# Quiz Tool

Final Report for CS39440 Major Project

*Author:* Mr. Michal Goly (mwg2@aber.ac.uk)
*Supervisor:* Mr. Chris Loftus (cwl@aber.ac.uk)

26th April 2017
Version: 1.0 (Draft)

This report was submitted as partial fulfilment of a BEng degree in
Software Engineering (G600)


Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

# Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name: Michal Goly

Date: 26th April 2018

# Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Michal Goly

Date: 26th April 2018

# Acknowledgements

I am grateful to coffee.

# Abstract

Student engagement and live knowledge monitoring are vital in the provision of good quality lecture content in 2018. It is important to make sure audience understands concepts presented, and quizzes can help lecturers judge students' understanding in real time.

Aberystwyth University currently uses Qwizdom[1] live polling tool during the provision of some lectures and practical sessions. The university operates under a single license forcing lecturers to book sessions before they can use the tool. Due to human nature, session hijacking occasionally occurs to the bemusement of both students and lecturers. For example, students could be shown biology slides half way through their geography lecture.

This project focused on the design and development of an in-house built Quiz Tool, enabling multiple lecturers to use it at the same time and potentially making Qwizdom redundant in the future. This ambition could only be achieved if the project was of high quality and its future maintainability was considered at all stages of the design and development.

The Quiz Tool allows lecturers to login using their Google Single Sign-on[2] credentials, upload their PDF lecture slides and create *Lectures* in the system. Each *Lecture* can be then edited, and eligible slides can be marked as quizes. True/false, single and multi choice style quizes are supported. Once a lecturer is happy with their *Lecture*, he can broadcast it and receive a session key which can be shared with students. Lecture slides will be shown to all students and the lecture can be delivered in a traditional fashion up to the moment a slide has been marked as a quiz. Students will then be able to answer the question and polling results will be presented in real time to the lecturer. Lecture sessions broadcasted in the past are kept, and students' answers can be exported as a PDF report for future analysis.

The tool is composed of a back-end with an associated database, and two front-ends. One for lecturers and one for students. Finally, the tool has been successfully developed using an agile methodology, adjusted for a single person project.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# Chapter 1

# Background & Objectives

## 1.1 Background

### 1.1.1 Motivation

I enjoy programming, and this project seemed to involve a lot of coding. I was also excited to build a complex, real time system, while applying my previous software engineering experience and learning new technologies at the same time. I knew I would be given a lot of freedom to choose the most appropriate tools for the task, and incorporate modern software development practices, including continuous integration and containerised environments, to deliver good quality software. I was also keen on developing something that could be actually useful to the university once I leave Aberystwyth. It is more motivating to develop a product, while knowing it could be potentially used in real life, as opposed to being forgotten after the submission. I have experienced being presented with wrong slides during a lecture in my first year at university, so I was happy to address the problem of a single session Qwizdom license with my tool.

### 1.1.2 Technology Considerations

#### 1.1.2.1 Programming Languages

The initial proposal was to create the classroom quiz system using Java[3] to run it natively on Android[4] mobile devices. The lecturer was supposed to create quizzes on his teaching machine, by interacting with the system using a web front end. Lecture slides were then supposed to be broadcasted to his audience, and they could use their mobile phones to answers questions, which would be then sent back to the lecturer for analysis. I have had previous experience with native Android development, therefore this approach seemed like a reasonable option. The only problem was, iOS[5] devices are very popular in the United Kingdom, and developing an app for Android would exclude a good percentage of students from being able to actively participate in lectures presented. I have therefore started to think about alternative approaches.

The second possibility was to use React Native[6], a JavaScript[7] framework allowing developers to create mobile applications in JavaScript and compile it down to both iOS and Android. This approach would still require the web front end for the lecturer to be developed, and a natural choice would be to use React[8] to keep the learning curve as low as possible.

The final alternative considered, was to develop the whole tool as a web application. This way both the front end for lecturers and students could be developed using the same framework. Members of the audience could participate in lectures by accessing the web application using web browsers installed both on their mobile phones, regardless of the operating system, and their laptops. I considered both React and Angular 4[9], since I have already briefly used it before. React is a library for developing user interface, whereas Angular is a web development framework. The only caveat with using Angular is that the developer needs to learn TypeScript[10], which compiles down to JavaScript.

#### 1.1.2.2 The WebSocket Protocol

The Quiz Tool was supposed to allow a lecturer to broadcast his slides to all the students participating in a lecture. This requires a bidirectional communication protocol between the server and the clients. For example, if a lecturer emits the next slide of his presentation, this change should be pushed to the back end, and then the back end needs to be able to send the new slide to all the students. Students' clients should also be able to push quiz answers back to the back end, so they can be presented to the lecturer in some form.

The WebSocket Protocol can be used to create such real time systems. It enables two-way communication between a client and a remote host, making it ideal for instant messaging, gaming applications, and the Quiz Tool. It uses a single TCP connection for trafic in both directions[11].

#### 1.1.2.3 Prototyping

I have followed various tutorials to quickly prototype proof of concept applications, in order to learn more about the technologies which could be useful during the Quiz Tool development. I started with the Socket.io chat tutorial. Socket.io is a JavaScript library enabling bidirectional event-based communication, and uses the WebSocket Protocol internally[12]. I have created a chat application following one of the tutorials on their homepage[13]. I have also learned the basics of Angular by following the Tour of Heroes tutorial[14], and finally tried to understand how Angular could work together with Socket.io by following the MEAN Socket.io tutorial[15].

### 1.1.3 On-site vs External Hosting

The initial plan to handle authentication in Quiz Tool was to use the university LDAP[16]. This way, lecturers would be able to provide their university credentials, which would be checked using the Bind operation against the university directory of users, to check if a given person is authorised to use the tool. This would require the Quiz Tool to be deployed to a production environment running within the university intranet. The alternative was to deploy the application to an external cloud hosting.

### 1.1.4 Similar Tools

Qwizdom[1], is the tool currently used by the university to embed quizzes into presentations to judge students' understanding of the content presented. It has to be installed on lecturer's machine as it is a desktop application. Qwizdom integrates with Microsoft PowerPoint[17], by adding an

extra toolbar allowing the lecturer to insert quiz questions into his presentations and then broadcast them. Once a presentation is started, a session key appears which can be shared with the audience. They can then use the key to join the lecture and answer questions once they become available.

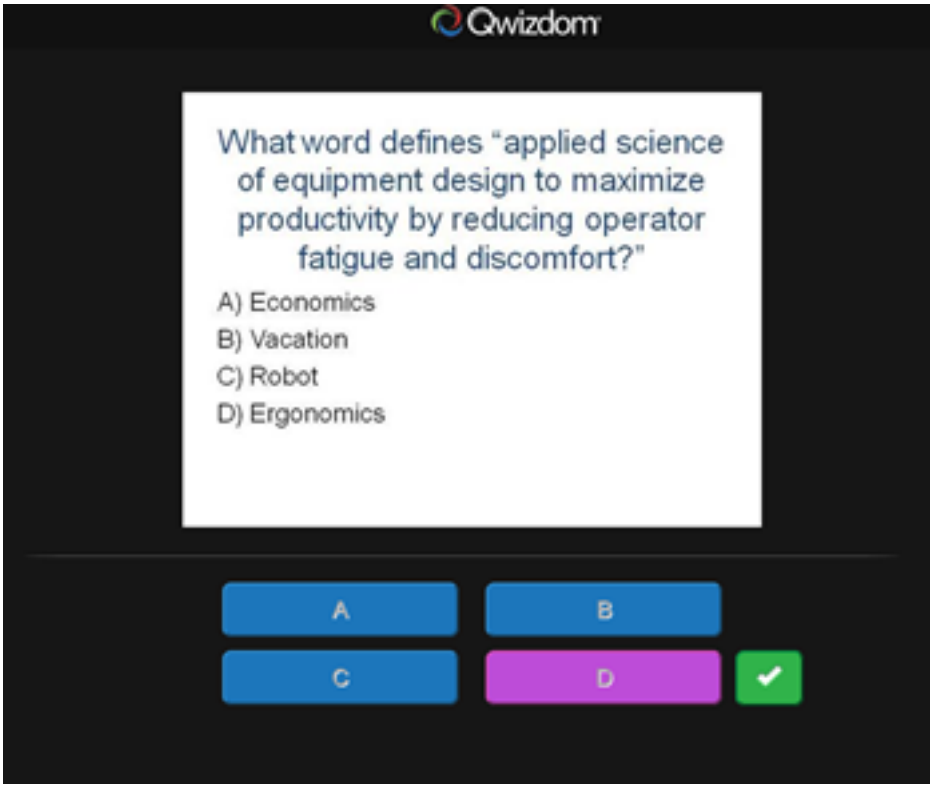


Figure 1.1: Qwizdom web view for the audience

## 1.2 Analysis

### 1.2.1 MEAN Stack

Having considered various methods of developing the Quiz Tool, I have decided to choose the web application approach. Allowing students to access the tool using both their mobile devices and laptops, combined with the relatively low learning curve both for me, and for anyone maintaining the software in the future, made it the best option in my opinion. Furthermore, I have decided to develop the application entirely in JavaScript based technologies using the MEAN stack[18].

MEAN stack consists of four elements:

- **M**ongoDB is a JSON document storage NoSQL database
- **E**xpress is a minimalistic JavaScript web development framework
- **A**ngular is a front end web development framework
- **N**odeJS is a JavaScript engine

Front ends for both lecturers and students would be written in Angular, NodeJS would be used on the back end together with Express, and the persistence layer would be provided by MongoDB.

### 1.2.2 Docker

Each part of the application would be containerised using Docker[19], and containers would run together in the same fashion on the developer's machine, and the build engine using docker-compose[20] container orchestration tool. Docker is an open source engine which can be used to wrap an application and all its dependencies into a lightweight container that can run on any machine capable of running containers[21]. Docker-compose on the other hand, is a tool for defining and running multi-container Docker applications.

### 1.2.3 AWS Production Environment

Unfortunately, the on-site hosting provided by the university to students to deploy their final projects to was inadequate for the DevOps infrastructure I wanted to put in place. I wanted to have a machine capable of running docker-compose, and some form of continuous integration agent. The LXC debian containers[22] were not compatible with either docker-compose, or Jenkins[23]. The Quiz Tool would be therefore deployed to the production environment provided by an external cloud provider AWS[24], and an alternative build agent would be used. The GitHub Student Developer Pack[25] offers $150 worth of credits for the Amazon cloud, which is why AWS was chosen.

### 1.2.4 Build

Continuous integration and deployment would be provided by Circle CI[26], as it is relatively easy to work with and is very similar to Travis[27], while not being overly complex like Jenkins[23]. Its major advantage over its competitors is the ability to build projects stored in private repositories, free of charge to a certain amount of build hours per month. Continuous integration and deployment are software development industry practices that enable teams to reliably release new features and products. Code is integrated and merged frequently, which shortens the release cycle, improves code quality and team's productivity[28].

Build of the Quiz Tool should at the very least:

- Checkout the source code from the version control
- Run all the tests
- Deploy the tool to the production environment when a production build runs

### 1.2.5 Authentication and Security

LDAP authentication could not be used with the production environment provided by AWS. The alternative would be to use the Google Single Sign On[2], where lecturers could login to the tool using their Google credentials. The major drawback of this approach is that anyone with a Google

account can login into the tool as a lecturer. The major advantage is, that the credentials handling would be outsourced to Google, making the Quiz Tool more secure and the tool would not have to store hashed passwords by simply relying on the confirmation of user's identity by Google.

Having said that, Google ids would still be stored in MongoDB to uniquely identify lecturers in the future, and to know who is the owner of lectures in the system. Due to the fact that the interaction of the Docker containers making up the Quiz Tool would be managed by docker-compose, the MongoDB container could be hidden from the outside world and could be only accessed from the server container running the back end code. This would make the database more secure.

### 1.2.6 Top Level Requirements

Even though the tool would be developed using an agile approach, as described in the next section of this document, certain top level functional requirements have been identified before the development work had started. The initial conversation with the client (the supervisor of the project) yielded the following requirements:

- **FR-1** - Add a login page for lecturers. As a lecturer it should be possible to login into the tool using lecturer's credentials.
- **FR-2** - Add a login page for students. As a student it should be possible to join an ongoing session using the session code provided by the lecturer.
- **FR-3** - Add a dashboard for lecturers to upload their lecture slides to.
- **FR-4** - Add the ability to add quizzes to lectures.
- **FR-5** - Add the ability to broadcast lectures.
- **FR-6** - Add the ability to export lecture session results in some format for future analysis.

## 1.3 Process

### 1.3.1 Version Control

Git[29] would be used for the version control during the development of the Quiz Tool, together with the GitHub[30] web-based hosting service. The source code would be stored in a private repository. There would be a `master` production branch, and suitable guards would be added so that new code cannot be pushed directly to the production branch. The feature-branch git workflow[31] would be used and each `feature-branch` would need to have an associated issue (story) on GitHub. Then a pull request could be opened between the `feature-branch` and the `master` branch, and Circle CI would checkout the code, run all tests and report back to GitHub to either allow the merge or block it, depending on whether the build was successful or not. Once the pull request is merged into `master`, a production build would run, where Circle CI would checkout the code, run all the tests and finally automatically deploy the new version of the Quiz Tool to the production environment provided by AWS.

A Kanban board provided by the ZenHub Chrome plugin[32] would be used. It would allow issues to be grouped into epics, and each story could have a corresponding amount of points assigned to it, depending on the likely complexity of the task. For example a single point would correspond to half day of work.

### 1.3.2 Development Methodology

The application would be developed using the SCRUM methodology adjusted for a single person project. The development work would be split into weekly sprints. Each sprint would start with sprint planning, where issues would be created and their complexity would be estimated using ZenHub story points. During the week, issues would be tackled, and the progress could be monitored using the Kanban board. GitHub labels would be used to quickly differentiate between different types of issues. For example there would be a different coloured label for front end tasks, back end tasks, documentation, DevOps etc. Time commitment would also be tracked on day to day basis using a Google Sheets spreadsheet[33]. Each sprint would end with a sprint retrospective, where a document would be produced containing a list of things that went well during the week, and things that could be improved. Weekly meetings with the client (supervisor) would provide immediate feedback on work done and allow re-adjustment of the approach necessary to stay on track to deliver the software before the deadline. Finally, velocity would be tracked and burn down charts automatically produced following each iteration. This would give more confidence in estimating the future work.

# Chapter 2

# Design & Implementation

The structure of this chapter follows the agile methodology used to develop the Quiz Tool. Each section contains information about a sprint, allowing the reader to gain more understanding of how the design and the tool itself evolved over time.

## 2.1 Sprint 1 - Hello Quiz Tool

### 2.1.1 Sprint Planning

The first sprint of the Quiz Tool focused on setting up the DevOps of the project, and deployment of a ”hello world” version of the tool consisting of the front end, back end and nginx[34] running together using docker-compose. It was also important to investigate how to best structure the application to include both the front and the back end of the application in a single GitHub repository. The following subsections cover the most important aspects of the sprint, and the entire list of estimated stories can be found in the Appendix C of this report.

### 2.1.2 Application Structure

The main goal of using docker-compose, was to have the whole application running in the same manner locally on the developer's machine, during testing on Circle CI, and in the production environment. This meant the application had to be containerised using Docker, and containers had to be able to communicate with each other appropriately. This was even more difficult considering Socket.io had to be incorporated, to allow real time broadcast of lecture slides to students in the future. I have decided to create a prototype of a very basic chat application, containerised using Docker and orchestrated using docker-compose. The prototype had to be written in the MEAN stack, and use Socket.io to prove it was possible to make all technologies work together.

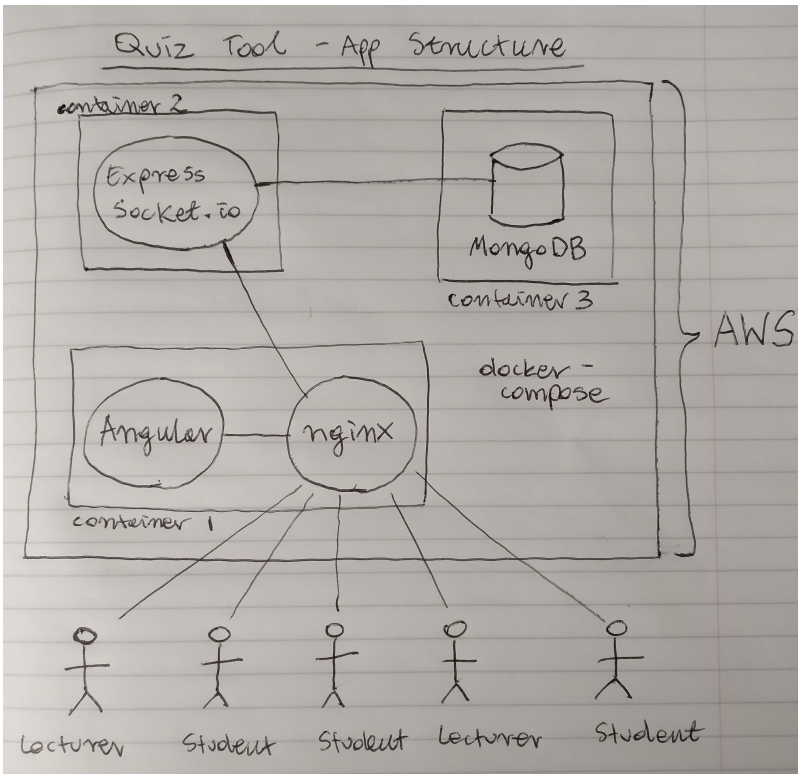


Figure 2.1: The proof of concept application structure

```
version: '2.0'
services:
  client:
    build: client
    ports:
      - "80:80"
    links:
      - server_node
  server_node:
    build: server
    links:
      - database
  database:
    image: mongo
    ports:
      - "27017:27017"
```

Figure 2.2: The docker-compose.yml file describing the tool's structure

#### 2.1.2.1 Front End Container

The front end container consisted of Angular 4 and nginx reverse proxy. The structure has been based on the *Dockerized Angular 4 App (with Angular CLI)* repository[35]. The Socket.io client dependency has been added to allow sending messages to the back end using sockets. The initial `Dockerfile` included in the Appendix E of this report as the Figure E.1 uses the multi-stage

build added in Docker 17.05. The Angular app is compiled to JavaScript and HTML files during the initial stage of the build, and then these files are copied to the nginx public folder to be served to clients. This results in a lean, production ready image.

#### 2.1.2.2 Back End Container

The back end container included in the Appendix E of this report as the Figure E.2 consisted of a Node.js runtime, the Express framework and the Socket.io engine capable of pushing messages to clients using Sockets. The `Dockerfile` illustrates the very basic Node container.

#### 2.1.2.3 Database Container

Finally, the MongoDB Docker image has been pulled automatically from the official mongo Docker Hub registry[36].

### 2.1.3 Continuous Integration

Circle CI has been integrated with the GitHub repository containing the source code of the Quiz Tool. Every time a pull request was made, Circle CI would be notified. It would then assign a virtual machine build agent from a pool, and spin up a clean build environment. It would then checkout the code and run the steps specified in the build config file, before reporting if the build was successfull back to GitHub.

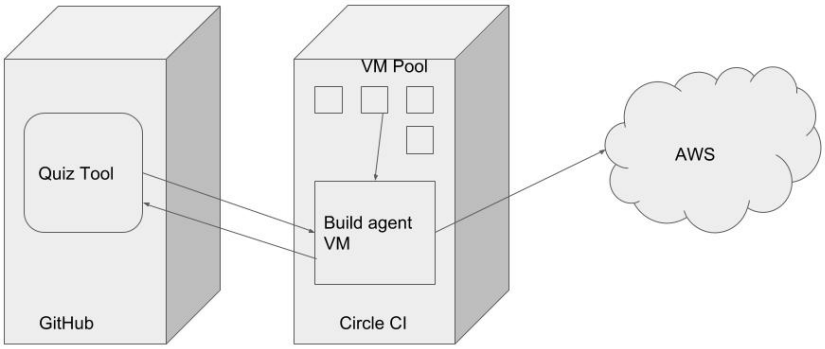


Figure 2.3: Continuous Integration and Deployment

The `.circleci/config.yml` file included in the Appendix E of this report as the Figure E.3, describes the initial build steps of the Quiz Tool. Code is checked out from the version control, all the dependencies necessary to perform following steps are installed, the project is then built and started using docker-compose, before the `curl localhost` command checks if the application is up and running. Finally, if the current branch being built is `master`, the `deploy.sh` bash script runs, which deploys the application to production.

### 2.1.4 Production Environment

The production environment of the tool is hosted on the AWS cloud. The AWS Elastic Beanstalk has been chosen specificaly, as applications in various programming languages can be deployed with ease, wihout having to worry about the infrastructure running these applications[37]. The Multicontainer Docker AWS Elastic Beanstalk[38] environment, creates a single Amazon EC2[39] instance and uses ECS (Amazon Elastic Container Service)[40] to coordinate container deployments to multicontainer Docker environments. The similarity with docker-compose, means the Quiz Tool would behave similarly on developer's machine, in testing and in production.

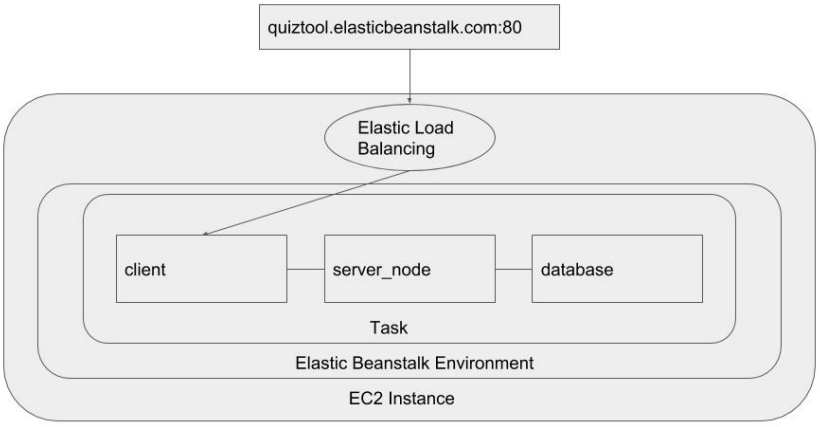


Figure 2.4: Quiz Tool Production Environment

`docker-compose.yml` files define how docker-compose should run Docker containers together. `Dockerrun.aws.json` is the equivalent configuration file to specify relationships between Docker containers running in the Multicontainer Elastic Beanstalk environment. The format of the config file included in the Appendix E of this report as the Figure E.4 is very similar to the docker-compose configuration files. The major difference is that the AWS Elastic Beanstalk does not build Docker images itself, and images have to be pulled dynamically from Docker registries. Docker registries are simply servers used for storage and distribution of Docker images.

#### 2.1.4.1 Circle CI and AWS Integration

The build agent automatically deploys the tool to production when the `master` branch is being built. The bash script included in the Appendix E of this report as the Figure E.5 performs the actual deployment, and the approach is based on the examples[41][42]. The `$AWS_ACCESS_KEY_ID` and the `$AWS_SECRET_ACCESS_KEY` environment variables have been added to the build configuration using the Circle CI web panel. The integration has been achieved by creating an AWS profile config file and installing the `awsebcli` Elastic Beanstalk command line utility as one of the build steps. Both `client` and `server_node` containers are then tagged and pushed to the private Docker image registry provided by Amazon Elastic Container Service, before the `eb deploy prod-env` command actually triggers the production deployment. Both front and back end containers are then pulled from the private registry specified in the `Dockerrun.aws.json` file, while the mongo image is pulled directly from the official mongo Docker hub.

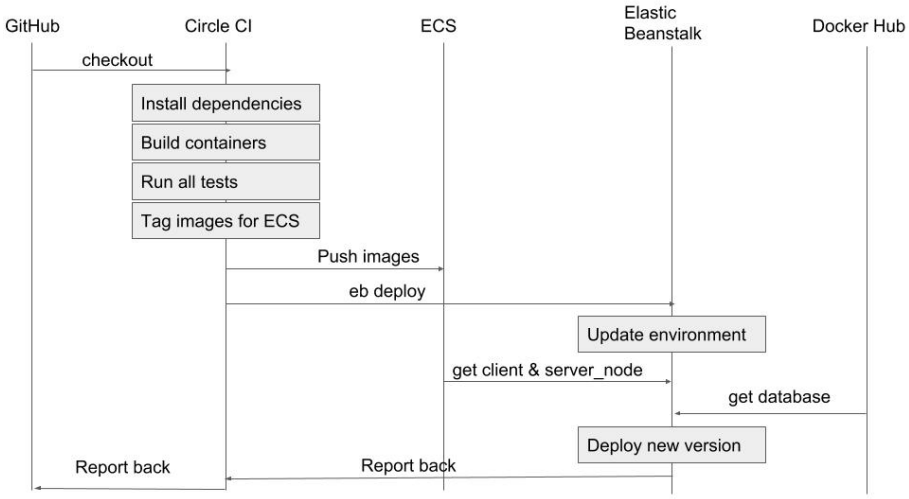


Figure 2.5: Production Deployment Visualised

### 2.1.5 Story Boards

Including the customer early in the development and design process allows teams to stay on track and readjust their design if necessary. The story boards below have been produced to gain feedback from the client, and ease the front end development in future iterations.

Figure 2.6 shows how the lectuer would login into the tool using Google Single Sign On. He could then upload his lecture slides using an action button in the bottom right corner of the screen, be able to edit his slides and mark certain slides as quizzes. A list of cards would be presented showing all lectures belonging to the lecturer and he could broadcast them to his audience. Subsequently, he could navigate through the slides and slides with embedded quizzes would split the screen in half to show a bar chart with answers as they come in. Finally, lecture would end once he clicks the end button.

Figure 2.7 on the other hand, shows how students could interact with the tool. Student could join an ongoing lecture using a session key provided by the lecturer. She could then participate in the lecture by watching the slides, and submitting answers to quizzes as they come in. The correct answer could be presented back to her in a form of a bar chart.

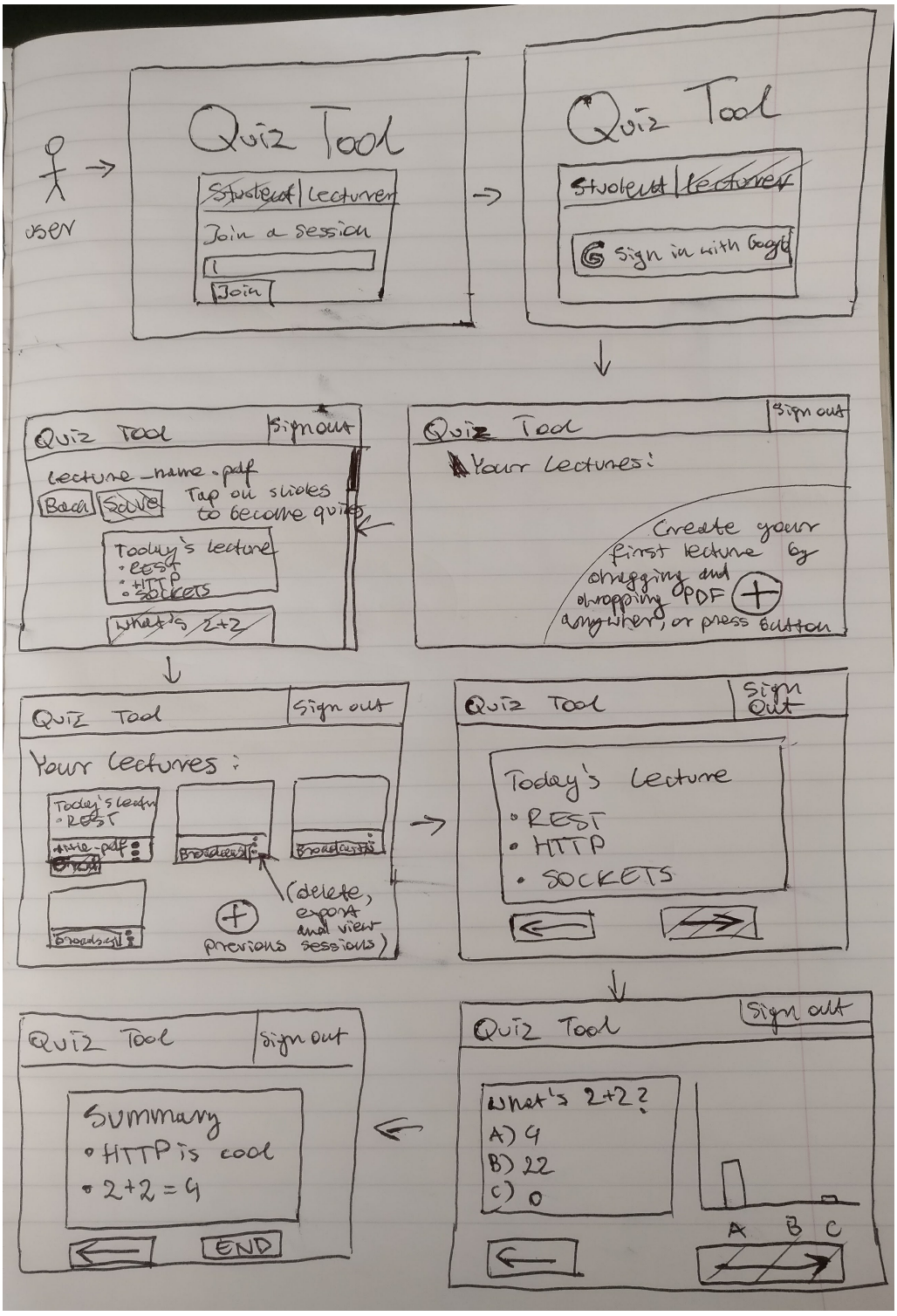


Figure 2.6: Story Board Lecturer Interaction

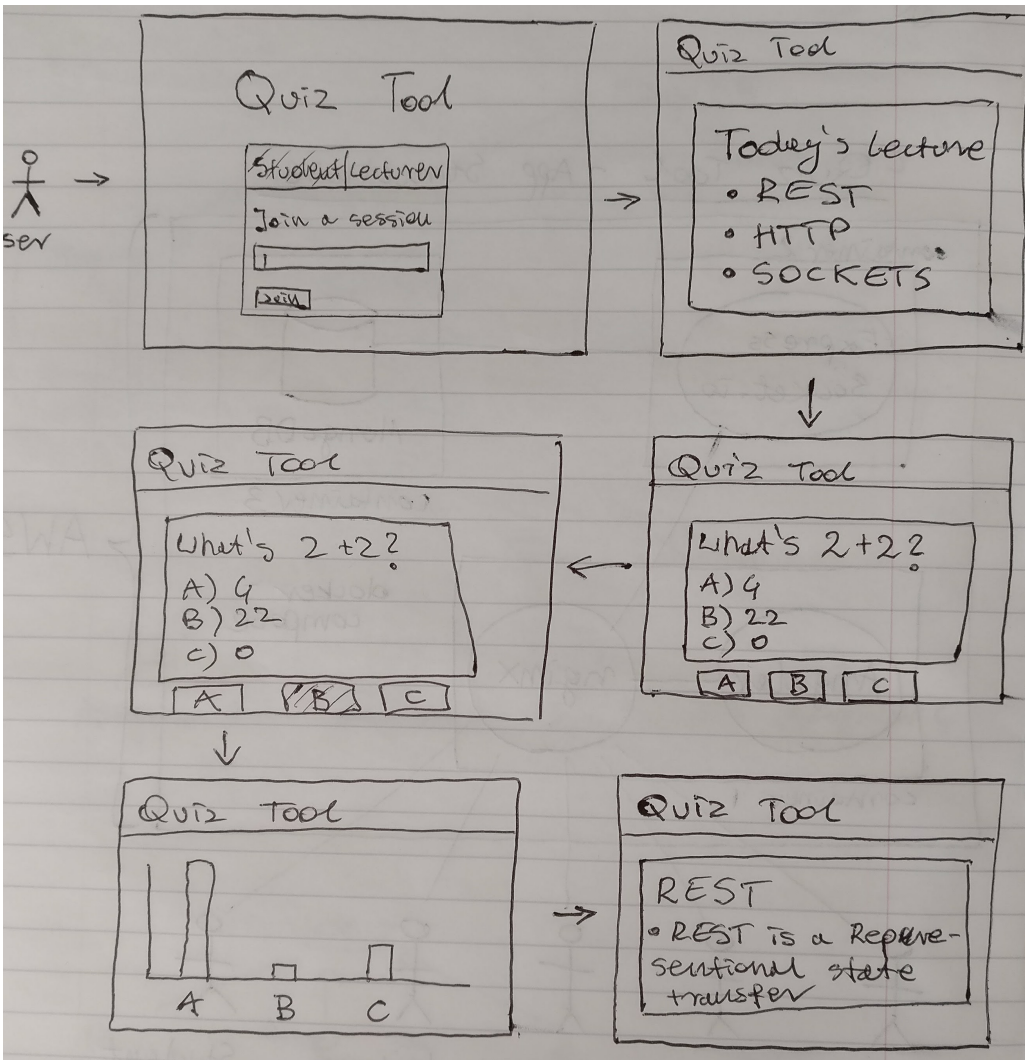


Figure 2.7: Story Board Student Interaction

### 2.1.6 Sprint Retrospective

The first sprint proved it was possible to deploy a MEAN stack, containerised application to production using Circle CI. Crucial DevOps has been successfully put in place, and the initial feedback has been gathered from the client thanks to the low fidelity prototypes. The full sprint retrospective document produced can be found in the Appendix D of this report.

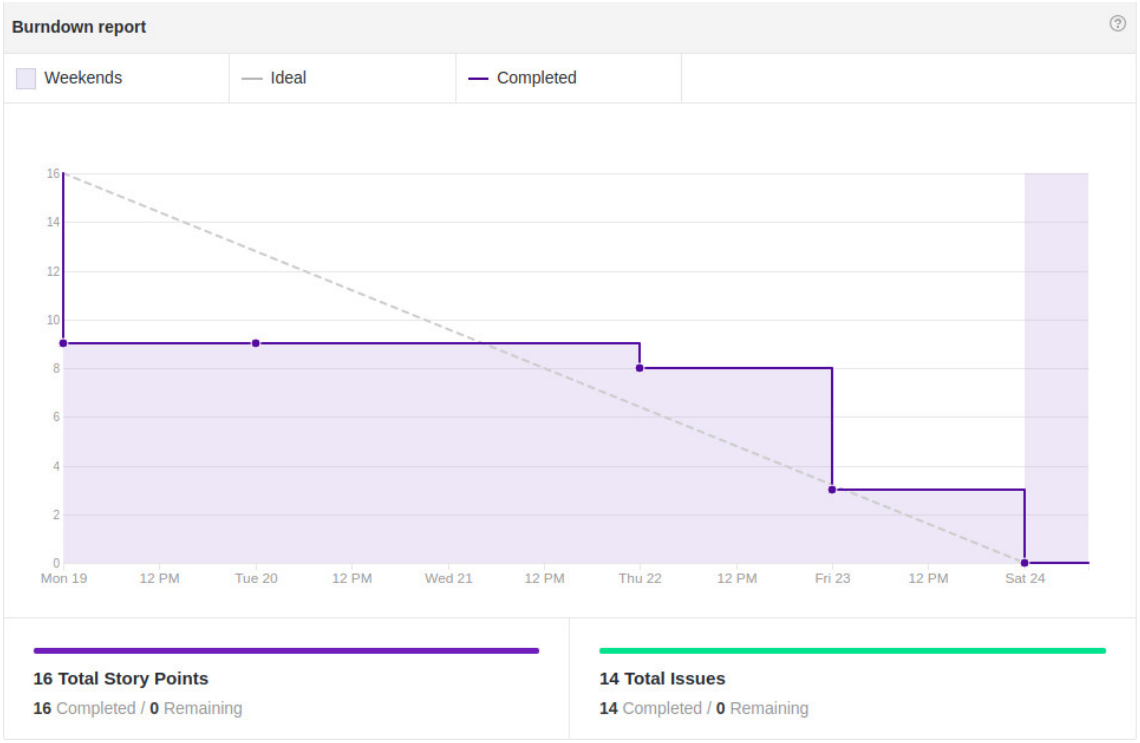


Figure 2.8: Burndown Chart Sprint 1

## 2.2 Sprint 2 - Bare Bone Application

### 2.2.1 Sprint Planning

This sprint focused on converting the proof of concept chat application created in the previous week into a basic Quiz Tool. Simple login page for both students and lectuers would be developed based on the paper prototypes from the previous sprint, and Google Single Sign On would be added. Once logged in, lectuers could upload their PDF lecture slides and broadcast them to all students, as session generation was not implemented yet. The entire list of estimated stories can be found in the Appendix C of this report.

### 2.2.2 Login Page and Authorisation

The login page has been created based on the low fidelity prototypes, and allowed lecturers to log into the tool, and students to join ongoing lectures with a session key. The Materialize[43] CSS framework has been added to achieve a modern user interface.

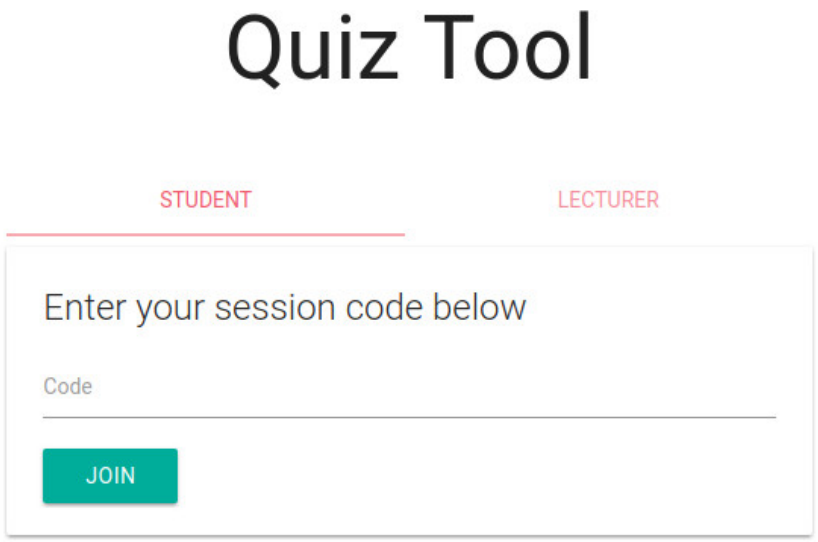


Figure 2.9: Student Login Page

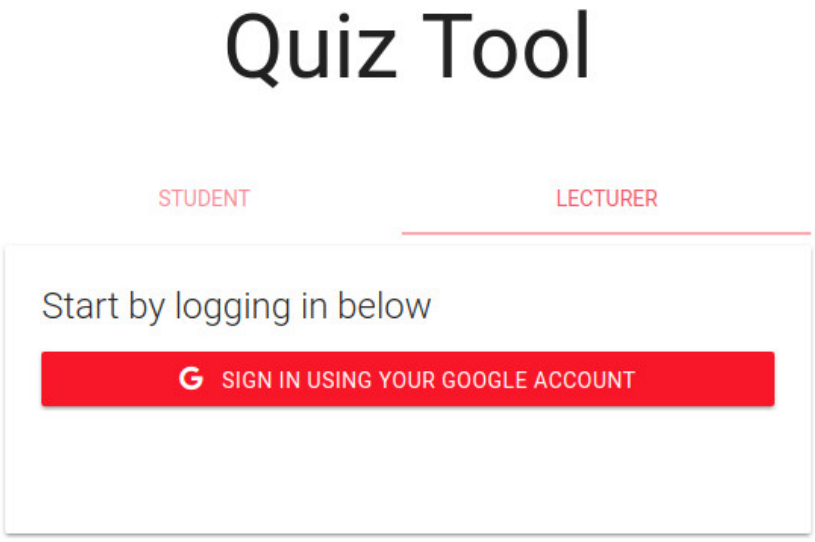


Figure 2.10: Lecturer Login Page

Angular has a concept of guards, which allow specified routes to be only accessible to authorised users. The code extracted from the `client/src/app/app.routing.ts` below shows how the desired behaviour has been achieved. All undefined routes redirect to the `/` which has a

`canActivate: [AuthGuard]` protection. The guard checks if the current user has a valid session cookie with a JWT token[44] present as described in the subsection below, before allowing the user to navigate to his dashboard.

```
const routes: Routes = [
  { path: '', component: DashboardComponent, canActivate: [AuthGuard] },
  { path: 'login', component: LoginComponent },
  { path: 'lecture/:code', component: LectureComponent },
  { path: '**', redirectTo: '' }
];
```

Figure 2.11: Angular Routing With Guards

#### 2.2.2.1 Google Single Sign On and JWT Tokens

Once the lecturer clicks on the Google Sign-In button, he is redirected to Google and asked to provide his Google credentials, and authorise the Quiz Tool to get basic information about him. The passport.js[45] Node.js middleware, together with the Google passport strategy[46] have been added to handle the OAuth2[47] authorisation. Once the lecturer authorises Quiz Tool to get the basic information about him, Google calls the `/auth/google/callback` callback defined in the `server/index.js`. The callback handler located in the `server/helpers/passport.helper.js` takes the access token, user's Google display name and his public Google ID, and stores them in the database by creating a new lecturer entry in MongoDB. Finally, a session cookie is sent back to the lecturer's client containing the access token.

As mentioned before, the `AuthGuard` on the Angular side makes sure user is logged in, before rendering his dashboard. The guard uses the `client/src/app/services/auth.service.ts` to make sure the session cookie exists. Each HTTP call Angular makes to the back end is intercepted by the `client/src/app/utils/jwt.interceptor.ts` which checks if user has the token, and if so, an extra authorisation header is added containing the token in the format `Authorization: Bearer fancyJSONToken123`. Finally, all routes on the back end are protected with the `server/helper/auth.helper.js`, which checks if the token provided exists in the database, is valid and has not expired, by checking with Google if it was issued against the Quiz Tool.

### 2.2.3 Persistence Layer

MongoDB is a document storage, enabling persistence of JSON-like documents with extra metadata including internal object ids. Mongoose[48] is a popular MongoDB object modelling library which has been integrated with the server side of the Quiz Tool. It removes the need to write boilerplate code and allows developers to focus on getting things done quickly. Schemas describing objects have been created and can be found in the `server/models` directory. They allow mongoose to validate data before it is allowed to be persisted in the database. Even though mongo is not a relational database, an entity relationship diagram depicted in the Figure 2.12 has been created to make the design more concrete.

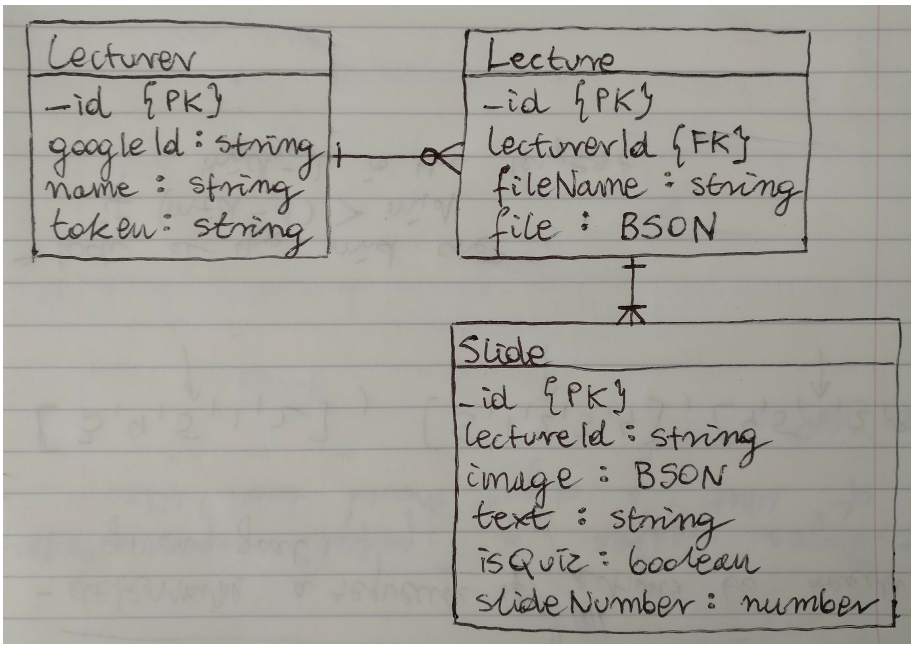


Figure 2.12: Initial Entity Relationship Diagram

### 2.2.4 Lecture Upload

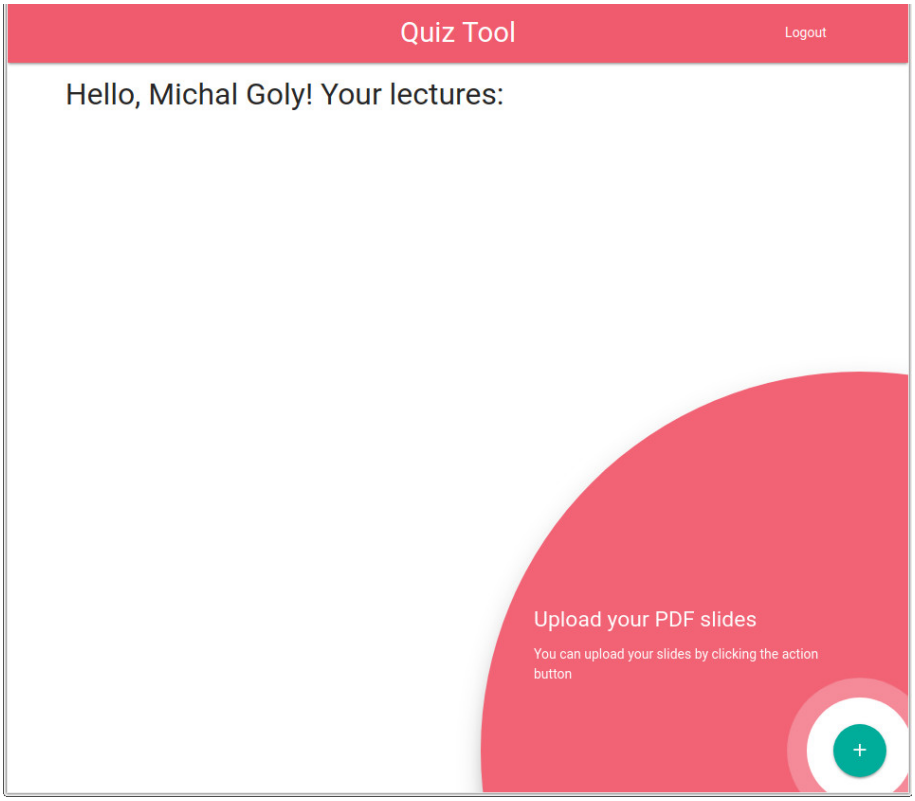


Figure 2.13: Dashboard View

Once lecturer logs into the system he is presented with a dashboard view visible in the Figure 2.13. The action button in the bottom right corner of the screen, allows PDF lecture slides to be uploaded. The tool does not work with other presentation file formats for simplicity reasons. This could however be extended in the future. Binary files can be stored in three major ways in MongoDB:

1. Using the Mongoose BSON (Binary JSON)[49] data type and setting it as type of a property of the document. This limits the file size to 16 MB.
2. Using GridFS[50] which relies on the client to have a driver capable of splitting the data into 255 KB chunks, and then streaming them into the database. This allows a binary file of any size to be persisted in MongoDB.
3. Storing a file on an external storage, and persisting the reference to the file in the database.

The BSON approach has been chosen for the Quiz Tool, as it significantly simplified the implementation and it is unlikely PDF presentations over 16 MB would be commonly used. Again, this is something that could be changed in the future if the lifespan of the tool extended beyond the final project submission.

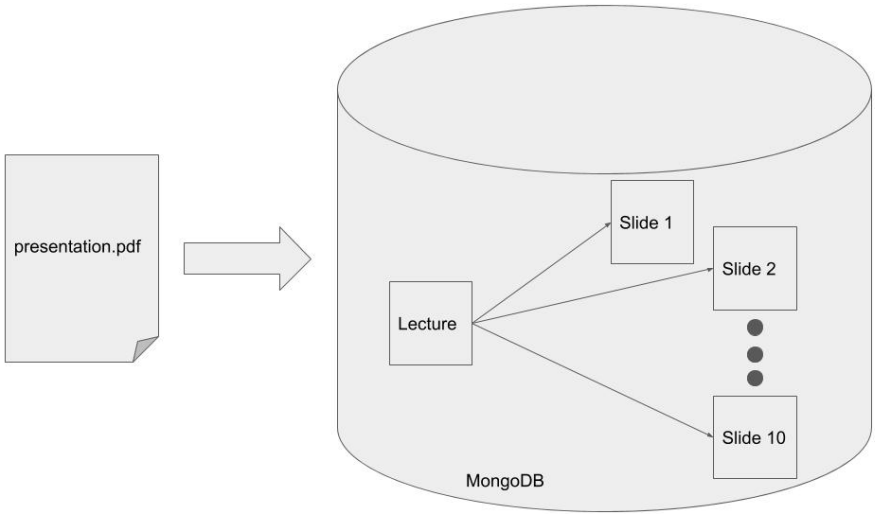


Figure 2.14: Initial File Upload

The initial implementation of the file upload, took the PDF file, stored it directly as BSON in the `Lecture` schema object and split the lecture into slides storing them in MongoDB using the `Slide` schema object. Once the file has been uploaded using the ng2-file-upload[51], and persisted in the database, it has been stored temporarly on disk. Text would be extracted from each slide using the pdf-extract[52] library, which would be instructed to leave single paged PDFs for each slide on disk. These PDFs would be then converted to PNG images using the scissors[53] library. Finally, slides' text and images would be grouped together and stored in MongoDB, before the temporary files would be cleaned from disk.

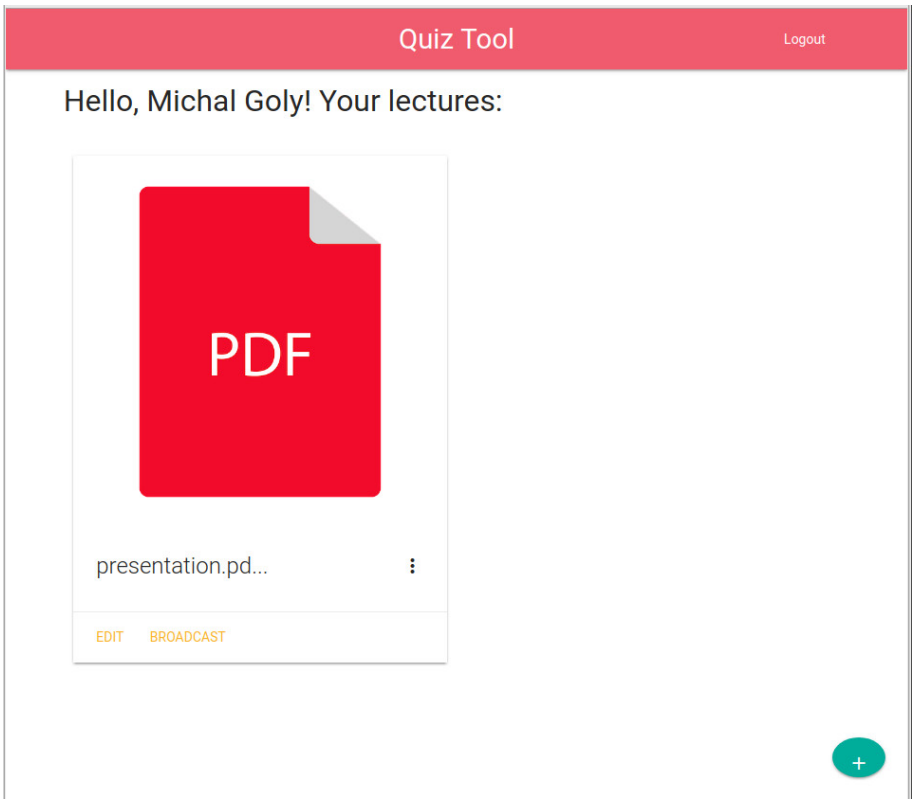


Figure 2.15: Lecture Uploaded View

### 2.2.5 Lecture Broadcast

Lectures could be broadcasted using the *BROADCAST* action button located on the bottom of each lecture card in the dashboard. Figure 2.16 depicts the lecturer's view, once the lecture starts. The initial implementation added in this sprint, did not have the concept of a session yet, therefore slides were sent to all students who used any session key to log into the tool.

Once the lecture was started, lecturer's client would pull all slides from the back end and emit the very first one to all the students. Socket.io would be used to emit a simple event *slide-change* containing a JSON message in the following format: {`"img": "base64img"`}. PNG representations of each slide were stored in the BSON format internally, but they were sent to clients using the Base64 binary-to-text format[54]. Students' clients would be then easily able to render such images using HTML `<img>` tags. Finally, once the lecture was over, a `null` image would be sent to everyone and students' clients would handle it by ending the lecture. The flow has been depicted in the Figure 2.17.

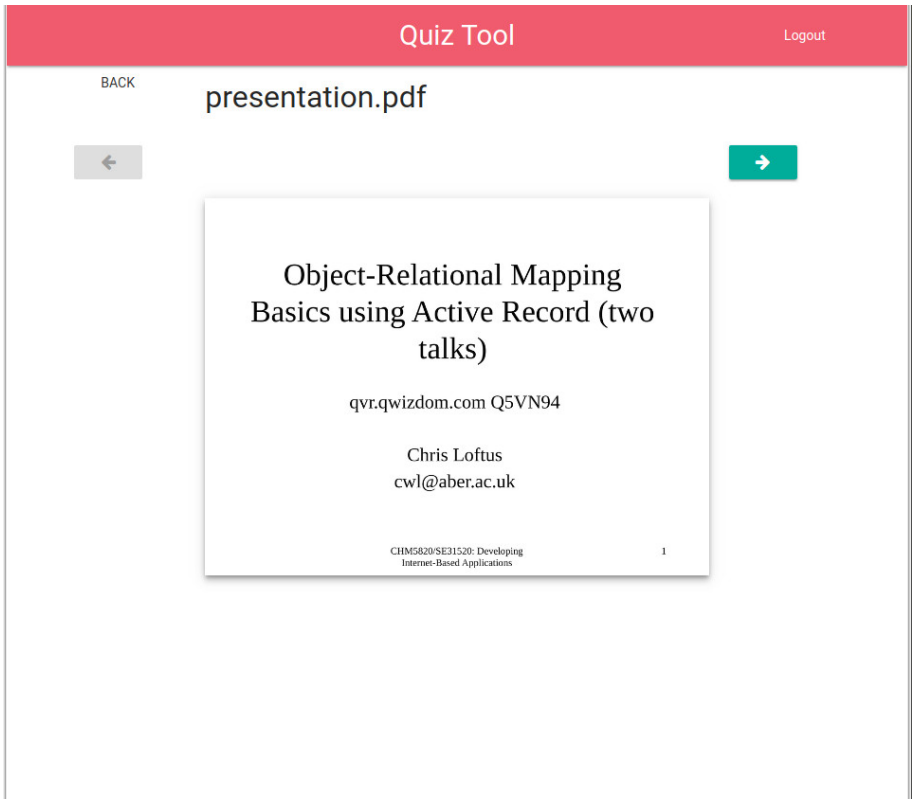


Figure 2.16: Initial Broadcast View

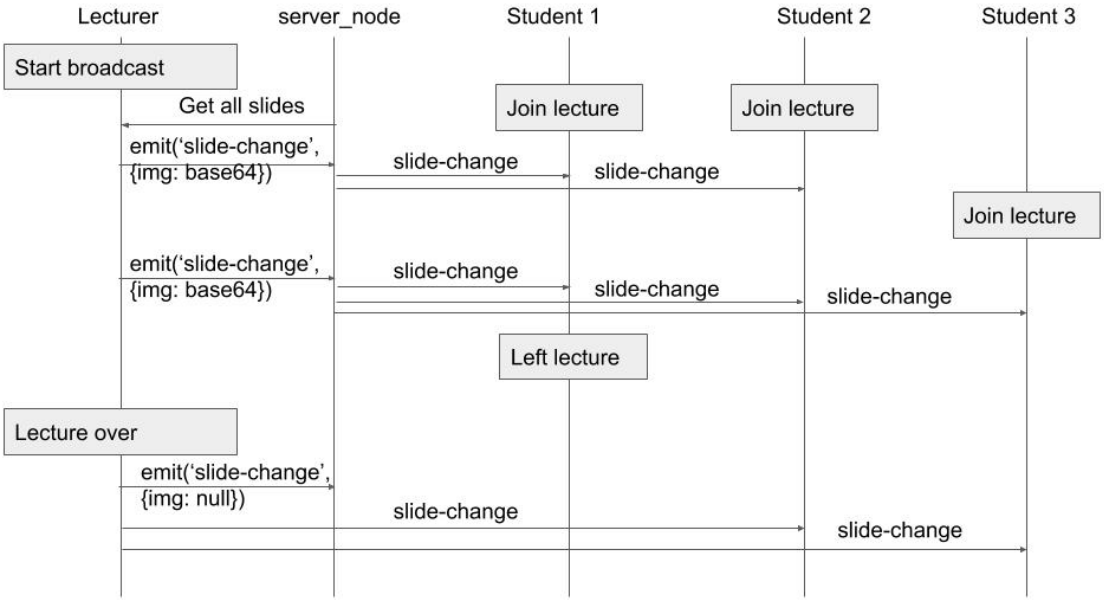


Figure 2.17: Initial Slide Broadcast Flow

### 2.2.6 Sprint Retrospective

This was a very important sprint as it shaped the overall structure of the tool. The basis of the Quiz Tool have been designed and implemented on both the front and the back end. Schemas have been introduced to handle persistence, file uploads were implemented and lecture slides could be emitted to all students using sockets in real time. The sprint took two weeks, instead of one, as it was overfilled with too many issues. The full sprint retrospective document produced can be found in the Appendix D of this report.

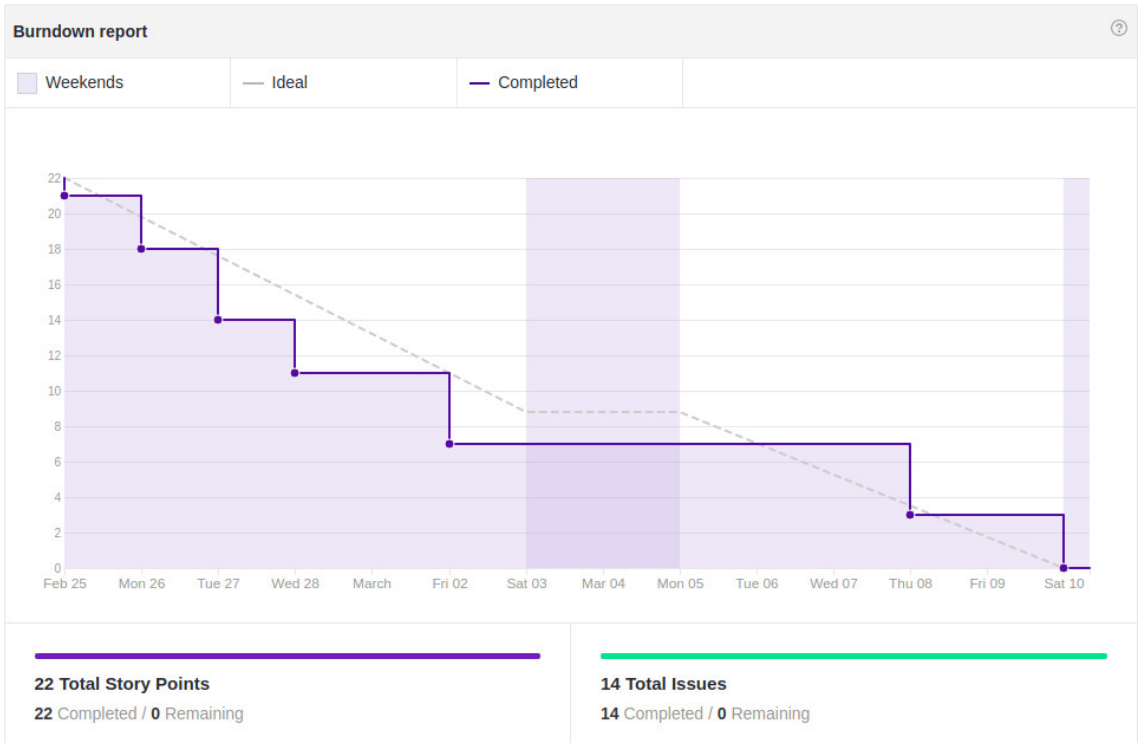


Figure 2.18: Burndown Chart Sprint 2

## 2.3 Sprint 3 - Add Quizes

### 2.3.1 Sprint Planning

This iteration focused on the ability to create quizzes, embed them into slides and broadcast only to people who had the appropriate session key. The entire list of estimated stories can be found in the Appendix C of this report.

### 2.3.2 Session Keys

Every time lecturer started a new broadcast, a random session code would be generated. It would consist of eight alphanumerical characters, and the random generator would be seeded with the current time, to decrease the probability of generating two idential keys for two sessions running

at the same time. The *slide-change* Socket.io event described in the previous sprint, has been extended to carry the appropriate session key with each broadcast.

```
{
  "img": "base64encodedImage",
  "sessionCode": "ABCD1234",
  "isQuiz": "false"
}
```

Figure 2.19: The JSON Message Broadcasted on Slide Change

Students' clients are not uniquely identified by the Quiz Tool, therefore slides would be broadcasted to all clients participating in the broadcast. Each client would then know whether they have to update their `<img>` tag, by checking if the session key entered by the student matches the key bundled with the image slide arrived.

### 2.3.3 Embedding Quizzes

Each `Slide` in the system consisted of a PNG image and text extracted from the PDF presentation uploaded by a lecturer. It also had a boolean flag `isQuiz` specifying if given slide was supposed to be rendered as a quiz or not. An edit panel visible below has been added, to allow lecturers to specify which slides should be treated as quizzes.

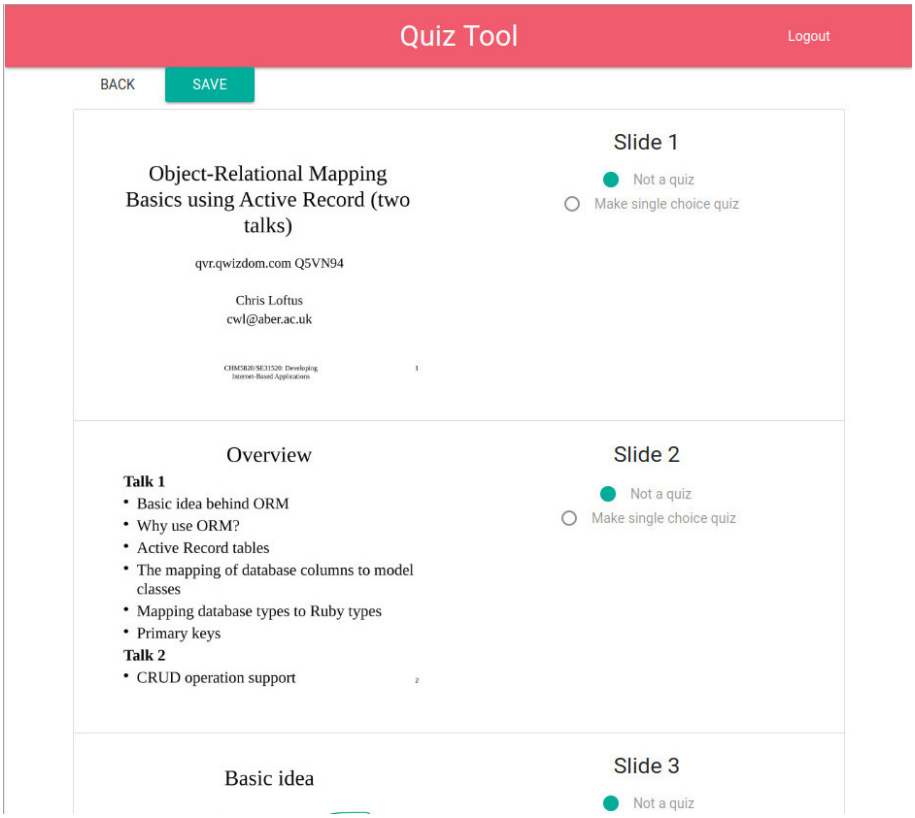


Figure 2.20: Initial Edit Page

Figure 2.20 shows two radio buttons for each slide of a lecture. These radio buttons could be used to mark certain slides as quizzes, by toggling the boolean flag underneath. Text extracted from each slide would decide if a given slide could become a quiz. If slide was not eligible to become a quiz, the radio button to make it one would be greyed out. The initial implementation of the eligibility algorithm checked if given slide contained at least two strings ”A)” and ”B)”. This was enough to make a slide eligible to be marked as a single choice quiz, and to be rendered as one, as described in the next subsection.

### 2.3.4 Quiz Rendering

Figure 2.21 shows the split screen view for a lecturer when a slide marked as a quiz was rendered during a lecture broadcast. The left side shows the currently broadcasted slide, whereas the right side shows a bar chart capable of displaying students' answers as they come in. Lecturer can click the *RETRY* button to wipe students' answers for current question and ask again. Once he is happy with the results, he can select the correct answer which will be shown to all people in the session, and then render the following slide, or even the previous one if necessary. Students' answers were only stored in memory at this point for the duration of the session.

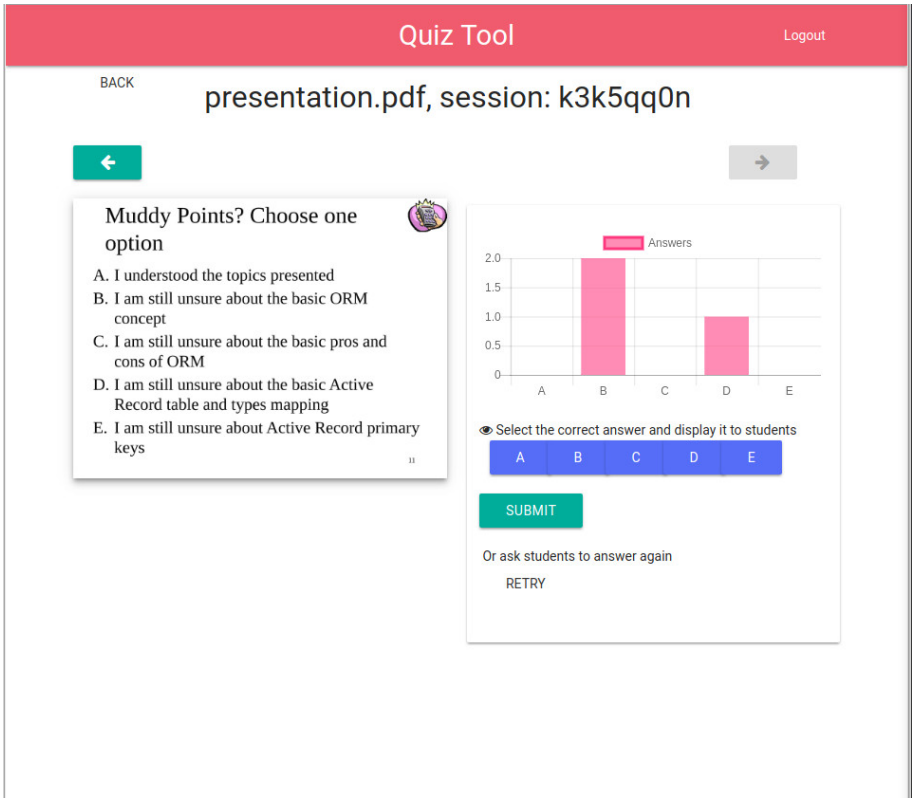


Figure 2.21: Quiz Rendering Lecturer

Figure 2.22 on the other hand, shows the view rendered on students' clients. They would be able to select one of the options and submit their answers only once. The ”A”, ”B”, ”C”, ”D”, ”E” buttons were generated automatically based on the text of the slide. Students submitting their answers would emit an *answer-received* Socket.io event, bundled with a message in the format `{"sessionCode": "ABCD1234", "option": "B"}`, which would be then received

by lecturer's client and rendered appropriately.

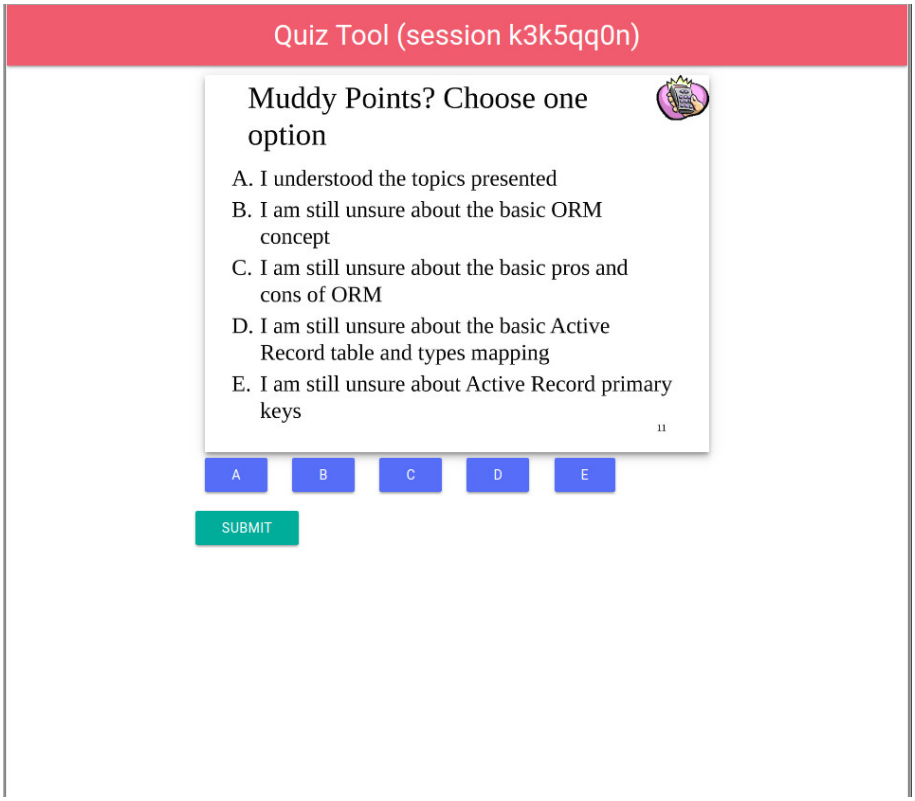


Figure 2.22: Quiz Rendering Student

### 2.3.5 Sprint Retrospective

Following this iteration, Quiz Tool was able to handle quiz embedding which was a major step towards the final implementation. Slides were no longer sent to everyone, and multiple lectures could run at the same time. The full sprint retrospective document produced can be found in the Appendix D of this report.

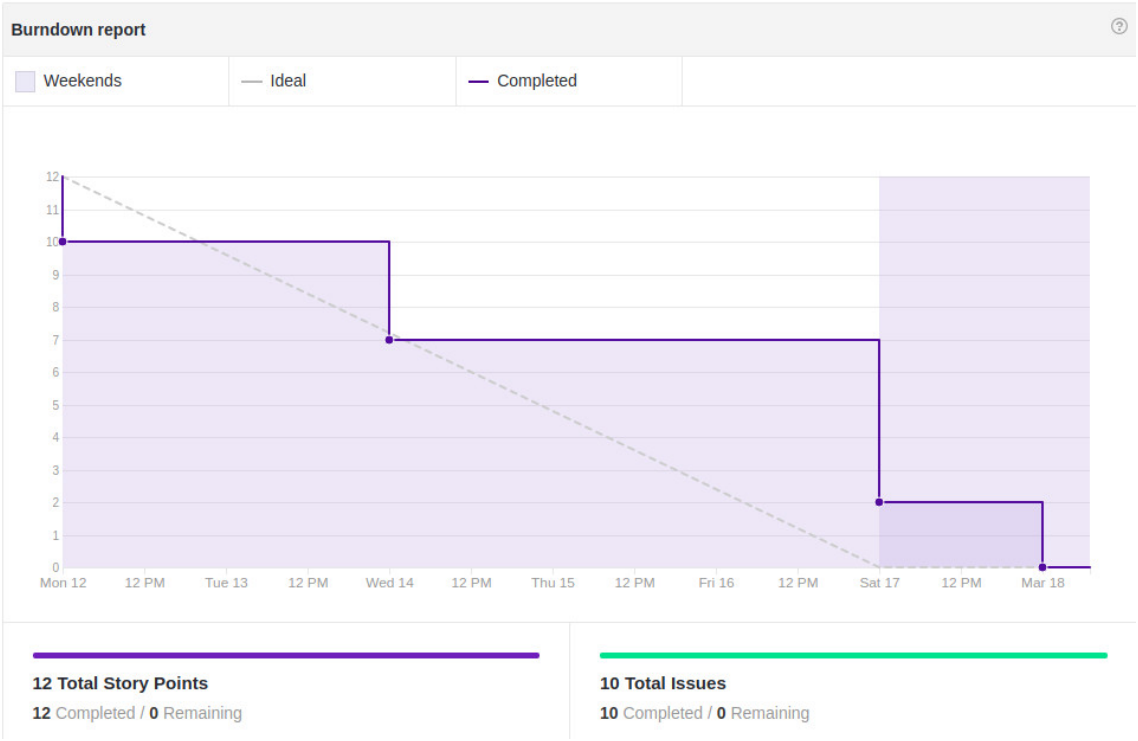


Figure 2.23: Burndown Chart Sprint 3

## 2.4 Sprint 4 - Fancy Quizes and Defect Fixing

### 2.4.1 Sprint Planning

This sprint focused on addressing a critial bug in the production environment concerning the PDF to PNG conversion. Extra types of quizzes would also be added to complement the single choice ones available at the moment. The entire list of estimated stories can be found in the Appendix C of this report.

### 2.4.2 Production Environment Bug

The presentation file upload and splitting of slides into PNG images persisted together with text implemented in sprint 2, worked as expected in both development and testing environments. The production environment on the other hand, would randomly crash and the underlying cause had to be investigated before any other work could progress. The process of splitting the PDF presentations uploaded relied on splitting the intial file into separate, single paged PDF files, extracting their text, converting them to PNG images, before grouping together and persisting in the database. The final part of asynchronously trying to convert single paged PDF files into PNG files, failed with an IO exception, where the underlying PDFtk[55] command line utility could not find a PDF file, even though it was there.

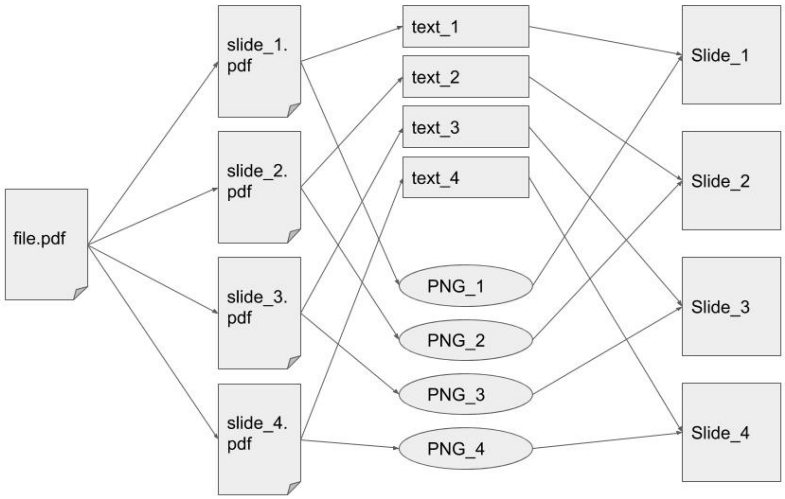


Figure 2.24: Initial Slide Extraction

My initial thought was that the problem was due to a race condition, where asynchronous code was trying to get access to the file system. This would mean the code worked as expected on my much more powerful laptop only by luck. Making the algorithm to process files synchronously, did fix the problem for smaller PDF presentations, however attempting to upload a larger test PDF crashed the production environment in the same manner. Finally, the algorithm has been changed to no longer use single paged PDFs, and convert them one by one to PNGs. The scissors library has been removed, and the node-pdf2img[56] used instead to take the whole PDF presentation, and extract all the PNG files from it using a single PDFtk call. The algorithm could stay asynchronous, and the file system access has been almost halved.

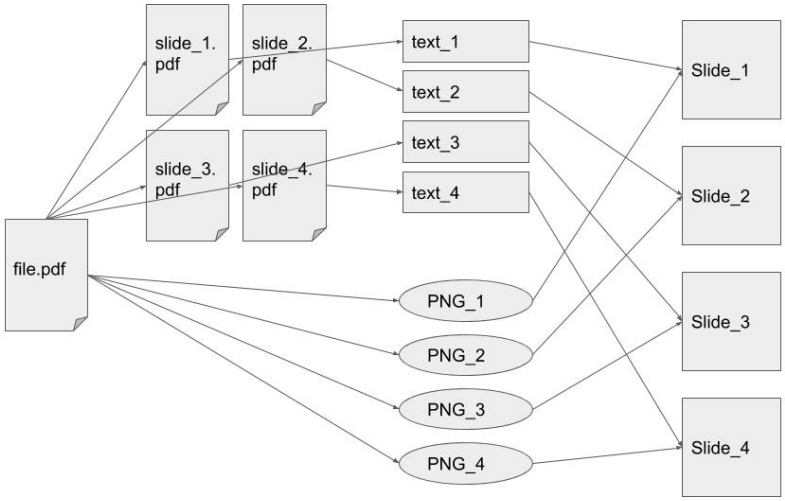


Figure 2.25: Final Slide Extraction

### 2.4.3 Extra Types of Quizzes

The `isQuiz` boolean flag was no longer appropriate if extra types of quizzes were to be supported. Every slide could be made into a true/false quiz, and quizzes eligible to become single choice, could also be marked as multi choice. Changing the slide eligibility algorithm was the first step towards having support for all three quizzes. The initial version checked if slide's text contained at least "A)" and "B)". The improved version would make a slide eligible to become a quiz if one of the following was true:

- "A)" and "B)" was found in the slide's text
- "A." and "B." was found in the slide's text
- At least two bullet characters were found

The `isQuiz` flag of the `Slide` schema object has been changed to a `quizType` of type `string`. It could take the value of `null`, `"truefalse"`, `"single"` and `"multi"`. Furthermore, the JSON message bundled with the `slide-change` Socket.io event was changed to:

```
{
  "img": "base64encodedImage",
  "text": "textOfTheSlide"
  "quizType": "truefalse"
  "sessionCode": "ABCD1234"
}
```

Figure 2.26: The Final JSON Message Broadcasted on Slide Change

This enabled the student's client to know how to render the quiz appropriately. For example if the current slide was marked as a multi choice, the client would use the text of the slide, extract the answers buttons e.g. "A", "B" and "C", and let student submit their answer. The answer could no longer be a single option. It was therefore changed to an array of options.

```
{
  "sessionCode": "ABCD1234",
  "options": ["A", "C"]
}
```

Figure 2.27: The Final JSON Message Emitted when Student Submitted his Answer

### 2.4.4 Sprint Retrospective

Following this sprint, the Quiz Tool was almost complete. The persistence and report generation was still missing, but the core functionality including the ability to add various quizzes into slides, and broadcast them to students was finished. The full sprint retrospective document produced can be found in the Appendix D of this report.

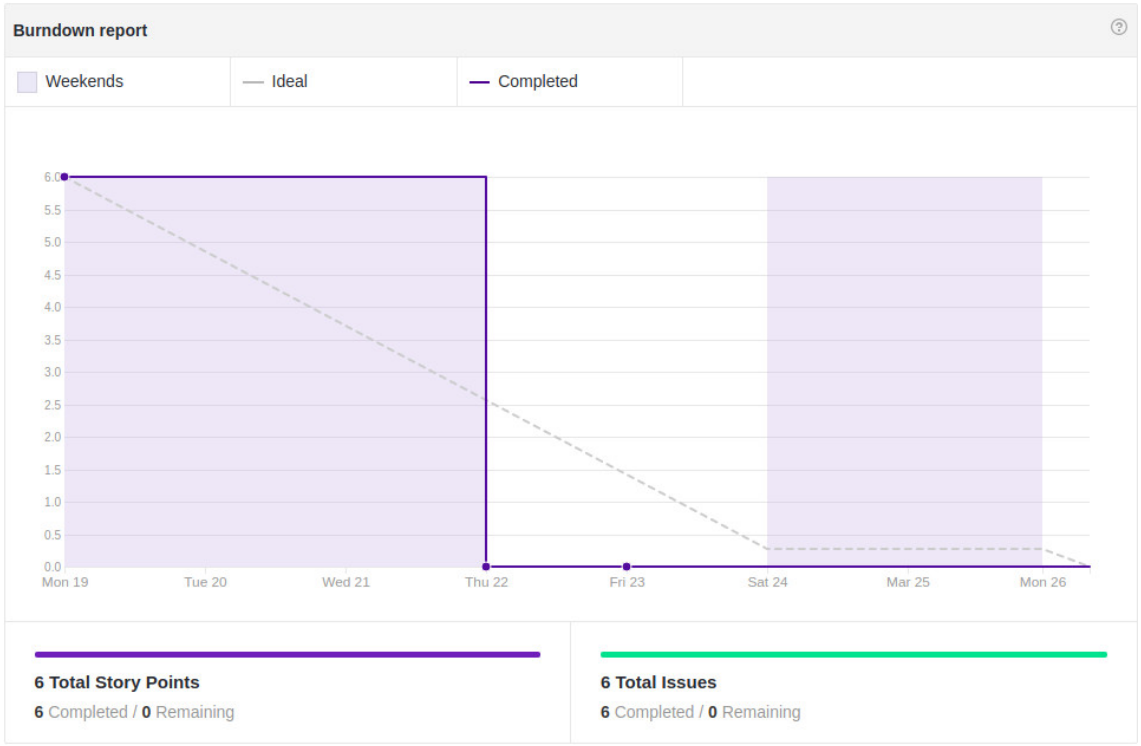


Figure 2.28: Burndown Chart Sprint 4

## 2.5 Sprint 5 - Persistence, Report Generation and Testing

### 2.5.1 Sprint Planning

The final sprint focused on the persistence of students' answers in the database, and generation of PDF reports based on these answers. User friendly error handler has also been added, and the application has been polished before the Quiz Tool implementation was over. As always, the entire list of estimated stories can be found in the Appendix C of this report.

### 2.5.2 Database Container Persistence

Docker containers do not persist data by design. Every time the `database` container has been brought up and down, by both docker-compose and the Milticontainer Elastic Beanstalk environment, any data stored in the MongoDB would be lost. This behaviour is perfect for running tests, where the starting environment is expected to be the same between tests. It was however unacceptable for the production environment.

The `Dockerrun.aws.json` file included in the Appendix E of this report as the Figure E.6, shows the final configuration of the production environment running on AWS. The volume mount called "mongo" has been added, which maps the `/data/db` directory in the `database` container onto the `/var/app/database` directory on the underlying EC2 instance running the container cluster. This causes the database to persist its data even when the application is redeployed. The only time the data could be lost, is when the EC2 instance is destroyed, when for example more RAM has been requested for the environment.

### 2.5.3 Lecture Answers Persistence

The final top level requirement **FR-6** identified by the client before the development had started, was the ability to export lecture sessions results in some format for future analysis. Regardless of the final format, session answers given by students during lectures had to be persisted in some fashion.

Every time a new lecture broadcast started, an empty `liveAnswers` object has been instantiated in memory on the lecturer's client. This object would be responsible for the storage of answers for the given slide being broadcasted at the moment. It was therefore reinstantiated to {} every time lecturer emitted a new slide. Once students' answers started to come in, the object would be treated like a dictionary. Figure 2.29 depicts a possible state of the `liveAnswers` object once a few answers came in.

```
{
  "A": 3,
  "C": 1,
  "D": 19,
  "correct": [
    "D"
  ]
}
```

Figure 2.29: The liveAnswers Object

The `correct` answers array would be added to the object, if the lecturer decided to display the correct answer to all the students. `liveAnswers` objects for each slide containing a quiz would be added to a hash map of type `Map<string, Object>`, where the key would be the `_id` of a given `Slide` schema object. A potential final state of the `answers` hash map is visualised in the Figure 2.30.

```
answers: {
  slideId123: {
      "A": 3,
      "C": 1,
      "D": 19,
      "correct": [
        "D"
      ]
    },
  slideId42424: {
    "true": 20,
    "false": 1
  }
}
```

Figure 2.30: The liveAnswers Object

Finally, a new schema object `Session` has been added to the database models, to enable persistence of answers in the database. Figure 2.31 shows the updated ERD of the system.

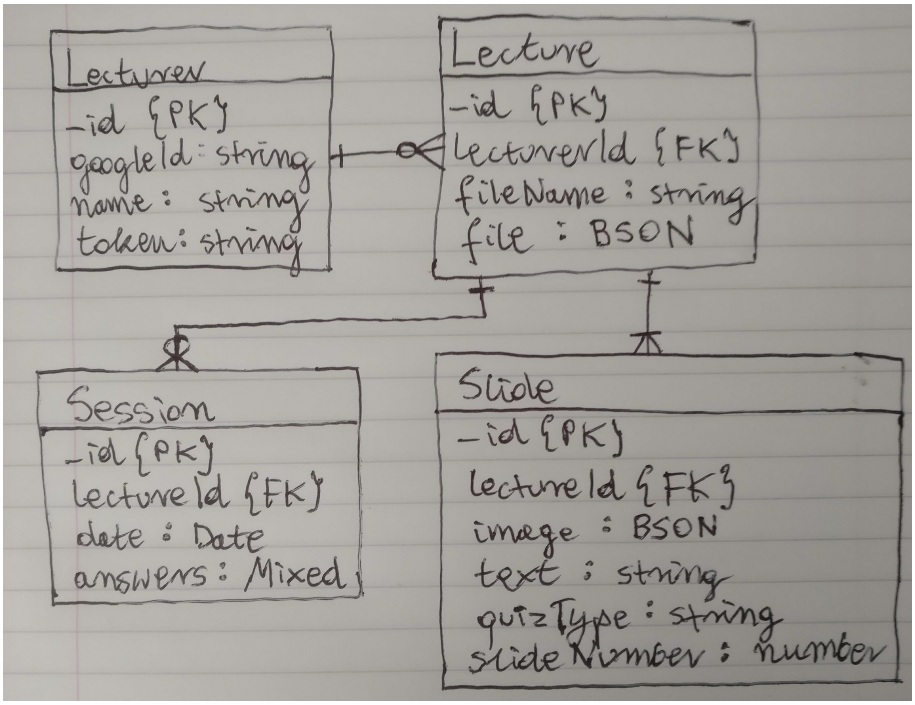


Figure 2.31: Final Entity Relationship Diagram

### 2.5.4 Report Generation

As mentioned in the subsection above, having the ability to export students' answers for future analysis was one of the initial functional requirements. Export of the information in JSON format was considered for a while, before realising PDF reports would make more sense, as this data was ultimately supposed to be consumed by humans. An extra view shown in Figure 2.32 has been added to allow PDF report generation. Lecturers would be presented with a list of all sessions they run for the particular lecture in the system. Finally, the jsPDF[57] library has been added, which allowed lectuers to export PDF reports for each session. An example PDF generated by the tool can be found in Appendix F of this document.

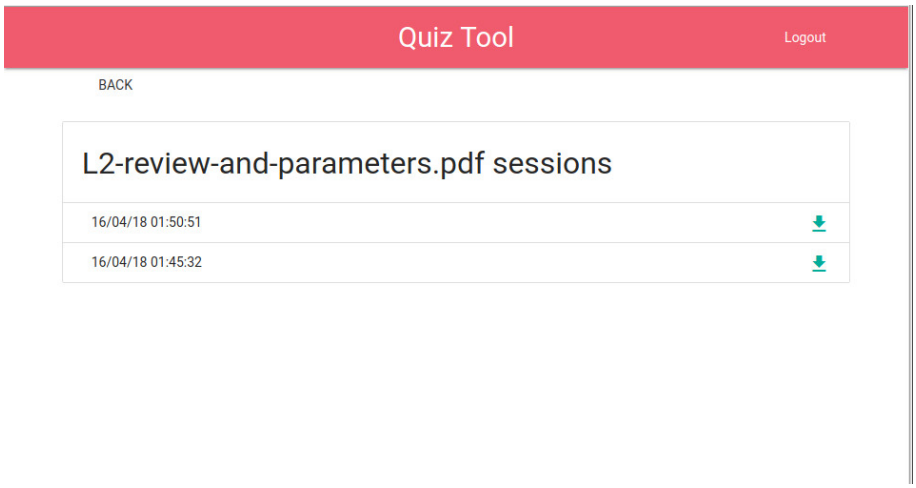


Figure 2.32: Session View

### 2.5.5 Sprint Retrospective

The final sprint brought the implementation of final requirements of the Quiz Tool. Persistence has been addressed, the ability to generate PDF reports added, and the overall look and feel of the tool has been polished. The sprint took two weeks, as opposed to one, as some issues proved more challenging than initially anticipated. The full sprint retrospective document produced can be found in the Appendix D of this report.

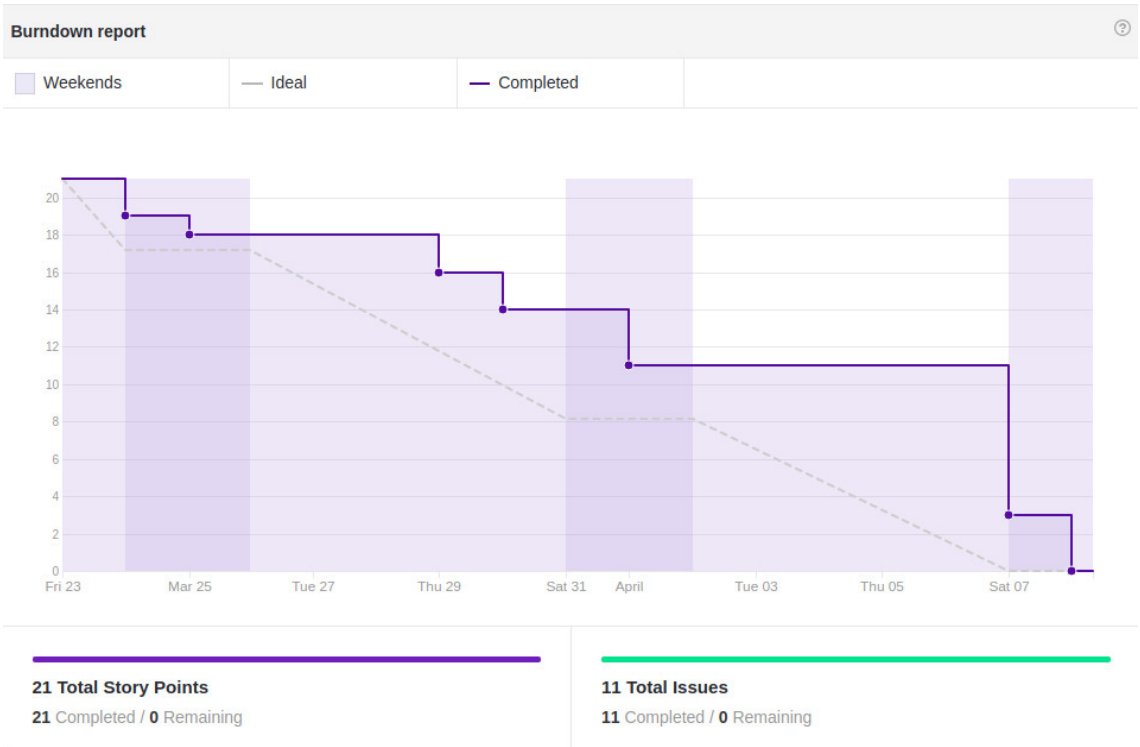


Figure 2.33: Burndown Chart Sprint 5

# Chapter 3

# Final Design

This chapter follows on the discussion of the individual sprints and summarises the final design of the tool.

## 3.1 Application Architecture

### 3.1.1 Technology Overview

Quiz Tool is a MEAN stack application developed using JavaScript based technologies. **M**ongoDB has been used as the persistence layer, **E**xpress.js has been used to develop the back end of the application, **A**ngular 4 to develop the front end, and the back end is running on a **N**ode.js runtime. Quiz Tool consists of three subsystems, each containerised using Docker. Docker image definitions of both the back and the front end components have been defined using Dockerfiles, the database container on the other hand is built using the official MongoDB image from the Docker Hub registry. Quiz Tool is a multi-container Docker application, and containers are configured to run together using docker-compose both in the development and testing environments. The Multi-container Elastic Beanstalk orchestrates containers to run together in the production environment hosted on AWS.

### 3.1.2 Application Structure

Figure 3.1 shows the overall structure of the multi-container tool. The structure is formally defined in both the `docker-compose.yml` and the `Dockerrun.aws.json` files located in the root directory of the project. The front end container `client` consists of an nginx reverse proxy routing the traffic of the application, and the front end compiled using Angular. The back end container `server_node` contains the Express.js server side logic, accompanied with the Socket.io engine which uses the the WebSocket protocol to enable the application to work in real time. Finally, the `database` container contains the MongoDB database.

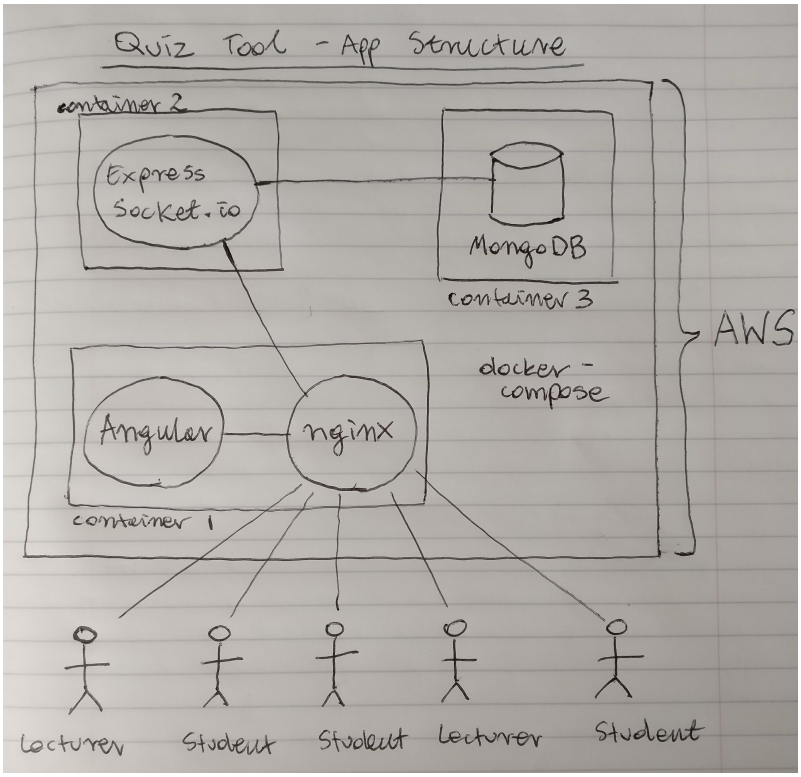


Figure 3.1: Final Application Structure

## 3.2 Back End Container

### 3.2.1 Docker Container

The `server/` directory contains the back end code of the tool. The `Dockerfile` included in the Appendix E of this report as the Figure E.7, describes the steps performed to build the `server_node` container. The image is based on the official `node:carbon` Docker image, dependencies are installed and the container is started with `npm start`, after the port 3000 has been exposed.

### 3.2.2 Back End Overview

Express.js is an unopinionated web development framework, meaning the structure of the application has to be decided by the developer. The `index.js` file is the starting point of the application. The summary of the routes handled by four controllers located in the `server/controllers/` directory is listed in Table 3.1.

| Request Method | Route | Purpose |
|---|---|---|
| GET | `/lecturers/logged-in` | Returns the currently logged in lecturer |
| GET | `/lectures` | Returns a list of lectures of current lecturer |
| GET | `/lectures/:_id` | Returns the lecture by id |
| GET | `/lectures/:_id/file` | Returns original presentation file |
| DELETE | `/lectures/:_id` | Removes lecture by id |
| POST | `/lectures` | Uploads a new lecture |
| GET | `/slides/:lecture_id` | Returns slides for lecture by its id |
| PUT | `/slides` | Updates slides in bulk |
| GET | `/sessions/:_id` | Returns sessions for lecture by its id |
| POST | `/sessions` | Creates a new session |

Table 3.1: Back End Routes

Directory structure of the tool is self-explanatory:

- `db/` contains database access code
- `helpers/` contains utility functions concering authentication and the Socket.io handler
- `models/` contains mongoose schema objects describing valid models which can be persisted in the database.
- `schemas/` contains JSON validator schemas
- `test/` contains server side unit tests

### 3.2.3 Authentication and Validation Middleware

Lecturers can login into the tool using their Google credentials. The authentication flow has been described in the subsubsection 2.2.2.1 of this report. Each route listed in Table 3.1 is protected by the authentication middleware defined in the `helpers/auth.helper.js` file. At a very high level, the middleware checks if the HTTP request contains an authorisation header containing a Google access token unique to each lecturer in the system, before autenticating and authorising them to access a given resource. Please refer to the implementation for more details.

Furthermore, routes **PUT** `/slides` and **POST** `/sessions` expect a JSON doument to be submitted with each request. The validity of the JSON provided is checked using the JSON schema validation provided by the jsonschema[58] JavaScript module.

### 3.2.4 Socket.io Engine

The Socket.io engine running in the `server_node` container, is used to allow slides to be broadcasted in real time to students participating in a lecture. Students can also submit their answers using the Socket.io client running on their machines, and lecturer can emit events indicating slide changes during a presentation. Lecture broadcast has been described in the subsection 2.2.5 of this report, and table Table 3.2 lists Socket.io events managed by the tool.

| Event | Purpose |
|---|---|
| `slide-update` | Emitted by a lecturer when lecture slide is changed |
| `answer-sent` | Emitted by a student when quiz answer submitted |
| `correct-answer` | Emitted by a lecturer when correct answer shown |
| `current-slide-request` | Emitted by a student to request current slide on joining a lecture |

Table 3.2: Socket.io Events Summary

## 3.3 Front End Container

### 3.3.1 Docker Container

The `client/` directory contains the front end code of the tool. The `Dockerfile` included in the Appendix E of this report as the Figure E.1, has not changed since the first sprint. The container steps have been explained in the subsubsection 2.1.2.1 of this report.

### 3.3.2 Front End Overview

#### 3.3.2.1 Nginx Reverse Proxy

The nginx reverse proxy routes traffic within the application. The `client/nginx/default.conf` file contains the web server's configuration. There are two notable aspects of the file, included in the Appendix E of this report as Figure E.8.

1. All HTTP traffic going through the `/express` route is navigated to the `server_node` container
2. All Socket.io traffic is also rerouted to the `server_node` container

#### 3.3.2.2 Angular Code

Angular 4 has been used to develop the front end. It focuses on development of reusable components, consisting of enchanced HTML and logic written in TypeScript. Code ready for production is compiled down to HTML, CSS and JavaScript, which can be then served by the nginx web server to all the clients. Components can be embedded in each other, and the example could be the `NavbarComponent` which has been used across the application.

The structure of the `client/src/app` directory is straightforward to follow:

- `components/` contains all the components in the application
- `guards/` contains the authentication guard preventing unauthenticated people from accessing certain components, as discussed in the subsection 2.2.2 of this report
- `models/` contains the models which mimic the mongoose schema objects on the server side
- `services/` contains services which consume endpoints exposed by the `server_node`, and offline services containing utility methods
- `utils/` contains the JWT interceptor discussed in the subsection 2.2.2 of this report

## 3.4 Database Container

The database container is pulled directly from the official MongoDB Docker Hub registry. The `database` container uses a volume mount in production to persist the data between deployments.

### 3.4.1 Entity Relationship Diagram

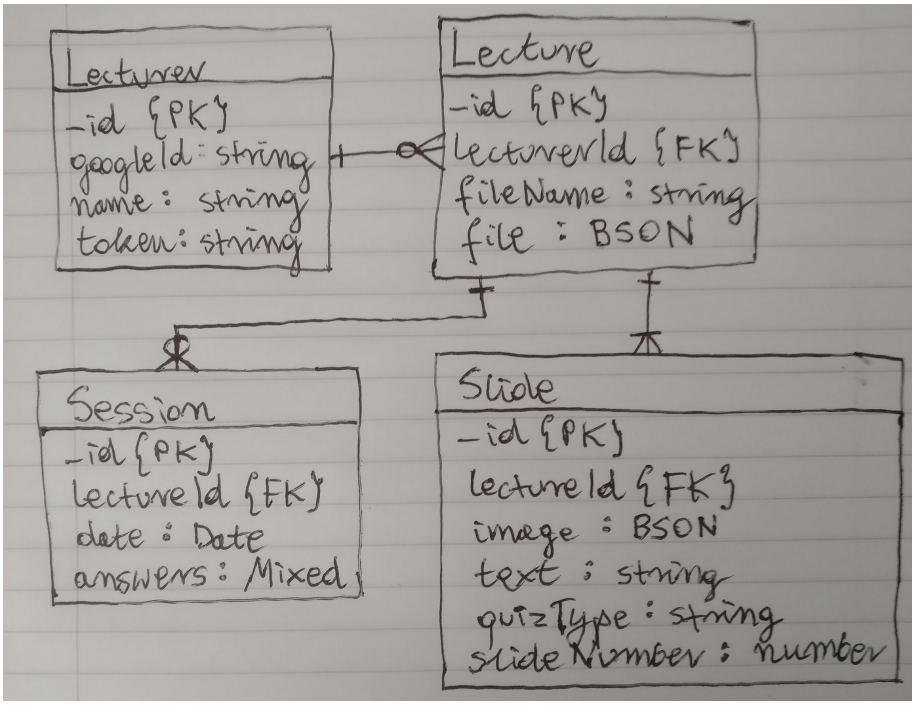


Figure 3.2: Final Entity Relationship Diagram

## 3.5 Environments

Development and testing environments have been described in the subsection 2.1.2 of this report. Production environment on the other hand, has been described in the subsection 2.1.4 of this report.

## 3.6 Build

### 3.6.1 Version Control

Version control has been described in the subsection 1.3.1 of this report.

### 3.6.2 Continous Integration

CI has been described in the subsection 2.1.3 of this report. The final Circle CI config file has been included in the Appendix E of this report as Figure E.9.

# Chapter 4

# Testing

## 4.1 Testing Strategy

A combination of white box and black box testing strategies have been incorporated into Quiz Tool testing, in order to test the system reasonably in depth given the amount of time available. Test automation is essential when software is developed using an agile methodology. The application has been developed using single person adjusted SCRUM as described in the subsection 1.3.2 of this report. Certain tests have been added incrementally over the sprints, while others have been added towards the end of the project development. Automated tests have been incorporated into the Circle CI build, which decreased the likelihood of regression defects being introduced with new features implementations. The Circle CI build agent would checkout the code, and run all the tests before allowing new pull requests being merged into the production branch. Tests made sure the top level functional requirements listed in subsection 1.2.6 were met. It was important to test both the expected behaviour, and the edge cases to make the tool less likely to fail in the production environment. Finally, acceptance testing has been performed by running a real life lecture with the end users, once the first version of the tool has been implemented.

## 4.2 Server Side Unit Tests

The `server_node` container has been tested using mocha[59], chai[60] and chai-http[61] libraries following the tutorial *Test a Node RESTful API with Mocha and Chai*[62]. Mocha is a JavaScript test framework running the test cases, chai is an assertion framework allowing to check if variables have expected values, and chai-http can be used to call endpoints exposed by Node.js applications. The automated tests added to test the back end of the application, could arguably also be called integration tests, as endpoints are called and responses examined to assess if the Quiz Tool works as expected. However the state of the database is also examined directly during some tests, therefore all tests are referred to as server side unit tests in this report.

Server side tests are located in the `server/test` directory, and are started with the `mocha --recursive 'test/**/*.js' --timeout 60000 --exit` command located in the `server/package.json` file. The `docker-compose.test.yml`, located in the root directory of the project, is used to define how containers should be linked together for server side testing. The major difference of this file, compared to the `docker-compose.yml`, is that an

extra `NODE_ENV` environment variable is set in the `server_node` container to make sure the application is aware it is being tested. For example a special test database is created for testing. The entrypoint of the Docker container is also overriden to `npm test` to start the mocha tests.

Server side tests cover the major requirements of the tool, and also make sure the common errors are handled appropriately by the application. Authorisation has been tested, or actually attempts to call endpoints without an appropriate authorisation header, or a header containing a malformed JWT token. Other tests relying on the user being authenticated, use a special `AUTH_DISABLED` environment variable to use a mocked lectuer called Bob for testing. Database is also cleaned before a new set of tests runs. Lecture creation is tested by uploading a test PDF presentation as part of the tests. Tests also make sure slides are created as expected, slides can be marked as quizes, lectures be removed, sessions associated with lectures and reports generated. Finally, one of the good things about the mocha tests combined with the chai assertion library, is that they are incredibily easy to read even for non technical people.

```
it('should return 401 for a fake JWT token', (done) => {
    chai.request(app).get('/lecturers/logged-in')
      .set('Authorization', 'Bearer verySecureJwtToken123').end((err,
          res) => {
        res.should.have.status(401);
        done();
      });
  });
```

Figure 4.1: Server Side Test Example

## 4.3 Client Side Unit Tests

The client side of the application has been tested using testing dependencies which come preinstalled with every Angular application. Karma[63] testing framework, and Jasmine[64] to actually define the test cases. Each Angular component consists of a HTML template, a stylesheet file, a file describing the logic of a component written in TypeScript, and finally a `*.component.spec.ts` file including tests of the component. Each Angular Service file also has an associated test file. Angular applications can be tested by running the `npm test` command. The `client` container's production `Dockerfile`, shown in the Figure E.1 of this report, uses the multi stage Docker build. Angular is compiled down to a distribution folder which is exposed via nginx to all the clients. This means `npm`, and other test dependencies are no longer present in the container, therefore a special `Dockerfile.test` file, included in the Appendix E of this report as Figure E.10, has been added to address this problem. The major difference is the lack of nginx, and the entrypoint changed to `npm test`. Every time tests are run, a headless PhantomJS[65] web browser is started to make sure code works as expected.

Client side tests focus on making sure the offline, helper methods and services work as intended. Tests developed are true unit tests, as they make sure code works well in isolation. Services are tested, as long as they do not make HTTP calls to the back end. In order to produce such tests, the back end would have had to be mocked for testing purposes, and selenium[66] tests described in the next subsection make sure HTTP services work as expected instead. Code responsible for options extraction from slides' text have been tested, and every Components' internally

used methods have been tested. Finally, the UI of the application is not tested, as selenium tests cover UI requirements as well.

## 4.4 Selenium Integration Tests

## 4.5 User Tests

# Chapter 5

# Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

- Were the requirements correctly identified?
- Were the design decisions correct?
- Could a more suitable set of tools have been chosen?
- How well did the software meet the needs of those who were expecting to use it?
- How well were any other project aims achieved?
- If you were starting again, what would you do differently?

Other questions can be addressed as appropriate for a project.

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

In the latter stages of the module, we will discuss the evaluation. That will probably be around week 9, although that differs each year.

# Appendix A

# Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. If third party code or libraries are used, your work will build on that to produce notable new work. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

As an example, you might include a definition such as:

Apache POI library - The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client's existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [**?**]. The library is released using the Apache License [**?**]. This library was used without modification.

# Appendix B

# Ethics Submission

Ethics Application Number: 9624

**AU Status**
Undergraduate or PG Taught

**Your aber.ac.uk email address**
mwg2@aber.ac.uk

**Full Name**
Michal Wojciech Goly

**Please enter the name of the person responsible for reviewing your assessment.**
Reyer Zwiggelaar

**Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment**
rrz@aber.ac.uk

**Supervisor or Institute Director of Research Department**
cs

**Module code (Only enter if you have been asked to do so)**
CS39440

**Proposed Study Title**
MMP - Quiz Tool

**Proposed Start Date**
29/01/2018

**Proposed Completion Date**
04/05/2018

**Are you conducting a quantitative or qualitative research project?**
Mixed Methods

**Does your research require external ethical approval under the Health Research Authority?**
No

**Does your research involve animals?**
No

**Are you completing this form for your own research?**
Yes

**Does your research involve human participants?**
Yes

**Institute**
IMPACS

**Please provide a brief summary of your project (150 word max)**
A web application allowing lecturers to upload their PDF lecture slides, convert certain slides to quizzes (e.g. a slide with bullet points into a single choice A) B) C) quiz), and then broadcasting them to students to monitor their understanding of lecture content presented. A trial session with students taking part in a lecture presented over the tool would run, to assess if the tool can handle a larger amount of users, and to potentially gain feedback about the user interface.

**I can confirm that the study does not involve vulnerable participants including participants under the age of 18, those with learning/communication or associated difficulties or those that are otherwise unable to provide informed consent?**

Yes

**I can confirm that the participants will not be asked to take part in the study without their consent or knowledge at the time and participants will be fully informed of the purpose of the research (including what data will be gathered and how it shall be used during and after the study). Participants will also be given time to consider whether they wish to take part in the study and be given the right to withdraw at any given time.**
Yes

**I can confirm that there is no risk that the nature of the research topic might lead to disclosures from the participant concerning their own involvement in illegal activities or other activities that represent a risk to themselves or others (e.g. sexual activity, drug use or professional misconduct). Should a disclosure be made, you should be aware of your responsibilities and boundaries as a researcher and be aware of whom to contact should the need arise (i.e. your supervisor).**
Yes

**I can confirm that the study will not induce stress, anxiety, lead to humiliation or cause harm or any other negative consequences beyond the risks encountered in the participant's day-to-day lives.**
Yes

**Please include any further relevant information for this section here:**

**Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?**
Not applicable

**Will appropriate measures be put in place for the secure and confidential storage of data?**
Yes

**Does the research pose more than minimal and predictable risk to the researcher?**
Not applicable

**Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?**
No

**Please include any further relevant information for this section here:**

**If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.**
Yes

**Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.**
Yes

**Please include any further relevant information for this section here:**

# Appendix C

# Sprint Stories

This appendix contains the user stories produced before each sprint.

## 3.1 Sprint 1

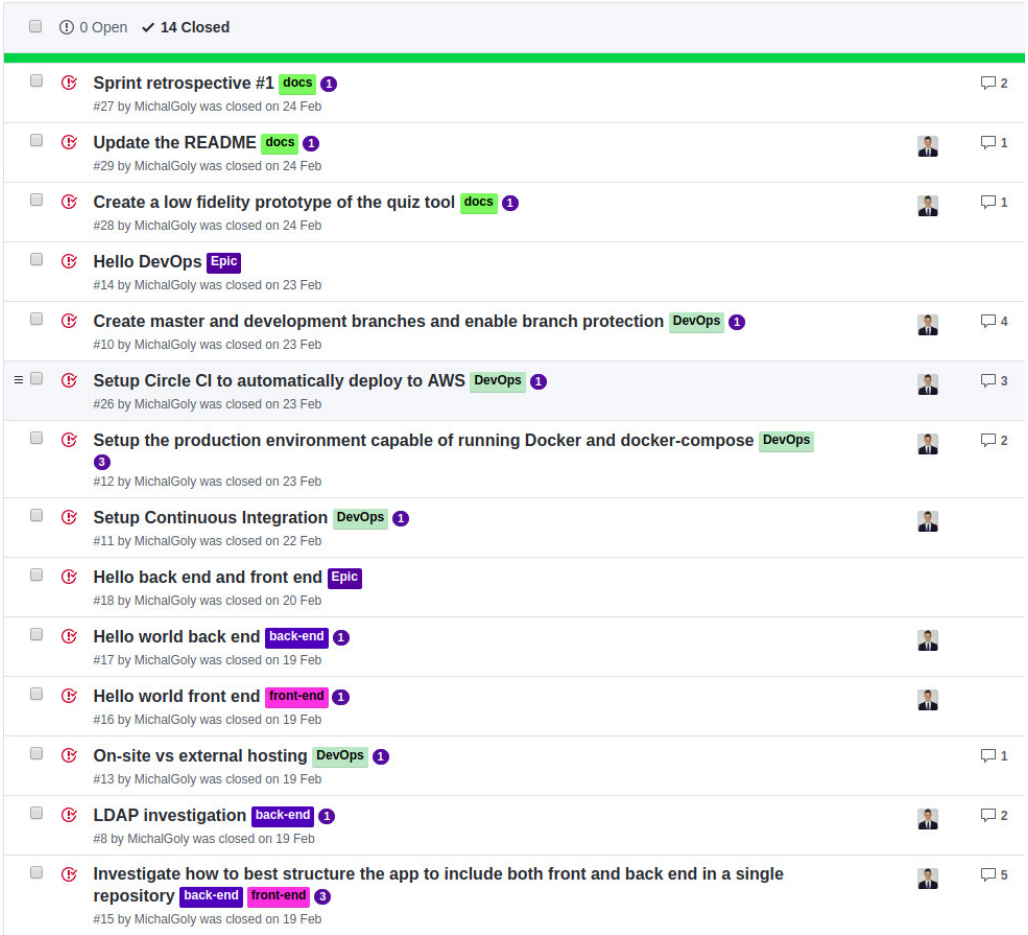


Figure C.1: Stories Sprint 1

## 3.2 Sprint 2

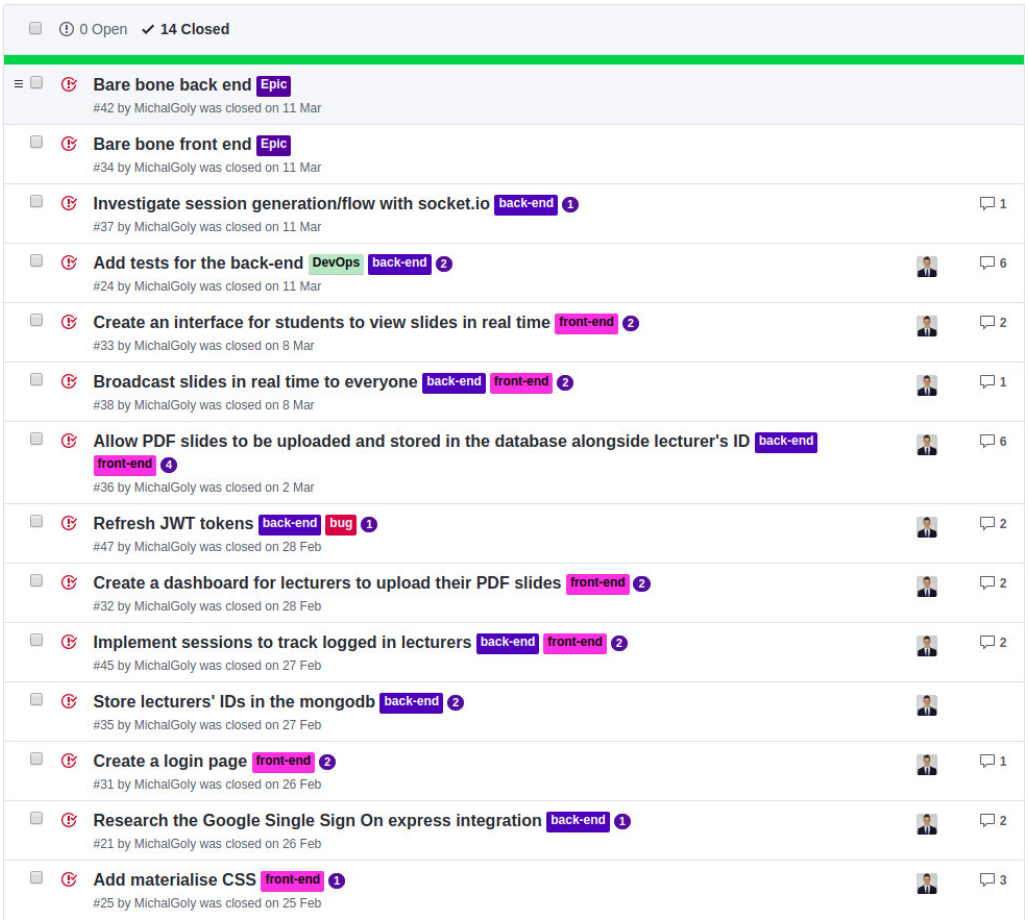


Figure C.2: Stories Sprint 2

## 3.3 Sprint 3

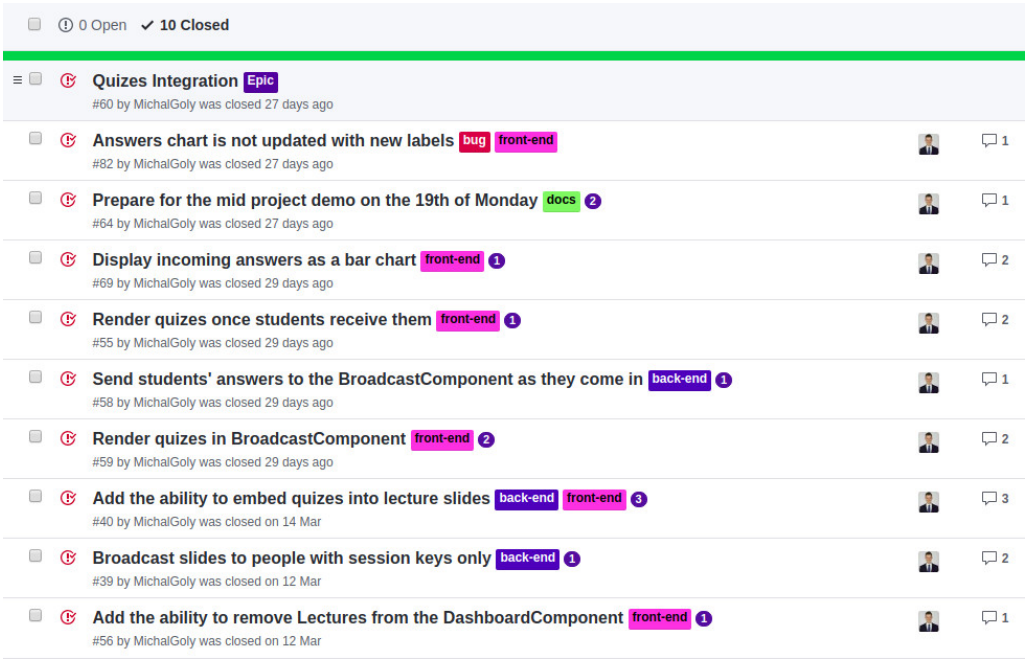


Figure C.3: Stories Sprint 3

## 3.4 Sprint 4

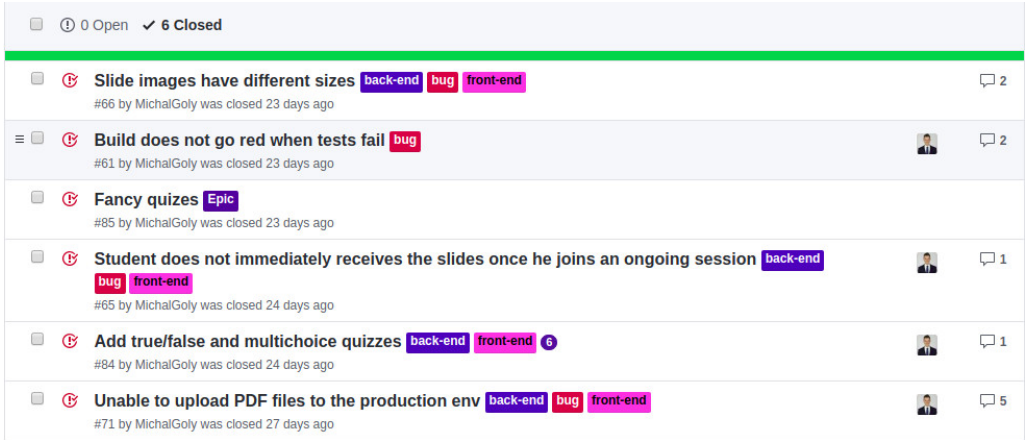


Figure C.4: Stories Sprint 4

## 3.5 Sprint 5

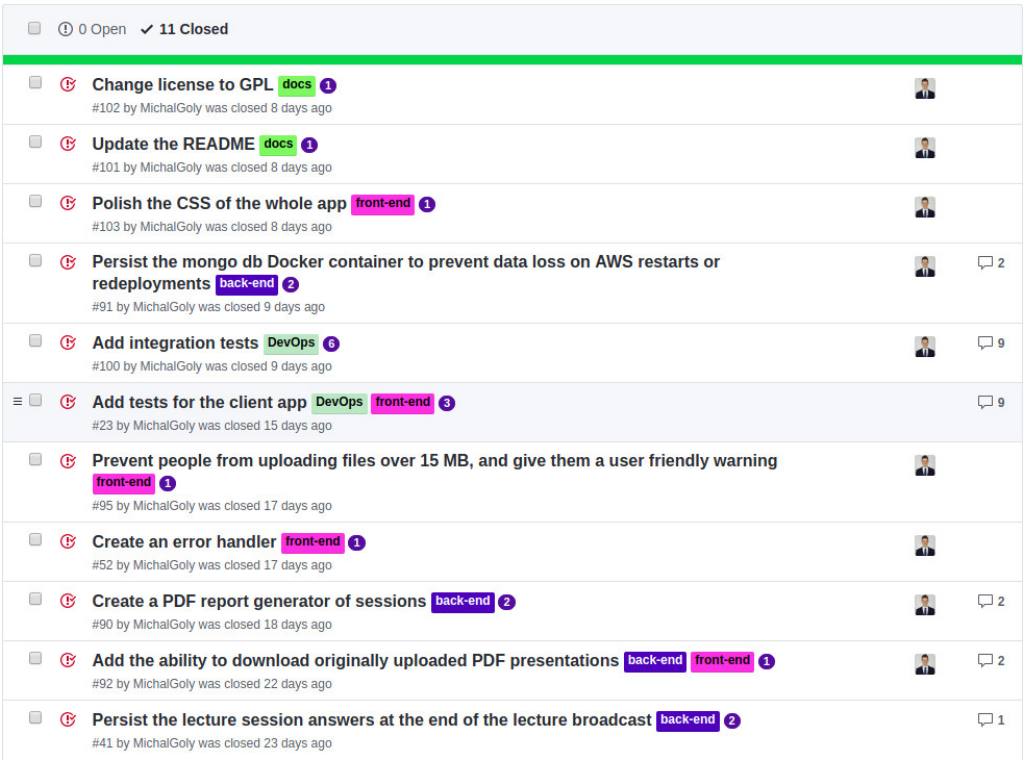


Figure C.5: Stories Sprint 5

# Appendix D

# Sprint Retrospective Documents

This appendix contains the sprint retrospective documents produced at the end of each sprint.

# Quiz Tool - Sprint #1 Retrospective

## What went well:

- LDAP authentication blocker resolved by going with the Google Single Sign On for Lecturers authentication
- Creation of the proof of concept “hello world” front and back ends capable on running in a docker-compose network with an nginx reverse proxy and socket.io
- DevOps setup is now complete. There is a Circle CI build in place which runs builds each time there is a pull request and auto deploys to AWS if master branch is being built.
- Circle CI <-> AWS integration. The proof of concept app is now running in production on the Amazon Multicontainer Elasticbeanstalk Environment and both the client and server Docker images are automatically pushed and stored in the Amazon Elastic Container Registry.
- I have created low fidelity prototypes of the tool which will allow me to plan the development in the future iterations
- I kept track of the time spent using google sheets

## What could be improved:

- I have only put 20 hours into the Final Project work this week due to 2 job interviews
- I could have probably squized more issues into the sprint, but I decided to start a new sprint one day early instead.
- Some stories estimated to take half-day work took 10 minutes, whereas others had to be extended to a day or a day and a half. Points allocation accuracy should hopefully improve once actual coding begins in the following sprints.

# Quiz Tool - Sprint #2 Retrospective

## What went well:

- Put in considerably more time into the development -> 46.5 hours over 5 days
- Added materialise CSS
- Login page done
- Google single sign on done
- Authorisation implemented using JWT session tokens stored in a session cookie
- Heavy refactoring of the back end -> no longer spaghetti bolognese
- Created a basic dashboard
- Designed the PDF upload flow
- Implemented the ability to upload PDF presentations, splitted them into slides by extracting PNG images and text which will be extremely useful later on in the development

## What could be improved:

- Sprint only lasted 5 days due to an interview in Zurich
- Sprint was overfilled with too much work, and the implementation of the upload PDF presentations was surprisingly difficult

# Quiz Tool - Sprint #2 (week two) Retrospective

## What went well:

- Put enough time to finish off the sprint ~20 h
- Implemented the ability to broadcast lecture slides in real time to everyone
- Added back-end tests and fixed some minor bugs and a major performance one where I am no longer sending files over the network unnecessarily.

## What could be improved:

- I did not have enough time to add front end tests, but they are not very crucial atm as the front end will be refactored anyway so adding Selenium tests does not make much sense right now.

# Quiz Tool - Sprint #3 Retrospective

## What went well:

- Put in 40.5 hours
- Implemented the ability to embed single choice quizzes into slides by marking them as slides. This is done by looking at the text extracted from each slide to make sure a certain format A), B), C) etc is followed.
- Rendered quizzes on students' side
- Rendered quizzes on lecturers' slide during broadcast by splitting the screen in half, and rendering a bar chart with answers as they come in
- Implemented the logic of navigating between the slides with keeping track of the answers submitted by students
- Thought about the way to persist the answers in the database
- Printed the session answers object to the console at the end of the broadcast, meaning the only thing left is to save it in MongoDB and add a PDF report generator
- Prepared for the mid project demo

## What could be improved:

- It took me a while to understand how updates of child components work, when property binding is used to inject mutable objects into the children. For example, rendering answers in real time as they come in by the broadcast/answers.component, which hold the liveAnswers object from its parent broadcast.component.
- Dealing with ng2-charts, chart.js wrapper was painful to make the bar chart update in real time
- Discovered a critical bug in production, where for some magical reason PDFtk command line utility randomly does not find files to convert to PNGs, even though they are there. This works perfectly fine on docker-compose in the development and testing environments, but for some reason ECS does not work. I was thinking I've introduced a race condition with the way my code works asynchronously, but synchronising access to the file system did not help. I will be therefore forced to re-implement PDF to PNG conversion using something other than PDFtk in the next iteration.
- Due to the unexpected bugs, I have run out of time to implement the ability to see the amount of people in the session in the broadcast component. This may not be as trivial as I initially thought, as students machines are not uniquely identified by the tool.

# Quiz Tool - Sprint #4 Retrospective

## What went well:

- Finished off all issues in 5 days.
- Fixed the issue of PDF to PNG conversion in the production environment. Removed the scissors.js dependency and used a different library to extract images from the PDF provided. Finally, extracted text and images were grouped together to produce slides in the system.
- Implemented true/false and multichoice quizzes into the tool. This required a slight redesign of answers objects being sent to the lecturer to use arrays instead of strings.
- Fixed bugs:
    - Newly joined students now receive the current slide if there is an ongoing session, without having to wait for the lecturer to broadcast next slide
    - Build goes red if tests on Circle CI fail
- Decided not to implement the ability to see the amount of currently connected students in a lecture. It's possible to get the total number of clients connected to the app using socket.io, but differentiating between them is not really possible without uniquely identifying students. The architectural redesign is not worth the added value of being able to see the amount of students, I have put it in the "Icebox" as a nice to have but beyond the scope of the final year project at this point.

## What could be improved:

- N/A

# Quiz Tool - Sprint #5 Retrospective

## What went well:

- Finished off the technical part of the Final Project
- Added Angular unit tests to test the front-end. Offline services and helper methods of certain components have been tested.
- I have added an error handler to display user friendly Toasts to users if something goes wrong
- Persisted lecture sessions, and added a component capable of displaying historical sessions for each Lecture
- Created a PDF report generator to export historical sessions
- Added the ability to download originally uploaded presentations
- Prevented people from uploading files over 15MB
- Persisted the mongo DB container on AWS using volume mount onto the EC2 instance.
- Polished the CSS of the whole application
- Changed the tool's license to GPLv2 and updated the README
- Added integration tests using Selenium. This required a creation of a tailored docker-compose file, using the selenium hub container, headless firefox and chrome, and adding an extra tester container capable of running the tests using a JavaScript implementation of the selenium web driver.
- Newly added tests are run as part of the Circle CI build

## What could be improved:

- I could have probably get everything done in a shorter amount of time
- I did not unit test Angular Components, and services which consumed the server side endpoints. This functionality is however tested with Selenium integration tests, and manual testing

# Appendix E

# Code Examples

This appendix contains code examples relevant to the discussion in the main body of the report.

## 5.1 Intial Front End Dockerfile

The initial `Dockerfile` of the client container added in the first sprint of the development of the tool.

```
### STAGE 1: Build ###

# We label our stage as 'builder'
FROM node:8-alpine as builder

COPY package.json package-lock.json ./

RUN npm set progress=false && npm config set depth 0 && npm cache
    clean --force

## Storing node modules on a separate layer will prevent
    unnecessary npm installs at each build
RUN npm i && mkdir /ng-app && cp -R ./node_modules ./ng-app

WORKDIR /ng-app

COPY . .

## Build the angular app in production mode and store the artifacts
     in dist folder
RUN $(npm bin)/ng build --prod --build-optimizer

### STAGE 2: Setup ###

FROM nginx:1.13.3-alpine

## Copy our default nginx config
COPY nginx/default.conf /etc/nginx/conf.d/

## Remove default nginx website
RUN rm -rf /usr/share/nginx/html/*

## From 'builder' stage copy over the artifacts in dist folder to
    default nginx public folder
COPY --from=builder /ng-app/dist /usr/share/nginx/html

CMD ["nginx", "-g", "daemon off;"]
```

Figure E.1: Intial Front End Dockerfile

## 5.2 Initial Back End Dockerfile

The initial `Dockerfile` of the server_node container added in the first sprint of the development of the tool.

```
FROM node:carbon
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD [ "npm", "start" ]
```

Figure E.2: Back End Dockerfile

## 5.3 Intial Circle CI Config File

The initial `.circleci/config.yml` added in the first sprint of the development.

```
version: 2
jobs:
  build:
    machine: true

    working_directory: ~/repo

    steps:
      - checkout

      - run:
          name: install docker-compose
          command: |
            set -x
            sudo chown -R $(whoami) /usr/local/bin
            curl -L https://github.com/docker/compose/releases/
                download/1.11.2/docker-compose-`uname -s`-`uname -m`
                > /usr/local/bin/docker-compose
            chmod +x /usr/local/bin/docker-compose

      - run:
          name: docker compose build image
          command: |
            set -x
            docker-compose build
            docker-compose up

      - run:
          name: unit tests
          command: |
            curl localhost

      # deploy only master branch
      - deploy:
          command: |
            if [ "${CIRCLE_BRANCH}" == "master" ]; then
              chmod +x scripts/deploy.sh
              ./scripts/deploy.sh
            fi
```

Figure E.3: Initial Build Config File

## 5.4 Initial Dockerrun.aws.json file

The initial version of the file specifying how containers should be linked together when running in the production environment on AWS.

```
{
    "AWSEBDockerrunVersion": 2,
    "containerDefinitions": [
        {
            "name": "client",
            "image": "993389244112.dkr.ecr.eu-west-2.amazonaws.com/
                quiz-tool-client:latest",
            "memory": 128,
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80
                }
            ],
            "links": [
                "server_node"
            ]
        },
        {
            "name": "server_node",
            "image": "993389244112.dkr.ecr.eu-west-2.amazonaws.com/
                quiz-tool-server:latest",
            "memory": 128,
            "essential": true,
            "links": [
              "database"
            ]
        },
        {
          "name": "database",
          "image": "mongo",
          "memory": 128,
          "essential": true,
          "portMappings": [
              {
                  "hostPort": 27017,
                  "containerPort": 27017
              }
          ]
        }
    ]
}
```

Figure E.4: Dockerrun.aws.json

## 5.5 Production Deployment Script

The bash script performing the deployment from Circle CI to the production environment hosted on AWS.

```
#!/bin/bash
set -x
set -e

AWS_CONFIG_FILE=$HOME/.aws/config

mkdir $HOME/.aws
touch $AWS_CONFIG_FILE
chmod 600 $AWS_CONFIG_FILE

echo "[default]"                                        >
    $AWS_CONFIG_FILE
echo "aws_access_key_id=$AWS_ACCESS_KEY_ID"            >>
    $AWS_CONFIG_FILE
echo "aws_secret_access_key=$AWS_SECRET_ACCESS_KEY" >>
    $AWS_CONFIG_FILE

$(aws ecr get-login --no-include-email --region eu-west-2)

docker tag repo_client:latest 993389244112.dkr.ecr.eu-west-2.
    amazonaws.com/quiz-tool-client:latest
docker tag repo_server_node:latest 993389244112.dkr.ecr.eu-west-2.
    amazonaws.com/quiz-tool-server:latest

docker push 993389244112.dkr.ecr.eu-west-2.amazonaws.com/quiz-tool-
    client:latest
docker push 993389244112.dkr.ecr.eu-west-2.amazonaws.com/quiz-tool-
    server:latest

eb deploy prod-env
```

Figure E.5: Production Deployment Script

## 5.6 Final Dockerrun.aws.json file

The final version of the file specifying how Docker containers should be link with each other when running in the production environment provided by AWS.

```
{
    "AWSEBDockerrunVersion": 2,
    "volumes": [
      {
        "name": "mongo",
        "host": {
          "sourcePath": "/var/app/database"
        }
      }
    ],
    "containerDefinitions": [
        {
            "name": "client",
            "image": "993389244112.dkr.ecr.eu-west-2.amazonaws.com/quiz-tool-client:latest",
            "memory": 300,
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "containerPort": 80
                }
            ],
            "links": [
                "server_node"
            ]
        },
        {
            "name": "server_node",
            "image": "993389244112.dkr.ecr.eu-west-2.amazonaws.com/quiz-tool-server:latest",
            "memory": 300,
            "essential": true,
            "links": [
              "database"
            ]
        },
        {
          "name": "database",
          "image": "mongo",
          "memory": 300,
          "essential": true,
          "mountPoints": [
              {
                "sourceVolume": "mongo",
                "containerPath": "/data/db"
              }
          ],
          "portMappings": [
              {
                  "hostPort": 27017,
                  "containerPort": 27017
              }
          ]
        }
    ]
}
```

Figure E.6: Final Dockerrun.aws.json

## 5.7 Final Back End Dockerfile

The final `Dockerfile` of the server_node container.

```
FROM node:carbon

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.
    json are copied
# where available (npm@5+)
COPY package*.json ./

# Install system dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
            ghostscript \
            pdftk \
            poppler-utils \
            graphicsmagick

RUN npm install
# If you are building your code for production
# RUN npm install --only=production

# Bundle app source
COPY . .

EXPOSE 3000
CMD [ "npm", "start" ]
```

Figure E.7: Back End Dockerfile

## 5.8 Final Nginx Config File

The final `default.conf` nginx configuration file, enabling routing within the application and preventing people from uploading files over 15 MB.

```
server
{
  listen 80;
  sendfile on;
  default_type application/octet-stream;

  gzip on;
  gzip_http_version 1.1;
  gzip_disable "MSIE [1-6]\.";
  gzip_min_length 256;
  gzip_vary on;
  gzip_proxied expired no-cache no-store private auth;
  gzip_types text/plain text/css application/json application/javascript application/x-javascript text/xml
      application/xml application/xml+rss text/javascript;
  gzip_comp_level 9;

  root /usr/share/nginx/html;

  location /
  {
    try_files $uri $uri/ /index.html =404;
  }

  location /express/
  {
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_set_header X-NginX-Proxy true;

    proxy_pass http://server_node:3000/;

    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";

    client_max_body_size 15M;
  }

  location ~* \.io
  {
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_set_header X-NginX-Proxy true;

    proxy_pass http://server_node:3000;
    proxy_redirect off;

    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
  }

}
```

Figure E.8: Final Nginx Config File

## 5.9 Final Circle CI Config File

The final `.circleci/config.yml` which evolved over the iterations.

```
# Javascript Node CircleCI 2.0 configuration file
#
# Check https://circleci.com/docs/2.0/language-javascript/ for more details
#
version: 2
jobs:
  build:
    machine: true

    working_directory: ~/repo

    steps:
      - checkout

      - run:
          name: AWS integration
          command: |
            sudo apt-get -y -qq update
            sudo apt-get install python-pip python-dev build-essential
            sudo pip install awsebcli --upgrade
            eb --version

      - run:
          name: install docker-compose
          command: |
            set -x
            sudo chown -R $(whoami) /usr/local/bin
            curl -L https://github.com/docker/compose/releases/download/1.16.1/docker-compose-`uname -s`-`uname -m`
                 > /usr/local/bin/docker-compose
            chmod +x /usr/local/bin/docker-compose

      - run:
          name: docker compose build image
          command: |
            touch .env
            set -x
            docker-compose build

      - run:
          name: run client unit tests
          command: |
            cd client
            docker build -t clienttest --file Dockerfile.test .
            docker run -it clienttest
            cd ..

      - run:
          name: run server unit tests
          command: |
            echo "GOOGLE_CLIENT_ID=$GOOGLE_CLIENT_ID" >> .env
            echo "GOOGLE_CLIENT_SECRET=$GOOGLE_CLIENT_SECRET" >> .env
            echo "GOOGLE_CALLBACK_URL=$GOOGLE_CALLBACK_URL" >> .env
            docker-compose -f docker-compose.test.yml up --abort-on-container-exit --exit-code-from server_node
            docker-compose -f docker-compose.test.yml down

      - run:
          name: run integration tests
          command: |
            docker-compose -f docker-compose.integration.yml build
            docker-compose -f docker-compose.integration.yml up --abort-on-container-exit --exit-code-from tester
            docker-compose -f docker-compose.integration.yml down

      # deploy only master and develop branch
      - deploy:
          command: |
            if [ "${CIRCLE_BRANCH}" == "master" ]; then
              chmod +x scripts/deploy.sh
              ./scripts/deploy.sh
            fi
```

Figure E.9: Final Build Config File

## 5.10 Front End Dockerfile for Testing

`Dockerfile.test` used for front end unit tests to address the lack of `npm` in the production container shown in the Figure E.1.

```
# Dockerfile for client side unit tests
FROM node:8

COPY package.json package-lock.json ./

RUN npm set progress=false && npm config set depth 0 && npm cache
    clean --force

## Storing node modules on a separate layer will prevent unnecessary
    npm installs at each build
RUN npm i && mkdir /ng-app && cp -R ./node_modules ./ng-app

WORKDIR /ng-app

COPY . .

RUN $(npm bin)/ng build

CMD npm test
```

Figure E.10: Dockerfile.test Used For Front End Testing

# Appendix F

# PDF Report Examples

This appendix contains examples of PDF reports generated with the Quiz Tool.

Lecture: L2-review-and-parameters.pdf
Date: 4/16/18, 1:45 PM

| Slide no | Answers | Correct |
|---|---|---|
| 1 | - | - |
| 2 | A: 2, B: 1 | B |
| 3 | - | - |
| 4 | BC: 2, AD: 1 | BC |
| 5 | B: 1, C: 1 | Correct not chosen |
| 6 | AB: 3 | C |
| 7 | - | - |
| 8 | - | - |
| 9 | - | - |
| 10 | - | - |
| 11 | - | - |
| 12 | - | - |
| 13 | - | - |
| 14 | - | - |
| 15 | - | - |
| 16 | - | - |
| 17 | - | - |
| 18 | - | - |
| 19 | - | - |
| 20 | - | - |
| 21 | - | - |

# Bibliography


[1] Qwizdom, Qwizdom homepage, 2018. [Online] Available: https://qwizdom.com/, [Accessed: Apr. 9, 2018].

Qwizdom is the tool currently used by the univeristy and will be replaced with this project.

[2] Google Sign-In, "Google Identity homepage", 2018. [Online] Available: https://developers.google.com/identity/, [Accessed: Apr. 10, 2018].

Google Sign-In is a secure authentication system that reduces the burden of login for your users, by enabling them to sign in with their Google account. The same account they already use with Gmail, Play, Google+, and other Google services.

[3] Java Programming Language, "Java homepage", 2018. [Online] Available: https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html, [Accessed: Apr. 11, 2018].

The Java Programming Language is a general-purpose, concurrent, strongly typed, class-based object-oriented language. It is normally compiled to the bytecode instruction set and binary format defined in the Java Virtual Machine Specification.

[4] Android, "Android homepage", 2018. [Online] Available: https://www.android.com/, [Accessed: Apr. 11, 2018].

Android is a mobile operating system developed by Google.

[5] iOS, "Apple homepage", 2018. [Online] Available: https://www.apple.com/uk/ios/ios-11/, [Accessed: Apr. 11, 2018].

iOS is a mobile operating system created and developed by Apple Inc.

[6] React Native, "React Native homepage", 2018. [Online] Available: https://facebook.github.io/react-native/, [Accessed: Apr. 11, 2018].

React Native lets you build mobile apps for both iOS and Android using only JavaScript.

[7] JavaScript Programming Language, "JavaScript Mozilla docs", 2018. [Online] Available: https://developer.mozilla.org/bm/docs/Web/JavaScript, [Accessed: Apr. 11, 2018].

JavaScript is a lightweight, interpreted, programming language, best known for being the scripting language of the web.

[8] React, "React homepage", 2018. [Online] Available: https://reactjs.org/, [Accessed: Apr. 11, 2018].

A JavaScript library for building user interfaces.

[9] Angular, ”Angular homepage”, 2018. [Online] Available: https://angular.io/, [Accessed: Apr. 11, 2018].

Angular is a TypeScript-based front end development framework supported by Google.

[10] TypeScript Programming Language, ”TypeScript homepage”, 2018. [Online] Available: https://www.typescriptlang.org/, [Accessed: Apr. 11, 2018].

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

[11] I. Fette, A. Melnikov, ”The WebSocket Protocol”, [Online] Available: https://tools.ietf.org/html/rfc6455, [Accessed: Apr. 11, 2018].

The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.

[12] Socket.io, ”Socket.io homepage”, 2018. [Online] Available: https://socket.io/, [Accessed: Apr. 11, 2018].

Socket.io is a JavaScript library enabling bidirectional event-based communication.

[13] Chat Application Tutorial, ”Socket.io homepage”, 2018. [Online] Available: https://socket.io/get-started/chat/, [Accessed: Apr. 11, 2018].

A Socket.io tutorial guiding the student in developing a basic chat application using Socket.io.

[14] Tour of Heroes Tutorial, ”Angular homepage”, 2018. [Online] Available: https://angular.io/tutorial, [Accessed: Apr. 11, 2018].

The Tour of Heroes tutorial covers the fundamentals of Angular, by building an app that helps staffing agency manage its stable of heroes.

[15] Building Chat Application using MEAN Stack (Angular 4) and Socket.io, ”djamware.com”, 2018. [Online] Available: https://www.djamware.com/post/58e0d15280aca75cdc948e4e/building-chat-application-using-mean-stack-angular-4-and-socketio, [Accessed: Apr. 11, 2018].

Step by step tutorial of building simple chat application using MEAN stack (Angular 4) and Socket.io.

[16] J. Sermersheim, Ed. Novell, ”Lightweight Directory Access Protocol (LDAP): The Protocol”, [Online] Available: https://tools.ietf.org/html/rfc4511, [Accessed: Apr. 11, 2018].

Aberystwyth University directory of users could be checked using the Bind operation to authenticate lecturers once they provide their email and password.

[17] Microsoft PowerPoint, ”products.office.com”, 2018. [Online] Available: https://products.office.com/en-gb/powerpoint, [Accessed: Apr. 11, 2018].

A proprietary tool owned by Microsoft to create presentations.

[18] MEAN stack, [Online] Available: https://github.com/linnovate/mean, [Accessed: Apr. 11, 2018].

The MEAN stack uses Mongo, Express, Angular(4) and Node for simple and scalable full-stack js applications.

[19] Docker, "Docker homepage", [Online] Available: https://www.docker.com/, [Accessed: Apr. 11, 2018].

Docker allows app to be containerised and run on multiple hosts in the same fashion.

[20] docker-compose, "docker-compose homepage", [Online] Available: https://docs.docker.com/compose/, [Accessed: Apr. 11, 2018].

Compose is a tool for defining and running multi-container Docker applications.

[21] M. OǴara, "Ben Golub, Who Sold Gluster to Red Hat, Now Running dotCloud", 26th July 2013. [Online] Available: http://maureenogara.sys-con.com/node/2747331, [Accessed: Apr. 11, 2018].

A journal article mentioning what Docker does.

[22] LXC, "LXC", [Online] Available: https://wiki.debian.org/LXC, [Accessed: Apr. 11, 2018].

Linux Containers (LXC) provide a Free Software virtualization system for computers running GNU/Linux. They are inappropriate for the task at hand, as they do not allow running docker-compose due to the lack of system permissions.

[23] Jenkins, "Jenkins homepage", [Online] Available: https://jenkins.io/, [Accessed: Apr. 11, 2018].

Jenkins is an industry proven automation tool. It is typically deployed on premises as a Java web application and gives developers more control over the build agents compared to its competitors.

[24] Amazon Web Services, "AWS homepage", [Online] Available: https://aws.amazon.com/, [Accessed: Apr. 11, 2018].

AWS provides cloud computing plaftoms to individuals, companies and governments. The production environment of Quiz Tool is hosted on AWS.

[25] GitHub, "Student Developer Pack", [Online] Available: https://education.github.com/pack, [Accessed: Apr. 11, 2018].

GitHub partners with top companies to allow students gain experience with real world tools. It funded the production environment hosting on AWS of the Quiz Tool.

[26] Circle CI, "Circle CI homepage", [Online] Available: https://circleci.com/, [Accessed: Apr. 12, 2018].

Circle CI is very similar to Travis, but it allows a certain amount of free build hours per month for private repositories.

[27] Travis, "Travis homepage", [Online] Available: https://travis-ci.org/, [Accessed: Apr. 12, 2018].

Travis is a popular continuos integration tool hosted typically on an external cloud. Free to use for open source projects.

[28] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices", 2017, [Online] Available: https://arxiv.org/pdf/1703.07019.pdf, [Accessed: Apr. 12, 2018].

This paper describes what continuos integration and delivery are.

[29] Git, Git homepage, 2018, [Online] Available: https://git-scm.com/, [Accessed: Apr. 12, 2018].

Version control system used for the Quiz Tool development.

[30] GitHub, Inc., GitHub homepage, 2018, [Online] Available: http://github.com/, [Accessed: Apr. 12, 2018].

A popular version control web hosting offering private repositories to students. Code will be stored there and checkout by CI for testing and deployment.

[31] Git Feature Branch Workflow, [Online] Available: https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow, [Accessed: Apr. 12, 2018].

Each issue has an associated feature branch, and code necessary to develop the issue is commited to the appropriate feature branch. Then a pull request is made between the feature branch and development/master, and once all tests pass the feature branch can be merged to development/master.

[32] ZenHub, "ZenHub homepage", [Online] Available: https://www.zenhub.com/, [Accessed: Apr. 12, 2018].

ZenHub is an agile project management chrome extension for GitHub. It provides the ability to assign points to issues, create epics, provides velocity tracing burndown charts for milestones, and a Kanban board to better manage work at hand.

[33] Google Sheets, "Google docs homepage", [Online] Available: https://www.google.co.uk/sheets/about/, [Accessed: Apr. 12, 2018].

A SaaS tool for creating spreadsheets owned by Google. A spreadsheet has been used to track time commitment on day to day basis during the development of the Quiz Tool.

[34] nginx, "nginx homepage", [Online] Available: https://www.nginx.com/, [Accessed: Apr. 12, 2018].

Nginx is a web server which is used as a reverse proxy in the Quiz Tool.

[35] avatsaev, "Dockerized Angular 4 App (with Angular CLI)", [Online] Available: https://github.com/avatsaev/angular4-docker-example, [Accessed: Apr. 12, 2018].

The starting point of the client side of the Quiz Tool.

[36] mongo, "mongo official docker repository", [Online] Available: https://hub.docker.com/_/mongo/, [Accessed: Apr. 12, 2018].

The official docker hub repository of MongoDB. The image is used by the Quiz Tool in both production and development.

[37] What Is AWS Elastic Beanstalk, "AWS docs", [Online] Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html, [Accessed: Apr. 13, 2018].

The Quiz Tool is deployed to AWS Elastic Beanstalk production environment.

[38] Multicontainer Docker Environments, "AWS docs", [Online] Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_docker_ecs.html, [Accessed: Apr. 13, 2018].

The Multicontainer AWS Elastic Beanstalk is what actually orchestrates Docker containers in production.

[39] Amazon EC2, ”EC2 homepage”, [Online] Available: https://aws.amazon.com/ec2/, [Accessed: Apr. 13, 2018].

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.

[40] Amazon ECS, ”ECS homepage”, [Online] Available: https://aws.amazon.com/ecs/, [Accessed: Apr. 13, 2018].

Amazon Elastic Container Service stores Docker images of the Quiz Tool tagged and ready for deployment into production.

[41] How to setup Elastic Beanstalk Deployment, ”circleci forum”, [Online] Available: https://discuss.circleci.com/t/how-to-setup-elastic-beanstalk-deployment/6154/4, [Accessed: Apr. 13, 2018].

The forum post shows an example of an AWS deployment script which was adapted for Quiz Tool deployment.

[42] Settings to deploy to AWS Elastic Beanstalk on CircleCi (EB Cli 3) , ”gist”, [Online] Available: https://gist.github.com/RobertoSchneiders/9e0e73e836a80d53a21e, [Accessed: Apr. 13, 2018].

The gist post shows an example of an AWS deployment script which was adapted for Quiz Tool deployment.

[43] Materialize CSS, ”Materialize homepage”, [Online] Available: http://materializecss.com/, [Accessed: Apr. 14, 2018].

The CSS framework used in the development of the Quiz Tool.

[44] JSON Web Token, ”JWT standard”, [Online] Available: https://tools.ietf.org/html/rfc7519, [Accessed: Apr. 14, 2018].

JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties. Quiz Tool uses JWT tokens during authorisation.

[45] passport.js, ”passport.js homepage”, [Online] Available: http://www.passportjs.org/, [Accessed: Apr. 14, 2018].

Passport is an authentication middleware for Node.js used during Quiz Tool authentication.

[46] passport-google-oauth, ”passport-google-oauth github page”, [Online] Available: https://github.com/jaredhanson/passport-google-oauth, [Accessed: Apr. 14, 2018].

Passport middleware Google authentication strategy used in Quiz Tool.

[47] OAuth2, ”OAuth 2.0 Authorization Framework”, [Online] Available: https://tools.ietf.org/html/rfc6749, [Accessed: Apr. 14, 2018].

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to a resource stored on a different server. Quiz Tool uses it to authenticate lecturers using their Google credentials.

[48] mongoose, ”mongoose homepage”, [Online] Available: http://mongoosejs.com/, [Accessed: Apr. 14, 2018].

Mongoose provides a schema-based solution to model data in applications using MongoDB.

[49] Binary JSON, ”BSON homepage”, [Online] Available: http://bsonspec.org/, [Accessed: Apr. 15, 2018].

A binary-encoded serialization of JSON-like documents. Quiz Tool uses BSON as datatype of files persisted in MongoDB.

[50] GridFS, ”mongo docs”, [Online] Available: https://docs.mongodb.com/manual/core/gridfs/, [Accessed: Apr. 15, 2018].

A convention for storing large files in a MongoDB database.

[51] ng2-file-upload, ”ng2-file-upload homepage”, [Online] Available: https://valor-software.com/ng2-file-upload/, [Accessed: Apr. 15, 2018].

Library simplifying file upload from Angular applications.

[52] pdf-extract, ”pdf-extract github repository”, [Online] Available: https://github.com/nisaacson/pdf-extract, [Accessed: Apr. 15, 2018].

Node PDF is a set of tools that takes in PDF files and converts them to usable formats for data processing.

[53] scissors, ”scissors github repository”, [Online] Available: https://github.com/tcr/scissors, [Accessed: Apr. 15, 2018].

PDF manipulation in Node.js, based on PDFTK. This library has been initially used to extract PNG images from PDF slides.

[54] The Base16, Base32, and Base64 Data Encodings, ”Base64 standard”, [Online] Available: https://tools.ietf.org/html/rfc4648, [Accessed: Apr. 15, 2018].

Base64 is a binary-to-text encoding scheme used in the Quiz Tool to broadcast image slides to all the students.

[55] PDFtk Server, ”PDFtk homepage”, [Online] Available: https://www.pdflabs.com/tools/pdftk-server/, [Accessed: Apr. 15, 2018].

PDFtk is a command line utility for working with PDF files. It is used by the Quiz Tool internally when PNG images are extracted from PDF presentations uploaded by lecturers.

[56] node-pdf2img, ”node-pdf2img GitHub repository”, [Online] Available: https://github.com/fitraditya/node-pdf2img, [Accessed: Apr. 15, 2018].

A NodeJS module for converting pdf into image files.

[57] jsPDF, ”jsPDF GitHub repository”, [Online] Available: https://github.com/MrRio/jsPDF, [Accessed: Apr. 16, 2018].

Client-side JavaScript PDF generator, used for report generation in the Quiz Tool.

[58] jsonschema, ”jsonschema GitHub repository”, [Online] Available: https://github.com/tdegrunt/jsonschema, [Accessed: Apr. 17, 2018].

JSON Schema validator is used to make sure JSON send together with certain routes to the back end of the Quiz Tool is valid.

[59] mocha, ”mocha homepage”, [Online] Available: https://mochajs.org/, [Accessed: Apr. 17, 2018].

Mocha is a test framefork for Node.js, it is used in back end unit testing of the Quiz Tool.

[60] chai, ”chai homepage”, [Online] Available: http://www.chaijs.com/, [Accessed: Apr. 17, 2018].

Chai is an assertion library used in the back end unit testing of the Quiz Tool.

[61] chai-http, ”chai-http GitHub repository”, [Online] Available: https://github.com/chaijs/chai-http, [Accessed: Apr. 17, 2018].

Chai-http is an integration testing library allowing creation of tests which can call endpoints exposed by the Node.js applications.

[62] S. Zaza, ”Test a Node RESTful API with Mocha and Chai”, [Online] Available: https://scotch.io/tutorials/test-a-node-restful-api-with-mocha-and-chai, [Accessed: Apr. 17, 2018].

Tutorial explaining how to incorporate automated testing in a Node.js project. The tutorial was the basis of the server side unit tests of the Quiz Tool.

[63] Karma, ”Karma GitHub repository”, [Online] Available: https://karma-runner.github.io/2.0/index.html, [Accessed: Apr. 18, 2018].

Karma is the test runner which comes with Angular, and has been used for the front end unit tests of the Quiz Tool.

[64] Jasmine, ”Jasmine GitHub repository”, [Online] Available: https://jasmine.github.io/, [Accessed: Apr. 18, 2018].

Jasmine is a testing framework for JavaScript. It comes with Angular and is used for front end unit testing of the Quiz Tool.

[65] PhantomJS, ”PhantomJS homepage”, [Online] Available: http://phantomjs.org/, [Accessed: Apr. 18, 2018].

PhantomJS headless web browser is spin up by Karma during front end unit tests.

[66] Selenium, ”Selenium homepage”, [Online] Available: https://www.seleniumhq.org/, [Accessed: Apr. 18, 2018].

Selenium is a tool which automates browsers. It has been used to driver browser interactions during integration/end-to-end tests.