KING'S College LONDON

University of London

6CCS3PRJ Final Year Orchestrator

Final Project Report

Author: Leonardo Ciocan

Supervisor: Jeroen Keppens

Student ID: 1308123

April 24, 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 24, 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

The rise of technology such as mobile devices has been sudden and unexpected compared to any other part of human history. While many industries have quickly adapted to the changes , education is one field where embracing technology is still underway. One of the most striking examples of how technology can improve education is the invention of the printer , which contributed greatly to literacy in developed countries going from 10% to 80% in a very short time [7].

A number of other studies show that introducing technology in a learning environment can have a positive impact [3] and that there is a international movement to augment learning with technology [6]. Today , we live in a world where every student in developed countries has access to a electronic device - in fact , for subjects such as engineering and computer science , virtually every student has access to such a device [2] [1] . However , in university lectures , paper handouts are still used; their purpose is for the students to be able to test how well they understand the current concept being taught and for teachers to gauge the student's understanding of the subject and adjust their teaching schedule accordingly.

A digital replacement of these handouts has the potential to increase student engagement and minimise the friction of distributing , collecting and analysing data.Overall allow the overall turnaround time for the teaching process to be as small as possible.

1.1 Motivation

Handouts can ask a variety of questions ; they can have images , tables , formatted text and more. If a digital alternative is to replace this , it must be flexible enough to support a variety

of question types. Further more , paper handouts allow the user to draw or express their answer in a number of ways , this can be a disadvantage since the students could write invalid answers (i.e select two answers in a single answer multiple choice question) however it also provides a lot of flexibility. A digital solution must solve both problems by providing flexible , tailored ways to express the questions and to answer them in a way that is valid and yet allow the user to express their creativity.

To account for this goal , the end product build will be tailored for computer science students. This is to showcase the features of the platform and allow for a focused development cycle that would prove the usefulness of a digital solution for this problem. Part of this tailoring will be a question type that let's students write their own code and have it executed and checked by the system.

The data visualisation part of this project provides a number of ways to explore novel ways to show the aggregated data in ways that is useful to the teacher and is tailored for each question type.

In section 8.2 (Future work) it will be discussed how the system , once shown to work as intended , can be extended further and be tailored for all the subjects that make up higher education. This is because the project will be developed as a platform and make it so adding question types will be simple.

1.2 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 talk about the platform chosen for the project and solutions that already attempt to solve the problem we're exploring.

Afterwards , requirements for the project will be laid out - the report will explore how they were solved in a later Evaluation section.

The design section will explore the system from a architecture point of view , laying out the structure of the system. This will be later expanded on in the implementation section , where the internals of the system will be documented.

After some reflections on ethics , the conclusion will bring the report to an end with some thoughts on the project and plans for the future of the project.

Chapter 2

Background

2.1 Relevant concepts

2.1.1 Teacher

The word teacher is used throughout the project and its associated materials. It does not necessarily refer to an accredited teacher ; any user of the system can create their own lectures while subscribing to others. Generally speaking , a teacher is one who created a lecture. As a concrete example , a TA[1] may also benefit from using this in labs.

2.1.2 Student

A student is one who joins a lecture and completes handouts. A user of this system may be both a teacher and a student (of different lectures).

2.1.3 Question

Questions are the core component of this project since handouts are composed of questions. In the context of this project a question is not only a piece of text - a question is composed of a body (the text of the question itself) as well as a type which defines the structure of the answer. Currently there are 3 types of question:

- Multiple choice: The user is presented some choice and selects one
- Input: The user may input any piece of text, multiple solutions can exist

[1] Teaching assistant

- Code: Users can write code to solve a posed question , the user's code can be run against an arbitrary amount of test cases to confirm it works for a range of inputs

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.

Each question also has a correct answer and a model answer. The correct answer is an object or piece of text that defines what the corrrect answer for the question is. The structure depends on the question type , for example the choice questions's correct answer is defined as a number (the index of the correct answer) , but for input questions it is a regex.

Likewise the model answer is type dependant ; it is shown to the user after the lecture so they may understand how the question is meant to be solved.

2.1.4 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 , all which have 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.

Going live When the teacher creates a sheet , it is initially not "live". In other words , students who are subscribed to the lecture cannot yet see it. The teacher may decide when to show the sheet.

Releasing answers After a sheet has gone live , the teacher may chose to end the sheet and stop students from submitting answers anymore. In this state the student can see the model answer's created by the teacher when designing the sheet as well as their score for that sheet.

2.1.5 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 that the teacher created for the lecture.

Each lecture has a colour , this colour is used throughout the user interface when the student is interacting with content relating with the lecture. This provides a sense of cohesion and makes the user more aware of the lecture they are currently in.

2.1.6 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 for that specific type.

A code question's dashboard widget will show the percentage of students who completed the question.

An input question's 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 question's 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.

2.2 Existing solutions

There are some products that have attempted to tackle the teacher - student interaction in the classroom. One of the most popular is called Socrative [2] , which allows teachers to ask the students quizzes. However the system is very limited and only allows text questions , where the answers can be a choice or piece of text. This is not flexible and rich enough to replace traditional methods , only to agreement them. However , it has been downloaded over 1000000 times just on android [3] , which shows there is a clear market for such solutions.

Another service that could be potentially a competitor is Moodle [4] . It is not usually used in classroom settings but it does provide support for making quizzes that the students can answer and the teacher can analyse the questions. However this feature is a minor part of Moodle and it does not provide any of the real time advantages of other solutions , including the one described in this project. Some teachers use this to collect code written by user's and then script a solution to execute and mark the answers - however this is not a good solutions as

[2]Socrative `http://www.socrative.com/`
[3]Socrative on Google Play `https://play.google.com/store/apps/details?id=com.socrative.student`
[4]Moodle `https://moodle.org/`

students get little feedback and the process is not automated.

Overall , one of the goals of this project is to bring together the advantages of all those different solutions in a package that is extensible and yet retains the ability to be easy to use for both students and teachers.

2.3 Platform

This project is built as a web app 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 computer science 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 starting 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 , especially as tablets and hybrid devices are increasing in popularity with students[8].

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.

Chapter 3

Requirements

T

Because the main target audience is made up of computer savvy student and lecturers, it was possible to take a few liberties on the front-end of 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, the project aim to require minimal guidance as the user interface will be built to be easy and intuitive to operate.

3.1 Functional requirements

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
9. Allow users to write code to be executed for code questions

3.2 Non-functional requirements

1. User code must be run in a secure environment so that user data is not at risk
2. Web app must maintain a reasonable size by using shared components and merging duplicate code
3. Users should not be able to view or change data that does not belong to them
4. Allow easy extension of question types
5. Not store identifying information

Chapter 4

Design & Specification

The project is split into 3 main components:

- Web app : Accessed by the user on their browser. The purpose of this is to provide a interface to the service on student's laptops.
- Backend : A ruby on rails application running on a remote server also containing the database. It is in charge of retrieving data from the database as well as provide endpoints for the client to interact with all entities of the system. It also acts as a proxy between the client and the following server:
- Code execution server : A server that has the purpose of securely running code written by students. It is isolated from the client and backend for security purposes.

4.1 Use cases

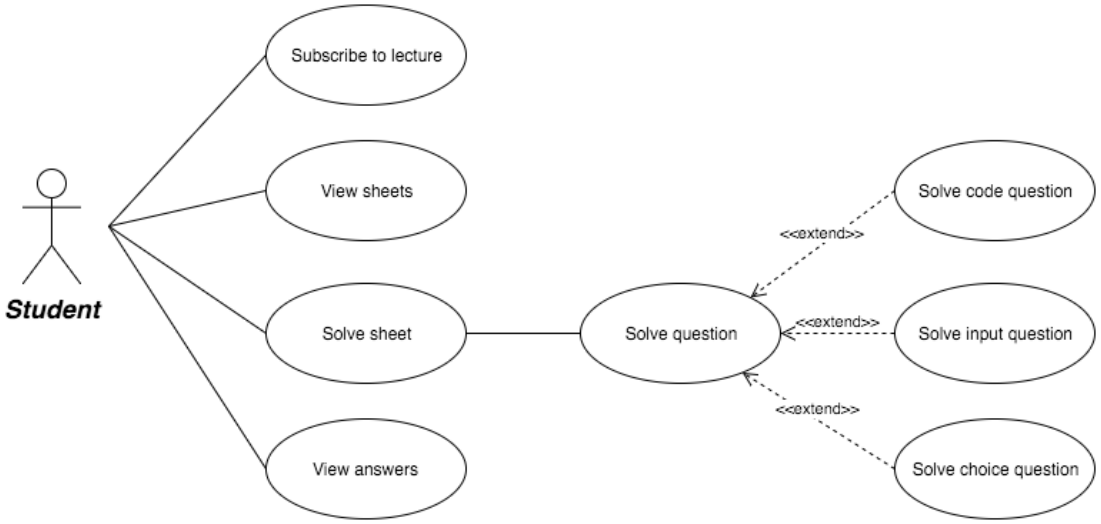

Figure 4.1: Use cases for students

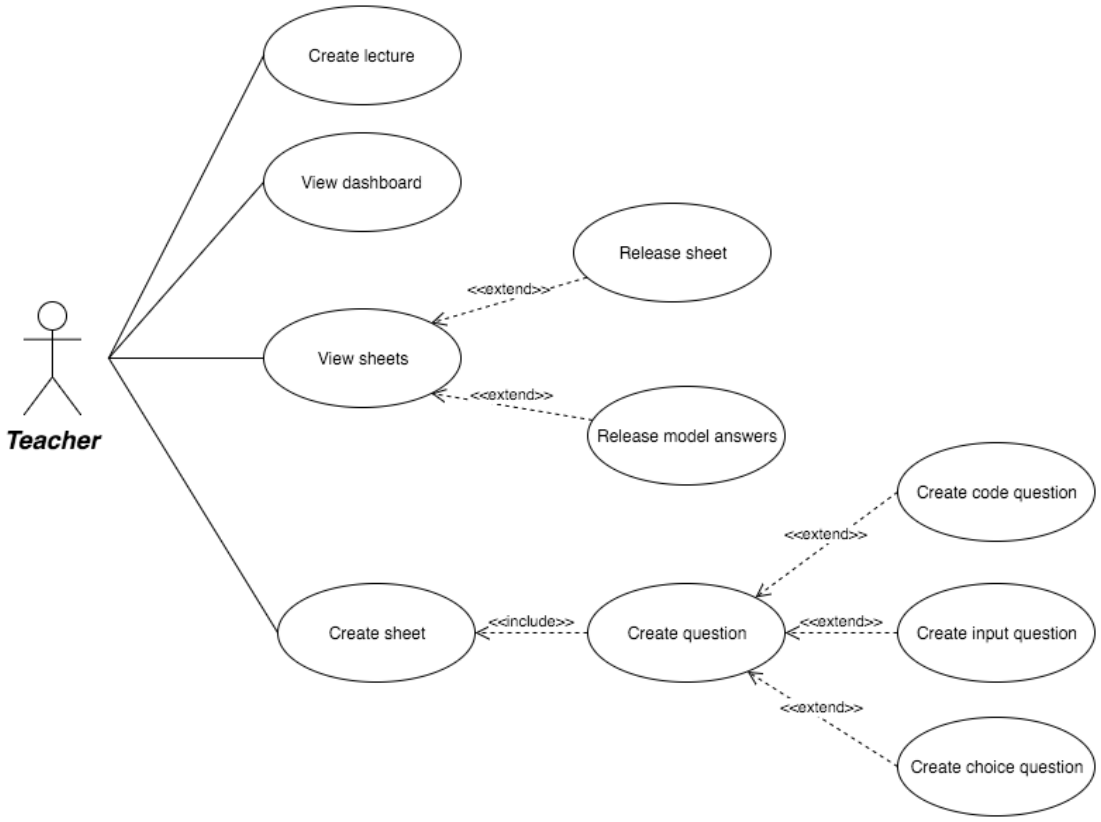

Figure 4.2: Use cases for teachers

4.2 System architecture

4.2.1 Diagram

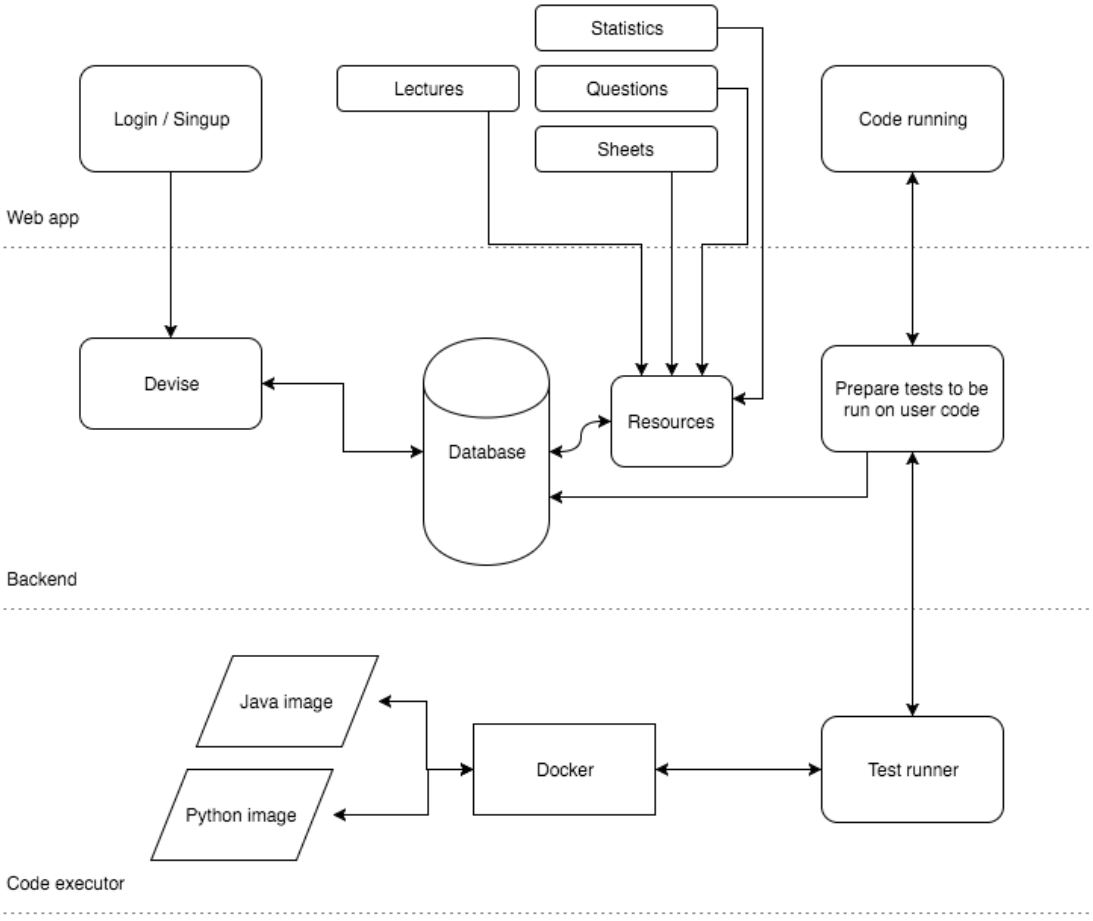

Figure 4.3: System architecture diagram

4.2.2 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.

It 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 5.7. 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.

4.2.3 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 from 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 pass it along to the user.

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 is used in websites like Stackoverflow [10]). 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 safely run arbitrary code , outside of a container being basically a virtual machine Docker added extra security features [5] such as restricting the capabilities of the linux kernel , namespaces and control groups.

4.5 Statistics

The dashboard is used to convey to the teacher how the students are faring with the sheet. Some useful metrics can be extracted implicitly from the answers , for example the percentage of students that completed questions correctly. However there is useful metadata about user behaviour that is lost by just looking at the current state of the answers. Past states , changes in answers , first choices are all fragments of informations that when put together can paint a more complete picture of how the students are solving the work.

To account for this , the system uses statistics - flexible objects stored in the database that record these temporary events. For example , a user changing his answer from A to B is a relevant piece of data the teacher might want to know about.

These objects are collected and appropriate data structures are computed from them when showing the dashboard. In the example above , a single change in answer is not very useful - however , when grouped with all other statistics it can be statistically relevant. In this case the changes of answers are converted to a transition matrix so that it may be easily rendered on the client.

4.6 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 Routes

See appendix B

5.3.2 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

5.3.3 Statistics collection

Statistics are generated when the student answers the sheet questions. They are used to capture metadata that cannot be extracted from just the current state of the user's answers.

To abstract their creation away from other parts of the system , the equals operator is overridden and creates the necessary statistics. The relevant code is shown below:

```
def data=(val)
    if self.question.type == 0

      write_attribute :result , {:correct => val == self.question
          .correct_answer}.to_json

      if self.data == nil or self.data == ""
        #this means that this is their first choice
        Statistic.create :answer => self , :kind =>
            @@stat_first_click , :data => val
```

```
      else
        Statistic.create :answer => self ,
                         :kind => @@stat_change ,
                         :data => {"from" => self.data , "to"=>
                            val}.to_json.to_s
      end

    elsif self.question.type == 1
      write_attribute :result ,
                      {:correct => val.match(Regexp.new(self.
                         question.correct_answer)) != nil}.
                         to_json
    end

    write_attribute :data , val
end
```

If the question type is one that should generate a statistic , a statistic is created. In the case the question is a choice question and the user moved from an existing answer to another , a transition statistic is created. This records the change in an object with a ”from” and ”to” field.

After a statistic is generated , it can be acquired by the client for the dashboard by just acquiring the /statistics/ endpoint with GET over http. The object that represents the transitions is computed this way:

```
transitions = Hash.new {|h, k| h[k]= Hash.new{ |h, k| h[k] = 0}}
c.each{
    |transition|
         transitions[transition['from']][transition['to']] += 1
}
```

A nested hash object is created and using the from and to values a 2d structure is represented , thus generating a grid of numbers with headers that can be easily rendered into a table by the client.

5.4 Front end analysis

The front end is mostly composed of React components; React uses a special file type called JSX[4] which allows the mixing of javascript and HTML in the same file. JSX files go though a compilation phase to become plain JS files.

In this project , Typescript is used instead of javascript , so the files written in the project are TSX files , which compile down to JSX and then to JS.

5.4.1 Entry points

The combination of Typescript , React and Rails does not have a lot of official support , this is both because of the fact that typescript/react are newer technologies and that rails favours a javascript alternative called Coffeescript. However a gem called typescript-rails[1] provides support for including typescript in the asset pipeline so that typescript .ts files are transparently compiled to .js files so they may work in the browser.

There are however issues when dealing with .tsx files , which contain Typescript + HTML code , the gem does not include them in the pipeline as it most likely has not been updated to support this as of yet. To account for issue , each page has a entry.ts file and that file marks .tsx files as dependencies to it such that the typescript compiler may merge them (after converting TSX to standard typescript). The entry.js is references in the HTML that the server renders and it draws the root React component (often a page component as you will see below).

5.4.2 React components

LectureCreatorPage The state contains a ***name*** and ***color*** which are selected by the user and send using POST when the user presses create lecture.

LectureItem The component takes a lecture object and renders a single Lecture box

LecturePage Takes two arrays of lecture objects , one for lectures created and one for subscribed lectures , then renders two sets of LectureItem

SubscribePage Takes a single lecture object and prompts the user to subscribe to it

SheetCreatorPage Takes a single lecture object ; this page let's the user create a sheet. It has the following state:

[1]Typescript rails gem `https://github.com/typescript-ruby/typescript-rails`

- ***items*** An array of questions that the user is creating for this sheet
- ***name*** The name of the sheet
- ***description*** The description of a sheet
- ***errors*** A map , where the key is a number representing the question index and the value is a list of errors for that question
- ***dragging*** Whether the user is dragging an element to create a new question

InputCreator A component for creating an input question. Takes the following parameters:

- ***color*** The color of the lecture
- ***question*** The question this element is editing/representing
- ***key*** Used internally by React
- ***onDelete*** A function called when the question is deleted
- ***errors*** A list of strings , representing errors with this question

CodeIO An object with an input field and output field (both strings)

CodeCreatorIO A component with an input textbox and an output textbox. Used to represent a test case in the code question. It takes a CodeIO object which contains a Input and Output field , changes are reflected inside this object

CodeCreator A component for creating a code question. It allows the user to choose the language and create tests which are instances of CodeIO and are represented by CodeCreatorIO. It has the following properties:

- ***color*** The color of the lecture
- ***question*** The code question this component represents
- ***index*** The index of the question
- ***onDelete*** A function called when the question is deleted
- ***errors*** A list of strings , representing errors with this question

ChoiceCreatorAnswer A component that represents an answer for a choice question. Takes an answer object , the lecture color and a onDelete handler.

ChoiceCreator A component for creating a choice question. It allows the user to create choices for the user to select.. Takes the following parameters:

- ***color*** The color of the lecture
- ***question*** The question this element is editing/representing
- ***key*** Used internally by React
- ***onDelete*** A function called when the question is deleted
- ***errors*** A list of strings , representing errors with this question

SheetDashboardPage The page where the teacher can see the stats of a sheet. Takes a lecture and a list of questions.

InputStats , CodeStats , ChoiceStats Components that shows the statistics for the specific type of question. Takes a question and the lecture color.

ActivityIndicator A component that shows a pie chart that represents the user's who have completed a question succesfully. Takes the following parameters:

- ***question*** The question
- ***color*** The color of the lecture
- ***percentage*** The percentage of students who solved the question
- ***correct*** The number of students who solved the question
- ***wrong*** The number of students who did not solve the question

SheetEditPage The page where the student can complete the sheet. Takes the following properties:

- ***questions*** A list of questions in the sheet
- ***shet*** The sheet the user is answering
- ***lecture*** The lecture the sheet belongs to

- ***answers*** The user's answers
- ***modelAnswers*** The model answers for this sheet
- ***percentage*** The percentage of questions that the user solved

InputQuestion , CodeQuestion , ChoiceQuestion Components that show the questions to the user and let them answer it. Each takes a question , an answer , a color , a model answer and a boolean that represents whether the sheet has model answers released.

SheetManagerPage The page where the teacher can manage the sheets in the lecture.

SheetControl A component that represents a sheet and let's the teacher release the model answers , delete it or access the dashboard

SheetsPage The student view of all the sheets , takes a lecture.

SheetItem A component that represents a sheet that the student can access. Takes a sheet object.

There are also some shared components:

CheckBox A checkbox , takes a boolean that represents whether it's checked , a color and a onChange handler.

ColorPicker Let's the user pick select a color. Takes a onPicked handler that is triggered when a color is selected.

Header A component that sits at the top of the pages , used for navigation. Takes the following properties:

- ***title*** The title of the page , shown on the left
- ***name*** The name of the user logged in
- ***color*** The color of the lecture
- ***onBack*** Handler , triggered when back button is clicked
- ***foreground*** The foreground color of the header
- ***hideBack*** Whether the back button should be shown.

It also has the following state:

- ***showMenu*** Whether the user menu should be shown
- ***showSearch*** Whether to show the search panel
- ***searchSheets*** The sheets found by searching
- ***searchLectures*** The lectures found by searching
- ***query*** The search query typed in the search box

IDFactory Used to generate new IDs for React keys

LCButton A reusable colourable button , takes a piece of text for the content , a color , an onClick function and a style object.

MDPreview A reusable component for previewing Markdown. Takes a piece of markdown code and renders it.

SearchDropdown Used to show search results. Takes the following parameters:

- ***shouldShow*** Whether the search panel should be visible
- ***sheets*** The sheets found by search
- ***lectures*** The lectures found by search
- ***color*** The color of the lecture
- ***query*** The text entered in the search bar

SegmentedButton A segmented button control that let's the user select between options. Takes the following parameters:

- ***color*** The color of the lecture
- ***labels*** An array of strings representing the choices the control provides
- ***style*** Additional styling provided externally
- ***onSelected*** A function triggered when the selection changes
- ***selectedIndex*** The currently selected item
- ***itemPadding*** The padding of each item in the control

TextArea A stylised text area

TextBox A stylised text box

5.4.3 Component styling

A powerful pattern in react components is to generate css at runtime , this enables features such as the dynamic coloring given by the lecture colour. The project uses some share components that are meant to be used in different pages. Often the page needs a slighly different version of the control , so instead of making flags for all posible variations the component exposes a style property.

```
interface LCButtonProps{
    text : string
    color: string
    onClick? : Function
    style? : any
}
```

The style can be any object , then when the component is rendered this object is merged with the static style. If the style property contains a new css rule it will get added ; else if the static style already has that rule , the style property one overrides the static one. It uses jQuery to do this:

```
if(this.props.style != undefined){
        $.extend(containerStyle , this.props.style);
}
```

5.5 Code execution server analysis

As mentioned in the Design section:

> ”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."

To create and execute code on the docker instance , the python program that handles the code execution server generates a bash command that can be executed. The program then executes it and hooks into the process STDIN and STDOUT so it can simulate input and get the output.

To illustrate this , here is how the command string for a java program is made. Note Code.java is written to the disk previously and folder is the temporary folder where it's located.

```
cmdStr = "sudo␣docker␣run␣-v␣{0}:/code/␣-i␣java␣/bin/bash␣-c␣\"␣
    javac␣/code/Code.java;cd␣/code/;java␣Code\"".format(folder)
```

The -v switch mounts the folder to the /code/ path on the container so that the code will be at /code/Code.java

5.6 Unit testing

Unit testing is used to make sure changes and additions to the system did not break or alter any previous functionality. More precisely it has the following advantages:

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

5.7 Third party libraries

5.7.1 jQuery

Used in a number of places , such as the above example on merging objects. It is also a dependency of other libraries.

5.7.2 React

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

5.7.3 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.

5.7.4 CodeMirror

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

5.7.5 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.

5.7.6 Highlight

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

5.7.7 JQCloud

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

5.7.8 Levenshtein-ffi

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

5.7.9 pg

A ruby gem for Postgres support

5.7.10 Devise

A ruby gem that simplifies authentication

5.7.11 ReactRails

Support for React in Rails applications

5.7.12 Typescript Rails

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

Chapter 6

Professional Issues

This project and all of its components abide by the Code of Conduct as detailed by the British Computer Society [9] for legal and ethical reasons.

The usage of open source code has greatly accelerated the development of this project and has provided functionality that was outside my means to create in the time frame allocated to this project. As such , every library and instances of code used in this project that were not written by me are clearly stated. Code that is not marked as such can be assumed to written completely for this project.

Since this project revolves around real people in a collaborative setting it was paramount that extra care was put into ensuring the privacy and anonymity of the end users were respected and built into the platform. Minimal user collection is done as the application contains no ads and does not profit from selling the user's information. There are also information barriers to prevent malicious users from exploiting other user's information.

Chapter 7

Results/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.

One of the core requirements was to let the teacher create different types of question and collate them into a sheet. Since this was a core mechanic of the system , it was one of the first things to be implemented. The teacher can create choice questions (student selects one of multiple choices) , input questions (students can write any text into a text box) and code questions (students can write their own code to solve a question. Furthermore the platform was built with this requirement in mind such that it can be extended later.

Markdown is used to write the body of the questions , this allows the teacher to include rich formatting within the question text. This useful because it can make questions more expressive and easier for the user to understand.

Teachers can easily create a lecture and invite students to subscribe to it by sharing a link. After they are subscribed they can access all the sheets that the teacher has chosen to go live. As student complete the sheet , the teacher can monitor the results so that they may know how to proceed further with the lecture. When the teacher decides the sheet time is over , he may release the model answers so the user can see their score and what the correct answer is. If the student accesses a sheet after the model answers are released , they will not be able to submit any changes , but may review their answers.

Docker and container technology allow the system to run student code in a safe way. That coupled with the fact that the languages do not have pointer arithmetic / unsafe operations and run in a non root mode ensures the risk of malicious attack using the student code as a

vector is minimised if not completely avoided.

The front end has a shared folder which contains components that can be reused across different pages ; they are generic components but can be customised without the need to modify their implementation.

Another requirement was the extensibility of the platform ; the code and database schema are made in such a way that adding more question types and their respective view in the dashboard is trivial. By using JSON for the core definitions of the questions and answers allows the platform to be well positioned for further development.

Anonymity is an important part of this platform , in fact the only identifiable information asked of the user is their email address. Furthermore data is partitioned such that a user may not get access to data they are not supposed to see , they may also not update or attempt to modify or create a resource that belongs to a resource they do not have permission to interact with.

7.0.1 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

7.0.2 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.

7.0.3 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] Why laptops? http://www.eos.ncsu.edu/soc/laptops/why_laptops.

[2] Lee Rainie Aaron Smith and Kathryn Zickuhr. College students and technology. http://www.pewinternet.org/2011/07/19/college-students-and-technology/.

[3] Thomas L. Davies Angeline M. Lavin, Leon Korte. The impact of classroom technology on student behavior. http://www.aabri.com/manuscripts/10472.pdf.

[4] Facebook Developers. Jsx in depth. https://facebook.github.io/react/docs/jsx-in-depth.html.

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

[6] Robert B. Kozma. Technology and classroom practices: An international study. https://www.stcloudstate.edu/tpi/initiative/documents/technology/Technology%20and%20Classroom%20Practices.pdf.

[7] Sanjay Kumar Pal. 21st century information technology revolution. http://dl.acm.org/citation.cfm?id=1399619&coll=portal&dl=ACM.

[8] Pearson. Student mobile device survey 2015. http://www.pearsoned.com/wp-content/uploads/2015-Pearson-Student-Mobile-Device-Survey-College.pdf.

[9] British Computer Society. Code of conduct. http://www.bcs.org/upload/pdf/conduct.pdf.

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

Appendix A

User Guide

A.1 Instructions

This appendix will contain a short guide to using the product , but note that the product is designed to be intuitive enough not to require a guide.

Accessing the web app You can either run the project (after installing dependencies) or use the test server at `http://178.62.36.214/lectures/`

Logging in If you are not logged in you will be prompted to either login or signup. Afterwards you'll be redirected to the lectures page.

Creating a lecture On the lectures page , press the + button next to the label that says "Your own lectures" , enter a lecture name and select a colour , then press "Create"

Creating a sheet Click your lecture then click "New sheet" , this will take you to the sheet creator page - just fill in the questions you want to make and press "Create Sheet"

Going live At this point the sheet is not yet visible to the students , press the "Live" button on the sheet card to let students see it

Monitor progress of sheet To monitor students , press the "Dashboard" button on your sheet card.