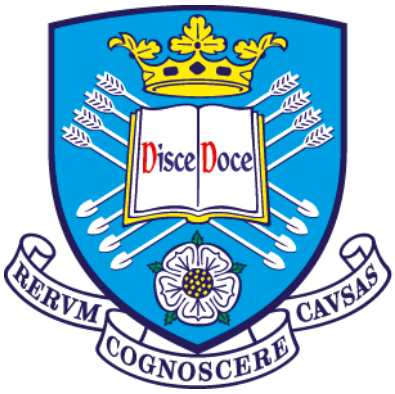
COM3610: Dissertation

Managing The Extenuating Circumstances Process



The University Of Sheffield

This report is submitted in partial fulfilment of the requirement for the degree of Computer Science by **Rushil Shah**.

Date: 03/12/2018

Supervisor: Dr. Siobhán North

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

_____

Rushil Shah

Abstract

Background

Students who suffer from personal problems usually fill out the Extenuating Circumstances Form (ECF) which are then processed by various departments, some with higher clearance than others. This project is based on creating a system which can achieve confidentiality, ease of access and increase efficiency by reducing paperwork. Data has to remain confidential not only with different levels of hierarchy but also from outsiders trying to gain access.

Project Aims

The system should make it easy for the students as well as the scrutiny committee to process extenuating circumstances efficiently and reduce the confusion that occurs when papers are involved. Confidentiality, ease of access and efficient data processing has to be achieved from this project.

Achievements to date

Research based on different types of potential systems has been completed with Django a web based system having the greatest ability for this project compared to Mobile phone applications or Artificial Intelligence or Augmented/Virtual Reality. Different web based programming languages were also accounted for and Django fit perfectly.

Acknowledgements

I would like to thank my supervisor Dr. Siobhán North for guiding and supporting me throughout the project, without the guidance it would not have been possible to complete this project.

I would also like to thank my family and friends for encouraging me and believing in me to complete this project.

Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Background

Students who suffer from personal problems usually fill out the Extenuating Circumstances Form (ECF) which then allows the examination board (the scrutiny committee) to consider their problems. The process usually begins with the student submitting the form and being reviewed. In situations where there is not enough evidence, the committee asks the student to provide more evidence and then processes the information accordingly, for example the committee may ask the student to present a death certificate. The form at this point has detailed information about the personal problem but from here on, only the general problem (without including details of the specific problem the student is facing) is taken ahead to the other departments and also added to the Students Record to allow leniency in terms of examinations and other assignments. Information regarding the ECF's can be found on the University Of Sheffield's page.[4]

1.2 Problem

Students sometimes fill out the form with less detail and so they need to be re-filled or the students have to provide more feedback/evidence. This leads to multiple documents being created and a lot of paperwork explaining the scenario. The forms are then accessed and duplicated by other departments and also added to the system for the Student Report. Multiple departments with different hieratic allowances creates a lot of confusion which then leads to improper book keeping and a reduction in information confidentiality. These forms may also be up for appeal and with multiple and mixed up forms the whole process tends to be biased.

1.3 Aims And Objectives

The system has to reduce paperwork, be secure enough to allow different levels of confidentiality and access and also be an easy but efficient to use system. A system which can be accessed by students, the scrutiny committee and can also be integrated into the student report. The forms contain important information in terms of text and so using a device such as a smart phone would not be efficient and could being up accessibility issues. On the other hand a web based system with the correct levels of security and encryption can be access by anyone as long as they have a device connected to the Internet. All University buildings have access to computers making browsing easy. A big screen would allow accessibility and information can be processed swiftly.
Django[5], a high-level Web Framework based on Python allows the design of the system we are looking for. Its built-in features allows the coder to avoid and prevent

security threats such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking[5]. It also data to be stored and encrypted in a database,

1.4 Report Summary

The report consists of four different chapters including this one. Chapter Two - Literature Review which expands on the use of web systems for solving the problem as well as other possible techniques that could be used but were not because of limiting factors. This chapter contains a broad variety of information and research. Chapter Three - Requirements And Analysis, contains specific information building on the previous chapter and contains evaluation of the project. Chapter Four - Progress, Conclusion And Project Plan; will talk about the the testing and results that have taken place with all achievements to date with a future plan.

Chapter 2. Literature Review

2.1 Introduction

Digitalising the Extenuating Circumstances Forms would have different platforms to choose from as technology has upgraded significantly over the past decade. Web based Systems, Smart Phone Applications, Virtual Reality, Augmented Reality, Artificial Intelligence[6] and recently introduced Blockchain Technology. However, for a system allowing users to fill forms it would be difficult to use Virtual Reality and Augmented Reality as these are used for altering the perception of the world[7]. Same goes for Artificial Intelligence as this would be costly to implement and not very useful when it comes to Extenuating Circumstances. In this section we will look at Web Based Systems, Mobile Phone Applications and the Blockchain Technology. Further ahead we will eliminate Mobile Applications and Blockchain and focus more on the Web Based Systems.

2.2 Web-Based, Smart Application Or Blockchain?

Blockchain is built on transparency. It will allow the storage of forms and data to be secure and encrypted with no central database holding everything. However, Blockchain is more useful when a digital property is involved as it works efficiently with trading and not storage for single entries. It still is a new technology and there is not enough detail on how exactly it can be helpful.[8]. Being a new advancement, it might not be stable enough in the long run and education on a Blockchain based system would be required for further maintenance and bug fixing.

Smart Phone Applications on the other hand are used almost daily. There are officially more Mobile Phones than people in the world[9]. All students use smart phones and it would be easy for them to use a simple application to fill in all the questions and feedbacks for the form. However, the same brings an issue for data processing and readability for the scrutiny committee and student report. The scrutiny committee would want the data to be readily available, not having to download a new application where they can barely read the large amounts of data and attachments uploaded by the student. Using mobile applications would allow instant communication as the system can use notifications for constant feedback and review processes but would only be advantageous for the students as they will be able to fill the forms on their finger tips, on the other end the scrutiny committee and other stakeholders will not like the idea of having to do everything from a small five inch phone.

Web Based Systems have been in the market for quite a while now and they still are the most used compared to any other digital technology[10]. Being accessible from any internet connected device this makes it usable from Mobile Phones,

Desktops/Personal Computers and other devices. Web system would also allow the scrutiny committee to access the system on a larger screen and allow them to make any significant changes and requests with enough detail and also move data from one stakeholder to another easily such as exporting as a PDF document. Students already use MUSE[11] and so our system would only need to be linked to the University of Sheffield logins allowing all users to login easily.

2.3 Web-Based System

2.3.1 Basic Web

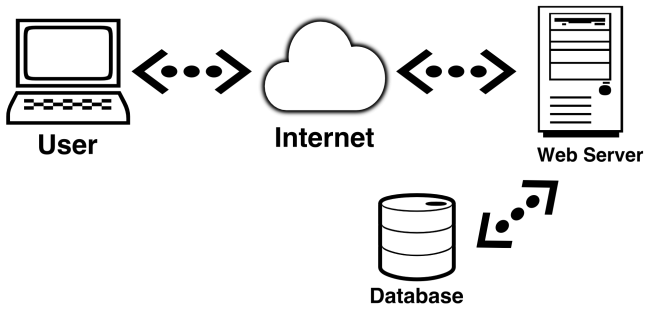


Figure 2.1: Basic Web Server.

Figure 2.1 shows the basic version of how a web server works. The user makes a request via their Internet Service Provider(ISP) onto the internet and that then connects to the web server hosting the site built and the database connected to it. Once the relevant information has been retrieved the server forwards it back to the internet and it is received by the user. Having a system like this would allow the user to access all data from any device, screen size, and hardware that is able to connect to the internet making it very flexible and easy to use.

2.3.2 The System

Web-Based systems are spread into various different languages with both positives and negatives. Java, .NET, PHP, ASP, Python, Ruby, ColdFusion are just a few of the most used language examples for web systems[12]. Figure 2.2 and Figure 2.3 shows us programming languages that are widely used for web systems and a comparison of their vulnerabilities. Selecting a secure language is an important aspect for security and confidentiality of this project.

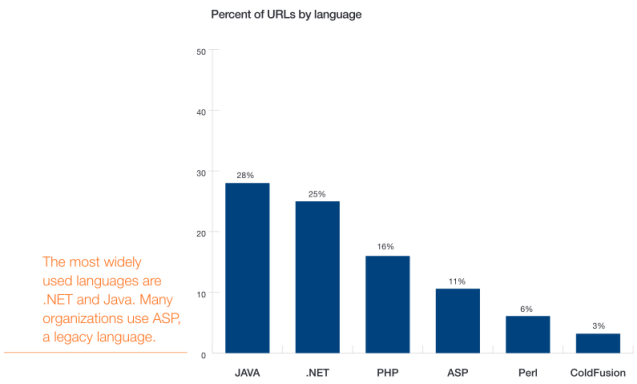


Figure 2.2: Most widely used languages for Web Systems - WhiteHatSec[1]

With all these languages, the highest priority would be to find a language which has Cyber Security as one of its strong points. .NET, Java, ASP, PHP all come with a high number of vulnerabilities as they are highly used[12].

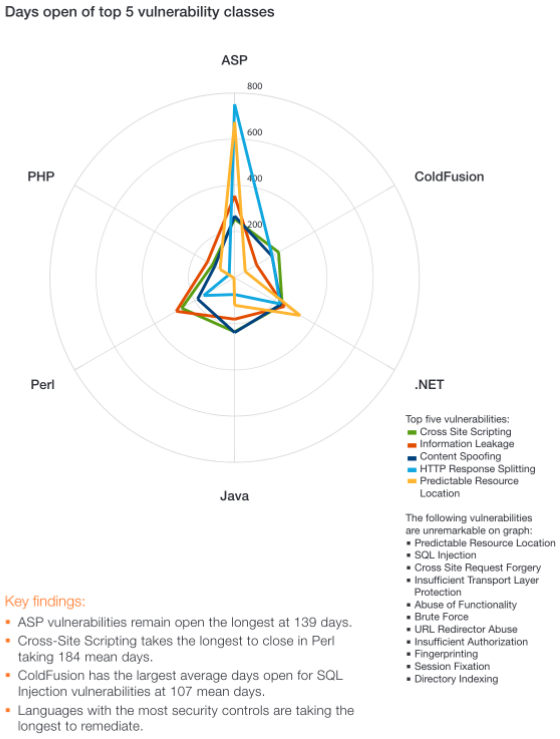


Figure 2.3: Security Vulnerabilites in ASP, PHP, Perl, Java, .NET and ColdFusion - WhiteHatSec[1]

This would then leave us with Python and Ruby which are both easy to code with and used widely. Using a web framework of the languages, Django(Python) or Ruby On Rails(Ruby) would be a smart idea because frameworks allow increased security as they are built for this and increase the efficiency and are cost effective[2].

2.4 Django

Django is a Python based web framework which allows clean and pragmatic design as well as rapid development. The framework is designed to enable the coders to code without worrying about the security of the code as it guides the coders through securing the vulnerabilities such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking. Django also has a user authentication system which can be customised to allow a secure way for data to be transferred[5].

2.4.1 Django vs Laravel

Django is a Python based framework as stated above but Laravel[13] on the other hand is a PHP based web framework. Django follows the Model View Template similar to Ruby on Rails but Laravel follows Object Oriented Programming as well as the Model View Template. In terms of website security, Django takes it extremely seriously and helps developers avoid the common mistakes which lead to the website having vulnerabilities while Laravel also has a guide to avoid making such mistakes but its contents are not as informative and detailed as Django. Django is naturally very fast as it uses Python which is known for its speed and processing. Djano beats Laravels speed in all; Plaintext, JSON and SQLite Fetch[2] as shown in 2.4.

Django: Plaintext test - 3127 requests/seconds, JSON test - 3060 requests/ seconds, Random SQLite Fetch test - 2137 requests/ seconds.

Laravel: Plaintext test - 212 requests/seconds, JSON test - 167 requests/ seconds, Random SQLite Fetch test - 140 requests/ seconds.

Figure 2.4: Django vs Laravel Speeds 2016 - CabotSolutions[2]

2.4.2 Django vs Ruby on Rails

Both are popular web frameworks, Django more than Ruby on Rails for professional developers. They are open-source which allows all code to be customised in any way. Ruby on Rails allows the use of gems which are designed by other users, these come in handy when programming personal or individual

projects however with professional projects licensing is a good idea to allow the code to be usable in the future[3].
Based on JetBrains research[14][15] we know that Python is a really popular language compared to Ruby and it is highly used for Data Analysis and Web Development among various other things while developers use a mixture Ruby Versions which makes future updates to the code harder. Table 2.1 shows the differences between Django and Ruby on Rails

	Django	Ruby on Rails
Pros	- Python is versatile - Fast - Caching System - Data Analysis - Great Security and Autentication	- Flexible - Large Community - Gems Available - Easy Migration
Cons	- Hard to debug - Monolithic architecture	- Bloated - No Data Analysis - Very explicit and inelegant to read

Table 2.1: Django vs Ruby on Rails Pros and Cons [3]

2.4.3 Summary

Django, compared to various other web frameworks, is a good environment to use for the Extenuating Circumstance Forms because it provides security we are looking for, avoids vulnerabilities and is really fast when accessing and using with SQLite. Even with small or large numbers of data, it will be able to process it really quickly. Upgrading the system at a later stage will also be simple as Python is a highly developed language.

Chapter 3. Requirement & Analysis

3.1 Introduction

This chapter will take Django as the main language used for developing the web based system and present ways in which the project can be brought together. From ways to submit the form to ways in which the scrutiny committee can access the same, this can be with extra feedback required or accepted the way it is. It will also provide detailed information on the interface of the website as well as provide test options.

3.2 Process

To begin with, the extenuating circumstances page[4] will have a link to take the student to the new website portal which will contain a login page allowing the student to login with his or her university email. Once the student has logged in, an option to start a new extenuating circumstance form will be displayed and clicking on that the student will be presented with a form to fill online. Once the online form is filled with all the details it will be stored on their profile and displayed on the "Dashboard" of the portal. The same form can then be accessed over time with new feedback, approval and other statuses. Any updates by the scrutiny committee will be notified to the student via email.

The portal will also allow the scrutiny committee to login and control the movement of the forms as well as provide feedback and comments to the student regarding their circumstance. They will also be able to request more proof and evidence if need be. The forms can then be processed further and exported as documents such as PDF's and TXT's allowing the relevant stakeholder to print or share. Figure 3.1 describes the authentication and users process.

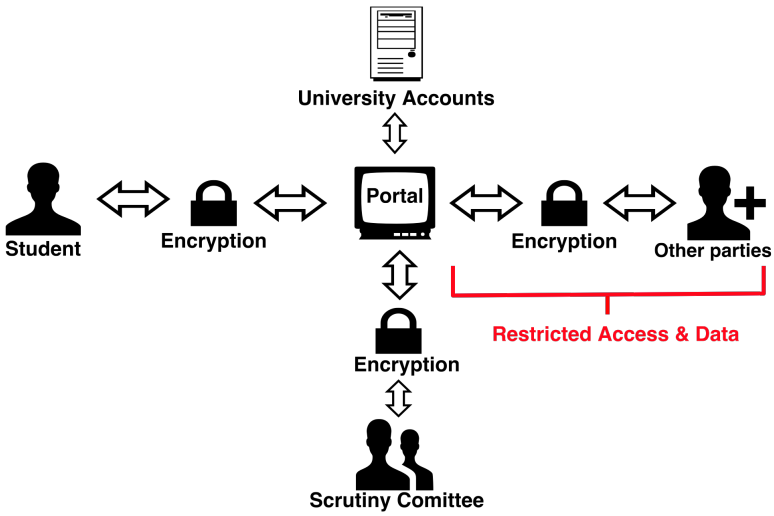


Figure 3.1: Portal and users.

The web system will be entirely encrypted and data will be stored in a secure database as presented earlier in Figure 2.1. This allows total security of the information stored and only users with access will be allowed to get that information. Some users will be restricted such as the other parties who get the students condition without knowing exactly what it is they are having trouble with. This allows confidentiality while allowing access at the same time.

3.3 Interface

The interface will consist of two different types of users; The Students and the Scrutiny Committee. There will be other third party users who will get the data from the student records such as the professors and personal tutors.

3.3.1 Student Panel

The student panel will consist of three main components; They should be able to submit a new circumstance form, access any previously submitted form and lastly check the status(described under Scrutiny Committee Panel) for any new feedbacks or requests from the scrutiny committee and upload a response or evidence as required. As stated in the previous section, the students should be able to login with their own university accounts making is easy to access from within the portal.

3.3.2 Scrutiny Committee Panel

This panel will be a little more detailed than the student panel. The panel would notify the scrutiny committee by email when a new form has been received The committee would be able to view the form and request any new feedback or

evidence and change the status of the form. The status could be "Approved" or "Requires changes" or "Pending" which would allow different members of the scrutiny committee to know the status of a form. In this case;

- Approved would mean the student does not need to provide more information or evidence and the details of the form have been passed on to the third parties.
- Requires changes would mean the student has to provide more information such as evidence or detailed writing. The requirements would be accessible by the student under the "Feedback" section of the form.
- Pending would mean the form has not yet been checked by the members of the scrutiny committee and so there is nothing to be done.

They can also export the same form as a PDF and download it for printing purposes or for documentation.

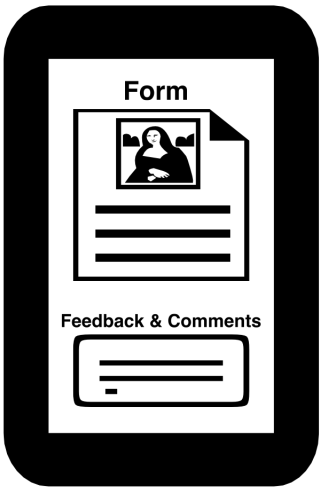


Figure 3.2: Form with a feedback section

Figure 3.2 shows the portal with a form being displayed with a feedback and comments section. This section will allow communication between the student and the scrutiny committee as well as allow uploads of new evidence.
The panel will also have a list of all the students who have submitted a form allowing the scrutiny committee to sort and search through the records to find any specific details they like which would be useful in cases of a review.

It will also enable the exporting of data to third parties, only approved details by the scrutiny committee will be accessible by the third parties that need access and knowledge of an existing extenuating circumstance form.

3.4 Testing

Testing is an important aspect of functionality and knowing if the system follows the requirements. In this system we can carry out testing in various different elements from automated to user tests. Below are a few of the tests that we can run in order to make sure the project is reachable.

3.4.1 SQL Queries

Since data will be stored into the database, we can use RAW SQL Queries to test if a certain field in the software is giving access to the database without checking the entry first. For example, if a student fills a RAW SQL Query instead of his details then the resulting should be an error and should not display the data being retrieved from the database.

3.4.2 Cross Site Scripting

Cross Site Scripting (XSS) is a malicious method of injecting scripts into the system which then target other users. These scripts can be used with JavaScript which will be used when designing the portal. Testing this will make sure that the portal is secure from outside attacks.

3.4.3 Automated Testing

Django allows writing of test code (unittest)[16] which then runs as a normal user(in our case) and checks if the response is exactly how we need it to be. For example if a student clicks on "Submit Form" then it should store the form in the database. The unittest will be able to test if the data is actually being stored in the database.

3.4.4 User Testing

In order to have covered all parts of the system it would be a good idea to have a user from the scrutiny committee as well as a student to test the system out, if they feel there is anything left out or any other relevant issues that need to be solved. This would be the last bit of testing once the system has passed the development stage.

3.5 Summary

This chapter has talked about how the development of a web based system would work, how the stakeholders involved in this project would access their data and process it based on the situation and other parties involved. Being able to make sure everything is following the requirements of the project is extremely important and so different types of testing is really helpful.

Chapter 4. Design

4.1 Introduction

The first option to begin designing would be to create a mock up of how the interface should be produced but with past experience in "Software Hut" it was decided that producing the HTML and CSS directly would be a better idea compared to a mock up which would be highly unlikely to achieve exactly. This chapter will discuss the design and reasons behind the selection of certain choices such as HTML template, JavaScript uses and Django itself.

4.2 HTML Template

The HTML template used in the production of this system needed to be able to support the different types of user groups i.e Students, Secretary Panel and Scrutiny Panel. With research a *Public Licensed*[17] Boostrap 4[18] HTML template[19] designed with a mobile first coding aspect by *Creative Tim* will be used as the base of development. The selected template is an "Admin Template" which has functionality built in such as HTML for statistics, graphs and much more which can be really useful when creating dashboards and profile pages. Each section needs specific HTML pages:

1. Basics
    - Login Page(Email and Password)
    - Register Page(Names, Email, Password and Date of Birth)
    - Password Change
    - Profile Page
    - A student profile page accessible to staff members
2. Student Panel
    - Dashboard page to display the already submitted forms and link to creating a new form
    - View form page which allows the user to see the already submitted form
    - Create a new form with required fields
    - Students available public data
3. Secretary Panel
    - Dashboard displaying all forms submitted by students
    - View form page with ability to change status of form and request more files

- Access to students public information and ability to change it

4. Scrutiny Panel
    - Dashboard page with all forms access
    - View form page to view data from the form

4.3 Javascript

JavaScript is another important part of the design as it will help manipulate the HTML in order to achieve the requirements for the system. Our system will use JavaScript in the following aspects:

- Bootstrap[18] - To enable seamless control of HTML on mobile devices
- DataTables[20] - Allow the users to search and sort data in html tables
- JsPDF[21] - Ability to download data from the HTML page to a PDF file
- html2Canvas[22] - Will work alongside JsPDF to produce the PDF

4.4 Database

Data will eventually need to be stored in a database and to do a schema for a database must exist. Figure 4.1 shows the schema created to store and retrieve data without bring up errors and making sure that the tables are interlinked. A single form can have multiple modules and files, similarly a single user can have multiple forms(One to Many). However, a single user can only have a one to one relationship with the users public data and profile.

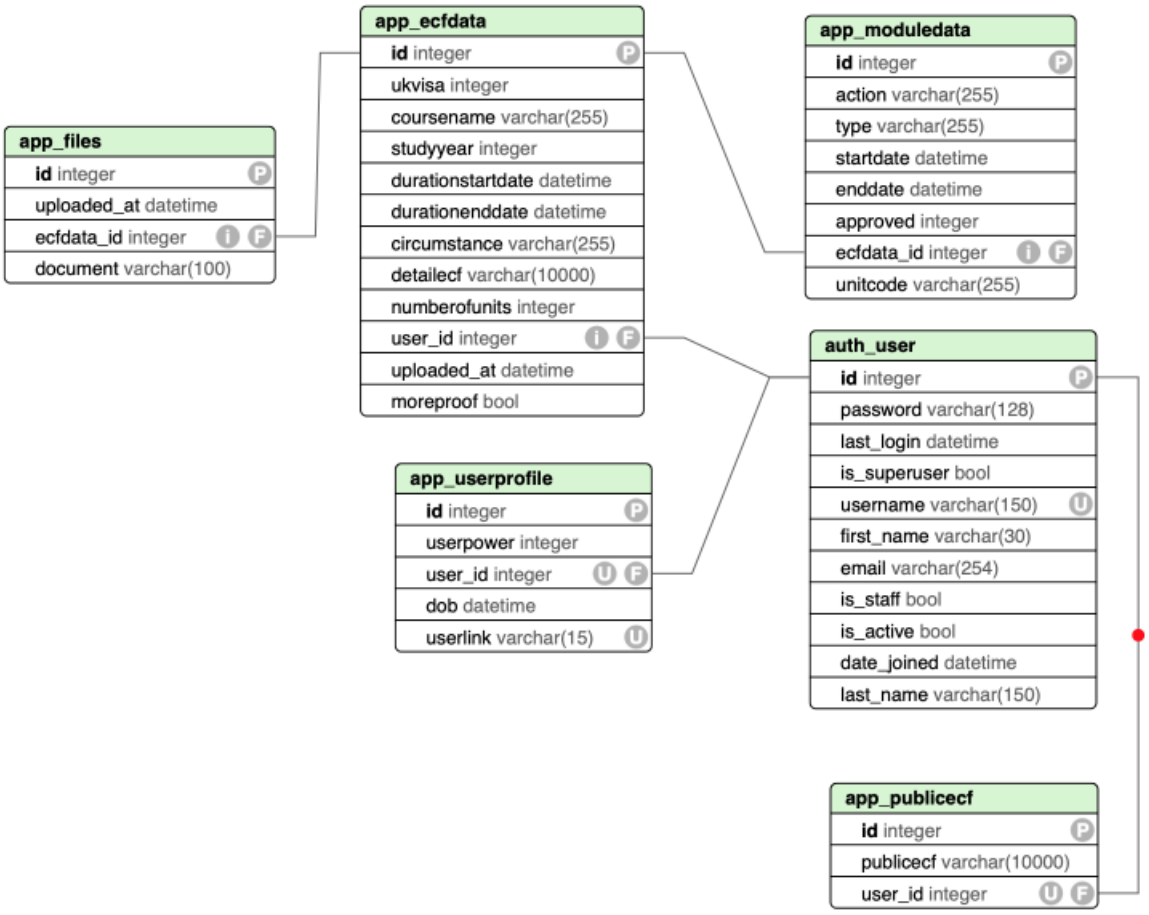


Figure 4.1: Database Schema

4.5 Django

Django will run the system as the main back-end code. Its inbuilt functionality allows us to use GET and POST forms as well as uploading and retrieving data directly to the websites HTML pages. Django also allows the use of external packages and this system will be using All-Auth[23] which is a precoded authentication package allowing the use of external accounts such as Google, Dropbox, Github, LinkedIn and many others to login directly into our system. All-Auth create a secure session system for the website which behaves as our authentication into the site. It has a *login_required* function which can be used with individual pages to prevent unauthorised accesses without logging in.

Bcrypt[24] is a password hashing function which can be combined with Django and All-Auth to securely store user passwords into the database with a salt hash. Our system will be using a SHA256 BCrypt Hasher which is currently one of the

safest password encryptions[25].

4.6 Summary

The design as stated above consists of three key elements which are HTML, JavaScript and Django. With all these available implementation of the system will be secure and easily accessible on all devices as well as allow the requirements to be fulfilled.

Chapter 5. Implementation & Testing

5.1 Introduction

This section will cover the implementation of the system. Development requires production, bug fixes and finally testing which will all be discussed in this section.

5.2 Implementation

The implementation process began with re-constructing the HTML template by Creative Tim[19] to the different sections mentioned in chapter 4.2, JavaScript going along with the HTML and finally the Django back-end.

5.2.1 HTML

Redesigning the HTML code required changing the CSS, HTML components and removing unwanted JavaScript code. This brought about various bugs and unexpected changes in the overall HTML which crashed a few elements of the code such as Tables. With the use of Google Chrome and Console the debugging was completed and successful HTML pages were created for all three different sections as mentioned in Chapter 4.2. However, since pages are going to be repeated(Dashboard, Viewform, Profile, etc) only one HTML code was required and repeated depending on which panel you are logged in as. Table 5.1 shows what components were created into their relative files by redesigning the public licensed template. Each of these were combined with CSS and JavaScript from Chapter 4.2.2

Page	**HTML Components**
Login	POST Form(Email, Password)
Register	POST Form(Firstname, Lastname, Email, Password1, Password2, DOB)
Password Change	POST Form(Current Password, Password1, Password2)
Profile Page	POST Form(Email;Disabled, Firstname, Lastname, DOB), TextArea(PublicData)
Dashboard	Statistics, Table(ID, Description, Date, View)
View Form	Text Fields(User Details, Circumstance), Table(Module Details), Table(Uploaded Files), POST Form(Files)
Units	GET Form(Number of modules)
New Form	POST Form(Circumstance, Module Details, Files)
Public Data	TextArea(PublicData;Disabled)

Table 5.1: HTML Components & Relative Pages

5.2.2 JavaScript

After completion of the HTML design, it was important to add JavaScript in order to allow smooth control of the website. Along side the JavaScript received with the template by Creative Tim[19] such as **Bootstrap**[18] other scripts were manually added.

Tables would load numerous entries for the *Secretary Panel* and the *Scrutiny Panel*, this caused the page to be extensively long without the ability to filter out or search for a specific form, user or find pending forms. In order to achieve efficiency and sort out the entries in the table the use of **DataTables**[20] came in handy. It automatically completed the following;

- Added a search field - Allows the user to search for data in the entries
- Limited entries shown and added more pages - Allows efficiency and speed
- Sort by ascending or descending - Useful when searching for *Pending* forms

The above are a requirement as being able to sort the forms, users and finding exactly something specific is an important factor which can be useful when a form needs to be reviewed or assessed.

Another important public script used was **jsPDF**[21], as per the requirements, the secretary needs to be able to print and download the forms. PDF would be a perfect format to download these files as any browser, or PDF compatible viewer can be used to view the form. This JavaScript code converts the content inside a specific *HTML container* and copies that data to a PDF which can then be downloaded by the user, in our case, the secretary. When the implementation of this was complete, the data from the form would be saved simply as text and would loose its colours and format. This would scramble up all the data and make it highly unreadable. To overcome this bug, **Html2Canvas** was used.

Html2Canvas[22] is a MIT licensed script which creates an image for everything on the current web page or a specific HTML container with the CSS (including colour and design). This is then copied onto the PDF created by jsPDF as mentioned above and made available for download. This allows the form to be printed exactly the same way it would be seen when viewing it on the web-page.

Lastly, the secretary can also directly print the form while viewing it. All modern browsers have the ability to print directly what is being displayed on the current web page and with a simple line of code(*window.print()*), a button on the view form html allows the browser to detect a print command being sent.

5.2.3 Django

The main part of the implementation is combining both the **HTML** and **JavaScript** to work along with **Django** being the main controller. In the implementation, Django 2.2[26] which is the latest version was used. The first step was simply to combine the HTML, JavaScript, CSS and Django together into a folder system and define it on Django's settings. Within seconds the entire HTML was accessible on a Django running web server. The three sections below explain the process of coding in Django, the difficulties faces and lastly a small summary on back-end development.

Authentication & Database

After combining all the above, the authentication system was first to be built. As stated in Chapter 4, **All-Auth**[23] was implemented into the Django system by importing the relevant packages. However, in order to avoid users creating multiple accounts with multiple emails, a manual **validation** was coded which only allows the student to register with their University Email Address (**sheffield.ac.uk**) only. All HTML files for All-Auth were overwritten by the HTML design already created in order to have a continuous site. At this point, all data being registered from All-Auth was bring stored into a **SQLite database** which was inbuilt with Django. Creating the database schema seen in 4.1 was

extremely simple with the **Model, View and Controller system** which Django follows. The database was ready in seconds after defining it in Models.

New Form

At this point, the authentication system was ready as well as the database in which all the data would be stored. In order to retrieve data from the database there needed to be some data already inserted and so in order to do that the "Create new form" page had to be completed with back-end code. Different students may have different number of modules which are affected by their circumstance and so the first step would be to ask them to fill in how many modules are affected by their circumstance. This would send in a GET request with the parameter '**units**' and its **value** as an integer. The form for creating a new extenuating circumstance would then get this integer from the GET link and produce those many 'input fields' for the modules section of the form. The creation of a HTML POST form is made extremely easy with Django's **forms.py** system where you simply define the fields of a form such as text input or email and it will automatically present the form on the template where it is called. On successful submit, the student is notified, redirected to dashboard and the form data is saved into the database. The student is also notified if the form was not submitted fully and if it should be re-submitted.

Dashboard

Student: Once the form has been submitted, the student can see it on the list of forms submitted. The dashboard provides statistics based on the current status of the all the forms submitted. These statistics are counted based on the status of each module under each form. If all modules are approved, the form will be '**entirely approved**' and this will be visible to the student under the statistics. The same goes for pending and rejected. However, if the modules are set to different statuses such as 'rejected' and 'approved' under one form, the overall result is '**partially approved**' or '**partially pending**' if the module has one or more pending modules. The student can see exactly which module has been rejected, approved or still pending. All data is retrieved from the database as follows mathematical calculations where each module is added onto the statistics based on the other modules within the same form. The calculations as well as the display of each form was completed easily with the use of **FOR** loops.

Secretary & Scrutiny Panel: If the secretary or someone from the scrutiny panel logs into the system, the dashboard follows the same format as that of the students but with access to all forms submitted by students and their public data. The statistics of the forms are also similar to that of the students allowing the secretary or the scrutiny panel to know exactly how many pending forms are there which need to be updated.

View Form

Student: The dashboard links each of the forms created to a view form page. This page retrieves data from the database based on the GET parameters(form ID) received when requesting the page. Interlinked data from the database is then retrieved based on the **Form ID** such as the *User* who created it, the *Files* uploaded under the form and lastly the *Modules* affected. Each of these is stored in different tables as they share a one to many relationship. Apart from simply displaying the form data already submitted, there is an option triggered by the secretary which creates a new **file up-loader** at the top of the page allowing the student to upload more files. The file up-loader submits the new files as a POST form and saves based on the form ID.

Secretary Panel: This page follows the same format as the student view form with an exception of allowing the secretary have control over editing the final status and action for each module and requesting more files. Each module is displayed in a table with individual buttons bringing up a '**pop-up**' also known as '**Modal**' in HTML5. The modal pop-ups then allow the secretary to change values for the 'Action' and 'Status' of the form. **HTML drop-down** allows us to display text as the displayed options while the back-end gets a '**value**' which is stored into the database as an integer. This same value is automatically selected when displaying the current selection as it retrieves the integer from the database. A **switch-case toggle** was designed to allow the secretary to request more files from the student. When the switch is toggled, the database value for '**requestfiles**' is switched. This then displays the additional files up-loader as stated previously. The switch is called by a POST submit button with a '**name**' attribute to differentiate between the different POST forms. Lastly, for ease of access, a 'textarea' was created allowing direct change of the students public data without having to open multiple tabs and make changes. This again uses a POST form with a different 'name' attribute. Django '**views.py**' controls all the data received from the POST request and cleans the data ready to be saved into the database with the '**save()**' function.

Scrutiny Panel: Since the scrutiny panel does not control the status directly on the portal, the view form page simply retrieves data from the database and displayed it with the use of HTML components such as p, a, h1, h2, h3, label, and many others.

Profile Page

The profile page simply displays the current users information. If the user is a student they can view their public data in a **disabled** '**textarea**', this allows scrolling of long paragraphs without having to extend the page size. Users cannot change their email address as this would bring about issues of multiple sign ups with different emails hence the **validation of only sheffield.ac.uk emails**. A simple POST form on the page retrieves the values current in the database and allows changing them after cleaning the new data. **This includes: firstname;**

lastname; dob only.

Student Record

Every user when registered is provided with a **8 digit random key** which can is used as a **GET parameter** to access their public record. Only users who have the **KEY** can access these records making it secure. This random key is stored in the database under '**User Profile**' and has a **unique schema** which makes sure that two users do not have the same KEY. The public data again is simply displayed with a '**textarea**' HTML component which is coded as mobile first. The purpose of using a random key is to **avoid pattern detection**(For example, using an email instead of a random key) and **unauthorised access.**

Secretary Panel: Since only the secretary can give out the direct link to access the student's record, they also have access to **regenerating the key**. This is a **safety feature** to avoid unauthorised access. The regeneration uses the same principle of using **Python's Random package** to generate an **8 digit key** and replace it in the database.

Email Notifications

Email notification were created with the use of **Gmail's SMTP services**. Django has an inbuilt system which makes mailing and bulk mailing extremely easy. The SMTP details of the sender email can be set in the settings.py page which directly links to the **Django mailer** and with a simple *sendMail* function we can send mail. To make debugging and later changes to the email text being sent, each message is stored as a TXT file which is called in the *sendMail* function when required. For example, when a new user registers, a confirmation email is sent to the user using the *sendMail* function and uses the data from **NewUser.txt** as the main message. Email notifications have been implemented for the following instances;

- New user registration
- Password reset and confirmation of password change
- Successful creation of extenuating circumstance form
  - Email is sent to both Student and Secretary
- Updated status of the form (Rejected/Approved)
- Requested additional files
  - Email is sent to the Student and once uploaded to the Secretary

5.3 Complications

Different types of complications were generated when building the system in Django. The tables 5.2 and 5.3 below explains each of the complications and its relevant solution for each of the panels.

Student Panel

Page	**Issue**	**Solution**
Dashboard	Calculate based on forms and not each individual module within the form.	Use nested FOR loops to calculate statistics for each module within the form and only display the final count as the main form statistic.
New Form	Create module input fields based on the GET parameter value and allow each of the modules to be stored into the database	Retrieve the GET parameter value from views.py and import it into forms.py where the input fields are generated from. Use a FOR loop to create an input field with increasing HTML attributes eg. unitcode1
New Form	Allow only ZIP files to be added and validated	Create a Django validations.py function which detects the format of the file and rejects if it is not a ZIP.
New Form	Store files with the same name without replacing them	Rename the uploaded file by adding a 5 digit random string to the end to prevent files being replaced when stored
View Form	Prevent other users from accessing current users form	Create validations which check if the form belongs to the current logged user, if so, then the user is allowed to access it.

Table 5.2: Complications generated and their solutions under the Student Panel

Secretary Panel

Page	Issue	Solution
Dashboard	Calculate based on forms and not each individual module within the form.	Use nested FOR loops to calculate statistics for each module within the form and only display the final count as the main form statistic.
View Form	Modal popup to edit form details would load on a different page instead of current one	Instead of having the modal on a different file, having it in the current file with a JavaScript code to retrieve data and fill it in allowed the popup to load on the current page seamlessly
View Form	Multiple POST forms would bring about issues when working on views.py as the controller	Provide each POST request with a unique 'name' attribute to differenciate which part of the views.py it should access using IF statements
View Form	Switch-case Toggle HTML would not change the switch based on the current value in the database	Added a JavaScript code to manually get the value from the database and switch the toggle CSS.

Table 5.3: Complications generated and their solutions under the Secretary Panel

5.4 Testing

In order to make sure that the implementation is secure a few tests were carried out as mentioned in Chapter 3. The first was **SQL queries** being executed directly from the any of the POST and GET forms. When the form was submitted the back-end directly took the data as being unsafe and produced an error. As stated in Django's **security documentation**[27] the SQL commands were **escaped and ignored** hence making it safe to use.

Another test executed on the system was **Cross Site request forgery(CSRF)**, this meant duplicating a cookie from a browser where the user was already logged in and trying to send POST requests from another browser. Each POST form in Django requires a **Secret Key** to be present, and when the duplicated cookie POST request did not have a Secret Key, Django automatically **denied access** to the action. This security feature is extremely important as common **Remote Administration Tools** can access your cookies and duplicate them.

Cross Site Scripting(XSS) had its limitations when testing, for example if a CSS Style was defined within the HTML code, a simple execution of a JavaScript code would be possible to access unauthorised data but as there was no such code present a significantly professional code would be required to test and penetrate through Django's security system for auto escaping unknown code.

Simple **Automated tests** called '**unittest**' were also executed using Django's inbuilt features to test out the execution of **POST and GET forms**. A simple unittest for POST forms such as login, profile update, and request more files was executed where the result was displayed in console. The result were verified using **response code** received when the code was executed as well as the page location. For example when testing;

```
client.post('/account/login/', {'email': 'rshah5@sheffield.ac.uk',
 'password': 'complexp@ssw0rd'})
```

The response received was '200' with the HTML code for the dashboard page.

Lastly, the system was **manually tested** for bugs and security loop holes as well as **user reviews** based on ease of access, confidence and comparison. When running manual tests, a user with Student privileges would be able to access pages where only a Secretary would have access. For example simply typing

```
<link to website>/secretarypanel/viewform?id=1
```

would allow the student to access and change data as though they were the secretary. Same applied for the Scrutiny panel as well as regular forms created by other Students. In order to prevent this, **validation checks** were implemented which would check the privileges of the user and only give them access to specific pages in the system. The checks would also make sure that the logged in **session user** is accessing only data created by them and not by other users(mainly for the Student Panel as the Secretary has access to all).
Students who had previously submitted the paper extenuating circumstance forms and students who did not know how to submit the extenuating circumstances were asked to provide their views on the system created on Django and four out of five found the system easy to use, user friendly, and extremely efficient compared to the previous paper based forms.

5.5 Summary

The implementation of the system was mostly completed except for a small security section which involved encrypting the uploaded files in order to store them securely on the hosting server. With future work this can be made possible. Security and user testing has played a key role to allow changes to be made for maximum security with easy access and usage.

Bibliography

[1] W. Security, "2014 website security statistics report." http://info.whitehatsec.com/rs/whitehatsecurity/images/statsreport2014-20140410.pdf, 2014. Accessed: 06.11.2018.

[2] Cabotsolutions, "Django vs laravel: A comparison of web application frameworks." https://www.cabotsolutions.com/django-vs-laravel-a-comparison-of-web-application-frameworks, 2018. Accessed: 01.10.2018.

[3] J. Hargan, "Django vs rails: Picking the right web framework." https://www.tivix.com/blog/django-vs-rails-picking-right-web-framework, 2018. Accessed: 11.11.2018.

[4] U. Sheffield, "Extenuating circumstances form: Explanatory notes - forms - ssid - the university of sheffield." https://www.sheffield.ac.uk/ssid/forms/circsnotes, 2018. Accessed: 01.10.2018.

[5] Django, "The web framework for perfectionists." https://www.djangoproject.com/, 2018. Accessed: 01.11.2018.

[6] F. J. DeMers, "7 technology trends that will dominate 2018." https://www.forbes.com/sites/jaysondemers/2017/12/30/7-technology-trends-that-will-dominate-2018/#7e4e30157d76, 2017. Accessed: 01.11.2018.

[7] J. Ronzio, "7 technology trends that will dominate 2018." https://cramer.com/story/the-difference-between-ar-and-vr/, 2018. Accessed: 03.11.2018.

[8] A. Castro, "Blockchain' is meaningless." https://www.theverge.com/2018/3/7/17091766/blockchain-bitcoin-ethereum-cryptocurrency-meaning, 2018. Accessed: 06.11.2018.

[9] Z. D. Boren, "There are officially more mobile devices than people in the world." https://www.independent.co.uk/life-style/gadgets-and-tech/news/there-are-officially-more-mobile-devices-than-people-in-the-world-9780518.html, 2014. Accessed: 06.11.2018.

[10] A. Larson, "How many websites are there?." https://www.independent.co.uk/life-style/gadgets-and-tech/news/there-are-officially-more-mobile-devices-than-people-in-the-world-9780518.html, 2018. Accessed: 08.11.2018.

[11] U. O. Sheffield, "Making the most of muse." https://www.sheffield.ac.uk/bms/undergrad/muse, 2016. Accessed: 01.12.2018.

[12] Upguard, "Which web programming language is the most secure?." https://www.upguard.com/blog/which-web-programming-language-is-the-most-secure, 2018. Accessed: 01.10.2018.

[13] Laravel, "Laravel - the php framework for web artisans." https://laravel.com/, 2018. Accessed: 11.11.2018.

[14] JetBrains, "Jetbrains - python." https://www.jetbrains.com/research/devecosystem-2018/python/, 2018. Accessed: 11.11.2018.

[15] JetBrains, "Jetbrains - ruby." https://www.jetbrains.com/research/devecosystem-2018/ruby/, 2018. Accessed: 11.11.2018.

[16] Django, "Documentation - writing and running tests." https://docs.djangoproject.com/en/2.2/topics/testing/overview/, 2018. Accessed: 20.11.2018.

[17] C. T. License, "Free html admin template - license." https://www.creative-tim.com/license, 2019. Accessed: 10.02.2019.

[18] Bootstrap, "Boostrap." https://getbootstrap.com/, 2019. Accessed: 10.02.2019.

[19] C. Tim, "Free html admin template." https://www.creative-tim.com/product/material-dashboard, 2019. Accessed: 10.02.2019.

[20] DataTables, "Datatables - table plug-in for jquery." https://datatables.net/, 2019. Accessed: 10.02.2019.

[21] MrRio, "Javascript generation for html to pdf." https://github.com/MrRio/jsPDF, 2019. Accessed: 10.02.2019.

[22] N. von Hertzen, "html2canvas - screenshots with javascript." https://html2canvas.hertzen.com/, 2019. Accessed: 10.02.2019.

[23] IntenCT, "Authentication for django." https://www.intenct.nl/projects/django-allauth/, 2019. Accessed: 10.02.2019.

[24] dwaiter, "Django-bcrypt." https://github.com/dwaiter/django-bcrypt, 2019. Accessed: 10.02.2019.

[25] D. Boswell, "Storing user passwords securely: hashing, salting, and bcrypt." http://dustwell.com/how-to-handle-passwords-bcrypt.html, 2012. Accessed: 10.03.2019.

[26] Django, "Documentation - 2.2." https://docs.djangoproject.com/en/2.2/, 2018. Accessed: 20.11.2018.

[27] Django, "Documentation - security." https://docs.djangoproject.com/en/2.2/topics/security/, 2018. Accessed: 20.11.2018.