One Semester Individual Project

CM3203

Final Project

Training and Recruitment Assessment System for South Wales Police

Author: Jamie Morris (C1123702)

Supervisor: Dr. Irena Spasić

Moderator: Dr. Andrew Jones

Abstract

The Learning Development Services Unit are responsible for the entire training provision for South Wales Police. The introduction of independent learning programmes, through workbooks and assessments, was designed to gauge current knowledge and understanding in an attempt to decrease the number of trainer contact hours. The current system of producing and securely storing hard copy assessments is inadequate and costly but is necessary to provide evidence of student learning. However, this has facilitated robust measures for quality assurance through version control, compliance with the Management of Police Information Act and security through access control.

During a period of austerity resulting in a reduced workforce, the process of assessing examinations, evaluations, workbooks and application forms has become onerous. This project aimed to reduce the human resources required to mark and assess and hence reduce costs in terms of labour and ease pressure on staff by eliminating the requirement for marking and assessing by hand where this is not necessary.

This project delivered a bespoke software application that has eased the assessment and evaluation process within the Learning Development Services Unit of South Wales Police. The system provides an automated method of creating and marking Student assessments, reducing the human resources required in this process.

Acknowledgments

I would like to thank my Supervisor, Dr Irena Spasić for flexible meeting times that enabled me to get regular helpful feedback that has assisted me in completing this project.

I would also like to thank my internal Client at South Wales Police who was regularly available, which enabled me to follow my development model and to discuss and demonstrate my progress on the project.

Table of Contents

Table of Figures

Introduction

Defining the Problem

The department within South Wales Police required a tool that would support learning and development of professionals, human resources personnel and management teams to assess and evaluate large volumes of written exams, application forms and performance appraisals in training and recruitment. The assessment system required capability for three types of questions within assessments:

- Free-text answers
- Multiple choice answers
- Call log answers

An automated solution to set and mark all of these types of questions, which could also be used as an instant feedback feature for Students. It was also required to allow Students to download a report containing their results for their own use. Assessments are version controlled and are divided into versions for each new assessment and to ensure assessments are fair for Students a version cannot be altered after it has already been use to test on.

The system is targeted for use by both Trainers and Students as both user types have some form of recordable activities on the system and both types of users are able to authenticate themselves against the system. Staff within the department have also recently received tablet devices to carry out their day-to-day activities, it was therefore required for the application to be usable on smaller resolution devices to allow them to perform all of the actions they could do on a desktop computer. The main technological problems lie around the choice of software tools and interoperability with the current system architecture within the Police.

Deliverables

This project has resulted in a fully functional system which incorporates all of the features mentioned in the previous section. The full source code is available at GitHub (https://github.com/J-Morris676/SWPAssessment). Deployment is relatively simple as it only requires Node.js and MongoDB.

I have also created weekly reports containing my progress on a weekly basis on the project using Google Docs, it has not only been a helpful form of documentation but a shared place where my Supervisor could regularly check my progress:

- https://docs.google.com/document/d/1mOfn7Nt0FDyH3OToEz1IG_EP2ZKlZ8XhmerfqpuEHwU/edit?usp=sharing

Background

Existing Assessment Systems & Austerity

It is no secret that during the past five years austerity measures and drastic budgetary cuts to the public service have and continue to cause considerable strain to the policing service (BBC, 2015 http://www.bbc.co.uk/news/uk-31771456). Redundancies and early retirement options have drastically reduced the establishment and for this reason this form of labour and cost saving software is so important to this department. Whilst there are software tools that allow one to create Student assessments, requirements unique to policing result in the need for bespoke features such as questions based on call logs. This makes it difficult for the Client at South Wales Police to find a suitable off-the-shelf solution. Some have been considered in the past, and quotes received but ultimately ruled out due to the extremely high costs involved.

Existing systems in place at South Wales Police are listed below and demonstrate some of the reasons the department have made the decision to source an alternative solution.

- Kallidus Learning Management System
    - An initially inexpensive basic version of this system is already in place at South Wales Police (SWP). Kallidus provided updates free of charge to its customers on the proviso that data is hosted on their own servers. Due to the enhanced security required, SWP host the system internally and therefore any updates are carried out by a Kallidus consultant at a cost of £895 per day,
    - Additionally, the consultants are each required to be vetted due to the sensitive information involved prior to being given access at considerable cost to SWP.
    - Historically, logistical difficulties have been experienced synchronising Kallidus consultant availability with that of SWP ICT Database managers and engineers.
    - Though generic in content, bolt-on features are expensive and in addition, the cost of adding additional administrators (trainers) to create assessments is prohibitive.
    - Kallidus provides an appraisal system which is incompatible with the existing Human Resources system and all updates proposed in the future are based on the assumption that Kallidus customers are utilising this which SWP is resisting.

- Adobe Captivate
    - Whilst this product has been supplied free of charge and is currently in use by SWP it falls far short of delivering the unique and sophisticated requirements of the Learning and Development Unit and is used only to supply randomly selected multi-choice style questions at a student.
    - This product does not have the capacity to store student data or analyse results.

Security Issues

As there are two different user roles on the system, some preventative measures needed to be in place to keep data secure. It was particularly important that Students should have no method (back door or otherwise) of accessing either Trainer data, other Student's data or indeed, the answers. In order to overcome this a structured authorisation method has been researched and implemented.

Deployment

It is required for the application to be hosted internally to South Wales Police as they currently already use locally hosted web applications and is the reason why it has not been deployed to any server to date.

Data Protection Act 1998

We aligned our privacy requirements with the Data Protection Act 1998, which notably states that the use of personal data must be:

1. Used fairly and lawfully
2. Kept safe and secure

This act of parliament is an important one to consider for use in the application and during the entire development process. I have used dummy data sets including mock assessments and random user data meaning that it will not be possible to affect any personal data. In the deployed version, as mentioned before, it will be locally hosted and protected from external access so it would not be possible to access any of the data externally to South Wales Police.

Internally, the service layer is protected using username/password authentication and the database itself would require credentials to gain access to protecting data from any potential internal attacks.

Specification and Design

The majority of my aims and objectives highlighted in my initial report provide a high level overview of what the system requires for it to satisfy client requirements. In order to achieve these aims and objects professionally, I have divided major aspects of the system up into sections. These sections target the requirements defined below, most of which are derived from my aims and objectives.

Requirements

The aims and objectives in my initial report outlined what I wanted to achieve throughout the duration of this project. The functional requirements outlined below define how the systems components should function and how each of the components should act when certain operations are completed.

As it was possible to have weekly meetings with the Client I chose to use these as 'iterations' throughout the project. This allowed me to implement requirements in iterations of one week and test them ready for the next Client meeting resulting in some form of working application that could be demonstrated to the Client. This concurrent testing and implementation is the reason I decided to utilise an agile software development approach, due to this not all of the requirements were gathered at the beginning of the project as you would normally expect. Instead, these requirements (mainly additional requirements) were gathered at each iteration when a new aspect of the system needed to be implemented, here I will define the requirements gathered over all of the iterations.

Both Users - Covers Trainer and Student requirements to prevent duplication

1. The system must allow the user to sign in to the system using their credentials and direct them to their respective system area
   Acceptance Criteria - On sign in, the system navigates user to a page displaying the users content and their profile details.
2. The system must allow the user to sign out of the system
   Acceptance Criteria - When signing out using a logout button or similar, the user should be navigated to the sign in page.
3. *Additional* - The system should utilise a common tooltip feature displaying helpful information about components
   Acceptance Criteria - All of the ambiguous components have visually rich guidance of what the components is and/or what it does

Trainers

1. The system must allow the user to create, edit, view and delete assessments, versions of assessments and the questions within each version
   Acceptance Criteria - A page is provided to:
   - Input assessments that when submitted, appear in the list mentioned below
   - View a list of all of the created assessments with options to delete and edit
   - When edit is selected, user is directed to a page where the assessment can be fully edited and saved

2. The system must allow the user to create, edit, view and delete assessment Schedules

   Acceptance Criteria - When requirement 1 of Trainers is complete, an assessment Schedule can be created, edited and deleted.
3. The system must allow the user to create, edit, view and delete student accounts and view any results they have in assessments

   Acceptance Criteria - When created, the Student can be seen by the Trainer and the Student will be able to sign in with the given password. An edited Student is able to sign in with the new password when edited by the Trainer.
4. The system must allow the user to edit their own details (first name, last name & password)

   Acceptance Criteria - When edited, the user is able to sign in with the new password and first name/last name will be updated
5. The system must allow the user to download a document displaying the results of Students in completed assessments

   Acceptance Criteria - When the user finishes a scheduled assessment a download button will become available on the results page for a particular Student, this will start the download of a PDF file displaying the results of the assessment

Students

1. The system must allow the user to sit an assessment that they have been scheduled to sit at the correct start time

   Acceptance Criteria - When the start time of the scheduled assessment is reached, the system should allow the user to sit the assessment during this period:
   - If the time taken runs over the end time without completion the session will be automatically terminated and the user will be redirected
   - If the user gets to the end of the assessment and submits the assessment, the system automatically marks their assessment and shows them the result
2. The system must allow the user to submit an assessment during the period of time between a scheduled assessments start and end time

   Acceptance Criteria - The assessment for the user should be marked as completed and a grade becomes available, user is automatically notified of this on submission
3. The system must allow the user to visit the results of their completed assessments

   Acceptance Criteria - When the user finishes a scheduled assessment the results of this assessment will become available to view on their home page
4. The system must allow the user to download a document displaying limited results of their completed assessment

   Acceptance Criteria - When the user finishes a scheduled assessment a download button will become available on the results page, this should start the download of a PDF file displaying the limited results of the assessment
5. *Additional* - The system will lock assessments that have been used for testing

   Acceptance Criteria - When a user starts an assessment, that assessment is **locked** for editing preventing any changes being made to it and can be tested by attempting to delete it

System Architecture

As the architecture and technologies I used for this project were entirely up to me, I was left with the responsibility of designing the structure of the system. With the requirement of a database, I had two main options to consider:

- A local application that makes direct calls to the database

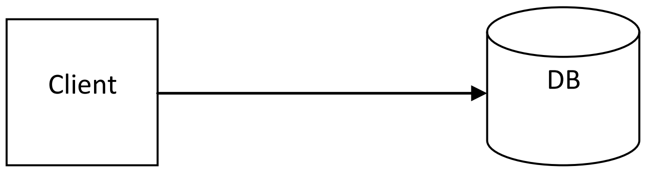


Figure 1: Client-Database model

  - **Advantages:**
    - Simple to setup
    - Client has only one remote dependency
  - **Disadvantages:**
    - Limited use of technologies
    - Overall client size will be **large** as it has to contain code for UI, logic & database communication

- A client/server model architecture where the server handles requests from clients and transactions to the database

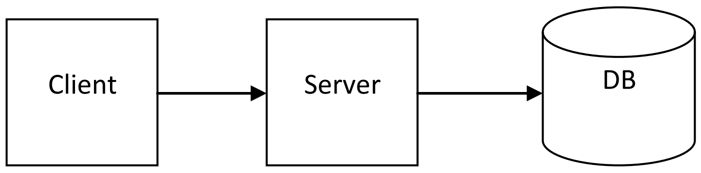


Figure 2: Client-Server-Database Model

  - **Advantages:**
    - Opens a new array of usable technologies (web architecture etc.)
    - Logic & communication code is split up over two nodes meaning client size is smaller.
    - Commonly used architecture
    - Scalable when used with common guidelines (REST)
  - **Disadvantages:**
    - Client depends on one more node (Server & Database)
    - More complicated to setup

To help in the decision of which architecture to implement, it's important to consider existing deployment methods and architecture within South Wales Police where they run local web applications on local servers that is entirely inaccessible outside of their firewall. With permission to implement this application using modern browsers (e.g. IE11/Chrome26+ https://html5test.com/results/desktop.html) I decided to implement a client/server model architecture as a web application.

With the size of the system looking like it was going to be larger than anything I have ever implemented before I was cautious about being able to easily scale the application up and encapsulate different areas as much as possible. I therefore decided to look into web services which provide an interface to retrieve a resource or perform some action, for example, a given client (e.g. web browser) request to the web server looks similar to the diagram in figure 3.

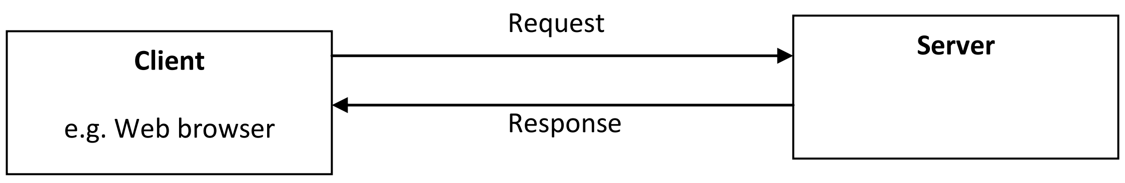


Figure 3: HTTP Server-Client Model

An important consideration is the transfer method and data interchange format to use as this can affect the development design and performance of the application, until recently, SOAP was the dominating method of web services but REST has quickly become the most used.

SOAP (Simple Object Orientated Protocol)	
Advantages	**Disadvantages**
Can be developed in any programming language	Contains a heavy overhead of XML content
It's platform independent	Difficult to develop with, requires tools
Utilises well know XML schemas	
Lacks support in modern web technologies	

Figure 4: SOAP Advantages and Disadvantages

REST (Representational State Transfer)	
Advantages	**Disadvantages**
Can be developed in any programming language	Lacks in agreed standards and it is **not** a protocol
It's platform independent and can be used by any technology using standard HTTP	Not suitable for large amounts of data
Can utilise XML or JSON as a data interchange format	
Very simple to develop with	
Well supported in modern web technologies	

Figure 5: REST Advantages and Disadvantages

The biggest driving force behind REST is how lightweight and simple it is compared to SOAP. The application required relatively small requests and responses, written with modern web technologies and needed to be fast to develop. With this being my first experience with constructing web services the advantages of REST compared to those of SOAP seemed much more ideal for this project.

REST consists of a set of software architectural guidelines for HTTP services to follow and found that the use of Representational State Transfer (REST) would allow me to:

- Construct a very easy to use and understandable API interface that follows file directory structures that we are all familiar with along with HTTP verbs, for example, it is easy to understand what the following request is supposed to do:
    - GET http://example.com/resources/students
- Use extremely lightweight requests with the use of any data format (No extreme XML overhead like SOAP)
- Gain experience in an increasingly used technology that is implemented by major companies (Twitter, Facebook, Microsoft etc. all provide some form of REST API for data access).
- Use JavaScript Object Notation (JSON) as a data interchange format which is lightweight and derived from JavaScript objects (Used in client web applications anyway).

JSON is an alternative to XML as a way of representing and transferring data in a way that is easily readable by humans and computers, a basic example representing a person can be seen in figure 6.

```
{
        "firstName": "Bob",
        " surname": "Smith",
        "telNumber": 0123456789,
        "isAdmin": true
}
```

Figure 6: JSON Example

A graph on Google Trends shows the XML/JSON interest based off news headlines and clearly displays how JSON has propelled forward as a data interchange format after an explosion of popularity around 2007.

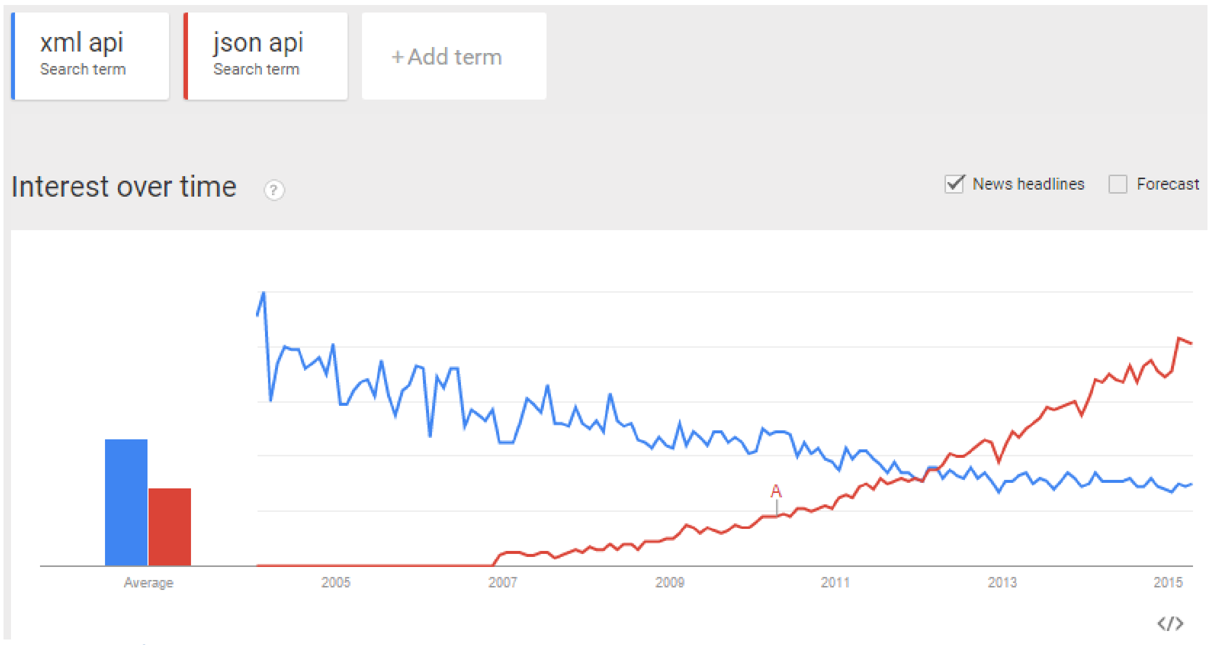


Figure 7: XML/JSON Interest Over Time

Use Case Diagram

As we have more than one type of user for the application it makes sense to represent the roles of these users in a diagram. Both user types of the system have to be given credentials:

- Trainers are provided with credentials from a database administrator who inserts their details directly into the database
- Students are provided with credentials from a Trainer who can create Student accounts.

It is also important to note that resources appear differently depending if you're a Student or a Trainer, this is difficult to represent in a use case diagram and has been outlined below:

- Downloading a results report as a Trainer will show questions, answers and marks whereas it'll only show the question and marks for a Student.
- Assessment schedules and assessments are not accessible to the Student unless they are enrolled onto them (with very limited access)
- Student results on the application appear with just the questions but Trainers will be able to see the answers as well

Below is a high level diagram displaying both users interactions with the system that contains all of the functional requirements defined earlier in the report. A Trainer can do a lot more than a Student as they are responsible for *creating*, *reading*, *updating* and *deleting* (**CRUD**) features of the system.

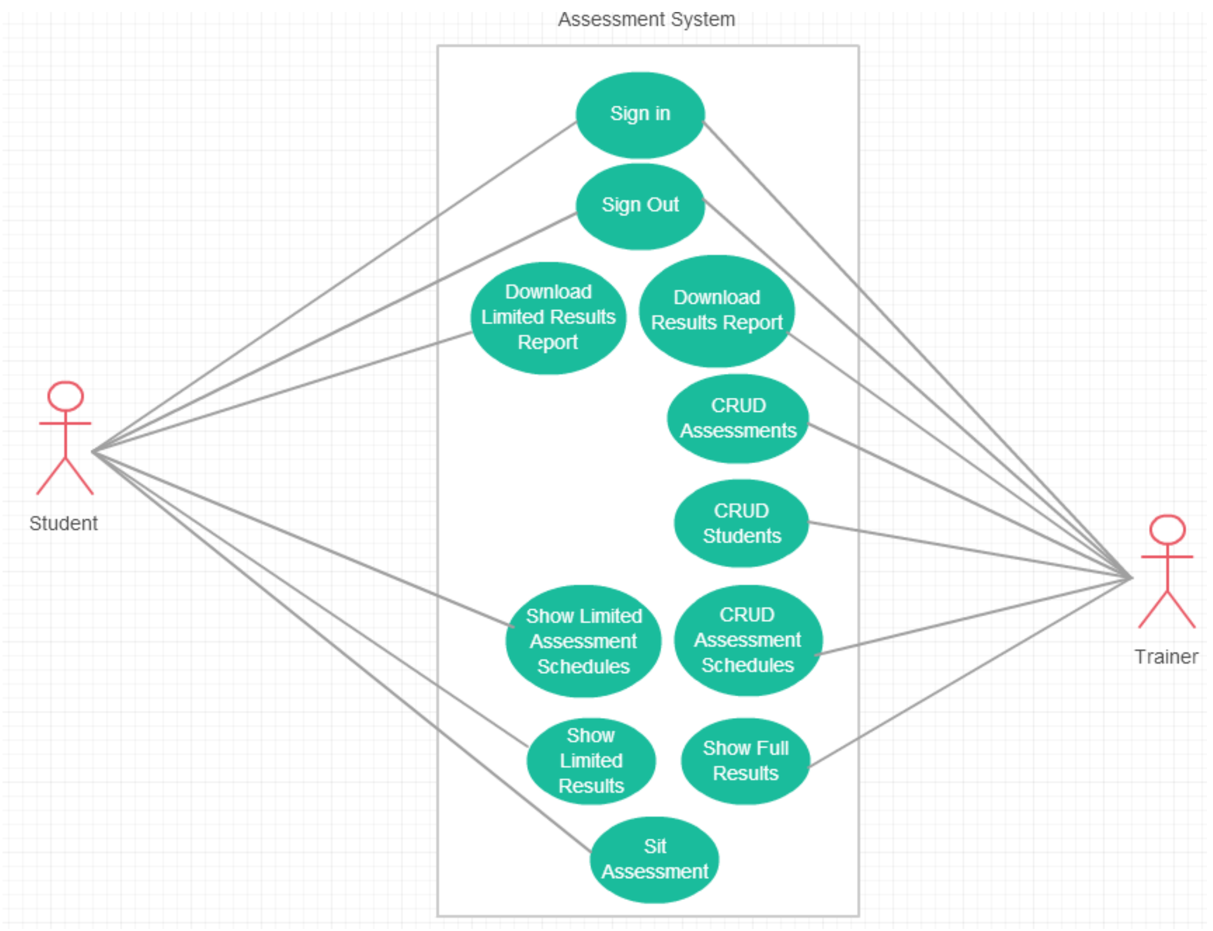


Figure 8: Use Case Diagram

Database Design

As the daily operations of assessment setting and marking are done entirely on paper, there is no existing data structure that can be used for this project. The following considerations and agreements with the client were made before I began designing the database:

- There are two types of users:
    - **Trainers**, which have:
        - First name, last name, username, password and created date
    - **Students**, which have:
        - First name, last name, username, password, created date and assessment results
- **Assessments**, which have:
    - Title, versions, created by (Trainer) and created date
- **Assessment Schedules**, which have:
    - Start date, end date, assessment (Assessment ID), version (version ID) , students and trainer
- There are multiple questions per version and assessments have multiple versions
    - Each question can be different types (Call log, free text, multiple choice)

Mostly due to the variability of the question types and how assessments can have versions (which can have multiple questions) I decided to look into alternatives to using a database with a strict schema. The data I require to store can't easily be structured into pre-defined 2-dimensional tables such as those used in SQL databases, for example, each assessment question and answer can have a different question type which means this question can have totally different properties to that of another question. SQL schemas won't easily allow for this as an explicit structure must be set.

An alternative to SQL is NoSQL which allows the storage of arbitrary data. However, when using NoSQL the structure must be known, maintained and controlled throughout the entire application in the form of a dynamic schema. Being that the only access requirement to the database is through the application itself, it is possible to safely persist arbitrary data from client to database and vice versa. Other factors must be considered when it comes to choosing NoSQL over SQL as there are some features of which we sometimes take for granted from SQL databases.

An example of a NoSQL database is MongoDB which stores its data as a collection of documents where each document is a standard JSON object. This is another benefit of using MongoDB in a web-based project as JSON which is based on a subset of JavaScript which is still growing as the prominent language on the web. The deciding factor of my choice to use MongoDB was to follow the MEAN development stack which is explained in the Service Layer Design section of this report.

Throughout my data it is also only really necessary to use an insert/update operation on a single document, for example:

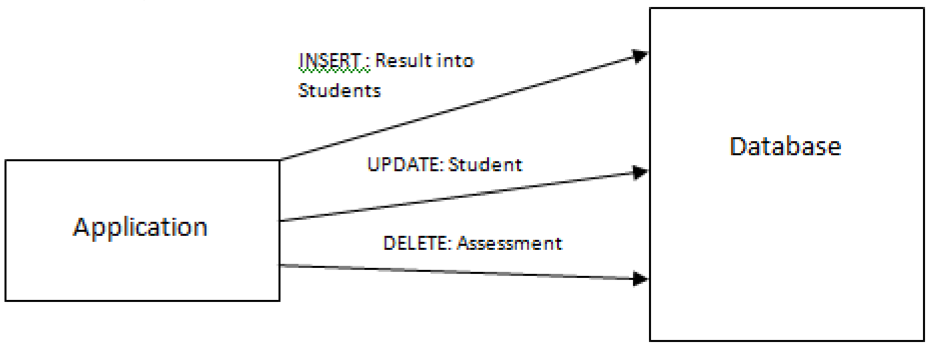


Figure 9: Database Operations

In MongoDB, these operations are atomic on the level of a single document (http://docs.mongodb.org/manual/tutorial/model-data-for-atomic-operations/) meaning that **all** of the write operations required for my application won't be affected by the atomicity that MongoDB lacks.

The usage scale of this application is also small enough that the lack of ACID properties won't be a problem as realistically, only one Admin will be authorised to make any assessment changes and a maximum of 20 Students in a test period will be able to submit their assessments in a small time scale for marking.

To get a better idea of the structure of the data using a document orientated approach I designed a schema with relationships (document references) to illustrate how arrays of objects can be pre-defined schemas (figure 10).

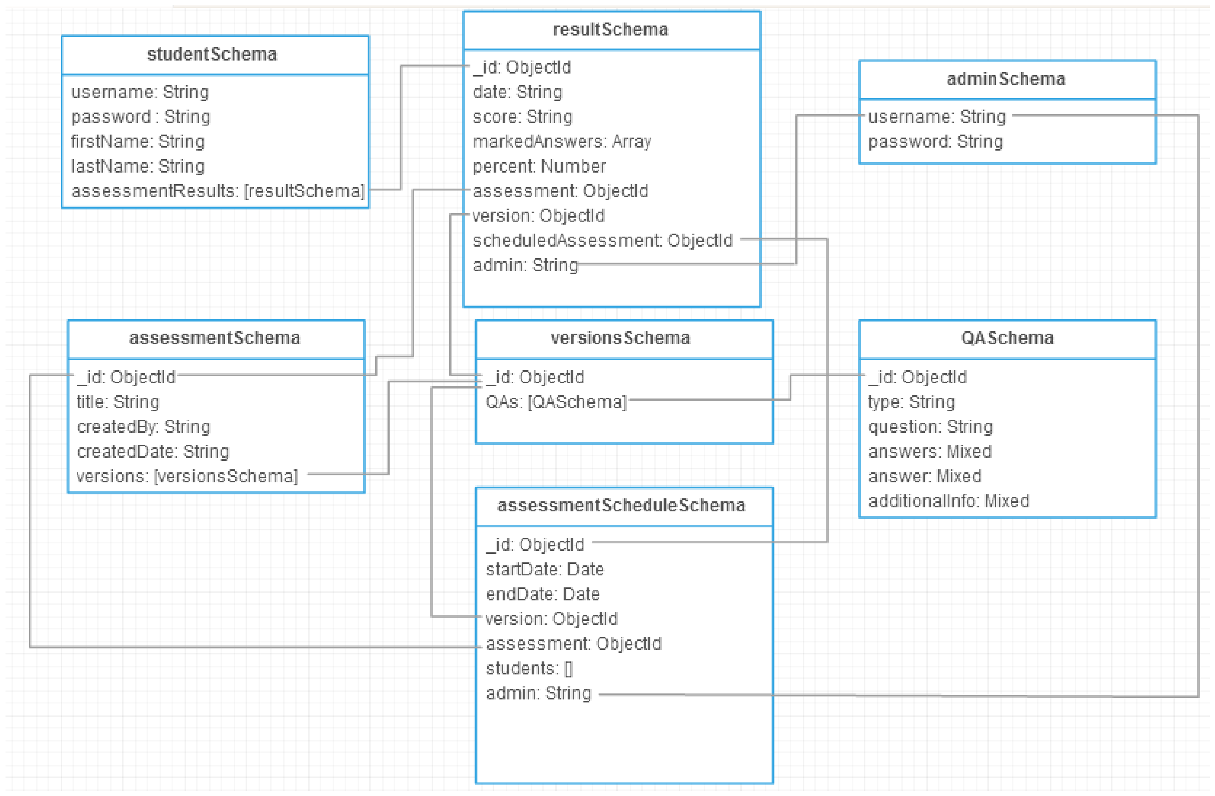


Figure 10: Database Schema

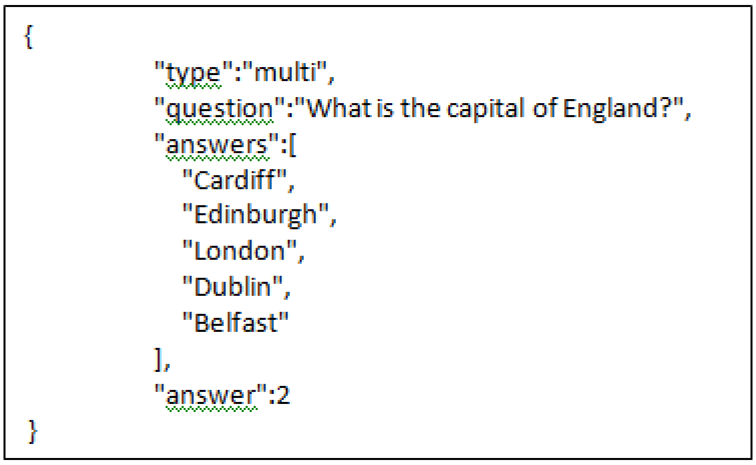
The most important things to note in this design are:

- The arrays of pre-defined schemas (i.e. [*versionsSchema*] – an array of versions)
- The Mixed data type that is available to use (i.e. for arbitrary answers to a question – *additionalInfo* is a mixed type that can be used for call logs!)
- The Array data type that is available to use (This is just an array of arbitrary objects can represent marked answers for an assessment result)

As for the question type objects, these will depend on the **type** of question:

- Multiple choice
- Free
- Call log

Call log will require more information than a multiple choice question and it is for this reason it is a good idea to select this arbitrary document store. A multiple choice question simply requires the question, answers (options) and the actual answer which could look something like this:



```
{
            "type":"multi",
            "question":"What is the capital of England?",
            "answers":[
              "Cardiff",
              "Edinburgh",
              "London",
              "Dublin",
              "Belfast"
            ],
            "answer":2
}
```

Figure 11: Multiple Choice Example JSON Object

A call log (and free text) is a different question and has to be stored differently to **make sense** and looks more like this (similar but requires other information for the user to read):

```
"type":"call",
          "additionalInfo": {
            "details": [
              {
                "date": new Date("2015-06-13T10:01:03.301Z"),
                "log": "Caller requests details of duty chemist as requires medication"
              },
              {
                "date": new Date("2015-06-13T10:05:00.001Z"),
                "log": "Checked NHS website - details obtained and passed to caller"
              },
              {
                "date": new Date("2015-06-14T10:06:00.001Z"),
                "log": "Incident for closure"
              }
            ],
            "background": {
              "uniqueID": 8130,
              "dateReceived": new Date("2011-06-13T10:00:00.001Z"),
              "response": "Resolved/referred",
              "openCodeCategory": "Administration",
              "dateRecorder": new Date("2015-06-13T10:01:00.001Z"),
              "contactMethod": "Non emergency call",
              "callerStatus": "Third Party",
              "callerName": "Joe Bloggs",
              "callerAddress": "121 Atlantic Hill, Imaginary City, AB1 2CD",
              "callerTelephone": "06302 445554",
              "incidentLocation": "121 Atlantic Hill, Imaginary City, AB1 2CD "
            }
          }
          "answers": [
            "Something",
            "SomethingElse",
            "SomethingElseAgain"
          ],
          "answer":2
```

Figure 12: Call Log Example JSON Object

Service Layer Design

Platform

The responsibility of the service layer is to take requests from the client and to interact with the database to respond to these requests. I have also mentioned that the services should be exposed as a RESTful interface adhering to these standards to enable scalability and simplicity for client applications. Due to the size of this project it is unrealistic to use a lower level programming language to implement services as there simply wasn't enough time to work with sockets and handle low level networking issues. Due to this there were five realistic language choices when considering the service layer:

- Java (JEE/Spring Data)
- PHP
- C#
- Python
- Node.js

Java, Python and C# are similar in that they are not primarily web languages and as such require other means of constructing web services (large libraries etc.). Java and C# are also strictly typed languages which was very attractive but Java takes too long to set up and re-build and I had no prior experience with C#. With further research I found an extremely useful framework called the MEAN stack that helped me select my implementation method. The MEAN stack comprises of:

- **M**ongoDB
  - A database which I have already discussed as being a good option to store the required data
- **E**xpress.js
  - A Node.js web application framework that fully supports implementations of REST API's
- **A**ngular.js
  - A JavaScript framework that assists in structuring JavaScript on the client which can quickly become messy without
- **N**ode.js
  - *"Node.js® is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices*."

The use of Node.js in a web environment alongside MongoDB results in a full JavaScript stack (client-server-database). Node.js is the fastest progressing programming platform I've ever seen and although mature, has support using the package manager that's built into it (NPM). JavaScript seems to be a fast growing technology with capability of full stack development and has been recognised as the most-used programming language on the Stack Overflow developer survey with mentions of Node.js and AngularJS:

- http://stackoverflow.com/research/developer-survey-2015#tech-lang

For all these reasons, alongside better integration with MongoDB and RESTful interfaces is the reason why I chose Node.js as a development platform over PHP and Python.

Separation of Operations

The service layer alone will be responsible for handling and responding to all create, read, update and delete (CRUD) operations, it is for this reason I decided to separate implementations to make it easier to navigate and understand. The full listings of the REST API can be found in this file (https://github.com/J-Morris676/SWPAssessment/blob/master/server.js) and is split up into three implementations:

- Data Access Services
  - Responsible for GET requests, examples:
    - /resources/schedules/:scheduleAssessmentId/students/:username/report
    - /resources/assessments/:assessmentId/versions/:versionId
- Data Upload Services
  - Responsible for upload/update/authentication requests, examples:
    - /auth/login
    - /resources/schedule
- Data Delete Services
  - Responsible for upload/update/authentication requests, examples:
    - /resources/schedules/:scheduleId
    - /resources/admins/:username

Designing the service layer in this way by dividing the services up into these three areas helps when implementing further features and debugging any issues that occur.

Authentication

The service layer is also responsible for the authentication of users and as there are two levels of authentication a design was required to understand what level of access a user should be granted.

The passwords in the database are stored using a cryptographic hash function called MD5 which is outputs a 16-byte hexadecimal value that represents a standard string, an example of this is demonstrated in figure 13.

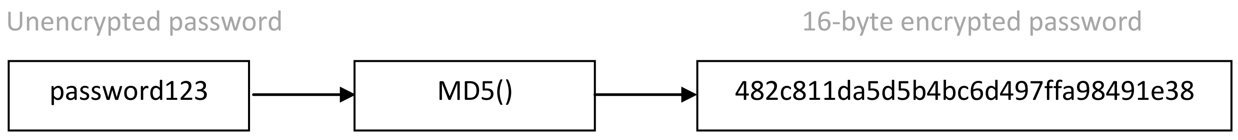


Figure 13: Password Encryption Example

A simple query is then used on the database with the given username and password to ensure correct credentials shown in figure 14.

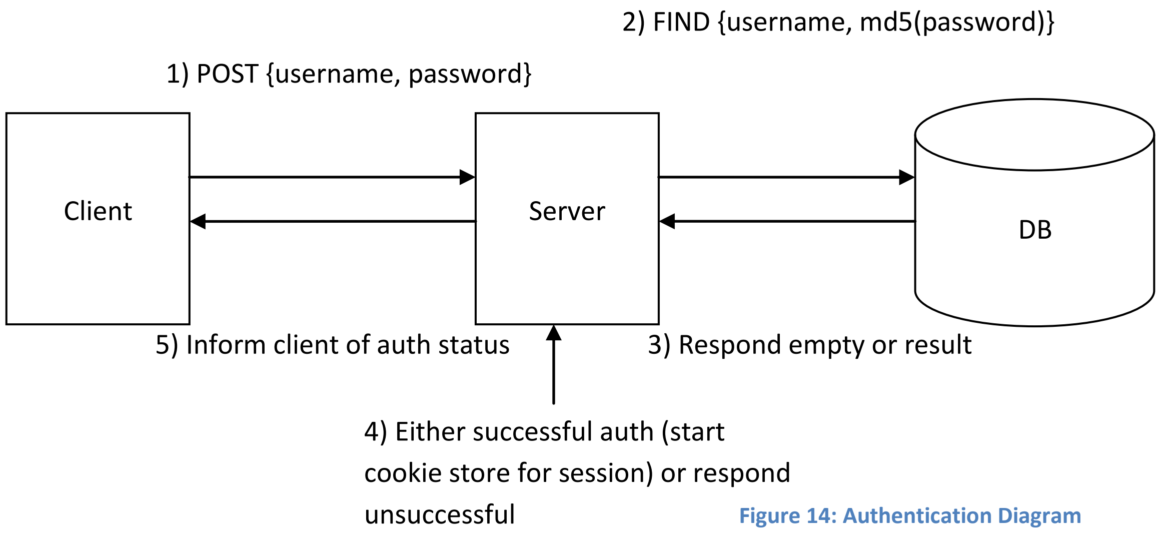

2) FIND {username, md5(password)}
1) POST {username, password}
Client
Server
DB
5) Inform client of auth status
3) Respond empty or result
4) Either successful auth (start cookie store for session) or respond unsuccessful

Figure 14: Authentication Diagram

The application uses cookies for sessions after a user is authorised so the user doesn't need to provide credentials again. In this model confidentiality and integrity aren't considered but are protected due to the fact it will be deployed internally to South Wales Police and it was decided SSL/TLS for demonstration was overkill due to the fact it wouldn't be used in an actual implementation.

UI Design

Due to the two user roles of the system the application has a login page and two main sections – one for Trainers and one for Students where the overall design and feel is the same and the application will determine whether they are a Trainer or a Student and direct them accordingly.

Corporate Colours

The Client would like the system to utilise corporate colours often found throughout their PowerPoint presentation:

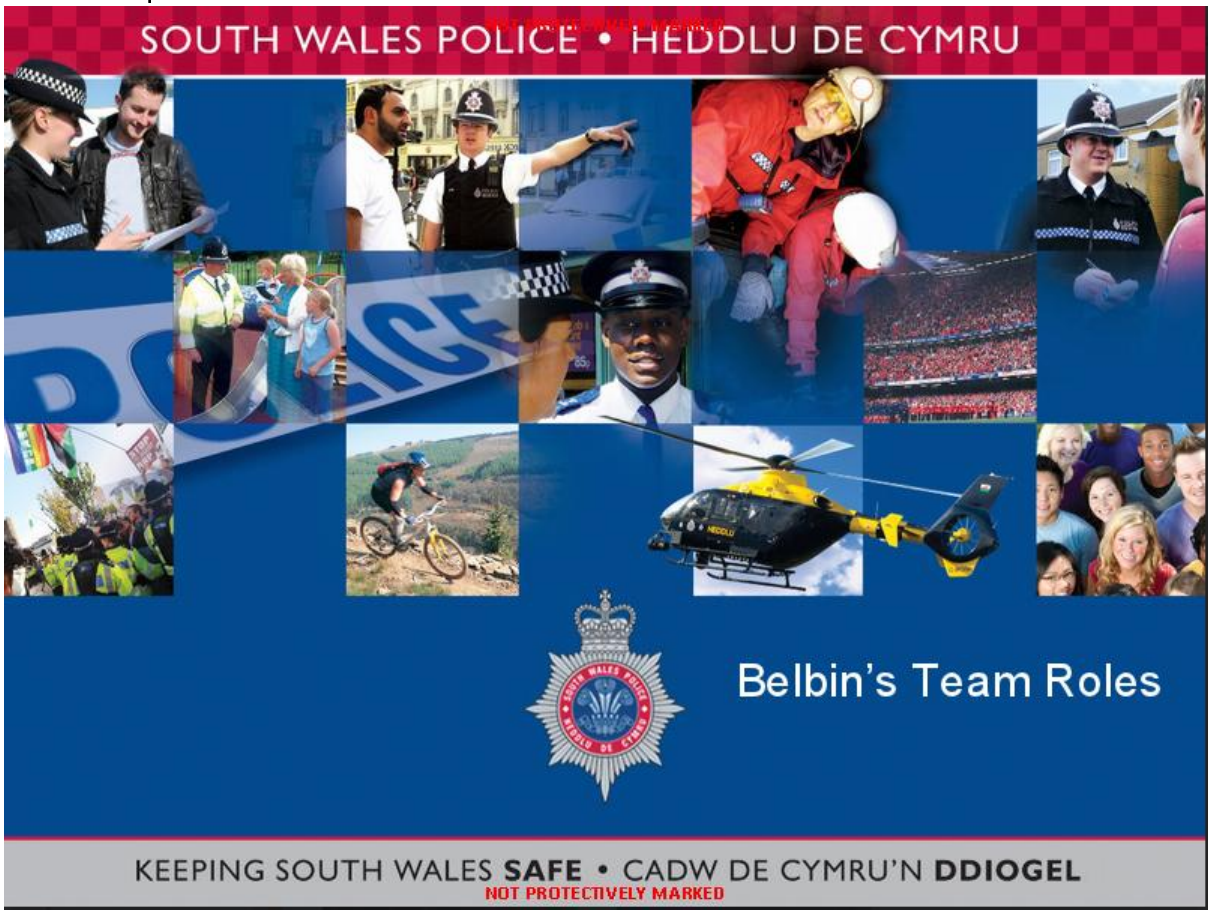

NOT PROTECTIVELY MARKED
SOUTH WALES POLICE • HEDDLU DE CYMRU
Belbin's Team Roles
KEEPING SOUTH WALES SAFE • CADW DE CYMRU'N DDIOGEL
NOT PROTECTIVELY MARKED

Figure 15: Example Corporate PowerPoint Slide

This is the colour scheme that is used for the entire application and I have taken colours and sections out of this image to ensure they are the same.

I have extracted the banner at the top displaying the Battenberg design as an SVG for a lightweight retrieval for the Client application and is used for the navigation bar. The code and image for this SVG can be found here:

- https://github.com/J-Morris676/SWPAssessment/blob/master/public/img/SWPBlocks.svg

I have also utilised this slide for use with the downloadable results report mentioned in the implementation section of this report.

The grey colour at the bottom has been utilised by panels and some buttons in the application and other buttons utilise the blue colour seen on the centre of the PowerPoint slide.

UI Framework

Due to the scale of this project I decided it was best to utilise a front-end, open source framework that supports responsive design as this resulted in a UI that was user friendly and quick to set up, which was ideal for this project. I made this decision because designing and implementing CSS libraries bespoke to an application that are user friendly and fully responsive can be a very time consuming process. The most obvious use of this on the modern web is Twitter Bootstrap but there are others that I considered for use:

- HTML KickStart
- Kickstrap
- Pure
- HTML5 Boilerplate

One benefit to using some over others is the integration with Angular.js (A in MEAN) as some frameworks/libraries are fully integrated as Angular modules/directives which is extremely useful for ensuring my view and control relationship is clean. Examples of these include angular-strap and angular-ui for Twitter Bootstrap components and is the reason why I eventually decided on Twitter Bootstrap over its competitors among these other reasons too:

- Right level of complexity and control (e.g. HTML5 Boilerplate was just for templates)
- Unrivalled documentation and support (http://getbootstrap.com/getting-started/)
- Form builders using drag and drop functionality have been implemented for Bootstrap components making it extremely quick to build forms (http://bootsnipp.com/forms?version=3)
- It's built mobile first meaning all of its features are responsive

System Layout

As I was working with a Client on this project, I had to consider ideas that would reduce the chances of a misunderstanding in meetings as this can be very time consuming and unnecessary. To help prevent this, I brought a small whiteboard, pen and paper to each meeting so we could not just discuss design ideas but visualise them too.

The first example I have of these is for the home page and as seen below, I sketched out rough designs based on the requirements that were set out and we fine-tuned them together in meetings. Doing this allowed us to mutually agree on a design before any form of implementation went ahead and this saved time.

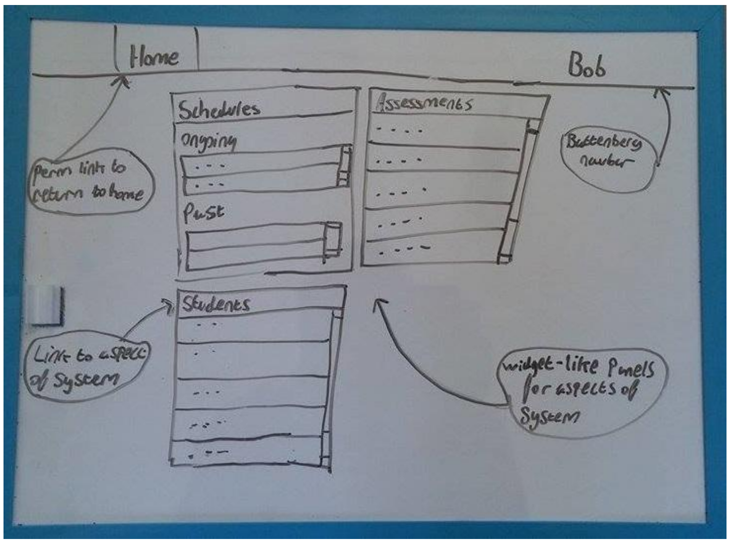
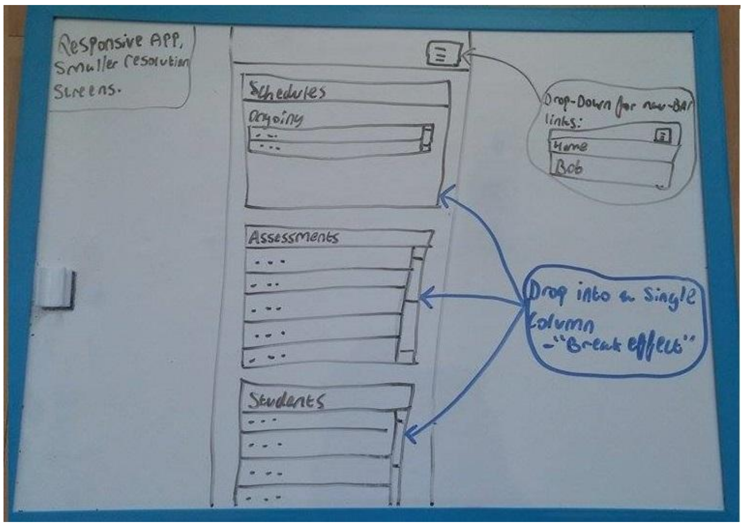


Figure 16: Whiteboard Homepage Design

As the Client requested the application be usable on smaller resolution devices, I also had to consider how these features would look on a mobile device. Below is another example of a drawing showing how the application will look with responsive features provided by the library framework that I decided to use to implement it which utilises row breaks.

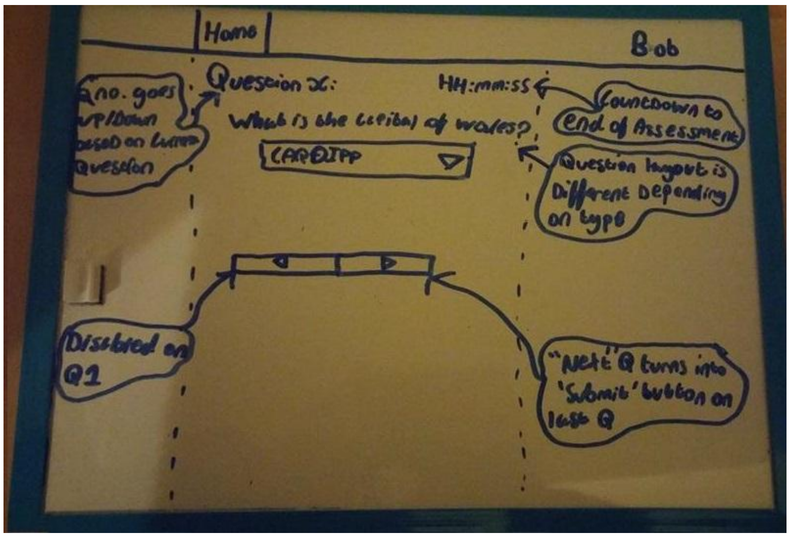


Responsive APP, Smaller resolution Screens.
Schedules
Ongoing
Assessments
Students
Drop-Down for nav-Bar links:
Home
Bob
Drop into a Single Column -"Break effect"

Figure 17: Whiteboard Mobile Homepage Design

Another important example of this was when the sitting of assessments section of the system was being designed. This page is interesting as it's state changes frequently:

- Countdown to end of assessment
- Disabled buttons based on current question number
- Submit button appears on last question



Home
Bob
Q no. goes up/Down based on current Question
Question x:
HH:mm:ss
What is the capital of wales?
CARDIFF
Countdown to end of Assessment
Question layout is Different depending on type
Disabled on Q1
"Next Q turns into 'Submit' button on last Q

Figure 18: Whiteboard Sit Assessment Design

These sketches were primarily for me so that I could get a good idea as to what the Client wanted and implement it directly from my sketch design at a later date. More examples of these sketches can be found in the appendix of this report.

Throughout the design of the system I utilised Jakob Nielsen's 10 usability heuristics to minimise the amount of usability issue's that may appear when performing usability tests, during the implementation section of this report I have included justifications for my layout choices based on a usability heuristic.

Implementation

Development Tools

Integrated Development Environment and Server Debugging

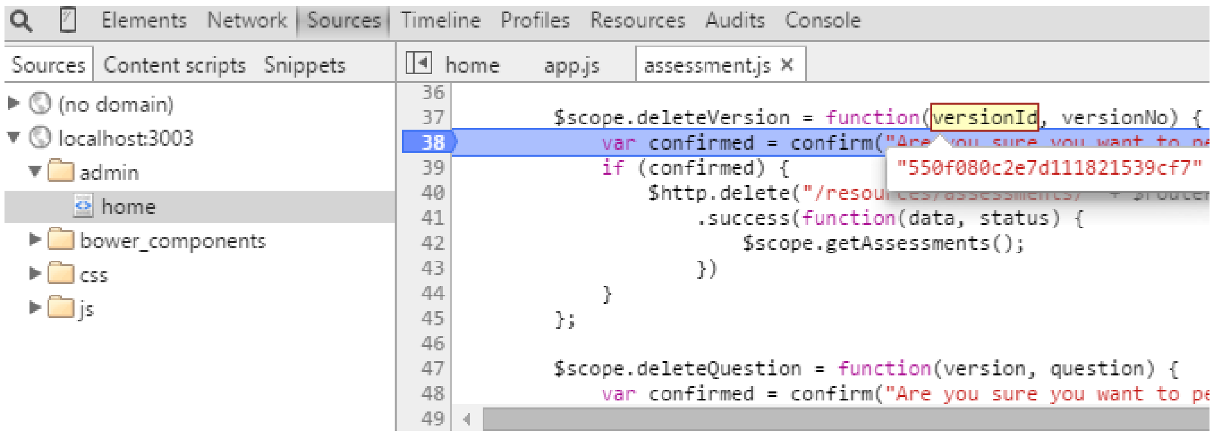
To make debugging and file traversal as simple and as quick as possible I decided to use an Integrated Development Environment (IDE). This decision of IDE was based off a simple Google search where I found IntelliJ are the leading light in commercial IDE products for web languages (PHP, Ruby, Python etc.). With knowledge of their IntelliJ IDEA IDE (Java) I decided to use WebStorm (best suited for JavaScript client & server side development) on a Student license.

Database Management

MongoDB does provide a JavaScript shell that can be used to query the database but it's often difficult to view and compare documents within the collections, I therefore decided to use Robomongo which is fairly similar to Oracle SQLDeveloper as it allows you to perform all standard operations using a GUI and using a query window, Robomongo has a GNU license.

Client Debugging and Network Analysis

To enable step-by-step code debugging for the client I primarily used the browsers that this application was intended for (Chrome & IE11), the tools provided by both of these browsers allowed me to analyse the state of the system at step-by-step stages. I also utilised the network feature provided by these browsers to monitor responses and timings of HTTP requests.


Elements Network Sources Timeline Profiles Resources Audits Console
Sources Content scripts Snippets
home app.js assessment.js
(no domain)
localhost:3003
admin
home
bower_components
css
js

```
    $scope.deleteVersion = function(versionId, versionNo) {
        var confirmed = confirm("Are
        if (confirmed) {
            $http.delete("/resou
                .success(function(data, status) {
                    $scope.getAssessments();
                })
        }
    };

    $scope.deleteQuestion = function(version, question) {
        var confirmed = confirm("Are you sure you want to pe
```

```
"550f080c2e7d111821539cf7"
```

Figure 19: Google Chrome Developer Tools

Device Mode Tool

A recent update of Google Chrome provides a method for scaling the view to see what a web application will look like on a variety of devices, this was particularly useful when ensuring the features were responsive by tweaking the mark-up based on the view at different resolutions

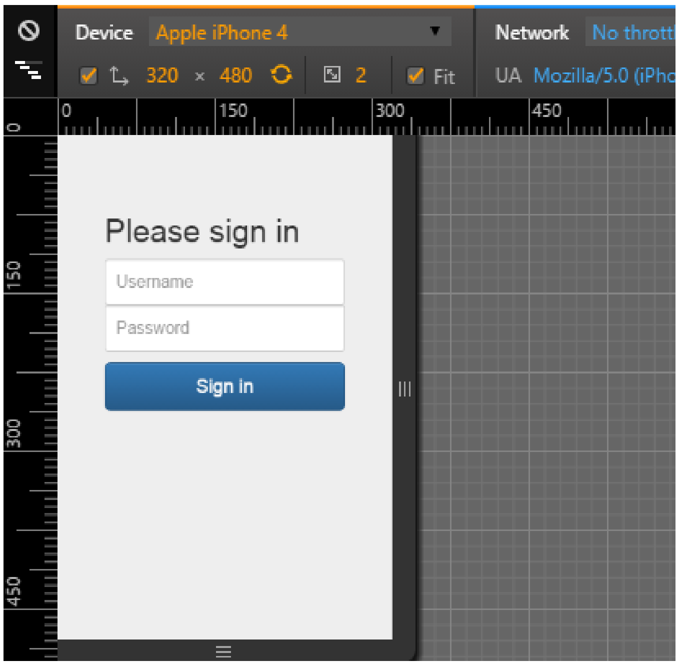
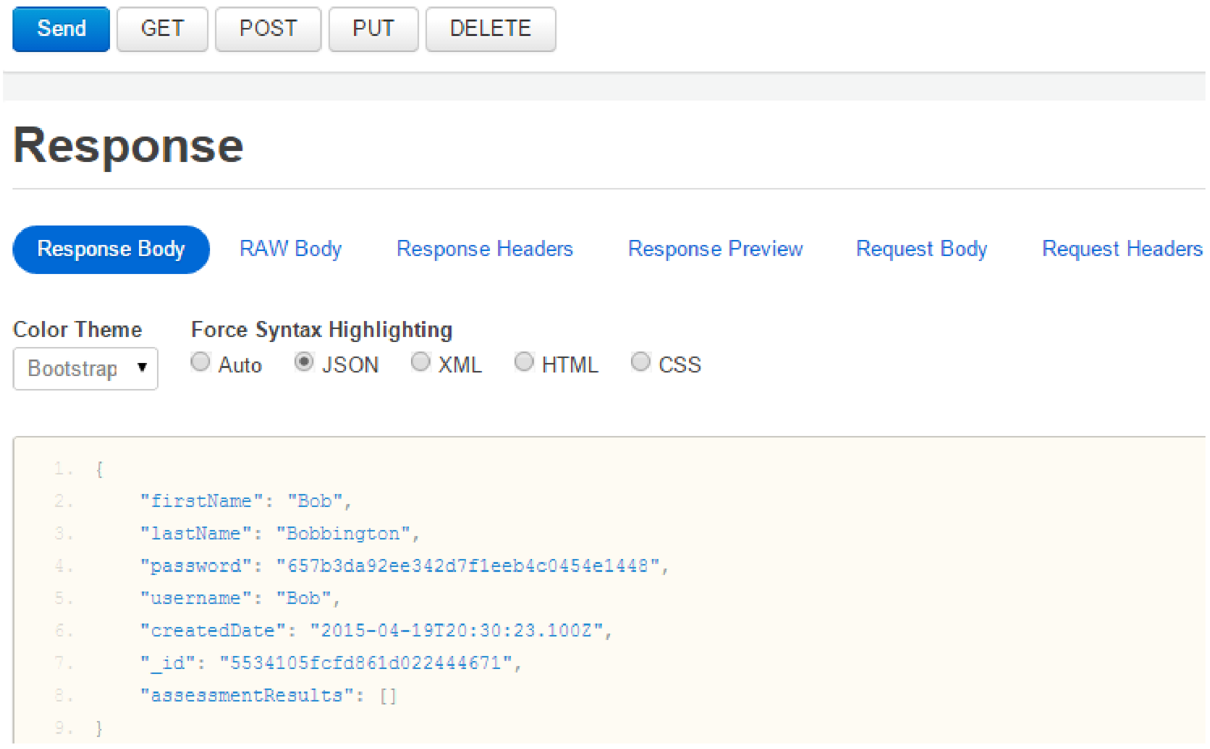


Figure 20: Google Chrome Responsive Mode

API HTTP Testing

To test the functionality of the API whilst constructing the services I used a Google Chrome extension called 'REST console' (MIT license) which allows you to easily send HTTP requests to a server using a graphical interface. This was particularly useful when needing to POST particular resources to the server (e.g. a JSON object containing assessment information).



Color Theme: Bootstrap

Force Syntax Highlighting: Auto [✓ JSON] XML HTML CSS

```
{
    "firstName": "Bob",
    "lastName": "Bobbington",
    "password": "657b3da92ee342d7f1eeb4c0454e1448",
    "username": "Bob",
    "createdDate": "2015-04-19T20:30:23.100Z",
    "_id": "5534105fcfd861d022444671",
    "assessmentResults": []
}
```

Figure 21: REST Console

Resolving and Handling Dependencies

Due to the size of the project and the limited time I had available the use of third party libraries was something I had to take advantage of to reduce the amount of time taken to implement system requirements. In fact, the system (server through to client) relies on a total of 30 dependencies so it was extremely important that I had some way of maintaining all of these. Without the use of these dependency handlers it can get very messy very quickly as dependencies all exist in different places with different versions:

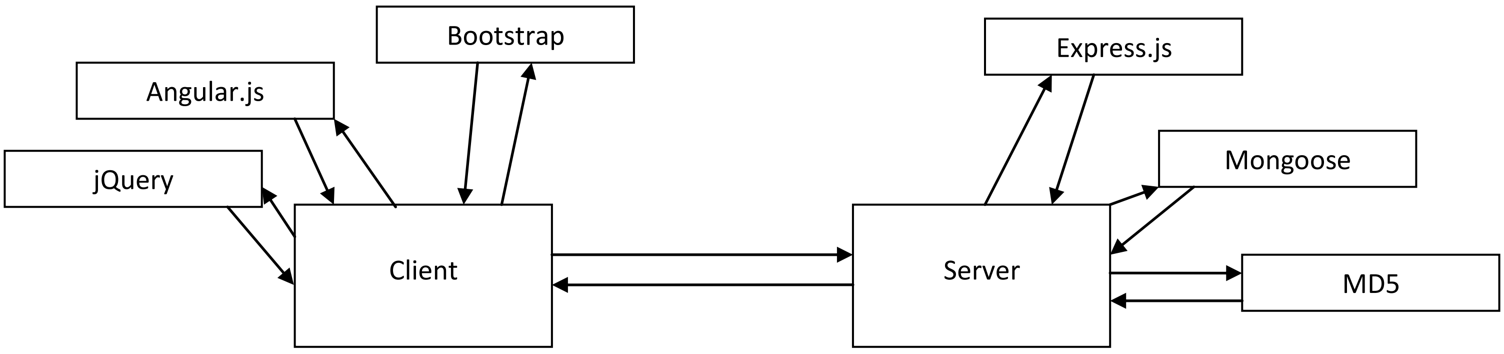


Figure 22: Dependency Diagram Pre-Management

As illustrated by the above diagram, it becomes obvious to see how quickly this can easily get out of control when a system has a large amount of dependencies, with the use of dependency handlers it becomes more simple:

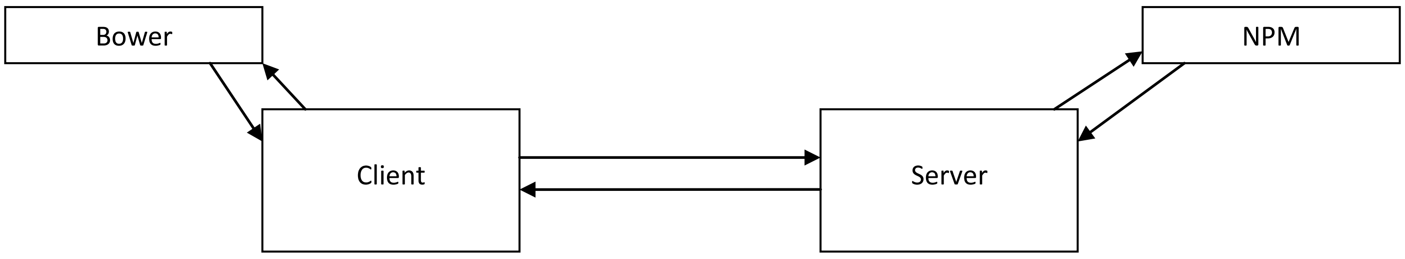


Figure 23: Dependency Diagram Post-Management

The above diagram now shows both components only have to interact with a single package manager server where dependencies for both are defined in a single file.

Server Dependencies

As mentioned in the design section of this report, Node.js has a built in package manager called NPM (Node Package Manager). This made it as simple as just defining a file in the project that describes the system being built and a list of all of the NPM packages it depends on. With a little help from an NPM tool called 'npm-install-missing' all dependencies within this file can be resolved from a single command. The package file for this project can be found at the below link:

- https://github.com/J-Morris676/SWPAssessment/blob/master/package.json

Client Dependencies

The client design was similar in some ways to the server in that it requires rich frameworks and libraries to really bring it to life. To handle client dependencies a third party package manager was required called Bower that:

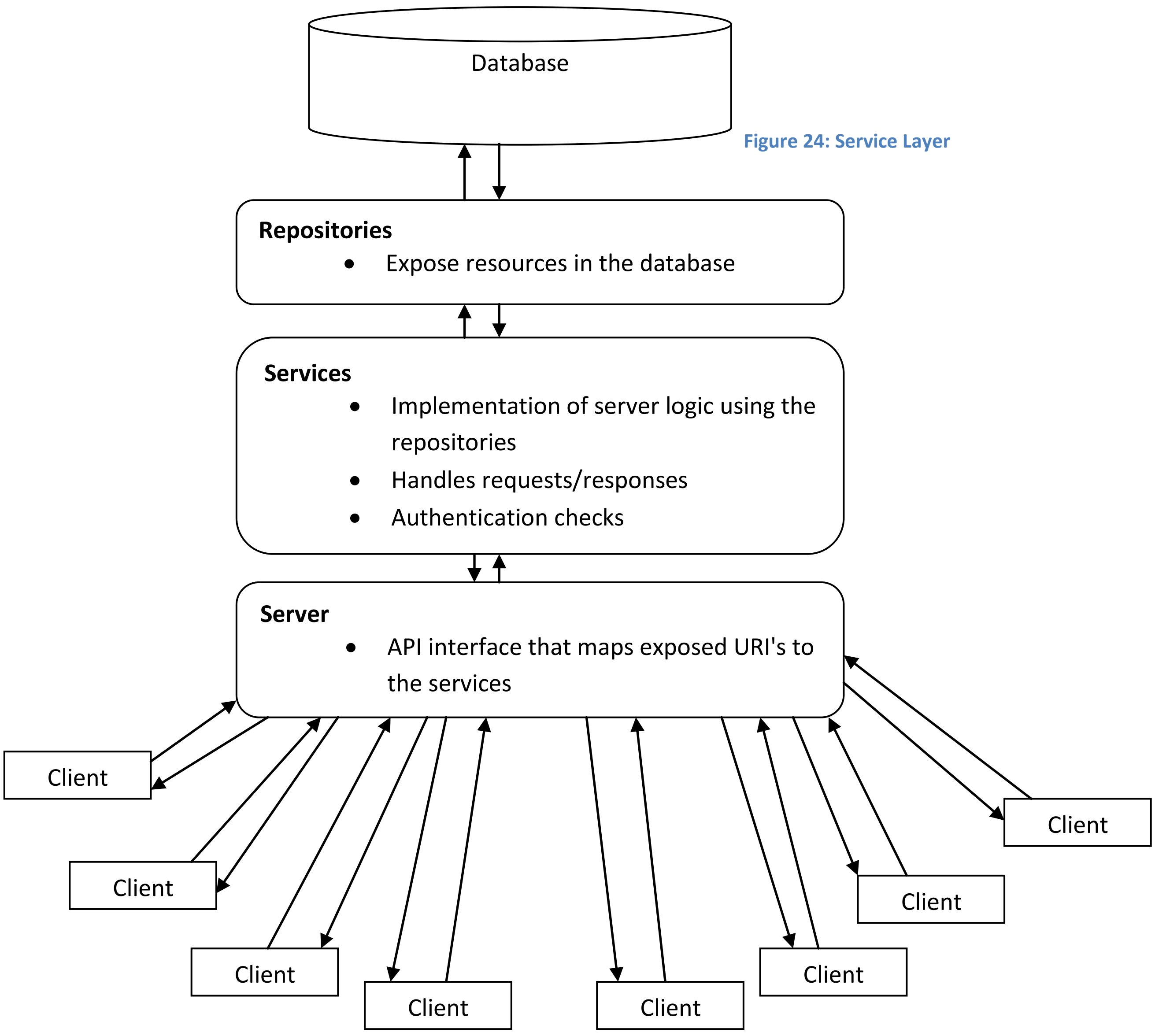
> *"Takes care of hunting, finding, downloading, and saving the stuff you're looking for"*
> - *http://bower.io/*

Bower works very similarly to NPM with npm-install-missing and all of these client dependencies are defined in a file called 'bower.json' which can be found at the below link:

- https://github.com/J-Morris676/SWPAssessment/blob/master/public/bower.json

Service Layer

The service layer utilises abstraction to make it as simple as possible to understand and navigate when it needs to be extended, the server is divided into three layers:



Database
Repositories
Expose resources in the database
Services
Implementation of server logic using the repositories
Handles requests/responses
Authentication checks
Server
API interface that maps exposed URI's to the services
Client
Client
Client
Client
Client
Client
Client
Client
Client

Figure 24: Service Layer

Database Interaction - Repositories

Node.js and MongoDB are known to work well together and it was very quick to setup a method of querying the database, my solution to this involves the use of Mongoose which allows for object modelling and was written because:

> *"writing MongoDB validation, casting and business logic boilerplate is a drag"*
> - *http://mongoosejs.com/*

The use of Mongoose allowed me to define strict or loose schemas depending on the particular resource in the database. Schemas are really simple to define and they take an object containing the schema of the MongoDB collection, for example:

```
var studentSchema = new Schema({
   "username": String,
   "password": String,
   "firstName": String,
   "lastName": String,
   "createdDate": Date,
   "assessmentResults": [resultSchema]
}, { versionKey: false });
```

Figure 25: Student Schema

Note: These exact schemas have already been seen in the design section of this report.

This above code defines a schema for the student collection where the types are standard JavaScript data types or other schemas, for example:

- **username** is a standard JavaScript String
- **createdDate** is a standard JavaScript String
- **assessmentResults** is an array of resultSchema documents

These schemas are then modelled directly to a collection within the database Mongoose is connected to. The full listing of these schemas and the database connection can be found here:

- https://github.com/J-Morris676/SWPAssessment/blob/master/repositories/database-connection.js

To provide an interface to the service layer which is responsible for handling HTTP requests I have divided the MongoDB query implementations into 3 different files:

- data-access-repository
- data-delete-repository
- data-upload-repository

The sole purpose of these is to act as repositories to the service layer, exposing functions that will perform database operations (and any other transformations that were required). Doing this helped with scaling the application up as it was easy to tell where to implement further features. Doing so has no effect on any other repository feature, a basic example of a repository function can be seen below:

```
exports.findStudentById = function(id, cb) {
   databaseConnection.students.findOne({"_id": id}, {password: 0}, function(err, data)
{repositoryCallback(err,data,cb);});
};
```

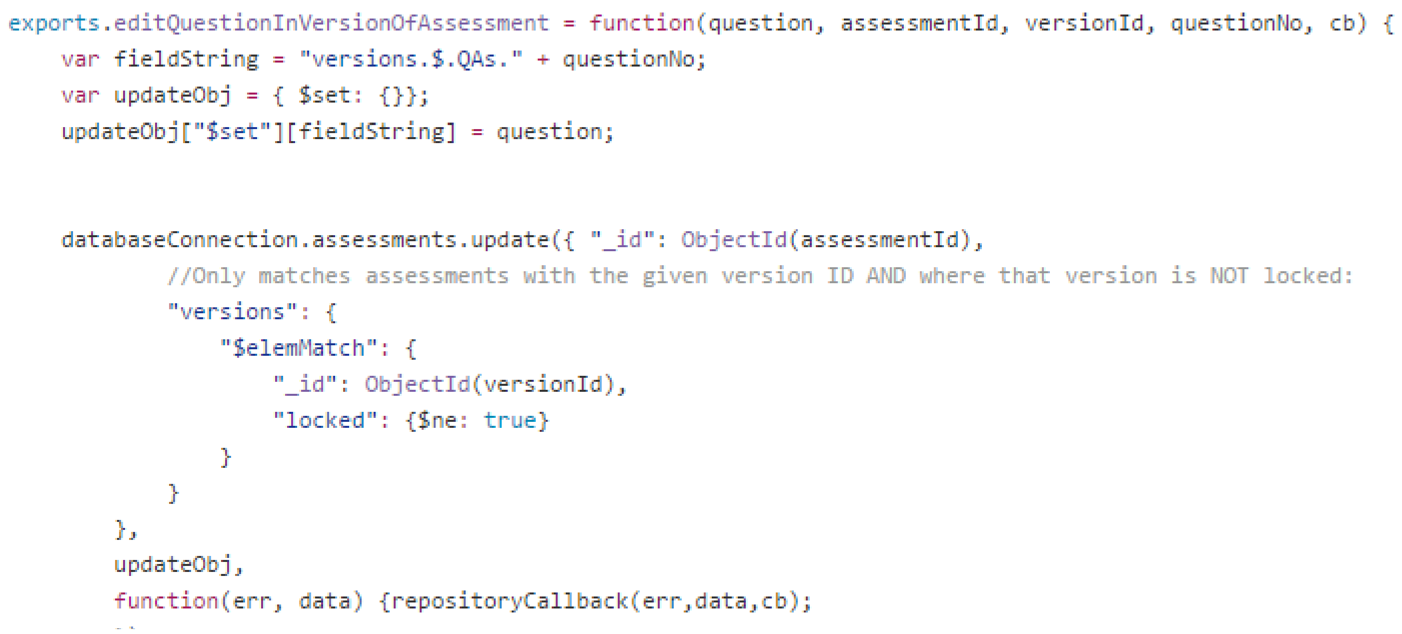
Figure 26: Find Student Repository Function

In this function the below MongoDB query can be seen:

```
db.student.findOne({"_id": id}, {password: 0});
```

This particular example retrieves the student with the given ID and uses projection (second parameter) to prevent retrieving the password (As it should not be retrievable!)

A more complex example can be seen in figure 27 which edits a question in a given version of an assessment.



```
exports.editQuestionInVersionOfAssessment = function(question, assessmentId, versionId, questionNo, cb) {
    var fieldString = "versions.$.QAs." + questionNo;
    var updateObj = { $set: {}};
    updateObj["$set"][fieldString] = question;


    databaseConnection.assessments.update({ "_id": ObjectId(assessmentId),
            //Only matches assessments with the given version ID AND where that version is NOT locked:
            "versions": {
                "$elemMatch": {
                    "_id": ObjectId(versionId),
                    "locked": {$ne: true}
                }
            }
        },
        updateObj,
        function(err, data) {repositoryCallback(err,data,cb);
```

Figure 27: Edit Question Repository Function

The function in figure 27 utilises the below Mongo query (Assuming questionNo = 1):

-
  ```
  db.assessments.update({"_id": ObjectId(assessmentId),
                      "versions": {
                              "$elemMatch": {
                                      "_id": ObjectId(versionId),
                                      "locked": {$ne: true}
                              }
                      }
              },
              {
                      "$set": {
                              "versions.$.QAs.1": question
                      }
              }
  })
  ```

This query updates the contents of all documents that:

- Match the given assessment ID
- Version array contains an object where the ID is the given version ID and the 'locked' field is **not** true

The update parameter is given a MongoDB $set operator to edit the contents of the question in the found version ('$' matches the index of the object found with $elemMatch):

- "versions.$.QAs.1" now becomes whatever the given question object contains.

Due to the level of nesting for assessments (shown below), the queries can become fairly complex which is why I have left inline comments in the code, so it can be easily understood when revisited.

- Assessments
    - Versions
        - Questions

To fulfil the requirements of the system I have had to look into a wide variety of MongoDB querying methods including the use of the following useful operators:

- $push: Inserts a new document into an array
- $pull: Remove matching document from array
- $unset: Removes a field from a document
- $set: Updates an object in a document
- $ne, $gt, $lt: Standard Mathematical operations - work on dates too. (not equal, greater than etc.)
- $elemMatch: Matches objects with the given object
- $size: Match given array index size

The system also utilises the more modern MongoDB aggregation framework (http://docs.mongodb.org/manual/aggregation/), here is an example aggregate query from the data-access-repository:

-
  ```
  db.assessmentSchedule.aggregate([
      {$unwind : "$students" },
      {
        $match: {
          _id: scheduledAssessmentId,
          "students.username":  studentUsername,
          "students.dates.endDate": {"$exists": true}
        }
      }
    ])
  ```

This query enabled me to determine whether the given Student (studentUsername) in the scheduled assessment (scheduledAssessmentId) has finished the scheduled assessment. It does this by *unwinding* the students array within each assessment schedule resulting in a larger collection of assessment schedule documents with each of their student array objects as an object.

Example

This:

```
[
        {"_id": ObjectId("55131c38ff6bbe6018eb5d36"), students: [
                {"name": "Bob"},
                 {"name": "Bill"}
        ]}
]
```

$unwinds to this:

```
[
        {"_id": ObjectId("55131c38ff6bbe6018eb5d36"), students: {"name": "Bob"} },
        {"_id": ObjectId("55131c38ff6bbe6018eb5d36"), students: {"name": "Bill"} }
]
```

This is useful because matches made **after** the transformation can query on **individual** Students in an array index.

The full listings for all of these repository functions and the queries within them can be found in the following files:

- https://github.com/J-Morris676/SWPAssessment/blob/master/repositories/data-access-repository.js
- https://github.com/J-Morris676/SWPAssessment/blob/master/repositories/data-delete-repository.js
- https://github.com/J-Morris676/SWPAssessment/blob/master/repositories/data-upload-repository.js

Service Handlers - Services

The services handle the requests and responses received by the server and contain the logic that ultimately determines what a response will be, they have three main responsibilities:

- Check user authentication level
- Interacting with the repositories to access data via queries
- Apply application level logic

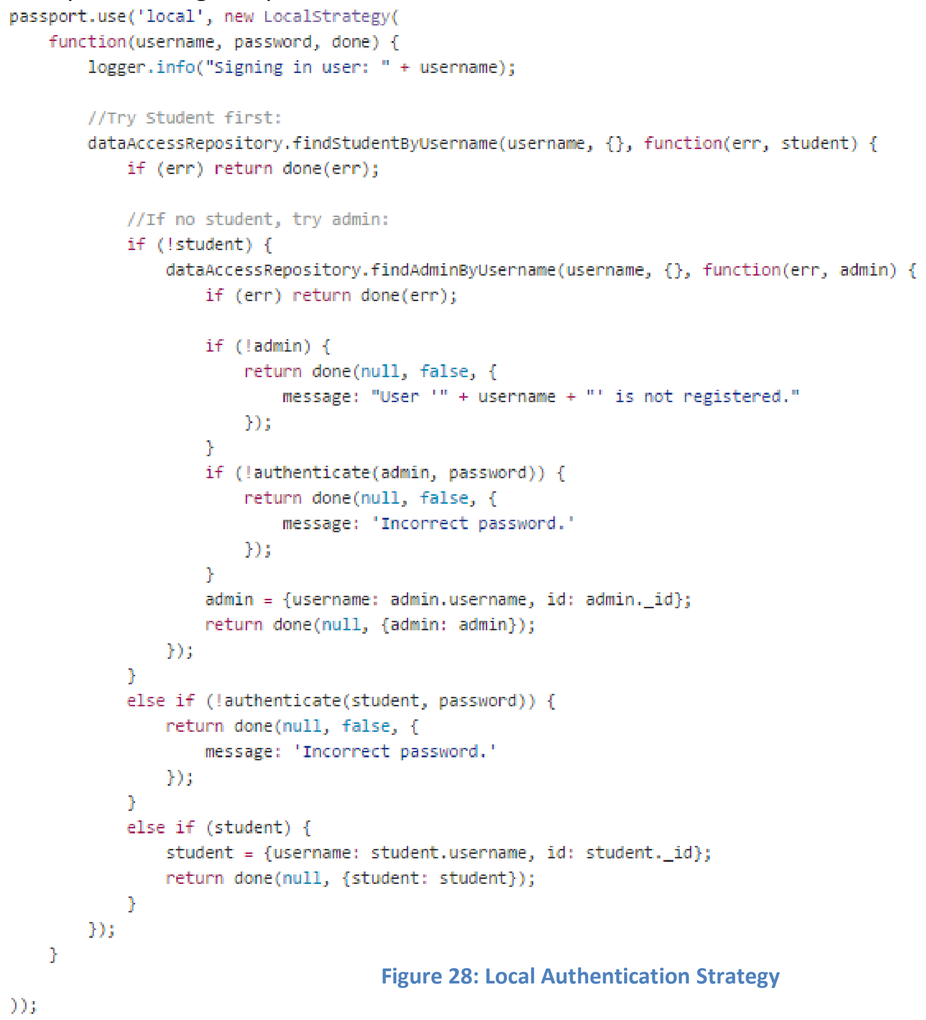
As mentioned in the design section I decided to take advantage of the Express framework to construct the API allowing me to simply define routes and the services that map to them, each Express service takes a request (req) object and a response (res) object.

- req contains information like URI parameters, HTTP body content, HTTP headers
- res contains methods of setting HTTP headers, statuses and functions to respond to the request.

Authentication

To satisfy the authentication requirements the server has to ensure whether the user requesting a resource is authorised to do so. This is handled by using sessions and requires a user to sign in before accessing resources. Node.js has a package called Passport.js and authentication is achieved using this package by utilising authentication strategies which take a username and password and returns whether the authentication is successful or not. As stated in the database design the username and password of users are stored in the Student and Admin collections and a query is made to check whether any matches are made.

NB: the authenticate function seen in the below snippet just checks whether the given user's password is equal to the given password



```
passport.use('local', new LocalStrategy(
    function(username, password, done) {
        logger.info("Signing in user: " + username);

        //Try Student first:
        dataAccessRepository.findStudentByUsername(username, {}, function(err, student) {
            if (err) return done(err);

            //If no student, try admin:
            if (!student) {
                dataAccessRepository.findAdminByUsername(username, {}, function(err, admin) {
                    if (err) return done(err);

                    if (!admin) {
                        return done(null, false, {
                            message: "User '" + username + "' is not registered."
                        });
                    }
                    if (!authenticate(admin, password)) {
                        return done(null, false, {
                            message: 'Incorrect password.'
                        });
                    }
                    admin = {username: admin.username, id: admin._id};
                    return done(null, {admin: admin});
                });
            }
            else if (!authenticate(student, password)) {
                return done(null, false, {
                    message: 'Incorrect password.'
                });
            }
            else if (student) {
                student = {username: student.username, id: student._id};
                return done(null, {student: student});
            }
        });
    }
));
```

Figure 28: Local Authentication Strategy

The above strategy follows the following logic to determine whether a session can be set up or not:

1. Attempt to find username in Student collection
    - If no Student is found, then repeat from step 1 for the Admin collection
    - If Student was found but fails to authenticate (users password is **not equal** to provided password) then return with incorrect password message
    - If Student was found and successfully authenticates (users password is **equal** to provided password) then set up the session object that'll be used to identify the user attempting to access resources

Full Passport strategy implementation can be found here:

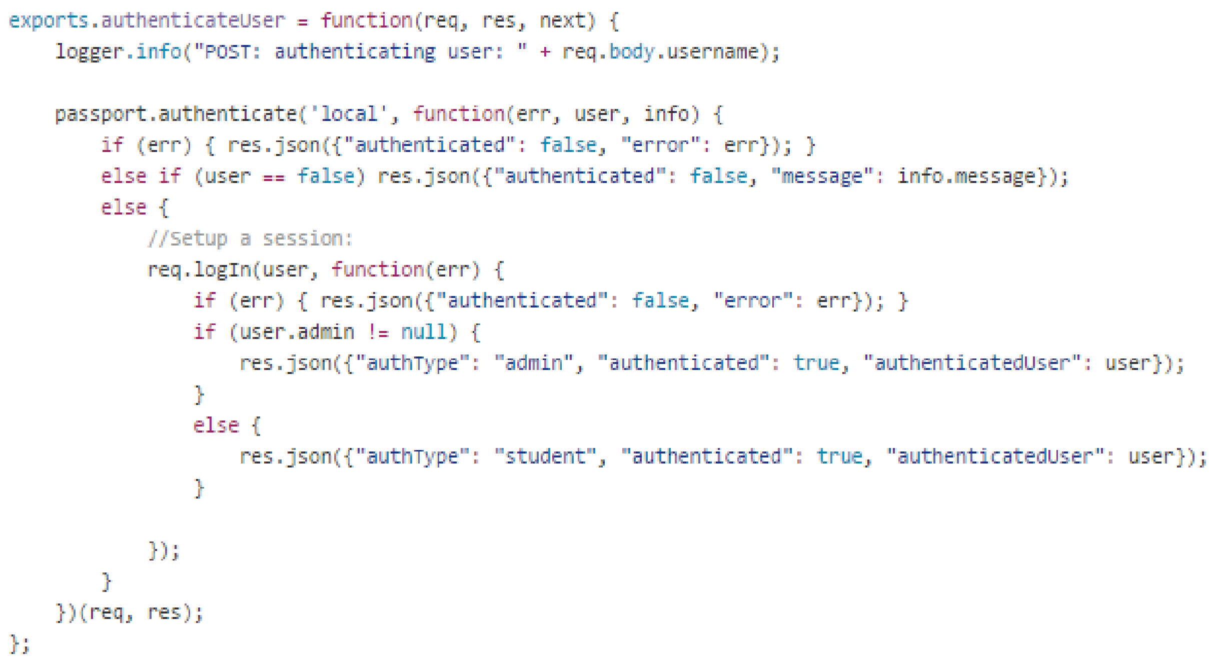
- https://github.com/J-Morris676/SWPAssessment/blob/master/authentication/local-strategies.js

When a user is successfully authenticated an object is created that will be used by each service to identify a signed in user, below is an example session object for a student:

```
{
        "student": {
                "username": "Bob",
                "id": "551d9634841c93c41c52cd67"
        }
}
```

Figure 29: Authentication Object

This strategy can then be implemented by a service to set up an Express session if the authentication was successful outlined in the *authenticateUser* service snippet below:

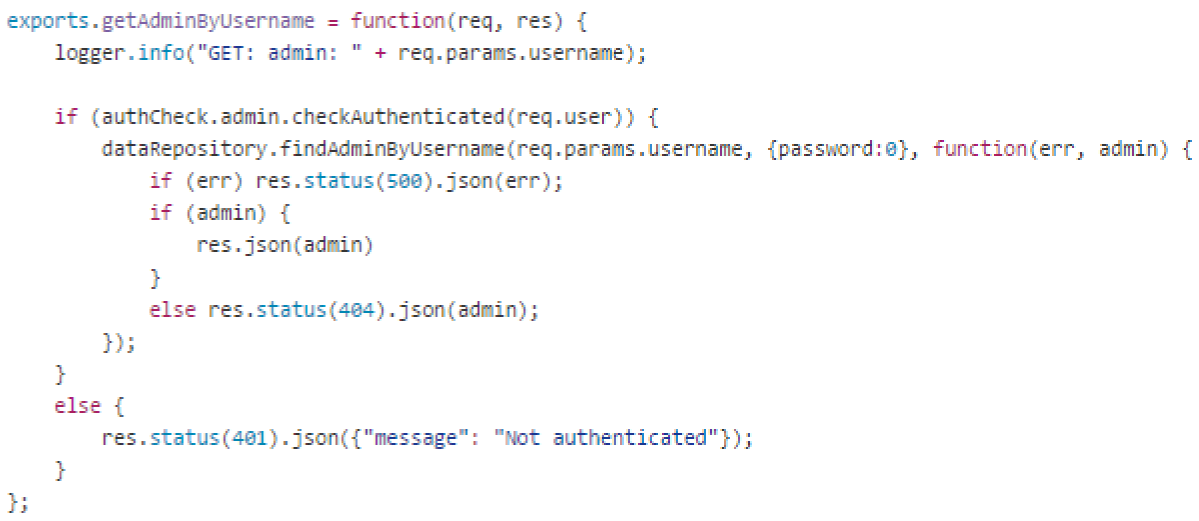


```
exports.authenticateUser = function(req, res, next) {
    logger.info("POST: authenticating user: " + req.body.username);

    passport.authenticate('local', function(err, user, info) {
        if (err) { res.json({"authenticated": false, "error": err}); }
        else if (user == false) res.json({"authenticated": false, "message": info.message});
        else {
            //Setup a session:
            req.logIn(user, function(err) {
                if (err) { res.json({"authenticated": false, "error": err}); }
                if (user.admin != null) {
                    res.json({"authType": "admin", "authenticated": true, "authenticatedUser": user});
                }
                else {
                    res.json({"authType": "student", "authenticated": true, "authenticatedUser": user});
                }

            });
        }
    })(req, res);
};
```

Figure 30: Authenticate User Function

The line of interest here is where it makes a call to 'req.logIn()' which binds the user object outlined above to the session (and cookie which is handled behind the scenes by Passport) and can then be used by every other service (via req.user) to check authentication. Logging a user out (ending a session) is very similar to this and a call to 'req.logout()' will wipe all contents of a session so all services will not be able to identify the user anymore.

To determine whether a user is authorised to access a resource some logic needs to be applied to determine the access rights. To prevent this clogging up the services layer, I have implemented a series of functions to 'check authentication'. Each service will have to call these functions to determine access rights, for example to get an admin by their username (you must be an admin!):



```
exports.getAdminByUsername = function(req, res) {
    logger.info("GET: admin: " + req.params.username);

    if (authCheck.admin.checkAuthenticated(req.user)) {
        dataRepository.findAdminByUsername(req.params.username, {password:0}, function(err, admin) {
            if (err) res.status(500).json(err);
            if (admin) {
                res.json(admin)
            }
            else res.status(404).json(admin);
        });
    }
    else {
        res.status(401).json({"message": "Not authenticated"});
    }
};
```

Figure 31: Get Admin Function

From the above snippet we can see that *authCheck* provides a quick and easy way to check whether the user bound to the current session is authenticated to access this type of resource and failure to succeed past this will result in a HTTP 401 (unauthorised) response. The below diagram shows how the *authCheck* object can be used to determine Admin or Student authorisation.

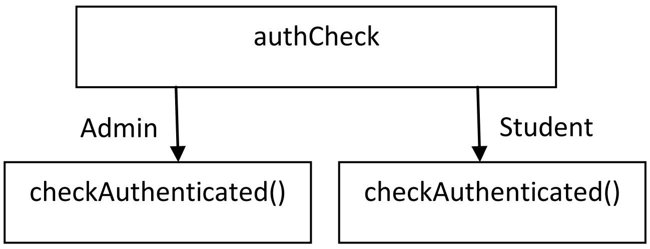


Figure 32: Authenticate Check Diagram

This method of authentication is very seamless and keeps all of the services lightweight as all that is necessary is to check whether the user is authenticated is a check up on the session object.

Server-Repository Interaction
To carry out their jobs, services rely on repository functions (discussed earlier in the report) to access and make changes to resources in the database. They do this by simply accessing their respective repository, the red lines on the below diagram illustrate the interaction between services and repositories:

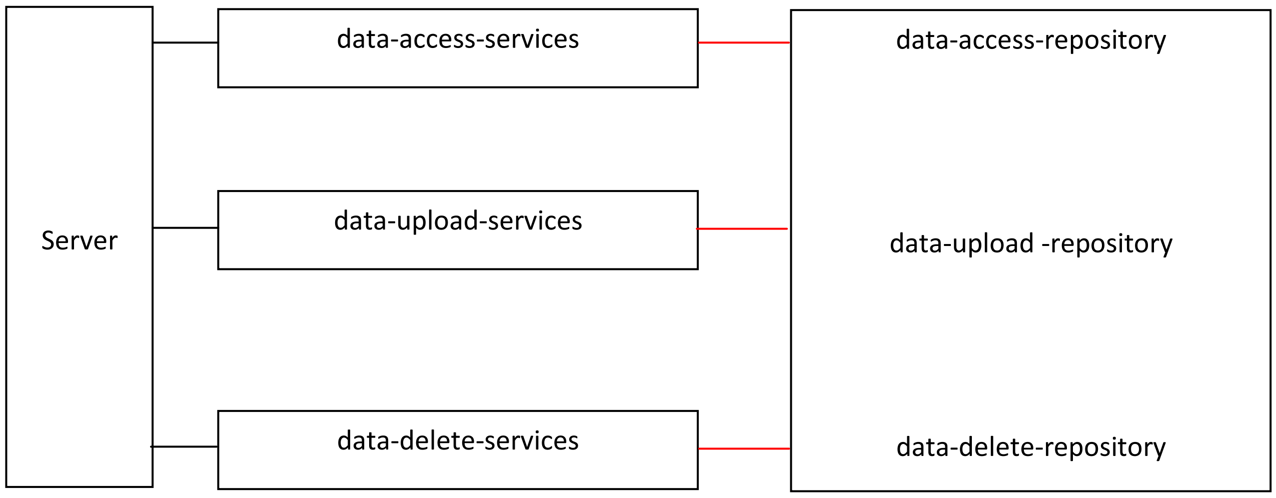


Figure 33: Server-Repository Diagram

Notable Service Implementations
Many of the services are very simple and only have a single role (e.g. GET all assessments), however, the requirements for this project call for a little more complex application logic and the use of the below packages have been used to carry out some of these roles cleanly and efficiently:

- MD5 - Used to hash passwords for database storage
- pdfkit - Used to generate the assessment results report
- async - Used to safely and reliably handle asynchronous calls within loop iterations

Service function: createAssessmentResultsReport(req, res)

The first notable service implementation helps in solving the below requirements:

- Student requirement 4 - download a document showing limited results
- Trainer requirement 5 - download a document showing results

Using the req.user object defined when authenticating (explained in authentication implementation) the service can identify who the user is and after authenticating, applies the following logic as to what the document should contain:

1. If session user is Student:
    - Check whether the requested Student result username matches the currently logged in session user:
        i. If they **match**: authorised, build a *limited* document based on results and respond
        ii. If they **don't match**: not authorised, respond with HTTP response code 401
2. If session user is Trainer:
    - Authorised, build document based on results and respond
3. If no session exists:
    - Not authorised, respond with HTTP response code 401

The NPM package pdfkit was used to easily build the document by using it's API functions to construct a pdf document. Each function in the API returns the updated document so a chaining approach in the code can be used to make the code short but still very readable, for example:

- doc.fontSize(2).fillColor("#004B8E").text("Assessment Summary").moveDown(1);

The above line performs four operations but only takes up one line of code which made it simple to write and understand, the generation of the document was swift and the API makes documents easy to write. Two example documents (full & limited) can be found in the appendix of this report and the implementation of this service function can be found here: https://github.com/J-Morris676/SWPAssessment/blob/master/services/data-access-services.js.

Service function: attemptAssessmentStart(req, res)

The second notable service implementation helps in solving the below requirements:

- Student requirement 1 - Allow a user to sit an assessment that they have been scheduled to sit at the correct time start time
- Student requirement 5 - *Additional*: lock an assessment version once it has been started for testing

In order to satisfy these requirements for starting an assessment session this service has to make three different requests to the repositories, after authenticating. The following logic is applied when a Student client requests to start a scheduled assessment.

1. Find the scheduled assessment in the database using the given request (schedule and student)
    - If it couldn't be found using the given Student, not authorised, respond HTTP 401
2. Check whether the current time is between the start and end time of the requested assessment schedule.
    - If it is after the end date of the assessment respond not authorised (401) with a message informing the client the assessment schedule has already been
    - If it is before the start date of the assessment respond not authorised (401) with a message informing the client the assessment schedule hasn't started yet
3. Start Student on assessment schedule by adding the start date in the database
4. Lock the assessment version by setting the locked field to 'true' in the assessments collection

Service function: updateAssessmentProgress(req, res)

It is also possible to 'update' the progress that a Student has made on an assessment by utilising the above service function, which applies a method to check whether a Student has started and updating the progress by updating the assessment result they posted by inserting it into the database. This also means that the remnants of an uncompleted assessment can still exist in the database and could even be utilised in future implementations mentioned later in the report.

Service function: attemptAssessmentEnd(req, res)

This service implementation helps in solving the below requirement:

- Student Requirement 2 - The system must allow the user to submit an assessment during the period of time between a scheduled assessments start and end time

This service will be used when a user submits their assessment and applies the following logic:

1. Find the scheduled assessment in the database using the requests schedule and student
   - If it couldn't be found using the given Student, not authorised, respond HTTP 401
2. Check whether the current time is between the start and end time of the requested assessment schedule.
   - If it is after the end date of the assessment respond not authorised (401) with a message informing the client the assessment schedule has already been
   - If it is before the start date of the assessment respond not authorised (401) with a message informing the client the assessment schedule hasn't started yet
3. Check whether the student has finished assessment schedule
   - If they have, respond not authorised (401) with a message informing client user has already submitted
4. Get the version of the assessment in the database and mark the questions that was posted along with the request
5. Respond with an end assessment confirmation and the marked assessment

This service is where the marking feature appears and is implemented using a separate service which is solely focused on marking a given assessment.

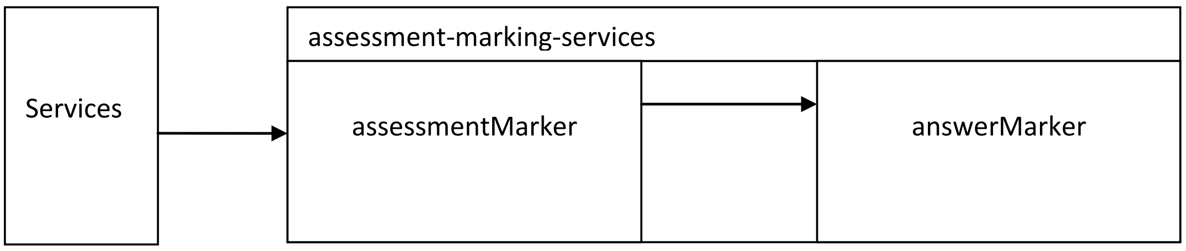


Figure 34: Assessment Marking Diagram

The diagram above shows how the services can utilise the assessment-marking-services which can be found in this directory:
https://github.com/J-Morris676/SWPAssessment/tree/master/services/assessment-marking-services

- assessmentMarker - This file exposes a Node.js export function which takes the assessment and the answers as arguments and iterates through each question summing up a grade utilising the answer marking file explained below.
- answerMarker - This file provides the three answer type implementations
  - **Free Text**: freeTextMark(question, answer)
  - **Call Log**: callLogMark(question, answer)
  - **Multiple Choice**: multiChoiceMark(question, answer)

The call log and multiple choice marking implementations are very simple as they add a mark by determining whether the selected answer is equal to the correct answer. The free text marking implementation is a little more complex as it needs to find pre-defined answers in a given piece of text.

During the meetings, whilst discussing this the Client expressed that it would be favourable for the system to be a little lenient on the free-text marking. It is for this reason I decided to look into the potential of utilising a string distance algorithm, to determine whether a word or term is similar to but not exact allowing for things like typo's.

The free-text mark initially checks for each answer in the text using a standard regular expression pattern, for example, the below pattern when the answer is 'test':

- /.*^test$.*/i

In order to allow for typos I had to look past the use of regular expressions and began looking into existing algorithms that can measure the distance between two strings preferably with an implemented open source solution with the little time I had. I soon discovered that Node.js has a Natural Language processing library that is still relatively mature in advanced natural language features but does provide the facilities that I require for this feature. NaturalNode provides string distance algorithms that I have used to determine whether two words are of any similarity. The library has these solutions to this problem:

Levenshtein Distance
The Levenshtein distance is a String metric that measures the distance between two sequences by calculating the minimum number of single-character edits (i.e. insertions, deletions or substitutions)

Figure 35 shows an alignment diagram for strings 'Monday' and 'Tuesday' and highlights how the edit distance can be calculated using a matrix, the grey cells represent the characters in the words and the blue cells represent where a character comparison was made (and incremented when an insertion, deletion or substitution operation needs to be made:

		T	u	e	s	d	a	y
	0	1	2	3	4	5	6	7
M	1	1	2	3	4	5	6	7
o	2	2	2	3	4	5	6	7
n	3	3	3	3	4	5	6	7
d	4	4	4	4	4	4	5	6
a	5	5	5	5	5	5	4	5
y	6	6	6	6	6	6	5	4

Figure 35: Levenshtein Alignment Matrix

As seen in the figure, the Levenshtein distance between 'Monday' and 'Tuesday' is 4.

Jaro-Winkler Distance

The Jaro-Winkler distance is a measure of similarity between two strings by normalising a score between 0 (Not similar) and 1 (exact match) - it is best suited for short strings. This distance is an extension of the Jaro distance by using a standard weight (0.1), the Jaro equation is shown in figure 36.

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

Figure 36: Jaro Equation

The equation in figure 37 is used to then figure out the Jaro-Winkler distance where:

- *l* is the length of the matching prefix
- *p* is a standard weight
- *jaro* is the result after the jaro equation (d_j from figure 36)

$$d_w = jaro + (l * p(l - jaro))$$

Figure 37: Jaro-Winkler Extension Equation

For example, The diagram below is an alignment diagram for the strings 'Monday' and 'Tuesday' and highlights how matching characters can be calculated using a matrix, the grey cells represent the characters in the words and the blue cells represent where two characters have been identified as a match.

	T	u	e	s	d	a	y
M	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0
n	0	0	0	0	0	0	0
d	0	0	0	0	1	0	0
a	0	0	0	0	0	1	0
y	0	0	0	0	0	0	1

Figure 38: Jaro-Winkler Alignment Matrix

Figure 38 shows three matching characters (*m*) and can now be substituted into the equations in figures 36 and 37 to calculate the Jaro-Winkler distance:

$$m = 3$$
$$|Monday| = 6$$
$$|Tuesday| = 7$$
$$t = 0$$
$$d_j = \frac{1}{3}\left(\frac{3}{7} + \frac{3}{6} + \frac{3-0}{3}\right) = 0.643$$
$$d_w = d_j + (0 * 0.1(1 - d_j)) = 0.643$$

As the length of the matching prefix in my example is equal to 0, the Jaro-Winkler is the same as the Jaro distance which is 0.643.

As it is possible for trainers to use 'terms' as answers and not just words an overall better solution is to check on a per-word basis (and every word must have an similarity over a set threshold). As the words can be of any size (more frequently between 3-10 characters) the Levenshtein distance may not be the best implementation for small words:

Levenshtein Examples		
Actual answer	**Given answer**	**Distance**
a	b	1
the	ase	2
thoroughly	thoroghly	2

Figure 39: Levenshtein Examples

From these small examples it is already clear that the given distance accuracy varies drastically based on the word size as it is the letters in the word that determine the output distance. The highlighted rows have the **same** distance but the larger word 'thoroughly' would be considered more correct (similar) to 'thoroghly' than the smaller word 'the' would be to 'ase'.

If we take the same examples using the Jaro-Winkler distance we can see more realistic distances between the words regardless of the word length:

Jaro-Winkler Examples		
Actual answer	**Given answer**	**Distance**
a	b	0
the	ase	0.56
thoroughly	thoroghly	0.96

Figure 40: Jaro Winkler Examples

With a little bit of testing and an analysis of the result from different types of common typos the Client decided to go for a similarity distance of 0.9 as it is still important that Students type accurately due to the work conditions that they move onto after successful completion of the courses that they are on.

The solution with the Jaro-Winkler distance involves finding all equally sized terms in the free-text and measuring the similarity in each word of each term and if **all** words are greater than 0.9 in similarity then it will mark the answer as correct.

<u>**Example (Case insensitive):**</u>
Actual answer: "Cupper with a copper"
Given answer: "The answer is Cuper with a coper."

Iteration 1			
Cupper	with	a	copper
The	answer	is	cuper
0	-	-	-
Iteration 2			
Cupper	with	a	copper
Answer	is	cuper	with
0	-	-	-

...

Iteration 4			
Cupper	with	a	copper
Cuper	with	a	coper
0.91	1	1	0.91

Figure 41: String Distance Solution Sample Iterations

Iteration 4 of equal sized terms is "Cuper with a coper " and it finds that each of these words are greater than 0.9 in similarity and will mark it as correct. If any word in a term is found to be less than 0.9 in similarity then the iteration will end to optimise the implementation. All three of the marking implementations can be found in this file: https://github.com/J-Morris676/SWPAssessment/blob/5e0d80ce569d1a142e10e0757c49fa50e5f72ea7/services/assessment-marking-services/answerMarker.js

Other methods to improve marking robustness is by implementing algorithms to detect other common types of spelling mistakes and if there was more time, phonetic algorithms like Metaphone and Soundex would be utilised to identify input words that are incorrect but sound and/or pronounced the same in English.

Service Interface - Server

This layer of the server consists of the API routes that are mapped to the services and also performs validation on input using the NPM package express-validation. The below table shows a few GET examples and how these resource URI's are mapped to validation schemas and services:

Resource URI	Validation Schema	Service
/resources/admins	-	dataAccessServices.getAllAdmins
/resources/admins/:username	User (parameter)	dataAccessServices.getAdminByUsername
/resources/assessments	-	dataAccessServices.getAllAssessments
/resources/assessments/:assessmentId	Assessment (parameter)	dataAccessServices.getAssessmentById
/resources/assessments/:assessmentId /versions	Assessment (parameter)	dataAccessServices.getAssessmentVersions

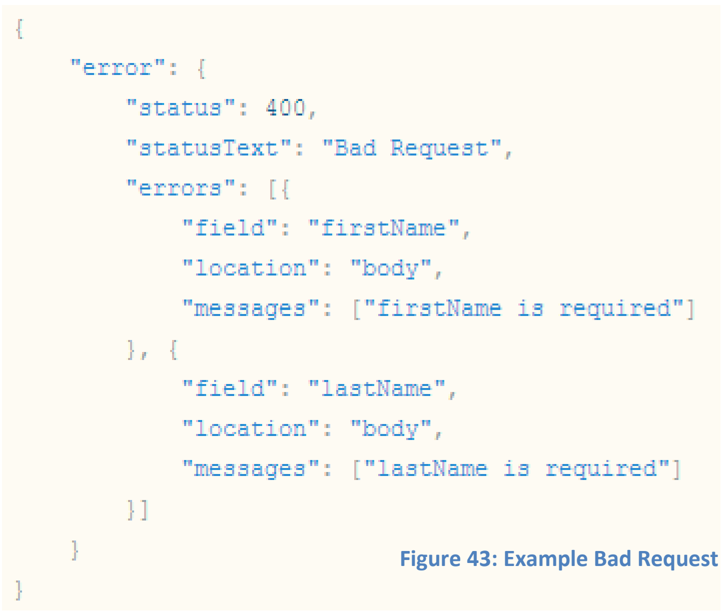
Figure 42: Server API Resource Example URI's

As mentioned earlier in this report, when using HTTP it is possible to take input data and to validate it using a set of pre-defined schemas which have been designed with express-validation, to respond to Clients with relevant data if the data they send is not valid. These validation schemas can be found in two files found here:
https://github.com/J-Morris676/SWPAssessment/tree/master/validation

- http-body-validation - defines valid schemas for HTTP bodies using POST requests
- http-url-params-validation - defines valid schemas for HTTP bodies using GET requests

For example, if we wanted to use the REST console to add a Student (signed in as an admin):

POST {"username": "Bob", "password": "asd"} **to** http://localhost:3003/resources/students
The response of this request is a 400 (Bad Request) and informs the Client of what was invalid, this object is generated by express-validation based on the schema set in the files discussed above. The full response can be seen below and is extremely useful for a Client:



```
{
    "error": {
        "status": 400,
        "statusText": "Bad Request",
        "errors": [{
            "field": "firstName",
            "location": "body",
            "messages": ["firstName is required"]
        }, {
            "field": "lastName",
            "location": "body",
            "messages": ["lastName is required"]
        }]
    }
}
```

Figure 43: Example Bad Request Response

Alongside this, this server layer is responsible for server configurations and is the entry point to starting the server and starting it is as simple as calling the below command:

- node server.js

Client

As mentioned in the design section of this report, the application I decided to implement would use the MEAN stack, the Angular.js part of this is a client MVC framework that maps HTML templates to controllers which control all of the logic behind the template. It also offers the following powerful features:

- **Two-way data binding**: If you've ever looked at Angular.js before you'd have come across this concept that allows you to bind JavaScript variables onto a $scope object which updates the DOM on any variable change
- **HTML Directives**: A method for extending the HTML vocabulary and can be used to create custom HTML elements
- **Native Angular Directives**: Angular provides a full library of directives that can be used to render DOM elements based on the state of some JavaScript variable (on the controllers $scope)

I also mentioned that I will be using Bootstrap as a UI framework to assist me in creating a fully responsive web application whilst minimising the amount of styling code I had to write.

It is up to the developer how an Angular.js project is structured but for pre-configurations and a quicker setup I decided to use the angular-seed (https://github.com/angular/angular-seed). This project contains a structure that can be adapted for larger projects (like this one). This Angular structure can be represented in the following high-level diagram:

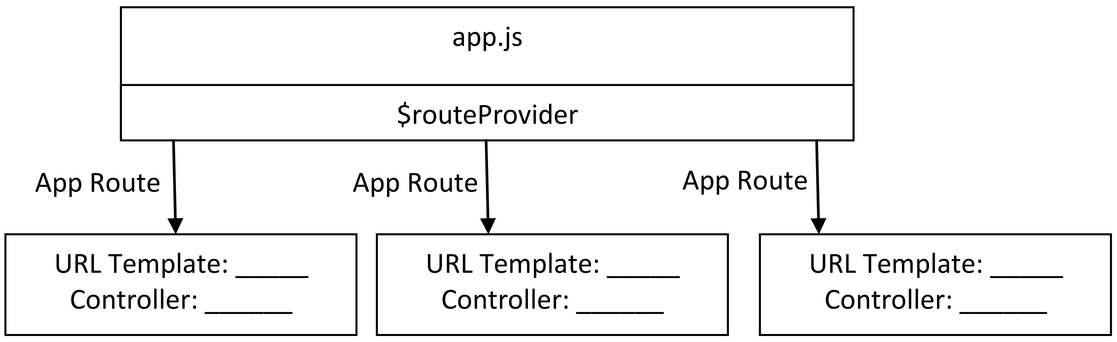


Figure 44: Web Application Architecture

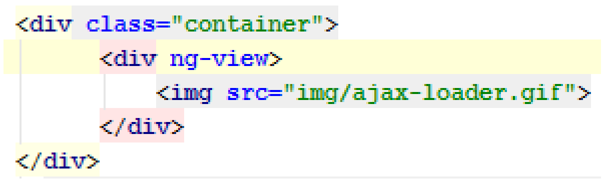
Angular controllers are defined in modules which can be injected into the high level app.js module and then uses Angular's $routeProvider (https://docs.angularjs.org/api/ngRoute/provider/$routeProvider) to map what controller and HTML template to use when a certain web application route is visited making it an MVC structure. Angular.js achieves this by utilising AJAX to retrieve the HTML template and compile them in place on the document, this is also very useful when creating a directive (extended HTML) as it is possible to retrieve and compile large HTML templates. This concept seems to be what coins the term 'web app' differing from 'website' as it doesn't provide a full page reload on a route change and is all controlled using the above configuration, this makes it possible for all scripts and CSS libraries (35 in total) to only be retrieved once whilst navigating throughout the web application.

The ngView (https://docs.angularjs.org/api/ngRoute/directive/ngView) directive is where the template HTML will go and can be seen in my index.html file. Everything outside of the ngView attribute element will not need to be rendered again as the template HTML files are placed directly inside this element everytime. The ajax-loader that it contains by default is what appears whilst Angular is fetching the HTML template letting the user know that the application is working on something.

The index.html file can be found here:

- https://github.com/J-Morris676/SWPAssessment/blob/master/public/index.html

The interesting snippet from this file is shown below:

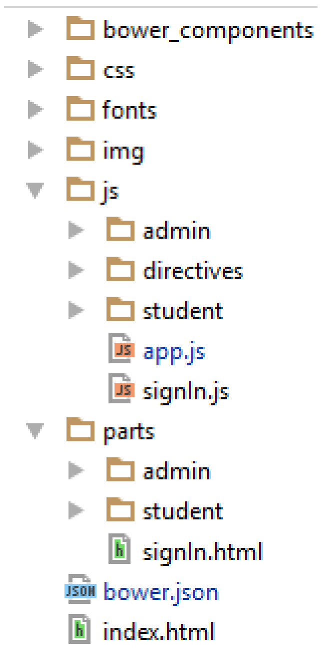


```
<div class="container">
        <div ng-view>
            <img src="img/ajax-loader.gif">
        </div>
</div>
```

Figure 45: AJAX Loaded Code Element

The overall file structure of the client project can be seen below:

- Admin/Student scripts and HTML are divided into different directories to aid navigation
- Note how the app.js (containing configurations already mentioned) and signIn are top level as they're both shared by Student and Trainer parts of the system



```
▶ bower_components
▶ css
▶ fonts
▶ img
▼ js
    ▶ admin
    ▶ directives
    ▶ student
      app.js
      signIn.js
▼ parts
    ▶ admin
    ▶ student
      signIn.html
  bower.json
  index.html
```

Figure 46: Web Application File Structure

Utilising the power of Angular.js and Bootstrap to construct this assessment system has been a very enjoyable experience and I couldn't imagine building a web application without using some form of similar frameworks in the future. This report will now provide a high level overlook at the various aspects of the client application and how they have helped to solve system requirements, highlighting how particular features adhere to Nielsen heuristics.

Login Page

The login page consists of one field that can be used for both Students and Trainers, it is not possible for both a Student and a Trainer to have a duplicate username so this is automatically identified and the user is navigated accordingly. The sign in page is a very basic splash page that requires a user to enter their credentials and satisfies the first system requirement for both users.

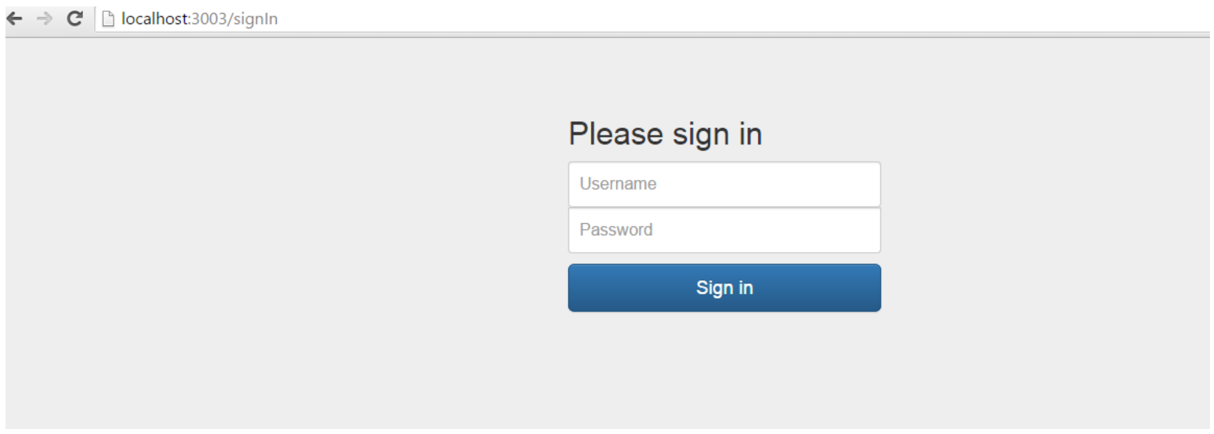


Figure 47: Sign In Page

Navigation Bar

In order to aid in *user control and freedom* a navigation bar will always be present after the user has logged in. This is to ensure that no matter what the state of the system may be - the user can always navigate back home using the navigation bar. The navigation bar design is the battenberg police pattern found throughout their corporate PowerPoint presentation slides.

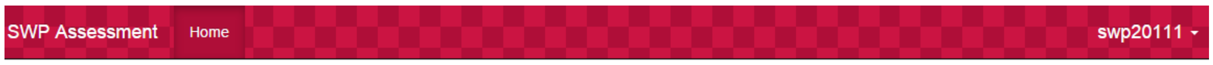


Figure 48: Navigation Bar

As it is responsive, the navigation bar looks more like this on smaller resolutions:

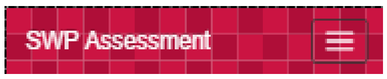


Figure 49: Mobile Navigation Bar

The menu button that appears here is a common button found throughout mobile applications and indicates that a menu is available by clicking it, this simply replaces the drop down menu that appears on the full screen version:

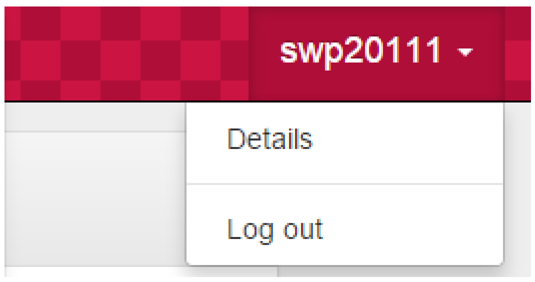


Figure 50: Selectable Navigation Item

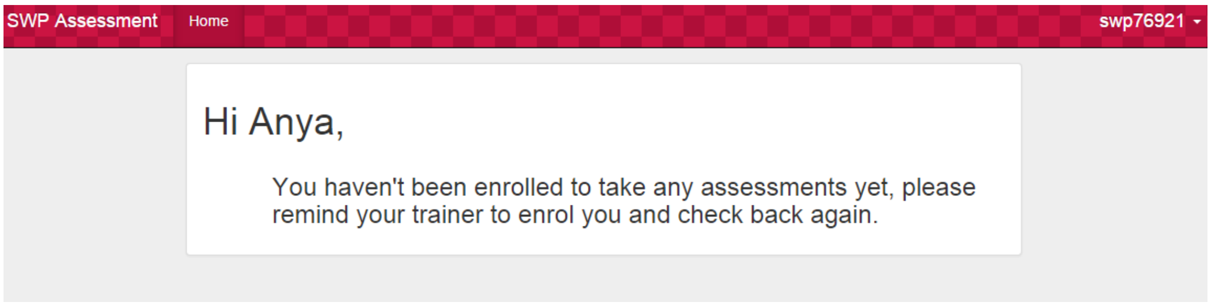
This navigation bar is obvious, similar to existing popular systems and utilises recognisable and relevant button features making it very easy to use even for a complete beginner. It also provides the option to log out of the system which satisfies the second system requirement for both users.

Home Pages

Once logged in, both Student and Trainers will be navigated to their respective homepage and both of these home pages look very familiar, the only difference between the two is the content that appears before them.

Student Home Page

A Student's initial view is the below page which gives them information about why they cannot view anything. The system requires that a Student is due to sit an assessment or has already finished assessments to get past this stage.

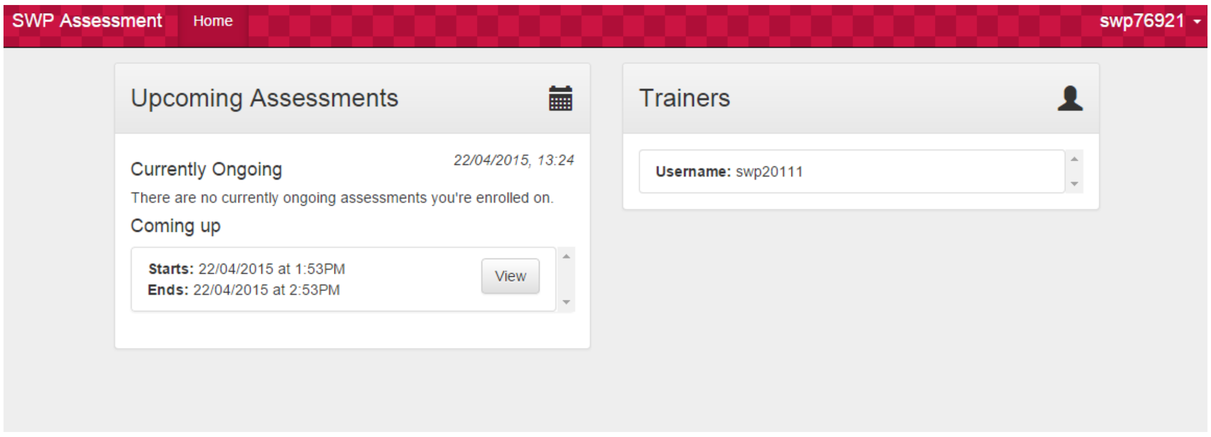


Hi Anya,

You haven't been enrolled to take any assessments yet, please remind your trainer to enrol you and check back again.

Figure 51: Student Homepage Pre-scheduled

Once the Student has been scheduled for assessment, the Student view will change to inform them that they are due to sit an assessment:



22/04/2015, 13:24

Currently Ongoing

There are no currently ongoing assessments you're enrolled on.

Coming up

Starts: 22/04/2015 at 1:53PM
Ends: 22/04/2015 at 2:53PM

Username: swp20111

Figure 52: Student Homepage Post-scheduled

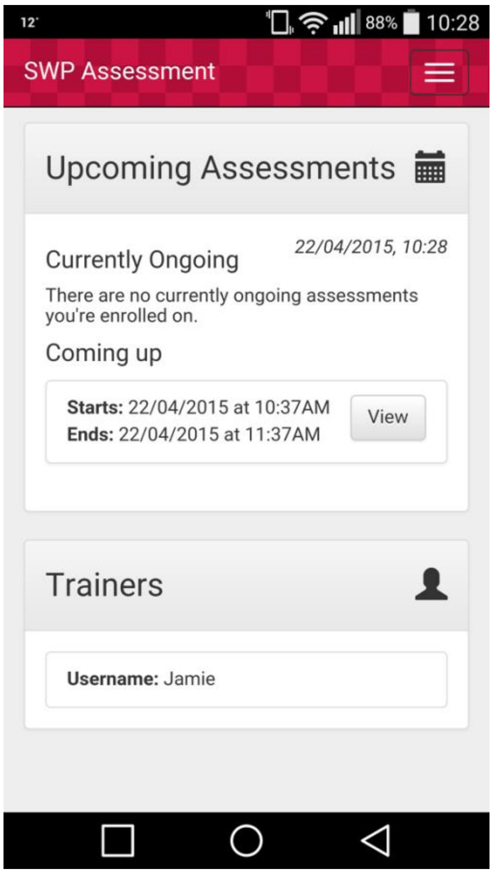
This view shows the different widget-like panels that show previews of the resources available shown in designs earlier in the report, to make this content responsive, the panels utilise the Bootstrap grid system (http://getbootstrap.com/css/#grid) which splits rows up into columns of 12 with column size being different depending on the current screen size highlighted in figure 53.

Bootstrap's definitions of screen sizes using the grid system http://getbootstrap.com/css#grid-options			
Extra small devices Phones	Small devices Tablets	Medium devices Desktops	Large devices Desktops
<768px	≥768px	≥992px	≥1200px

Figure 53: Bootstrap Screen Size Definitions

To visualise this concept I have tested the application using the Google Chrome tool, a tablet device (iPad) and mobile device (LG G3).

LG G3 Student front page



Upcoming Assessments

Currently Ongoing

22/04/2015, 10:28

There are no currently ongoing assessments you're enrolled on.

Coming up

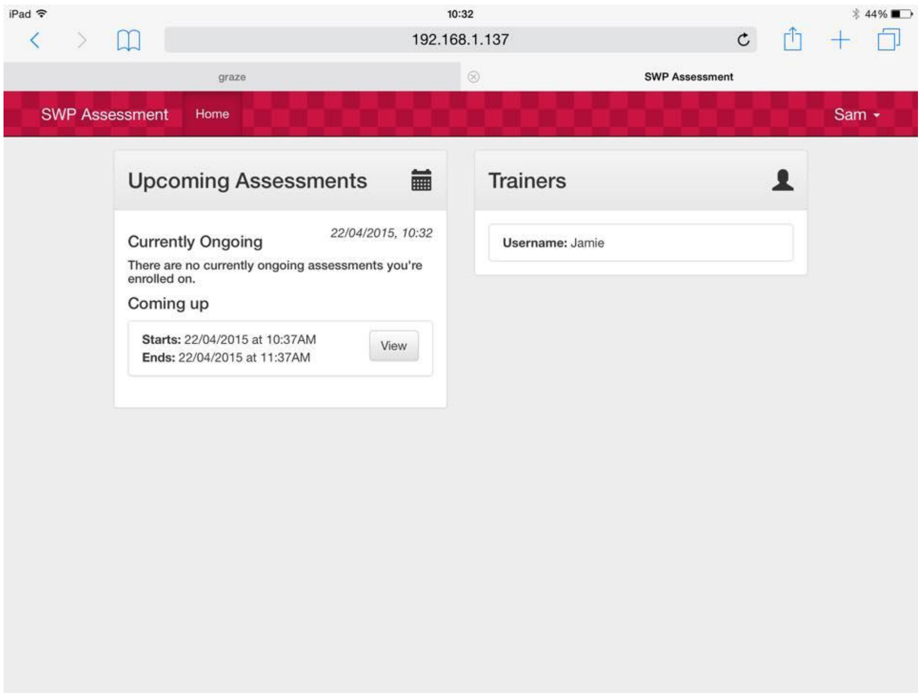
Starts: 22/04/2015 at 10:37AM
Ends: 22/04/2015 at 11:37AM

Trainers

Username: Jamie

Figure 54: Mobile Homepage

iPad Mini front page



Upcoming Assessments

Currently Ongoing 22/04/2015, 10:32

There are no currently ongoing assessments you're enrolled on.

Coming up

Starts: 22/04/2015 at 10:37AM
Ends: 22/04/2015 at 11:37AM

View

Trainers

Username: Jamie

Figure 55: Tablet Homepage

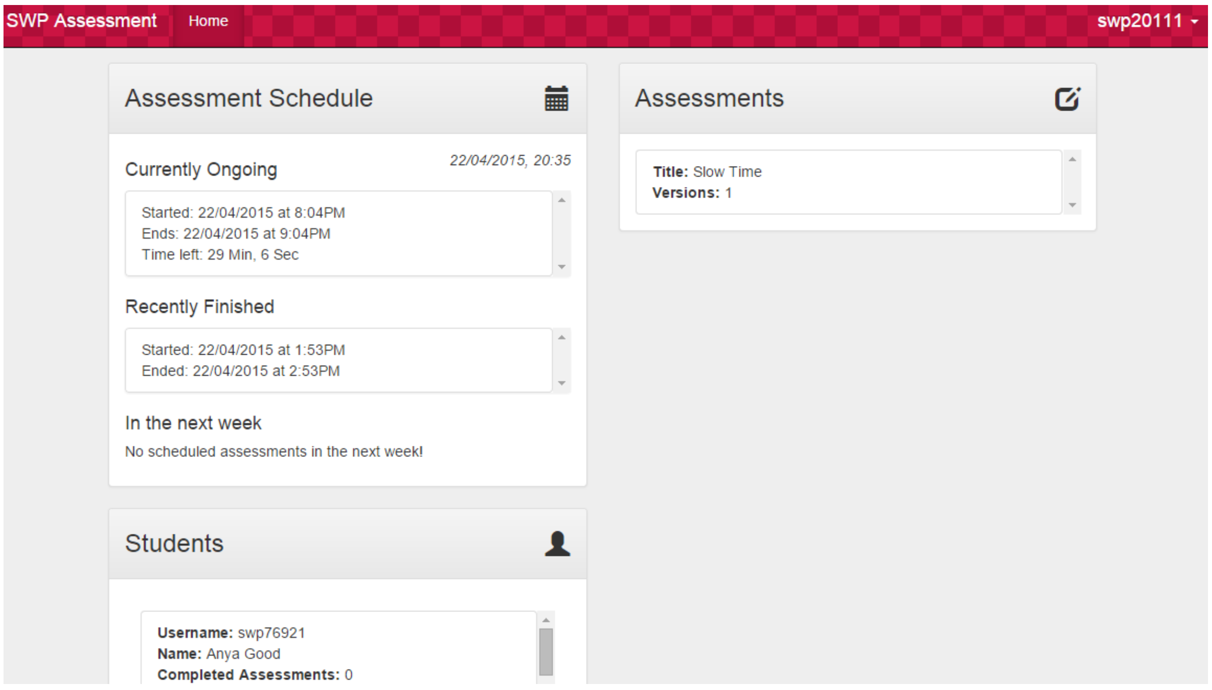
A directory full of screenshots for mobile and desktop can be found in the appendix.

Trainer Home Page
The Trainer home page is very similar to the Student home page but the panels that appear before them are:

- **Assessment Schedule**: Can be used to navigate to create, view, update, and delete assessment schedules
- **Assessments**: Used to navigate to create, view, update, and delete assessments
- **Students**: Used to navigate to create, view, update, and delete Students

The below screenshot also demonstrates how ongoing assessments appear to users of the system, to utilise v*isibility of system status* as much as possible the system as a whole is as active as possible by:

- Using countdown timers
- Using real time dates and 24-hour clocks
- Showing the state of assessment, student and schedule resources (locked/unlocked etc.)
- Clear page titles to inform the user of where they are and what they're looking for



Currently Ongoing
22/04/2015, 20:35
Started: 22/04/2015 at 8:04PM
Ends: 22/04/2015 at 9:04PM
Time left: 29 Min, 6 Sec

Recently Finished
Started: 22/04/2015 at 1:53PM
Ended: 22/04/2015 at 2:53PM

In the next week
No scheduled assessments in the next week!

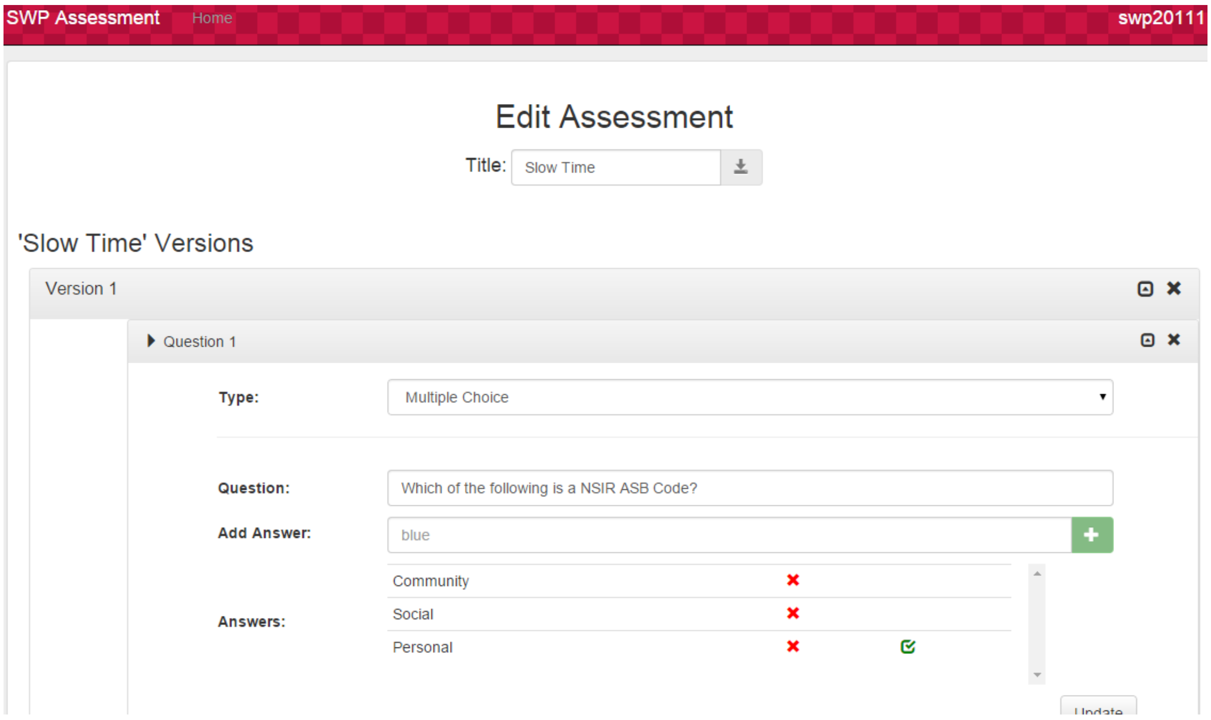
Title: Slow Time
Versions: 1

Username: swp76921
Name: Anya Good
Completed Assessments: 0

Figure 56: Trainer Homepage

Editing Assessments (Trainers)

When an assessment is created by a trainer, it can be edited by navigating into it by pressing an edit button, this takes them to the page shown below:



SWP Assessment Home swp20111

Edit Assessment

Title: Slow Time

'Slow Time' Versions

Version 1

Question 1

Type: Multiple Choice

Question: Which of the following is a NSIR ASB Code?

Add Answer: blue

Answers: Community ✗ / Social ✗ / Personal ✗ ✓

Update

Figure 57: Edit Assessment Page

From here it is possible to do the following operations:

- Edit the title of the assessment
- Add/delete new versions of the assessment
- Add/edit/delete new questions into the versions of each assessment

This implementation satisfies requirement one of the Trainer requirements and allows them to have full control over assessments.

As mentioned before, each question can be different and the form that displays below the 'Type' selection box will change based on what is selected, this is implemented using Angular directives and takes on the structure below:

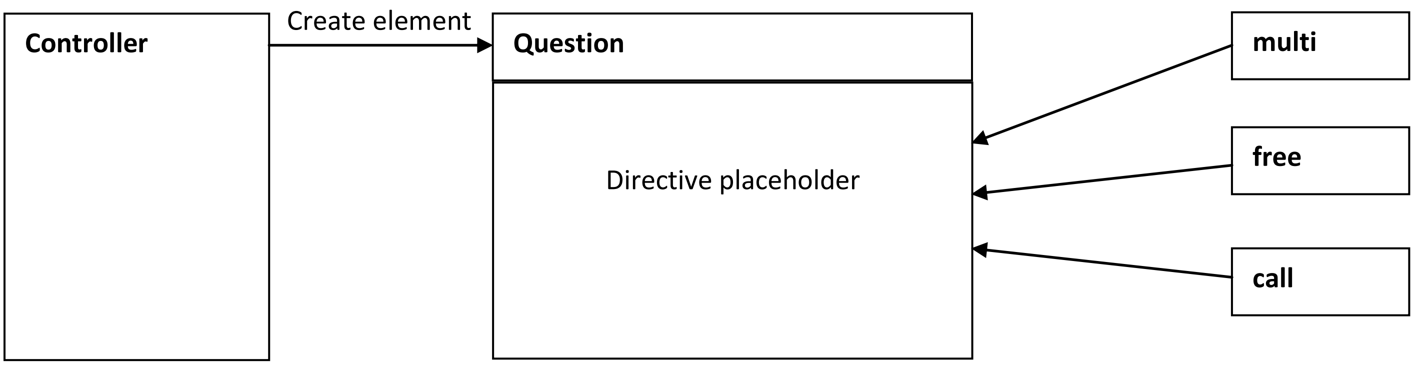


Figure 58: Angular Directive Structure

- The higher-level question directive is responsible for the 'type' selection and when selected will place the directive for the selected question type on the template and compile it
- The mark-up and logic for the question is then handled by the particular question directive

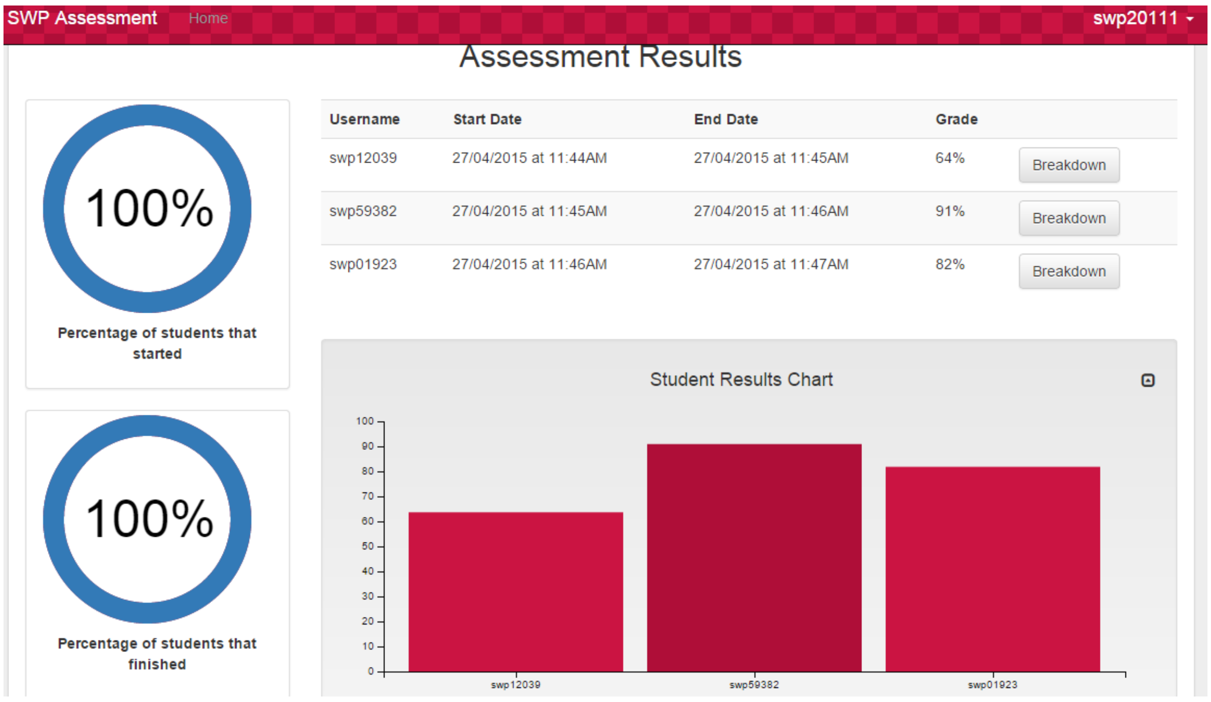
This is all possible by binding the controller variables to the directives and they will update whenever the user makes changes to the content in the directives. The point of doing this is to encapsulate the often complex functionality of each question, the source code for this question selection implementation can be found in the following files:

- All four directives:
    - https://github.com/J-Morris676/SWPAssessment/blob/master/public/js/directives/assessmentEditingDirectives/assessmentEditingDirectives.js
- The HTML templates that they use can be found in this directory:
    - https://github.com/J-Morris676/SWPAssessment/tree/master/public/js/directives/assessmentEditingDirectives/templates

Viewing Assessment Results (Trainers)

Although not a direct functional requirement, when an assessment has finished it has been made possible to view the list of all students who completed the assessment. This makes it much easier to assess how a Trainers Students did all in one place. Other features of this page include visualisations which show:

- Percentage of Students who started the assessment
- Percentage of Students who finished the assessment
- A bar chart displaying the results of each Student


SWP Assessment Home swp20111

Assessment Results

Username	Start Date	End Date	Grade	
swp12039	27/04/2015 at 11:44AM	27/04/2015 at 11:45AM	64%	Breakdown
swp59382	27/04/2015 at 11:45AM	27/04/2015 at 11:46AM	91%	Breakdown
swp01923	27/04/2015 at 11:46AM	27/04/2015 at 11:47AM	82%	Breakdown

Figure 59: Assessment Results Page

These features make the results page visually rich, additional features have been planned and outlined in the future work section of this report.

All visualisations were made possible by using the d3.js library (http://d3js.org/) which provides a practical API for manipulating SVG elements on the document making these visualisations lightweight, as they are just a series of HTML elements. D3 was selected over other alternatives as it is possible to create visualisation with just pure JavaScript and is easily integrated with Angular. Due to the amount of control a developer has when using D3 it's also really easy to make visualisations responsive by monitoring the width of the window that the document is running in. Each function in D3 returns the element you've called the function on so a chaining code method can be utilised making it really simple to use, for example:

```
var chart = svg
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
    .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");
```

Figure 60: d3 Library Usage Snippet

The above code simply adds attributes to the 'svg' d3 object by simply 'chaining' the functions to each other and says:

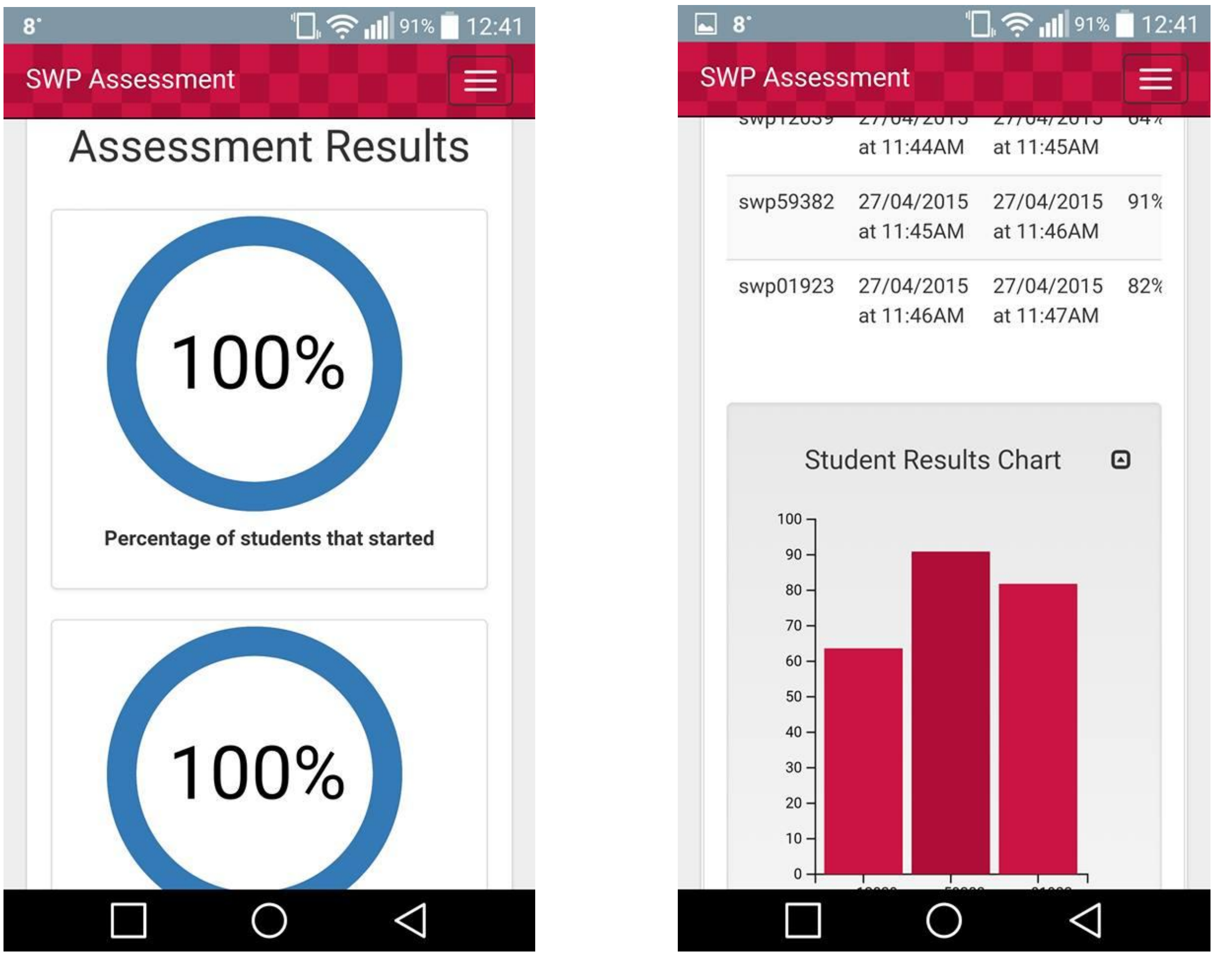
- *"Set the width attribute to this, the height attribute to this, append a <g> SVG element to it and apply a transform attribute to the <g> element just created"*
- The value of 'chart' will now be the new <g> element created at the end of the chain of functions

These visualisations have also been wrapped into Angular directives to encapsulate the code that creates them and makes them reusable (i.e. the two pie chart graphs are the same directive). The HTML tag used to create a pie chart visualisation can be seen below:

- <pie-chart ng-model="startPercentage"> <pie-chart>

This directive takes a single parameter which is a two-way variable used to model to this visualisation and that represents the percentage of pie to display.

Due to it being possible to make the visualisations responsive, the graphs also work on mobile devices and examples can be seen on the screenshots below.



	at 11:44AM	at 11:45AM	
swp59382	27/04/2015 at 11:45AM	27/04/2015 at 11:46AM	91%
swp01923	27/04/2015 at 11:46AM	27/04/2015 at 11:47AM	82%

Figure 61: Mobile Assessment Results

Both of these graph directives can be found in the this file: https://github.com/J-Morris676/SWPAssessment/blob/master/public/js/directives/studentResultsGraphsDirectives/graphDirectives.js

System Popovers

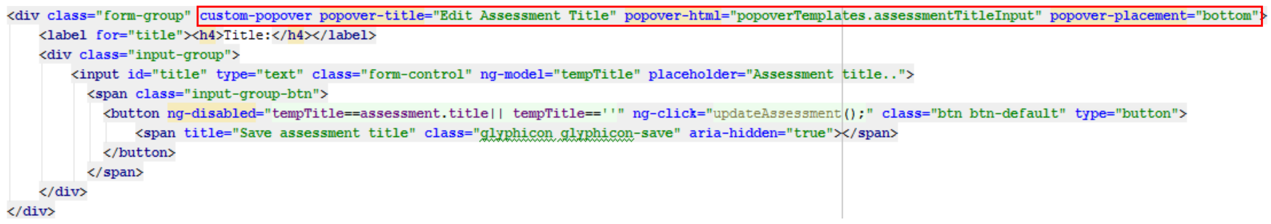
The purpose of the popovers in this web application is to aid in the below Nielson heuristics:

- *Aesthetic and minimalist design*
    - Popovers are very lightweight and are optional to view (on hover). They also contain colour coded symbols and small pieces of separated text to explain what a component does
- *Help and documentation*
    - As popovers are used on almost every element on the system, users will know to hover over an element to see an explanation of the particular component

It was discussed in length during meetings how to provide useful information on the system without clogging it with too much text that users simply will not read. After a little inspiration from existing systems (discussed more in depth in my weekly report document) it was decided that the system would utilise rich tooltips called popovers (http://getbootstrap.com/javascript/#popovers) in the Bootstrap framework. As there would be many popovers throughout the system they are implemented as Angular directives to reduce repetitive jQuery throughout the controllers, this directive binds to an attribute called 'custom-popover' which expects the below attributes as input:

- **popover-title <Text>**: The title that this popover will display
- **popover-html <URI>**: The URI that points to the HTML template that should be displayed on this popover
- **popover-placement <position>**: Where this popover should appear relative to the element

It is very simple to use this directive and the outer div element in the snippet below provides an example of its usage:



```
<div class="form-group" custom-popover popover-title="Edit Assessment Title" popover-html="popoverTemplates.assessmentTitleInput" popover-placement="bottom">
    <label for="title"><h4>Title:</h4></label>
    <div class="input-group">
        <input id="title" type="text" class="form-control" ng-model="tempTitle" placeholder="Assessment title..">
          <span class="input-group-btn">
            <button ng-disabled="tempTitle==assessment.title|| tempTitle==''" ng-click="updateAssessment();" class="btn btn-default" type="button">
                <span title="Save assessment title" class="glyphicon glyphicon-save" aria-hidden="true"></span>
            </button>
          </span>
    </div>
</div>
```

Figure 62: Popover Directive Usage Snippet

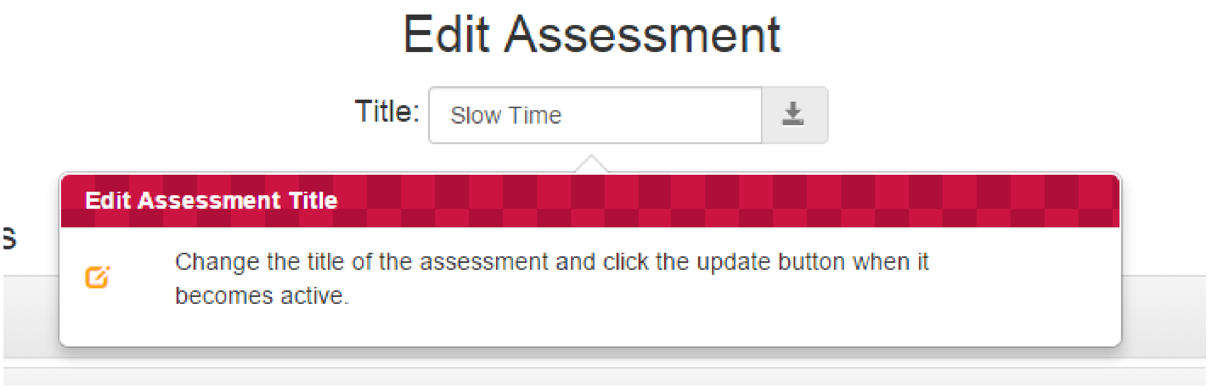
In this example the popover-html attribute can take a variable on the controllers $scope and the value of 'popoverTemplates.assessmentTitleInput' is:

- parts/admin/popoverTpls/assessments/assessmentTitleInput.html

The full source code of the directive that uses the Bootstrap popover can be in the below repository:

- https://github.com/J-Morris676/SWPAssessment/blob/master/public/js/directives/customPopOverDirective.js

The popover for the mark-up in figure 62 can be seen in figure 63.


Edit Assessment

Title: Slow Time

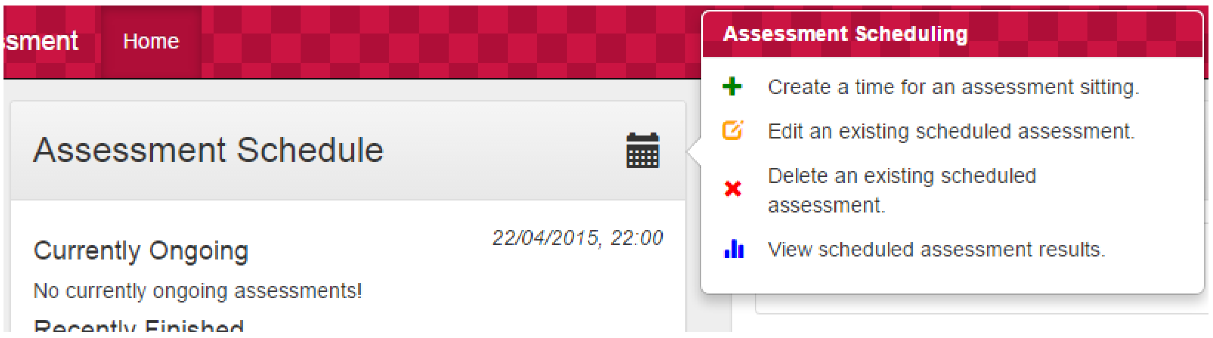
Edit Assessment Title

Change the title of the assessment and click the update button when it becomes active.

Figure 63: Edit Assessment Popover

This appears when the user hovers over the input box and informs them of what this input box does and how to update the title once text has been added to it.

A more rich and useful example is for the assessment schedule panel on the Trainer home page where it displays a list of operations that can be made by clicking this item:



Assessment Schedule

Currently Ongoing

22/04/2015, 22:00

No currently ongoing assessments!

Recently Finished

Assessment Scheduling

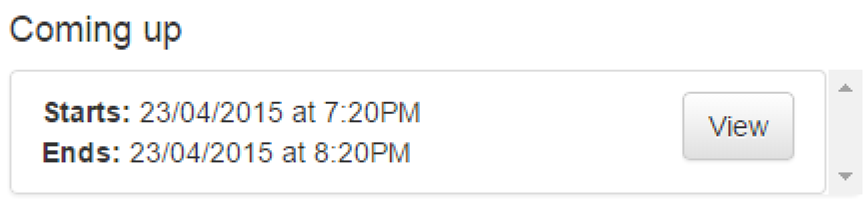
- Create a time for an assessment sitting.
- Edit an existing scheduled assessment.
- Delete an existing scheduled assessment.
- View scheduled assessment results.

Figure 64: Assessment Scheduling Popover

Sitting an Assessment (Students)

Student requirement 1 is the ability to sit an assessment they're scheduled to sit between the start and end time. As seen before, the Student is presented with an assessment schedule item that they can use to navigate to a landing page:

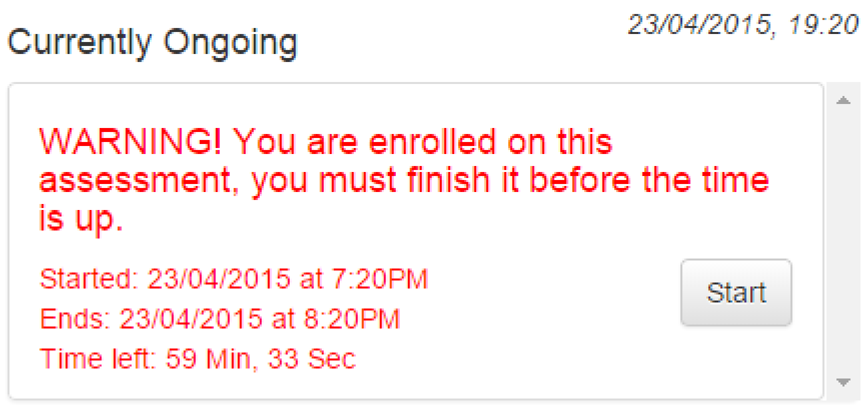
Schedule Item


Coming up

Starts: 23/04/2015 at 7:20PM
Ends: 23/04/2015 at 8:20PM

Figure 65: Scheduled Item (Coming up)

Schedule Item Warning


Currently Ongoing

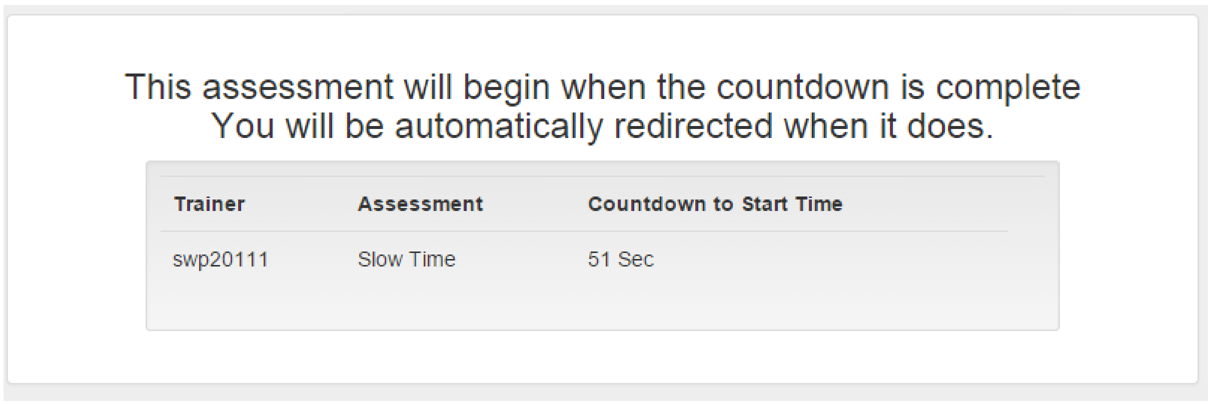
23/04/2015, 19:20

WARNING! You are enrolled on this assessment, you must finish it before the time is up.

Started: 23/04/2015 at 7:20PM
Ends: 23/04/2015 at 8:20PM
Time left: 59 Min, 33 Sec

Figure 66: Scheduled Item (Warning)

Waiting Page


This assessment will begin when the countdown is complete
You will be automatically redirected when it does.

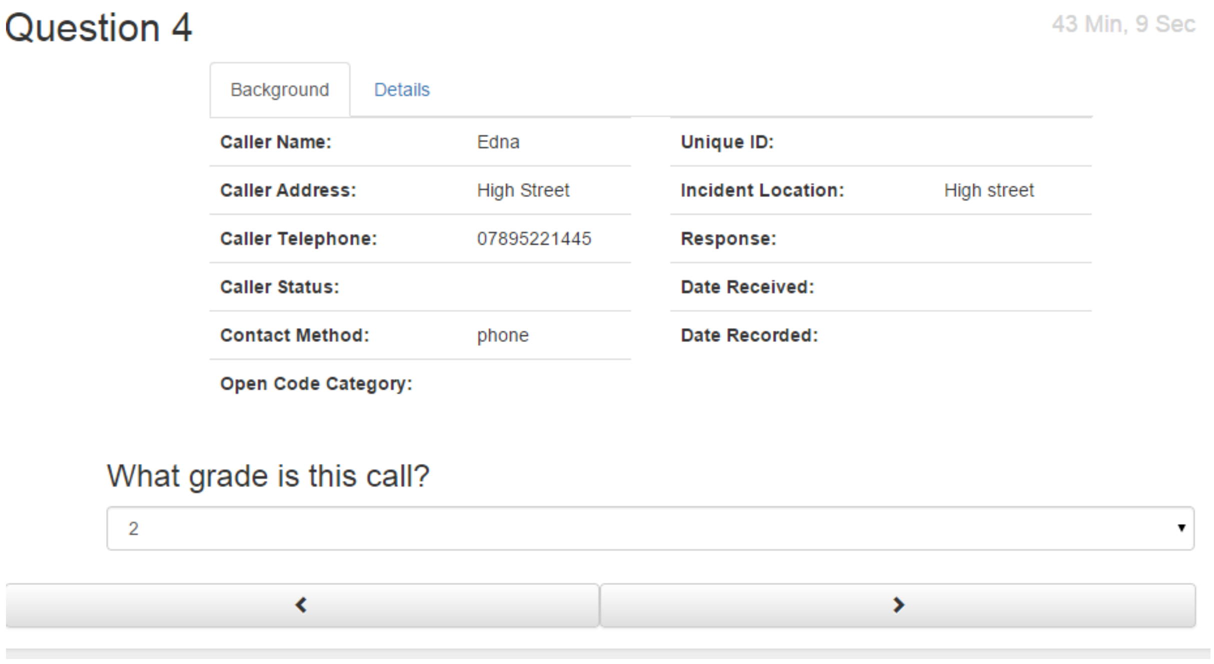
Trainer	Assessment	Countdown to Start Time
swp20111	Slow Time	51 Sec

Figure 67: Scheduled Item (Waiting page)

The above screenshots display how easy it is to start an assessment when it has started and clearly warns the user that they need to sit it within a given period of time.

Call Log Question

The below screenshot shows a call log question being asked in the assessment, the point of this question is to use the call log information to determine an answer to the question. This implementation utilises a service that updates their assessment everytime they navigate question keeping a log of what they have answered if they were to not make it to submission.



Question 4

43 Min, 9 Sec

Background | Details

Caller Name:	Edna	**Unique ID:**	
Caller Address:	High Street	**Incident Location:**	High street
Caller Telephone:	07895221445	**Response:**	
Caller Status:		**Date Received:**	
Contact Method:	phone	**Date Recorded:**	
Open Code Category:			

What grade is this call?

2

Figure 68: Call Log Question View

Testing

When designing and implementing the system extensibility was always a key factor to consider, there is evidence of this throughout this report. This has also been carefully considered by the testing methods that I have used, it was important to have a structured and standardised way of testing the application at different iterations. This ensured a new implementation had not affected other aspects of the system. As it was a requirement for the system to be usable on desktop and mobile devices, two cases have been considered where necessary and the test specifications can be seen below:

Desktop Testing

- **Operating System**: Windows (8.1)
- **Device**: Laptop
- **Peripherals**:
  - Mouse
  - Native laptop keyboard
  - Native laptop monitor
- **Web Browser**: Google Chrome (v. 42.0.2311.90)

Mobile Testing

- **Operating System:** Android (v. 5.0)
- **Device:** LG G3-D855
- **Web Brower:** Google Chrome (v. 42.0.2311.109)

Test Cases

To test the full functionality of the system as a user, each step of the system needed to be tested using a set of conditions which will specify if that aspect of the system is functional. In the case of this system test oracles were used in the form of requirements that were set at the beginning of this report. The template form and table used for all of the test cases can be seen below:

Database URL:
Application URL:
Prerequisites:
Web Browser:
Device:
Date:

ID	Step	Expected Result	Actual Result	Pass/Fail	Remarks
0					
1					

Figure 69: Test Case Template

As of 26/04/15 all of these test cases pass with no functional issues and all test cases can be found in the appendix.

Usability Testing

At some points throughout the project's iterations the usability of the system was considered and after aspects of the system was found to be fully functional usability tests were used using a sample of users to consider how usable the application was.

As there was a requirement for the application to be usable on smaller devices two types of tests were conducted and a different 'magic 5' was used for the sample size for each device. Nielsen's 'magic 5' comes about from studies where different usability problems were found by different users, a graph of from this study is shown in figure 70.

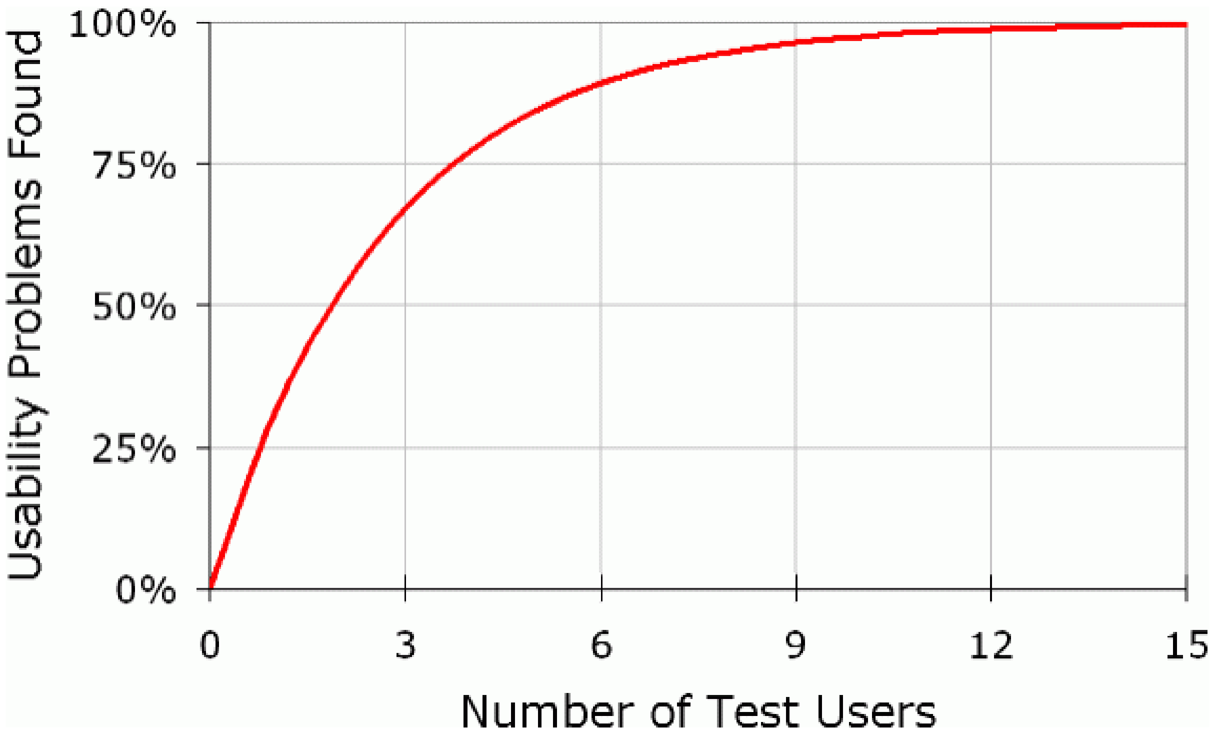


Figure 70: Usability Problems over Number of Test Users

Another reason for this sample size is due to the number of time and people (i.e. resources) I had available but also has a lot to do with the theory that after 5 test users the feedback begins to get very repetitive.

The tests on the users were entirely task-based using instructions which were read out to the user, it was then entirely up to the user to navigate to and perform the actions required. Any ambiguity the user found in the system was noted for future use and considered as a 'ticket' for potential future implementations.

The table structure for logging users actions can be seen below and a full listing of all usability tests conducted can be found in the appendix.

UserNo: **Test Type**: Desktop/Mobile			
Task No.	**Task**	**Comments**	**~ Time Taken (s)**

Figure 71: Usability Test Template

Usability Testing Findings

There were many interesting findings during the usability tests and the whole process was very productive and it really highlighted issues in the system that I simply couldn't realise as I implemented them myself. The testers highlighted a total of 3 usability issues that have been added for future work and these issues were repeated among multiple users. After the fourth test no new usability issues were found which supports the 'magic 5' test count selection. The issues are shown below:

1. When creating a student or assessment schedule, the collapsible 'add' menu does not collapse anywhere on the panel, only on the 'collapse' button
2. Three users felt that they could be able to click on the popovers that appear which would navigate them to the resource
3. It didn't seem to be obvious enough that the collapse buttons for versions and questions when editing assessments could be used to show more information about the respective version/question

Unit Testing

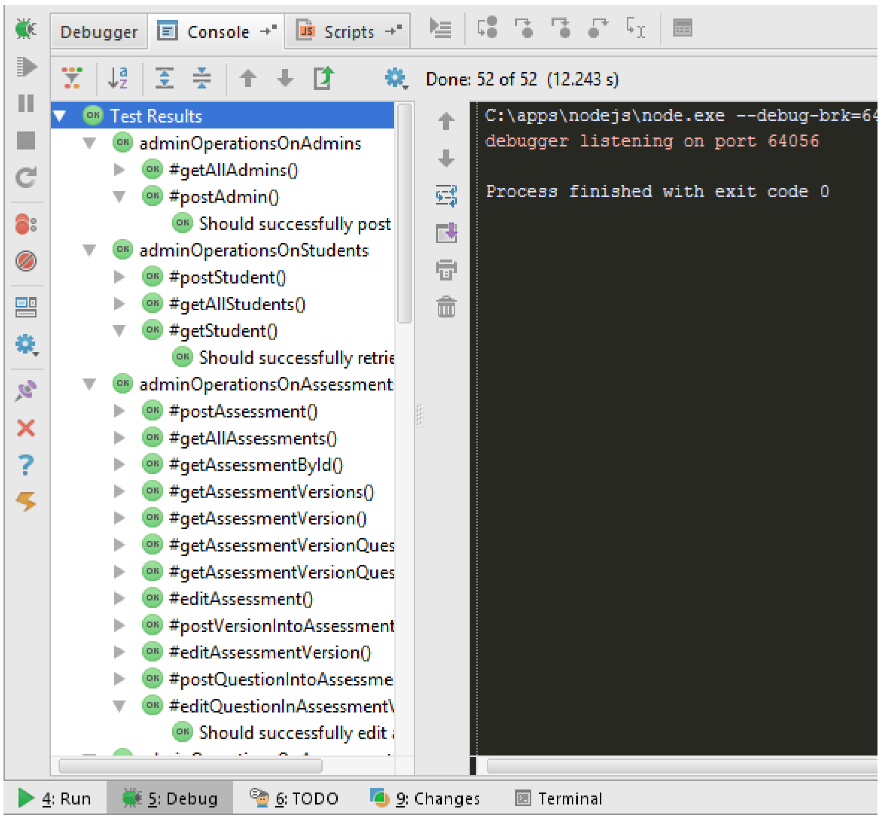
The API, described in the service implementation section of this report, contains almost 50 RESTful services therefore it was vital to ensure each service worked as expected without having to manually check with each new implementation.

To overcome this I decided to write a set of unit tests that would emulate a browser and fire HTTP requests at the REST API asserting the results returned from it. The below packages were used to assist in creating these unit tests:

- Mocha (http://mochajs.org/) - The most popular test framework for Node.js is and has all the features you'd expect from a test framework (describing of tests, assertions, hooks etc.).
- Supertest (https://github.com/visionmedia/supertest) - Can be used to make HTTP requests and test the responses - powerful when partnered with Mocha.
- Mongodb (https://www.npmjs.com/package/mongodb) - The raw driver for MongoDB, used to insert data into the database to test the API with (e.g. create an admin before performing tests)

This setup allowed me to ensure all of the requests made from the client web application receive valid responses when the services were updated during any iteration, all of the tests can be run in under 10 seconds and have proved extremely useful.

Another advantage of using Mocha is it's integration with the IDE that I used, Webstorm. All tests can be run from within the IDE and can even run in debug mode so I can quickly figure out as to why a test may not be working as expected. The UI of the unit testing can be seen in figure 72 and provided a very quick and easy way of setting the tests up. The server is started using the 'Run' tab and the 'Debug' tab is used to run the Mocha tests.



Done: 52 of 52 (12.243 s)

```
Test Results
  adminOperationsOnAdmins
    #getAllAdmins()
    #postAdmin()
      Should successfully post
  adminOperationsOnStudents
    #postStudent()
    #getAllStudents()
    #getStudent()
      Should successfully retrie
  adminOperationsOnAssessment
    #postAssessment()
    #getAllAssessments()
    #getAssessmentById()
    #getAssessmentVersions()
    #getAssessmentVersion()
    #getAssessmentVersionQues
    #getAssessmentVersionQues
    #editAssessment()
    #postVersionIntoAssessment
    #editAssessmentVersion()
    #postQuestionIntoAssessme
    #editQuestionInAssessment\
      Should successfully edit
```

```
C:\apps\nodejs\node.exe --debug-brk=64
debugger listening on port 64056

Process finished with exit code 0
```

Figure 72: Webstorm Test View

As seen in the above screenshot, a test structure has been used to ensure all of the below factors are considered:

- Authentication for each user type for each operation
- Response codes for each user type for each operation
- Response bodies for each user type for each operation

Tests are divided into two different test files for each user type covering each operation of the API, the structure for each file is outlined below:

- User
    - Operations on Admins
    - Operations on Students
    - Operations on Assessments
    - Operations on Assessment Schedules

On the last development iteration of the projects, a total of 52 tests passed successfully. The full implementation of both test files can be found in the below directory:

- https://github.com/J-Morris676/SWPAssessment/tree/master/test

Evaluation

To evaluate whether the system implementation constructed was effective, this report will evaluate the projects core functional requirements determining whether each requirement has been met. The evidence columns in the below tables briefly explain what the system does to tackle the requirement. Detailed information about the evidence can be found in the implementation section in this report and the functionality can be found on the client application implemented during this project.

The tables below highlight all of the requirements and whether the requirements were met based on the requirements acceptance criteria.

Meeting Requirements

Both User Requirements

Requirement	Requirement Met?	Evidence
The system must allow the user to sign in to the system using their credentials and direct them to their respective system area.	Yes	The system does this by identifying which collection in the database the signing in user belongs to. Using this information the client application navigates them to their respective area of the system.
The system must allow the user to sign out of the system.	Yes	The system does this by removing a cookie that was set up when signing in. The previously set up cookie is no longer accepted after this and the session has been ended.

Trainer Requirements

Requirement	Requirement Met?	Evidence
The system must allow the user to create, edit, view and delete assessments, versions of assessments and the questions within each version	Yes	The system does this by providing services to perform these operations which can all be controlled through the client application with a session setup as a Trainer. A page on the application is also available satisfying the acceptance criteria for this requirement.
The system must allow the user to create, edit, view and delete assessment Schedules	Yes	The system does this by providing services to perform these operations which can all be controlled through the client application with a session setup as a Trainer.
The system must allow the user to create, edit, view and delete	Yes	The system does this by providing services to perform these operations which can all

student accounts and view any results have in assessments		be controlled through the client application with a session setup as a Trainer.
The system must allow the user to edit their own details (first name, last name & password)	Yes	It is possible to use the client application to alter the details of the signed in user (Only possible as Trainer), services are used to make these changes in the database.
The system must allow the user to download a document displaying the results of Students in completed assessments	Yes	Results are available to the Trainer by either directly accessing the Student and viewing there results or by accessing the scheduled assessment to see all results of enrolled Students. Downloading the PDF report can be done by accessing Student results either of the above ways.

__Student Requirements__

Requirement	Requirement Met?	Evidence
The system must allow the user to sit an assessment that they have been scheduled to sit at the correct time start time	Yes	When a trainer has set an assessment schedule, all enrolled Students can sit the assessment. Manual and unit tests cover the acceptance criteria's and work as expected.
The system must allow the user to submit an assessment during the period of time between a scheduled assessments start and end time	Yes	It is possible to sit the entire assessment and submit the answers providing it is done so by the end of the assessment, other operations defined as acceptance criteria also work as expected.
The system must allow the user to visit the results of their completed assessments	Yes	The results of an assessment is available instantly after the submission of their answers, it can also be viewed at any time by using the client application to look at their 'completed assessments' page
The system must allow the user to download a document displaying limited results of their completed assessment	Yes	Similar to how the above requirement was met, it is possible to download a PDF document displaying the results the moment they submit their answers.

These tables show how each and every functional requirement gathered throughout the project have been met and the test cases which are designed to highlight these issues passed with no issues.

Intuitive User Interface

The Nielsen heuristic's that I followed throughout the development of the application resulted in positive usability results and feedback. New users found the system easy to use and consistent throughout, resulting in it being easy to learn how to use on either desktop or mobile devices. Feedback also showed that the use of glyphicons (recognisable icons and symbols) helped users who take a visual approach in using applications as it helped them disambiguate and decide on where they needed to click quicker making it more efficient for them to use.

The intentionally consistent design made it very memorable to users as during the tests it was found that once they had used one aspect of the system, any hesitation they previously had in a similarly looking area decreased every time. The popovers with visual aids and small text entries led to two new users to actually calling the system 'intuitive' showing how the considerations taken throughout the duration of the project has really paid off.

SUS Questionnaire

To gain a better understanding of how; effective, efficient and satisfying the application was to use, in comparison with other systems, I asked the tester to fill out a System Usability Scale (SUS) form immediately after each usability test. The questionnaire consists of the following ten statements which the testers rated from Strongly Disagree (1) to Strongly Agree (5).

1. I think that I would like to use this website frequently.
2. I found this website unnecessarily complex.
3. I thought this website was easy to use.
4. I think that I would need assistance to be able to use this website.
5. I found the various functions in this website were well integrated.
6. I thought there was too much inconsistency in this website.
7. I would imagine that most people would learn to use this website very quickly.
8. I found this website very cumbersome/awkward to use.
9. I felt very confident using this website.
10. I needed to learn a lot of things before I could get going with this website.

I have used the results provided by the testers to gather a series of composite measures of overall usability calculated using the steps defined by SUS scoring shown below:

1. For each **odd** item number in the form, subtract 1 from response
2. For each **even** item number in the form, subtract response by 5
3. Sum all 10 numbers up and multiply them by 2.5 to get a scale from 0-100

Note: Numbers in brackets are the values after scaling the given score

	Statement Numbers										
Tester ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**SUS Score**
1	1 (0)	4 (1)	5 (4)	2 (3)	5 (4)	1 (4)	5 (4)	1 (4)	5 (4)	2 (3)	77.5
2	5 (4)	2 (3)	4 (3)	2 (3)	5 (4)	1 (4)	5 (4)	1 (4)	4 (3)	1 (4)	90
3	5 (4)	4 (1)	5 (4)	3 (2)	5 (4)	1 (4)	5 (4)	2 (3)	5 (4)	1 (4)	85
4	5 (4)	4 (1)	5 (4)	4 (1)	5 (4)	1 (4)	5 (4)	1 (4)	5 (4)	2 (3)	82.5
5	4 (3)	1 (4)	4 (3)	2 (3)	4 (3)	1 (4)	4 (3)	2 (3)	3 (2)	1 (4)	80
										Average:	83

Figure 73: SUS Results Table

The average SUS score from 500 studies is a 68 and the average results for my system from all five testers comes out at 83 meaning that my SUS score is well above average. This result seems very high and there are various factors to consider before I can consider it a pleasant user experience:

- **Application size**: The questionnaire was based on experience with the application and generally how easy the user got on with using it for the first time. As the application was fairly small (an individual, 11 week project) there are less features on the system to interact with making it generally easier to use compared to a larger system. As the system may grow in size I believe the SUS score will also come down as there will be more aspects of the system for a new user to become confused at.

- **Potentially biased feedback**: As much as I hoped that there was no biased feedback in the questionnaires I had to consider that the testers may have been conscious about any low scores they gave, no matter how much they were told to be honest.

- **How to interpret the score**: A percentile rank graph exists that can be used to grade a SUS score on a curve. Using this graph I could conclude that the application received an A, being in the top 10% of scores from the studies.

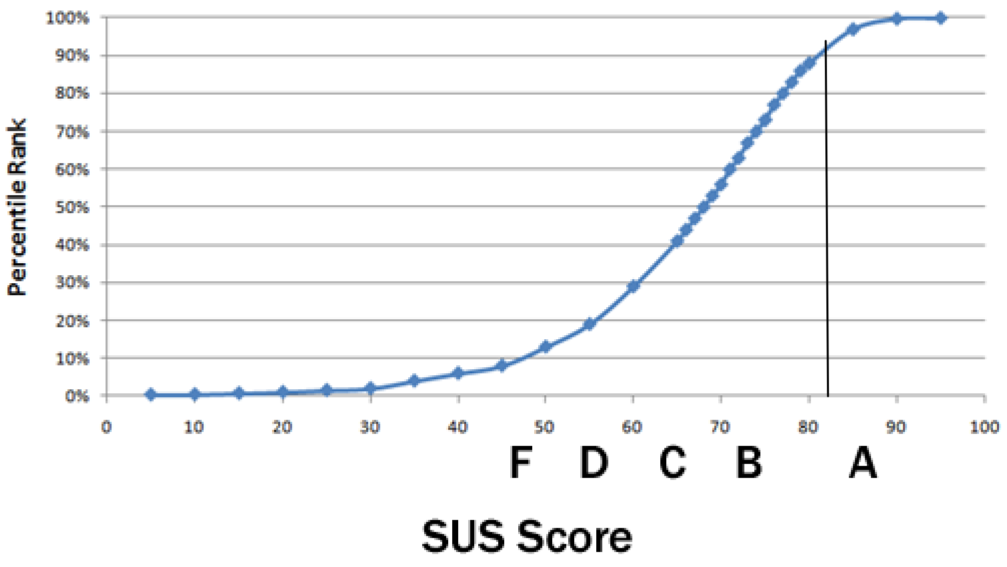


Figure 74: SUS Score Percentile Graph

Regardless of these potential flaws in the SUS questionnaire method, the individual feedback and overall SUS score analysis left me satisfied with the user experience. The system was verbally complimented several times throughout the testing day and with some minor fixes mentioned in the usability testing section, I believe this application will be very easy to use in the near future. All questionnaires filled out by the testers can be found in the appendix.

Future Work

Even though all of the core requirements the Client has provided have been successfully implemented, there could be many potential additions in the future. These additions were discussed in the final meeting I had with the Client and are listed below. As I already had a good idea about how to implement most of the additions, I have described my potential solution for each addition.

Additional Feature	**Description**	**Potential Solution**
Mobile application implementation	With the introduction of tablets into the office it would be quite useful to have a mobile application created that could be used to carry out the same operations. As the application was created using a standard HTTP API and the mobile application could simply use the same server as the web application.	My solution to this would be to create an Android application that will either be available on the Google Play Store or to install them directly to each device using the resulting APK file. If there was a potential requirement of having an application available for other mobile operating systems as well I would consider using a tool called PhoneGap (http://phonegap.com/) which allows the developer to create mobile apps using HTML, CSS, JavaScript and an API provided by PhoneGap which can then be compiled to use on multiple device Operating Systems: • iPhone • Android • Blackberry • Windows Phone
More parameters for question types	It is currently possible to set three question types for testing and in the future it would be nice to add some further options for Trainers to select that would affect the way they appear and/or are marked. An example of this is to allow the Trainer to choose the distance (between 0-1) of string marking.	I have implemented the system in such a way that features like this become fairly easy to include, the following basic procedures would have to be followed to implement this: • No schema changes have to be made as question schemas are arbitrary • Update any actions on the server that these new parameters might require • Update the client application to make the parameters available to choose and any other actions that may be required to update
More parameters for assessment schedules	Similarly to above, a feature to choose options for assessment schedules would improve the	This could be implemented very similarly to the solutions described above (For question

	system by making it possible to toggle how much assessment result information will appear to Students	type parameters) but would include storing a flag in the assessment schedule documents to determine how much of the results to the Students
Customisable usage assistance	Features like popovers are great for new users of the system as it becomes very easy to figure out what a component does but it could potentially become annoying for regular users. It would be nice if the user had the option to switch these off to prevent them appearing every time a component has been hovered over.	This could be implemented very similarly to the solutions described above (For question type parameters) but would include storing a flag in a user collection to determine whether to show them on the client and make it a selectable feature on the client application**.**
Resource paging	The system works great at the moment with between 15-20 items per resource but as this grows there will be an issue with querying, data transfer and client rendering. This will require a method to group resource items by something (i.e. Students to a Trainer) or to implement paging onto all resources.	My solution to this would begin by improving the API making it utilise paging features which would make it easy for a client to get a given number of items with paging information as HTTP headers. A library already exists called express-paginate (https://github.com/expressjs/express-paginate) that can be used to implement these features whilst querying the database in such a way to return this information for the client. An example HTTP request with pagination can be seen below: • GET /resources/assessments?pageSize=10 &pageNumber=2
Going beyond a proof of concept	The system in its current state is only a proof of concept, only available for demonstrations and tests.	To deploy this application it would involve working internally within South Wales Police and deploying onto local servers within their firewall. I do not have the security clearance to fully consider all factors of this problem.
Real time update for trainers of student assessment activity	As an assessment is going on it would be beneficial if a Trainer can see Student activity in real time by showing them Student answers as they are answering them.	As the system already stores user results as they navigate questions I only need to consider how to update the Trainer's client with real time updates. There are two possible solutions to this**:** • Continuous requests - an easy but

		resource demanding solution that will make requests to the server every x seconds to find any Student progress updates • Bidirectional event-based communication - An NPM package called Socket.io (http://socket.io/) looks like an exciting solution to this which allows real-time two-way communication between client and server (Commonly used to build instant messaging applications)
Statistical features on completed assessments	There are graphs showing the percentage of Students who started and ended an assessment and a graph of Students and their results. It is possible to provide more than this by providing various visualisation based on data collected from around the system.	A solution to this could be a new page that's sole purpose is to have users select what to show on a particular visualisation (e.g. percentage of passes on an assessment)

Reflection on Learning

This project was a great opportunity for me to familiarise with state-of-the-art Software Development and gain practical experience in this area. My University experience primarily focused on small projects with a single requirement, whereas this project allowed me to branch out to new and upcoming frameworks and technologies to turn what seemed like an impossible task into something that could be implemented quite comfortably.

I have become much more aware of the importance of using existing frameworks and libraries to minimise the amount of time spent writing it myself. Researching and integrating existing software packages is a skill in its own right and I have learnt what seems a very simple, but very important lesson from this:

"Research existing implementations before writing your own."

Whether that implementation is an existing library or a solution on Stack Overflow, understanding other ideas and integrating them into your own solution is extremely helpful when implementing larger software projects. It was often quite difficult and frustrating at times attempting to identify how all of the technologies work together and how to use them effectively but it was well worth the learning curve. The overall development environment was very enjoyable to use and I look forward to further exposure in the future.

The use of MongoDB in this project was a risky decision considering my lack of exposure to anything outside of SQL. I have only positive things to say about my experience with it but in my opinion, developers should be aware of the luxuries provided by a standard RDBMS and develop around this sensibly. For example, one luxury I sorely missed was the "all or nothing" proposition of RDBMS transactions.

On reflection, I am very happy with my decision to implement this project as a web application as it appears that web applications have superseded desktop applications. This is largely due to the capability of HTML5 and its ability to run in any web browser on any device. The future of software applications seems to be in mobile development and web applications and this exposure has even resulted in me securing a permanent role in a company primarily using Node.js and MongoDB.

The interaction with the Client was also a very valuable experience as it helped develop my professional skills. It was also interesting to be involved in this experience as it was often surprisingly very difficult at times to establish exactly what the Client required. I am happy that, having effectively identified solutions despite some ambiguous discussions the outcome was an overall success.

If I were to do the project again I would spend a little more time on the question types, particularly the free text style questions as although it currently satisfies the requirement it would be nice to possibly determine synonyms and some abbreviations to mark the answer correct based on further parameters provided by a trainer and thus enable them to be a little more creative when designing their assessments.

Whilst implementing the system I found through research a useful superset of JavaScript called TypeScript which tightens up the features of JavaScript and allows you to define typed features of object-orientated programming such as interfaces and classes. Learning from this, I would certainly use this in any future project as typing all variables makes it much easier to understand.

Project Management

Moderator feedback from my initial report highlighted the challenging time plan necessary and hence the importance of frequently reviewing the progress of the project to prevent falling behind schedule. It is for this reason that I decided to strictly manage the progress of the project at each stage and will now discuss the approaches that were used.

Weekly Reports

It was often difficult to meet with my Supervisor every week to discuss the project and early on we agreed that I would write weekly updates that would be made public using Google Docs. This allowed my Supervisor to include inline comments to the report to provide support and any suggestions.

When meetings were arranged, it was much easier to update my Supervisor by referencing sections in the report which contained diagrams and images some of which have been adapted into this final report.

This use of logging work also ensured I was up to date with the work plan as I was eager not to fall behind. The updates of the report start at week 1 and finished when the implementation stage of the project was completed (Easter break week 2) meaning a total of 11 weeks of work was recorded and has become an extremely useful resource whilst writing this final report.

The document is entirely public and can be accessed using the below link:

- https://docs.google.com/document/d/1mOfn7Nt0FDyH3OToEz1IG_EP2ZKlZ8XhmerfqpuEHwU/edit?usp=sharing

Agile Approach

Due to the nature of the project, research led me to realise the benefits of an agile approach compared with a waterfall approach. In order to provide a system that can be demonstrated at the end of the project and regardless of the number of requirements that are met, an agile approach is safer, particularly when considering the short time frames involved. To satisfy the documentation requirement of an agile approach I maintained a report as mentioned in the previous section which documents the design and implementation during each iteration.

The agile approach is a development methodology and mainly applied to weeks 4-9 in my overall project work plan. The diagram below shows how this iterative procedure of agile software occurred (in my case it was weekly iterations due to the short duration of the project). During these iterations I worked through the requirements and at the end of each iteration I delivered, demonstrated and gathered feedback from the Client which helped towards the next iteration of development.

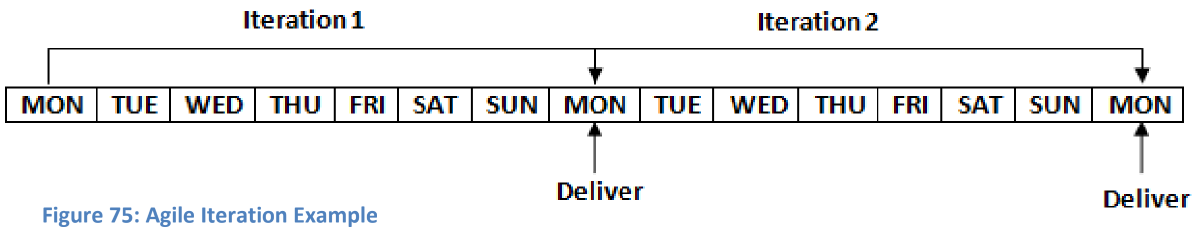


Figure 75: Agile Iteration Example

I believe my decision to use an agile model was better than using the waterfall model as I had an application by week 9 that was ready for demonstration. The remaining weeks were used to implement remaining requirements and write this report. Therefore, by adopting this development approach I was able to develop my personal and professional skills while meeting the Client's expectations and requirements.

References

1. Adobe, 2015. Captivate [online] Available at: <http://www.adobe.com/uk/products/captivate.html> [Accessed 28 April 2015].

2. Bevan, N., Barnum, C., Cockton, G., Nielsen, J., Spool, J. and Wixon, D., 2003. *The "magic number 5": is it enough for web testing?*. ACM New York.

3. Bootstrap, 2015. Grid Options [online] Available at: <http://getbootstrap.com/css/#grid-options> [Accessed 28 April 2015].

4. Bower.io, 2012. Bower A package manager for the web [online] Available at <http://bower.io/> [Accessed 28 April 2015].

5. Brooke, J. 1986. SUS - A quick and dirty usability scale. [online] Available at: <http://cui.unige.ch/isi/icle-wiki/_media/ipm:test-suschapt.pdf> [Accessed 27 April 2015].

6. Cinergix Pty, 2008-2014. Creately. [online] Available at: <http://creately.com/diagram-type/use-case> [Accessed 28 April 2015].

7. GOV.UK, 2015. Data Protection - GOV.UK. [online] Available at: <https://www.gov.uk/data-protection/the-data-protection-act> [Accessed 29 April 2015].

8. Jetbrains, 2015. Webstorm [online] Available at: <https://www.jetbrains.com/webstorm/> [Accessed 28 April 2015].

9. Kallidus, 2015. LMS Learning Management [online] Available at: <http://www.kallidus.com/Products/Kallidus-LMS-Learning-Management/> [Accessed 28 April 2015].

10. Leenheer, N. 2015. html5test. [online] Available at: <https://html5test.com/> [Accessed 28 April 2015].

11. MEAN.IO, 2014. MEAN.IO [online] Available at: <http://mean.io/> [Accessed 28 April 2015].

12. Microsoft, 2012-2015. TypeScript [online] Available at: <http://www.typescriptlang.org/> [Accessed 28 April 2015].

13. MongoDB, inc, 2011-2015. Aggregation [online] Available at <http://docs.mongodb.org/manual/aggregation/> [Accessed 28 April 2015].

14. MongoDB, inc, 2011-2015. MongoDB CRUD Operations [online] Available at <http://docs.mongodb.org/manual/crud/> [Accessed 28 April 2015].

15. Nielsen, J. 1995. 10 Usability Heuristics for User Interface Design. [online] Available at: <http://www.nngroup.com/articles/ten-usability-heuristics/> [Accessed 29 April 2015].

16. Nielsen, J. 2000. Why You Only Need to Test with 5 Users. [online] Available at: <http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/> [Accessed 29 April 2015].

17. Node.js, 2015. Node.js [online] Available at: <https://nodejs.org/> [Accessed 28 April 2015].

18. Paralect, 2013-2015. Robomongo [online] Available at: <http://robomongo.org/> [Accessed 28 April 2015].

19. Sahni, V. n.d.. Best Practices for Designing a Pragmatic RESTful API. [online] Available at: <http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api> [Accessed 28 April 2015].

20. Sauro, J. 2011. Measuring Usability With The System Usability Scale (SUS). [online] Available at: <http://www.measuringu.com/sus.php> [Accessed 28 April 2015].

21. Shaw, D (Home Affairs Correspondent), 2015. Police forces all face major budget cuts. *BBC News*, [online] 7 March. Available at: <http://www.bbc.co.uk/news/uk-31771456> [Accessed 28 April 2015].

22. Stack Overflow, 2015. 2015 Developer Survey [online] Available at: <http://stackoverflow.com/research/developer-survey-2015> [Accessed 28 April 2015].

23. Surguy, M. 2013. Bootsnipp [online] Available at: <http://bootsnipp.com/> [Accessed 28 April 2015].

24. Umbel, Ellis and Mull, 2012. NaturalNode [online] Available at: <https://github.com/NaturalNode/natural> [Accessed 28 April 2015].

25. Winkler, W. E. (1990). "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage" Proceedings of the Section on Survey Research Methods (American Statistical Association): 354–359.

26. Yujian, L., Liu Bo. 2007. A Normalized Levenshtein Distance Metric. IEEE Computer Society Washington.

Appendix

Appendices can be found in the zip folder alongside this report and are numbered the same as below.

1. Design Sketches
2. Sample Results Documents
3. Desktop Screenshots
4. Mobile Screenshots
5. Test Cases
6. Usability Tests
7. SUS Questionnaires
8. Weekly Update Document
9. Source Code