KING'S College LONDON

University of London

6CCS3PRJ Final Year Orchestrator

Final Project Report

Author: Leonardo Ciocan

Supervisor: Jeroen Keppens

Student ID: 1308123

April 21, 2016

Abstract

In university settings, lecturers often teach classes of up to 200 students. Often they distribute papers to gauge the studentâĂŹs understanding of the current material being taught. The distribution, collection and analysis of these materials makes it hard for the lecturer to evaluate which parts of the material the students need help with. I have created a digital solution that helps the teacher get a better understanding of his students progress in real time.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Leonardo Ciocan

April 21, 2016

Acknowledgements

It is usual to thank those individuals who have provided particularly useful assistance, technical or otherwise, during
your project. Your supervisor will obviously be pleased to be acknowledged as he or she will have invested quite a lot of time overseeing your progress.

Contents

Chapter 1

Introduction

1.0.1 Summary

There is quite a sizeable market that would benefit from this application, there are 23729 universities around the world; a high percentage of them have Computer Science departments. Competitors Some companies have products that aim to facilitate teacher âĂŞ student interaction but I believe they are rather incomplete and further more none appear to provide tools specifically for computer science students. In other words, they provide generic tools for letting students answer questions but that is not tailor for computer science specifically. Motivation The main scope of my project was to create an easily extensible platform that may be useful for teachers specifically teaching computer science as this subject in particular requires some types of question that are more complex than simple text or choices. In other words, letting users write and execute code on my platform was a core goal. Executing arbitrary code is a dangerous so security is an important part of this system. This project is a great way to experiment with visualisation of data and how digital solutions can enrich previous analogue methods , it also allowed me to work with technologies I had not worked before and build a full product which enriched my knowledge of security , containers , server and backend technologies.

1.0.2 Platform

I have decided to build this project for the web instead of a mobile application for a number of reasons. Since this is meant to be used by a lot of students, it is a requirement that it should be as accessible as possible. Statistically speaking, a web app can reach the most people, especially account for the fact that computing students are the main target audience and they are likely to have laptops with them. The main disadvantage of making mobile apps is that to build

a good native app would mean to focus on one platform (as cross platform solutions are not up to the task for the scope of the project) which would exclude students from the process. The web is a free, universal platform and so it is the perfect medium for this project. Ideally, fully native mobile apps are further down the roadmap as they provide a better experience and further expand the pool of users that are able to use the application. The backend part of this project has been built in such a way to allow for easy expansion to other platforms. In fact, the web app could be considered just a consumer of the backend and not being renderer by the server such that another front-end could be used instead without any friction.

1.0.3 Objectives

Because the main target audience is made up of computer savvy student and lecturers, it was possible to take a few liberties with the platform. For example the questions can be written in Markdown, which is something computer scientists are already familiar with from websites such as StackOverflow and Github. Nevertheless, I aim to make a project that requires minimal guidance as the user interface will be built to be easy and intuitive to operate.

The basic requirements for this project are:

1. Let teacher create sheets made of a variety of question types
2. Allow question types to include rich formatting such as tables and code
3. Allow students to subscribe to a lecture such that they may have access to those sheets
4. Allow a teacher to create a class and invite students with a link
5. Let the teacher decide when a sheet should go âĂIJliveâĂİ and be visible by the students
6. Allow the teacher to attach model answers to each question
7. Let the teacher release model answers to the students so they may review their answers
8. Monitor the students progress and present it to the teacher in an anonymous way Allow users to write code to be executed for code questions

1.0.4 Report structure

The next section will be the background which will begin by exploring the keywords or concepts that are related to the project. It will also analyse and contrast current implementation that attempt to solve the problem described with the project's implementation.

Chapter 2

Background

2.0.1 Relevant concepts

Question Currently there are 3 types of questions:

- Multiple choice: The user is presented some choice and selects one
- Input: The user may input any piece of text, multiple solutions can exist
- Code: Users can write code to solve a posed question

The system is built in such a way that adding further questions is trivial and does not require breaking backwards compatibility or redesigning data structures used on the server.

Sheet The application revolves around Sheets. Analogue to a paper sheet handed in class, a sheet is a collection of questions of various types. Each sheet belongs to a lecture which has a teacher.

A teacher can create sheets by mix and matching any number of question types which enable quite a varied way to test their students.

Lecture A lecture is a collection of sheets. It has a teacher and students can subscribe to a lecture so they can have access to all the sheets. Dashboard An important part of this project is for the teacher to be able to monitor the student progress so that they may respond accordingly (for example, write some hints on the board). Each sheet has a dashboard which only the teacher can access. For each type of question, there is a different specialised user interface that is tailored to convey the progress of the students. A code question's dashboard widget will show the percentage of students who completed the question. An input questionâĂŹs

dashboard shows the percentage of completions, as well as a word cloud of popular words in the questions, this is a specialised control that visually conveys to the teacher common words that are being used. A choice questionâĂŹs dashboard will show the userâĂŹs completion, a bar chart for top first choices (which could help identify misleading questions amongst other things) and a transition matrix table, which shows the way students move from one answer to another.

The dashboard is also meant to be extensible, so any future question types could provide their own way of visualising student progress.

2.0.2 Differentiation from competitors

The project is meant to be an extensible platform that can be enriched with more types of question along the development process to increase the variety of sheets the teachers can create. This gives it more potential than competitorâĂŹs who over inflexible solutions that are tailor made for very specific types of questions. Furthermore, by focusing on computer science students we allow users to run arbitrary code on the platform, which while some websites allow that âĂŞ they provide that in a different setting such as code competitions âĂŞ which means they do not compete with us , as this project enabled the teacher to write a question that can be answered with code as part of a sheet along with other questions.

Chapter 3

Report Body

The central part of the report usually consists of three or four chapters detailing the technical work undertaken during the project. **The structure of these chapters is highly project dependent**. They can reflect the chronological development of the project, e.g. design, implementation, experimentation, optimisation, evaluation, etc (although this is not always the best approach). However you choose to structure this part of the report, you should make it clear how you arrived at your chosen approach in preference to other alternatives. In terms of the software that you produce, you should describe and justify the design of your programs at some high level, e.g. using OMT, Z, VDL, etc., and you should document any interesting problems with, or features of, your implementation. Integration and testing are also important to discuss in some cases. You may include fragments of your source code in the main body of the report to illustrate points; the full source code is included in an appendix to your written report.

3.1 Section Heading

3.1.1 Subsection Heading

Chapter 4

Design & Specification

This section will present an abstract view of how the system works. The project is split into 3 main components - the web app , the backend and another external backend exclusively for running user code.

4.1 Use cases

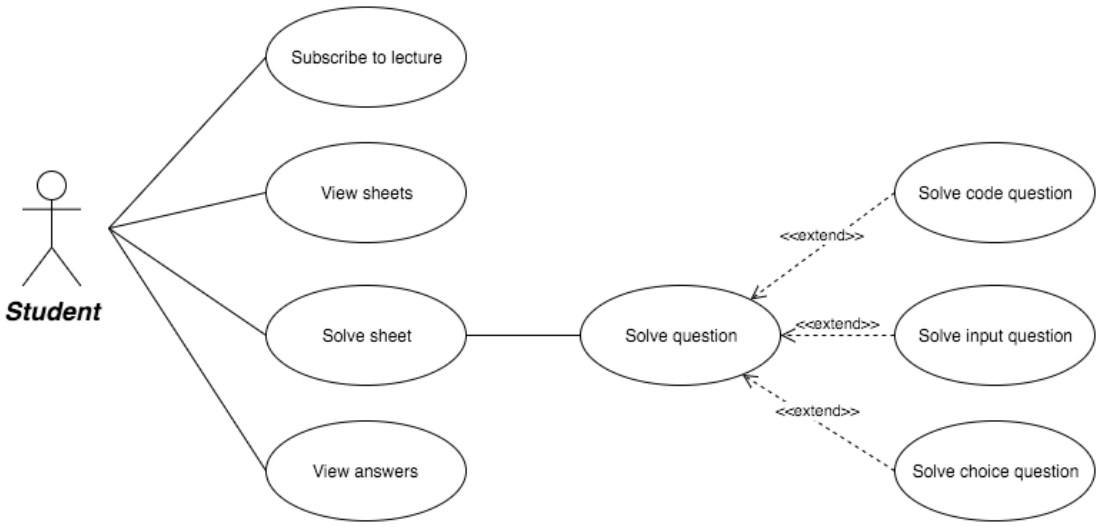

Figure 4.1: Use cases for students

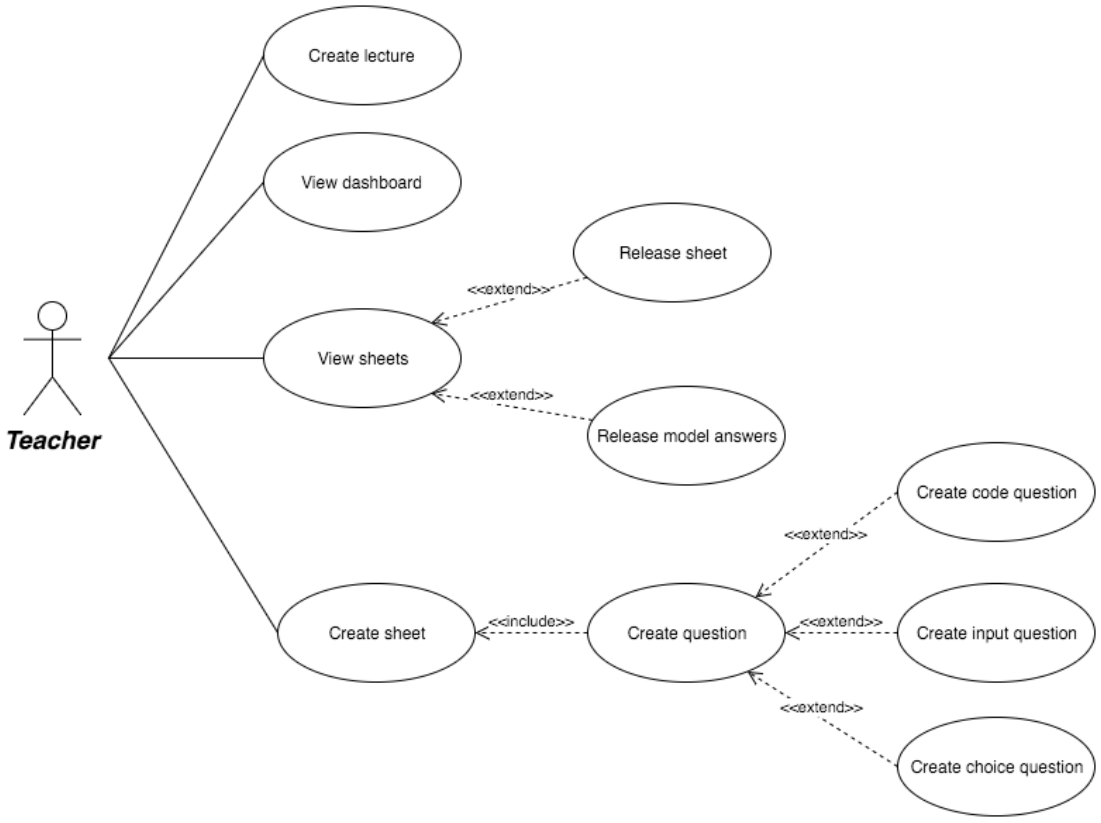

Figure 4.2: Use cases for teachers

4.2 System architecture

4.2.1 Backend

The backend provides a simple HTTP REST API that the front end can use. It is built in such a way that the front-end is a replaceable component and may be updated or switched during further development , data is encoded as JSON unless the payload represents one single value (like a number) in which case it is not wrapped in a JSON object.

The backend provides all the data needed for the application to render what it needs to show , as well as allowing the client to POST data to the server to update resources.

The endpoints of the API which may return private information have safeguards such that they will not expose information that the user should not be seeing - this is achieved by checking the user that is authenticated. This is to conform to Data Protection laws that impose limit data access where possible.

Authentication works via session cookies , this is implemented using the Devise ruby gem as reference in section 5.5. If the user is not logged in he is redirected to the login/signup page.

The database itself is a PostgresSQL database, it resides on the same server as this backend and is protected with username/password authentication.

4.2.2 Code running backend

The other backend is solely responsible for running arbitrary code written by users , this acts as a security measure in case a malicious agent manages to escape the container and compromise the server.

The endpoint this server exposes takes a piece of code , the language it's written in and a map of inputs and outputs. The code is written to a temporary file in a temporary folder. Then , depending on the language , a docker instance with the right runtime is started - when creating this instance , the temporary folder is mounted (mapped) to the docker instance inside the container's root drive. Then the file is executed inside the virtual machine with all the inputs , if the outputs generated match the given model answers then the API returns true ; else false.

Since the container only has access to the temporary folder where the code file is , it cannot harm the outer filesystem. In fact , deleting the root folder inside the container would only delete the code file and nothing else. If the code does not compile or there is a runtime error , the sever will wrap the error in a JSON object and return it to the main backend which will show it to the user.

4.2.3 Diagram

4.3 User interface

Since the user's are assumed to be somewhat technically savvy , the application takes some liberties with having a limited amount of help and information banners since it works similarly to other website the target audience is familiar with (markdown used in websites like Stackoverflow [2]). Overall the application is meant to have a flat design that looks good on any resolution and is not based on bitmaps but rather on shapes - as they are more versatile and do not need to be updated as often. Each lecture has a colour chosen by the teacher and shows up throughout the user interface so the user knows which lecture they are looking at and provides a sense of cohesion to the application.

4.4 Security

Because the project allows users to input and run arbitrary code it is important to have proper security in place. There are four layers of security for running code:

- Users can only run non-native code , currently Python and Java
- All user code runs in secure Docker containers
- Containers are contained in a separate server
- User code runs as non-root within the container

Because the code execution server and the database/main server are separate , even if a malicious agent used a vulnerability to break out of the container, he may not access any information or maliciously disrupt server operations.

Docker is used to create containers to safetly run arbitrary code , outside of a container being basically a virtual machine Docker added extra security features [1] such as restricting the capabilities of the linux kernel , namespaces and control groups.

4.5 Database

There is one database in use which contains all the information relevant to the system , it is located on the same server as the backend ; further in the report you will find some notes on how this can be improved and why. The database schema is very important since changing it could break parts of the system , it was important to chose a structure that would not drastically change later in the development cycle. Below is the structure of the tables in the system

Lecture table

This table contains all the lectures created

- ***Name*** The name of the lecture , there may be multiple lectures with the same name
- ***Author*** The teacher who created the lecture , can be any user of the system
- ***Color*** A Hex , RGB or any valid HTML colour , the current UI allows the teacher to choose a colour from a predefined set. It is used in a number of pages , often used to style all controls on the page - such as buttons or tabs

Sheet table

This table contains all the sheets created

- ***Name*** The name of the sheet
- ***Lecture ID*** The ID of the lecture this sheet belongs to
- ***Live*** A boolean that represents whether the students who are subscribed to the lecture can see this sheet (the teacher can use this to choose when to let the students start completing the sheet)
- ***Released*** A boolean value that represents if the model answers are released (and the user may no longer change his answers)

Question table

This table contains all the questions created

- ***Title*** The body of the question as Markdown formatted text (stored as plaintext)
- ***Sheet ID*** The ID of the sheet this question belongs to
- ***Data*** JSON that provides metadata needed to render the question
- ***Type*** An integer that defines the type of question
- ***Correct Answer*** A JSON object that defines metadata needed to compute whether the answer is correct or not
- ***Model Answer*** A string that represents an example answer

Answer table

This table contains all the answers created

- ***Data*** A plaintext representation of the student's answer
- ***Question ID*** The question that is answered
- ***User ID*** The ID of the student who created this answer
- ***Result*** Plaintext field , used to store the cached computed result of the question.

Statistic table

This table contains all the statistics created when students answer questions

- ***Answer ID*** The ID of the answer this statistic is about
- ***Data*** JSON that contains relevant statistical information about the action
- ***Kind*** The type of statistic this represents

Subscription table

This table contains all the subscriptions

- ***Lecture ID*** The ID of the lecture the user is subscribed to
- ***User ID*** The ID of the user subscribing to the lecture

Chapter 5

Implementation

5.1 Languages and Frameworks

5.1.1 Front end

Since the front end of the project is a web app , HTML and CSS were used to create the user interface. However for the programming Typescript was used instead of Javascript.

Typescript is a super set of Javascript , made by Microsoft. It adds useful language features to Javascript such as a type system , generics , classes and more. The addition of these features helps the code base be more readable and more maintainable.

In particular having types allows the code to be more easily refactored and reduces redundant documentation (such as including type information in comments).

It is compiled to Javascript before being served to the user and all the functionality is compile-time only so there is no performance penalty in using it.

User interface The fronted of the application is mostly tailored , in other words the UI components are made specifically for this project. One of the reasons for this is because lecture colour is an important visual cue in the application's visual design and colouring the components of existing platforms such as Bootstrap can be glitchy as they were not meant to be used in such a way.

ReactJS was used to create the UI components. React is a light javascript library used to create reusable components in a functional way. Using it has helped create powerful , contained reusable components that can be customised externally without needing to dig into

their implementation (which can be useful for future development) . They also allow for simple colouring by passing around the lecture colour when creating a component.

5.1.2 Backend

The main backend written in Ruby , using the Ruby on Rails web framework. The main reason for choosing this framework was that it is proven to be reliable and has a very healthy developer ecosystem. Thus there are a lot of up to date libraries (or 'gems') available to extend the core functionality. In particular , the Typescript gem works seamlessly as part of the Rails asset pipeline to convert the typescript files to javascript.

Another backend was written for handling the execution of user code inside docker contains. This was done using Python and Flask , a python web micro framework. A micro framework was used instead of rails because this part of the system does not involve any advanced features found in full web frameworks (authentication , models , migration for example)

On the same server as the python backend , Docker is used to create instances where the user code can run safely.

5.2 Development Tools

The backend was developed using RubyMine and PyCharm. The front-end was written using Visual Studio Code , mainly because it was the only editor at the type that supported TSX files (Typescript + React inline).

5.3 Rails backend analysis

This section will explain the structure of code within the backend responsible for the main functionality.

5.3.1 Controllers

Controllers are split by page that the user sees , as well as an additional ApiController which is engineered to be decoupled from the HTML such that it could power a potential mobile or native client.

Application Controller Has an action *index* that renders the index page

Lecture Creator Controller Has an action *index* that renders the page where the teacher can create a lecture

Lecture List Controller Has an action *index* that renders the page where all the lectures the user created and subscribed to are listed , as well as a *subscribe* method that handles rendering the subscriptions page

Sheet Creator Controller Has an action *index* that renders the page where the teacher can create sheets

Sheet Dashboard Controller Has an action *index* that renders the dashboard page that the teacher sees

Sheet Editor Controller Has an action *index* that renders the sheet editing page where the student can answer a sheet. It also contains an action *update_sheet* that handles updating the user's answer

Sheet Manager Controller Has an action *index* that renders the page where the teacher can manage all the sheets of his lecture

API Controller Contains the following methods:

- ***subscribe*** Subscribes the current user to a lecture with id *lecture _id*
- ***statistics_for _question*** Collates and returns the statistics for a given question
- ***completions*** Returns the answers who correctly solved a question
- ***lectures*** Returns an object with two properties , *subscribed* which contains the lectures the user is subscribed to and *created* which the user created (is a teacher of)
- ***lecture*** Returns a single lecture information given an id
- ***full _sheet*** Returns information about a sheet. Object returned contains the following field : *lecture , sheet , questions , answers , modelAnswers* and *percentage*
- ***sheets*** Returns a list of sheets
- ***create _sheet*** Creates a sheet given a sheet and an object which represents the new sheet. It will return any errors if any

- ***update _sheet*** Updates the name , live status or result status of a given sheet
- ***create _lecture*** Creates a lecture with a given name and colour
- ***delete _sheet*** Deletes a given sheet
- ***delete _lecture*** Deletes a given lecture
- ***lecture _users _count*** Return the number of users subscribed to a lecture
- ***update _lecture*** Updates the name or colour of a lecture
- ***stats*** Collates statistics and any metadata computed from them (such as transition matrices)
- ***search*** Searches for a keyword and returns all sheets and lectures that contain it
- ***user _info*** Return the email of the logged in user

//typescript stuff here

5.4 Unit testing

Unit testing was used to make sure changes and additions to the system did not break or alter any previous functionality.

Purpose:

1. To make sure existing functionality was not compromised
2. To ensure that the database settings worked as intended (defaults , uniqueness etc)
3. Components of the system all complied with security , such as preventing user's from editing other people's date or gaining access to data they are not allowed to see

5.5 Third party libraries

5.5.1 React

A javascript library that allows the project to have isolated , reusable components that build up all of the user interface

5.5.2 Chromath

A javascript library that can manipulate colours. It is used within the dashboard to compute darker shades of the lecture color - this is useful for both the charts (each section of the chart is a different colour) and for the transition matrix (the higher the transition count , the darker the shade).

It is statically included in the project.

5.5.3 CodeMirror

A javascript library that provides an embeddable code editor. Used to let user's write their own code.

5.5.4 Markdown-it

A javascript library used to render Markdown , in both the title preview on the sheet creation page and the actual sheet that the user sees.

5.5.5 Highlight

A javascript library to highlight code syntax , used internally by Markdown-it

5.5.6 JQCloud

A javascript library used to render a word cloud in the dashboard for input questions.

5.5.7 Levenshtein-ffi

A ruby gem that allows fast calculation of Levenshtein distances for answer's in the input question's dashboard.

5.5.8 pg

A ruby gem for Postgres support

5.5.9 Devise

A ruby gem that simplifies authentication

5.5.10 ReactRails

Support for React in Rails applications

5.5.11 Typescript Rails

Automatic compilation of typescript files as part of the rails asset pipeline.

Chapter 6

Professional and Ethical Issues

Either in a seperate section or throughout the report demonstrate that you are aware of the **Code of Conduct & Code of Good Practice** issued by the British Computer Society and have applied their principles, where appropriate, as you carried out your project.

6.1 Section Heading

Chapter 7

Results/Evaluation

7.1 Evaluation

In this section we will explore whether the project has met the requirements that were set in the requirements section of this report as well as other general aspects of the software that can be evaluated.

7.1.1 Limitations

There were some things that were simplified or postponed for the sake of keeping in line with deadlines and achieving all the compulsory requirements that I have set. Thus the system does have some limitation (solutions to those limitations will be explored in the next section)

- Code question can be either Java or Python based. This is a limitation imposed to decrease the surface area for bugs that would arise from allowing more languages.
- Sheets cannot be edited after they are created. This is because changing the questions as the sheet is live could invalidate existing statistical data retrieved from users completing the question.
- Teacher cannot kick out or manage users. This is because currently the system aims to anonymise users so they may not feel discouraged from completing the questions
- The website is glitchy and may not work at all on some mobile devices. This is especially true for pages where the user may enter a core

7.1.2 Security

While the system is not meant to contain any sensitive data , it is important that the users of the system can be assured that their data is private. There are safeguards in place to ensure that a malicious agent may not successfully acquire information they are not entitled to , this functionality is unit tested to ensure it stays this way across releases of the software. Another layer of security is in the system that executes the student's arbitrary code. The code runs in a docker container and can only run within either the java virtual machine or the python interpreter. Not allowing low level code to run discourages any traditional security holes such as buffer overflows. Furthermore , even if someone manages to execute code such that they break out of the language runtime and the docker container - the containers run on a server separate from the main server. Thus they cannot access the database or compromise the system.

7.1.3 Overall

The main purpose of the system was to provide teacher's a way to handle handing out sheets digitally and analyse the result easily in real time. It was built in such a way that not only fulfilled this requirement but also provides a platform that is easily extendable such that more question types can be added without having to modify the existing infrastructure.

Chapter 8

Conclusion

8.1 Thoughts on project

This project has taught me a number of things , the most important of which I believe to be making decisions about how time is allocated. When this project began , my main focus was data aggregation and I had planned to do a lot of work regarding engaging the user in novel ways to try to gauge their under understanding of a subject. However it became apparent to me (after being advised by my supervisor) that data visualisation and extracting information from the aggregated data is equally important and in fact , the data is not of much use in an unrefined form. Another thing I learned was the importance of proper mocking up and feature freezes as I have rewritten several components of the system multiple times during the course of the project. This was especially time consuming as I have written most of the UI elements from scratch.

I firmly believe that this project has fulfilled its purpose - which is to create a platform for teacher to user interaction. Both the underlying framework and the user interface have been prepared in such a way that they can extended very easily with more question types and more visualisation options.

8.2 Future work

There are a number of ways to improve the project , the system is implemented in such a way that adding more functionality can be done seamlessly without breaking or having to alter any major components of the system.

Ideas for improvements:

- More question types. For example questions that revolve around web development (such as asking the user to create a html/css layout from an image) or unix command line tools (asking the user to complete some command line workflow). This should be easy to implement as the database schema for questions is very flexible and does not make any assumptions about the type of data a question expresses

- Better user management for the teacher , allowing them to kick , filter and analyse student progress. This would involve adding a lot more UI components and possibly other pages

- Allowing teacher to edit the title/body of questions after the sheet is created. Easy to implement the technical aspect of it , however it would not fit within any current place in the UI and would require an additional page

- Password enabled lectures , or alternatively geo-locked lectures. Fairly simple to implement but needs to have extra logic in place to deal with changing the password and so on

- Mobile apps for Android and iOS to expand the pool of students who have a device that can use the project. Takes time but overall the backend would not have to be changed much as it already exposes most functionality via a simple REST API

References

[1] Docker. Docker security. https://docs.docker.com/engine/security/security/.

[2] Stackoveflow. Markdown editing help. http://stackoverflow.com/editing-help.

Appendix A

Extra Information

A.1 Tables, proofs, graphs, test cases, ...

The appendices contain information that is peripheral to the main body of the report. Information typically included in the Appendix are things like tables, proofs, graphs, test cases or any other material that would break up the theme of the text if it appeared in the body of the report. It is necessary to include your source code listings in an appendix that is separate from the body of your written report (see the information on Program Listings below).

Appendix B

User Guide

B.1 Instructions

You must provide an adequate user guide for your software. The guide should provide easily understood instructions on how to use your software. A particularly useful approach is to treat the user guide as a walk-through of a typical session, or set of sessions, which collectively display all of the features of your package. Technical details of how the package works are rarely required. Keep the guide concise and simple. The extensive use of diagrams, illustrating the package in action, can often be particularly helpful. The user guide is sometimes included as a chapter in the main body of the report, but is often better included in an appendix to the main report.

Appendix C

Source Code

C.1 Instructions

Complete source code listings must be submitted as an appendix to the report. The project source codes are usually spread out over several files/units. You should try to help the reader to navigate through your source code by providing a "table of contents" (titles of these files/units and one line descriptions). The first page of the program listings folder must contain the following statement certifying the work as your own: "I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary". Your (typed) signature and the date should follow this statement.

All work on programs must stop once the code is submitted to KEATS. You are required to keep safely several copies of this version of the program and you must use one of these copies in the project examination. Your examiners may ask to see the last-modified dates of your program files, and may ask you to demonstrate that the program files you use in the project examination are identical to the program files you have uploaded to KEATS. Any attempt to demonstrate code that is not included in your submitted source listings is an attempt to cheat; any such attempt will be reported to the KCL Misconduct Committee.

You may find it easier to firstly generate a PDF of your source code using a text editor and then merge it to the end of your report. There are many free tools available that allow you to merge PDF files.