were running. Submissions that would pass and fail the CI tests were inserted into different project repositories to ensure that the pipelines were functioning as expected. This also allowed us to test the *Assignment Report* page for Git assignments, which shows the status of the different pipelines with icons. This test also showed that markers and lecturers were given correct permissions.

Assessment Environment

The assessment environment was tested by creating assignments with Environment bases. Multiple Environment based assignments were needed to be made to test the different functions, such as internet restrictions, computing power restrictions and the inclusion of shared folders. The findings of these tests were that under positive testing conditions, the environment performed as expected. The configuration time of the environment on a Stellenbosch University computer consistently took approximately 45 seconds, the time limit on the environment was correctly enforced by the CLI application, the internet restrictions were created as expected and torn down correctly, the shared folders were correctly mounted and persisted correctly after the environment was shut down and the server was notified of any actions correctly in every instance. All of the communications with the server were verified by inspecting the *Assignment Report* page, which also shows us that the reporting is working correctly.

CLI Application

The functions within the CLI application were mostly verified by the testing of the Assessment Environment. To test the rest of the application, all of the different instructions were executed with all of the different levels of permission. There was no unexpected behaviour, which shows that the CLI application is working as expected.

Marking Microservice

The marking service was tested by creating an assignment with all of the different allowed submissions included. This meant that there were submissions where directories had been compressed, there were empty submissions, there were submissions that would fail the marking script, there were submissions that would pass the marking script and there were submissions that would force the marking script to time out. The service was then told to mark all of these assignments. When the logs were downloaded, it was clear that the marking service had behaved as expected and handled all submissions correctly.

4.3.2 Real World User Testing

This section will discuss our findings from the Assessment Management System being used in a real world assignment situation. Once a student made a submission through the Assessment Management System, they would be emailed a confirmation and an invitation to complete a questionnaire to rate and/or comment on the system. All of the data and feedback discussed in this section came from that questionnaire. This test was essentially an end to end test, from the assignment creation, to the assignment being available during a time frame, to late submissions being allowed and disallowed. The initial test was an assignment that was available for submissions for five days. This allowed for fixes to be made in response to criticism received from users. There were three stages of testing, which allowed for an iterative feedback loop to be formed.

Method

At the start of the first stage of the test, there was a bug that gave every user a persistent error if one user made an invalid submission. This bug accounted for two out of the four reported bugs and was immediately fixed and was not reported again. The other reported bug for this stage of the test was that a submission did not register the first time, but this bug could not be replicated⁸. From this stage, all of the comments and criticism addressed the appearance of the web app, which was taken into account in the preparation for the next stage.

The second stage of the test was a new submission being made available to the students. In this stage, the last error was reported, which was that if late submissions were disabled and a student had already made a submission, they could still make a re-submission. This was caused by the server skipping a submission check when receiving a re-submissions and was easily fixed. A suggestion to add an option for a student to download their own submission was made during this stage, which was implemented.

The third stage of this test was the same as the second stage, with a new submission being made available to the students. In this stage questionnaire responses did not report any errors, nor did they make any new suggestions.

⁸My hypothesis is that the student attempted to submit the first time while the server was being restarted to fix the first bug.

The Numbers

During the real world user testing, forty unique users made submissions through the Assessment Management system. twenty-two responses to the questionnaire were recorded and these responses are illustrated by the figures below. Of the twenty-two responses, more than half were Computer Science students, which is the target group of this project. The students were asked to rate the Assessment Management System compared to both the normal Scientific Computing submission system as well as the SunLearn submission system.

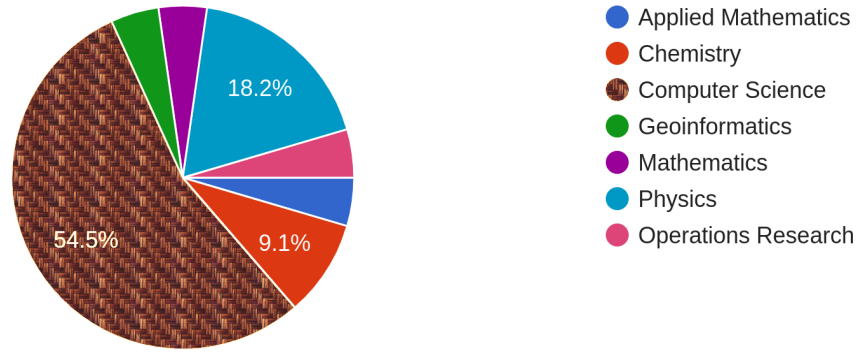


Figure 6: The area of study of questionnaire respondents.

In Figure 7 we can see the summary of the responses to the statement “the new submission system is easy and intuitive to use compared to the current Scientific Computing 272 submission form”⁹. This figure shows us that the students feel mostly neutral about the Assessment Management System compared to the Scientific Computing submission system. This is to be expected, as the Scientific Computing submission system is a very simple and intuitive system.

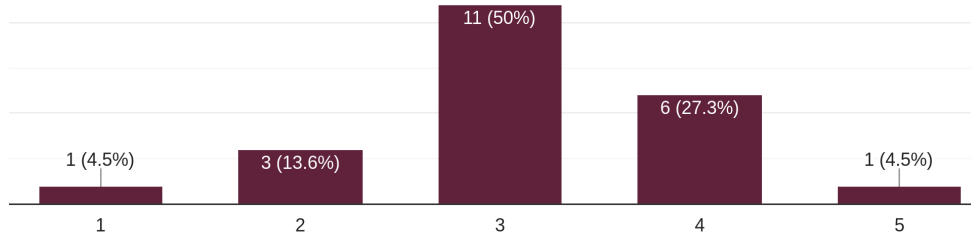


Figure 7: The comparison of the Assessment Management System to the Scientific Computing submission system.⁹

Figure 8 shows us the summary of the responses to the statement “the new submission system is easy and intuitive to use compared to SunLearn”. This figure gives us a better indication of the success of the test. In this figure, we can see that most students responded positively to the Assessment Management System when comparing it to SunLearn.

⁹1 = Strongly disagree, 2 = Disagree, 3 = Neutral, 4 = Agree, 5 = Strongly agree

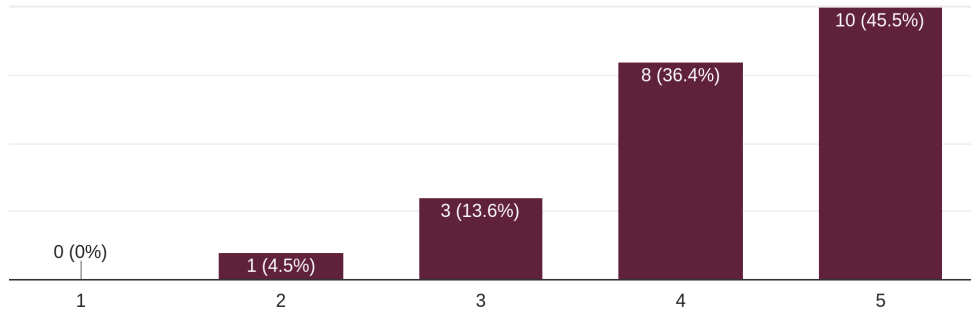


Figure 8: The comparison of the Assessment Management System to the SunLearn submission system.

It is notable for the SunLearn comparison that the response that rated the system 2 and one of the responses that rated the system 3 were received during the first stage of testing before the first bug was fixed. Of the four responses that were received after the third stage of testing, three rated the system 5 compared to SunLearn. This shows us that the iterative nature of this real world user testing was an effective way to test that the system was functioning correctly and to help to improve the user experience, which assisted with one of the main goals of this project.

5 Discussion

In this section, we will compare this system to other existing systems and discuss one major change that I would make to this project if I had the time.

5.1 Comparison

This system is not meant to be a replacement for SunLearn, but rather an uncomplicated alternative for simple functionality that is made complicated by the large variety of options on SunLearn. The student responses that can be seen in Figures 7 and 8 show that students preferred the Assessment Management System over SunLearn. These responses also show that since the students seemingly saw no difference in the ease of use of the Assessment Management System and the Scientific Computing submission system, and they appear to enjoy the Scientific Computing submission system more than SunLearn.

These results show us not only that an uncomplicated solution is needed, but also that this project achieved the goal of being that solution.

5.2 Regrets

The one thing that I would change in this project is that I would have used a JavaScript framework to create a completely independent front end for the web app. This would have created a more modular architecture and would have made good design easier than it was with pure HTML and Jinja.

6 Future Work

The nature of a system with many different components is that there will always be something else that can be added. This can only lead to massive systems such as the one the SunLearn is created with. It is for this reason that any future work on this project should not be to add functionality, but rather to supplement current functionality.

6.1 Gitlab Continuous Integration

Although the Gitlab CI works in the form that it is currently in, it would be difficult to update the CI files, as each repository contains its own CI files. Future work could change this so that the CI files are stored in one place and the CI configuration file in the assignment repositories can point to that one place.

6.2 Assessment Environment

The assessment environment is currently only set up with the Ubuntu 16 operating system as an option. Future work would be to set up OVF's for different versions of Ubuntu and other operating systems.

6.3 Server

Future work for the server would be extra options for managing assignments. This could include options such as archiving assignments when they are no longer needed and saving assignment configurations to be used at a different time.

7 References

- [1] AlDanial. *cloc*. 2019. URL: <https://github.com/AlDanial/cloc>.
- [2] Oracle Corporation. *Oracle VM VirtualBox User Manual*. 2019. URL: <https://www.virtualbox.org/manual/>.
- [3] Python Software Foundation. *Miscellaneous operating system interfaces*. Nov. 2019. URL: <https://docs.python.org/3/library/os.html>.
- [4] Every Interaction. *User testing*. 2019. URL: <https://www.everyinteraction.com/definition/user-testing/>.
- [5] Monkey Test It. *Monkey Test It Readme*. 2019. URL: <https://monkeytest.it/readme>.
- [6] JReport. *3-Tier Architecture: A Complete Overview*. 2019. URL: <https://www.jinfony.com/resources/bi-defined/3-tier-architecture-complete-overview/>.
- [7] Asad Memon. *How we used Docker to compile and run untrusted code*. 2016. URL: <https://blog.remoteinterview.io/how-we-used-docker-to-compile-and-run-untrusted-code-2fafbffe2ad5>.
- [8] Luke Paris. *Baking Flask cookies with your secrets*. Jan. 2019. URL: <https://developers.google.com/sheets/api/quickstart/python>.
- [9] Kenneth Reitz. *Requests: HTTP for Humans*. 2012. URL: <https://requests.kennethreitz.org/en/v1.1.0/>.

- [10] Chris Richardson. *What are microservices?* 2019. URL: <https://microservices.io/>.
- [11] Dhanshri Salvi. *What is Monkey Testing?* 2019. URL: <https://www.guru99.com/monkey-testing.html>.
- [12] Benedikt Schmitt. *py-filelock*. May 2019. URL: <https://github.com/benediktschmitt/py-filelock>.
- [13] Mindfire Solutions. *Flask vs Django*. May 2018. URL: <http://www.mindfiresolutions.com/blog/2018/05/flask-vs-django/>.
- [14] Pallets Team. *Jinja Documentation*. 2007. URL: <https://jinja.palletsprojects.com/en/2.10.x/>.
- [15] Cameron White. *Flask-CAS*. Apr. 2019. URL: <https://github.com/cameronbwhite/Flask-CAS>.
- [16] Paul White. *UFW - Uncomplicated Firewall*. Mar. 2017. URL: <https://help.ubuntu.com/community/UFW>.