Software Engineering Large Practical - Audience Response System

Cameron Gray - s1230461

December 18, 2014

Project Idea and Reasoning

Introduction

For this project I built an "Audience Response System", similar to the "clicker" system used within the university. The idea came about after seeing clear frustration with the current system that is in in use due to a few main failings - It is slow, the software is confusing to use and students are required to collect and look after a piece of hardware. This would be implemented as an entirely web based application where both the tutor and students interact with the software using a web browser.

Technologies

Backend I decided to use the Django web framework with Python 3. I chose this for several reasons - I am familiar with the Python programming language (which Django uses), I am already very well versed in Model, View, Controller web development and the framework is very well established and supported.

CSS I have used Twitter's Bootstrap framework to provide most of the CSS styling for this application - This allowed me to not only build a good looking application, it saved the time that I would otherwise have had to spend building a custom CSS template. Bootstrap is also extremely well documented which will aid maintenance in the future since there is already very comprehensive documentation for the frontend CSS.

Frontend Javascript I have used the jQuery Javascript framework as it is an extremely well established framework that provides several time saving functions that would otherwise need implemented manually such as DOM manipulation and AJAX.

Database Throughout development I have used an SQLite database, this was used due to its simplistic nature and because it does not require that a database server is running on every machine that I am developing on. In production use SQLite would not be suitable as it is not designed to support several concurrent users, in this case the application would need to use a more powerful database server such as PostgreSQL. Thankfully due to the high level of abstraction from the database that Django provide, changing to a new database server is as simple as changing a single configuration file to reflect the settings for the new database - The underlying application would not require any changes.

Improvements over current "Clicker" system

This new web based audience response system has several clear advantages over the hardware based "Clicker" system that the University currently uses.

No requirement on hardware From the University's perspective, they would no longer need to loan out "Clickers" to students and deal with handling returns and losses. There would also be no requirement for the specialised receiving hardware that is installed in lecture theatres. This poses clear cost savings in terms of purchasing the system along with the cost of ongoing maintenance. This would also benefit other users who may not be able to afford the specialised hardware required.

System can be used over the Internet Due to the current system's reliance on radio hardware, students can only respond from within the same room as the receiving hardware is located. Having a web based system means that students can respond to questions from anywhere with an internet connection which opens the system to be used for live web tutorials and online conferences.

Disadvantage over the current "Clicker" system

Requires students to own hardware Students are required to be able to provide their own hardware to respond to questions. In most cases this would not be an issue as the vast majority of students own at least one device that offers a web browser.

Terminology

Before proceeding I feel that it is important to clarify some of the system specific terminology that is used in both this report and in the software itself in the context of the system being used live in a lecture environment.

Tutor A tutor is a person who would be inputting questions, sending them out to students and looking at the responses. This person would most likely be the lecturer. Every tutor has exactly one django user for authentication.

Course A course is a specific subject that is being taught, this would generally be directly related to the courses that the university teaches. A course for example could be "Software Engineering Large Practical" or "Operating Systems".

Session A session is a group of questions, these would usually be grouped together in the same way that a lecture would - In most cases you would have one session for each lecture. An example of a session could be "Git - Source Code Control" or "Memory Management".

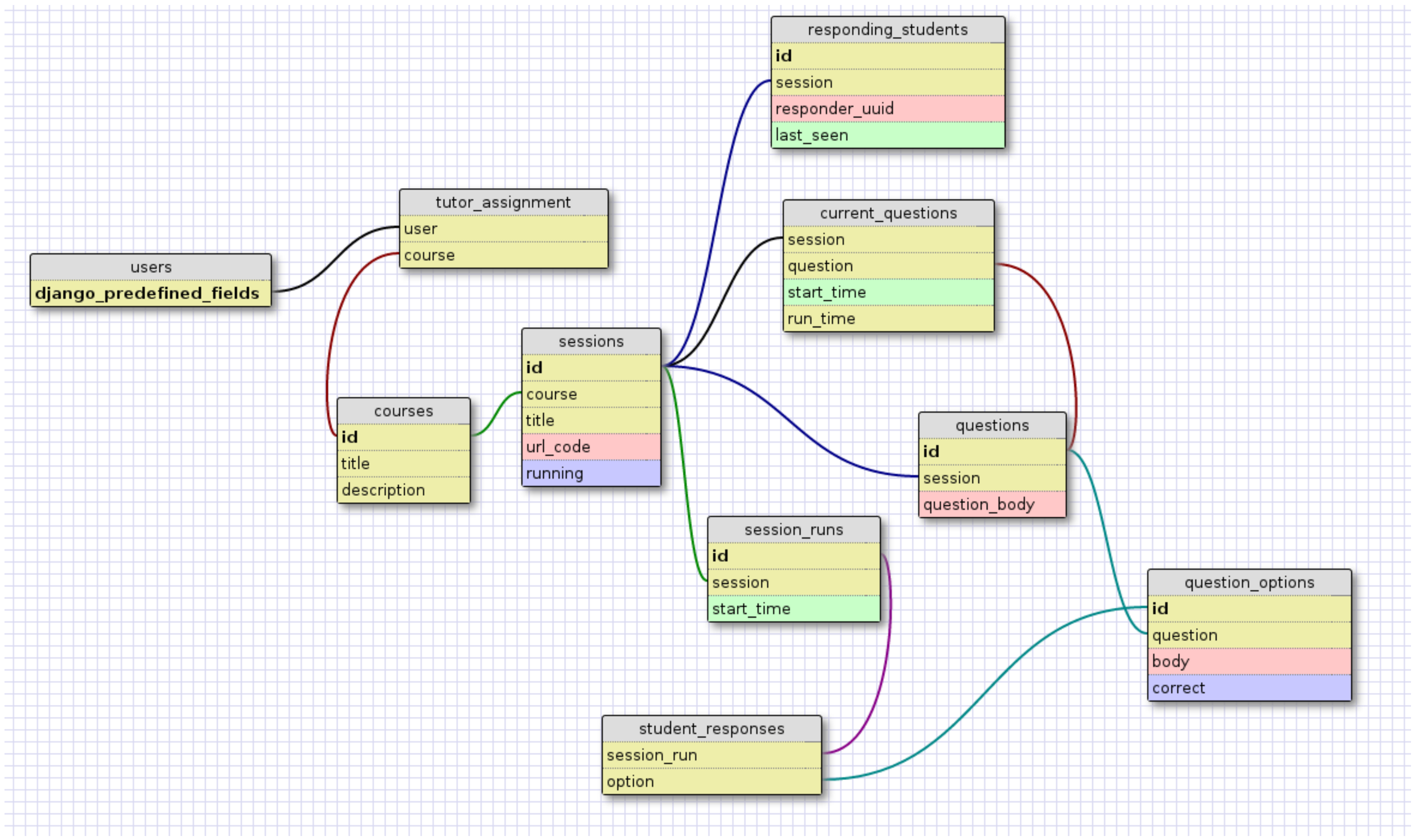
Session Run A session run can be thought of as an instance of a session. Each individual lecture would relate to a single session run. The reason that they are differentiated is because the same lecture may be repeated at different times and the results from the questions should be kept separate between both of them. A session run has both a time at which it was started and the single session that it is an instance of.

Question Option A question option would relate to a single possible answer to a multiple choice question. It contains a "body" and is related to a single question. A question option can either be correct or incorrect. As an example, for the question "What colour is the sky?" the question options could be: Red (Incorrect), Green (Incorrect), Blue (Correct).

Implementation Details

Database Design

In order to improve understandability of the database, I created a diagram to represent the database using WWW SQL Designer[1]. This tool allows the diagram to be exported and saved as an XML file meaning that the diagram can even be stored in version control (as I have done in "database.wwwsqldesigner"). The finalised diagram for the database that I used is as follows:



Using this diagram has proven extremely useful as it not only makes development easier, it aids maintainability down the line as developers can easily visualise exactly how the data is structured instead of manually having to look at the models or database itself.

Code Structure

Backend

Due to the use of the Django framework, the structure of code is very rigidly defined which means that it would be familiar to any Django developer who is required to work on the project. The code is divided up into three different Django "apps". The first powers the tutor area, the second powers the student area and the third app called "main" contains components that are shared by both the tutor and student areas.

[1]https://code.google.com/p/wwwsqldesigner/

Models This ”main” app contains all of the models, the reasoning behind this is that in this system, all of the models are shared very closely between the two apps. If I were storing data that would be used only in the student area or only in the tutor area, then the models would be stored in the respective app rather than in ”main”. The individual models themselves are kept very simple and mostly just define the data that is to be stored in them as well as default values. Every model also contains a ”__str__” method which is used to give each model a human readable string. This is not used in the running of the application but makes the data a lot easier to understand when manually querying the models from the django shell or when using Django's premade admin area.

Views The views.py files in both the tutor and student apps holds most of the backed code. Django confusingly calls the file views.py but in the sense of MVC this file acts as a ”controller” and the Django ”templates” act like a view. The main job of the code located in these view files is to take data received in the request and then either use this to insert data into the database, modify existing data in the database or to retrieve data from the database. The result is then rendered into a template that is displayed to the user.

Templates The templates are stored both in the root directory (for base templates that area later extended) and in the individual apps themselves (for app specific templates). Inheritance is used throughout the templates to ensure that code is not duplicated between pages, this means that if changes are to be made to a common area (e.g. the navigation bar), only one file needs to be changed. This also prevents inconsistencies between pages.

Project Settings The flash_response directory is a Django default directory which holds project level files including the main settings file and the base URL routes file.

Decorators The decorators.py file inside the main app is where I define Python decorator functions, at the moment there is a single decorator called ”tutor_course_is_selected” - When a function is decorated with this, the function can only execute if the tutor has selected a course from the course selection menu. If a course is not selected, the page will instead redirect to the tutor index page where a message will instruct the user to select a course.

Context Processors The context_processors.py file in the tutor app contains a single context processor called ”course_assignments.
This is used to ensure that the list of courses that the tutor is assigned to teach is always available in all templates in the tutor area since this list is used to create the common navigation bar. This saves a lot of code reuse as otherwise, every single view that renders a template would have to get the courses and pass it into the template renderer.

Frontend CSS

This project uses the Twitter Bootstrap CSS framework therefore there is only a small amount of hand-written CSS. All of the CSS is stored in the /assets/css/ directory. The bootstrap files are all part of the Bootstrap framework. The style.css file was written by myself in order to add some extra styling and to override some of Bootstrap's default styles.

Frontend Javascript

All of my frontend Javascript is located in the /assets/js/ directory. The bootstrap and jChart files are both libraries that I did not write but are used inside the application itself. The student.js file provides the Javascript functionality for the student area and the tutor.js file provides the functionality for the tutor area.

This Javascript code has been written in an object oriented manner. This is mostly to ensure that the individual parts of the Javascript code is kept separate from one another and so that the different components are clearly separated.

The frontend Javascript code in this application provides two main functions at the frontend. I extensively use AJAX in order to asynchronously transfer data between the backend server and the frontend software. An example of this is the AJAX request that is made to the backend when the user picks an answer to a question. The Javascript is also used to provide a lot of the user interface; it is used for showing/hiding parts of the page as well as being used to populate parts of the page with dynamic data received from the backend.

I have relied heavily on the jQuery Javascript Library throughout this application as it provides extremely useful time-saving methods for AJAX and DOM manipulation. It is also an extremely mature library that is used on the vast majority of modern websites and web applications.

Testing

For this project I chose to implement functional tests, due to how well they tie into a web application. I have implemented them using Django's own testing framework and used their Test Client which allows you to send simulated POST and GET requests to the server and make assertions on the server's response. These tests are located in /main/tests/. For these tests I test both basic functionality such as adding a question and then checking it exists in the database as well as much more advanced functionality such as attempting to edit sessions in a course that the tutor is not assigned to (this should be blocked) and testing a full simulation of a question where 1000 random responses are sent to a question and the results recorded in the database are checked to ensure they match up with what was sent. To run the tests you need to run:

```
$ python manage.py test -v 2
```

Interesting Points about the Implementation

Keeping everything in time

Because the student area poll the backend every 2 seconds to check if a question is available, it becomes tricky to ensure that all students are presented with the question at practically the same time. I was able to solve this problem by implementing the 5 second countdown that is seen between clicking to start a question and the question actually being seen by students.

During this countdown, the student clients will poll the backend and be presented with both the question as well as the number of milliseconds until they are to present the question to the user. Once this period of time has elapsed, the client will display the question to the user. This proved to be a very successful and simple way to keep everything in time.

I also made sure to never use the time on the client device, everything is timed to the time set on the server, client devices only deal with time intervals (numbers of milliseconds) - This prevents time discrepancies across devices from causing problems.

Counting users that are currently waiting to respond

Another challenge I discovered was to be able to display the number of users waiting to receive a question in the session console. I was able to solve this issue by having each student device send a UUID (unique identifier) along with every request to check if a question is available. The server then stores this UUID in the database along with the date/time the user was last seen. The application then regards a student as waiting to respond to a question as long as the server has received a request from that user's UUID within the last 5 seconds.

Instructions

Installation

The following instructions detail how to install and run this project on DICE.

Create Virtual Environment Change to the directory that contains the application directory. You will need to create a Python 3.4 virtual environment. To do this, run the following command:

```
$ pyvenv-3.4 ENV
```

Enter Virtual Environment Before running any other parts of this project, you must switch into the virtual environment that you just created. To do this run:

```
$ source ENV/bin/activate
```

To deactivate the virtual environment you can then run:

```
$ deactivate
```

Installing Dependences This project has some dependencies that need to be installed. These are stored in the requirements.txt file and can be installed using pip as follows. Remember, you must be in the virtual environment. Change into the directory containing requirements.txt (The root of the project) and run:

```
$ pip install -r requirements.txt
```

Insert Sample Data In order to insert the data you must run two commands, one to create the database itself from the models and the second to import the sample data which is sufficient for testing the system. Change to the flash_response directory (the one containing manage.py) and run:

```
$ python manage.py syncdb
$ python manage.py loaddata ../sample_data.json
```

Running syncdb will ask you if you want to create a superuser, you can safely say "no" to this as it is not required in this case.

Running Tests The test suite can be executed by changing into the flash_response directory (which contains the main code) and running:

```
$ python manage.py test -v 2
```

Starting the Server In order to start using the software you must start the Django development server. To do this, change into the flash_response directory and execute:

```
$ python manage.py runserver
```

The server will now begin to listen on localhost at port 8000. You can now access the software at http://127.0.0.1:8000/

Usage

This section will give a rough tour of the application to allow you to gain enough understanding in order to be able to use it effectively. Before doing any of these steps, ensure that the server is running as detailed above.

Logging In as a Tutor To login as a tutor go to http://127.0.0.1:8000/. You will be presented with a simple welcome page and a button to take you into the tutor area, click on this button to be presented with a login form. In the sample data there are three tutors named tutor1, tutor2 and tutor3, all with the password 1234. Tutor1 is enrolled to teach Maths, tutor2 is enrolled to teach Computing and tutor3 is enrolled to teach both. Pick one of these tutors and login.

Selecting a Course When you log in you will be asked to select a course to manage, to do this use the drop down menu in the top right of the page to select a course, this will only show the courses that the tutor you have logged in as is enrolled to teach.

Sessions The "Sessions" button at the top of the page will take you to the sessions page. Here you can create new sessions, edit existing ones (such as adding/changing questions) or start a session to send questions to students.

Starting a Session To run a session and therefore ask students questions, pick one from the sessions page and click "Start New Session". This will open the session console. In here you will see a blue box containing the "Response URL" - This is the URL that students would navigate to in order to answer questions. For testing I would recommend opening a few copies of this URL in different tabs/browser windows. Notice how the session console will update to show the number of students currently waiting to respond.

Asking and responding to a question To ask a question to the class, pick one from the list in the "Run a question" section and enter a time in seconds to represent how long the question should be run for. Then press "Start Question." After a 5 second countdown, the question will start and the session console will display a countdown as well as live updating to display the number of responses received. While the question is running, look at the pages you have opened with the response URL (they must be open before you start the question). On these pages you should see the question along with the possible options, on each page, click one of the options to simulate a student answering a question. Once the question finishes, the session console will display a bar chart that represents the results of the question.

Viewing Reports The reports section (accessed via the "Reports" link in the tutor navigation bar) allows you to generate reports for each session run. This will rank questions by how accurately they were answered and by how many people responded to them. On the Reports page, select a session and then select the date and time of the run from the "Session Run" box. Now click "Generate Report" to display the report for that session run.

Future Enhancements

There are many different improvements/enhancements that I would implement in this application given more time. These are detailed in this section.

WebSockets

At the moment the frontend Javascript communicates with the backend server entirely using AJAX requests. This means that when the frontend is waiting for something it has to poll the backend at a regular interval until it receives the response that it is waiting for. This method is used in areas of the application such as when the student area is waiting for the tutor to start a question or when the tutor area is retrieving how many students have responded to a question while the question is running.

The problem with this method is that it creates a lot of unnecessary requests to the server. For a prototype this is not a problem however if we were to scale this application up to hundreds of users it could become a problem. For example, the student area polls to check if a question is available every two seconds. Therefore if we had 100 students in a class, the server would be receiving $(100/2) * 60 = 3000$ server calls a minute! Now imagine if this was deployed across a university where there could be several lectures of 100 students using the system at any one time. This could also be an issue if people are responding on battery operated devices as these additional server calls could potentially impact on battery life.

The solution to this problem would be to implement WebSockets. Therefore instead of the frontend constantly polling the server for data, the frontend opens a socket connection to the server. The frontend can then pass data down this to the server (as it would do with AJAX) but it has the advantage that the server can pass up data to the frontend. Therefore instead of students' devices constantly polling the server to see if there is a question available, the server will instead push the question out to the students.

Due to time restrictions this was not implemented in this version of the software however if the software were ever to be released or used in a live environment, this functionality would be implemented as a top priority.

Change question while in progress

Ideally I would like to make it possible for the tutor to make changes while the question is in progress, this would mean that the tutor could stop the question early or add more time if students are taking longer than expected to complete the questions.

API

I would like to create an API for the application, mostly for the student area. This would allow applications for smartphones to be developed instead of requiring students to use a web browser.

More Testing

At the moment I have written a reasonable number of functional tests for the application. However I would like to not only write more tests, I would like to run mutation testing on the application to make sure that the tests that I do have are comprehensive.

The application also turned out to be a lot more frontend-heavy than I originally predicted. Therefore I would also like to have some tests for this frontend. I would do this using some sort of browser automation testing framework such as Selenium.

Refactor Code

Javascript At the moment the Javascript is written using classes. While this is useful in many cases, in this situation it offers way more power than is required and adds significant complexity. This is especially noticeable due to the asynchronous nature of the Javascript code. For example, if you use setTimeout or setInterval inside a class it will detach and can no longer access private attributes and methods of the class. Instead I would like to refactor this so that the Javascript is namespaced, this would still allow the different functionality to be separated but would solve a lot of complexities that exist with object orientation.

Backend I would like to take some time to look through the Python code to look for any duplicated code that could be separated out into a separate file/class, this would improve maintainability and readability.