# 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: 01/05/2019

**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

This project manages the extenuating circumstances process by eliminating paper-work and increasing efficiency. Django, a Python framework, provided us with a platform to create a web-based system allowing students to submit their problems and circumstances on a portal. The scrutiny committee and the secretary can also access these forms and decide if the student gets approved/rejected for leniency toward their academics or needs to upload additional proof. The secretary fills out a sanitised version of the form which can be accessed with unique links and shared on the student's portfolio. In situations where a review is called for, the data can be retrieved, filtered, printed or downloaded from the portal. The system not only allows data to be organised but also accessible on any device as long it has internet. The system passed through different tests and scenarios in terms of security as well as functionality. The current system has a decent element of security, provided by Django's inbuilt features and additional packages from other developers. However, with further work, the system can be made bulletproof in terms of security and have additional features which can make communication easier between all stakeholders.

# Acknowledgements

# Contents

# List of Figures

# List of Tables

# Chapter 1. Introduction

## 1.1   Background

Students who suffer from problems 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 which then gets 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 or require more proof, and so they need to be re-filled or additional proof needs to be uploaded. The data from the form is simplified and confidential data is removed after which it is made available to the staff members and relevant departments. The simplified version of the form is also exported into the Student's portfolio. Various departments with different hieratic allowances creates much confusion which then leads to improper bookkeeping 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 and loss of data is inevitable.

## 1.3   Aims And Objectives

The system has to reduce paperwork, be secure enough to allow different levels of confidentiality/access and also be easy but efficient to use the system. A system which can be accessed by students, secretary, the scrutiny committee and can also be exported to the student portfolio. The forms contain essential information in terms of text, and so using a device such as a smartphone would not be efficient and could bring up accessibility issues. On the other hand, a web-based system with the correct levels of security and encryption can be accessed 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 allow 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 seven different chapters including this one. Each chapter will talk about the process of research, design, implementation, results and lastly the conclusion. Every chapter will introduce something new while the basis of the project remains the same, to solve the problems of extenuating circumstances.

# 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 challenging 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, we will eliminate mobile applications and blockchain and focus more on Web-Based Systems.

## 2.2  Web-Based, Smart Application Or Blockchain?

Blockchains production base is 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 smartphones, and it would be easy for them to use a simple application to fill in all the questions and feedback 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 fingertips, 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 quickly 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 log in smoothly.

## 2.3 Web-Based System

### 2.3.1 Basic Web
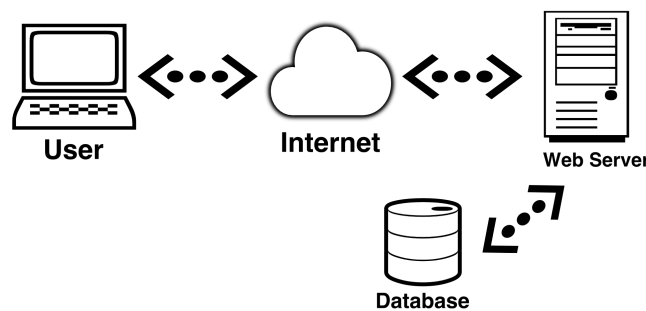


Figure 2.1: Basic Web Server.

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 the user receives it. Having a system like this would allow the user to access all data from any device, screen size, and hardware that can connect to the internet making it very flexible and easy to use. Security would need to be an essential aspect.

### 2.3.2 The System

Web-Based systems use 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 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.

Percent of URLs by language

The most widely used languages are .NET and Java. Many organizations use ASP, a legacy language.

JAVA 28%  .NET 25%  PHP 16%  ASP 11%  Perl 6%  ColdFusion 3%
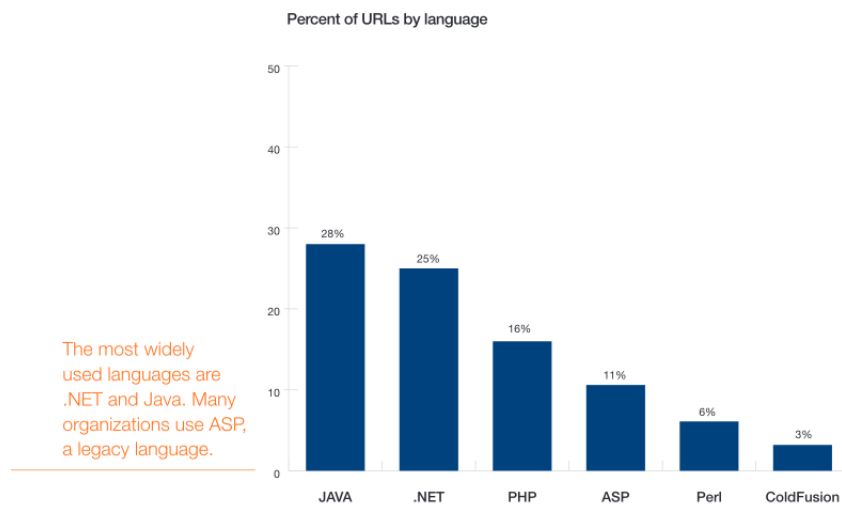
Figure 2.2:   Most widely used languages for Web Systems - WhiteHatSec[1]

Days open of top 5 vulnerability classes

ASP      ColdFusion      .NET      Java      Perl      PHP

Top five vulnerabilities:
- Cross Site Scripting
- Information Leakage
- Content Spoofing
- HTTP Response Splitting
- Predictable Resource Location

The following vulnerabilities are unremarkable on graph:
- Predictable Resource Location
- SQL Injection
- Cross Site Request Forgery
- Insufficient Transport Layer Protection
- Abuse of Functionality
- Brute Force
- URL Redirector Abuse
- Insufficient Authorization
- Fingerprinting
- Session Fixation
- Directory Indexing

Key findings:
- ASP vulnerabilities remain open the longest at 139 days.
- Cross-Site Scripting takes the longest to close in Perl taking 184 mean days.
- ColdFusion has the largest average days open for SQL Injection vulnerabilities at 107 mean days.
- Languages with the most security controls are taking the longest to remediate.

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

With all these languages, the highest priority would be to find a language which has cybersecurity as one of its main components. A huge number of programmers use .NET, Java, ASP, PHP and hence they come with a high number of vulnerabilities making them highly maintainable and certainly less secure.[12]. Python which is another highly used programming language is said to have more security than some of its alternates; this is due to the huge community and preference of cyber researchers.[13]

All the above are regular languages which have a framework under web development. Python has Django; Ruby has Ruby on Rails, JavaScript has NodeJS and many others. Each of these can help develop a system with similar functions, but not all can provide the same levels of security, speed or scalability.[2][14].

## 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 code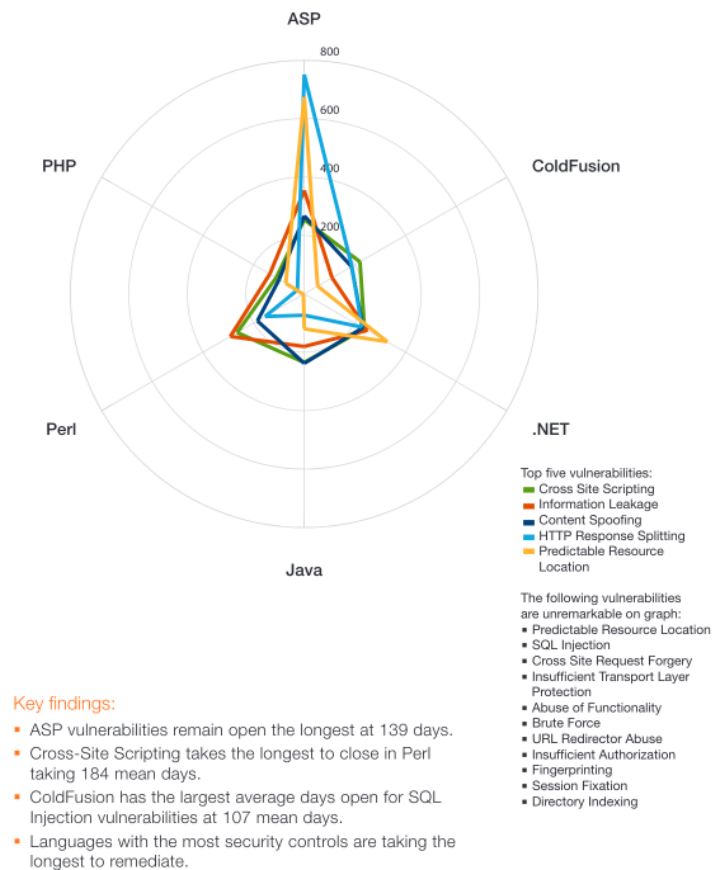rs to code without worrying about the security of the code as it guides the coders through securing the vulnerabilities such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking. Django also has a user authentication system which can be customised to allow a secure way for data to be transferred[5]. Not only does Django provide security, but it also allows efficient and secure coding as it follows the model, controller view system which avoids repeated code. With Python being the primary language, numerous frameworks are available to make working with Django even easier.[15]

### 2.4.1 Django vs Laravel

Django is Python-based as stated above but Laravel[16] 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 avoiding 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. Testing of security would need to be done manually unlike Django where the system makes sure that all security protocols are in place with the latest updates. Django is naturally very fast as it uses Python which is known for its speed and processing. It beats Laravels speed in all; Plaintext, JSON and SQLite Fetch[2] as shown in Figure 2.4.
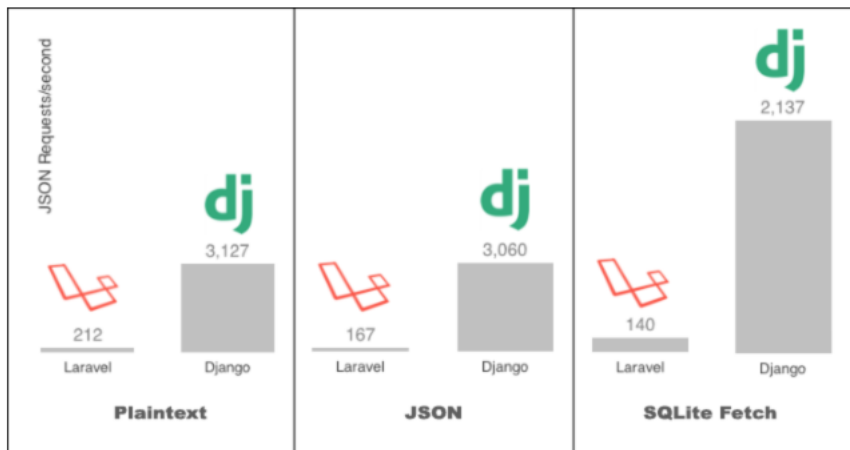
**Django:** Plaintext test - 3127 requests/seconds, JSON test - 3060 requests/ seconds, Random SQLite Fetch test - 2137 requests/ seconds.

**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 customisation 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]. Django, on the other hand, has user coded packages available for use with the application in development. An example would be *django-pgcrypto*[17] which allows the encryption of database entries.
Based on JetBrains research[18][19] we know that Python is a prevalent 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<br>- Fast<br>- Caching System<br>- Data Analysis<br>- Great Security and<br>Authentication | - Flexible<br>- Large Community<br>- Gems Available<br>- Easy Migration |
| Cons | - Hard to debug<br>- Monolithic architecture | - Bloated<br>- No Data Analysis<br>- Very explicit and<br>inelegant to read |

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

### 2.4.3  Django vs NodeJS

NodeJS[20] uses C, C++ and JavaScript to build the web application. The use of JavaScript mainly is excellent when it comes to the design and feel of the web system but not when it comes to security. NodeJS can be used to produce a progressive web application which will make the website feel and work like a smart application even when there is no network as well as work perfectly on larger screens too. However, being that security is an important requirement for our system, Django would win over NodeJS[21]

### 2.4.4  Security

A paper on Web Development Done Right[22] has considered Django's security as well as all other aspects to build a secure web-based application in Django. According to the research, attackers need to find a single flaw in the system, and they would gain access to almost everything. Django allows mitigating this difficulty and prevents most of the common mistakes that coders make. The paper numerous security attacks such as SQL Injection, Cross-site Scripting, Session Forging, Email Header Injection, Directory Traversal, Exposed Error Messages and more with solutions on how Django can prevent each of them. A great number of professional coders has appreciated Django's security overall.

## 2.5  Summary

Django, compared to various other web frameworks, is a excellent environment to use for the Extenuating Circumstance Forms because it provides the security we are looking for, avoids vulnerabilities and is fast when accessing and using with SQLite. Regular updates provided by the developers of Django allow security maintenance at all times. Even with small or large numbers of data, it will be able to process it 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 compiled. Django provides a platform through which we can create a fully functional web-based design, but within the design, there are alternative ways to go about. Eventually, there will be changes in the implementation, but this chapter will cover the basis of the implementation decisions.

## 3.2 Analysis

The extenuating circumstances page[4] would need to have a link to the portal which will contain a login page allowing the student to log in 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. On successful completion of the online form, the data 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 the secretary and the portal.

The portal would 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 would 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. In this case, the secretary is part of the scrutiny committee.
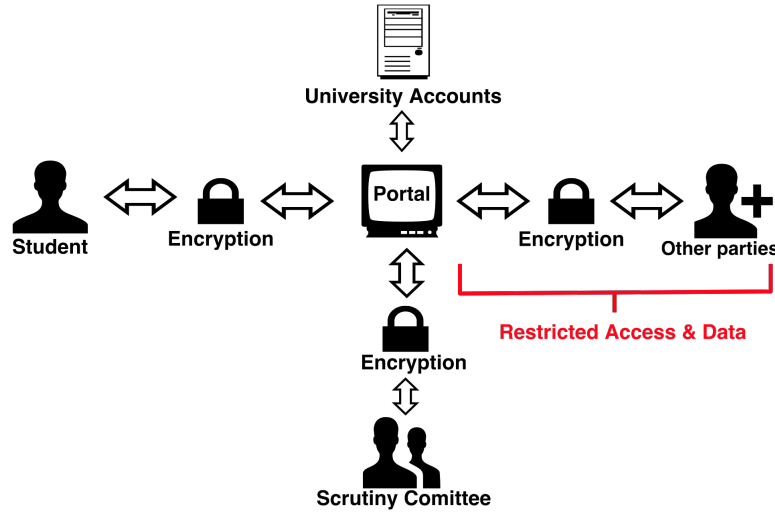
Figure 3.1:  Portal and users.

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 sanitised version of the students condition.

If the system can complete all the above steps and keep the security intact, then we will have been able to achieve the requirements for this project successfully. In order to evaluate the project, each of the requirements will be analysed with the implementation and if it was fulfilled successfully.

## 3.3   Interface

The interface will consist of three different types of users; The Students, the Secretary 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 university accounts making it easy to access from within the portal. However, this may turn out to be difficult as accessing the university Intranet would be a security risk to the University.

### 3.3.2 Secretary Panel

This panel will be a little more detailed than the student panel. The panel would notify the secretary by email when a new form is received. The secretary can then process the form and take it to the next scrutiny panel meeting. The panel would also allow searching and filtering of all the forms submitted by all students. The scrutiny committee will have access to the system as well which allows all users in the meeting to view the form and decide its fate. Once the scrutiny committee has decided, the secretary will be able to change the status of the project to either *Rejected* or *Approved*. Alternately, if the scrutiny committee decides they would like more proof, the secretary would be able to request these from the student directly from the panel. As exporting data was one of the requirements, the interface should provide the secretary and the scrutiny panels a button through which they can print and save the form as a PDF document.

### 3.3.3 Scrutiny Panel

This would be a read-only portal where the members of the scrutiny committee can access and read all the data and submit their concerns to the secretary under their minutes for the meeting. This follows the same procedure they already have in place where the secretary would be the one in contact with the student. Figure 3.2 shows an example of the scrutiny panel's read-only access
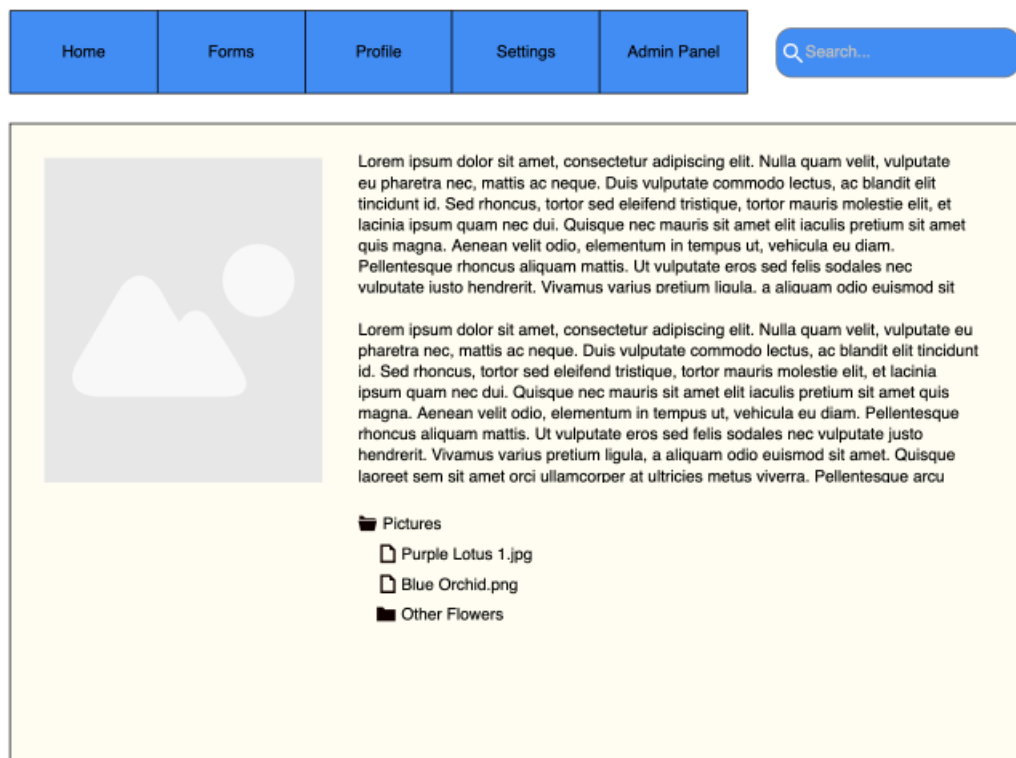


Figure 3.2: Portal mock-up

## 3.4   Testing

Testing is an essential aspect of functionality and knowing if the system follows the requirements. In this system, we can carry out testing in various 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.

### 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.

### 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.

### Automated Testing

Django allows writing of test code (unittest)[23] which then runs as a normal user(in our case) and checks if the response matches our needs. 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 being stored in the database.

### 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 significant and so different types of testing will be important.

# 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 when developing. 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*[24] Bootstrap 4[25] HTML template[26] 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 more. Table 4.1 shows the relevant panel and its pages.

| Panel | Pages |
|---|---|
| All Panels | 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 |
| Student | 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 |
| Secretary | 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 |
| Scrutiny | Dashboard page with all forms access |
| | View form page to view data from the form |

Table 4.1: Development panels & their pages

## 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. The system will use JavaScript in numerous ways. Most of the code will be from readily available packages while a small portion will be coded based on the requirements. *Bootstrap 4*[25] will allow the HTML to be accessible on mobile devices as well as make the HTML elements look smoother. Along with mobile first coding, the HTML tables will need a sorting and searching script which will be provided by *DataTables*[27]. One of the requirements is to be able to download and save the data from the forms, in order to do so, the HTML page must be converted to PDF and the use of *jsPDF*[28] along with *html2canvas*[29] will be beneficial. Each of the scripts mentioned above will be further explained under Chapter 5.2 with detail.

## 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 final schema currently implemented. This schema was updated twice based on Django migration logs. All-Auth[30] automatically generated *Authuser* and the rest of the tables are designed specifically to the requirements. All the fields were designed before implementation except for *userlink* in the *userprofile* table and *moreproof* in the *ecfdata* table. These were added during implementation to allow the students to have a unique public data and to allow the additional proof to be uploaded with the use of notifications respectively. The tables are interlinked between each other; The *user* table has a one to many relationship with *ecfdata* and a one to one relation with *userprofile* and *publicecf*. Along with the user table, the *ecfdata* table is linked with a one to many relationship to both *moduledata* and *files*. These simple relationships make it easy to input data into them without having issues when running queries.
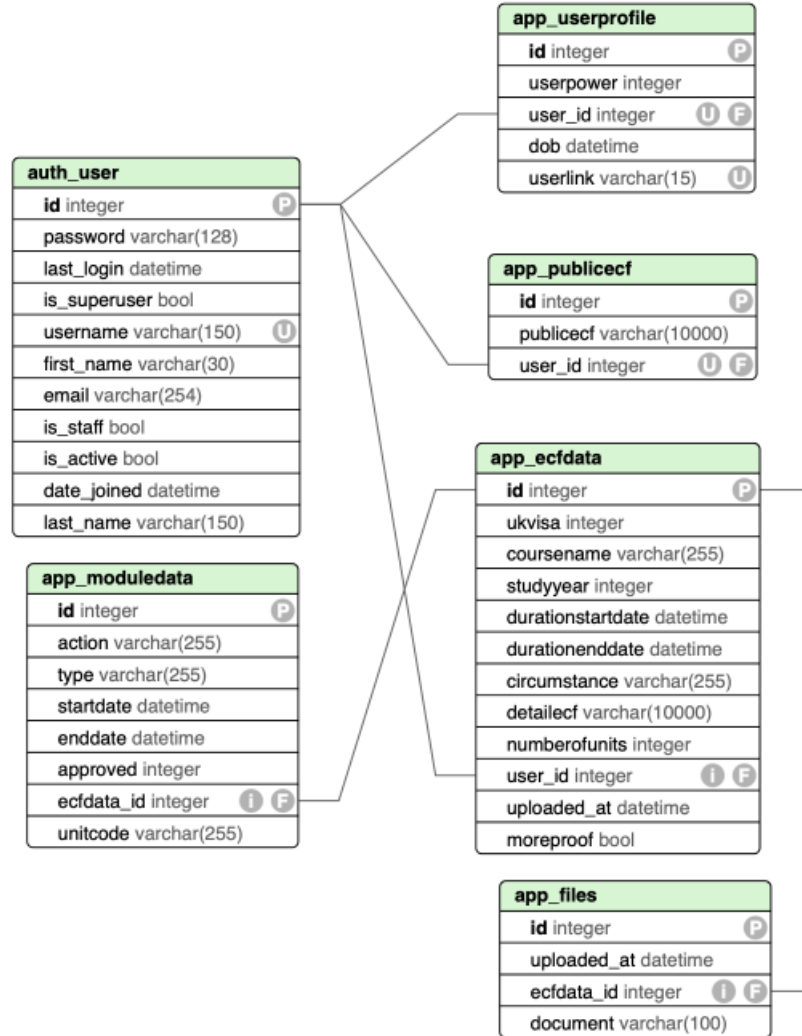
Figure 4.1: Database Schema

## 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, All-Auth[30] which is a precoded authentication package allowing the use of external accounts such as Google, Dropbox, Github, LinkedIn and many others will be used to allow logging in directly into our system. All-Auth uses sessions to authenticate users into the system and prevent unauthorised logins. It has a *login_required* function which can be used with individual pages to prevent unauthorised accesses without logging in.

Bcrypt[31] 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[32].

## 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. The implementation covers the requirements as well as additional features that could be useful.

## 5.2 Implementation

The implementation process began with re-constructing the HTML template by Creative Tim[26] to the different sections mentioned in chapter 4.2, JavaScript going along with the HTML and finally the Django backend.

### 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 Console, the debugging was completed and successful HTML pages were created for all three different sections as mentioned in Chapter 4. However, since pages are going to be repeated(Dashboard, Viewform, Profile, and others), only one HTML code was required and repeated depending on which panel you have logged in as. Table 5.1 shows what components were created into their related files by redesigning the public licensed template. Each of these was combined with CSS and JavaScript from Chapter 4. Figures 5.1 and 5.2 show how the template was redesigned to fit our requirements.
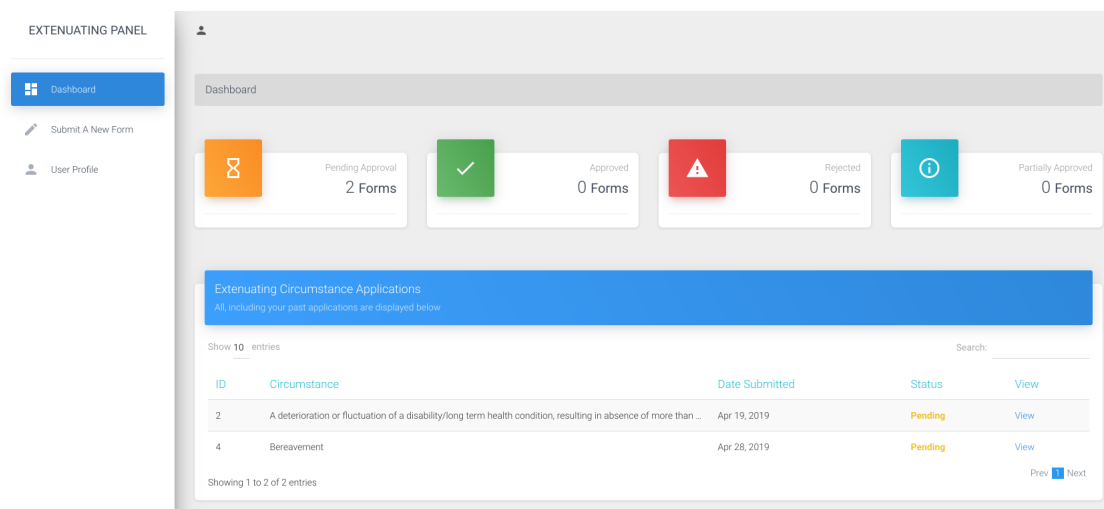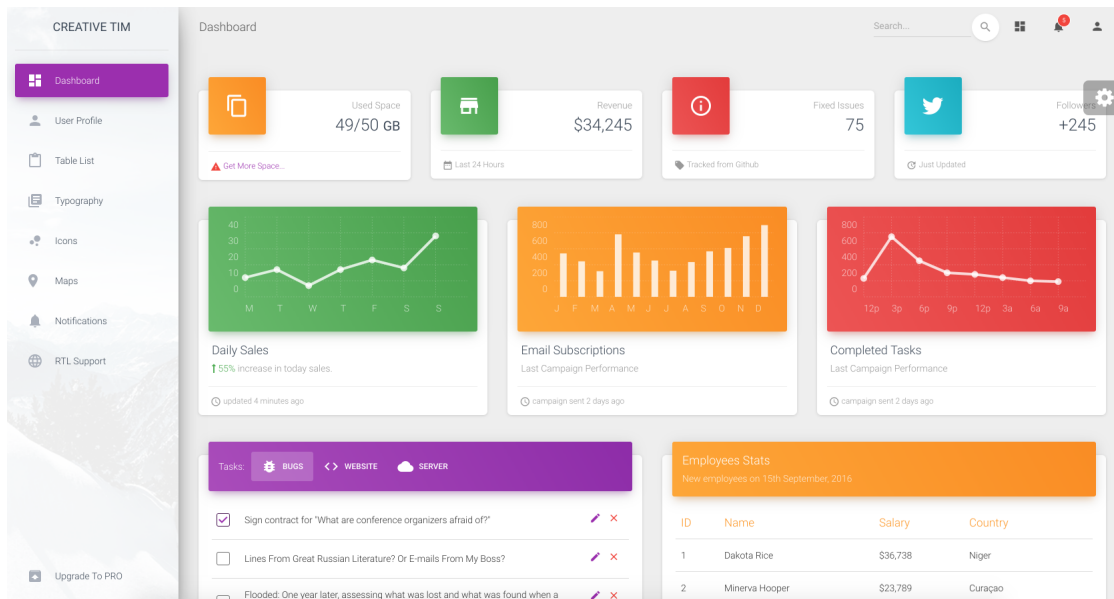


Figure 5.1: Redesigned Dashboard

Figure 5.2: Creative Tim Dashboard

| 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. Alongside the JavaScript received with the template by Creative Tim[26] such as Bootstrap[25] 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[27] 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
Being able to sort and filter through the tables was an important requirement for organised data.

Another great public script used was *jsPDF*[28], 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 lose its colours and format. This would scramble up all the data and make it highly unreadable. To overcome this bug, *Html2Canvas* was used.

*Html2Canvas*[29] is an 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 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 can 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.
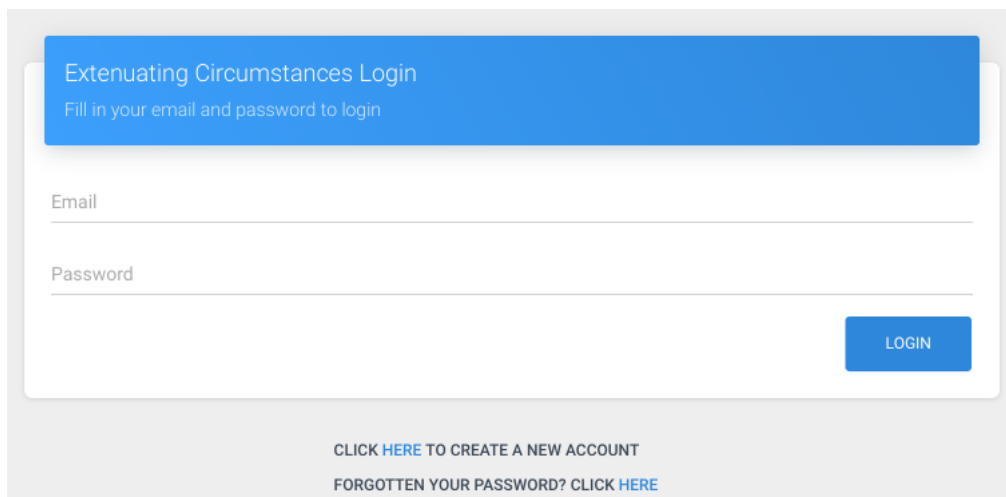
### 5.2.3   Django

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[33] which is the latest version was used. The first step was to combine the HTML, JavaScript, CSS and Django 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[30] 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 an SQLite database which was inbuilt with Django. Creating the database schema, as built-in Chapter 4 seen in 4.1 was straightforward with the Model, View and Controller system which Django follows. The database was ready in seconds after defining it in Models. Figure 5.3 and 5.4 show the login and register pages respectively with All-Auth and Django being the back-end controllers.



Figure 5.3:  Login Page

Figure 5.4:  Register Page

**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 a new form" page had to be completed with the back-end code. Different students may have a 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 an HTML POST form is made extremely easy with Django's *forms.py* system where defining the fields of a form such as text input or email automatically present the form on the template where it is called. When the POST data is received, Django runs a *cleaneddata* test which checks for any code in the data received which would be harmful to the system. 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. Figures 5.5 and 5.6 show the process of creating a new form with the modules from 5.5 used as a GET request in 5.6

21

Figure 5.5: Number of modules



Figure 5.6: Submit a new form

**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 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. Figure 5.7 shows the dashboard created including statistics for the students.

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 precisely how many pending forms are there which need to be updated. Figure 5.8 shows the dashboard created excluding statistics for the students.
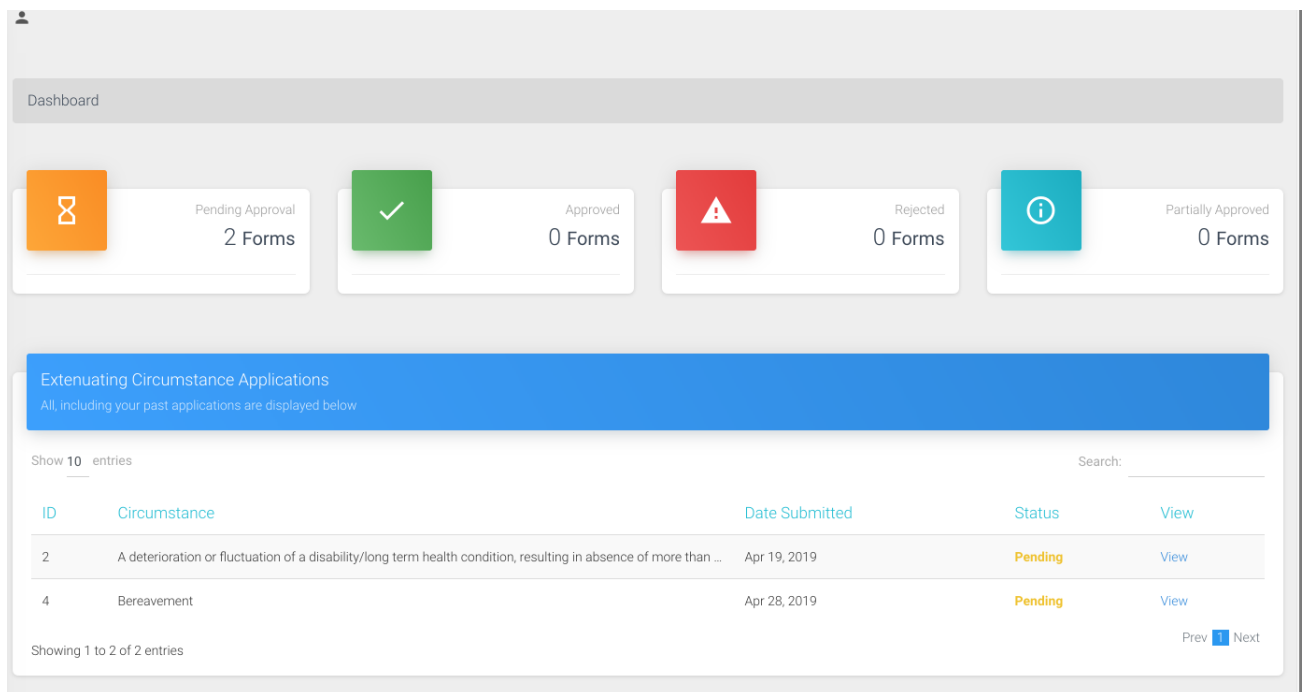


Figure 5.7:  Student Panel Dashboard

Figure 5.8:  Secretary Panel Dashboard

**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 relationships. Apart from simply displaying the form data already submitted, there is an option triggered by the secretary which creates a new *file uploader* 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 except for allowing the secretary to 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 student's 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. Figures 5.9 and 5.10 shows PART of the view form page with and the popup which shows up when the edit button is clicked respectively.

Scrutiny Panel: Since the scrutiny panel does not control the status directly on the portal, the view form page retrieves data from the database and displayed it with the use of HTML components.



Figure 5.9:  PART of the Secretary Panel View Form

Figure 5.10:  Secretary Panel Edit Module Modal

**Profile Page**

The profile page displays the current user's 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 currently in the database and allows changing them after cleaning the new data. *This includes: firstname; lastname; dob only.* Figure 5.11 shows the student panel profile page with randomly generated text as the public data.



Figure 5.11:  Student Panel Profile Page

**Student Record**

Every user when registered is provided with a *eight 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 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. Figure 5.12 shows how the private link can be accessed and the data it would show. Since it is a *textarea* HTML component, the data can be copied into a TXT when needed.

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.
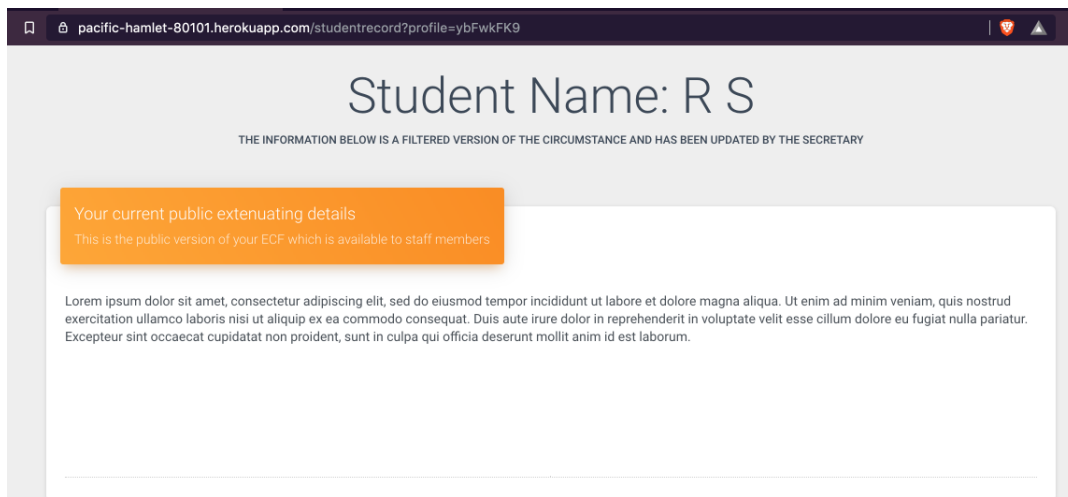


Figure 5.12:  Student Record Via Secret Link

**Email Notifications**

Email notification was 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; Updated status of the form (Rejected/Approved); Requested additional files.

# 5.3 Complications

Different types of complications showed up while compiling the code. The most difficult ones are described below under two sections; Students and Secretary. The Scrutiny Panel did not have as many severe complications as most of the code was reused from the other two.

## 5.3.1 Student Panel

### Dashboard

Issue: Calculate based on forms and not each module within the form.
Solution: 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

Issue: Create module input fields based on the GET parameter value and allow each of the modules to be stored into the database.
Solution: 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 example, unitcode1

Issue: Allow only ZIP files to be added and validated.
Solution: Create a Django validations.py function which detects the format of the file and rejects if it is not a ZIP.

Issue: Store files with the same name without replacing them.
Solution: Rename the uploaded file by adding a five digit random string to the end to prevent files being replaced when stored.

### View Form

Issue: Prevent other users from accessing current users form.
Solution: Create validations which check if the form belongs to the currently logged user, if so, then the user is allowed to access it.

## 5.3.2 Secretary Panel

### Dashboard

Issue: Calculate based on forms and not each module within the form.
Solution: 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**

Issue: Modal popup to edit form details would load on a different page instead of current one.
Solution: 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.

Issue: Multiple POST forms would bring about issues when working on views.py as the controller.
Solution: Provide each POST request with a unique 'name' attribute to differentiate which part of the views.py it should access using IF statements.

Issue: Switch-case Toggle HTML would not change the switch based on the current value in the database.
Solution: Added a JavaScript code to manually get the value from the database and switch the toggle CSS.

## 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 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[34] 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 critical as common Remote Administration Tools can access browser cookies and duplicate them.

*Cross Site Scripting(XSS)* had its limitations when testing, for example, if a CSS Style were 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 the console. The result was verified using the 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 loopholes 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 have played a key role to allow changes to be made for maximum security with easy access and usage.

# Chapter 6. Evaluation

## 6.1 Introduction

Chapter 3 discussed the requirements as well as the process of building a system which would allow easy and secure usage of the Extenuating Circumstances for both Students as well as the Scrutiny Panel who process their minutes through the Secretary. The results have been evaluated in this section with relevant discussions and possible future work as well as other possible uses for such a system.

## 6.2 Results

The purpose of the project was to produce a system which will allow the students, secretary and the scrutiny panel to efficiently and securely store and access Extenuating Circumstance Forms as well as allow other staff members to access the sanitised version of the student's circumstance. Django provided a simple and secure platform with the ability to access the system on any device unlike the alternates of Smart Applications and Blockchain. Smart applications would not be accessible on all devices and Blockchain would not be able to keep the data extremely confidential. Each of the following describes the requirement as well as the results achieved and why the use of Django was the right choice.

### Organised Data

One of the issues with the previous system was the processing of data. Data would need to be duplicated which lead to unorganised data records. Students would submit multiple forms at times with either a different circumstance or the same one which would again lead to improper data storage. With the Django system, data is well organised and easily accessible. Data is stored in tables with the ability to search for specific records and view past submissions as well. Figure 6.1 and 6.2 show extenuating circumstance forms for both Students as well as what the Secretary and Scrutiny Panel would see. The status of the form allows all stakeholders to remain informed and students avoid multiple forms filled with the same data. The search field allows sorting and filtering through the forms extremely easily when searching for students, specific forms or status search. This solves the problem of data being unorganised.

Figure 6.1:   Student Panel Submitted Forms



Figure 6.2:   Secretary/Scrutiny Panel Submitted Forms

## Additional Files

In situations where there secretary and scrutiny panel decide they need to get more proof and data from the student, the secretary can request files via the panel; automatically sending an E-mail notification to the student requesting them to submit more proof. The student can then access the form on the panel and upload the files. On successful upload another E-mail is sent to to the Secretary confirming that the student has uploaded more proof and should be taken to the next Scrutiny Panel meeting. This provides the student, and the secretary simplicity as any new proof is directly linked to the form. With a web-based platform like Django, using IF statements made such tasks straightforward to code with. Figures 6.3 and 6.4 show the section of additional files under the student panel.
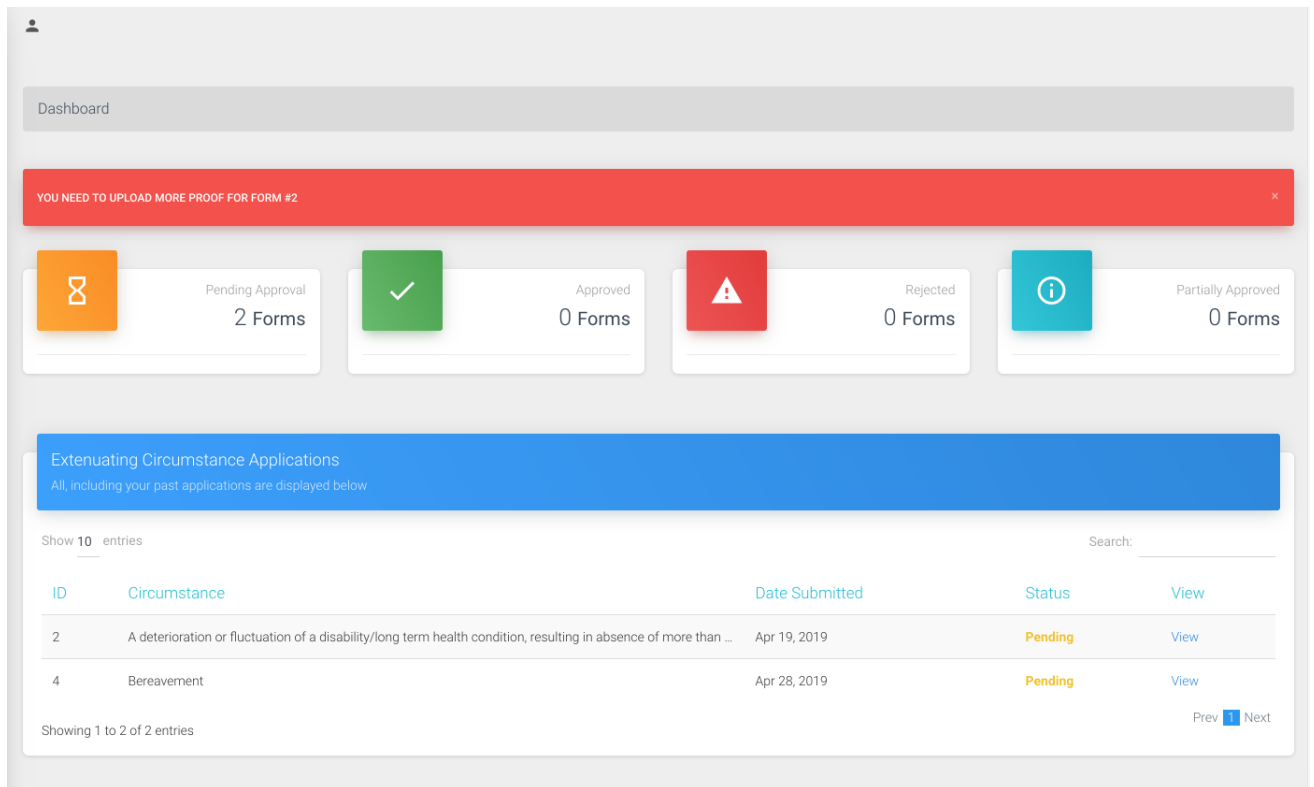
Figure 6.3:   Dashboard notification for more files
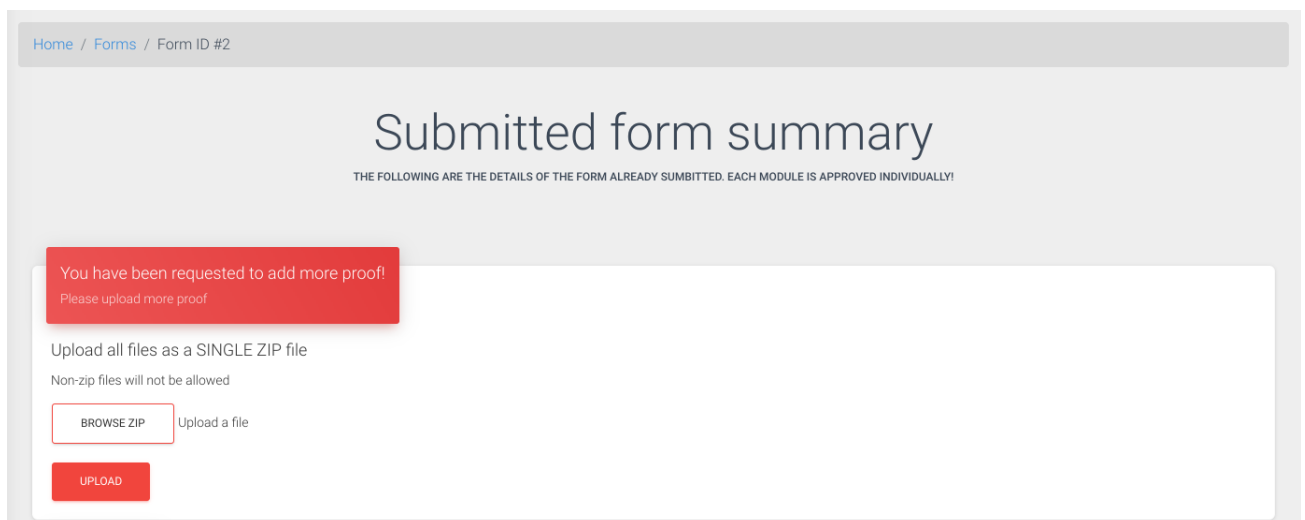


Figure 6.4:   View Form upload files

## Print, Download & Student Record

Being able to share the data to other third parties was an important requirement
as well as being able to get it on paper. With the use of JavaScript, printing and
saving as a PDF document was implemented successfully with the output being
the same as what a user would view on the website as explained in Chapter 5.2.

Figure 6.5 shows two simple buttons which have the JavaScript code behind it to print or download the current page in the same format it is viewed.
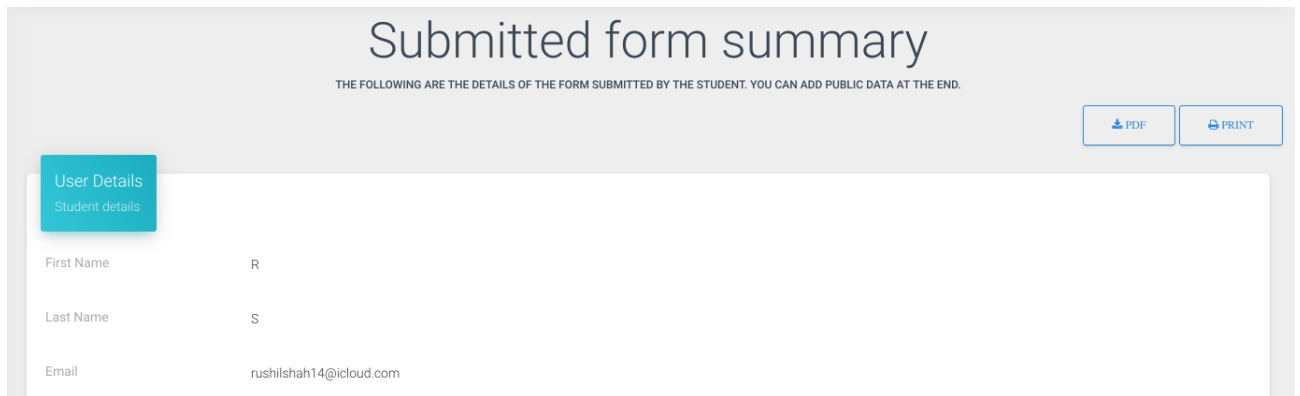


Figure 6.5: Print and Download Files

Another requirement was being able to share the student's record where the sanitised version of data is stored to the student's portfolio. The current implementation uses a unique link to each students profile which can be added to the student's portfolio in order to share the data. Only staff members will have access to the student portfolio hence access to the records. Figure 6.6 shows the secretary panel with the ability to copy the link or re-generate it to share with others. This regeneration process became extremely easy as Django uses *Python* which has a *random* package as mentioned under Chapter 5.2 allowing the link to follow a random 8 digit format.
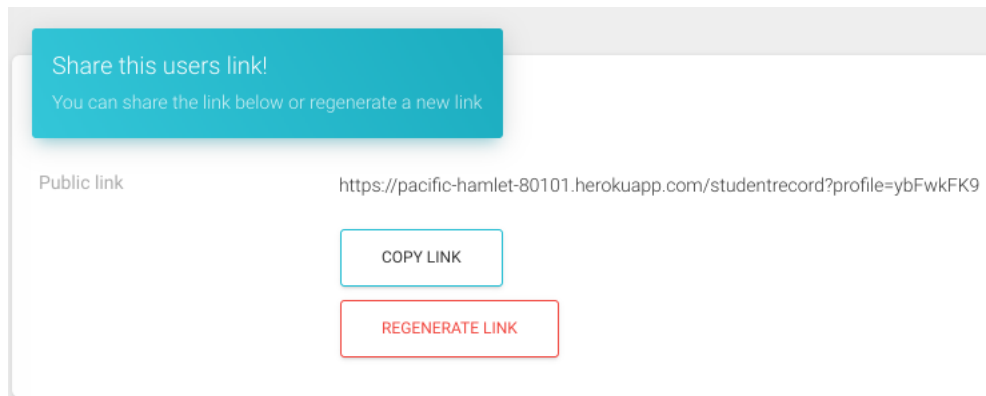


Figure 6.6: Student Record link

## Security

Security was one of the most important requirements for this system, and with Django being the back-end controller it made the system secure. The University's login system would only work if the web-based system is installed on its Intranet; the system currently uses another authentication system. All-Auth[30] provides validation for both logging in and sign up. It uses *Bcrypt*[31] encryption to

securely store the password into the database. If the password is not complex enough, All-Auth would automatically bring up an error and ask the user to create a stronger password. The database itself is password protected when storing data, for example, *Heroku* uses a *25 digit complex password* to secure the database itself which prevents brute-force attacks onto the database directly. The current implementation does not encrypt every single field in the database, this is in order to avoid unmaintained external code, but with the use of *django-pgcrypto*[17] as a prototype, each field can be encrypted making the data even more secure than it already is.

Apart from the database security, it was important to make sure other users who can log in to the system such as Students cannot access data which is specifically meant only for the Secretary or the Scrutiny Panel. This was successfully implemented with the use of FOR loops and IF statements as described in Chapter 5.2. Django also has a controller for validation where such a function can be defined to check if the user has permission to access the pages. Other systems, for example, *Ruby on Rails*, would require manual and extensive code to ensure that the link cannot be accessed.

The student record page (Sanitised version of the student's circumstances) which is accessible to staff members would need to be behind *University of Sheffield's Intranet* in order to authenticate if the user is a staff member or not. Even without staff authentication, the student records page can only be accessed by having the student's key as explained under Chapter 5.2. A Key system would prevent unauthorised access to the student's records as the link would only be accessible in the student's portfolio. In cases where the link has been shared numerous times, it can be regenerated by the secretary to prevent access on the old link. With access to the University's login authentication, student record would be entirely secure. However, the current system does prevent unauthorised access almost as good as the specific authentication

Other security tests were also completed, some provided by Django's inbuilt tests while others required manual testing. The security requirements were satisfied based on the tests ran such as *Raw SQL Queries, Cross Site Request Forgery and Unittests*. However, not all code has been tested due to lack of expertise on site penetration. Experienced coders could find bugs in the code which can later be exploited if the current system is unmaintained with Django's new security updates. This issue can occur on any other system, web-based or even smart applications. Without keeping the code up to date with the latest developments in the respective language, penetration is highly possible.

Lastly, uploaded files are stored directly onto the hosting server. In order to secure the files, the use of *SSH-Keys* to access the hosting server would provide increased security. Even with other web-based or smart applications, the use of *SSH-Keys* is crucial as all data would need to be hosted somewhere on the cloud. Another approach, currently not implemented in the system, is to individually encrypt each file as it is stored onto the system. Two stable, but not maintained, Django

encryption projects can be used. One creates a transparent layer of encryption above the files[35] while the other encrypts the files based on the form input in Django[36]. Both provide the right level of encryption which would make all the files extraordinarily secure and confidential. However, as they are not maintained it would be a security risk to implement them onto the current system.

## Mobile First

Since the website is coded in Bootstrap 4[25] the system follows a mobile-first coding approach which allows the website to be easily accessed on any mobile devices (Smartphones, tablets or laptops) with full functionality. Students can apply for extenuating circumstances at the comfort of their home in cases where their circumstance does not allow them to move out to submit a printed form to the office. Figure 6.7 shows the system running on a *Heroku* Server and accessed on a mobile device as a Student trying to create a new Extenuating Circumstance Application. All panels including the Secretary Panel are mobile friendly and so can be accessed with the smallest smartphone. This is where Django turns out to be better than a straightforward smart application as it requires a smart-device, but with Django, access is possible from any browser.



Figure 6.7:  Screenshot of the system running on a mobile device

**Email Notifications (Additional Feature)**

Communication between the system and the users is an important feature to have. The implementation uses *Django's mailer system* to send E-mails to the students as well as the secretary on successful actions such as *new form submitted*. Figure 6.4 showed the upload form HTML on the student panel, but in order to get the attention of the student towards it, an automated E-mail is sent asking the user to submit the additional files. This is shown in Figure 6.8 and 6.9
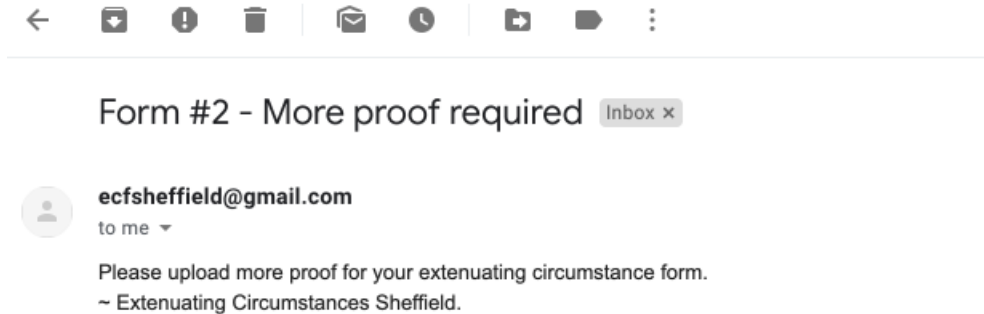


Figure 6.8: Screenshot of E-Mail received for additional files by student.
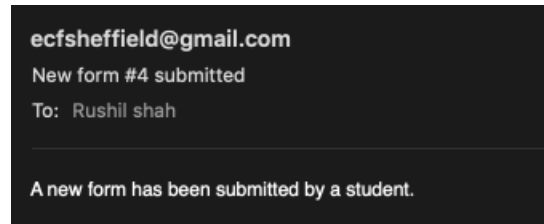


Figure 6.9: Screenshot of E-Mail received by secretary.

## 6.3 Discussion

Looking at the project from the very beginning, creating a web-based system over a smart application or block-chain was a good decision because everything that a smart application can achieve has been achieved with the web-based system and more such as a mobile first code and accessibility. Block-chain, on the other hand, would have been a good base for development but would not be highly maintainable or confidential even though it would be secure and would prevent data losses. Ruby on Rails or Laravel were also good options to use when producing this system as they would produce a similar system with a few distinct conditions. It has been easier to use Django compared to Ruby on Rails, based on past experience, as it allows a much easy platform to code with. Ruby on Rails has *GEMS* that can be used, but Django allows usage of packages created by other users who publish their code under the MIT or Public licenses. An example of such a package would be All-Auth which is highly used and extremely secure due to continuous maintenance of the code.

Personally, it has been a great learning experience with Django. Even though having used various other web-based coding platforms such as NodeJS, Ruby on Rails, a little PHP; Django has been the easiest to work with not only because it follows a simple model, view and controller system but because it allowed creating database schemas just as easily as creating an input form in HTML. The same database schemas can then be used as forms to receive data from the website. This makes production extremely easy and quick compared to other systems where each attribute needs to be defined. Django also provided an easy way to debug the code without having to figure out exactly what went wrong in a huge pile of code.

NodeJS would have been a good alternate option as it would allow creating a progressive web application with the ability to work offline. This would behave like a smartphone application. However, the security of such a system would be a high concern as using a lot of JavaScript can allow Cross-Site Scripting attacks and unlike Django, it would be difficult to protect against such attacks.

## 6.4 Future Work

The current system can be the base for an entirely secure and efficient future system. The following is a list of possible implementations which can be completed by further studies. Each states the difficulty of producing it as well as if the code would be maintainable.

Use of Django file encryption projects[36][35] to create a custom encrypted file upload system which stores the uploaded files with a *SHA256 encryption* onto the hosting servers folder system. Both projects are not maintained and hence would require re-coding the whole encryption system by using one of them as the base and keeping it up to date at all times due to security. Producing something like this would require encryption expertise as well as time and research. It would be highly maintainable as 40 security flaws are detected each day based on a 2017 survey[37]. File encryption is important because in case an unauthorised user gains access to the hosting server of the system, they can access the form files directly and view all the data.

Django-pgcrypto[17] can be implemented onto the *PostgreSQL* database server in order to encrypt every single entry into the database which will protect against any unauthorised access directly into the database. This system would work with the current *forms.py* and encrypt data as it is entered into the database. Again in a scenario when an unauthorised user gains access to the database directly, in order to view the data inside, they would need to decrypt each of the fields making it extremely difficult to crack and hence making the system secure. Django-pgcrypto[17] can be used as the base of creating an encryption system specific to extenuating circumstances making it highly difficult and maintainable as you would need to keep the encryption up to date.

Code to allow the secretary and student to communicate. A messaging system as

such would allow both the secretary and the student to converse within the form. Producing such would require knowledge on *WebRTC* and would not need to be highly maintained, once the code has been placed it would not require much attention unless a security bug is detected. A messaging system was not part of the requirements and therefore was left out but as a few students recommended it would be highly useful.

A text area within the form accessible by the secretary and the scrutiny panel on writing down the minutes of the meeting. This would be very useful as the minutes can be referred to directly when accessing the form. However, these would be hidden from everyone else including the staff members and the student. Producing this would require an editor inbuilt with the *textarea* box and the need to *autoescape* the data filled in. Once the code is developed, it would not require much maintenance as long as the code is secure.

## Alternate uses for such a system

This project has been focused on security, ease of access and the ability to get data off users and accessed by other users. A system like this can be edited into various other branches which involve interaction between two groups of users who require confidentiality and security. Below is a list of possible uses and scenarios where such a system can be highly useful if edited to the needs.

Survey System - Gathering information with questions being asked in place of the extenuating circumstance form and assessed by the supervisors. Such a system can be edited to fill the form and submit without having the user to log in or provide personal details. Statistics can be added including graphs to calculate the individual results with the use of *Python* and *NPM*.

Grading System - Users can upload files and data similarly to the current Extenuating Circumstance Form, and a superior user can assess it and graded accordingly. To achieve this, the system would need to duplicate the status system(Pending, Approved, Rejected) and allow it to grade based on integers for example; *80/100*. It can be used in all kinds of grading scenarios such as education, work even personal data storage with grades. For example, a student can decide to store all their assignment on the system as well as the grade they have achieved.

Medical Appointments & Meetings - The system can be adjusted to allow patients to fill out medical forms directly onto the system which are then assessed and viewed by superiors. A payment system can also be directly added onto the system which allows the patient to have records of their payment and each medical form they have filled or to record every meeting occurred between the two. With further implementations, each meeting can have its minutes recorded onto the system.

These are just a few examples of where such a system can be useful. With more

encryption as mentioned under Chapter 6.4 this system can be developed into very many different branches of data retrieval.

## 6.5   Summary

This section has mainly connected requirements and implementation for this project. The main requirements have been satisfied but with additional work, the system can be made extremely secure and very useful not only for what it is built to achieve but for different projects which follow a similar requirements list.

# Chapter 7. Conclusion

The overall goal of the project was to successfully create a system which allows the students to submit extenuating circumstance forms and the secretary/scrutiny committee to have access to it without having to go through the troubles of multiple paper forms creating confusion. Being that the circumstances collect personal data, security was a big concern to digitisation. Throughout the documentation from Chapter 2 until Chapter 6, the requirements have been the sole purpose of decisions made. Django, even though it is a web-based framework, provided us with high-end security control without having to write down huge amounts of code for data safety. The system also provides flexibility such that it can use both *SQLite* and *PostgreSQL* databases to store the data, with the database being password protected.

Secure authentication is present in the current system; if the system is hosted under the University's Intranet, it would have an extra authentication forcing only access by students and staff. This would be extremely useful for the Student Records page which contains a sanitised version of the circumstances, however, even without the extra authentication a secret key system would prevent unauthorised access as described under Chapter 6.

With future work on this system, it can be made extremely secure not only for University of Sheffield's extenuating circumstances but also for other systems which require communication between users and submission of forms as discussed in Chapter 6.4.

To conclude, developing this system has been an interesting journey, one of the key aspects was understanding a completely new system and its abilities; how similar yet different technologies such as Django, Laravel and Ruby on Rails can provide different back-end results yet produce the same front-end system as described under Chapter 2. Bundling up all the requirements from (Chapter 3) with the research (Chapter 2) and the initial design (Chapter 4) of the system, the final implementation has come close to being entirely completed. With the addition of two encryptions(Chapter 6.4), one for each entry in the database and second for each file uploaded, this system can be complete while fulfilling all the requirements as well as adding additional features which serve to be useful. Deploying the code on *Heroku* is proof that the implemented system is fully functional.

# 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, "What is the difference between ar and vr? a lesson in altered realities." `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] M. J. Garbade, "Vulnerabilities reached a historic peak in 2017." `https://hackernoon.com/5-best-programming-languages-to-learn-for-cyber-security-be97071919f9`, 2018. Accessed: 10.04.2019.

[14] Martin, "Top programming languages used in web development." `https://www.cleverism.com/programming-languages-web-development/`, 2015. Accessed: 20.04.2019.

[15] EDUCBA, "Uses of django." `https://www.educba.com/uses-of-django/`, 2015. Accessed: 20.04.2019.

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

[17] D. Watson, "django-pgcrypto." `https://django-pgcrypto.readthedocs.io/en/latest/`, 2014. Accessed: 10.04.2019.

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

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

[20] NodeJs, "Node.js® is a javascript runtime built on chrome's v8 javascript engine.." `https://nodejs.org/en/`, 2019. Accessed: 20.04.2019.

[21] DailyRazor, "Django vs node.js – here's a 3 minutes detailed comparison analysis." `https://www.dailyrazor.com/blog/django-vs-node.js/`, 2018. Accessed: 20.04.2019.

[22] A. Holovaty and J. Kaplan-Moss, "The definitive guide to django: Web development done right," 2007.

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

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

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

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

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

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

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

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

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

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

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

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

[35] D. Quinn, "django-encrypted-filefield." `https://github.com/danielquinn/django-encrypted-filefield`, 2016. Accessed: 10.04.2019.

[36] Ruddra, "django-encrypt-file." `https://github.com/ruddra/django-encrypt-file`, 2018. Accessed: 10.04.2019.

[37] M. Ángel Mendoza, "Vulnerabilities reached a historic peak in 2017." `https://www.welivesecurity.com/2018/02/05/vulnerabilities-reached-historic-peak-2017/`, 2018. Accessed: 10.04.2019.