Docker Einstein

Functional Specification

Name: Ben Coleman
Student Number: 15487202

Supervisor: Stephen Blott

Date Completed:
28/11/2018

Functional Specification Contents

0. Table of contents

1. Introduction

1.1 Overview

The purpose of this project is to rebuild the Einstein script correction utility in DCU with Dockers. The current Einstein system was developed piece by piece with no end goal in mind. The aim of this new system is to start from the beginning and redevelop the system completely. The Docker Einstein will utilise the original marker script used for automated correction, but will work in a completely new docker environment.

The Einstein application is a simplistic file correction tool. Users interact by uploading their scripts to the system. Administrators/Lecturers can setup marking schemes with sample input and output that indicates a correct or incorrect script. The feedback of each upload is returned to the student, and is shown to a lecturer in a complete overview of student submissions.

The system currently experiences high traffic, especially during examinations or assignment submission deadlines. Workload tends to be delegated to the client side rather than the server to ensure the application runs smoothly. With more and more students interacting with the system, the need to change the core design is even more vital. With the new application, the entire system will be inside a Docker image that can be configured and deployed where necessary.

Another aim of this project alongside the extensibility is enabling it to run on servers outside of DCU. Einstein is currently hosted on a DNS server in DCU, however the new system should have a plug and play type configuration. Any Apache/DNS configuration files needed to host the application in DCU will be fed into the docker image before a container is created. The idea is that the system should work regardless of what credentials are given in these configuration files. A different set of parameters detailing another server, in another university for example, should allow the Docker Einstein to create a container tailored to that server.

Having the system built with Dockers increases scalability and possible future growth. If a much larger population wanted to interact simultaneously, the system could be extended for individual containers to be created per student submission and torn down after. These could work in parallel regardless of the volume of interactions with Einstein and could therefore reduce the load on the application during correction.

There will also be some redesigning of the new system interface. The only element being utilised from the old Einstein is the marker script. This means that a new front-end will be built within the new docker

environment. A new UI design will cater for more than just DCU students and will ensure ease of access no matter where the system is deployed.

1.2 Business Context

While there are no businesses sponsoring this product, the aim is to have it be deployable outside of DCU. The current system is only seen within DCU and is not tailored to other university server configurations. The new Docker environment should allow the system to be adopted and utilised by any other institution with the server credentials to host it.

1.3 Glossary

Docker - A computer program that performs operating-system-level virtualization, also known as "containerization".

LDAP (Lightweight Directory Access Protocol) - Software protocol for enabling anyone to locate organizations, individuals, and other resources such as files and devices in a network.

Docker Image - Template for docker container including all setup and configuration.

Docker Container - Running instance of a docker image.

2. General Description

2.1 Product / System Functions

Logging in - When a user wishes to access Einstein they will be prompted for their login credentials, either on the application or through their module website. A student or admin may login at this point, indicating to the system which level of access should be granted. An admin/lecturer login will tell the system to allow access to the student details and overview pages, while a student login will only see the basic functionality. Unauthorised users i.e not in the system should not be granted access to the uploads page or beyond.

Uploading a script - The Docker Einstein system will work similarly to the current Einstein utility. Students will be able to upload their scripts for either lab examinations or assignments and get feedback on the correctness based on a number of predefined test cases for that script.

Viewing feedback - Depending on the script and circumstances (examination or assignment), a student may be able to view more detail about individual test cases after uploading. With the new Docker system, the intention is to retain this simplistic approach and instead just change what is happening in the background.

Admin overview - Lecturers/Admin may have access to a details page, showing submissions and grades from different students. They may then view individual records of a student's upload history, and comment on scripts which the student receives via email.

Admin configuration - An admin may at any point create new sample input and output, along with test cases for a new lab or exam. These can then be added to the system to allow for new scripts to be corrected by the marker.

Most recent upload access - A student should be able to access their most recently uploaded file. This is helpful for a student to verify if they have already uploaded a file, or in the case that a script was uploaded initially and a student needed to go back and view the feedback of that script without re-uploading it.

Progress page - In some module cases, a lecturer may opt to allow students to access a progress page, showing the status of each student against each other in the case of lab work.

Student details - Details about the student may also be on display such as username, email and module title. The system will feature a new front-end UI, accessible to all students. The aim of having Docker Einstein be deployable on any server configuration outside of DCU means any range of students may need to interact with it. Therefore the UI will not add any unnecessary complication, but will ensure unambiguous interactions. These details allow a student to see at a glance what module they are uploading for and whether they are on the correct account (personal/student/exam).

2.2 User Characteristics and Objectives

The current Einstein system is utilised across multiple courses/modules within DCU. During lab examinations, assignment submissions or regular lab work students have been interacting with Einstein to correct their scripts. All students within DCU computing courses are already familiar with the current implementation, and should find no issue with the docker version.

The students in DCU became familiar with the system relatively quick, because of it's simple design and straightforward upload/feedback process. Retaining this functionality is important for scaling this application externally. Therefore the system should also be quick and easy to learn for any new students outside of DCU that are not familiar with it.

The objective of the system from a user perspective is script correction as simplistic as possible. The containerised setup on the backend should greatly improve system performance, but should not hinder user interactions in any way. It is required by the user that the system be operational at any given time when an upload may need to happen. The aim of having a docker environment is to ensure that this requirement is met and that high traffic on the system does not affect an individual user's experience.

From an administrative perspective, the system should be easily deployable outside of DCU. Staff in another university should be able to take the docker image that contains the core Einstein system, feed in some server configuration files, and have it deployed to their internal network.

Wish list - Student
As a student who has used Einstein, the idea that the system is simplistic is key. Uploading a file for correction is the core functionality, meaning that adding more features to the system is not necessary. External students being introduced to this new system will not want added complication of superfluous features, again backing up the design philosophy of sticking to the basic approach currently used.

Wish list - Lecturer
From the perspective of a lecturer, simplicity is also the main concept. However, the simplicity required is in the day to day running and maintenance of Einstein. Once this system is deployed, there should be no extra input required in server management. The only time a lecturer wants to deal with the system is to upload some new test cases or sample input/output, which should slot straight into a plug and play style structure.

2.3 Operational Scenarios

Scenario 1:

John is a student in DCU. He is on his computer programming module website completing lab work. He has finished a Python script and would like to upload it to Einstein to check whether he is correct. He accesses Einstein through a link on his module website, and is redirected to the Einstein server. He is then prompted for his student username and password. After providing the correct details, he is redirected to the upload

page. Here he can drag and drop his file onto the upload area of the screen, or click browse to search the file explorer. Once the upload is successful, he is redirected to the feedback page where he can view the results of his script running against various test cases and can see whether he is correct or not.

Scenario 2:

Sarah is a student in UCD. She accesses the Einstein system through her module website which leads her to where it is hosted within her university. She has been working on a script and would like to see the feedback she received on her initial upload. She is prompted to login, and clicks the “Most recent upload” link on the upload page. She is then redirected to the feedback screen of the uploaded script from a few days prior so that she can again consult the failed test cases without having to re-upload her file.

Scenario 3:

A lecturer has setup a lab exam for his module student group. He logs into Einstein using his credentials and gains access to the student details on the system. He then uploads test cases to run with the marker script, along with expected input and output. The marker script then runs student submissions using these files. As students upload scripts during the exam, he is able to see an overview of submissions and explore any particular students uploads.

Scenario 4:

An administrator in another university would like to deploy Einstein within their school of computing. He gathers any credentials needed for hosting, i.e Apache/DNS configuration files, and feeds them into the Einstein application docker image. This image creates a running container of Einstein on the university server, and can then be accessed by individual modules.

2.4 Constraints

Performance gain - The main purpose of the project is rebuilding Einstein to improve server workload. Containerizing the application is a process to help in achieving these goals, but may require further investigation to determine precisely how much performance, if any, is gained with the new system.

Server access - Einstein is currently established on a DNS server within DCU using various configuration files. As a student, access to all of this information may not be authorised, so there could be issues in trying to acquire any credentials needed.

External servers - The scalability aim of this project is to create a docker image that can be deployed to any server in any other university. I won't have access to other university servers, and will have to replicate this idea by deploying the image to another server i.e a personal one. This is the most accessible way to show the docker image working outside of DCU.

Time - As with any project, there is an obvious time constraint to be wary of. With this project, I must be sure to follow a personal schedule of deliverables and allocate enough time to each of the core elements.

3. Functional Requirements

3.1 Logging in

Description: A user will be prompted to login when accessing Einstein. This may be done through the module website that Einstein is being accessed from.

Criticality: This requirement is the most important as a user who is not authorised should not be able to interact with the system. Logging in also lets the system determine whether a user has administrative rights or not.

Technical issues: The login should be a prompt when accessing Einstein and should not redirect a user to the system unless they are a valid student/lecturer/admin. From a security standpoint, no users outside of this should be able to access the system.

Dependencies with other requirements: None.

3.2 Uploading a file

Description: A student uploads a script to be marked by the Einstein system. The file may be dropped onto an area of the UI or browsed for by clicking upload.

Criticality: This requirement is highly important from a functional perspective as it encapsulates the purpose of the entire Einstein application.

Technical issues: A user may try to upload any file to the system. There should be a check on the file extension (Python ".py", Java ".java") before

the system accepts the script and passes it to the correction utility. Uploading a specific file may trigger some marker configuration uploaded by a lecturer to become available to students for correction. The test cases and sample output for specific tasks should not be accessible to students before uploading that same task, as they could then predict questions on an exam.

Dependencies with other requirements: Depends on a user successfully logging in and gaining access to the upload page.

3.3 Viewing Feedback

Description: After uploading a script, a user should be redirected to the feedback page where they will see the result of the script correction and any test case information.

Criticality: This requirement is as critical as the previous because they work together. The purpose of uploading a script is to verify whether it is correct, or to gain some guidance on how to get closer to being correct. Without seeing this feedback page, the user does not know the outcome.

Technical issues: The page should only redirect after successful upload/check of file extension.

Dependencies with other requirements: Depends on a file being uploaded successfully.

3.4 Admin overview

Description: An admin user or lecturer will have access to different pages.

Criticality: This requirement is critical for controlling the workflow of a course and gaining an overview of progress among students.

Technical issues: The admin access must only be given to users with correct permissions such as the lecturer. Student records should not be made available to unauthorised users (i.e other students).

Dependency on other requirements: Depends on whether an admin user has logged in.

3.5 Admin configuration

Description: An admin user or lecturer will be able to upload files for the marker script to run student submissions against. These may include sample test cases and expected input/output.

Criticality: This feature is important from a lecturer perspective in keeping the Einstein system managed and updated throughout a semester.

Technical issues: The uploading of these files should be straightforward and should require very little interaction with the system from a lecturer. The system should be setup in a plug and play style that can take in new sample files and run them with the marker script easily.

Dependency on other requirements: Depends on whether an admin user has logged in.

3.6 Recent upload link

Description: Link to view feedback of student's most recent upload.

Criticality: This is a less critical feature but is a helpful addition. This could avoid a student having to re-upload a file several times, which also helps the entire system by reducing workload.

Technical issues: Last script results must be stored and ready to display to user whenever requested.

Dependency on other requirements: Depends on a student having already uploaded a file.

3.7 Progress page

Description: Link to a page where students can compare progress to each other.

Criticality: This is a less critical feature but is a helpful addition. This link may not be suitable for all modules, or lecturer preferences.

Technical issues: Server must have all student progress ready to be displayed to a student and update individual student progression after receiving a new upload.

Dependency on other requirements: Depends on a student successfully uploading files.

3.8 Student Details

Description: The username and email of the user should be on display along with the module title.

Criticality: This is a less critical feature but is a helpful addition. Provides an overview of student details at a glance and ensures they are using the correct account within the system, which could be critical for the student rather than the system.

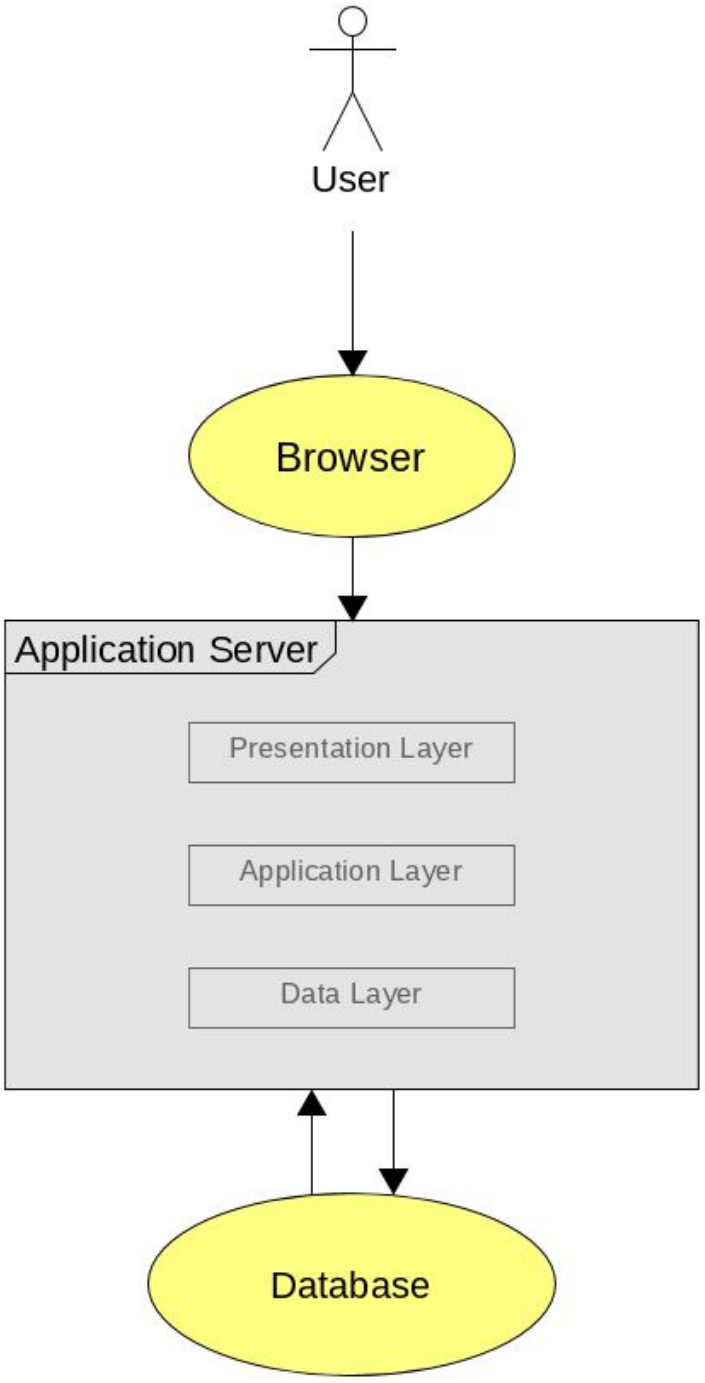
Technical issues: This information should be taken securely when a student logs in. The module title may be taken based on where the student accesses Einstein.

Dependency on other requirements: Depends on the student successfully logging in.

4. System Architecture

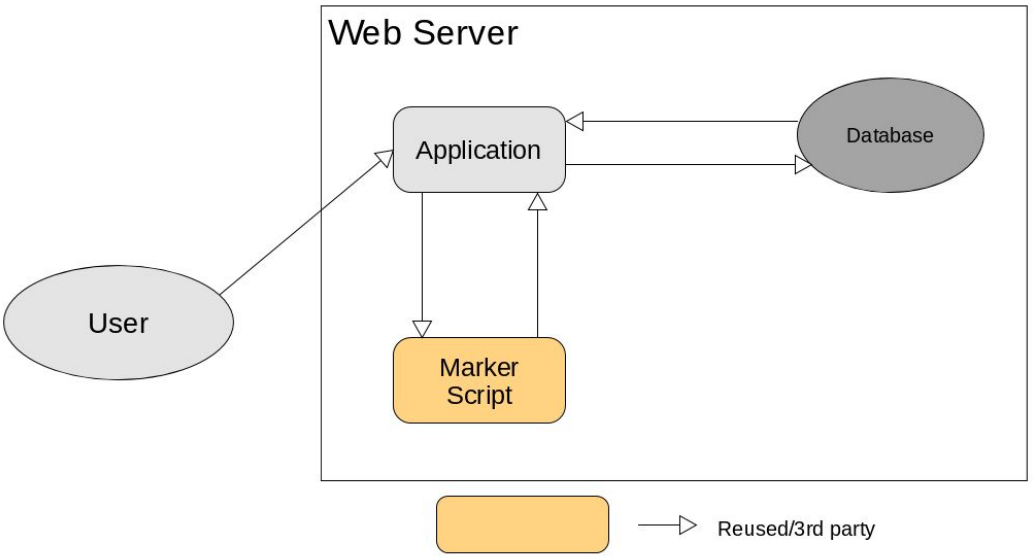
4.1 Overview

The user will interact with the system via browser using a desktop PC or laptop. The application will be hosted on a DNS server internally in whichever university it is deployed. A database is used for student records and storing student submissions. This design is the conventional approach for web application based systems.



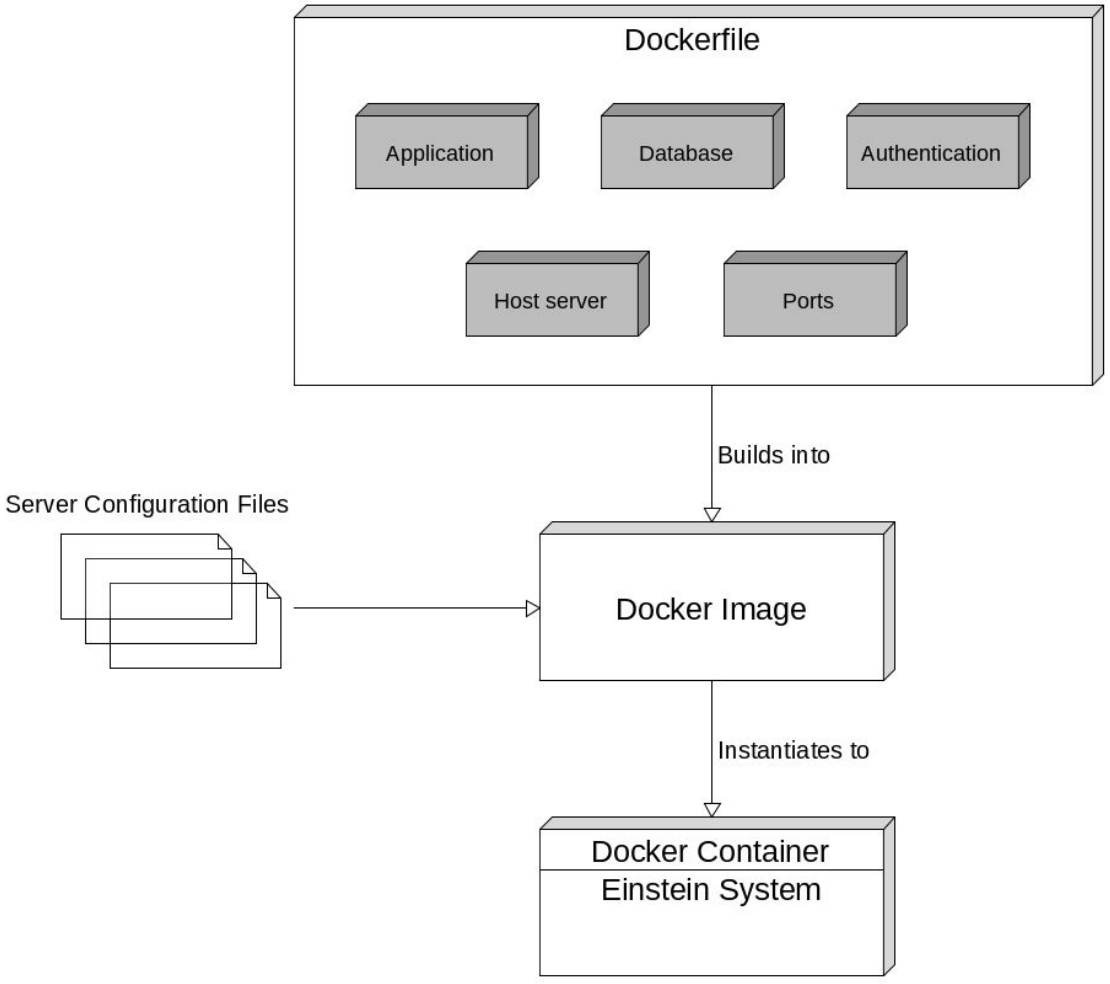
4.2 Interaction

The marker script used for the automated correction will be the same one used in the current Einstein system, but remodelled slightly to fit the docker environment. The application itself interacts with this script, passing student uploaded files and receiving feedback about their output.



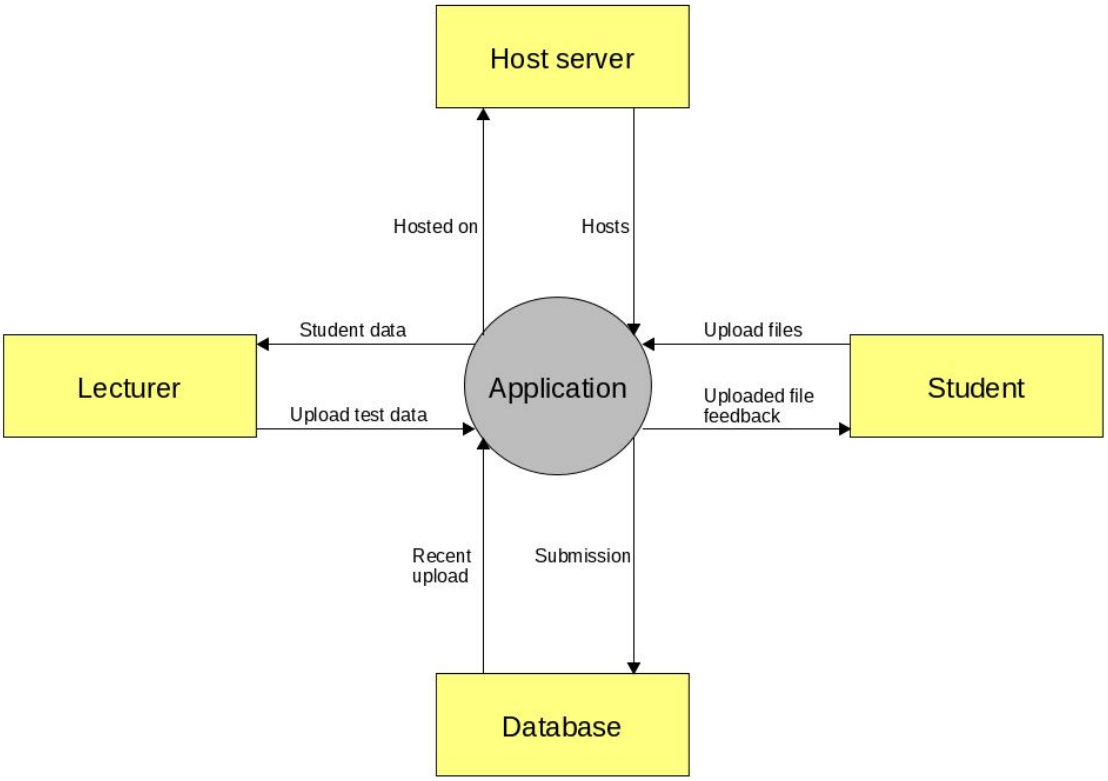
4.3 Docker Structure

The system will be laid out in standard docker environment conventions. A dockerfile will contain all necessary information about the system, and will be built into a docker image. The docker image will be the uninstantiated model of the Einstein system, and will be the core module that can be deployed anywhere. This docker image will take in server configuration files that dictate the parameters for hosting the application either inside DCU or another university. This image will then be hosted with the valid server credentials and will instantiate to a running docker container.



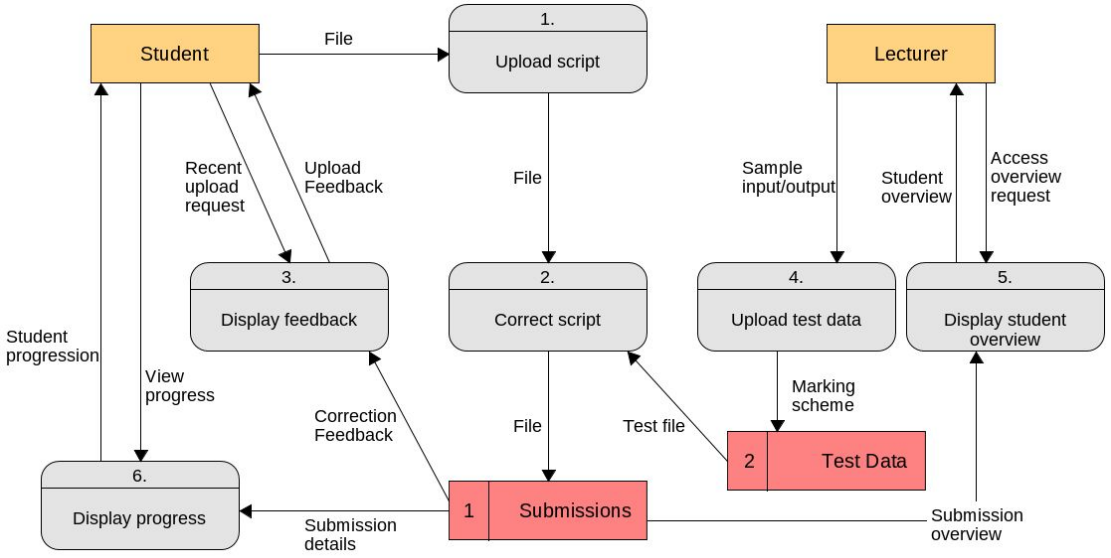
5. High-Level Design

5.1 Context Diagram



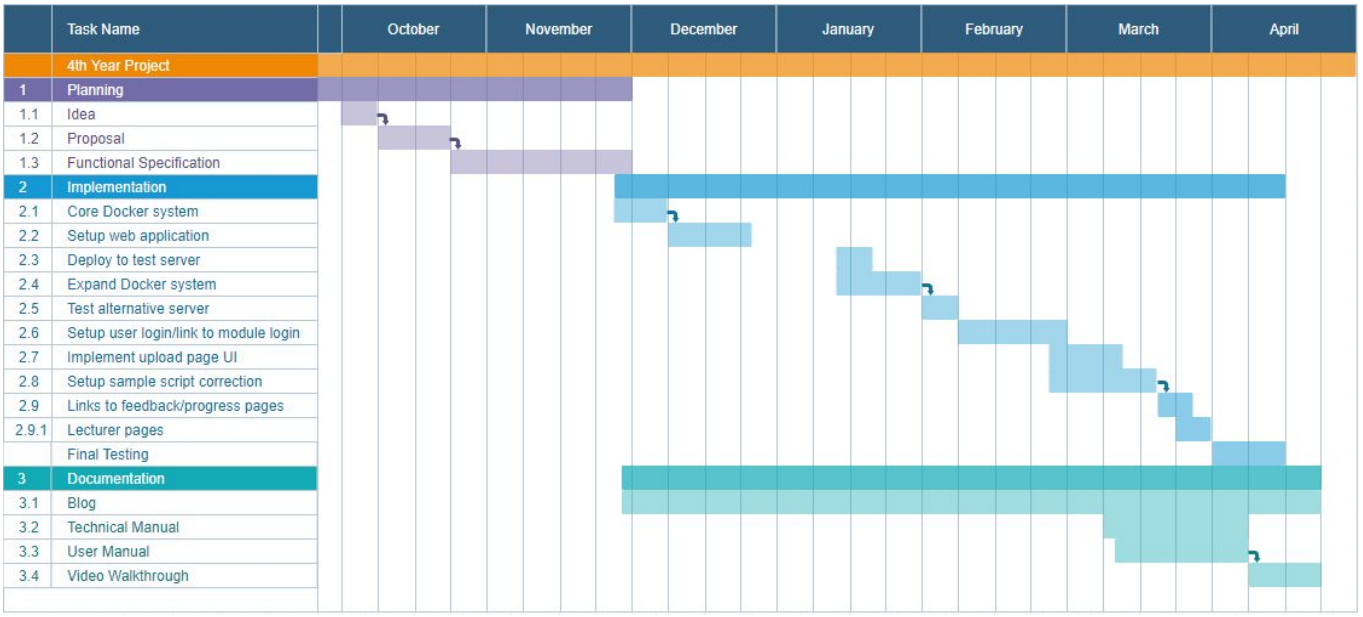
This context data flow diagram shows a high level overview of the system. Simple interactions with the application can be seen between the two core user groups; students and lecturers. The diagram also shows at a top level how the web application seen by the user can interact with the database for their file storage and retrieval. Another external entity in the infrastructure of this system is the host server, since the docker container may be deployed anywhere outside of DCU. The main artifact being passed through the system is files, either those uploaded by the student/lecturer or those being stored/viewed from the database.

5.2 Data Flow Diagram - Level 1



This level 1 Data Flow Diagram shows the general movement of data around the Docker Einstein system. The two actor groups using the system will be students and lecturers, both of which interact with the application differently. A student may upload their file to be corrected, which is the core functionality of the system. They can also access the feedback for a file directly without uploading (recent upload) or view progression with other students. A lecturer may upload test data and sample input/output to the system, which can then be used by the marker script to correct any incoming student files. There is also the option to access all student submissions directly through the user interface.

6. Preliminary Schedule



The preliminary schedule for this project can be seen above in the GANTT chart. The functional specification marks the end of the planning phase and beginning of the implementation phase. I aim to step through each major component of the system, leaving most of the lighter UI functionality to the end. I aim to carry out testing with each component implemented, however I have also allocated two weeks at the end to make sure the system performs to satisfaction. There is also a gay between December and January to cater for exams, and I hope to finish up the core design before the end of April to be clear of semester 2 exams.

7. Appendices

Docker technology:

https://www.docker.com/