Group Project Report - Argumentation for Social Networks-style E-Learning

Alan Vey, Lu Ningyuan, Crystal Pang, Naomi Bassett, Octavian Tuchila, Taehyun Lee

12th January 2015

Contents

Chapter 1

Executive Summary

1.1 Elevator Pitch

RadiCalc is a web based social networking E-Learning platform designed to improve communication between students and lecturers about their university courses. Users are able to create topics which they can invite other users to subscribe to. Each topic has space for open discussions about general issues and technical course material. Course administrators can create assessments for students to complete so they are better prepared for exams. The site also provides users with proficiency graphs for each of their subscribed topics. Our application provides a clearer picture to lecturers of how well their students are understanding the course, and also help students to improve their proficiency through discussion.

1.2 Intended Users

The initial intention of this project was to develop a platform which was to be used, maintained and distributed exclusively within the Imperial College Department of Computing. Therefore the intended users are the staff and student members of the department and meeting their needs will be the top priority. However, should the application be well received by these users, RadiCalc could easily be used outside the department or even outside the college.

RadiCalc allows the lecturers to gain feedback on their performance directly from their pupils, which in turn, allows them to help students improve their performance further. It is also a convenient medium in which they could upload materials for students and assess the students' current understanding of the course (by creating assessments). Each student's understanding is quantised as their proficiency in that topic. This is important in identifying key areas for lecturers to improve upon by providing them with them with a simplistic, informative overview.

For students, RadiCalc is a convenient medium for discussion, where accuracy and specificity of the answers can be debated. The site is also a source of sample answers as provided by lecturers which greatly aids exam preparation. Moreover, it provides an accurate indicator of their current performance by tracking their academic activities on the website (such as answering questions in an open discussion or completing an assessment). These can be useful for students to identify which areas of a course they need to improve upon.

Chapter 2

Introduction

2.1 E-Learning Platforms

In order to better understand the requirements of our own project, we researched a wide range of existing E-learning platforms. Currently, many E-Learning platforms are very limiting in what they allow users to do. They are either tools to aid teachers in creating topics or they are online school environments, with predetermined topics and materials aimed at helping students to learn a new topic or to provide more material on a topic they are struggling with.

Platforms for creating topics allow teachers to upload materials such as videos, notes and exercises on a topic of their choosing and so these platforms tend to offer a wide range of topics, such as Siminars or SlideWiki. These platforms are aimed at teachers and students, but they limit any interactions between teachers and students to comments students make about a topic. The teachers do not interact with the students, as they upload materials they feel are appropriate. Due to this, the only feedback students may receive are the answers to set exercises.

The second category of E-Learning platforms is where the creators of the website upload topics themselves, where the website is aimed specifically at students. Udacity and Skillshare have taken this approach. These platforms also limit how much interaction there is between teachers and students and once the website is running, expansion of topics is much slower than the other platforms.

There are several platforms that have implemented other notable features. For example, Aplia has a system for automatically grading assignments and providing detailed feedback. Teachers can set how severe the grading is and it allows for much faster feedback to the students. Some platforms, such as Claroline and Sakai, have implemented a form of social media, allowing for student-to-student instant messaging and also forums where students can pose questions and answers they have. Blackboard has taken a different approach to E-Learning and has instead provided online resources to be integrated into the school environment to aid teachers and students in managing and tracking their assignments. They also offer analysis on the achievements of the students after the topic has ended, allowing teachers to modify their curriculum accordingly.

We created a survey on SurveyMonkey asking for people's opinions on current E-learning platforms. We asked them which features were useful and which ones were lacking in the platforms. We also asked how useful they would find features that we were planning on implementing for our own WebApp. Some of the results of this survey are included below. The full survey is provided in Appendix A.

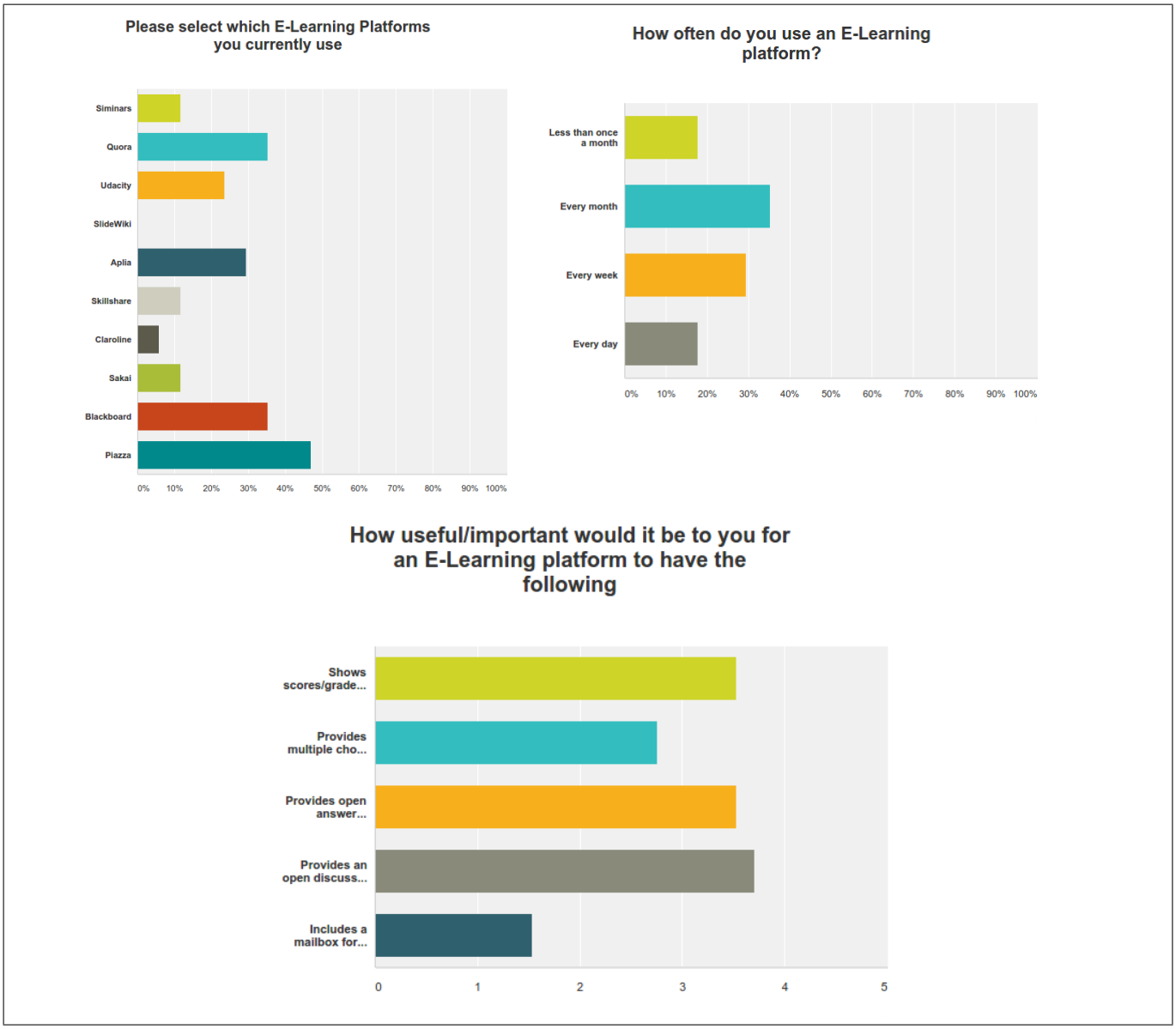


Figure 2.1: Results of SurveyMonkey survey about E-Learning platform use

The surveys have been useful in providing information about which platforms people currently use, how often they are using them and how important some basic features would be. This aided in the feature prioritisation process (see Section 3.1.2) and gave us an insight into how often we might expect users to be interacting with our website.

2.2 Agenda

Currently students of DoC only receive feedback from lecturers after completing coursework meaning that students often do not know where the gaps in their knowledge are until they begin revising for exams. RadiCalc allows lecturers to set up assessments within a topic so students can gauge their proficiency in topics more frequently and continuously. The intention is for assessment questions to be either multiple choice, possibly with justifications, or written answers and assessments can contain a mixture of these question types.

2.3 Project Objective

By the end of the project we planned to have a working web application where students and lecturers from Imperial can register to the site and subscribe to some default topics. When a user first logs in they should be able to create a topic, for which they will become the administrator. When other users log-in, they should be able to see topics and subscribe to them. The administrator should then be able to accept or reject the subscription. If a user is subscribed to a topic they should be able to see the uploaded materials and any corresponding assessments for which they can upload their answers for set coursework and assessed exercises.

Users should have a profile page showing graphical representations of their proficiency and participation within their topics. The graphs should show the overall proficiency of the student within the topic and when the user goes into a subtopic, the graph should expand to show the proficiency of the student in each chapter within a module. Users should have the ability to hide their graphs from other students should they wish, giving them more control over their profile and privacy, but the administrator for the topic will always be able to see the students graph for that topic. Users should be able to compare proficiency graphs between themselves so they can check compatibility between potential group members. Lecturers should also be able to compare graphs for a year group, which provides feedback on which topics were particularly well received and which ones were not, so the lecturer can modify their syllabus or teaching methods accordingly.

Each topic should have three different types of question feed ; a technical feed where users can ask and answer questions about the content of the topic, an assessment feed where users can debate about answers to assessments set by the administrator, and a general question feed where users can ask about broader problems such as when a deadline is set for. Administrators should have the ability to move debates between the three types if a debate is placed in the wrong section. The question feed is implemented by Quaestio-it, an existing API created by the supervisor. Quaestio-it will also weight the questions and answers according to their utility and the statistics taken from this should be taken into account when calculating a student's proficiency, which in turn should be used to help calculate the utility of the student's questions and answers. Users can support or disagree with comments, encouraging debate on the topic.

In order to encourage students to interact with the website more frequently, gamification techniques should be applied to the website, such as awarding medals for achieving certain goals or allowing lecturers to set quizzes with rewards if someone does particularly well.

2.4 Main Achievements

- The biggest achievement was that we managed to implement the minimum viable product (the core features) of the website. These are:
  - All users can create topics
  - All users can subscribe to topics
  - All users can have debates about topics they are subscribed to
  - All users can see their proficiency for each topic they are subscribed to
  - Administrator users can create assessments and upload the answers for their topics
  - Subscribed users can upload answers to assessments
- We also managed to implement some gamification techniques in the form of a reward system where users can attain badges by interacting with the website
- There is an information page for common aspects of the website that users found counter-intuitive

Chapter 3

Project Management

3.1 Initiation

3.1.1 Project Selection and Risk Management

This initial stage was very important because the group was comprised of people from different technical backgrounds with different areas of interest and expertise. The selection of available projects covered a wide variety of fields in computing and so it was critical to ensure the project was chosen to best meet our team's skills and interests so everyone involved could contribute their best work.

Our project selection process involved a scoring system where each member would mark the project proposal out of 15 marks according to 3 separate criteria: Deliverability (7 marks), Usefulness (5 marks) and Extendibility (3 marks). Deliverability was a measure of how easy we estimated it would be to complete the project to its specification by the deadline. Usefulness was how useful completing the project would be to us in the future in terms of knowledge gained and technology used. Extendibility was a measurement of how much room there was for creativity so that we could extend and build upon the given specification. After considering all of the project proposals with this marking scheme we short-listed those with the highest score. Our top three choices were then discussed and researched further, and we met with the supervisors before making our final decision. By using this technique we established good group communication as all members were considered from the very first stage.

3.2 Planning

3.2.1 Team Structure

The discussion about this project during the selection process helped to inform us of how to divide the group into sub-teams that would work on different aspects of the project. Originally, we split the group into two teams - the front-end and back-end teams. However, over time, we realised that the merging between the code was a bigger task than we realised, so we created a new role- the integrator. In the end, the team structure was as follows:

The Back-End Team - worked mostly with Ruby on Rails :

- Ed worked on the back-end and the overall system design. He helped to improve user experience of the application by proposing extra functionalities. He was the project manager, assigning tasks to each member and keeping track of the group via his own log book and the Kanban board. (Log book entries are included in Appendix E)

- Alan worked on developing the back-end and was the CTO of the group as he had the most experience with using Ruby on Rails. As a result, he initialised the project and ensured the coding environment was healthy. He also set up the cloud integration service and sorted out all deployment to the production server.

- Octavian also worked on the back-end. He wrote the test suites that our code was run against each time someone tried to push to the repository. He was also one of the more experienced programmers on the team.

The Front-End Team - worked with HTML, CSS, Bootstrap, and other APIs :

- Na was the Creative Director and worked on the design of the website and implementing these designs for the front-end of the WebApp. She created paper designs of the web pages, which acted as a prototype for the site.

- Crystal was the liaison between the group and supervisor, ensuring that the website was actually doing what the supervisor wanted. She also helped Na with the front-end implementation and design. She kept track of the teams progress and ideas and did most of the research into current E-Learning platforms and good coding techniques to use for group projects.

The Integrator

- Taehyun, originally assigned to the back-end development team, acted as a the merger of the back-end and the front-end, due to rising demands in the integration process. This included bug-fixes and examining potential conflicts between the front-end and the back-end.

3.2.2 Agile Software Development

Given the short time span of the project, we needed to make our development process as efficient and productive as possible. In order to achieve this, we looked into Agile Software Development techniques as they promote continuous improvement, early delivery and encourage a rapid response to change. Based on our research into Agile Software Development, the most appropriate development processes were Scrum, XP and Kanban.

Tools / Techniques Used

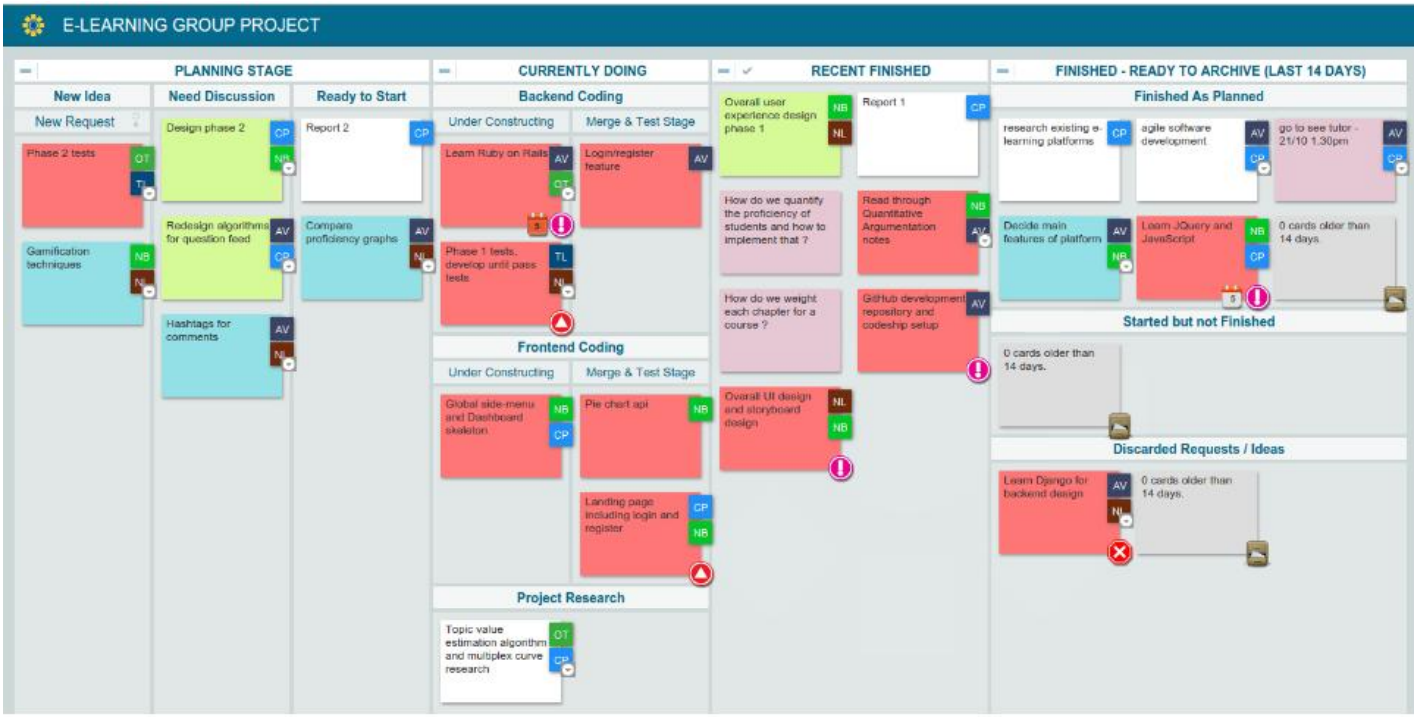
Our group made the informed decision to use Kanban, Scrum and XP development practices, which meant that our team developed different parts of the project in parallel. To address any potential merge problems in later stages, continuous integration became a central concept in our planning process. Therefore frequent delivery of the product was prioritised with utmost importance which, in practice, meant numerous pushes, as often as possible. Furthermore, a dedicated integrator was also assigned to address any issues if there were any at this stage.

We decided to use the Kanban board because it shows how items flow through the development process. By limiting the amount of work-in-progress at each step it prevents overproduction and reveals bottlenecks dynamically so that you can address them before they get out of hand.

From XP, we decided to use principle of managing goals rather than activities as we could not guarantee that the requirements of the project would not change.

Keeping the customer involved with the development process was taken from Scrum, and we organised regular meetings with the supervisor. We also decided to use the pair programming method, as we had varying levels of proficiency in the languages used.

Below is a screen shot of our Kanban board taken as we were completing phase 1 and about to start phase 2. A screen-shot of the board at the end of the project is included with the logbook entries in Appendix E.



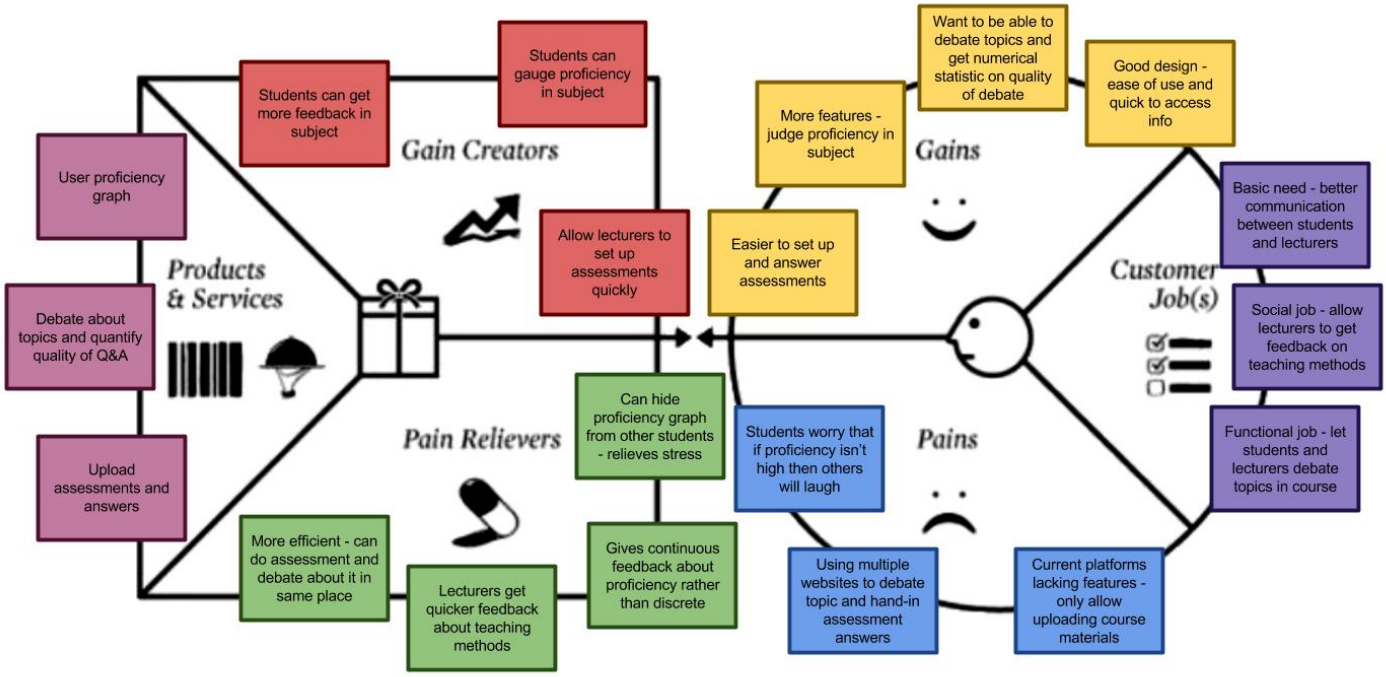
Tools / Techniques Not Used

The Crystal methodology was not incorporated into our process as we felt that it would be obsolete because most of the important principles had been achieved through the other development processes. The Dynamic Systems Development Method was also rejected because it is more appropriate for long-term commercial development, whereas this project is a relatively short term assignment.

3.2.3 Feature Prioritisation

Required Features

We decided on a list of required features according to what the minimal marketable product should offer - as discussed with the client. One of the required features for the website was to integrate an existing debate system (Quaestio it) that analyses comments and gives them a normalised weighting based on their utility. We also had to implement a method of calculating user proficiency and create a platform on which lecturers could share course materials and set-up assessed exercises. The importance and usefulness of these required features was also reinforced by the results of the survey we conducted about E-Learning platforms. From this, we created a value proposition canvas:



Prioritising Features

To help us prioritise the features, we made an ordered list of the features to be implemented: Users, Topics, Debate, Assessments, Proficiency, Partner Finding, and Social features. Users and Topics are the main entities within the website and features involving these entities include debates, questions and answers and proficiency calculations. These features were given the highest priority by the client (the supervisor), so we decided to implement them first.

The first iteration concentrated on creating users. Each topic has to be managed by at least one user so Users had to be implemented first. Topics was implemented next because it was the other main entity which the later features would build upon. We implemented the structure of the topic tree to define the relationship between topics and their sub-topics. Once these elements had been implemented, we integrated Topics into Users so that topics could be owned, allowing the user to edit the topic. It also means that users can subscribe to topics in order to access the course material.

Once these fundamental features had been implemented, the next priorities were the Debate and Assessments because these were the most important to the client. Another reason the debate function is so significant is because the client wants to be able to contribute to helpful discussions about assessed exercises. Next was user proficiency because another fundamental objective of the website was that users should receive feedback on their proficiency in a topic. Together, all of these features would constitute the minimal marketable product, as the core features will have been implemented.

The fourth iteration was to implement the optional features. The client mentioned that they would be interested in introducing gamification techniques to encourage student participation, so we implemented a badge system where users would be awarded badges according to achievements they have managed to attain.

3.3 Execution and Iterative Collaboration

We decided to use the version control software Git and its branching functionality as a fundamental tool as is illustrated by figure 4.6 and described below. (Note: the Development Process diagram is based on the git tree and so progress is shown flowing *up*wards)

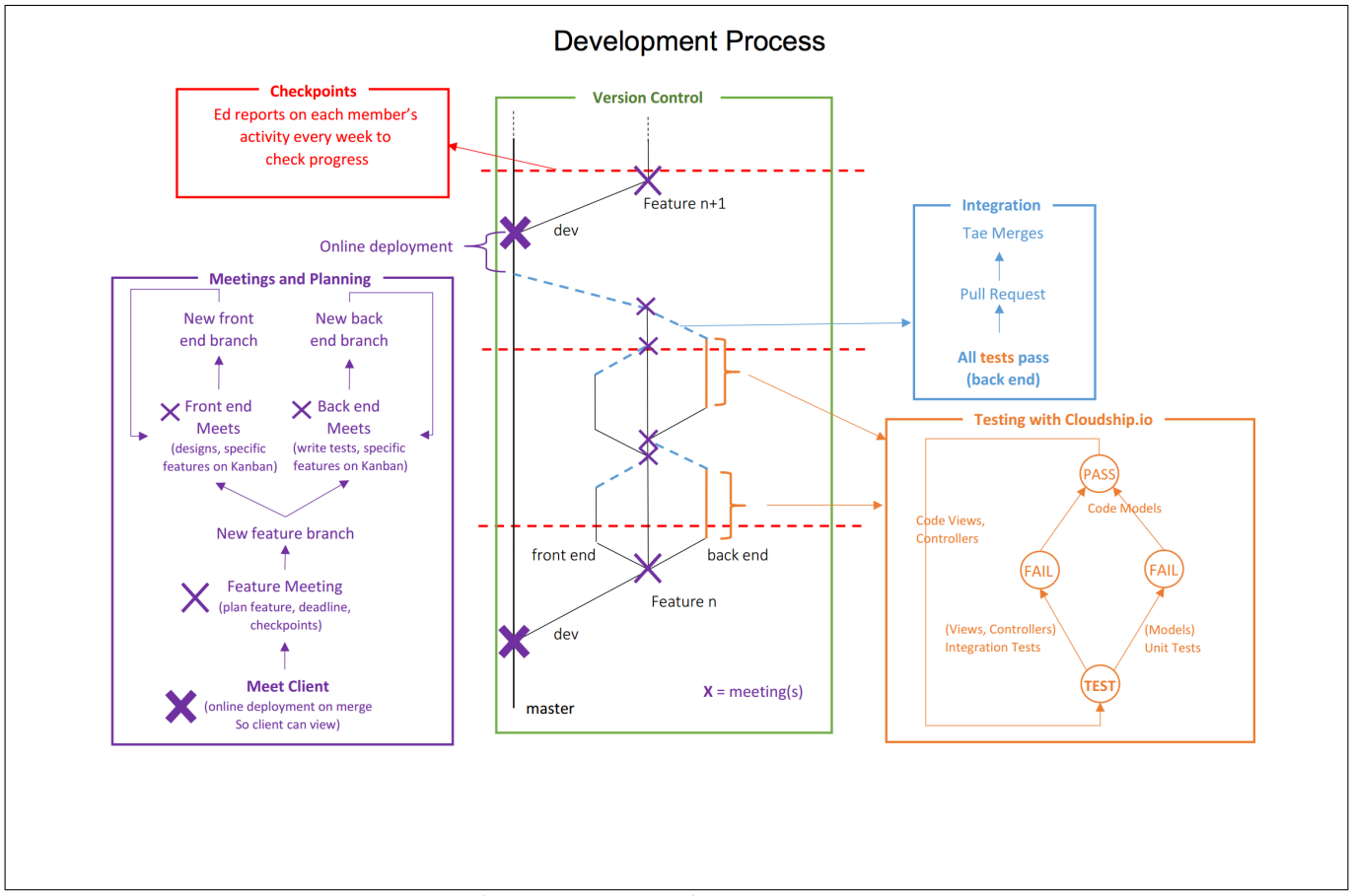


Figure 3.1: Development Process

We progressed sequentially through the features starting the next one only when both front and back-end teams had completed their work on the previous one. At the start of each module we met with our supervisor to evaluate the previous module and then discuss the details for the next.

After the client meeting we would make any required changes and then meet again as a team to plan the next steps including all features, checkpoints and deadlines. A new feature branch was created off of the development branch at this point and we then split into front and back-end teams. The individual teams divided up their tasks based primarily on preference and ability however all final say was in the hands of the group leader. This was done with Kanban so that everyone could see the progress and deadlines.

The back-end team always commenced each iteration by writing the entire testing suite and then branching off of the feature branch and creating a pull request for the integrator to merge back in when the test all pass. Unit tests are written for the models and integrations tests are used to test all components of MVC together. We used the cloud integration (CI) service Cloudship.io, which would go through each of the following steps for each git commit:

- Fetch the latest copy of the project from the head of the repository
- Build our app in the Rails production environment

- Run our module tests
- Run our integration tests
- Email each team member if any of the tests failed
- Deploy our app if the push came from the master branch

When the feature was complete, we would merge it into master and ensure all tests still passed at which point our CI deployed the site and we scheduled our next meeting with our supervisor.

3.4 Group Management and Direction

Group Management strategies were customised to closely meet the needs of the team by considering the team structure, available resources and the personalities of each team member. This ensured that we could maximise the team's performance and minimise any possible conflicts within the group. In this section we list some of the management techniques used throughout the course of the project.

3.4.1 Deadline Control and Delegation Of Work

In practice, good control of deadlines appeared to be very important to ensure every member could finish their tasks on time. This was especially crucial due to the limited time in which we had to complete the project. We had weekly iteration meetings in order to calculate the time needed for new tasks or completing old tasks.

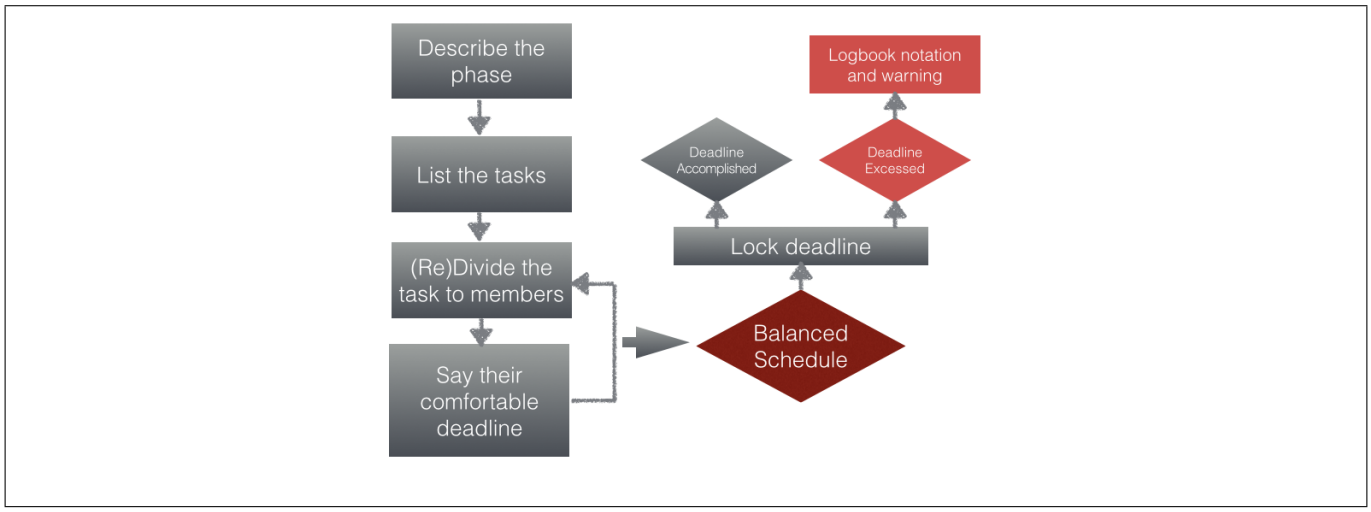


Figure 3.2: Deadline Control Model

After each meeting with the supervisor, we followed the phases described in the deadline control model based on what was discussed. Firstly we would summarise the requirements detailed by the supervisor as these determined the work to be completed in the next phase of development. Then we broke the phase down into each task we needed to complete. The work was then delegated to each of the team members based on the pre-defined team structure or, should the task not fit into this structure, the member's own skill-set and interests. Team members estimated a comfortable deadline for the completion of each of their assigned tasks and the group manager recorded this and formulated a schedule for everyone to follow for the course of the development phase.

This schedule was fine tuned after further group discussion until all group members were confident that they could follow the given plan and that all necessary tasks would be completed on time. The result of

this was the Balanced Plan as shown in the deadline control model. At this point all of the deadlines were "locked" and if any member did not accomplish the work they had been assigned they would be issued a warning by the group manager who would also note the missed deadline in the log book.

By using this model, group management was more democratic and the group leader took more of a coordinator role. Everyone's workload and deadlines were agreed before work commenced which reduced any cause for disagreements or unfair demands on group members. The highly organised nature of the schedule maximised the group's performance and allowed for more efficient development.

3.4.2 Information Control

Information control was another important component of group coordination. In particular it was important to ensure transparent and efficient group communication and information exchange. In addition to updating our shared Kanban board we also employed several other communication methods:

- Consistent and regular use of e-mail, always making sure to Cc every group member and the supervisor
- Regular use of the Facebook group page to share resources, post useful information or ask questions related to the project
- Use of Facebook group chat for efficient instant communication with all group members
- Exchange of every group member's contact information including telephone and e-mail
- Summaries of meeting minutes (including task allocation) posted on the Facebook group

Through the use of these communication strategies, all group members have always been kept up to date with the progress of the project and can react quickly if anything happens.

3.4.3 Emergency Control

Good emergency control allowed the team to respond quickly when any unforeseen issue occurred. By assigning multiple team managers for situations where the group leader was indisposed we ensured that our team structure was more robust and therefore communication and development could continue with little disruption. Examples of emergency situations which occurred during the project are:

- In week 3 one of the branch team members had an emergency situation meaning he could not work for several days. This was addressed by his team manager speaking with another team manager to formulate a new schedule by reassigning his work to another person (borrowing human resources).
- In week 6 the group leader was taken ill and another team manager quickly assumed the role to coordinate the team. This was made possible by the consistent communication between group leader and deputy team managers.

In practice it is not possible for every member to exchange detailed information about all progress updates at all times. Therefore, efficient information exchange between members and team managers is a better solution for emergency control in a small group such as ours.

3.5 Closing of the Project

At the end of the project we plan to file together all logs and resources including: all of the code, this report, the log book, the most important posts from the Facebook group and the list of unfinished tasks from the Kanban board. This is all of the information that should be required in future for improvement and maintenance of our product. We will arrange a final meeting with the supervisor to give an overall conclusion of what we have done and draw the project to a close.

Chapter 4

Design And Implementation

4.1 Design

4.1.1 Front-End Design

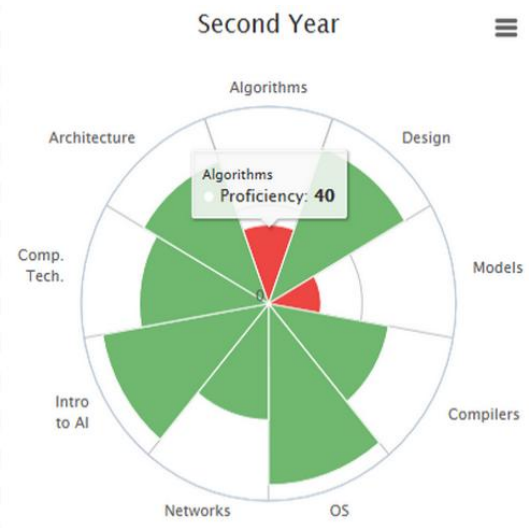
Overview of Design Process

Firstly, the basis of the front-end design was the idea of having two types of user (lecturers and students) and then designing a platform to allow them to communicate with each other about different topics. The starting point of the design process was to consider all of the desired functionalities and interactions between users and, using this list, determine the possible types of pages to be included in the website.

Given a list of the necessary pages we then designed a flowchart detailing the different links between them and we used this to see how a user could navigate the site (this flowchart is included in Appendix C). Paper designs were created for each page according to the functionalities they needed to include and the links they needed to provide to other pages. These designs were subject to constant change after each meeting and after receiving feedback from the end user.

Displaying User Proficiency

The ability to profile students based on their academic proficiency in given courses was a fundamental part of this project and therefore we needed to devise a clear method of displaying these proficiencies. We decided on a polar area chart for each topic, wherein each subtopic would form a sector of the chart with a score between 0 to 100.



We also introduced a colour coordination scheme so that if a subtopic score fell below a certain threshold (50%) it would be coloured red, and above this threshold would be coloured green. The benefit of this approach is that, at a glance, it is very clear how well the user is doing and which precise areas need the most improvement. The polar area chart scheme also adds to the ease of navigation because each sector is also a hyperlink to that subtopic.

Web-page Organisation

The functionalities which were considered during the initial design phase were as follows

Function	Lecturer	Student
Create a topic	✓	
Subscribe to a topic	✓	✓
View student proficiencies	✓	
View own proficiency		✓
Ask question	✓	✓
Answer question	✓	✓
Create Assessment	✓	
Complete Assessment		✓
Provide Assessment answers	✓	
Access Assessment answers	✓	✓
Other account/profile settings	✓	✓

We then used this list to organise the functions into different page types:

- Landing Page: the welcome page, allows users to register and log-in
- My Topics: home page seen as soon as the user logs in, shows which topics you are subscribed to and which you have been invited to join
- Create a Topic: allows the user to make a new topic
- Personal Profile: shows details about the user including proficiencies in different topics
- Topic Page: shows details about the topic including description, subscribers, links to question feeds and assessments, subtopics etc.
- Question Feed: shows all of the questions being asked about a given topic
- Question Page: shows an individual question (using Quaestio-it API)

On each page (excluding the landing page) there is also a side menu which provides links to the user's own Profile Page, My Topics, Create A Topic and an FAQ popup. This was done to ensure that there was always a direct way to get to these commonly used pages.

Page Designs

Once we had a better understanding of what we hoped to achieve with each page we made an initial template design for each of them in order to facilitate front-end development.

The following diagrams are the preliminary designs for the My Topics and Question Feed pages respectively. Additional page designs are included in Appendix D.

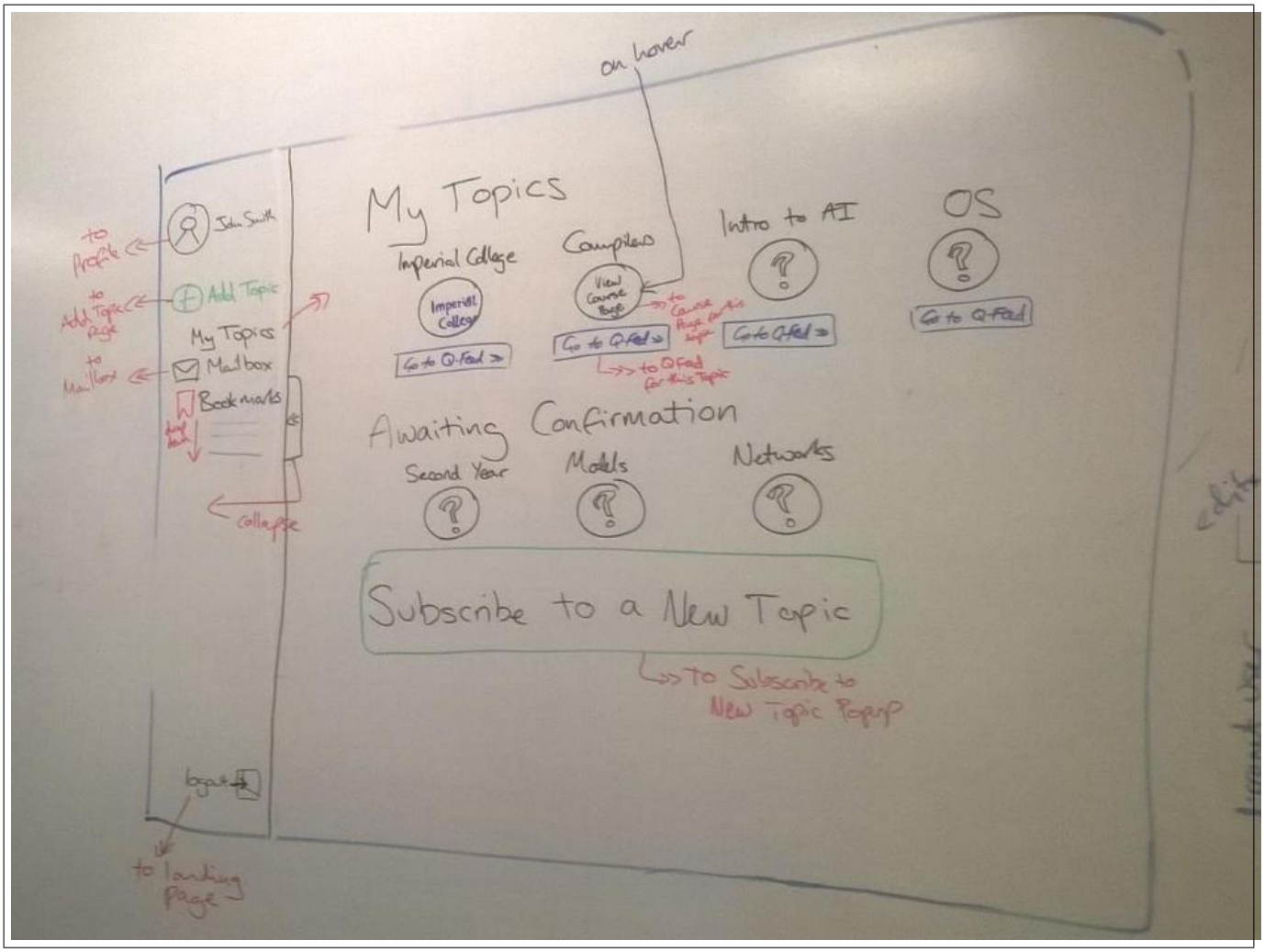


Figure 4.1: My Topics

This diagram shows the global side menu which has links to the user's own profile, the Create A Topic page, the My Topics page and a log-out button. It also includes links to a mailbox and bookmarks list which were not included in the final design. The link to the FAQ page is also not included in the initial design as this was introduced at a later stage as a result of incorporating user feedback.

The main page shows a list of the topics to which the user is currently subscribed and a list "Awaiting Confirmation" which includes the topics the user has been invited to. There is also a Subscribe to a New Topic button which was intended to allow the user to choose a topic to subscribe to by searching all existing topics but this was later removed after it was decided that the topics would operate on an invite-only basis.

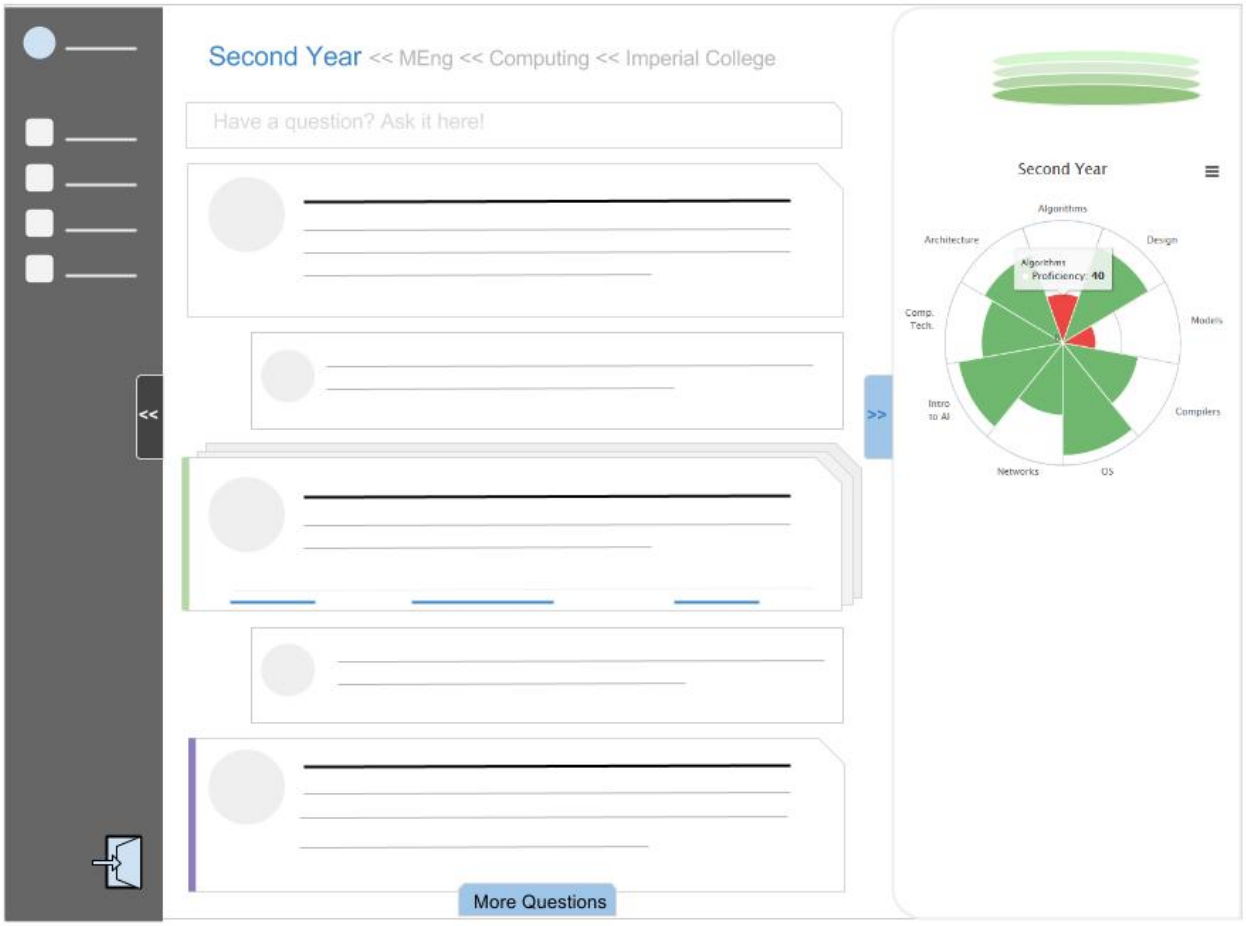


Figure 4.2: Question Feed

This diagram shows the proposed design for the question feed of a topic called "Second Year". The top of the page shows a chain of links to the ancestor topics of "Second Year" which would lead to their question feeds. Under this is a text field to ask a new question followed by the list of questions being asked about the given topic. The question feed was designed to display the top responses to questions however this was not included in the final implementation.

The other main feature of the question feed was intended to be the side widget displaying the user's current proficiency levels in this topic. Each sector of the graph would link to that subtopic and each disk displayed above the graph would link to an ancestor topic. This side widget was not implemented and instead the user's proficiency is displayed in its own section of the Topic Page.

Another alteration in the final implementation is that there are two different categories within the question feed page- General and Technical. Both of these categories have their own tabs in the implemented question feed page.

4.1.2 Back-End Design

In the early stages of the project, we discussed all components of the website with the client and among ourselves. We were able to break it into the following modules:

- Users: Log in, Sign up and Account Settings
- User Privileges: Teachers and Students permissions
- Topics: General information, Hierarchy
- Questions: General, Technical and Test
- Tests: Multiple Choice Questions, Open Answer Questions, Answer/ Solution uploads
- Topic Subscriptions: Inviting users for administrative or normal roles, each with different permissions
- User Profile: General information, Proficiency in topics and Badges
- Testing: Unit and Integration

Design Principles

The Rails framework predetermined most of our approach to the design of our site. We followed the Model View Controller pattern by doing the following:

- Putting all business logic in the models
- Letting controllers deal with fetching formatted data from the models
- Filling the correct views with the fetched data

Any further styling was handled by the front-end team.

When starting a new feature we would always begin by designing the required database tables and the relations between them. We did this by sketching a relational model between the required fields. This process involved the entire back-end team so everyone knew how the feature worked at the most fundamental level.

We then decided on appropriate data types for each field as well as any necessary validations (such as not being null). We determined the required functionality for creating and deleting an element in the table. This included things such as deleting all relations linking to the entry to be deleted. Finally we created a migration for the database, specifying rollback details. At this point we also wrote all unit tests for the module.

We then moved on to designing all actions we wanted to expose. As well as any necessary custom methods, we generally used the standard RESTful actions detailed below:

- GET /feature == index action
- GET /feature/:id == show action
- GET /feature/new == new action
- POST /feature == create action
- PATCH /feature/:id == update action

- DELETE /feature/:id == delete action

When this was done we set up the routing file for the application so everyone was effectively programming to an interface. At this point we wrote all integration tests for the feature.

We then filled in all method stubs in the controller, determined by the routes file. Any helper methods were placed in a separate file which was always imported.

4.2 Implementation

4.2.1 Front-End Implementation

Overview

The front-end implementation involved the assets (including images, JavaScripts and stylesheets) and the HTML views with embedded Ruby. The images included the default user and topic icons as well as all of the achievement badges. The pre-compiled JavaScripts were, for the most part, externally sourced scripts which were relevant to the whole application. Other more specific JavaScripts were rendered on the appropriate page using embedded Ruby in the HTML view.

For each feature implemented by the back-end a default view, or set of views, would be created with the minimal amount of necessary content. Given the view(s) the front-end team would then expand on this content, adding to the relevant JavaScripts and stylesheets to match the preliminary page designs as required.

Screen shots of the final product are included in Appendix D.

Bootstrap

Within the pre-compiled assets we have included the latest available distribution of Bootstrap (v3.3.1) which includes CSS and JavaScript. This was used for the formatting of basic components and for the fluid grid layout. Bootstrap alert components were used to format all of the error messages and notifications.

Codrops

Codrops is an online collection of web related resources and provides a large variety of example components. From this we chose to use a template for a registration/log-in form (by Stephanie Walter) and for a collapsible side menu (by Mary Lou). Using the downloaded templates we took the necessary JavaScript and CSS elements to adapt the example components to match our own design.

The registration/log-in forms were used in our landing page. JavaScript was used in order to allow the user to switch between the two forms with a sophisticated, more fluid animation and without having to reload the page.

The collapsible side menu is rendered in every page apart from the landing page. A script is generated on each of these pages, and it allows the user to hide the menu if it is not needed. This provides more space to view the main content. When the side menu is displayed, the main page section is effectively pushed to the side of the screen. When the side menu is hidden, it is placed beyond the left edge of the screen. This could be particularly useful on narrower screens.

Highcharts

JavaScript files from the latest distribution of Highcharts (v4.0.4) are included in the pre-compiled assets. This is required for the interactive proficiency charts which are displayed on user profiles and in topic pages. In addition to the main pre-compiled JavaScripts, each relevant HTML view includes Ruby code to generate a script for each chart to be displayed including the specific data to display. Another script is used for the colour coding based on the proficiency score in each sector of the chart.

For each chart that is displayed, there is an option to download the chart in a variety of formats. There is also an animation that takes place when the page is loaded. When the user hovers over a sector, an animated caption will appear with the name of the subtopic and the user's proficiency score. The sectors of the chart also form a hyperlink to that subtopic's page.

4.2.2 Back-End Implementation

Database

The figure below provides a complete description of our database tables, fields and relations. We used a PostgreSQL database as this was a prerequisite of our production server Heroku.

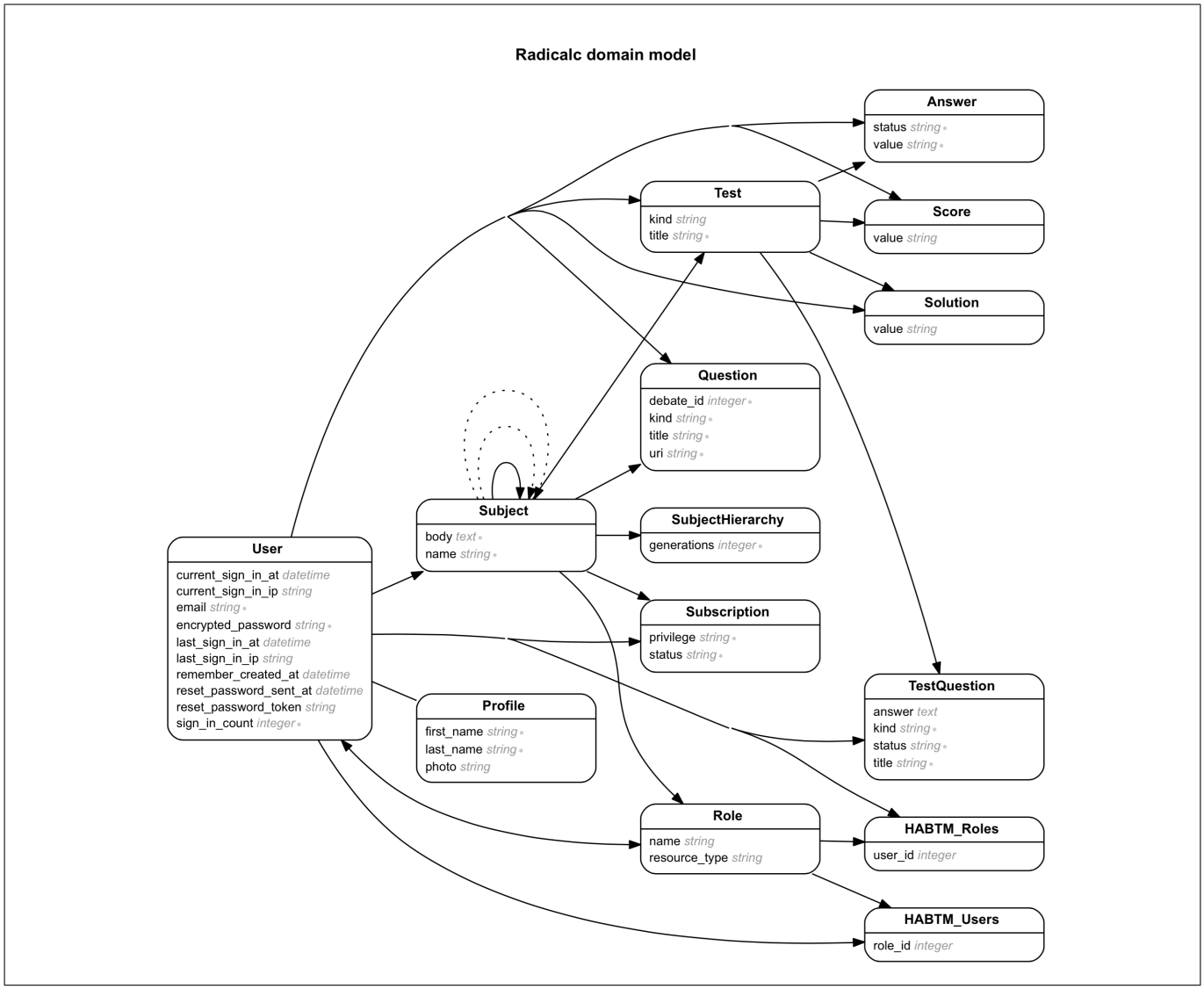


Figure 4.3: Database Schema

The Closure Tree gem helps us construct the 'SubjectHierarchy' table, which creates a connection between the ancestor and descendant elements of the Subject table. We also added a binary tree index for this table in order to efficiently access individual elements as we traverse the tree.

The Rolify gem creates the Role table, that labels roles onto subjects, which it then associates with users, such that the User table has a many-to-many connection with the Subject table. The HABTM_Roles and HABTM_Users tables are constructed for efficient memory storage: HABTM_Roles stores the users which have each role, while HABTM_Users stores the roles of each user.

Users

We implemented users with the devise gem. Devise is a flexible authentication solution for Rails based on Warden. It:

- Is Rack based
- Is a complete MVC solution based on Rails engines
- Allows you to have multiple models signed in at the same time
- Is based on a modularity concept: use only what you really need

Warden is a Rack-based middleware, designed to provide a mechanism for authentication in Ruby web applications. It is a common mechanism that fits into the Rack Machinery to offer powerful options for authentication.

This allowed us to easily implement users, however the devise routing required some customisation for both the sessions and registrations as we did not want to route to the devise views upon success/failure. The front end team was responsible for the customisation and styling of these devise views.

When a user is created on our site they are also automatically created on Quaestio-it.com, however we do not use the same password for security reasons. More information about how this happens can be found in the Quaestio-it API section below under the createUser instance method.

User Privileges

We have implemented the user privileges using the gems Authority and Rolify.

The Authority gem encapsulates classes into corresponding Authorizer classes, to which it then manages access. Within those Authorizer classes, we have implemented methods which allow different privileges for users(such as the possibility to read or to update). The methods decide whether to give the current user permissions by looking at the roles the user has. We have used Authority rather than other similar gems such as CanCan or Pundit because it is more widely used and also has a more comprehensive documentation; moreover, its API is better tailored for our functionality.

The users have different roles on subjects and tests. The roles are managed by the Rolify gem, which transforms topics and tests into resources; then, the current user is given a set of roles (such as admin or member) on a resource. For instance, this happens when a user accepts an invitation for a topic: it receives the role given to him by the invitation on the current topic and on all the subtopics. The roles are used by the Authority gem as described in the above paragraph.

Topics

The topics are organized in a tree structure, for which we have used the Closure Tree gem. The tree structure has allowed us to organize the subjects from the most general to the most specific. Within the page of each topic, the user can access all its subtopics (descendants) and the more general topic (the current topic's parent). We chose Closure Tree rather than other gems which offer similar functionality such as Ancestry, because it's compatible with Rails 4 and is constantly maintained by a larger number of contributors. The gem offers an API for accessing a topic's descendants, ancestors and siblings.

File Uploads

All file uploads are handled by the carrierwave gem. Carrierwave provides a simple and extremely flexible way to upload files from Ruby applications. It works well with Rack based web applications, such as Ruby on Rails. We chose carrierwave over its competitors Paperclip and Dragonfly because it is the most mature and full-featured image processing and uploading library.

We chose to use the fog gem in combination with carrierwave. Fog is the Ruby cloud services library, top to bottom:

- Collections provide a simplified interface, making clouds easier to work with and switch between.
- Requests allow power users to get the most out of the features of each individual cloud.
- Mocks make testing and integrating easier.

Carrierwave combined with fog integrates best with Amazon s3, Rackspace or Google storage for developers. We chose to use Amazon s3 as it was free to use for a year and the simplest to set up.

Quaestio-it Application Programming Interface

We used the curb gem for all API calls. Curb provides Ruby-language bindings for the libcurl(3), a fully-featured client-side URL transfer library.

- Libcurl is a reliable and portable library which provides an easy interface to a range of common Internet protocols.
- Libcurl can be used for free in our application.
- Libcurl is probably the most portable, powerful and often used C-based multi-platform file transfer library.

The following is a list of the API calls used and our instance methods for our Quaestio class (lib/quaestio.rb) that makes the calls and does all error handling is provided below:

- POST http://www.quaestio-it.com/api/generic/createUser
  - Parameters:
    - username (text)
    - firstname (text)
    - lastname (text)
  - email (text)
  - Returns: success(200)/failure(500)

  - Description: Request to create the user within Quaestio-it. Allows reuse of functionality on the host side for gathering user statistics etc. This is called every time someone signs up to our web application.
  - Quaestio class instance method: createUser(firstname, lastname, email)

- POST http://www.quaestio-it.com/api/generic/newDebate
  - Parameters:
    * username (text)
    * question (text)
  - Returns: URL
  - Description: Creates a new debate and returns a reference of the debates URL to open from an iframe
  - Quaestio class instance method: newDebate(question, firstname, lastname)

- GET http://www.quaestio-it.com/api/generic/getUserStats
  - Parameters:
    * username (text)
  - Returns: JSON object with user statistics
  - Description: Responds with the statistics of a specific user.
  - Quaestio class instance method: getUserStats(firstname, lastname)

Questions

We have three types of questions in the site:

- General: These questions are used for general enquiries such as what time a lecture is schedule for or who the lecturer is etc.
- Technical: These questions provide an area for students to have debates and clarify different areas within the course or even outside of the scope of the taught course.
- Test: These questions allow students to attempt to discuss specific problems set by the teacher together much like a tutorial session.

All three types of questions are accessed through an iFrame linking to the online debating platform Quaestio-it.com.

More information can be found in the Quaestio-it API section under the newDebate instance method.

Tests

Our implementation of tests was very simple. We allow teachers to create tests and then add as many questions as they like to them. They can then upload example answers as a pdf file for all students to view.

Students can see all the tests and upload answers in pdf files to them and can then see their grade once the teachers publishes them.

All uploads are handled by our Carrierwave, Fog and Amazon s3 combination as described in the File uploads section

Topic Subscriptions

We created a mapping between the unique topic id and user id and have a Boolean field, indicating whether the invitation has been accepted, or not. We then just check against the presence of this entry in the table when any subscription permissions are required.

User Profiles

There are three main components to the user profiles:

- General: This contains general information such as the name of the user. It is all part of the information gathered when a user subscribes to the website.
- Badges: Make use of the getUserStats instance method from the Quaestio class detailed in the API section. This method parses the JavaScript Object Notation returned into a hash of number of items answered and the strength/ quality of the items answered. After a user reaches a certain number of questions, answers and comments they receive certain badges. Some badges also take into account the time period over which these numbers increase, giving users an incentive to participate and continue to do so.
- Proficiency: Makes use of the same method as previously mentioned now accessing the quality attribute of the hash. We encountered many technical difficulties with this section, detailed later in the report and as such only have a basic system in place where we just consider all the strengths of answers and questions and calculate the mean at each level in the hierarchy of topics.

Image upload for profile pictures are again handle by our combination of carrierwave, fog and amazon s3 as detailed in the uploads section

Application Testing

For testing we have used RSpec, a testing framework for Rails applications because it is specifically tailored for Behaviour Driven Development. Using RSpec, we would first specify the required behaviour of a feature in a test, then we would implement that feature. We had two types of tests:

- Unit tests - describe the functionality of individual components
- Integration tests - which specify the behaviour of the site as a whole

The unit tests have mostly covered the functionality of the models, which are easy to check individually, while the integration tests were better for checking the controllers and the views, which have a strongly connected behaviour. For implementing the integration tests, we have used the Capybara gem as it is designed to simulate how a real user will interact with the application (its API has functions which imitate the filling in of a field, the clicking of a button, visiting a page etc.). Furthermore, the hierarchical structure which can be given to the tests allows the composition of error messages, such that they become very descriptive of the error.

In both the unit tests and the integration tests, we have used the FactoryGirl gem. This gem can be used to quickly instantiate objects for our tests. For this purpose, within the 'spec/factories' directory, we have created factories for each of the models. We chose the gem in order to factor out data instantiation from the tests.

4.2.3 Changes In Tools Used

Development environment

We initially planned to write our project in Python using the Django Framework because these were used in the development of Quaestio-it. However we soon noticed that we would only be making calls to the API. Due to the fact that the team was more experienced with Ruby on Rails, and its growing popularity, we decided this would be a better development environment.

Deployment

We initially considered using the Department of Computing cloud server, however we then noticed more online support was available for deployment of a Ruby on Rails application to Digital Ocean. We eventually settled on using Heroku which was free, unlike Digital Ocean, and also allowed for a much simpler deployment process.

Continuous Integration

Our initial proposed method of continuous integration was to use Django and Jenkins. However, when we changed our development environment to Ruby on Rails we opted for using Cloudship.io which allowed for simple setup and good customisation options. Another benefit of this service was its GitHub integration which shows if the build and tests pass on any pull request.

Behaviour Driven Development

At first we had a dilemma over choosing between the two main testing frameworks of Rails applications: RSpec and Test Unit. After some debate, we chose RSpec rather than Test Unit as our testing framework because it has a Domain Specific Language (DSL) tailored for testing. RSpec's DSL describes the behaviour of the code using keywords such as 'it', 'should', 'be nil', so the syntax is much closer to the English language and tests are much easier to read/write. Furthermore, when a test fails, RSpec doesn't return the message from an assert (unlike Test Unit), but the message from a specific area of functionality, making the cause of the error clearer.

Database

Ruby on Rails comes packaged with SQLite3 which is very inefficient in terms of concurrency and is designed for development purposes so we quickly chose to discard it for the more advanced MySQL, which is unfortunately harder to set up. When we later deployed to Heroku we noticed it only works with PostgreSQL. At this point we changed the production environment of our application to use PostgreSQL and kept the development and test environments using MySQL. We were aware this was not the best practice but it seemed to make more sense than having the whole team install a different database after the difficulties we faced installing MySQL. By rigorously testing the site every time we deployed to our production server, we were able to avoid having issues with this choice of implementation.

Front-End

For the front-end, we changed from using pure CSS to including Bootstrap because Bootstrap gave extra functions and provides an advanced grid system which was fundamental to the implementation of the design of the site.

4.3 Technical Challenges

4.3.1 Setting Up MySql

By default, MySQL adjusts its installation process to the target system environment. Therefore conflicts arose, which were resolved by modifying and unifying the enviromental variables. One significant clash we anticipated was reconfiguring the address of the socket in the database.yml file.

4.3.2 DoC user integration

To provide a better service for Imperial College Department of Computing staff and students, the ability to register with a DoC account would be very important. Under normal circumstances we could achieve this using the college's LDAP service to authenticate users. However, after several attempts we were advised that this feature is beyond the scope of this project.

There were several reasons for this decision. Firstly the project has been published on Herokuapp and the DoC server is within Imperial College. Therefore communication between the app and DoC's LDAP server is blocked by firewall. Secondly the application has been designed with the intention of being used from inside or outside the college campus. This ability to use the service from outside campus is considered necessary to be an accessible e-learning tool. Finally, to go through DoC's firewall would also be an issue. CSG has advised us that, if the project is intended by the supervisor to be used by DoC members in future, the firewall issue can be addressed at such a time, but this would be outside the scope of this group project

4.3.3 Application Programming Interface - Quaestio It

The Quaestio-it API we needed to integrate into our site was used by a different site and was being developed alongside our project. We planned to incorporate it into our site around week 7 but many technical difficulties faced by the developers resulted in the API being completed only days before we intended to start user testing. We managed to implement user profiles on RadiCalc and link them to user accounts on Quaestio-it so users could create questions for each of our debate types. However, we were not able to fully implement user proficiency and thus were not able to gain an abundance of user feedback on that aspect.

We planned on authenticating users on each API call, but this was not implemented in time by the client team. We would create a new debate with just the username and were returned a link for our iFrame. When we tried to add an answer or a comment to the site through the iFrame we got an authentication error but there was no possibility of signing in. This meant that we could not build up any user statistics because the getUserStats method always returned the default values. As a result, we could not test or develop the profile and badges functionality sufficiently so it does not accurately reflect a users ability.

4.3.4 Proficiency Widget

The first design of the question feed page included a collapsible side widget which would display the user's proficiency in the given topic. However, this was not implemented due to time constraints and the complexity of involving two collapsible components pushing the main page content from both sides.

4.3.5 Highcharts

In order to include the highcharts polar area charts in our application we installed the highcharts-rails gem. Although this worked in our development environment, the charts would not display in our production environment Heroku. In order to overcome this difficulty we directly downloaded the necessary JavaScript files from the latest Highcharts distribution and added these to the pre-compiled assets.

Chapter 5

Evaluation

5.1 Quality Assurance

The performance of our application is highly dependent on the platform supporting its deployment. For this we chose Heroku because it offers a variable bandwidth, so if a large amount of traffic is generated in a short amount of time, our site will not crash. Furthermore, Heroku allows free usage for web sites which demand few resources. Another advantage of Heroku is that it has been constructed specifically for Rails applications, and is therefore easy to use: deployment only requires a git push command.

The usability of our website is guaranteed by the integration tests, which simulate user interaction. We tested every interaction before implemention, so it is unlikely that there will be deficiencies exposed to the users. Furthermore, we gained as much feedback as possible about the user experience throughout the development process so that our website could be made more intuitive.

To ensure security, we made all controller methods that were indirectly called by the user private. Moreover, we always regulated the number of parameters that can be sent by the user to the controller via the URI. Some were marked as required, and others as permitted; no other parameters were accepted by our code. Therefore, if a user tried to hack our website by passing it extra parameters, those would be rejected.

We took several steps in order to ensure the functionality and reliability of our code:

- Behaviour Driven Development - we specified the required behaviour of our application before implementing it. We wrote the tests first, then we made them pass by writing the code.
- We used gems which are continuously maintained and which have a large number of contributors, and are therefore unlikely to have unreliability issues.
- We used Codeship for continuous integration. If one of us pushed to the Github repository, Codeship would run the tests and would deploy the application to Heroku, e-mailing all of us with the result.

The modularity of our application was strongly helped by Rails, which separates the functionality into three parts:

- Model - contains business logic and interacts with the database
- View - contains code which will be displayed to the user
- Controller - deals with user requests, connecting the Model and the View

This structure allowed us to associate multiple Views to the same Controllers and Models, therefore avoiding code duplication.

In order to ensure that our product was produced at a high standard, we implemented unit tests for each phase to ensure that the product objectives were achieved on a fundamental level. We checked each property of models individually using unit tests. Furthermore, in order to factor out the data instantiation from the test itself we created factories for each model. For verifying the controllers and views and their interactions, we created feature tests which simulate a succession of clicks and form fill-ins done by the user and checks on the information which is displayed as a result. Also, we tested the routing of our website in order to increase its security. We also had regular meetings with the supervisor to check if any changes to the product or its aims were necessary.

The integrator was in charge of merging each pull request and ensuring the tests still passed. Whenever the integrator detected issues he would get other team members to assist in fixing problems before continuing development. We also had the team leader, Ed, give a feedback report to each team member on their performance at the end of every week. This helped to ensure that each team member's work was consistently of high quality.

5.2 What We Have Not Achieved

There are extra functions that we wanted to implement that we did not have time for, such as:

- The ability to compare graphs between users
- Adding more social networking features, such as personal inter-user messaging
- Multiple choice questions for tests
- Having an intelligent user profiling system taking into account the hierarchy of the topics as well as the changing of proficiency over time

5.3 Deviations From The Original Plan

In our original team structure, we only had the back-end and front-end teams. However, due to merging issues, we created a moderator role as well.

At first, we wanted to implement the back-end using Django because the existing API (Quaestio-it) was written in Django and so we thought it would be easier to integrate it as a result. However, we soon changed this as no-one in the group had much experience with Django but Alan was proficient in Rails and it was still relatively easy to integrate the API. Also, Rails has many gems that we could utilise to make the WebApp more intuitive and fluid.

For the online assessments, the original idea was to only do multiple choice assessments as they did not require the lecturer to do any extra marking. Then the plan was changed so that the lecturer could choose to make multiple choice questions, with the option to ask for justification, or questions that require explanations as answers. Assessments were supposed to be able to be a mixture of these types of questions. In the end, multiple choice questions were too difficult to implement so we only implemented questions with justified answers.

On the front-end side of things, the original design was to have a side widget so that you could always see your proficiency in a topic. However, this was later removed because having both the side menu and the side widget open would take up too much of the page if both were opened. People surveyed also found the side widget confusing to use because of the way we planned to show the subject tree, so we modified the way in which you navigate through topics.

5.4 Product Feedback

5.4.1 Supervisor Involvement and Feedback

We managed the relationship with the client by having regular weekly meetings to get feedback on the last feature and discuss the next feature to implement. This ensured that the project was kept on track and that the objectives were being fulfilled.

Another example of client feedback was during the first design stage, we showed the supervisor some of the web page designs we had created but she found certain aspects hard to grasp. The aspect that confused her the most was the side widget displaying the user proficiency graphs. Originally we wanted the widget to show user proficiency at the current level in the tree. Graphs for higher/lower levels would be displayed as a stack of disks above/below the current graph to represent where it belongs in the tree. This was changed so that the user gets to sub-topics and parent topics through a menu selection at the top of the page.

Most of the feedback from the supervisor was about defining the types of assessments and the use of the debate and questions features. We initially decided to implement only multiple choice questions as assessments because they are a quick way of providing feedback on how proficient a student is in a subject. It also meant that the lecturer didnt have to spend time marking each students answers because the answers could be uploaded with the questions when they created the assessment. However, the supervisor thought that that would be too simplistic and so we were supposed to give lecturers the option of deciding between multiple choice and justified answer questions. This later changed as we did not have time to implement the multiple choice questions.

5.4.2 Gathering Feedback

In addition to the comments from the supervisor we also gained feedback by other means. Throughout the development process we asked several students to try out the website to gather verbal feedback on whether they would find the product appealing to use. Over the Christmas holidays, the PhD students and the supervisor tested the product in order to identify any bugs and give feedback about the user experience.

We also created some SurveyMonkey questionnaires to get feedback on the website from the general public. Firstly we made a survey to determine what types of e-learning platform people are currently using and what features are considered the most and least important to users. We then created two different questionnaires, one for students and one for lecturers, to determine which features were being well received and where the website needed improvement. These surveys are provided in Appendix A

5.4.3 Survey Results and Feedback Incorporation

If a user did give criticism on the product, we asked them to elaborate on the problem and explain how they would like us to modify the website. If we felt the problem was significant, we would work on it and ask the user to retry the product to see if they are satisfied. However, if the problem was not significant, then we told them that we would consider modifying the website at a later stage when there was less pressure to finish implementing the core features. Students and lecturers have different expectations of the website so we had to note the differences so as to design different experiences for the types of users.

SurveyMonkey provides an analysis service which allows us to clearly view the results of the surveys. The results of the website feedback surveys for students and lecturers are included below.

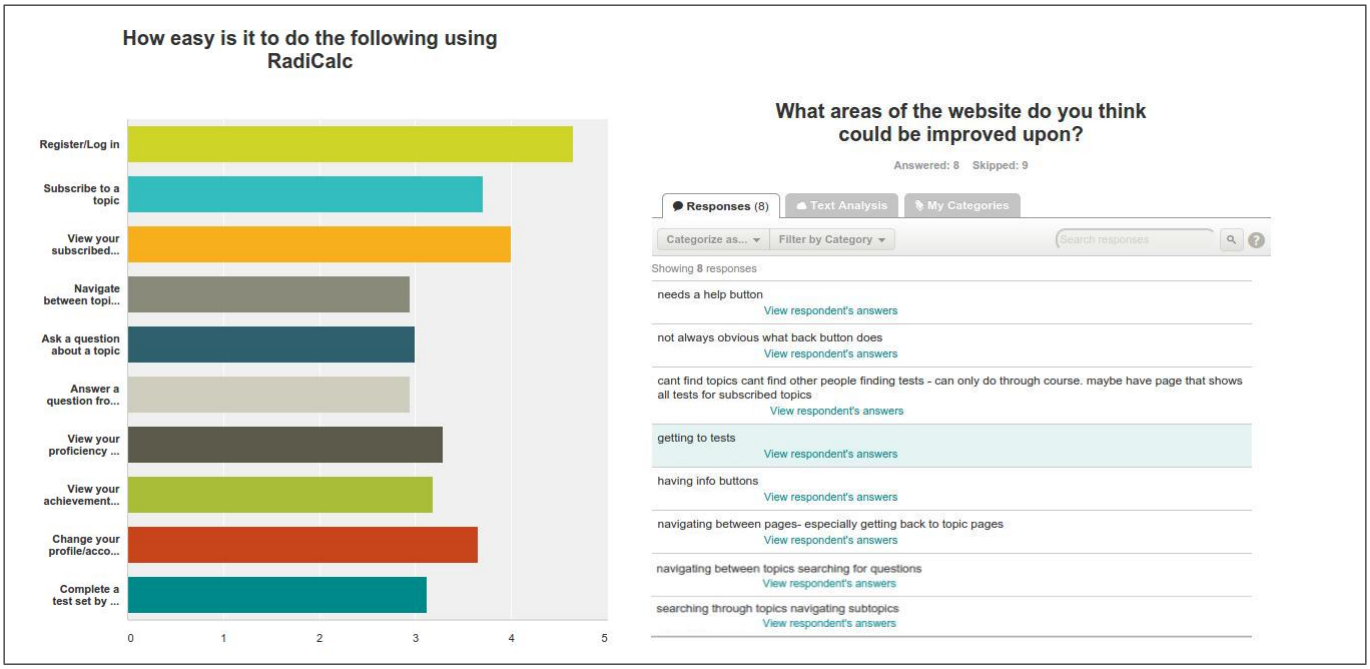


Figure 5.1: Results of Student Survey

From the results of the student survey we can see that overall the features have come across as relatively easy to use because the scores (from 0 to 5) are all above or very close to 3. The easiest of the features seem to be:

- Registering/logging in: 35.3% of respondents found this "Easy", 64.7% said "Very Easy"
- Subscribing to a Topic: 70.6% responded "Easy", 5.9% responded "Very Easy"
- Viewing Subscribed Topics: 58.8% responded "Easy", 23.5% responded "Very Easy"
- Changing profile/account settings: 35.3% responded "Easy", 17.7% responded "Very Easy"

The other features have a very similar score which suggests that student users find them equally usable. The features with slightly lower scores are navigation between topics and subtopics and asking/answering questions. The difficulty with navigation could be accounted for by the lack of the proficiency side widget which was not implemented which would have provided another method of navigation. The difficulty with asking and answering questions could be due partly to the number of separate web pages being larger than we had initially intended. We have also encountered difficulties with the Quaestio-it website which has caused some problems with asking and responding to questions through our application.

The comments also reflect these issues, in particular that the main difficulty people have is with navigating between topics. Another point which was brought up was that the "Back" button, present on every page, was not always clear. Some users may have found that going Back did not take them where they intended. The inclusion of a help/info feature was also suggested. To incorporate this feedback we created a Help/FAQ page which is always accessible via the side menu. This takes the form of a pop up modal that includes helpful questions and answers based on the feedback received through the survey.

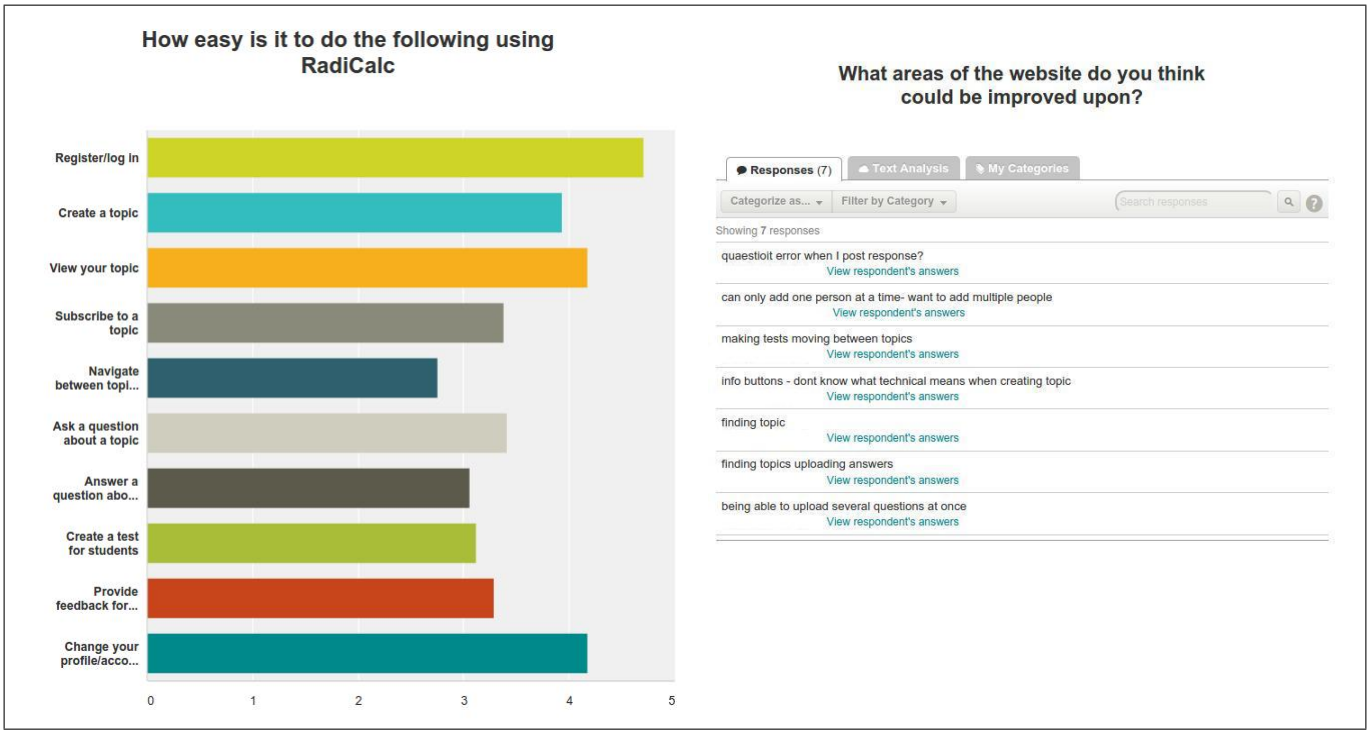


Figure 5.2: Results of Lecturer Survey

The results of the lecturer survey are generally similar to those of the student survey. Each feature has a relatively high score apart form navigation between topics and subtopics which fell below 3. The easiest features to use are:

- Registering/logging in: 29.4% responded "Easy", 70.6% responded "Very Easy"
- Creating a topic: 41.2% responded "Easy", 29.4% responded "Very Easy"
- Viewing a topic: 47.1% responded "Easy", 35.5% responded "Very Easy"
- Changing profile/account settings: 58.8% responded "Easy" and 29.4% responded "Very Easy"

The feedback received about asking and answering questions is similar for both students and lecturers. The results suggest that lecturers have a little more difficulty creating and submitting answers to tests. These problems are again reflected in the comments left by lecturers. Finding topics and creating tests seem to be the areas which need the most improvement. Again the suggestion of the info button has been raised. One user has also encountered a Quaestio-it error when attempting to respond to a question.

An issue that lecturer/admin users seem to face when using the website is that, when creating a test, questions have to be created individually. To incorporate this feedback we would ideally create a more efficient way of creating multiple test questions within the same web page, so this is one of the potential improvements we would have wanted to include.

5.5 Back-End Evaluation and Testing

We separated the tests into categories depending on whether they test models, controllers or features. We often found repetition between tests so we factored out the repetition into a helper folder. We also factored out the object instantiation code into a folder where we kept all the factories.

Below is a summary of the results of back-end testing. These results are accurate at the time of writing this report and are still subject to change as back end development continues.

- Models : 26 tests, 11 failing (42.3%)
- Controllers : 34 tests, 3 failing (8.8%)
- Features : 33 tests, 7 failing (21.2%)

The failure rate of these tests may be misleadingly high to give an informative indication of the correctness of our product. This is because problems with testing occurred when we decided to change the interface of the website. When we did that, many of the integration tests which were designed for the old interface started to fail. This happened for instance with a test which logged in a user and checked the components of the dashboard. When we changed the layout of the dashboard, this test was no longer accurate. We had the same problem with tests which created a tree structure of topics and checked that we can access a node's descendants and ancestors. While we created new tests for new interfaces before creating them, some of the old failing tests remained. As passing the old tests wasn't immediately necessary for the development of our project, we postponed fixing all of them.

5.6 Evaluation Tools And Techniques

5.6.1 Tools / Techniques Used

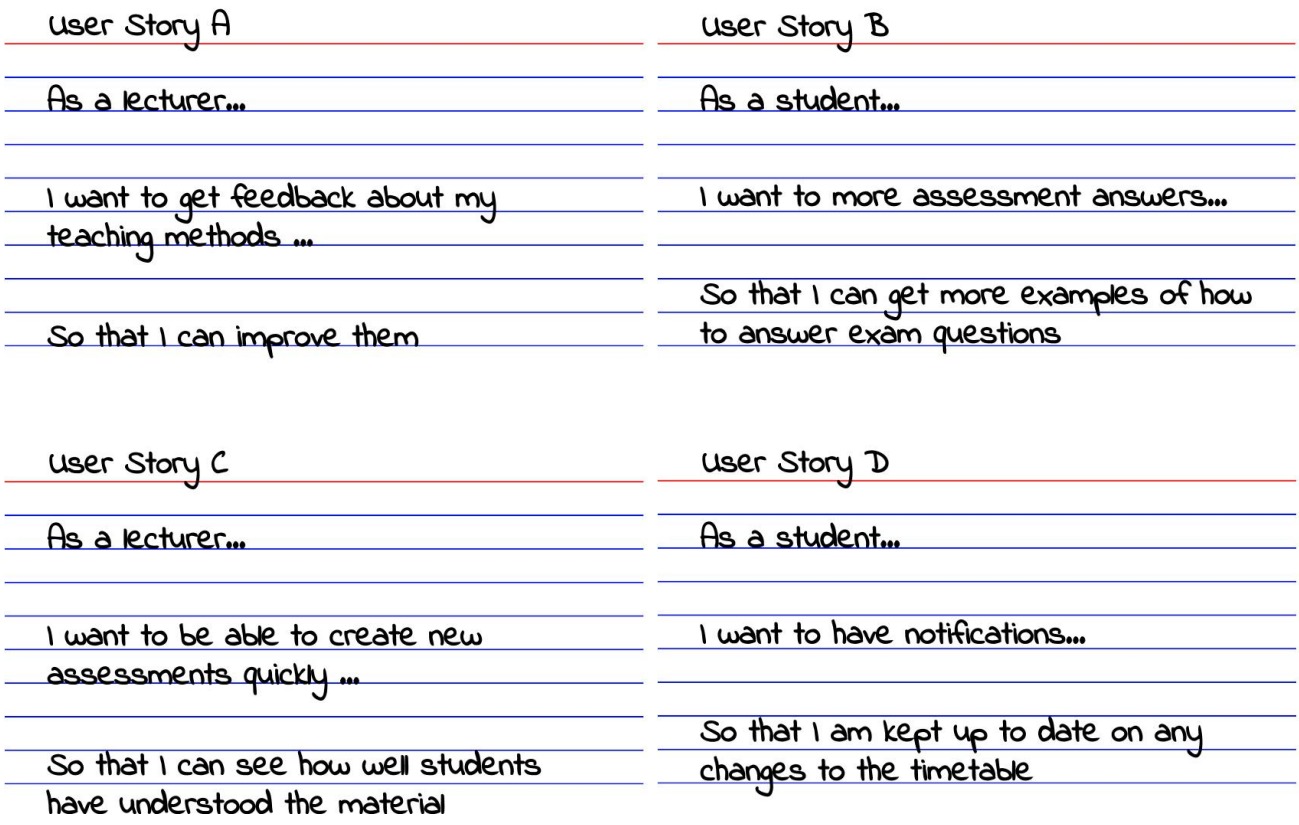
We used several evaluation techniques discussed in the lecture and from our own research in order to check that the product we were making was correct and successful.

At the beginning of each phase, we created a test suite to check the functionality of the feature, in order to ensure that it had the required functionality. Every time someone committed, the code was run through every test. If every test passed, then the code was deployed to Heroku, our cloud application platform.

We also created several prototypes; a paper prototype, which acts as the minimum viable product and allowed us to have a rough idea of the final design. A base HTML prototype (called hovercircles) was also created in order to quickly implement the front-end if the back-end were behind the front-end. The HTML prototypes could be easily incorporated into the back-end once it was completed. For the paper prototype, we drew the designs of each page and had a storyboard flowchart for the pages and then asked several students and the supervisor to try it out so we could gather feedback on the user experience and how the pages should interconnect (see Design and Implementation section 4.1.1).

The end product for the project was a minimal marketable product because time constraints determined the number of features that could be implemented, so it only incorporates the core features. In order to identify the core features, we created a value proposition canvas that helped us to understand what the consumers would want most out of our product. We could then compare the product to the value proposition canvas in order to ensure that the product was correct.

In the beginning, we interviewed a couple of students and lecturers to find out what they expected or wanted from the website. To further our understanding of the consumer, we created several user story cards from these interviews stating their requirements from the website :



5.6.2 Tools / Techniques Not Used

One of the evaluation techniques discussed in the lecture was to create multiple versions of the product and get users to give feedback on which one worked the best. We did not have time to use this method because of time constraints and because we improved on the design after each iteration, so there was not much need to create multiple designs. We would meet up with the supervisor and pose several different ideas as opposed to fully formed prototypes.

Creating a video to show how the product should work was not viable, again because of time constraints and because we felt the website itself was more conducive to receiving helpful feedback from potential users than a video.

From our research into impact maps, we felt that they included the use of too many assumptions, limiting the utility of this method. Since we had created a value proposition canvas, we felt that there was no more information to be gained from this technique.

We decided not to create a Business Model Canvas because it did not seem to have relevance to our project, as this is a short term project as opposed to a business plan.

Chapter 6

Conclusion And Future Extensions

6.1 Conclusion

6.1.1 What we have learned

Over the course of this project we have learned a great deal about group management, in particular we have come to understand the importance of holding productive meetings. During the first stages of the project, due to the fact that we were working with mostly new people, we learned each others' skill sets. This initially meant holding some longer meetings while we became familiar with the other members but this paid off because it facilitated our team organisation and delegation of tasks later on. We learned that in order to make a meeting more efficient and productive it was important to have a clear aim beforehand and some record of what was discussed. We learned that it is vital to leave the meeting with conclusive and precise instructions as to what each person would be doing for the next week.

As a team we have also learned about different agile software development methods and, perhaps most importantly, how to incorporate ideas from a variety of methodologies to match our group specifically. From experience we have seen that it is not the quantity of techniques used or meetings held which determines the efficiency of the group, but rather the engagement of each member in discussions and how active team members are in updating one another. As a result our group worked more smoothly because of the close communication between front-end and back-end teams.

Team members have also developed technical skills in different areas. The technologies used were familiar to some group members and not others, therefore some of us greatly improved our knowledge of these technologies in order to reach a similar standard to that of the experienced members. The front-end team worked with a completely unfamiliar API for Highcharts to produce the animated proficiency charts and this knowledge could be applicable to a wide range of web applications which require any sort of graph or chart. Also, by analysing the examples from the Codrops archive, the front-end team grasped how the JavaScript animations worked and how they could be used to create more fluid and interactive views.

6.1.2 What we would do differently

Given the knowledge that we have gained there are several things we could improve if we were to complete a similar task now. Firstly, we would have started development with our final selection of tools as this would save time. However this could not have been avoided at the time because it was the technical difficulties which we encountered that led us to change our choice of tools.

Team members could have been more active on Kanban. The front and back-end teams often communicated directly with each other before updating the interactive Kanban board to reflect these changes. We met with the supervisor regularly which was important to ensure the quality and correctness of our work. However, we could have also taken more time to meet as a group beforehand to decide more thoroughly what was to be discussed with the supervisor.

We could have put more time and effort into gathering feedback on the website earlier on, and from a wider selection of people. Had we received feedback as early as possible we might have been able to address some more of the issues with the front-end layout. Also, although we had a decent number of responses to our online surveys, we could have gained more information by actively publicising the website and the surveys.

Given more time we could have improved the layout of the website by fitting it more closely with the design, thereby having fewer different pages and fewer clicks between pages as intended.

6.2 Future Extensions

The following is a list of features we would like to see completed in the future:

- Implement what we did not manage to finish as listed in Chapter 5.
- Make the side menu customisable so people can more easily find pages that they frequently access.
- More gamification: a wider variety of achievement badges and possibly a levelling system for contribution to discussions.
- Incorporate more user feedback to improve the layout and navigation of the website.
- Allow users to select topics from a list of existing topics in order to subscribe to them.
- Allow users to search through user profiles, introduce contacts/friends and other social networking features
- Allow administrators to select and invite multiple users to a topic in one go.
- Implement synchronisation with the DoC teaching database to provide a better service to DoC staff and students.
- Implement mailers to notify users of changes on their subscribed topics/ invitations etc.
- Create our own internal version of Quaestio-it which is more tailored to the needs of this application. It would also run more smoothly and look better as it would not be accessed through an iframe.

Chapter 7

Bibliography

Research Into Current E-Learning Platforms

- bestelearningplatforms.com. Reviews of eLearning Platforms, Tools & Software for Teacher & Coaching. 2009-2014.
  http://bestelearningplatforms.com/software-tool-reviews/
- bestelearningplatforms.com. Reviews of eLearning Platforms, Tools & Software For Academic Institutions. 2009-2014.
  http://bestelearningplatforms.com/platform-reviews/
- 2012 OnlineUniversity.com. Online University: Todays Ten Most Important eLearning Platforms. n.d.
  http://www.onlineuniversity.net/elearning-platforms/
- 2003-2014 Education Portal. Best Online Learning Platform, People's Choice Awards. n.d.
  http://education-portal.com/articles/Best_Online_Learning_Platform_Peoples_Choice_Awards.html

Research Into Agile Software Development

- Don Wells. Agile Process Proverbs. 2009.
  http://www.agile-process.org/proverbs.html
- Martin Fowler. Using an Agile Software Process with Offshore Development. 18 July 2006.
  http://martinfowler.com/articles/agileOffshore.html
- Kelly Waters. What Is Agile? (10 Key Principles of Agile). 10 February 2007.
  http://www.allaboutagile.com/what-is-agile-10-key-principles/
- David Peterson. What is Kanban?. 2009.
  http://kanbanblog.com/explained/
- Atlassian. Do agile right. 2014.
  https://www.atlassian.com/agile/project-management
- Mike McLaughlin. Agile Methodologies for Software Development. 2014.
  http://www.versionone.com/Agile101/Agile-Development-Methodologies-Scrum-Kanban-Lean-XP/

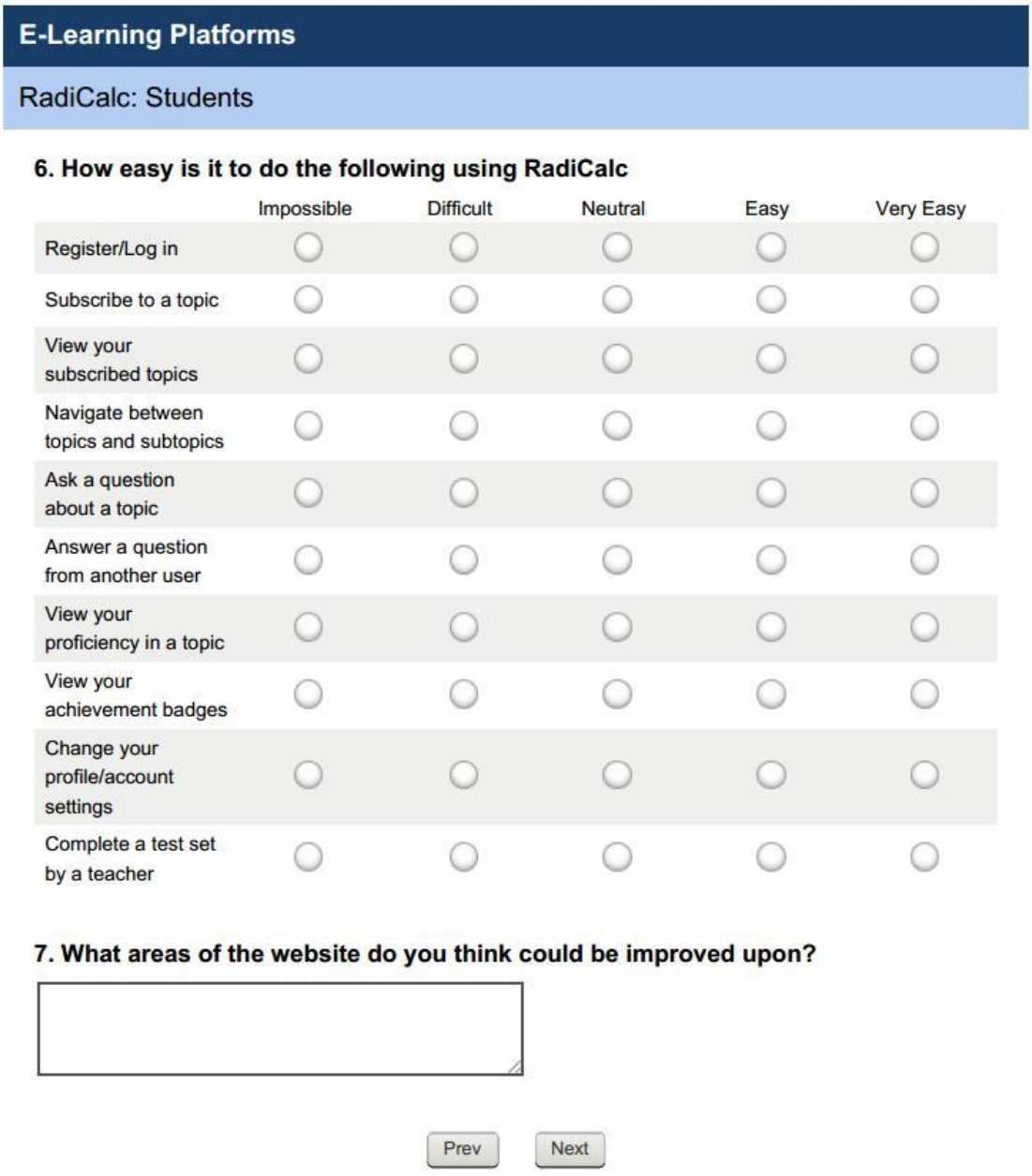
Research Into Evaluation Methods

- www.stattys.com, strategyzer.com. The Value Proposition Canvas. n.d.
  http://www.businessmodelgeneration.com/downloads/value_proposition_canvas.pdf
- Pichler, Roman. The Minimum Viable Product and the Minimal Marketable Product. 9th October 2013
  http://www.romanpichler.com/blog/minimum-viable-product-and-minimal-marketable-product/
- Leffingwell, D., Behrens, P. A User Story Primer. 2009.
  http://trailridgeconsulting.com/files/user-story-primer.pdf
- Optimizely. What is A/B Testing?. 2014.
  https://www.optimizely.com/ab-testing/
- Adzic, Gojko. Make a big impact with software products and projects!. 2012.
  http://www.impactmapping.org/

Appendices

Appendix A

SurveyMonkey Surveys



E-Learning Platforms

RadiCalc: Students

6. How easy is it to do the following using RadiCalc

	Impossible	Difficult	Neutral	Easy	Very Easy
Register/Log in	○	○	○	○	○
Subscribe to a topic	○	○	○	○	○
View your subscribed topics	○	○	○	○	○
Navigate between topics and subtopics	○	○	○	○	○
Ask a question about a topic	○	○	○	○	○
Answer a question from another user	○	○	○	○	○
View your proficiency in a topic	○	○	○	○	○
View your achievement badges	○	○	○	○	○
Change your profile/account settings	○	○	○	○	○
Complete a test set by a teacher	○	○	○	○	○

7. What areas of the website do you think could be improved upon?

Prev Next

E-Learning Platforms

RadiCalc: Teachers

8. How easy is it to do the following using RadiCalc

	Impossible	Difficult	Neutral	Easy	Very Easy
Register/log in	○	○	○	○	○
Create a topic	○	○	○	○	○
View your topic	○	○	○	○	○
Subscribe to a topic	○	○	○	○	○
Navigate between topics and subtopics	○	○	○	○	○
Ask a question about a topic	○	○	○	○	○
Answer a question about a topic	○	○	○	○	○
Create a test for students	○	○	○	○	○
Provide feedback for a test	○	○	○	○	○
Change your profile/account settings	○	○	○	○	○

9. What areas of the website do you think could be improved upon?

Prev Done

E-Learning Platforms

1. Please select which E-Learning Platforms you currently use

- [ ] Siminars
- [ ] Quora
- [ ] Udacity
- [ ] SlideWiki
- [ ] Aplia
- [ ] Skillshare
- [ ] Claroline
- [ ] Sakai
- [ ] Blackboard
- [ ] Piazza

Other (please specify)

2. How often do you use an E-Learning platform?

- ( ) Less than once a month
- ( ) Every month
- ( ) Every week
- ( ) Every day

3. What components/functions of these platforms do you think are helpful? Please provide comments below

4. Are there areas of these platforms which you think could be improved? Please provide comments below

5. How useful/important would it be to you for an E-Learning platform to have the following

	Not at all useful/important	Not very useful/important	Neutral	Very useful/important	Essential
Shows scores/grades to show your proficiency in a course	○	○	○	○	○
Provides multiple choice assessments	○	○	○	○	○
Provides open answer assessments	○	○	○	○	○
Provides an open discussion forum for students and teachers	○	○	○	○	○
Includes a mailbox for messaging with other users	○	○	○	○	○

Done

Appendix B

External Libraries

- Devise: Used for user authentication (https://github.com/plataformatec/devise)
- Curb: Used for Quaestio-it API calls (https://github.com/taf2/curb)
- Carrierwave: Used for uploading images (https://github.com/carrierwaveuploader/carrierwave)
- Fog: Used for user uploading images to Amazon S3 (https://github.com/fog/fog)
- Minimagick: Used for uploaded images processing (https://github.com/minimagick/minimagick)
- Authority: Used for access control (https://github.com/nathanl/authority)
- Rolify: Used to assign roles to users on topics (https://github.com/RolifyCommunity/rolify)
- Closure Tree: Used for organizing topics in a tree structure structure(https://github.com/mceachen/closure_tree)
- Factory Girl: Used in testing for instantiating data (https://github.com/thoughtbot/factory_girl)
- Capybara: Used for integration testing (https://github.com/jnicklas/capybara)

Appendix C

Front-End Designs

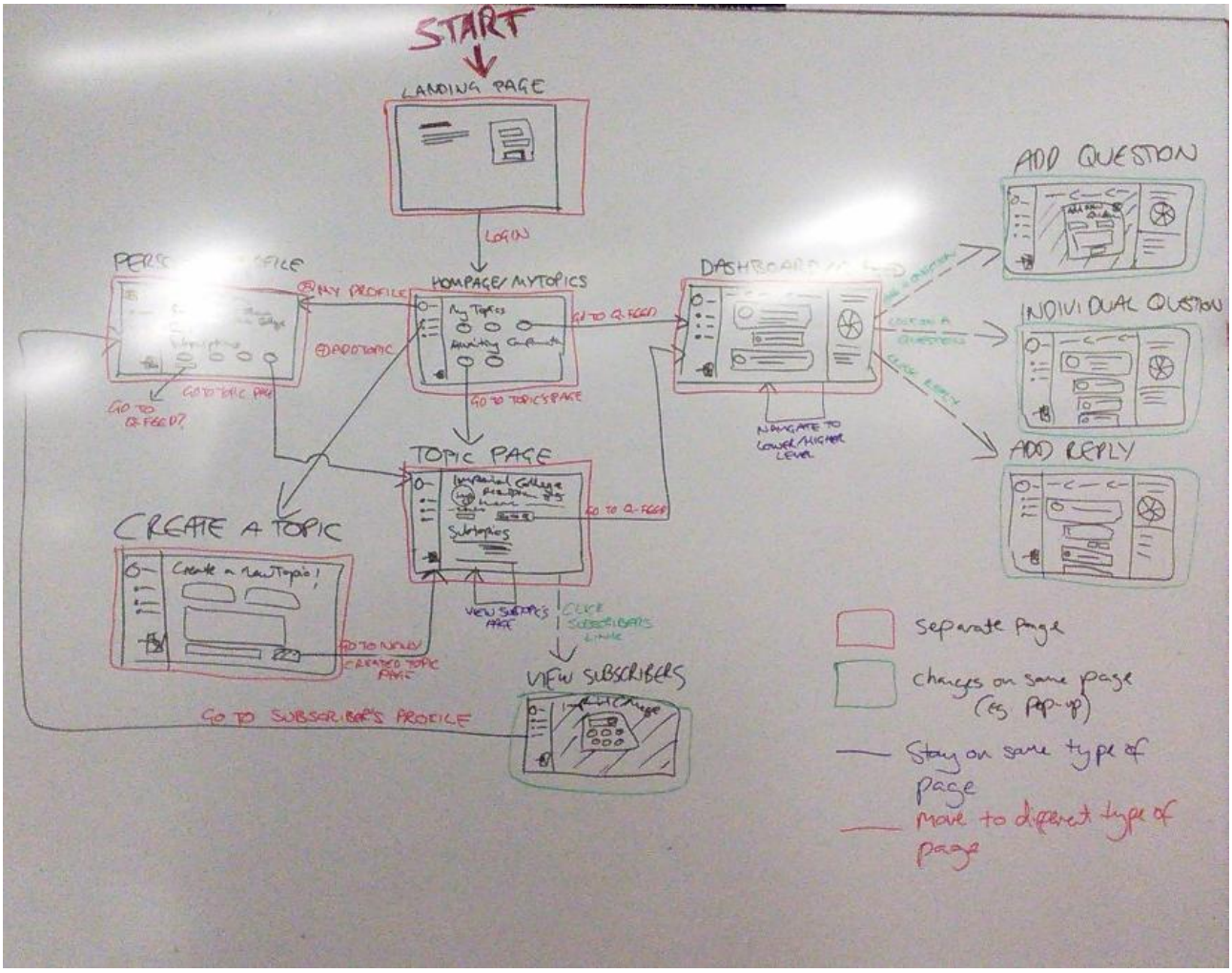


Figure C.1: Website Navigation Flowchart

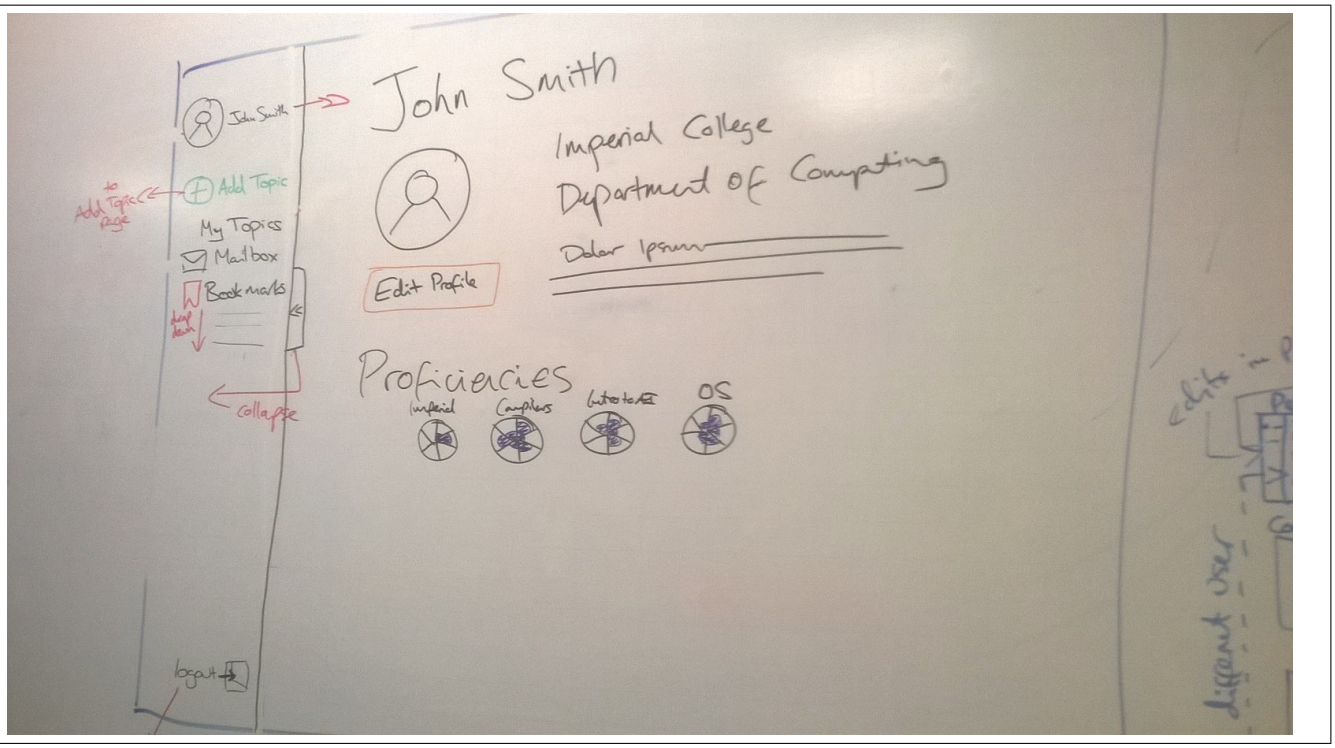


Figure C.2: Personal Profile

This diagram shows a profile page as seen by the user themselves. It shows a display picture, space for their personal details, an Edit Profile button and a list of their proficiencies in their subscribed topics.

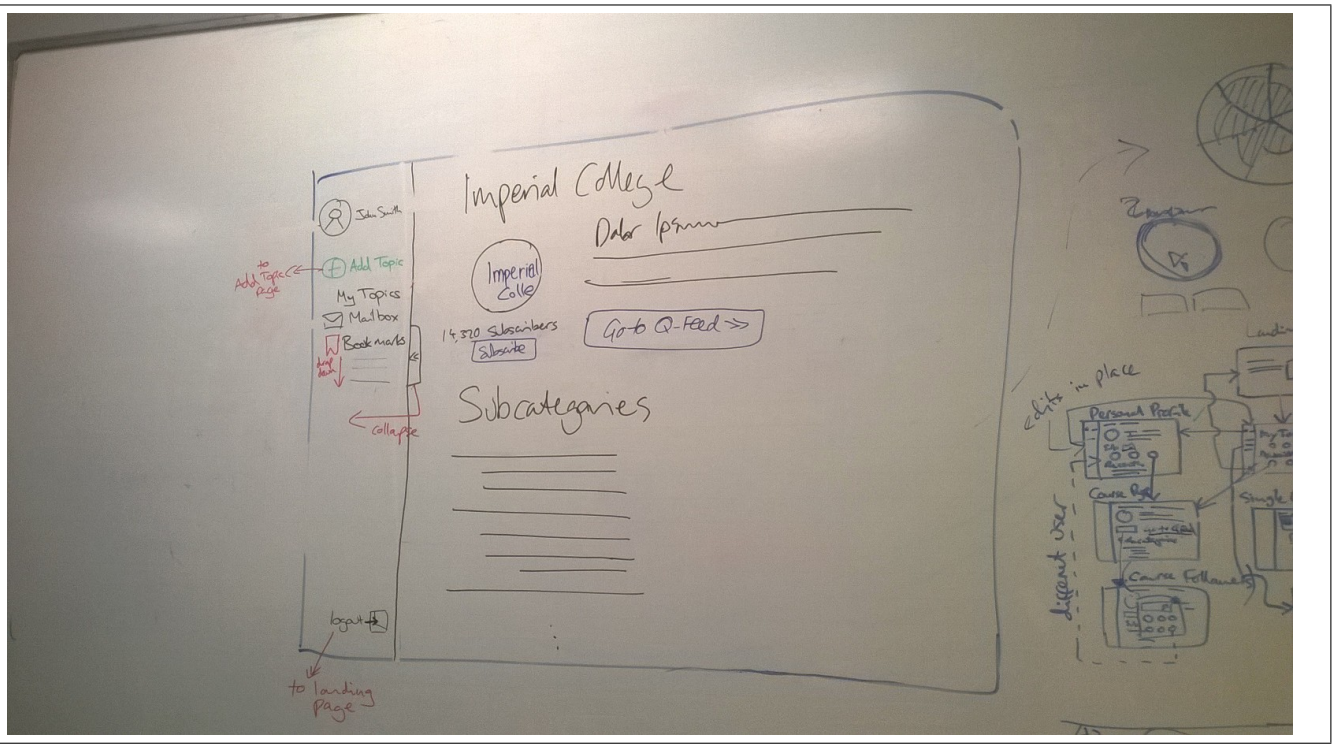


Figure C.3: Topic Page

The topic page design includes details about the topic, a link to the question feed, the option to Subscribe/Unsubscribe, access to the list of subscribers and a list of subtopics.

Appendix D

Screen Shots Of Final Product

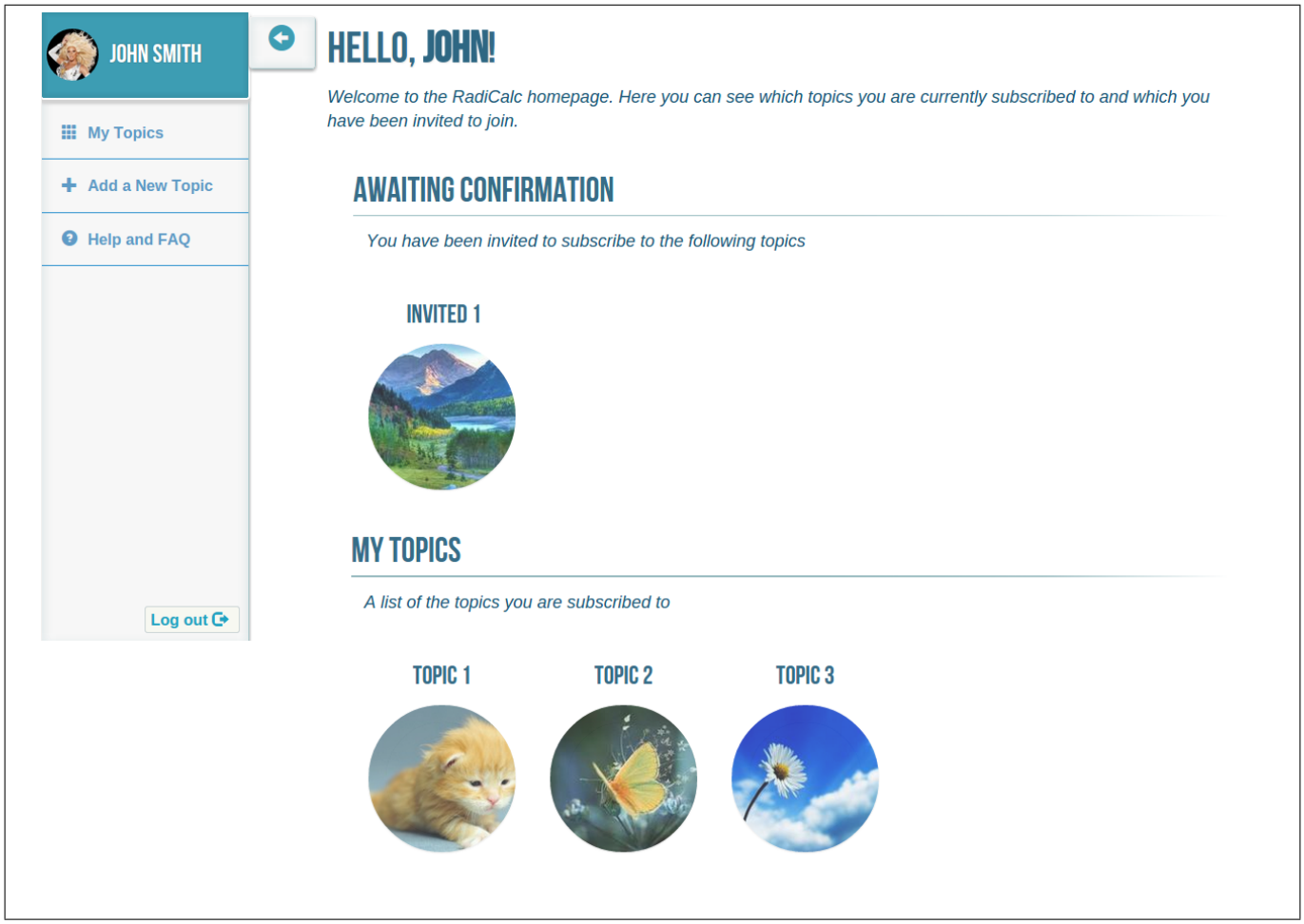


Figure D.1: My Topics Page (menu expanded)

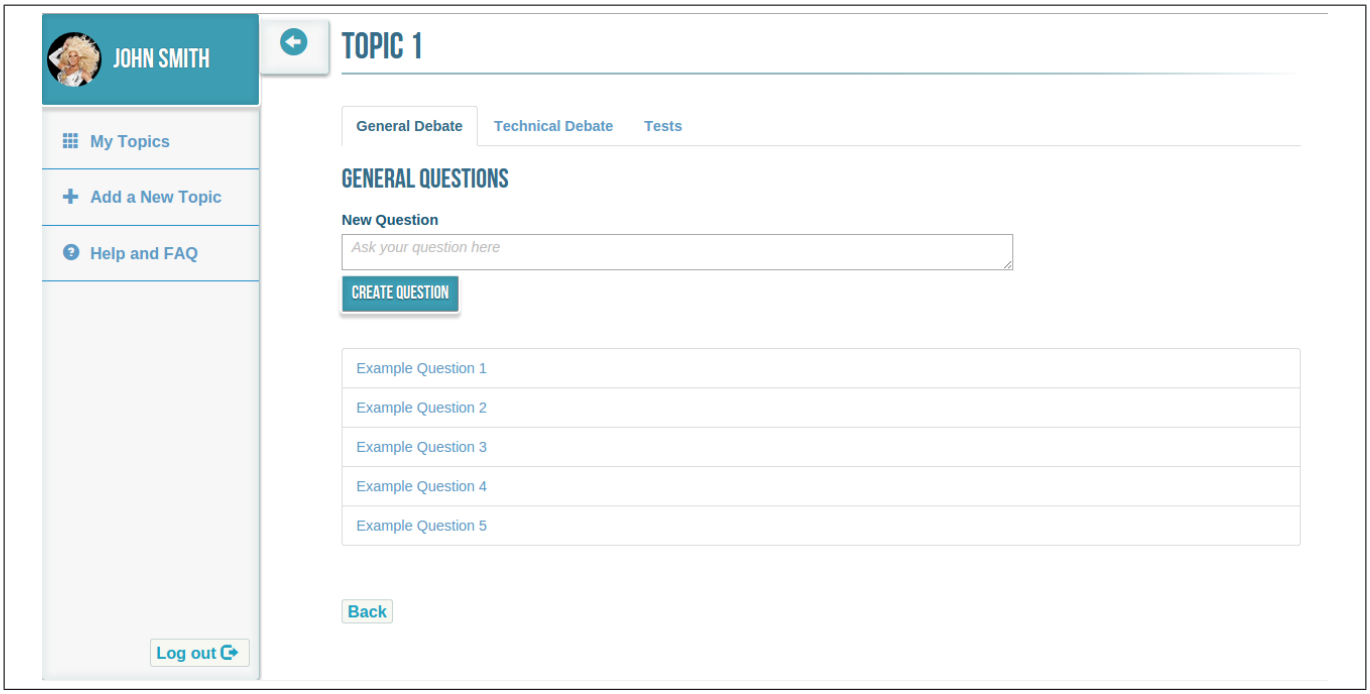


Figure D.2: Question Feed for Topic 1 (menu expanded)

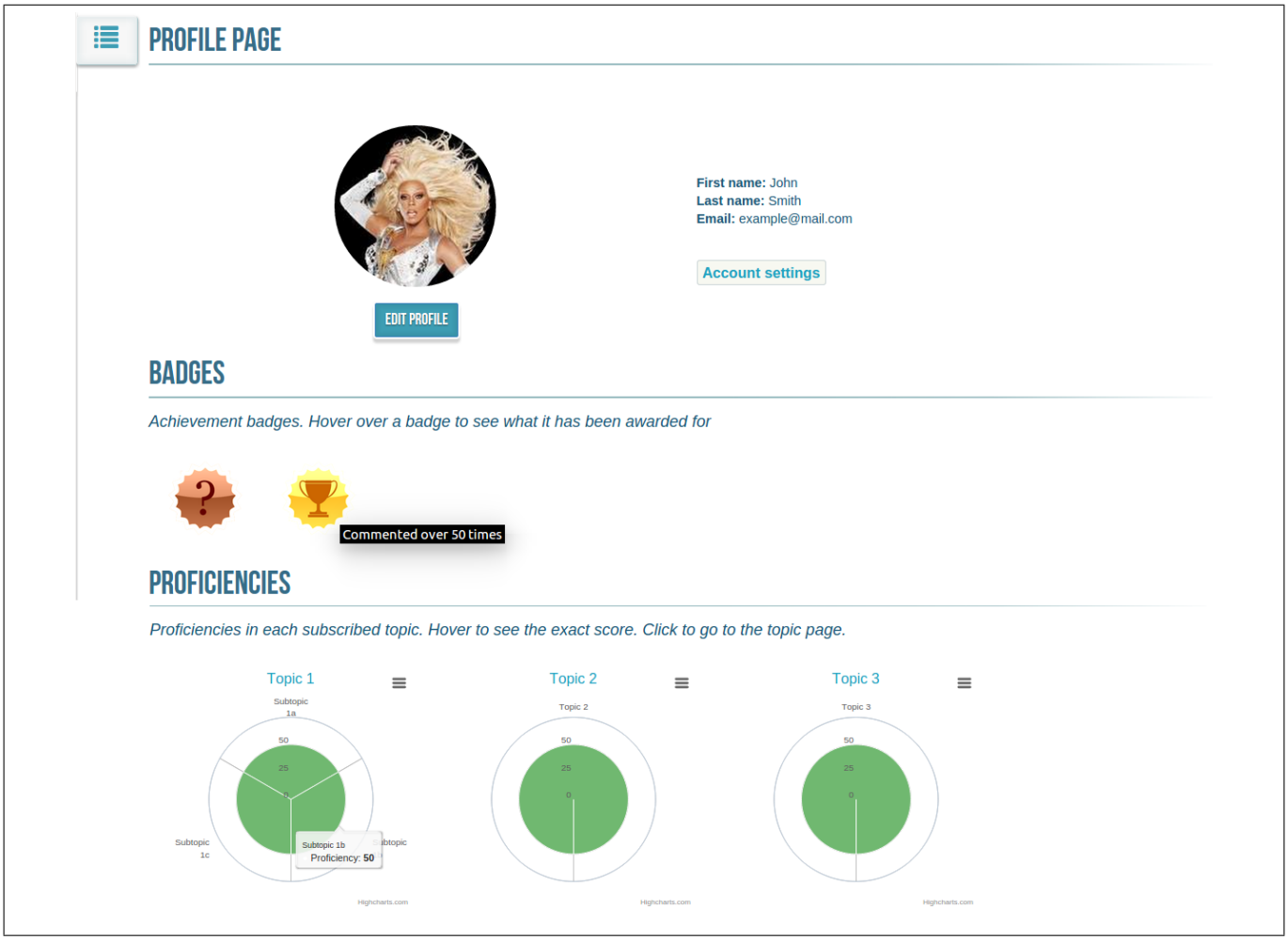


Figure D.3: Profile Page for John Smith (menu collapsed

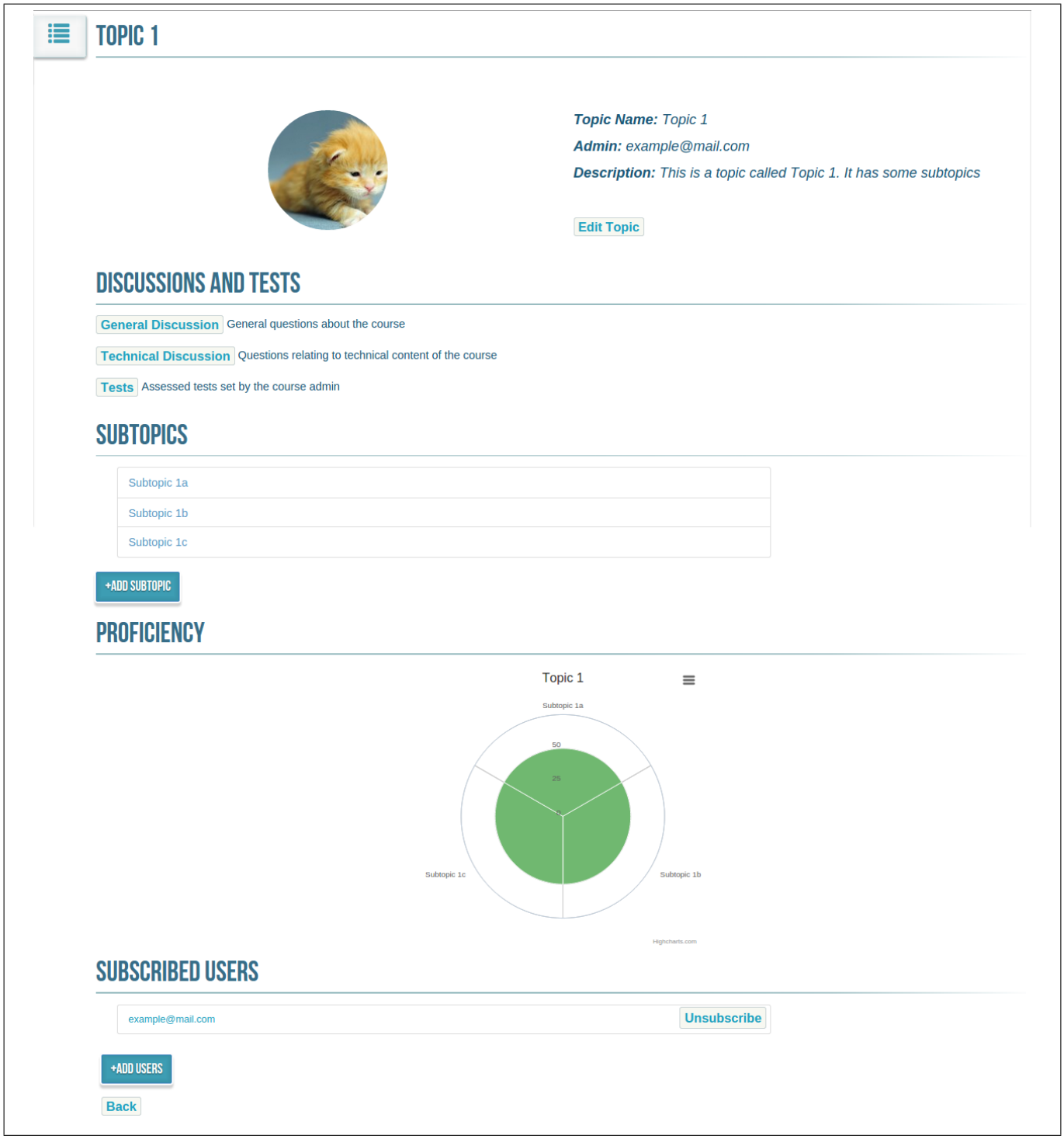


Figure D.4: Topic Page for Topic 1 (menu collapsed)

Appendix E

Logbook Entries

Figure E.1: Kanban Board after the website was completed

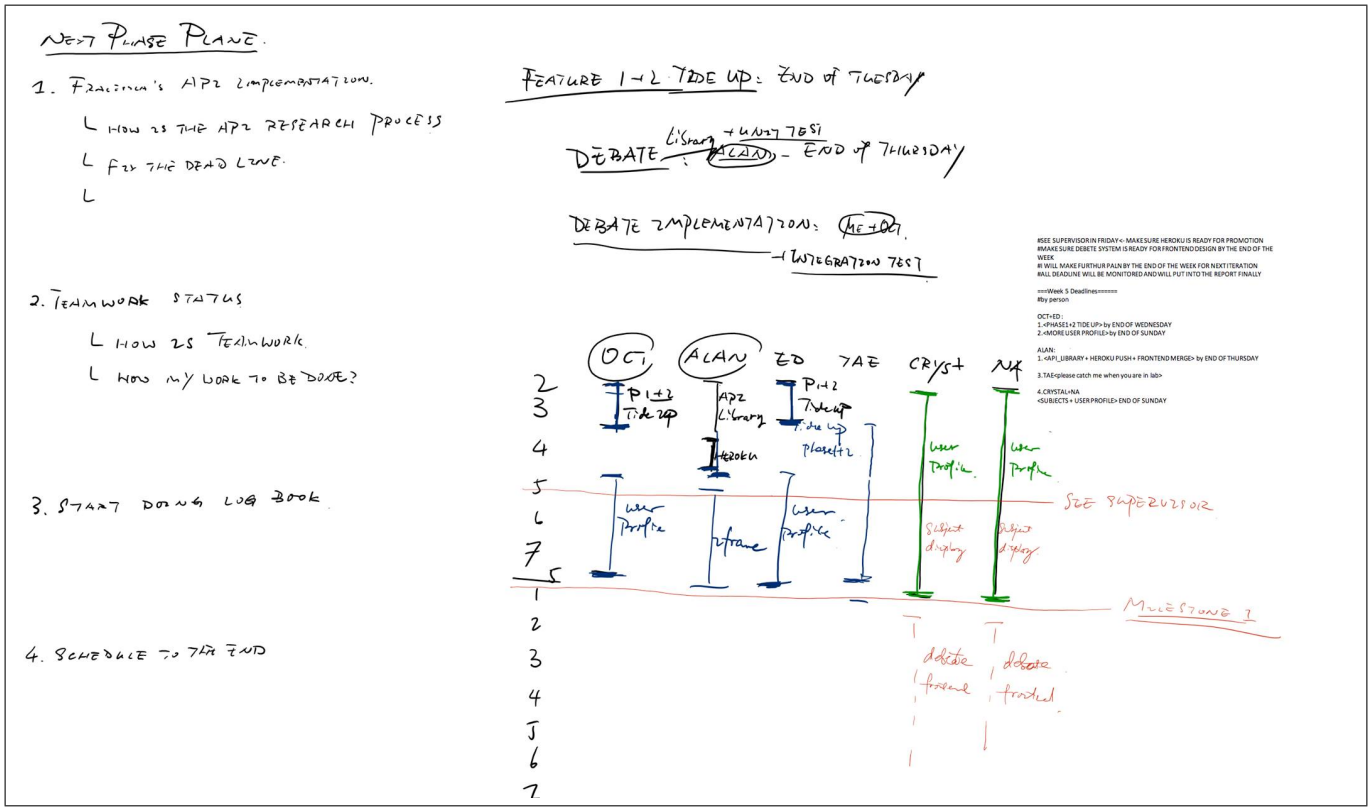


Figure E.2: Logbook entry from week 5

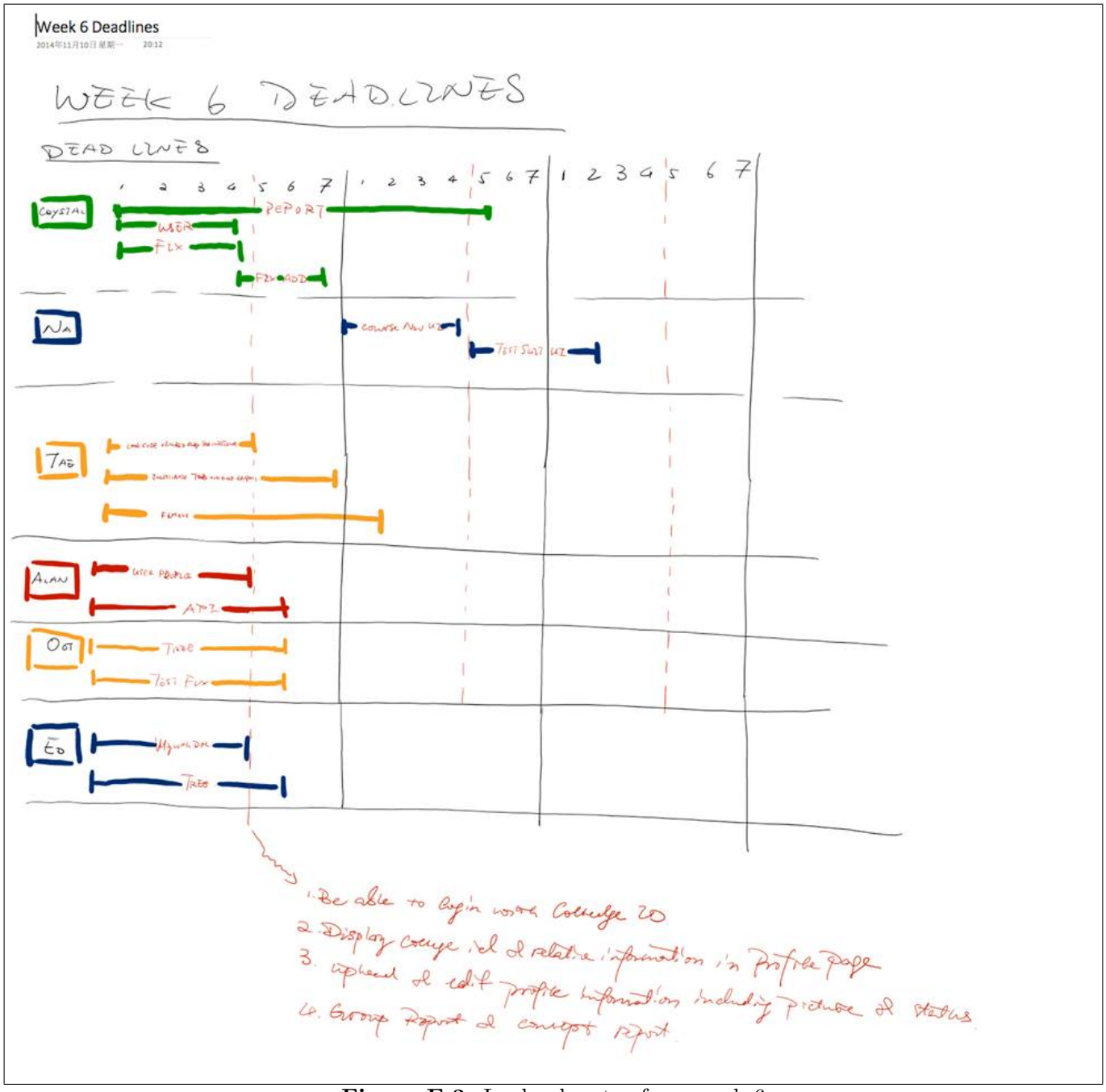


Figure E.3: Logbook entry from week 6