University of Glasgow | School of Computing Science

Team V - How Not To Kill Your Dog

Ross Adam
Andrew Gardner
Nicole Kearns
Mamas Nicolaou
Asset Sarsengaliyev

Level 3 Project — 18 March 2013

Abstract

This project aims to produce a learning application to be used within the University of Glasgow School of Veterinary Medicine by students wishing to revise and test their knowledge. The Django Web Framework was used to produce a web based application that is hosted in the veterinary school. It consists of both teaching slides with course content and affiliated questions. There is also administration functionality to allow the course co-ordinator to edit the content of the application to evolve with the prescribed course content.

Education Use Consent

We hereby give our permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____   Signature: _____

Name: _____   Signature: _____

Name: _____   Signature: _____

Name: _____   Signature: _____

Name: _____   Signature: _____

Name: _____   Signature: _____

Contents

Chapter 1

Introduction

1.1 Motivation

Team V is comprised of five students currently in their third year of a Computing Science degree at the University of Glasgow. The team will be working for 7 months on this project which is the sole piece of coursework for the Team Project 3 course. By the end of this period of time it is hoped that there will be a fully functional piece of software that can be delivered to our client.

This project is "How not to kill your dog" a macabre title for our veterinary student revision program. Our client is Dr Fiona Dowell, a senior lecturer at the University of Glasgow's Veterinary School. In addition to our client there are several more stakeholders for this application: the veterinary students who will be using this application throughout their studies; other lecturers and finally I.T. support staff who will interact with this application extensively. Dr Dowell found that her students struggle with learning how to do drug calculations more than they do with any other part of their course. She came up with the idea of having an application available to veterinary students to help enhance their drug calculation skills.

Dr Dowell believes that students will find it more entertaining to have a game-like learning application, which they could use in their free time to entertain themselves, but also learn and become better with their drug calculations. She would therefore like a program that can be used as a revision aid for students which she can add her own educational slides to and create tests and questions to engage students.

The motivation for developing this software was to bring all the education resources currently in use and create a more centralised location. Currently the client simply uses PowerPoint slides filled with information which her students can download from their respective Moodle site. Using our software would allow the user to forgo the need for specific software to read .ppt extension files and reduce compatibility issues. Also users do a lot of calculations and all tests with pen and paper; with new software it would be hoped that calculations could be performed online which would automatically compare them and tests would be submitted online for marking. This would result in faster and more efficient work for both students and lecturers. Dr Dowell also considered the possibility of promoting this application outside of the the veterinary school in hopes of further enhancing the reputation of the University of Glasgow.

1.2 Project Aim

The aim of the project was to create a learning application for veterinary students that allows them to learn new content and revise and refresh existing knowledge. As this project has been heavily client dependant the team has had several meetings with Dr Dowell to ensure that high quality software that meets her requirements has been produced.

Through this process we have identified these aims for the project:

- To centralise existing revision material.

- Provide a system students can use to learn and revise course content.

- Provide a system which can include informal test material to assure students of their knowledge.

- Allow the evolution of the application content to match course content through the adding, deleting and editing of topic slides and questions.

- Make the learning process engaging and enjoyable for users.

1.3 Dissertation Outline

The dissertation outlines in detail the process of carrying out the "How Not To Kill Your Dog" project. Below is an outline of our report:

Chapter 2 discusses existing applications and similar revision tools which we found useful in the design process.

Chapter (Requirements) discusses the requirements gathering process and outlines the functional and non-functional requirements for our application.

Chapter (Design) discusses the general design of the application.

Chapter (Implementation) discusses the implementation of the project.

Chapter (Testing)

Chapter (Evaluation)

Chapter 2

Background

This section discusses several different existing artifacts, both medical applications and other revision tools. These applications were selected to provide inspiration for our own application design, specifically assessing the current tools within the market and how each displays their content to the user.
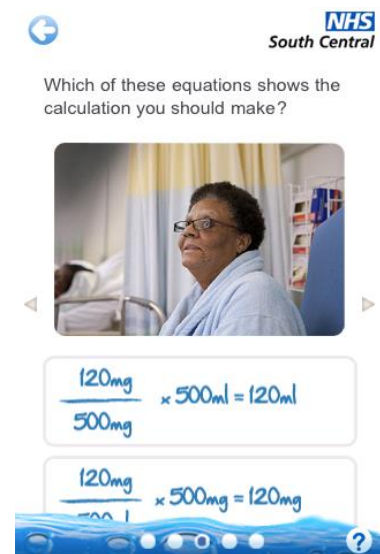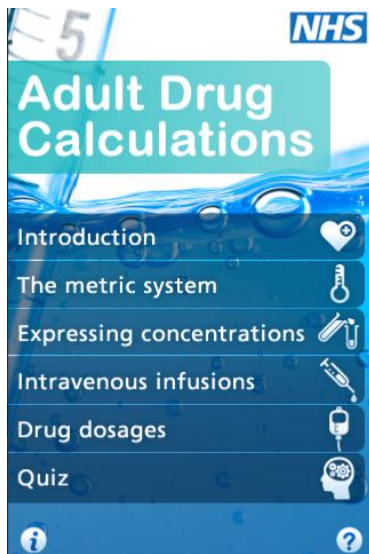
2.1 Investigation of Existing Artifacts

There are a few similar tools and learning resources available for vetinary students. This section discusses some of the current tools and a few tools which our client, Dr Fiona Dowell, has shown us in order to give us an idea of what she wanted for the application.

2.1.1 Medical Helper Applications

NHS Adult Drug Calculations

The NHS Adult Drug Calculations is a smartphone based learning tool aimed at nurses and midwives who are newly qualified or returning to practice. The tool allows the user to browse through different topics. Within each topic, there are a number of slides providing the user with the information and theory behind the calculations, and then has a few "Check your knowledge" slides containing multiple choice questions to test the user on what they have read. The interface is very simple and easy to use. The main menu allows the user to simply select which topic they qish to go over. Within each topic, there are arrows on each page which will allow the user to navigate back and forward between the pages and there are breadcrumbs at the bottom of each page indicating how far through the topic they are. There is also an arrow at the top left-hand corner which will allow the user to return to the main menu at any point. One feature which is particularly useful to new users is the help button, which informs the user how to use the application. This application is very similar to the application which our client would like for the vetinary students.

Adult Drug Calculations

NHS

Introduction
The metric system
Expressing concentrations
Intravenous infusions
Drug dosages
Quiz

NHS South Central

Which of these equations shows the calculation you should make?

$$\frac{120mg}{500mg} \times 500ml = 120ml$$

$$\frac{120mg}{500...} \times 500mg = 120mg$$

Vet Calculator

VetCalc

Fluids | Energy
Conversions | Flow Rates
Drugs | More

Basic Rate

Weight
15
Rate
4
Giving Set
Use giving set?   Yes
20 drops/ml
Calculate

Results

Rate
60 ml/hr
Drip Rate
20 drops/min
One drop every 3 seconds
3.3 drops every 10 seconds

VetCalc is an android based tool aimed at verterinarians, students, nurses and technicians, which allows the user to enter the necessary details, such as weight; dose; formulation; volume of fluid; etc. and will produce the result for the user. The interface for the vet calculator is very simple and easy to use. The menu is a simple grid-layout of different topics and sections for drug calculations. Within each topic, the user has to simply enter the necessary details into the clearly labelled text boxes and click 'Calculate' in order to get the result of the calcuation. On each page, there is a home button which will easily allow the user to return to the main menu at any point. This application is useful for users who simply need to get the results of a calculation. However, out client wants an application that will teach the students how to carry out the calculations and not simply produce the result for them as they will need to know how to carry out these calculations throughout their career.

2.1.2 Revision Tools

Before desgining our own application the team agreed that we should research existing tools for revision. The following were suggested as members had had some experience using them.

Bitesize

The Bitesize revision tool is a service provided by the BBC which operates in the user's web browser. It is a simple implementation of linked webpages that cover a variety of subjects from across the UK national curriculum. The team chose to research the Bitesize system because several of the Scottish Students had used it before and found it helpful for their studies at high school. First we decided to analyse how the content is displayed within the tool.

Figure 2.1: Bitesize Layout

Figure 1.1 shows the typical layout of a Bitesize page: each page of content is framed by the Bitesize logo (implemented as a clickable image that returns the user to the main page) and on the right of the screen is the index for various subjects allowing for quick navigation.

A consistent frame is a concept that Team V also wanted to achieve within our own design. Instead of a bitesize logo we would use the Veterinary School logo which would make our application appear more professional. Our index page will feature links to different topics for fast navigation for experienced users.

Figure 1.2 illustrates how each page is displayed in the tool. Content is simply loaded into the page in the center and navigation is achieved by clicking specific buttons. Our goal is to have a simple

Figure 2.2: Bitesize Content

slider to sequentially navigate pages.

Hot Potatoes

The second system Team V assessed was Hot Potatoes, a simple quiz creation system that allows users to create online quizzes which they can then publish for other users to interact with.

Figure 2.3: Hot Potatoes Program
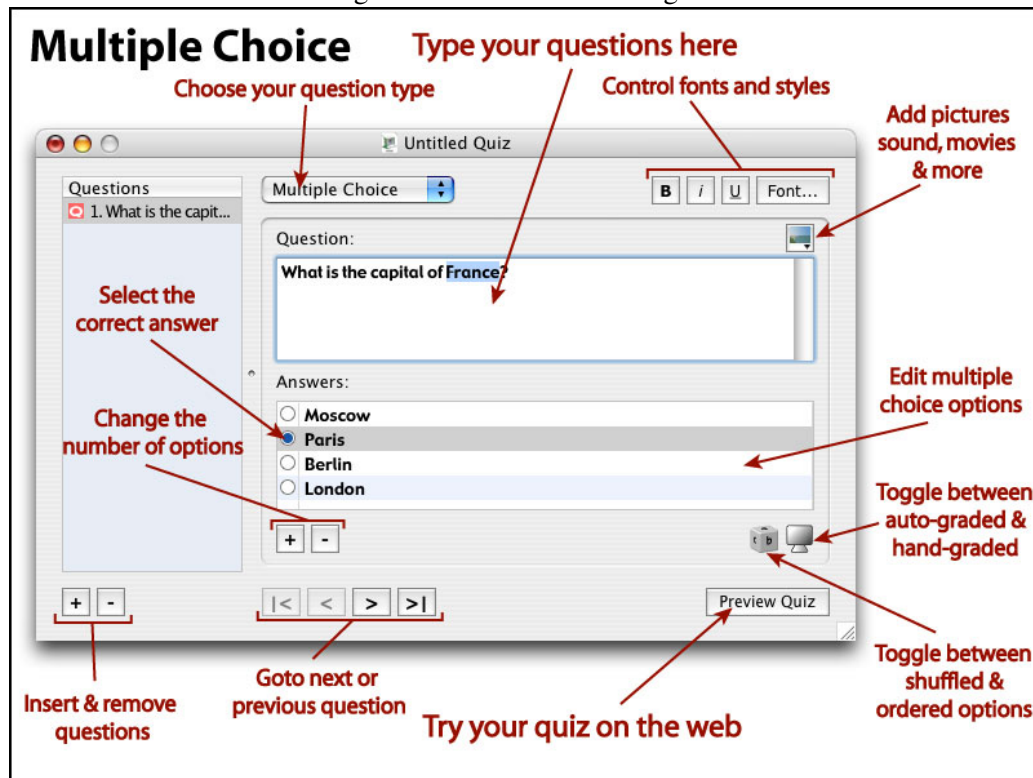
The system itself is a comprehensive suite of features and programs that can produce different types of quizzes: multiple choice; image matching and several more.

The included figure shows how a user can create a multiple choice question. This is much more complex than our implementation will be but it gave us an idea of how to make the process streamlined and easier for the user to work with.

The client has requested a system that includes questions which can be answered at the same time as the course content is being displayed. This would hopefully ensure the student retained the information better than simply reading the slides. Our system will only allow multiple choice questions to be created.

Chapter 3

Requirements

This section discusses the requirements elicitation process for our project. As we have an external-client, this process is important to ensure that we identify all requirements to produce the application that fully meets their needs.

3.1   Client Interview

The project was proposed by Dr Fiona Dowell, a senior lecturer at the University of Glasgow School of Veterinary Medicine. Initially, we met with Dr Fiona Dowell in order to gain more information about the requirements and specification for the application to ensure that we produced an application relevant to what the client wanted.
Our initial requirements for the vet application were :

- A learning application for vet students, providing step-by-step tutorials for different sections of the vet course.

- Simple example questions within each topic to test how much the user is learning about that specific section.

- A larger assessment at the end of all sections to test users on all the content within the application.

After the initial meeting with our client, we thought that it would be useful for the content within the application to be able to be updated , for example if the content changed; to fix errors or update to include different questions. We presented this idea to the client who agreed that this would be a useful feature as it might be good to be able to update after some user feedback. The functional requirements were then updated to include an administration system.
The administration system allows admin staff to upload and remove topics. It also allows the content of each topic to be updated, slides and questions can be added and removed.

3.2 Functional Requirements

3.2.1 Requirements Table

Student - Uses the application to browse through the different topics, answering the sample questions as they go along and completing the assessment at the end of all topics.

User Admin - Responsible for adding new users to the system, setting their permissions to determine their access levels and deleting them when necessay.

Content Admin - Responsible for adding new topics, slides and questions to the system, and ensuring that all content is kept up to date with the course content.

Student

Id	Requirement
SR1	View Topic
SR2	View Slides
SR3	View Questions
SR4	Answer Questions

User Admin

Id	Requirement
UAR1	Add User
UAR2	Delete User
UAR3	Set Permissions

Content Admin

Id	Requirement
CAR1	Add Topic
CAR2	Edit Topic
CAR3	Delete Topic
CAR4	Add Slide
CAR5	Delete Slide
CAR6	Add Questions
CAR7	Edit Questions
CAR8	Delete Questions

3.2.2   Use Case Diagrams
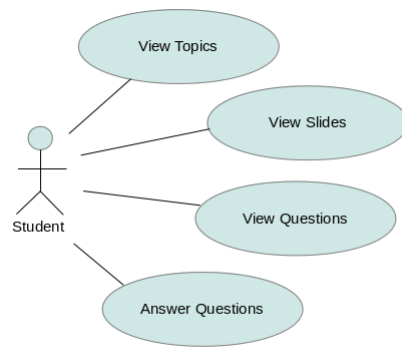
Student
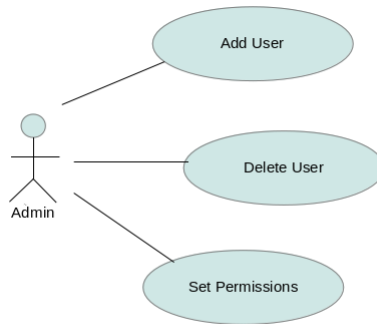
User Admin

3.2.3   Student Requirement Details

SR1: View Topics
Description: Users must be able to view the different topics in the application.
Rationale: The students must be able to view the different topics in the application. This will allow them to navigate through the different topics as the progress through the slides and topics.
Priority: Must Have

SR2: View Slides
Description: Users must be able to view the different slides in the application.
Rationale: The students must be able to view the different slides in the learning application. The student will be able to navigate back and forward through the different slides of a topic and read through the content on each page.
Priority: Must Have

SR3: View Questions
Description: Users must be able to view the different questions in the application.
Rationale: The student must be able to view the questions in the application. This will allow the students to go through the different questions related to the information given in the slides.
Priority: Must Have

SR4: Answer Questions
Description: Users must be able to answer the questions in the application.
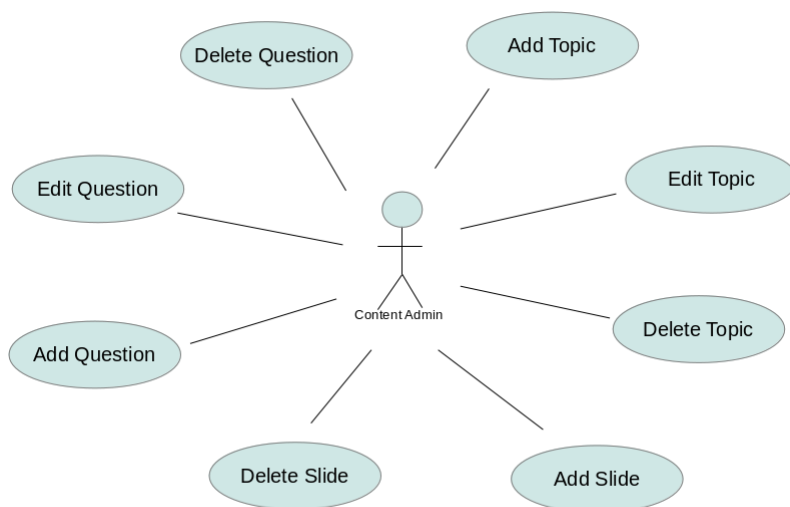**Rationale**: The student must be able to answer the questions in the learning application. This will allow the students to test the knowledge they learned from the information provided in the slides.
Priority: Must Have
Dependencies: SR3

3.2.4   User Admin Requirements Details

UAR1: Add User
Description: The admin user must be able to add a user.
Rationale: The admin users must be able to create a new user account to allow other users to use the application and have access to the content management.
Priority: Must have

UAR2: Delete User
Description: The admin user should be able to delete a user.
Rationale: The admin user should be able to remove users from the system, meaning that they will no longer be able to login or have access to any of the content management.

Priority: Should have
Dependencies: UAR1

UAR3: Set Permission
Description: The admin user must be able to set user permissions.
Rationale: The admin user must be able to set user permissions in order to provide users with the appropriate access.
Priority: Must have
Dependencies: UAR1

3.2.5   Content Admin Requirements Details

CAR1: Add Topic
Description: The admin users must be able to add a new topic.
Rationale: The admin user must be able to create a new topic within the admin site.
Priority: Must have

CAR2: Edit Topic
Description: The admin users should be able to edit a current topic.
Rationale: The admin user should be able to edit a current topic for the application. This will allow the admin user to update the topic if it any details change or are inaccurate.
Priority: Should Have
Dependencies: CAR1

CAR3: Delete Topic
Description: The admin users must be able to delete a current topic.
Rationale: The admin users must be able to remove a current topic from the application. This will mean that the topic will no longer be available within the application.
Priority: Must Have
Dependencies: CAR1

CAR4: Add Slide
Description: The admin users must be able to add a new slide.
Rationale: The admin users must be able to add new slides to the application. This will provide the content for the students to browse through.
Priority: Must Have

CA56: Delete Slide
Description: The admin users must be able to delete a slide.
Rationale: The admin users must be able to remove a slide from the application. This means that selected slide will no longer be available in the application.
Priority: Must have

Dependencies: CAR4

CAR6: Add Question
Description: The admin users must be able to add questions to the application.
Rationale: The admin users must be able to create new questions and add them to the application, allowing the students to test how much they have learned.
Priority: Must Have

CAR7: Edit Question
Description: The admin users should be able to edit questions.
Rationale: The admin users should be able to edit questions. The users should be able to update the questions to keep the questions up to date or to fix error.
Priority: Should Have
Dependencies: CAR6

CAR8: Delete Question
Description: The admin user must be able to delete questions.
Rationale: The admin user must be able to remove questions from the application. This will mean that the question is no longer available on the application for the students to view.
Priority: Must have
Dependencies: CAR6

3.3 Non-Functional Requirements

ID	Non-functional Requirement
NFR1	Availability
NFR2	Performance
NFR3	Educational
NFR4	Portability
NFR5	Concurrency

NFR1: Availability
As our application is a web-based application, it is important that users are able to access the application at all times, provided that the server it is running on is operational.

NFR2: Performance
Performance is an important part of the usability of our application. Our application should be able to load content from the database and display it within our application in a short amount of time. Failure to do so may result in the application being an inefficient and ineffective learning tool.

NFR3: Educational
The application should be educational to users, whether it be new information or revising the mate-

rial. Allowing the users to read through the slides and answer related questions should enhance the user's knowledge.

NFR4 : Portability

Having a web-based application means that our application is not restricted to a certain operating platform. This means that our application should be able to run on any platform without having to install additional software.

NFR5: Concurrency

As a study tool we know that many users will be accessing the software simultaneously when there are upcoming exams and tests. The application must therefore support concurrent user access to the system.

Chapter 4

Design

This chapter covers the various aspects of the application's design process. As our project was client-based the design was very much focused on satisfying the requirements laid out in the previous chapter. Fiona Dowell, our client from Glasgow University's Veterinary School, evaluated our initial user interface design and we then revised it according to the feedback that we received from her. When we were satisfied with the design we moved on to the implementation phase.

4.1 Design Goals and Considerations

The overall design goals for the application follow principles already well established in the field of web development.

These were:

1. **Precedence**
   The use of positioning, colour, size and contrast to naturally guide the user around the application. For example, most sites have a logo featured in the top left corner because studies have shown this to be the first place users look when accessing web sites. This enables natural navigation sequences: elements like the navigation bar being strategically placed so users' eyes will follow from the larger, relatively contrasted logo to the navigation bar.

2. **Navigation**
   The principle of making it easy and intuitive for users to navigate your web site. For example, links and buttons should be well positioned and clearly indicate their function while offering appropriate feedback when clicked. Measures should also be taken to inform the user where they are currently within the application.

3. **Typography**
   The text of a web application is the most common element of design and therefore deserves significant consideration. Font type, size, spacing and colour are essential to the application's overall clarity and readability.

4. **Usability**
   The encompassment of previously defined principles such as precedence, navigability and text. They relate to how usable the application is for its users. An important usability consideration is that of 'standards adherence'. This is the following of conventions laid out and used by millions of other websites, such as underlined text indicating a link.

5. **Consistency**
   The principle of making the application's design elements "match" coherently between pages and on the same page of the site. Attributes like colour and font choice, for example, should remain consistent across the site.

6. **Clarity**
   The appropriate use of design techniques such as positioning, contrast, font aliasing and others to make your design stand out crisply and clear.

4.1.1 Admin

The admin user is solely responsible for the content management of the application. With that in mind we approached the admin interface's design with a heavy focus on usability and navigation, rather than fancy themes and textual styles. We felt that the admin interface should possess a natural gravitas; that it should be taken seriously as a tool that controls the entire functionality of the application and not as a novel feature. To this extent our goal was to use a very simple theme, contrasting sharply in style and presentation with the student user interface. This was a wilful

breaking of the consistency design principle to enforce a measure of aforementioned seriousness onto the admin interface.

We also had to consider that the likely admins of the application would be relatively inexperienced with regards to administrating an entire web application's content. Being mindful of this we had to design the interface to be accommodating for the inexperienced, allowing for easy navigation and informative feedback with every action taken.

4.1.2 Student

In stark contrast to that of the admin user interface, the student's view of the web application had to embody all design goals previously defined in this chapter. They had to be able to navigate the site easily, knowing where they were at all times and how to get to where they would like to go with no fuss or ambiguity.

As this application's core functionality is assessment based it was important to have a gentle colour theme that would contrast naturally with the displayed content and not distract the user. This also held true with appropriate user feedback. We made an early design decision to provide the user with timely responsive feedback when answering questions.

An important design goal was to create an interactive learning experience for the user. In order to achieve this we decided to include an animal avatar with an associated health bar. A correct answer would have no effect whereas an incorrect answer would result in health loss. This design feature helps to engage the student on a personal level as their progress will affect the animal's lifespan.

4.2 Initial Design

4.2.1 ER Diagram

After establishing our initial design goals we created an entity relationship diagram. Defining our data models at this phase allowed us to clearly understand the distinction between entities and how they relate to each other.

4.2.2 Paper Prototypes

Paper prototyping was an effective and inexpensive method used to determine the design direction of the student user interface. The use of paper prototypes allowed us to experiment with various design configurations without the overhead of time and resources. This method of design was particularly suited to this application as it was simple to review and to iterate upon. Client involvement was a requirement for the success of the project. Therefore it was imperative that our early design ideas were subject to client review and feedback. Sending several prototypes for client evaluation meant that there was more potential for varied, constructive feedback. Receiving said feedback from the client afforded us more opportunity to revise our designs in accordance with their wishes.

Figure 4.1: Application ER Diagram
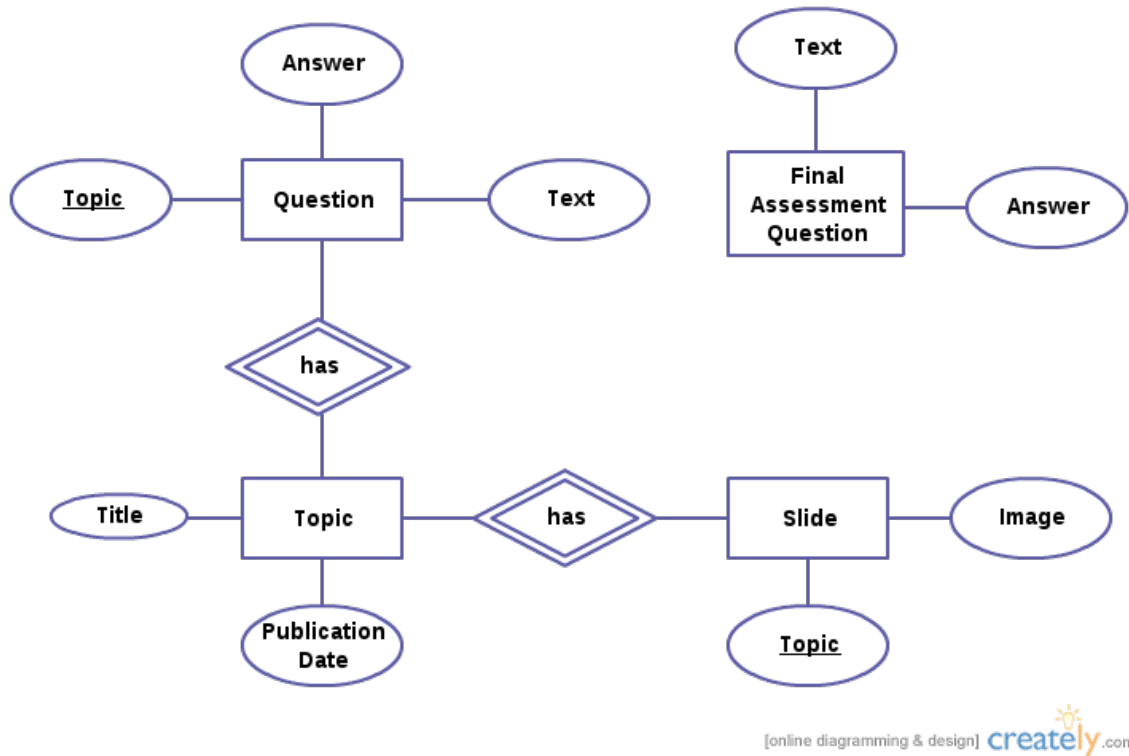
The paper prototypes aimed to fulfil the student user's functional requirements as laid out in the previous chapter. This meant that we had to design mechanisms for the following use cases:

- Viewing Topics **(SR1)**

- Viewing Slides **(SR2)**

- Viewing Questions **(SR3)**

- Answering Questions **(SR4)**

As will be discussed in the Prototype Evaluation section, our client indicated that two prototypes were particularly suitable in meeting the requirements of the application.

The above-left image (**Fig 1.1**) shows the individual topic view for the student user interface. The "title" at the top of the page represents the topic's name, with the slides for the topic being shown in the "content" section. The arrows contained within the content section represents the possibility of a slider being used, enabling the user to easily switch back and forth between slides. The breadcrumbs, shown below the content box, were designed to enhance user navigability by allowing the user to directly select a slide to navigate to. The rounded buttons to the right of the content section represent clickable links to navigate between individual topics. These design elements satisfy requirements **SR1** and **SR2**.

This prototype satisfies design goals laid out in the beginning of this chapter. The judicious use of whitespace enhances the user interface's clarity. With regards to design precedence, the size of the

Figure 4.2: Topic View
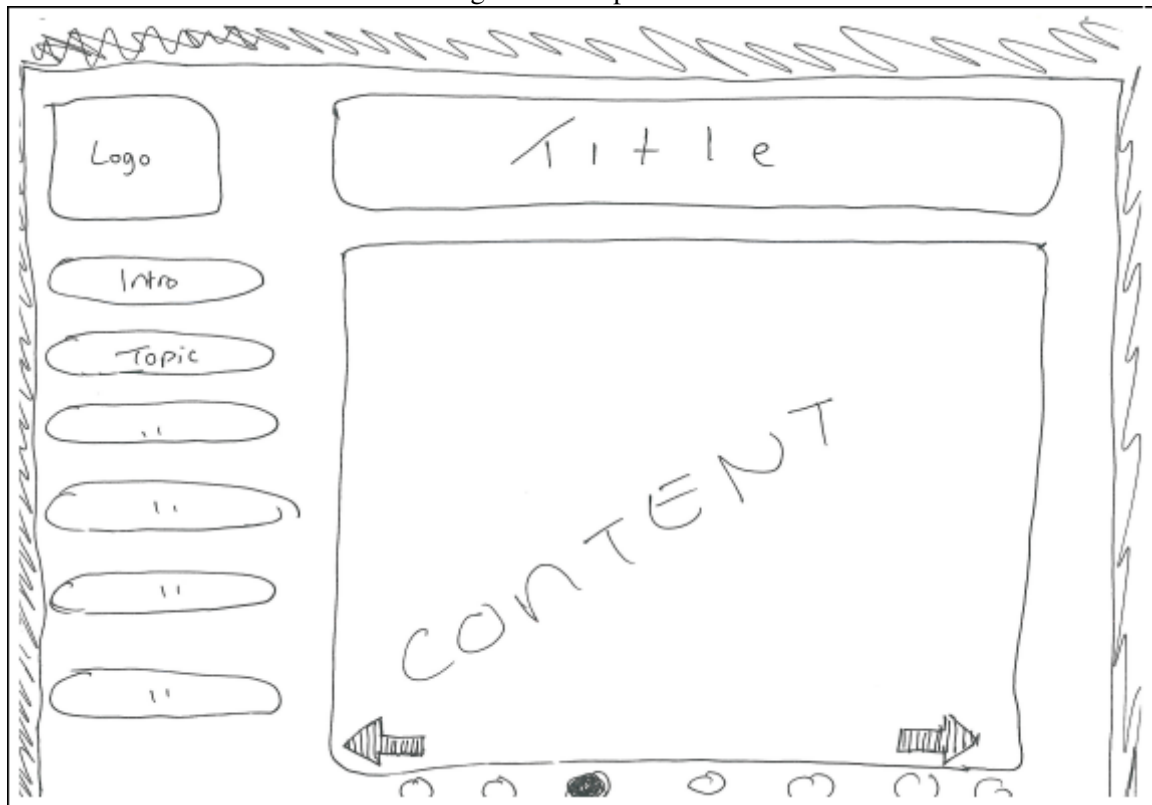
content box (compared to other design features) functions as a focal point, demanding the user's attention.

The above-right image (**Fig 1.2**) shows the final assessment page for the student user interface. The inclusion of an animal avatar and health bar satisfies the design goal of interactive student engagement, resulting in a heightened learning experience. Similar to **Figure 1.1**, the use of whitespace and size precedence allows the user to understand and easily navigate the interface presented to them. The problem description section represents the functional requirement **SR3** and the box below plus the apply/inject button represents the functional requirement **SR4**.

We chose not to create a paper prototype of the admin interface because we knew that Django (the web framework we had chosen to implement the application with) would provide a suitable pre-defined interface. This will be discussed further in the Admin User Interface Subsection.

4.2.3   Prototype Evaluation

The evaluation process involved sending the paper prototypes to our client, Fiona Dowell. This was an important part of the design process as her feedback would naturally impact the application's final design.

With regards to the paper prototypes, the feedback confirmed our initial design choices. Both designs preferred by our team were also chosen by Fiona and her colleagues. The decision to revise and continue with these designs was reinforced by our client's feedback. This not only resulted in

Figure 4.3: Final Assessment
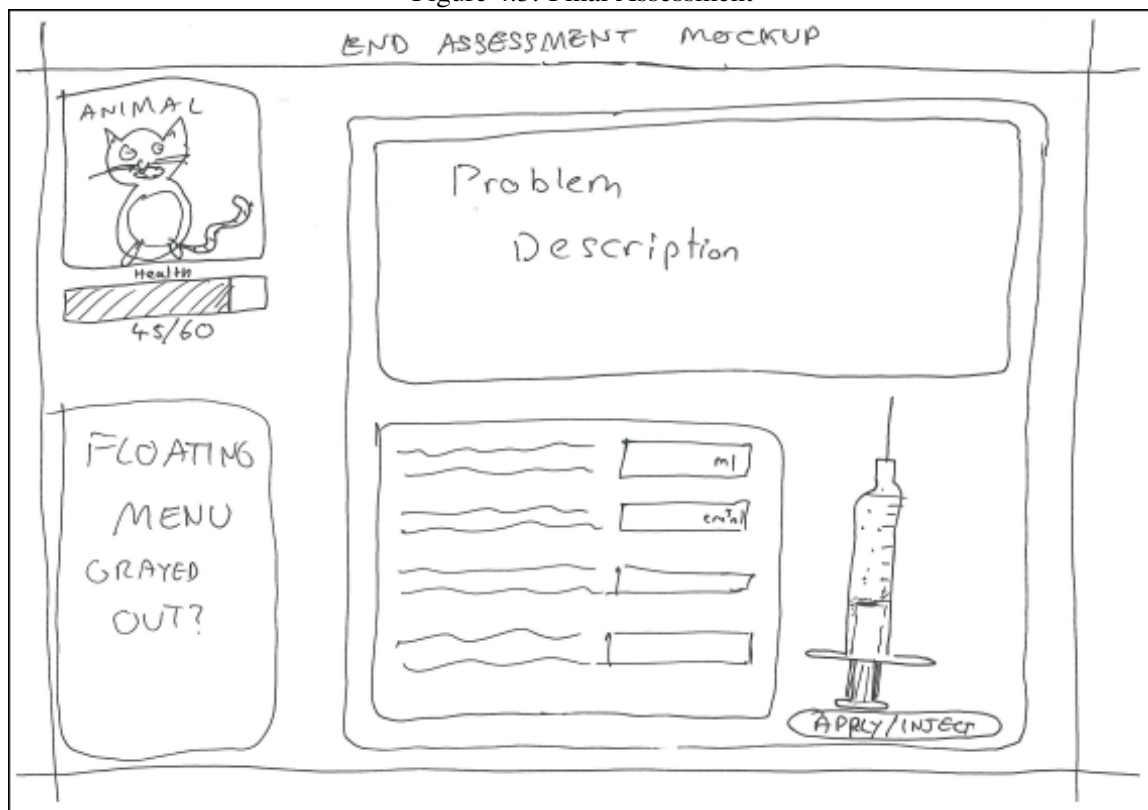
an easy design decision but also supported the overall design direction we had previously agreed on. While we knew further iteration and changes were likely required for implementation, we felt this was a solid base to start from.

4.3 Interface Design

This section covers the user interfaces for both admins and students, analysing how the design meets specific functional requirements and overall design goals.

4.3.1 Admin User Interface

Figure (x.x) shows the main page of the admin interface after the admin user has logged in. The interface's specific design theme was largely out of our hands as Django provided this for us. The lack of design control wasn't an issue however, as it conformed to our earlier design goals that the interface should be sufficiently different in theme and style from the student user interface. As figure (x.x) shows, the overall theme is quite minimalistic. There is no extraneous details or features that could distract the admin user.

Annotation 1 shows the welcome message and menu for the user that has just logged into admin interface. It's placement at the top right hand side of the screen adheres to current best practise in

25

Figure 4.4: Admin User Interface

web development, which in turn increases the usability of the interface. In terms of typography, the use of bold text confidently directs the user's eye towards expected features such as "Change password" and "Log out".

Annotation 2 highlights several of the user administration use cases in that they should be able to add, remove and edit current users within the application. These options are displayed logically by being grouped together and have icons that adhere to common user interface idioms. For example the ability to add a user is annotated with a plus symbol, enabling inexperienced users to intuitely deduce how that function would operate.

Annotation 3 covers the content administrator's functional requirements in being able to add, edit and remove topic content. This annotation's style is identical to that of annotation 2, in that it contributes to the overall consistency of the admin user interface, providing a clear and usable experience for the user. As with annotation 2 a clearly recognisable icon (a pencil) is used to indicate that the link in question leads to editing options for the topics.

Annotation 4 shows the "Recent Actions" bar of the interface. This enables admin users to see all their previous administrative actions for the application. This is very useful as it allows the administrator to track down previous changes to content and navigate to them directly. This navigation is further enhanced by the blue colour of the text, indicating that the text is a link and can be clicked. This blue text link consistency greatly adds to the expected functionality and to the usability of the interface as a whole.

The only design issue we faced with the admin user interface was the lack of creative control over the look and feel. While we still felt it was appropriate and suitably met our design considerations, it would have been better if Django allowed us greater customisation.

Figure 4.5: Content Screen
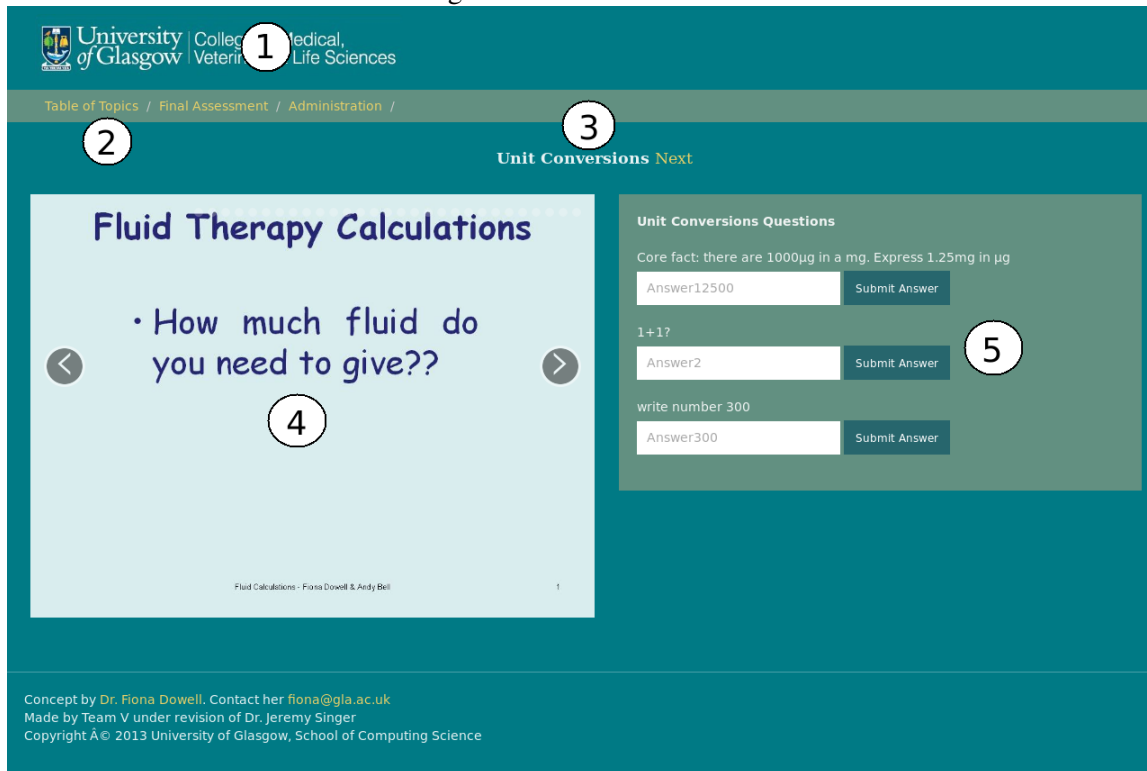
4.3.2 Student User Interface

Main Content Screen

Figure x.4 shows the main page users will see when they interact with the system.

Annotation 1 highlights the image of the University of Glasgow's Veterinary School logo. This was included for two reasons: firstly it identifies the University to external bodies that may stumble across the application. Secondly it makes it appear more professional and therefore improves the overall aesthetics.

Annotation 2 shows the application's main navigation section. These are clearly labelled for clarity to identify their purpose. This also has the additional property of increasing usability. Expert users such as students who have used the program before and want to jump to a specific page can utilise these navigational tools quickly.

Annotation 3 is the sequential navigator for proceeding through topics. Depending on the topic selected there will also be a back button included. This function once again increases the usability of the application as well as adding to consistency to the design by providing a constant frame for content.

Annotation 4 is the final function available for navigation. This is our main content window where topic slides are displayed. When the mouse is hovered over the slide an arrow appears on either side of the screen to facilitate navigation. The use of arrows promotes recognition rather than recall for the user.

Annotation 5 shows the questions associated with each topic. Each question is affiliated with a slide and users can answer them in their own time. This promotes the interactive learning element of the application. Each question gives instant feedback upon submission by altering the text in the box to display "Correct Answer" or "Wrong Answer."

Final Assessment Screen

Figure 4.6: Final Assessment

Figure x.5 shows the assessment section of the application which presumably will be accessed when the user feels suitably prepared for it.

Annotation 1 shows an example of a question being answered wrong. Feedback is instantaneous with the question background changed to a subtle red colour, this is an intentional design decision to keep users engaged and willing to learn without discouraging them. Textual feedback is also provided for futher clarification.

Annotation 2 shows an answer that is correct. This time the colour has changed to a slightly lighter green colour. Both the correct and incorrect colours are intrusive and fit with the design so as to keep the overall look pleasing.

The following three annotations illustrate the gamification of our application. By including the following elements we hope to engage users more effectively and provide an insentive to learn. Firstly they will attain a higher score and secondly they will keep the animal alive.

Annotation 3 is a progress bar which portrays the remaining health of the animal. For every wrong answer the bar will decrease indicating you hurt the animal (as a vet would do if they got these

calculations wrong in the real world). Insentive to keep the animal alive will hopefully encourage the suer to get as many right answers as possible.

Annotation 4 is an image of the animal the students attempting to keep alive. For every decrement of 30 points the image changes to a steadily unhealthier looking donkey eventually resulting in a "dead" donkey. As with annotation 3 it is included to spur users to anser questions correctly.

Annotation 5 shows the users current score. This increases by ten points for every correct answer submitted and does not increase with wrong answers.

4.4 Conclusion

The main focus of the design for the entirety of the application was to provide an interface that would be easy to use and reduce user frustration. Throughout development the idea of keeping users engaged with their learning and the content has been at the forefront and with the inclusion of a "game" of sorts on the final assessment the team believes they have succeeded in achieving this.

Chapter 5

Implementation

5.1 Abstract

5.2 Developing a Web Based Application

One of the requirements for our project was that our final product should be available to use to any user with an Internet Connection. More specifically, any student should be able to access the application and practice on their drug calculations in their free time wherever they are with any device that can access the Web. This can be a laptop computer, a mobile phone or a tablet computer.
It was, therefore, clear to us that creating an application that needs to be installed on a machine in order to be usable was not an option. After a discussion with the team we decided that the most suitable solution would be to implement a application that would be accessible via a Web Browser. A web application would make it possible for any user to access our application from any device simply by navigating to our Web Applications URL address.

5.2.1 Using a Web Application Framework

After doing some research looking for the best possible option for developing web applications we decided that we would make use of a web development framework rather than writing the code for the whole application from scratch.

Developing from Scratch

Developing an application from scratch has the potential to take up much needed time from the Development Phase. For each different page of an application there needs to be a unique file (for example an html document) that will be sent to the client's web browser when the page is called. An application usually has the same layout on every page with basic components, like the applications logo or the applications default buttons, being displayed on each page. It is therefore clear that having a separate file to correspond to each different page of an application can lead to a lot of

coding overhead and 'boiler plate' code. In the case of a change request, a developer will need to modify the code in each different page separately which would take much of our development time. Time was a major concern for us as we only had a limited amount of time to get the application designed, developed, evaluated and delivered to our client.

Developing with a web application framework

A Web Application Framework is a set of prefabricated software building blocks that programmers can use, extend, or customise for specific computing solutions" (Leif Azzoperdi, DIM3 Lecture 3). A framework would allow us to start implementing our application with a concrete base to support us by providing default functionality whilst allowing us to extend or override functionality to suit our specific requirements. "One significant advantage to using a framework is that you're required to write only a minimal amount of code to get up and running from scratch"(Professional Python Frameworks: Web 2.0 Programming with Django and Turbogears page 48). This was one of the most important reasons for which we opted in using a Web Application Framework for our project. In addition, web appliccation freamworks support software reuse and component based programming which can help us separate concerns in our web application by breaking it down into different components. Afterwards, there was one more decision that needed to be undertaken. There are a number of frameworks available on the web so we had to decide on one of them before starting our implementation.

5.2.2   Which framework to use?

We came down to three popular web application frameworks that would help us develop our software but we had to decide on one of them. Our supervisor suggested that three of us take one of the three frameworks each and try to build a simple application in one week. This would not only show us what can be built with that particular framework but it would also show us how long it take for a developer to learn how to use the framework, and how much can be built in a week by using that framework. The three options were

- Web2Py

- Django

- Ruby on Rails

After our frameworks evaluation we came to the following conclusions about them: Ruby on Rails offers useful code generators which can produce functions out of one line of code written by the developer and has a build in testing framework which can be particularly useful for our testing. However, the mystery behind the code generators can cause confusion to developers and thus increase development time.

Web2Py uses a Python-based template language and supports development from a Web Browser. We have been using python for our introduction to programming course in Level 1 of our university studies so it would python is a language we have all used and are comfortable with. Moreover, a developer needs not have their own computer with them to work on the development of the application. Web2Py gives the ability to a developer to work on the code of the application via a web

interface which can be particularly useful in case one of the team members is traveling and is not able to use their own computer to work on application.

Django is an MVT (Model, View, Template) based framework that runs with Python on its back end. This gives us the advantage of having separation of concerns in our application since different components in the framework act independently from each other so a developer can work on one part of the application while another can work on a different component without having to wait for a different component to be completed. Django is an extremely customisable framework since it comes packed with a lot of functionality which can be used out of the box or can be further modified to reflect our goals and objectives. On the other hand it might take time to learn since it has its own way of doing things and a developer needs to adapt to it.

5.2.3 Why Django

We decided to use Django since it gives us great flexibility as to what our end product can be like, it is documented extremely well online and even though learning it can take some time, we consider it to be a well worth investment since we will have to use Django in one of our courses in second semester as well.

5.3 Development in Django

As its being described on djangoproject.org, Django is "The Web framework for perfectionists with deadlines". It is a rapid web development framework that can save you the trouble of writing repetitive boilerplate code. A developer using Django can achieve greatness with minimal coding. Django comes with an object relational mapper which means that we can define our database schema in Python code by defining classes and Django will then produce an sql code that can be injected in our database with minimal effort from the developer in order to create any tables required for the project. It has a dynamic build in administration interface which can be customised according to our needs. This can be used, in our case, as a way for a Course Coordinator or a Tutor to create new topics, add slides and questions to each topic, create questions for the final assessment page and create new users or groups of users with special permissions. This will be discussed further in this report.

5.3.1 The MVT Model

The MVT model used by django is a development mode very similar to the MVC model which is widely used by a number of web application frameworks such as Backbone.js SproutCore and the Cocoa framework used in Mac OS X and iOS applications. Django, since it likes to do things its own way, uses a "modified" version of the MVC model. Its model is called MVT (Model, View, Template). The model in the MVC plays the same role as the model in MVT which is sensible. However, the "View" in the MVT maps to the "Controller" of the MVC and the "Template" of the MVT maps to the MVC's "View". Of course this is slightly complicated so it might cause some confusion to a developer coming from an MVC background. The MVT in general defines the way everything works in Django. Everything in Django can be broken down to 3 components; The Models, the Templates and the Views.

5.3.2 Models

"A model is the single, definitive source of data about your data. It contains the essential fields and behaviours of the data you are storing"(Django website). In other words, any a table in the database can be created by defining a model in the Models.py file. A database field for that table can be defined as an attribute of the corresponding model. The command "manage.py syncdb" will then read the Models.py file create an SQL code and then run it agains the applications database to create any tables defined by the developer. For example if the code in picture below is run it will create a Database table of topics where each topic has a title and a publication date.
class Topic(models.Model):
title = models.CharField(max_length=100)
pub_date = models.DateTimeField('Date Published')

5.3.3 Views

A view takes care of what is sent to a browser when a specific URL is requested. It is responsible for making any requested action as this is defined in the tags of the html page associated with each view. Such actions might be read or write to the database commands or any other actions that are required by a page.

5.3.4 Templates

A template is usually an html file. It is the base of what will be send on to a web browser client. It described how the data should be presented to the user and can contain a number of Django tags and variables. A tag is surrounded by "Django that a special action needs to be taken at that point of the page. This special action can be an "if" statement, a "for" loop etc. A variable is simply telling Django to load a specific value from the Database and display it at this point of the page. Templates are used by Django as the model for a page to be sent to a web client.

5.3.5 Controller

Each Django application has a "urls.py" file which holds definitions that will be matched against requests by web clients. An example url definition in "urls.py" could be
```
"url(r ^'contents', 'views.contents')".
```
In this scenario if our server is hosted on 127.0.0.1 port 8000 and a client requests "127.0.0.1:8000/contents" Django will match the request with the definition in "urls.py" and call the appropriate view which is the second argument of the url() function. In this case it will be the "contents" view.

## 5.4   3-Tier Architecture

For our implementation we used a 3-Tier Architecture.

Figure 5.1: 3-Tier Architecture Diagram
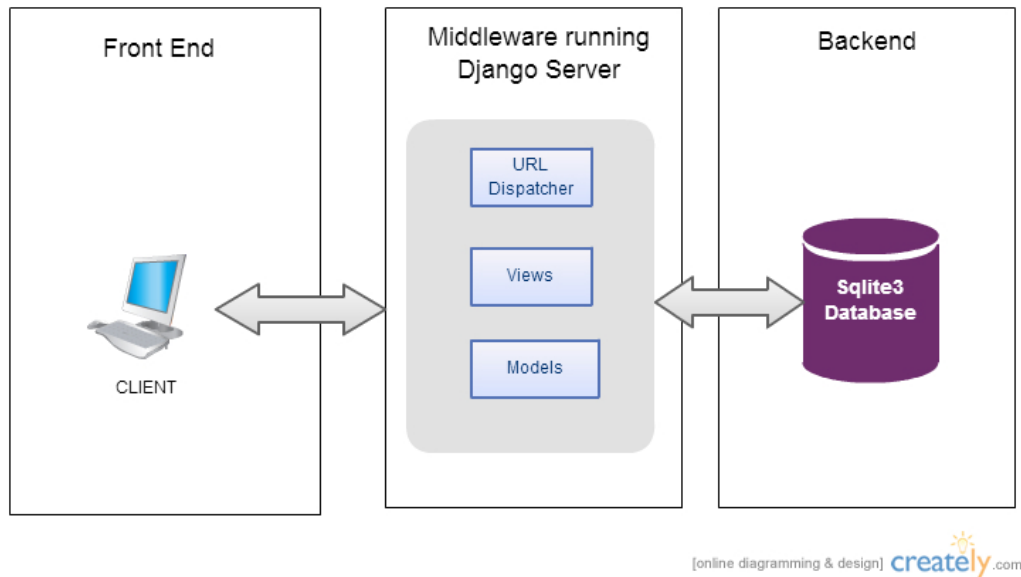


### 5.4.1 Front End

The application can be accessed by any user with a web browser such as Internet Explorer, Google Chrome, Mozzila Firefox, Safari and so on.

### 5.4.2 Middleware

This is our Django server. Responsible for matching requests from Clients using the URL dispatcher which in return calls the appropriate View which then fetches any required context from the Models.py and our Backend Services.

### 5.4.3 Back End

This is our Sqlite3 Database. Other database options included MySQL, Oracle or PostgreSQL. However we opted in for Sqlite3 since it offers all the functionality we need and Django provides more support for it as it is its default option.

## 5.5 Back End - Data Models

Our database consists of 4 basic data models. Namely, Topic, Question, FinalTestQuestion and Slide. According to our requirements, a Topic can have 0 or more questions and slides. Thus, the Question and Slide objects both have a foreign key that points to the Topic they belong to. More information on the definitions of our Data Models can be seen in the tables below:

### 5.5.1   Data Model:Topic

| Field Name | Type | Notes |
|---|---|---|
| title | CharField | 1000 characters max |
| pub_date | DateTime | |

### 5.5.2   Data Model:Question

| Field Name | Type | Notes |
|---|---|---|
| qTopic | ForeignKey | Points to a Topic object |
| text | CharField | 1500 characters max |
| answer | CharField | 100 characters max |

### 5.5.3   Data Model:FinalTestQuestion

| Field Name | Type | Notes |
|---|---|---|
| text | CharField | 2000 Characters max length |
| answer | CharField | 100 characters max |

### 5.5.4   Data Model:Slide

| Field Name | Type | Notes |
|---|---|---|
| sTopic | ForeignKey | Points to a Topic object |
| image | ImageField | |

## 5.6   Managing the Front End

The way Django is generally managing the Front End of an application is quite straight forward. It simply takes a predefined template (html file), adds any required content from the models (database objects) and sends the file over to a web browser. However creating simple html files and feeding them the data from our models is not enough to produce a high quality web application. Producing a rich interactive user interface was extremely important in our case since the application would be used by students to assist their studying. Students can easily get tired of studying and abandon their task so we had to make the interface as attractive as possible, in the little time we had, so that a user can stay motivated and enjoy using it the application.The fact that we were not experienced in web application was not helping us find a solution without investing some time on researching. In our quest to find a way to manage the front end design we looked at a number of solutions. Some of them were the "Backbone.js", "Tastypie" and "Pyjamas". At a first glance each of these frameworks seemed promising. However, trying to use them to produce high quality front end design became more of a time wasting challenge rather that a time worthy investment.

### 5.6.1 Possible frameworks for Front End management

**Backbone**

As its website states, "Backbone.js gives structure to web applications by providing models with key-value binding and custom events, collections with a rich API of enumerable functions, views with declarative event handling, and connects it all to your existing API over a RESTful JSON interface". Fair enough, now how can we proceed and use this Framework with our Django application? A number of github repositories were available online, all with example applications using Backbone. However, since all of them were out of date and description for Backbone and Django integration was vague at best, we soon abandoned the idea of using Backbone.

**Pyjamas**

Pyjamas is a framework which takes code written in Python and translates that into javascript and jquery code without the need of having a developer experienced in using javascript and jquery. We managed to get Pyjamas set up successfullly, but the result was not exactly what we expected. The design looked poor and we had to invest even more time in advancing our skills on coding a GUI with python

**Compination of traditional Django friendly frameworks**

Luckily for us, one of our team members who is really confident on his web design skills convinced us that surely using traditional web design methods, such as writing the code for the templates directly in javascript and using a simple collection of tools such as the Twitter Bootstrap might be worth looking into. He suggested that he can try work out a first draft of the Topics page so that we can see what he can produce with these tools and judge for ourselves. Creating the page actually took him less than one hour and that was a wakeup call for us since the sample page he created was both looking aesthetically nice and the technologies used were working with the Django seamlessly with minimal effort. We started by creating a base.html template. Django gives a developer the option to create templates that inherit from other templates. So any code that needs to be repeated in every page needs not be defined more than once. Our base page with no additional content can be seen on figure 1.2 . This includes the basic layout that should be displayed on ALL templates

Figure 5.2: Base.html Template

of our application. So to make the connection between our base template and all other template we had to define an empty Django tag called content block in our base.html template (see figure 1.3). Then on every page that needs to inherit from base.html we can define the content block so that it

Figure 5.3: The Base.html template with no additional content

```
                    <li><a href="/contents">Table of Topics</a> <span class="div
                    <li><a href="/final">Final Assessment</a> <span class="divid
                    <li><a href="/admin">Administration</a> <span class="divider
                </ul>
            </div>
            {% endblock %}

            {% block content %}

            {% endblock %}

            <div class="row-fluid">
                <hr>
            </div>
            <div class="span12">
                Concept by  <a href="http://thomaspark.me">Dr. Fiona Dowell</a>.
                Made by Team V under revision of Dr. Jeremy Singer <br/>
                Copyright © 2013 University of Glasgow, School of Computing Sci
```

includes any additional data that each specific template may require. An example of how a block is defined in a base template is shown in figure baseblock.jpg and then the way the base template is inherited and the content block is used can be seen on figure "extendsbase".

Figure 5.4: Contents template inherits from base.html.

```
{% extends 'DrugNinja/base.html' %}

{% block content %}
    <div class="span0"></div>
    <div class="span11">
    <h1> Contents </h1>
        {% if topics %}
            {% for topic in topics %}
                <a href="/topic/{{forloop.counter0}}"><h3><li>{{topic}}</li></h3></a>
            {% endfor %}
        {% else %}
            <h3> No topics in database </h3>
        {% endif %}
    </div>
{% endblock %}
```

## 5.7   Middleware: Linking the back with the front

Django makes the connection between the backend data models and the front end templates by interpreting a request from a web client, finding the required template for that request and rendering the context on the page by replacing all Django tags (text surrounded by '%' and '%') with data returned by the responsible view for that request. For instance, figure **contentsView** shows how the contents View is defined. The tag topics is declared as a list of Topic objects and then by adding the key topics into the contents view context we are telling the view to return the list topics every time the tag topics is found on the template contents.html. Then in template contents.html we make use of a for loop to iterate through the topics list, and print out the topic name with a link to its unique url. After the template has been rendered it is sent on to the web browser where it is presented to the urer who requested the page.

5.8 Message Passing

5.8.1 Database Request Format

Database Requests are handled by Django. A template may require information about a specific data model. The view that makes use of that specific model makes SQL request to the database which in turn returns any matching results from our SQLite database.

Figure 5.5: Database Request figure
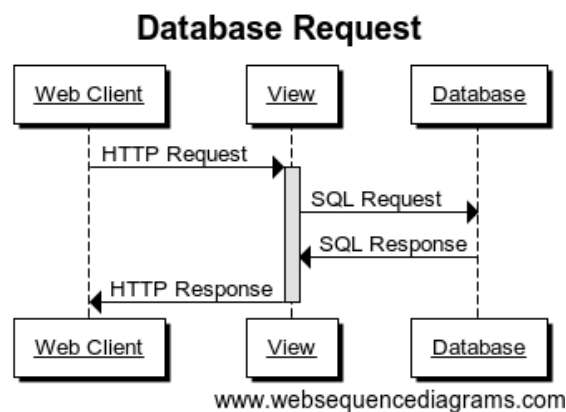
5.8.2 Answer Validation Request

The validation of answers is done by an Ajax request to the validateAnswer view or to the validateFinalAnswer view when the answer to be validated belongs to a final assessment type question. The validation is done by passing the question id and the provided answer as URL parameters. The view then makes a request to the database requesting the answer of the required question. If the answer provided by the user is the same as the one stored in the database the view will respond to the web client with an appropriate Http Response as on figure 1.6.

Figure 5.6: Answer Validation Request figure.
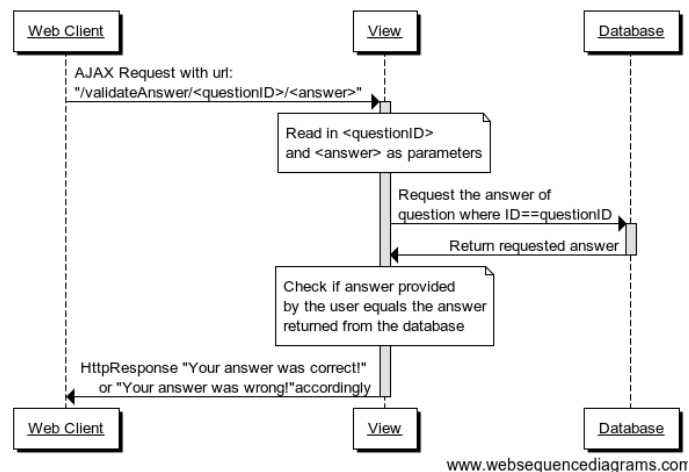
5.8.3 Topic Related Data Request

Topic requests make use of the Topic view. The view uses the topic.html template and renders it so that it gains access to a list of Topic objects. When, for example, a user needs to navigate to the appropriate page for the first topic in the Topic objects list, they should navigate to /topic/0 where 0 is the position of the first Topic objects in the topics list. When the slides of a specific Topic need to be displayed, a for loop is used to iterate through the list of slides and return only those that point to the Topic that was originally requested by the URL.

Figure 5.7: Topic Related Request
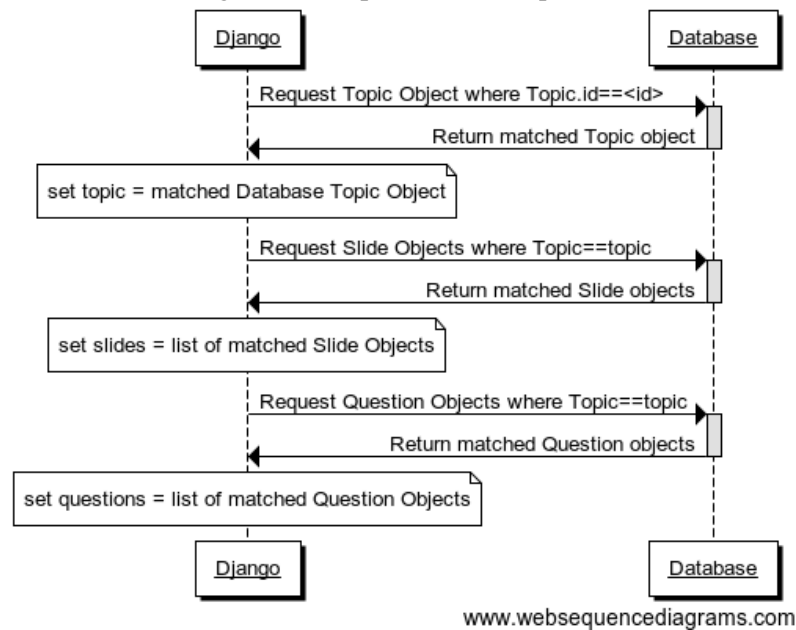
5.9 End Product

5.9.1 Desired Functionality

The Desired Functionality section aims to describe how the desired functionality mentioned in Requirements and Design sections was realised through the implementation will now be demonstrated in more detail.

Welcome Page

The welcome page is implemented through the index view. It is maybe the simplest template in our project. It is a simple template that inherits from base.html and includes a static text with a brief description of our web application and a button that points to the first topic page.

Figure 5.8: Final Welcome Page

Topic Page

The topic page consists of two equal width side bars. The left sidebar is a carousel which displays all the slides related to each topic and the right sidebar consists of a for loop that displays all the questions related to each topic along with a text box for a user to enter their answer and a submit button. Once a submit button is pressed, an ajax call makes a request to the server which responds with a correct answer or false answer accordingly. The response from the database updated the string on the submit button so the text on the button changes from Submit to Correct answer or False answer.

Figure 5.9: Topic Page

Contents Page

The contents page is implemented so that a user can navigate directly to a specific topic page without having to go through all other topics via the Next button on each topic page. A for loop is used on contents.html template which is used so that the contents view will replace the contents of the for loop with a list of all Topic Names currently stored in the database with links that point to that specific Topics page.

Figure 5.10: Contents Page

Final Assessment Page

The final page is one of the core components of our application. The requirements were clear. A user should be faced with 10 random questions from the FinalAssessmentQuestion database model and he or she should only be allowed to have one attempt to answer the question. This is to make sure that a user pays the required amount of thinking before clicking the Submit Answer button. Our client initially provided us with a number of sample lecture slides so that we could get an idea of her lectures contents. We noticed that a picture she uses often to make her lecture slides more interesting to students is a picture of donkey from the Shrek movies. So we thought that since the users are probably already familiar with the Shrek donkey so we added it on a side bar in an attempt to make the Final Assessment Page more interesting and appealing to the users. We also added a health bar and a Score label. The score label has a maximum value of 100% and it defaults to 0% while the health bar has a maximum value of 100% but defaults to the maximum. Each time a user enters a value to a questions text box and clicks the Submit Answer button an Ajax request is sent to the Django server which responds with Correct Answer or False Answer. If the answer was correct, the Score label will be update with the current score + 10 and the health bar will not be modified. However, if the answer was wrong the value of the health bar will go down by 10% and the score

will remain the same. In addition, if the value of the health bar falls below 50%, the image of the donkey will change from the default happy donkey to a sad donkey and then if it falls below 1% an alert will be displayed to the user with a You have killed your animal notification and the picture of the donkey will be changed to a dead donkey. A colour coded response is also displayed to the user as the question box turns red if the users answer was wrong, or green if the users question was correct.

Figure 5.11: Final Assessment Page

Administration Page

The administrator page is dynamically created by Django. We only needed to define which models needed to be modifiable from the admin page and the way we want them to be displayed. Two groups of modifiable data models were defined. Topics and FinalAssessmentQuestions. The reason behind this was that questions and slides have a foreign key that points to a Topic so by selecting a Topic in the administration page a user can see and modify all slides and questions that are related to that Topic. By navigating to the FinalAssessmentQuestions a user can add, delete or modify a Final Assessment Question. By default, Django also provides a user management interface by which a user with administrative rights can manage users by adding, removing or modifying permissions for other users. The administration home page can be seen in Figure 1.4 and the Modify Topic interface in Figure 1.5

Figure 5.12: Administration Home Page

Figure 5.13: Modify Topic Interface

5.9.2 Interaction Diagrams

Topic Page

Interaction diagram for communications and message passing between Topic Page and the server

Figure 5.14: Topic Page Interactions
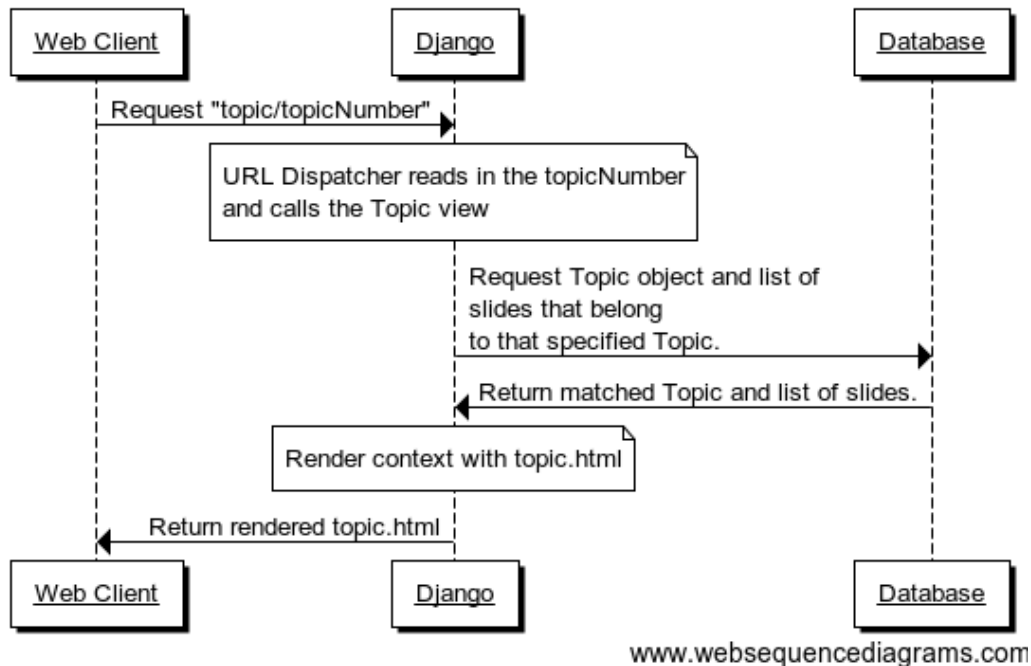
Contents Page

Interaction diagram for communications and message passing between the Contents Page and the server

Figure 5.15: Contents Interaction Diagram
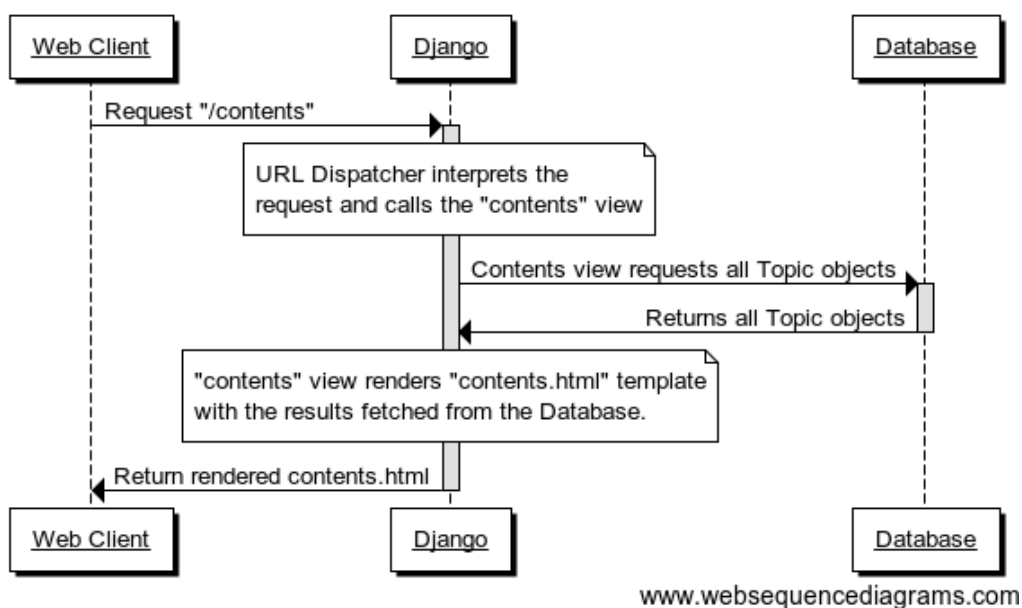
Final Assessment Page

Interaction diagram for communications and message passing between the Final Assessment Page and the server

Figure 5.16: Final Assessment Interaction Diagram
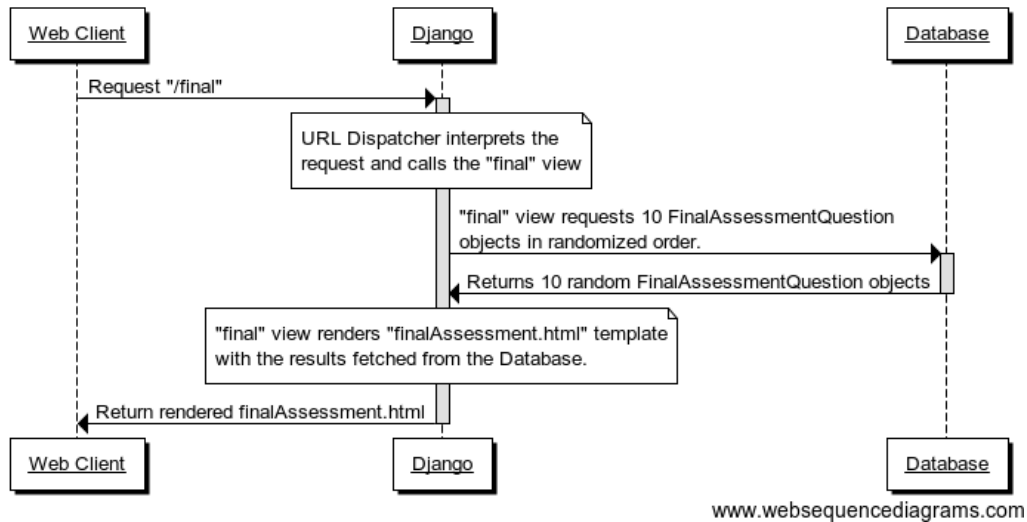
5.10 Challenges and Solutions

5.10.1 Referencing topics by topic.id

When we first started working with topics we were using the topic ids provided by Django to reference and call topics from a template's perspective. However we soon realised that Django was using the IDs provided by SQL and not some IDs that were generated by Django. The result of this error on our behalf was that when a Topic was removed from the database, the ID numbers for topics were not automatically regenerated so clicking the next button (this calls topic with topic ID equal to the current topic's ID+1) on a Topic page would call a Topic ID that might not exist in the database. Consequently, that would cause the application to run on an Index error In order to bypass this issue, we looked at alternative ways of indexing the list of Topics and returning available Topic numbers. Instead of using Topic IDs we started using Topic List positions. More specifically, we are now creating a list of Topics in Topic view by requesting ALL the Topics currently held in the Database and then iterate over that one. For example, when /topic/0 is called, instead of requesting Topic with TopicID==0 from the database, we are requesting the Topic in position [0] of our runtime created Topic list. This way, the numbers used to request topics are always up to date with exactly what's currently held in the Database. We have also added checks to see whether the current topic is the first or last topic in the Topics list. In which case the Next or Previous buttons are hidden accordingly.

5.10.2 URL Dispatcher for Validating Questions

As mentioned in previous sections, the way we validate answers is by passing them as parameters with an Ajax Request. The URL dispatcher then matches the url the web client requests and matches that with the "validate" url definition in urls.py. The url definition for "validate_answer" is: `"url(r'^validate_answer/(?P<question>\w+)/(?P<answer>\w+)', 'views.validate')"`. By definition `(?P<answer>\w+)` accepts any string as answer. However when a user provides an answer such as 2.5, the URL dispatcher accepts 2 and disgards anything after the decimal point. This was causing answer validations to fail even when the answer provided by the user was correct. The Django documentation on URLs was not particularly helpful so through a process of trial and error we arrived at a valid solution that can match both strings (this can be a non decimal number as well) and decimal numbers. In addition to our `url(r'^validate_answer/(?P<question>\w+)/(?P<answer>\w+)', 'views.validate')` regular expression we added `url(r'^validate_answer/(?P<question>\\w+)/(?P<answer>\\d+\.\\d+)', 'DrugNinja.views.validate')` right before the first expression. In essense, when a validate answer request arrives, the URL dispatcher will try and match it to the new reqular expression which only matches answers with a decimal point. If the answer does not match, Django will try to match it to the second URL definition which accepts strings or non decimal integers and respond accordingly.

5.10.3 Static Files

Django has some strict safety rules when it comes to static files. When we started our implementation, having only basic web design knowledge, we though that defining a static files directory would be enough for Django to be able to write and read files. This seemed to be the case since when we were uploading images to the system through the administration interface, the system seemed to be reading the files just fine and saving them in the correct directory. However, when a web browser made a request for a static file Django was returning 404 errors referring to a local file that we were sure that exists. This made absolutely no sense to us so resolving this error was particularly time consuming. After a lot of research in Django's Documentation we found out that Django does not allow a web browser to access the static files directory directly. A new directory should be defined as public in Django settings file. This way, a browser would make calls against the public directory and even though that directory is empty, Django will copy files to it and make them accessible to the web browser without making the original files valnurable to malicious activity by a web client user.

5.11 Known Issues

There are currently two known issues in our application which are planned to be resolved after this release.

- Font can be different from web browser to web browser.

- The slides carousel behaviour is slightly different when accessed through Google Chrome. This is not a serious issue and does not affect the user experience in any major way.

Chapter 6

Evaluation and Testing

This section discusses the methods used for testing and evaluating our application. Testing is important in order to ensure that all requirements are met and to identify and rectify any errors. Using various methods of evaluating allows us to test the usability and functionality of our final application.

6.1 Testing

6.1.1 Test Plan and Strategy

Firstly, we created a test log in which we added the feature tested, the date it was tested and the result of the test. Each time we tested a new feature it was added to the test log in order to keep track of what had been tested and whether or not it had been successful.

Initially, for testing our application, we tested each new feature as they were implemented. After running the application to see if the new feature worked as expected, we then tested it with the previous feature already implemented to ensure that they didn't conflict or cause the application to crash.

If an error was detected or the feature caused the application to crash, this was documented in the test log and was raised as an issue on the team's GitHub. It was then assigned to a developer so that the issue could be diagnosed and fixed. Doing this allowed us to keep track of any and all issues and errors that had occurred throughout the implementation process.

However, we found that this was very time consuming, as opposed to having a pre-defined set of test scripts. We then created a set of test scripts for each page within the application. After implementing several features, we would run the test-script in order to see if they worked as expected and produced the correct output.
This will give us a generic view of our application's status. This would dynamically generate a pie chart to give a visual presentation of the number of 'Passed', 'Failed' and 'Blocked' tasks.

- Passed Result - a test task that runs successfully and produces the expected result.

- Failed Result - a test task that does not produce the expected result.

- Blocked Result - A test task that cannot be run, due to an issue thats waiting to be resolved. For example, if the answer validation function is not yet implemented, then a test that includes answer validation cannot be run.

6.1.2 Issues

Some of the main issues we encountered are listed below.

Table 6.1 - Main issues occurred throughout development

Issue	Current Status
ImageField throws 404 error when trying to access saved Image	Resolved
No error for 0 slides in a topic	Resolved
Application crashes when clicking next topic button	Resolved
Application font appears different on different web-browsers	Unresolved
System returns "Wrong answer" for correct answer.	Resolved
Questions in final assessment are not displayed in random order	Resolved

The table shows some issues which we encountered throughout the development of our application.
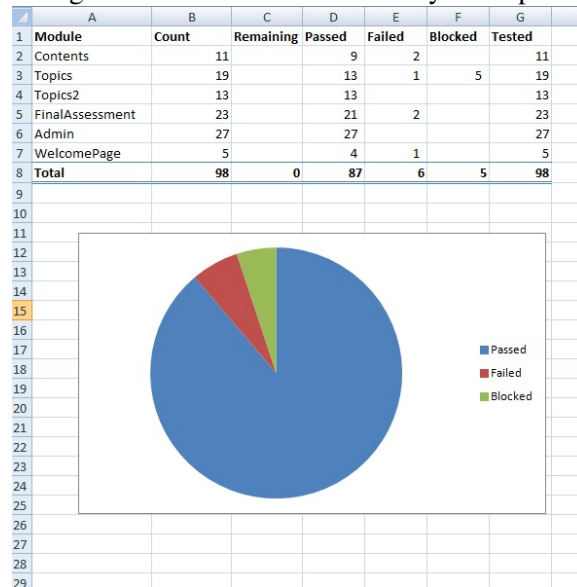
6.1.3 Testing Results Sample

An example test script can be seen in Figure 6.1 This includes a set of tasks with their expected behaviour and the results achieved in each test cycle.

Figure 6.1: Test Script Example

Figure 6.2 below shows the test results summary for all the test scripts produced.

Figure 6.2: Test Results Summary Example

	A	B	C	D	E	F	G	H
1	**Module**	**Count**	**Remaining**	**Passed**	**Failed**	**Blocked**	**Tested**	
2	Contents	11		9	2		11	
3	Topics	19		13	1	5	19	
4	Topics2	13		13			13	
5	FinalAssessment	23		21	2		23	
6	Admin	27		27			27	
7	WelcomePage	5		4	1		5	
8	**Total**	**98**	**0**	**87**	**6**	**5**	**98**	
9								

6.2   User Evaluation

In order to test the usability and functionality of our application, we identified the features of the application which would provide us with the most accurate results:

- Content management: admin users should be able to easily upload, edit and delete content.

- Browse through topics successfully.

- Answer all questions available.

- Complete the final assessment and see their final score.

We then created a test plan to evaluate the usability and aesthetic appeal of the application. To test both the administration features and the learning functionality for the students, we created a set of tasks based on the features above. We used the 'Think-Aloud' method during the evaluation and provided participants with a short questionnaire afterwards.

6.2.1   Functional Requirement Fulfilment

Before carrying out our evaluation with a group of participants, our team carried out an evaluation on ourselves in order to test the application for issues and to see how many of the requirements were satisfied. From this evaluation, we found that our application satisfies each of the client's 15 specified functional requirements. The admin user is able to upload new content, edit and delete content. The student user can view all topics available and see all related slides and questions for each topic. The student user can also complete the final assessment and see their final score, indicating how well they answered.

The application's implementation has been a very iterative process, and meeting with our client repeatedly over the course of the development has meant that we have been steadily improving the application and refactoring our code.

Developing beyond the 'must have' requirements, we have developed a game-like final assessment page and also refined our application design to increase usability and aesthetic appeal for the end user.

6.2.2 Users

Firstly, we wanted to do an evaluation with our client: Dr Fiona Dowell. This would allow us to see how satisfied our client was with the application we had created and allow us to receive any feedback. Following that, we wanted to get a mixture of Computing Science/Software Engineering students and other students to evaluate our application. As the Computing Science/Software Engineering students had done coursework with Django recently, we thought that having participants who are unfamiliar with Django would provide us with more reliable results for our evaluation.

6.2.3 Tasks

We asked the participants to carry out a number of tasks in order to evaluate our application. We have tasks for both an admin user, to test the administration features and content management, and for a student user, to test the functionality and usability of the learning features.

Admin User Tasks

Table 6.2 - Admin User Tasks

	Task
1	Login as administrator.
2	Add a new user.
3	Set the users permissions to allow them to update content - topics, slides, questions.
4	Add a new Topic.
5	Add a new slide to that topic.
6	Go to Topic 2 and remove slide 3.
7	Add a new question to Topic 1.
8	Edit question 2 within Topic 2.

This table shows the tasks to be carried out by partipants in order to evaluate the admin user functionality.

Student User Tasks

Table 6.3 - Student User Tasks

	Task
1	Go to Topic 1 and browse through the slides.
2	Answer 2 questions within Topic 1.
3	Go to Topic 2.
4	Go to the final Assessment and complete the test.

This table shows the tasks to be carried out by participants in order to evaluate the student user functionality.

User Consent

In order to avoid any ethical issues, all participants were issued a consent form, which informed them what the testing would involve and what exactly it was for. It also stated that the participant information would be kept private and that they are able to leave the testing at any time if they did not wish to continue. The consent forms had to be signed and returned before any testing was allowed to commence.

Think-Aloud

All evaluations were performed using the Think Aloud method which involves observing the participants performing the tasks and having them describe out loud what they are doing and why. This is a useful evaluation method as it allows us to see first-hand the process that the participants go through to complete the tasks and highlights any difficulties or errors they may encounter.

Questionnaire

After the evaluation, each participant was asked to complete a quick questionnaire about the application and the tasks they had completed, and provide any additional feedback. The first 4 questions are related to the usability of the content admin tasks, and questions 5 to 8 are about the usability of the student user tasks.

Table 6.4 - Questionnaire

	Questions Key: 1 - very easy; 5 - very difficult
1	Overall, how easy was the admin system to use? 1 2 3 4 5
2	How difficult did you find it to add a new user to the system? 1 2 3 4 5
3	Were you able to change a user's access permissions? If so, how difficult was this? 1 2 3 4 5
4	How easy was it to add new content - topics, slides, questions? 1 2 3 4 5
5	Overall, how easy was the application interface to use? 1 2 3 4 5
6	How easy was it to navigate to different topics? 1 2 3 4 5
7	Were you able to browse through slides and questions? If so, how difficult was this? 1 2 3 4 5
8	Were you able to see your final score on the Final Assessment? If so, how clear was this? 1 2 3 4 5
	Additional Comments

The table above shows the questionnaire each participant completed in order to evaluate the difficulty of the tasks, on a scale of 1 to 5.

6.2.4   Results

Overall, the data collected from all the evaluations was positive. However a few suggestions were made for ways to make some tasks more user-friendly and easier to use.
While each participant was completing the set of tasks, the time it took for each was recorded. This is important as the amount of time taken can indicate how easy or difficult that task was to complete.

Admin User Task Results

The table below shows the average times for the admin user tasks.

Table 6.5 - Admin User Results

Task Number	Average Time Taken (seconds)
1	20
2	24
3	55
4	44
5	23
6	33
7	21
8	22
	n=11

This table shows the average times the 11 participants took for each of the admin user tasks.

From Table 6.5, it its clear that most participants found task 3, setting the user permissions, the longest task to do. Most users were unsure what access different users were supposed to have and found that there were too many options which made it difficult to find the correct permissions to set.
Due to the users and topics being within different sections in the administration page, most participants were unable to locate the 'Topics' section once they had completed the new users tasks, resulting in task 4 taking them longer to complete.
Task 6, deleting a slide, also proved to be slightly time consuming for some users. To delete an item from a topic, the user has to tick the delete check-box for that item and select 'Save'. However, some participants were expecting immediate, visual feedback, once they had selected the delete check-box , to indicate that the item had been deleted and were unaware that to remove it from the topic they had to click the 'Save' button at the bottom of the page.

Student User Task Results

The table below shows the average time taken for each of the tasks carried out for the student user.

Table 6.6 - Student User Results

Task Number	Average Time Taken (seconds)
1	17
2	20
3	7
4	62
	n=11

This table shows the average times the 11 participants took for each of the student user tasks.

As shown in Table 6.6, it is clear that participants had very little problems carrying out the tasks for the student user. The time taken task 4 is quite large due to having to answer all the questions within the Final Assessment. The only problem the participants faced was that most were unaware

that the submit button had to be selected for each question, so after answering all the questions, they had to go back to the top of the assessment and select the 'Submit' button for each question.

Questionnaire Results

After completing both sets of tasks, each participant was asked to complete a short questionnaire, containing questions about the usability of the application. The table below shows the average difficulty results from the questionnaires.

Table 6.7 - Questionnaire Results

Question Number	Average Result
1	2.18
2	1.45
3	2.36
4	1.54
5	1.45
6	1.09
7	1.00
8	1.09
	n=11

The table above shows the average difficulty result from the 11 participants involved, with 1 being very easy and 5 being very difficult.

Table 6.7 shows that, overall, the participants found the application easy to use, particularly the student user tasks. Although the average results does not indicate that participants had any real difficulty with setting the permissions of users, it is clear from the questionnaire that the participants found this the most difficult task to complete.

6.2.5 Feedback

After completing the questionnaire, participants were asked if they had any additional comments or feedback about the application. Overall, the participants were pleased with the application and found it very simple to use. Some improvements were suggested for both the student view of the application and the administration page.

For the application, although the participants enjoyed the gamification of the Final Assessment with the health bar of the animal, it was suggested by a number of the participants to have a 'Submit All' button to avoid having to submit all answers individually.

Another suggestion was to have the sample questions within each topic provide some sort of visual feedback to indicate more clearly to the users if they have the answered the question correctly, for example, change the 'Submit' button to green if correct or red if incorrect.

Within the topics, there are 'Next' and 'Previous' links in order to change quickly between the topics without having to repeatedly go to the Table of topics page to select the next topic. However, most particpants did not notice these links, so it was suggested that the size of the text is increased or changed to buttons to make them stand-out and more obvious to the user.

For the administration of the application and the users, some participants found that when deleting items from the application, it wasn't obvious how to do this and once they figured it out, they were unsure if the item had actually been deleted. So, it was suggested that there is a delete button for each item, instead of a checkbox, and when clicked the user to receive some immediate, visual feedback to show that the item has been deleted.

Most of the participants had some difficulty when it came to setting the user permissions as they found that it was not clear what permissions each type of user was supposed to have. Some of the participants suggested that users are added to 'User Groups' which have different levels of access and permissions, and when adding a new user, you can simply select the user group to add them to, avoiding having to set permissions individually for each user account created for the application.

Chapter 7

Conclusion

7.1 Summary

The aim of our project was to create a learning application for veterinary students that will allow them to learn and revise course content for learning about drug calculations in animals. After a full evaluation of our application with our client, Dr Fiona Dowell, and a group of students, we are satisfied that we met all product requirements and will integrate successfully within the veterinary school.

A student user is able to successfully and easily view all the topics available, browse through all of the slides within each topic, answering questions as they go along, and they can complete a final assessment at the end to test how much they have learned.

An admin user can successfully manage all content for the application - including uploading new slides and questions, editing topics and questions and delete any topics, slides or questions.

7.2 Future Work

After evaluating the application with our client, we have concluded that the following improvements could be made:

- Within the Final Assessment page, add a 'Submit All' button to avoid having to submit answers individually. However, as a consequence, this will remove the gamification element of the final assessment page so will need to be discussed further with our client.

- Provide visual feedback when answering questions within topics, to indicate to the user if they answered correctly or incorrectly. For example, change the 'Submit' button colour to green if correct, and red if incorrect. This will be more intuitive than simply displaying 'Correct' or 'Wrong' as plain text.

- Within the administration page, set up several custom user groups with different access permissions. This will allow the administration user to assign a new user to a group, therefore, streamlining the permissions process and eliminating current confusion.

- Categorise all the questions within the Final Assessment page, into easy and difficult questions, having easier questions appearing at the top of the assessment. As a result, the final assessment pages will be more structured and reflect exams more accurately.

- Add support for the inclusion of both audio and video content to create a more dynamic and interactive experience.

- Make our application available for smart phones and tablets. This will involve making dynamic resizing pages and formatting for smaller screens.

- Add a Help section to provide users with 'How To' instructions for each function of the application.

- Generalise the application so that it can handle content from other courses.

7.3 Reflection

We found this project intensely rewarding as our work will help other students during their studies at the University of Glasgow, rather than simply being forgotten about. We enjoyed having an external client to work along side with which allowed us to put into practice elements of our Professional Software Development 3 course. We learned many new technologies, such as Django, Jquery, Ajax and built upon our knowledge of Python from first year. This was a fantastic learning experience because working as a team was not something that most of our group had experienced.

If we were to start this project again, we would probably set ourselves stricter deadlines, with several meetings each week to keep on top of the project progress. We would also place more trust in each other to complete assigned tasks on time.

7.4 Contributions

Ross Adam:

- Organised Presentation
- Contributed to Introduction section of dissertation
- Contributed to Background section of dissertation
- Contributed to Design section of dissertation
- Contributed to Evaluation section of the dissertation
- Managed team as Project Manager

Andrew Gardner:

- Contributed to initial direction of project

- Contributed to requirements gathering of project

- Contributed to design of project

- Contributed to implementation of project

- Contributed to Design section of dissertations

- Liaised with client as Customer Liaison

Nicole Kearns:

- Secretary for the team.

- Communications with the client, Dr Fiona Dowell.

- Contributed to the initial direction of project.

- Contributed to the initial design of the project.

- Contributed to the requirements gathering of the project.

- Contributed to the Introduction section of the dissertation.

- Contributed to the Background section of the dissertation.

- Contributed by writing the Requirements section of the dissertation.

- Contributed by writing the Testing and Evaluation section of the dissertation.

- Contributed to the Conclusion section on the dissertation.

- Contirbuted to the evaluation process of the project.

Mamas Nicolaou:

- Contributed to the initial direction of project.

- Contributed to the initial design of the project.

- Contributed by creating the Django Models for the Project.

- Contributed by creating the Django Views for the Project.

- Contributed by creating the Django URL Dispatched for the Project.

- Contributed by implementing the Django templates.

- Contributed by resolving bugs raised by other team members.

- Contributed by creating the Administration Interface for the Project.

- Contributed by writing the Implementation section of the dissertation.

- Contributed by creating the Final Assessment, Topic, Contents pages.

- Contributed to the Testing and Evaluation section of the dissertation.

Asset Sarsengaliyev:

- Contributed to design of project

- Contributed to design implementation of project

7.5   Appendix A: Evaluation Consent Form

Veterinary Study Application Consent.

Thank you for volunteering to be part of this short user evaluation study. You will be interacting with a revision tool designed for use within The University of Glasgows Veterinary School. It will include revision material for students to study and questions to test their knowledge.

The tasks we will be asking you to complete will cover a variety of different aspects of the program. There will also be a questionnaire at the end of the study to elicit further information from you. Please feel free to provide additional feedback at the end of the questionnaire.

If you agree to take part in this survey please sign below as proof of consent. Also note that you may retire from the study at any time should you not wish to proceed further.

Signed:

Date: