National College of Ireland

BSc (Honours) in Computing

Cybersecurity Specialisation

2020/2021

Josh O'Regan

X17521289

X17521289@student.ncirl.ie

Secure Property Management (SPM)

Technical Report

Contents

Executive Summary

This report covers the background, requirements, and implementation of the Secure Property Management application. The main catalyst for such an idea was my experience working closely with property managers during my internship throughout the Covid-19 pandemic. As I got to know them, I also got to know the way in which they work. Through consultation with them I was able to elicit requirements for such an application which is undoubtedly needed in the current remote focused climate of today. In summary the application allows for a landlord to add properties to a portfolio, from here they can manage aspects of the property securely. The application maintains strict access control and only presents data associated with the user back to the user. Documentation associated with each property can be uploaded, the system then encrypts and stores the document accordingly, with a corresponding database entry. The system follows a defence in depth strategy to ensure all information remains secure.

The following report covers how these main aims were achieved, what technologies were used to achieve them, the reasoning behind implementing them and how I tested them. also included code screenshots, diagrams and elements of the UI to highlight the innovative and unique nature of this project and the host of security features it provides.

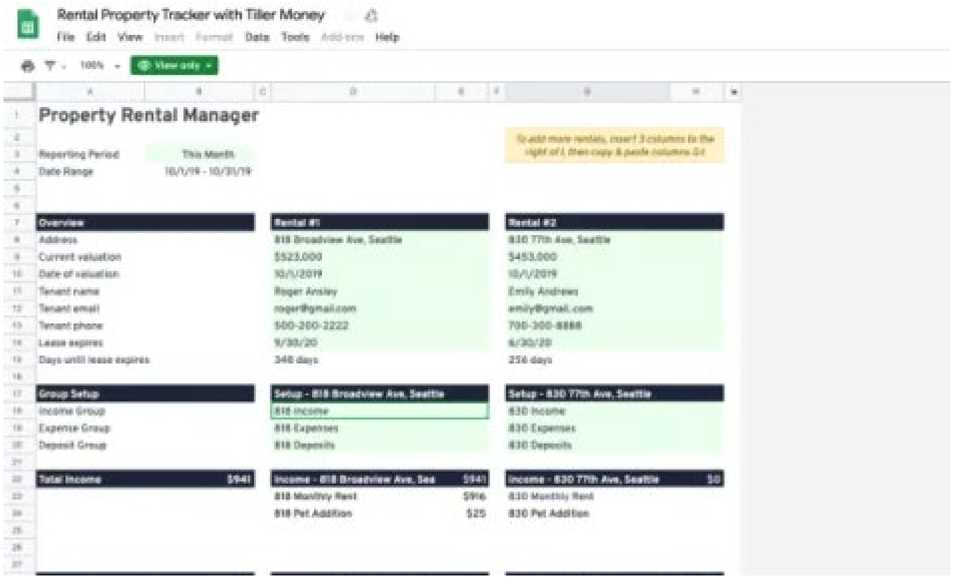
1.0 Introduction

1.1. Background

The Covid-19 pandemic has forced businesses to adapt and change in an unprecedented environment. Through my experience in my internship, I noticed that some businesses were more prepared than others. One sector I spotted a major flaw in regard to the changeover from in person to remote working was the property sector, more specifically property management. Onboarding new renters became a huge challenge for many companies due to government restrictions and the issue of safety. Remotely, the challenges of security became paramount. Getting onboard with a new method of completely onboarding new renters remotely became a real issue as there was no readily available solution to do so. Documentation can contain extremely sensitive information meaning the use of email and other such services was not ideal.

While working on helping these businesses gain some level of remote work I came up with the concept for my project. I listened to the needs and wants of the stakeholders in this sector and as such was able to gain a great understanding as to what technology exists out there already and its limitations. Using this I gained an all-encompassing view into the background of what is in demand from the industry but also the background of what technologies are currently in use and how they can be integrated into my solution.

Fig. 1: Typical Existing System (Spreadsheet)



Rental Property Tracker with Tiller Money
File Edit View Insert Format Data Tools Add-ons Help
100%
View only

Property Rental Manager

Reporting Period	This Month				
Date Range	10/1/19 - 10/31/19			To add more rentals, insert 3 columns to the right of I, then copy & paste columns G-I	
Overview		**Rental #1**		**Rental #2**	
Address		818 Broadview Ave, Seattle		830 77th Ave, Seattle	
Current valuation		$523,000		$453,000	
Date of valuation		10/1/2019		10/1/2019	
Tenant name		Roger Ansley		Emily Andrews	
Tenant email		roger@gmail.com		emily@gmail.com	
Tenant phone		500-200-2222		700-300-8888	
Lease expires		9/30/20		6/30/20	
Days until lease expires		348 days		256 days	
Group Setup		**Setup - 818 Broadview Ave, Seattle**		**Setup - 830 77th Ave, Seattle**	
Income Group		818 Income		830 Income	
Expense Group		818 Expenses		830 Expenses	
Deposit Group		818 Deposits		830 Deposits	
Total Income	**$941**	**Income - 818 Broadview Ave, Sea**	**$941**	**Income - 830 77th Ave, Seattle**	**$0**
		818 Monthly Rent	$916	830 Monthly Rent	
		818 Pet Addition	$25	830 Pet Addition	

Technology background

When initially considering all the options for developing the application, I thought about the technoligies I was most familiar with, Android Studio, React and Java to name a few. These however would not present a huge amount of difficulty or challenge and would therefore be a bit too simple. I resolved to try a completely new framework that I had never used before, Django. not solely because of the difficulty but rather because it had the capabilities to fit all of the defined requirements while also being extremely relevent and modern (the newest realease being the 6th of April). Django itself is a very Python heavy framework which meant a large amount of research had to be conducted into the language, this proved challanging as it was the first experience I had with the Python language and consequently Django. To this end I used online resources such as Udemy to educate myself on Python and how to develop applications using it. From here I read up on Django doccumentation and discovered how the framework uses Python to create a web application. By then implimething Bootstrap as a lightweight, responsive frontend I was able to estabish a solid base for the application.

The backend needed a large ephesis on security and needed to include a secure way to store user details. For this purpose I went with an SQLlite database. I went with this instead of using Firebase or any other storage/authentication manager I had used in the past as it presented unique oppertunities to try out industry standard hashing for passwords along with the ability to enforce the most recent password and authetnication standards set forth by OWASP*(OWASP, 2021)*

1.2. Aims

This project aims to bridge the gap created between property managers and renters created by the unprecedented circumstances we find ourselves in. Through this project, I aim to create a convenient and most importantly secure solution for onboarding and managing tenants completely remotely. I will meet the needs of convenience by ensuring the web application has a high level of availability. The application will allow for the secure upload of documents, These documents are then encrypted using a user defined password. Once this has been done, the documents will be categorized, and stored within the relevant property entry in the database.

In order to provide an all-encompassing system, there will not only be a focus on onboarding new tenants but also managing existing tenants. with an emphasis on ease of use. With this aim in mind a very strict scope for the project has been established in order not to overcomplicate the process.

Security is one of if not the main aims of this project. There has to be a trust established from both types of users, the tenant and the property manager. The documents being transferred could be tenancy agreements, contracts, details of employment and bank details. Due to the sensitivity and the importance of the documentation and information being transferred, security must be the foremost and most paramount aim. This means that the backend of the application must be robust and entirely secure. This is accomplished by encrypting the documents and ensuring any access to said documents is only made by accounts created along clearly defined strong password requirements. Defence in depth will be followed to meet an commercial standard of data security.

At the highest level, the main aims are to allow the onboarding of tenants remotely. To allow all relevant documentation to be categorized, encrypted and stored automatically when uploaded by a user. The property manager will also have a dashboard from which they can manage properties and their tenants.

1.3. Technology

<u>The following software is used in this project:</u>

- Bootstrap – I used Bootstrap as the frontend development language, It allows for the web application to be easily translated to a mobile or desktop counterpart.
- Visual Studio Code – The main development platform I am using for this project, allows for integrations and debugging.
- Python – This was used for some of the more complex language processing, first time using this language. Used multiple packages to assist with encryption of documents.
- SQLite – Used for storage of hashed credentials and database models
- Balsamiq – Used to for the purpose of wireframes and conceptualising code.
- SQL – Used the SQL language to allow for interaction between the front and back end.

1.4. Structure

Introduction: Provides a background into the project and the main aims behind it. The aims will then be followed by the technologies that will be used to implement them. Also will provide an insight into existing technologies.

System: This section will give information on the overall topic of requirements. This includes requirements gathering, functional and otherwise. Will include a number of use cases to provide a conceptual and visual element to the requirements of the project.

Implementation: This section will cover how the features described were actually implemented, it also contains reasoning for why I chose certain technologies and methods to approach implimentation.

Conclusions: This section will give a reflective view of the project, the successes and failures. It will outline what parts worked well and what parts encountered issues.

Further Development & Research: This section covers the possible future development of the application. How it would be expanded and the scope would change given more time.

References: This section covers all referenced materials used in the project. Sources for these materials will also be included.

Appendices: My project proposal and any other relevant documents will be found under this section.

2.0 System

2.1. Requirements

All below requirements were gathered from potential end users, both property managers and renters.

Registration & Secure login: Users should be able to create an account that is trusted to be secure. Once accounts are created users should be able to login using their credentials. A secure authentication system must be used.

Documentation: Documentation should be easily uploaded by the user. The system will handle the processing, encryption, and categorisation of the document on behalf of the user. This documentation needs to be convenient to access.

Payments: Payments should be able to be requested by the property manager and paid by the tenant. Payments must be made via the secure medium provided.

Dashboard: Both users should have access to a dashboard showing information relevant to their role (Tenant/Property Manager). Property managers will have information available on all of their properties.

Availability: The application must have a high uptime and minimal interruption to service on the front or backend. Proper resource allocation must be observed.

2.1.1. Functional Requirements

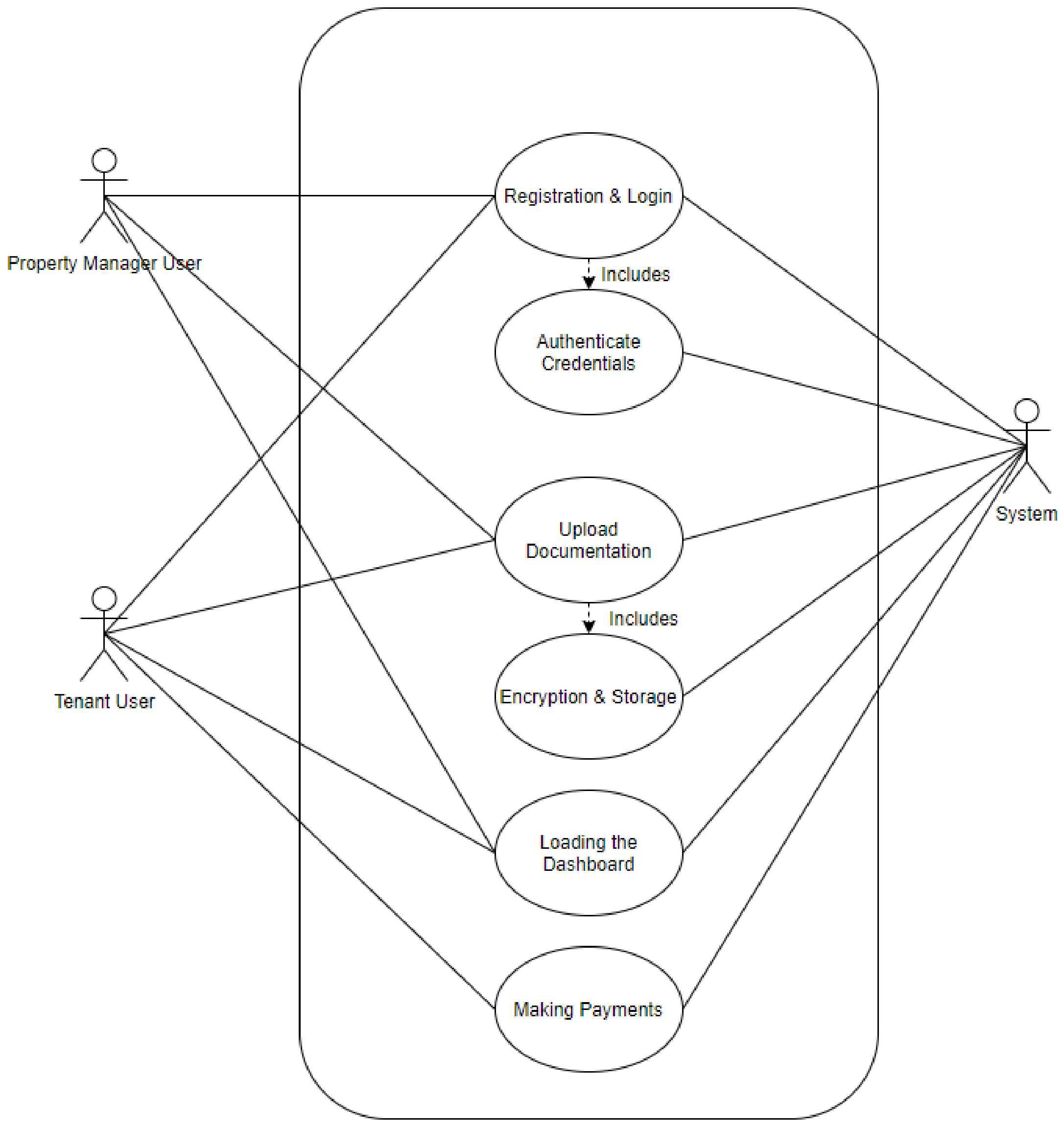
FR1 *Registration & Secure login* - Any user that attempts to interact with the system must be authenticated using the mechanism at the back end for the purpose of secure access control. New users will have their passwords checked to meet a minimum standard and then hashed.

FR2 *Dashboard* - System should pull all information relevant to the authenticated user from the backend database and display it.

FR3 *Documentation* - System will take uploaded documents, process them in the back end using an encryption algorithm, assign rules and store them accordingly.

FR4 *Payments* - System will take payments and process them securely in the backend from authenticated users, amount outstanding can be set by property managers which in turn is changed on the database.

2.1.1.1. Use Case Diagram



2.1.1.2. Requirement 1 <Secure Login & Registration>

2.1.1.3. Description & Priority

This use case shows the system of authentication behind the login & registration actions that can be initiated by either type of user (Tenant/Property Manager). High priority.

2.1.1.4. Use Case 1

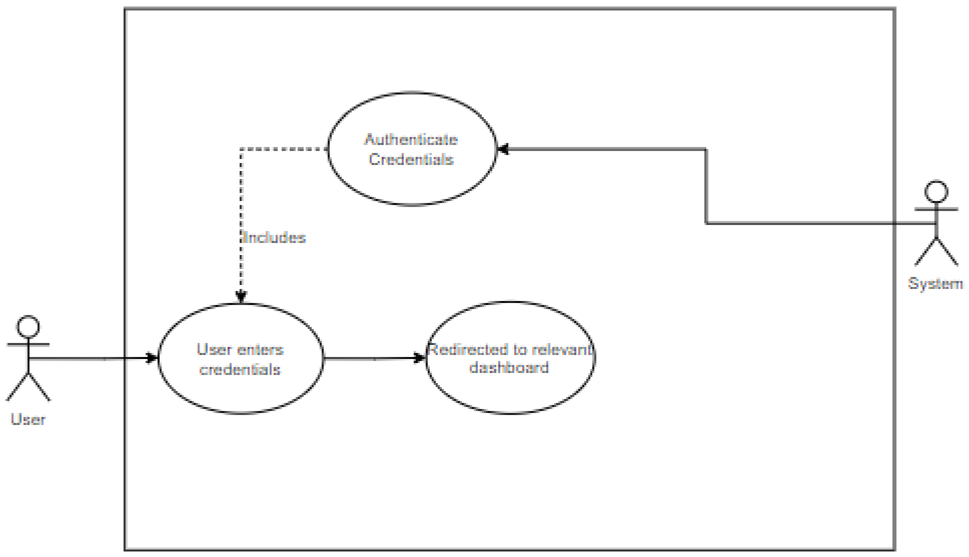
Scope

The scope of this use case is to take user input from the username and password fields and authenticate the user. If a user is not signed up already there will be an alternate flow for registration.

Description

This use case describes the way in which a user can gain access to the system via the secure login & registration system.

Use Case Diagram



Flow Description

Precondition

The system is online and awaiting input.

Activation

This use case starts when either type of user selects the “Login” option.

Main flow

1. The user enters their credentials & selects login.
2. The system authenticates the credentials provided. (AF1) (E1)
3. The system presents the user with the dashboard according to their role.
4. The user now has access to the system & all role specific functions.

Alternate flow

A1 : User does not have an account.

1. The system prompts the user to create an account.
2. The user enters a valid, not in use email and a password in line with NIST standards.
3. The user selects their desired role.
4. The system presents the user with the dashboard.

Exceptional flow

E1 : Account is locked.

1. The system informs the user that their account is locked and that they must change their password.
2. The user follows the password reset flow.
3. User returns to step 1 in the Main flow.

Termination

The system presents the relevant users dashboard.

Post condition

The system goes into a wait state.

2.1.1.5. Requirement 2 <Loading the Dashboard>

2.1.1.6. Description & Priority

This use case describes what happens after the user logs in and is directed to the dashboard page. The data relevant to the user is pulled from the database and presented to them. High priority.

2.1.1.7. Use Case 2

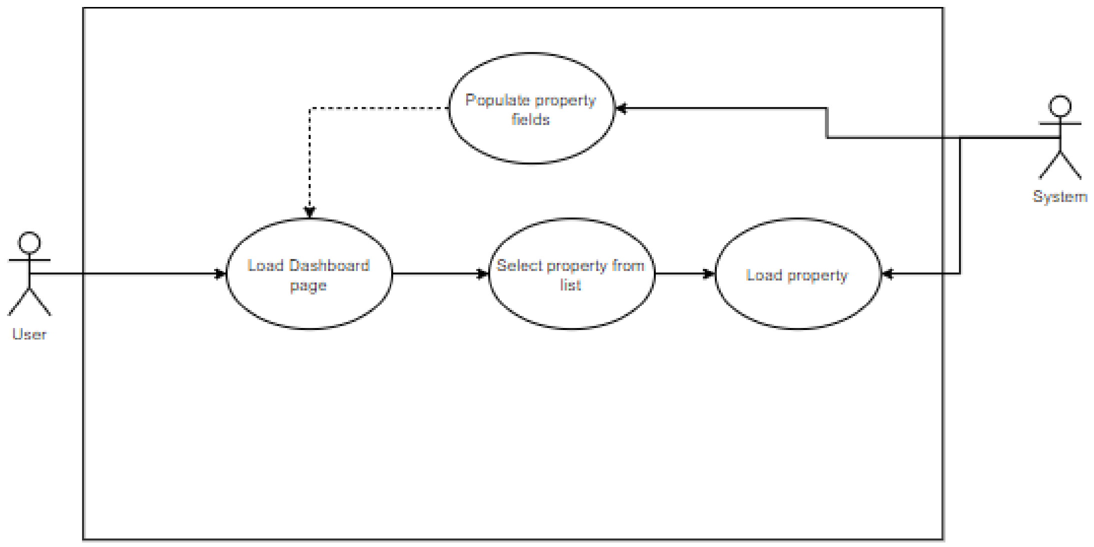
Scope

The scope of this use case is the display of the user’s relevant dashboard by the system after they have successfully logged in.

Description

This use case describes the way the dashboard is displayed to the user after they have logged in.

Use Case Diagram



Flow Description

Precondition

The user has successfully logged in.

Activation

This use case starts when an either type of user initially logs on or selects the "Dashboard" button.

Main flow

1. The system pulls the information relevant to the user from the database. (AF1)
2. The system displays the acquired information to the authenticated user.
3. The user can now view and edit the provided information according to their role.

Alternate flow

AF1 : Database has no information relevant to the user.

1. The system prompts the user to create a new record.
2. The user creates a new record.
3. Return to step 1 in the main flow.

Termination

The user is presented with information relevant to them on the dashboard.

Post condition

The system goes into a wait state.

2.1.1.8. Requirement 3 <Upload Documentation>

2.1.1.9. Description & Priority

This use case shows the system of document upload and processing. Once a document is uploaded by the user the system categorises it. After this it encrypts and stores it accordingly. High priority.

2.1.1.10. Use Case 3

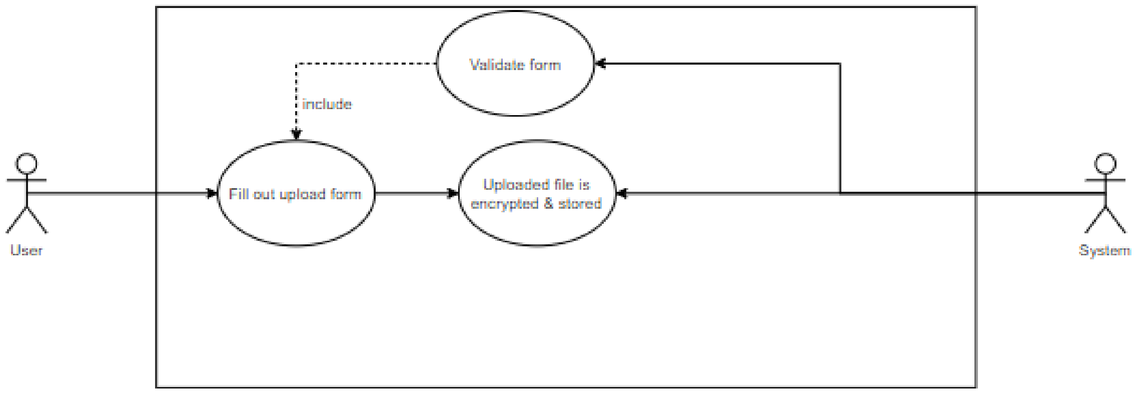
Scope

The scope of this use case is to encapsulate the user uploading documentation and the processing of that documentation by the system.

Description

This use case describes the way In which the system accepts documentation uploads from the user and how the uploads are processed.

Use Case Diagram



Flow Description

Precondition

The user has successfully logged in.

Activation

This use case starts when either type of user selects the “upload” option.

Main flow

1. The user selects the desired file to upload and clicks the “Upload” button.
2. The system verifies the file type and file size (AF1) (E1)
3. The system scans the document, encrypts it and stores it under the users profile.

4. The user can now view the document and all attached content flags attached by the system

Alternate flow

AF1 : Invalid file type or file size

1. System prompts the user to upload a valid file.
2. User uploads a file in a valid format.
3. Proceed to step 2 in the Main flow.

Exceptional flow

E1 : Connection interrupted while uploading

1. User is informed by the system that the upload was unsuccessful and to try again.
2. The user begins their upload at a later time.
3. User returns to step 2 in the Main flow.

Termination

The system stores the document and presents the user with a confirmation message.

Post condition

The system goes into a wait state.

2.1.1.11. Requirement 4 <Payments>

2.1.1.12. Description & Priority

This use case shows the payments system. It demonstrates how the tenant user can make a payment against their outstanding balance once that information is retrieved from the database.

2.1.1.13. Use Case 4

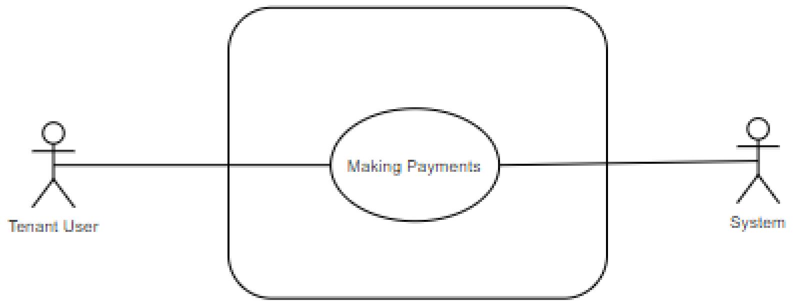
Scope

The scope of this use case involves the payment system. The tenant user can make payments on the outstanding balance against their account for charges such as rent.

Description

This use case describes the way in which a user can make payments against their outstanding balance. Once the outstanding balance has been retrieved the tenant has the option to input how much they wish to pay off the outstanding balance.

Use Case Diagram



Flow Description

Precondition

User is logged in as a “tenant” user.

Activation

This use case starts when the user selects the “payments” section.

Main flow

1. The system retrieves the outstanding balance from the database. (AF1)(E1)
2. The user is prompted to select an amount that they wish to pay.
3. The system authenticates and processes the payment method (E2)
4. The user is presented with a “payment successful” message along with the updated balance.

Alternate flow

AF1 : User has no outstanding balance

1. The system returns a message to the user telling them they have no monies outstanding.
2. The user is presented with the outstanding balance (zero).
3. If the user wishes to bring the account into credit they can enter a desired payment amount.
4. The system authenticates and processes the payment method.
5. Proceed to step 5 in the Main flow

Exceptional flow

E1 : Outstanding amount could not be retrieved

1. The system records an error and notifies the relevant property manager and system administrator via email.

2. The user is informed that an error has occurred and to try again at a later time.
3. Proceed to step 1 in the Main flow.

E1 : Payment fails

1. The system informs the user that the payment has failed and to try another method.
2. Return to step 2 in the main flow.

Termination

The system presents the updated outstanding amount to the user.

Post condition

The system goes into a wait state.

2.1.2. Data Requirements

Seeing that the application takes input from users and sanitises it, data security is undoubtedly one of the main requirements of this project. There must be trust established between the source of the data and its destination.

All data in this project must be required to be held to the standard of the CIA triad.

Confidentiality: The data must be compartmentalised and properly encrypted. The robust authentication method ensures that unauthorised parties will be prevented from accessing the system.

Integrity: Integrity of the data is required to be kept at a high standard using encryption for the uploaded documents and hashing for the user's login details. Establishing trust between the user and the system is a key requirement.

Availability: The data will be required to have a high level of availability. This will be accomplished by appropriate resource management and compartmentalisation of data. Implementation of levels of redundancies using a defence in depth approach will assist in this endeavour.

2.1.3. User Requirements

In order to firstly access the system, a user will need to register as either a property manager or a tenant. The user will not be able to navigate to any other page until they have successfully logged in. Each page is tailored to the data stored on the user by the system. The more information the user has provided to the system, the more fleshed out the dashboard will be.

The user must have an internet connected device to use the application, this application will be accessible via a desktop, laptop or mobile device.

Property managers must provide their tenants with a generated code in order for them to register as their tenants. Property managers do not need any such code to register their own accounts.

Property managers must provide initial information on properties, this could include the address, monthly rent amount and number of tenants.

2.1.4. Usability Requirements

The focus of the application should be letting the complex operations happen without any visibility to the user, ensuring that the application is not overly complex from the perspective of the user.

Tooltips will be given to first time users in order to help them understand the main functionalities of the system.

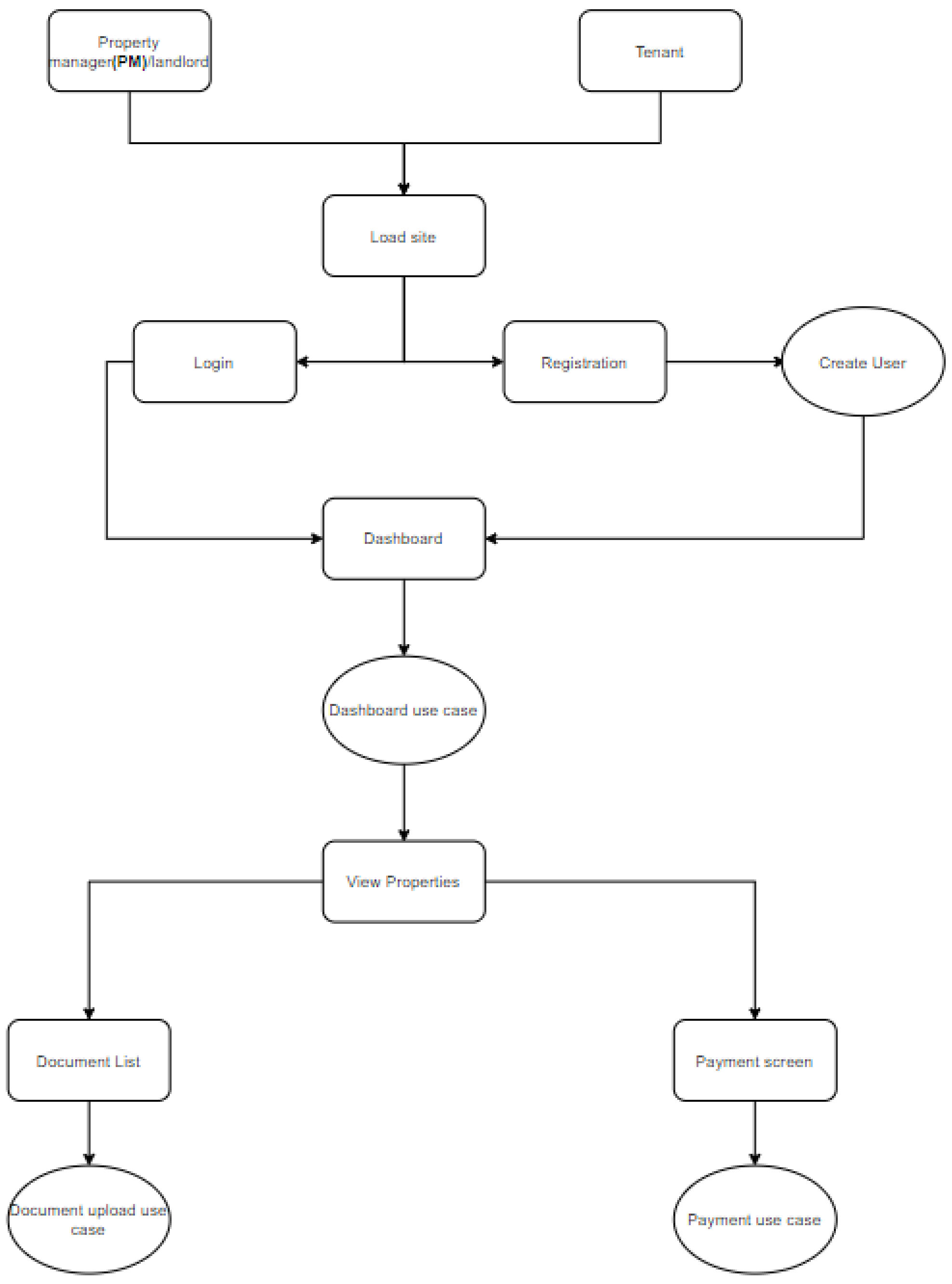
Each use case will be separated into its own page on the web application, this will ensure each page is not overly cluttered. This will make navigation and finding each function much easier.

Accessibility will be required for colour-blind and other visually impaired users. This can be done using certain colours and ensuring alt tags are available for all users, including those using screen readers.

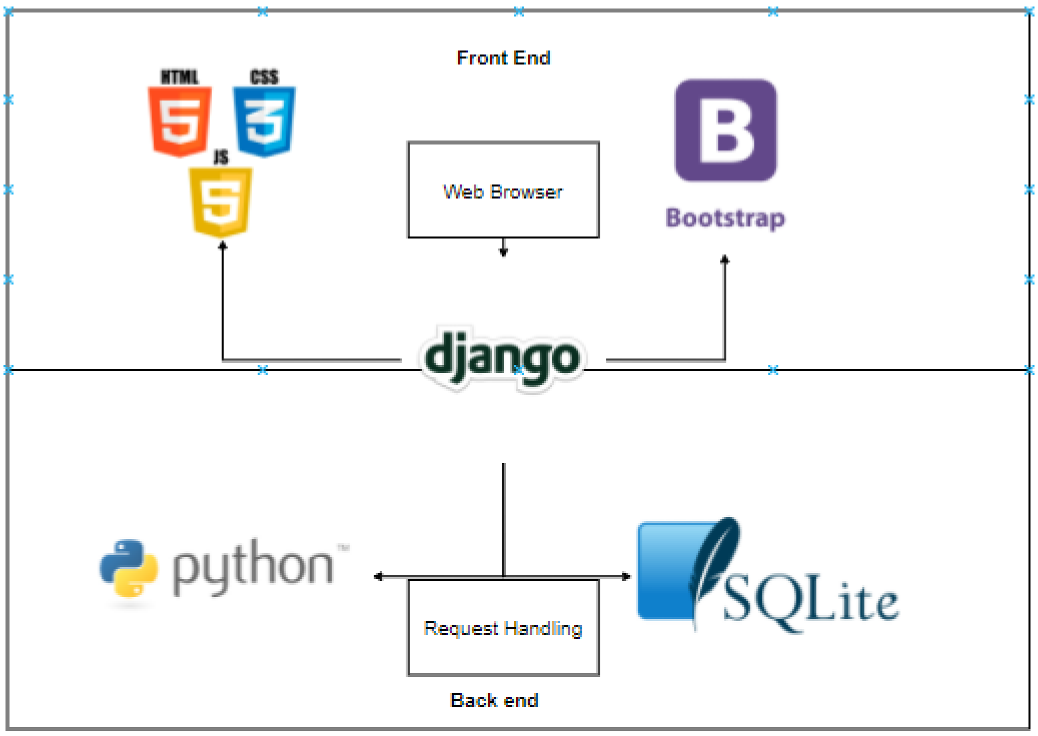
There will be testing required to ensure the application is running as efficiently as possible, any abnormally high resource usages will need to be investigated and patched in order to ensure a high level of convenience for the user.

2.2. Design & Architecture

Below is a diagram of how the application is laid out, including how the architecture and the use cases tie in.



Below is a view of the system architecture as it currently designed. Django acts as the intermediary between the front and the backend, it manages the display of pages to the user while also getting requests from these pages and passing them through to the backend.



2.3. Implementation

2.3.1. File Management

In this section I will discuss how file upload/download is handled in the application. Uploaded files are taken via a Django file field and sanitised before being saved to the file system. Each file is associated with an entry in the database that sets a relationship with a property, the database is modelled in a many to one relationship such as one property can have many files. I will explain how each field in the model works and their relevance to the file system I have created. The attached screenshot is taken from main/models.py

Property:

This is the foreign key used to establish the many to one relationship with the Property model, it is used to protect against the possibility of files being uploaded and orphaned with no parent property object. I added the 'on_delete=models.CASCADE' condition to this attribute to ensure that any changes in the parent, in this case the property model will be reflected when it comes to deletions and updates. If a property is deleted, the files will be deleted along with it to ensure no data leakage.

Name:

The name attribute is a custom varchar that is set by the user to help them identify the file, It is what is displayed to the user when they view their file list but it is not used in any computations done on the pdf or its storage.

Pdf:

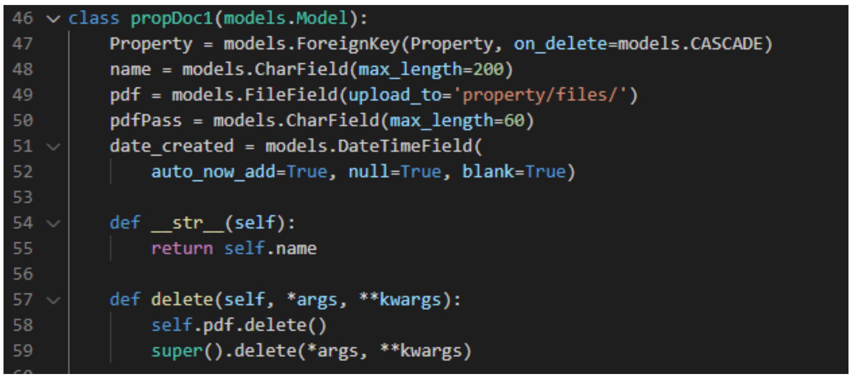
The pdf attribute takes the file and physically stores it within the filesystem, using this attribute is essential for modifications made to the actual file itself such as getting the url using the .url parameter which can then be used for downloads and deletions.

PdfPass:

This is the password to be fed into the encryption algorithm during the initial stages of the record creation. The length of the field is limited to prevent long password attacks that could stall the hashing algorithm, which would result in denial of service. The user can have a different pdfPass for every document they upload to the system.

Delete:

A major issue I faced during development was the record of the file deleting but not the actual file itself. To resolve this in a secure fashion I wrote an overridden delete() method that ensured that when all the attributes were deleted, the stored file was also.



```
class propDoc1(models.Model):
    Property = models.ForeignKey(Property, on_delete=models.CASCADE)
    name = models.CharField(max_length=200)
    pdf = models.FileField(upload_to='property/files/')
    pdfPass = models.CharField(max_length=60)
    date_created = models.DateTimeField(
        auto_now_add=True, null=True, blank=True)

    def __str__(self):
        return self.name

    def delete(self, *args, **kwargs):
        self.pdf.delete()
        super().delete(*args, **kwargs)
```

2.3.2.File Encryption

One of the key features of the application is the encryption of PDF files to ensure security and to provide an innovative aspect from a commercial point of view. In order to implement this, I made use of the PyPDF2 library to read and write to the uploaded files. From this baseline I developed anj algorithm to encrypt the PDF files dynamically using the user defined PDF password taken from the upload form. The decision to encrypt using a password rather than having the system automatically decrypt and view the files on demand was due to my following of the defence in depth strategy when it comes to security. Having two layers of authentication to view sensitive files (the first being the user account login, the second being the pdf password) provides a much more secure application while not overly hindering user experience.

Initially I had the system set up wherein the PdfReader would search for keywords inside the document, apply labels and encrypt only when these labels were found. I changed to giving the user the option to encrypt every uploaded document for a few reasons. The main reason is the speed of the algorithm, before implementation I had no idea how long encryption would take, after some fine tuning however the system now encrypts the vast majority of documents in under .5 of a second. This makes encrypting a no brainer as it provides massive amount more security while also allowing a document to remain unencrypted by leaving the PdfPass field (mentioned in the file management section) blank. Applying labels such as [Payment Details, PPS Number, Home Address] to a file entry is also essentially putting a “Please Hack Me” label on it. The labels were decided against for the security issues they could present.

The encryption algorithm itself follows a number of steps and conditions.

POST Request Check (line 123):

The first step in the algorithm ensures that the request made by the user is a secure POST request, if this is not the case, for example if a malicious party is attempting to modify the request, the algorithm will not run and will redirect back the original file upload page. This is an innovation developed to counter vulnerabilities I discovered during penetration testing of the application wherein requests could be targeted and forged.

Grabbing the file from the request (line 131):

Using ‘request.FILES[‘document’]’, the file that has been submitted with the form can be grabbed and assigned a variable, using this variable is essential to the operation of the encryption algorithm.

Creating a PdfReader and PdfWriter (line 135,136):

Using the PyPDF2 package to read and write to the file is essential for the purpose of encryption, to this end the PdfReader is instantiated with the uploaded file from the previous step being passed to it. The PdfWriter is then instantiated with no arguments passed for the time being, it will be used later.

Creating the for loop (line 138,139):

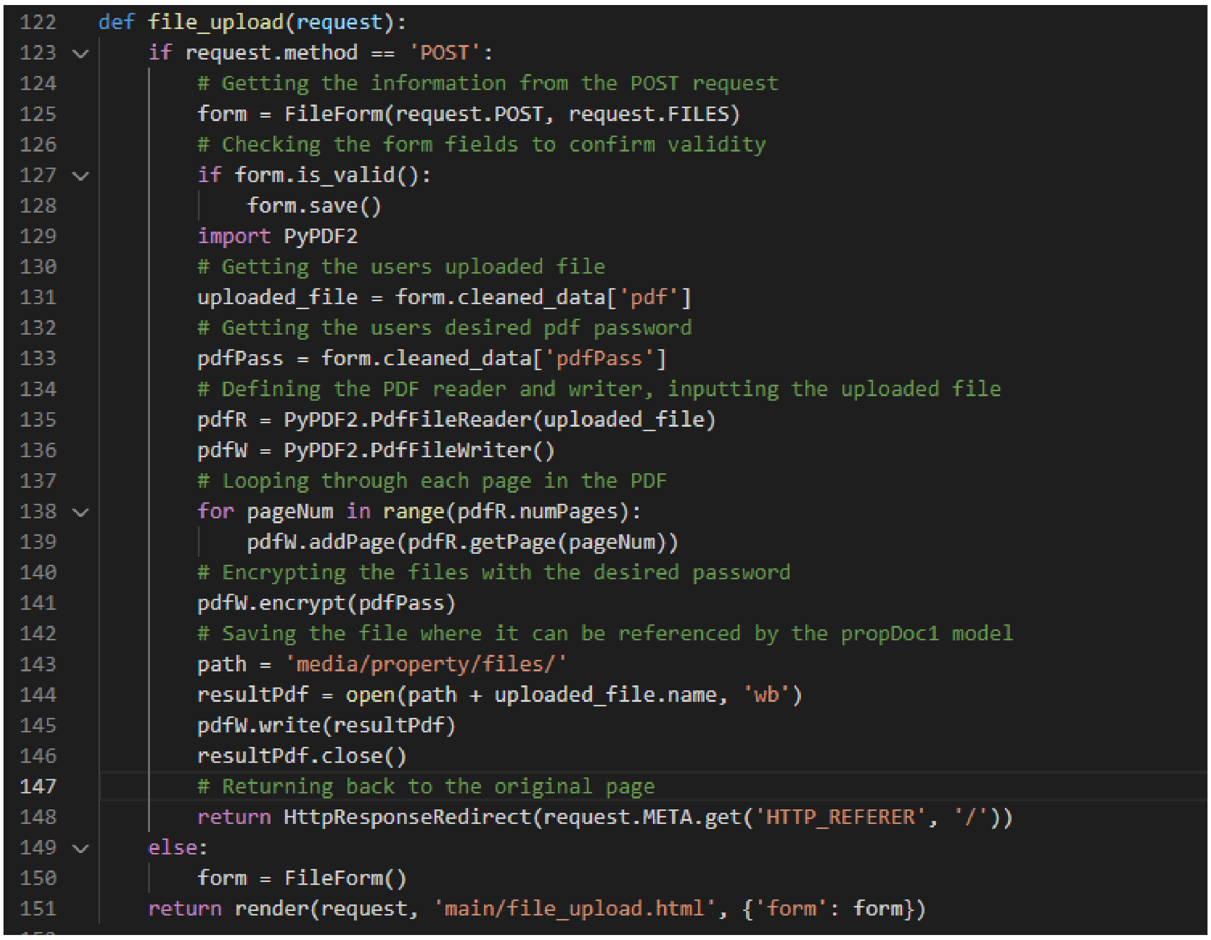
In order to ensure that all pages in the document will be encrypted I implement a for loop which will go through the number of pages in the document, this ensures that the new encrypted document will match the original via the use of the pdf writer.

Encrypting using the user defined password (line 141):

Using the information gained from the form at line 133 we can now encrypt the document using the password the user provided

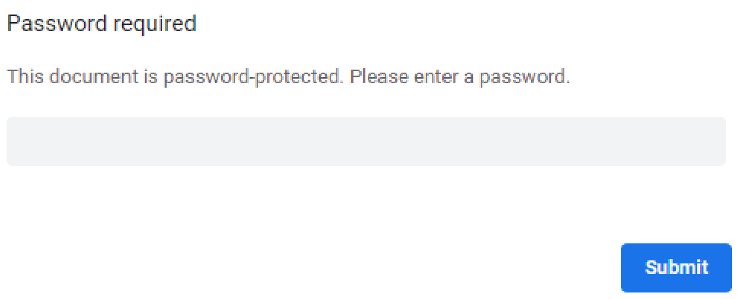
Saving the encrypted document (line 144):

In this line we are assigning a variable to the new encrypted file. The file is being saved to the directory entry that aligns with the data saved in the propDoc1 model, meaning that information about the document such as the property it is related to, can be fetched and displayed for the user. The actual information within the pdf is now saved and fully encrypted.



```
122  def file_upload(request):
123      if request.method == 'POST':
124          # Getting the information from the POST request
125          form = FileForm(request.POST, request.FILES)
126          # Checking the form fields to confirm validity
127          if form.is_valid():
128              form.save()
129          import PyPDF2
130          # Getting the users uploaded file
131          uploaded_file = form.cleaned_data['pdf']
132          # Getting the users desired pdf password
133          pdfPass = form.cleaned_data['pdfPass']
134          # Defining the PDF reader and writer, inputting the uploaded file
135          pdfR = PyPDF2.PdfFileReader(uploaded_file)
136          pdfW = PyPDF2.PdfFileWriter()
137          # Looping through each page in the PDF
138          for pageNum in range(pdfR.numPages):
139              pdfW.addPage(pdfR.getPage(pageNum))
140          # Encrypting the files with the desired password
141          pdfW.encrypt(pdfPass)
142          # Saving the file where it can be referenced by the propDoc1 model
143          path = 'media/property/files/'
144          resultPdf = open(path + uploaded_file.name, 'wb')
145          pdfW.write(resultPdf)
146          resultPdf.close()
147          # Returning back to the original page
148          return HttpResponseRedirect(request.META.get('HTTP_REFERER', '/'))
149      else:
150          form = FileForm()
151      return render(request, 'main/file_upload.html', {'form': form})
```

Upon attempting to access the file, the user is met with the following screen:



Password required

This document is password-protected. Please enter a password.

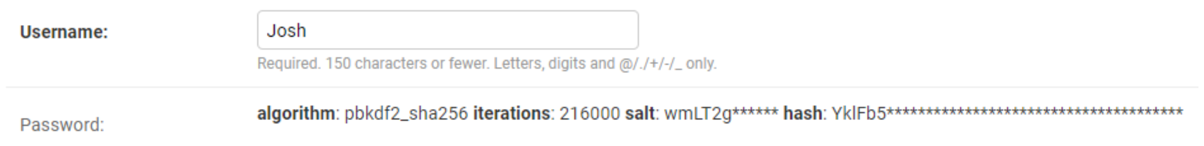
2.3.3.Registration & Authentication

In order to present a modern and innovative system of registration and authentication I attempted to best balance security with performance of the application. This resulted a system where the registration form is checked and sanitised before any information is entered into the database. The entered password is checked against a number of industry standard and leading policies such as:

- The password cannot be too close to the username or email.
- The password has to be at least 8 characters (OWASP Standard).
- The password can not be a generic or widely known cracked password.
- The password must contain at least a letter, it cannot be entirely made up of numbers.

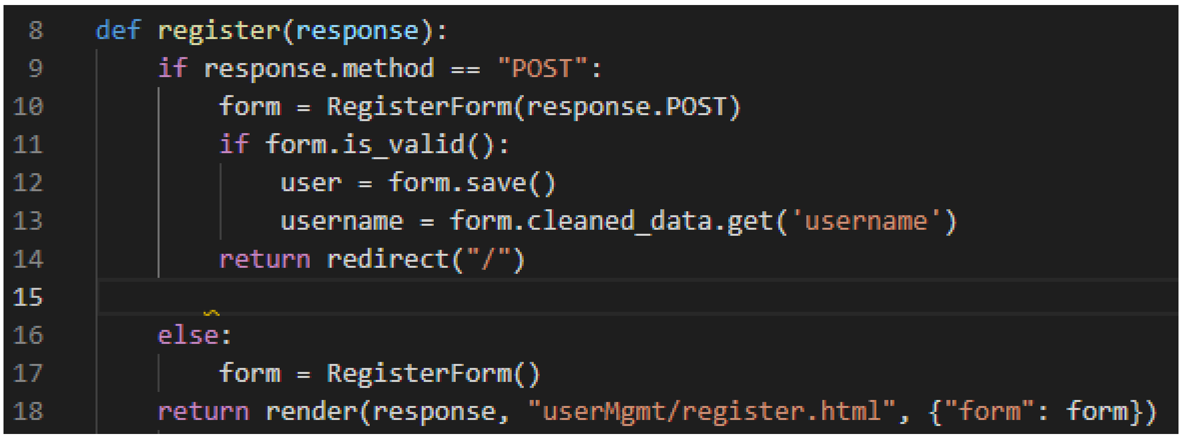
This method of actively checking for vulnerabilities in a password by comparing it for similarities against the user details is much more dynamic and innovative than the traditional requirements for a password. My thinking was that if a password simply required a symbol and a number, a user could easily set their password to their username plus a number and symbol. For example, if we had a user Jonathan who wished to set his password to Jonathan1! He could do so in a traditional system under the guise of the password being complex enough. In the password system implemented in this application the similarity check would not allow the username to be a part of the password. These sorts of checks are emerging more and more in new applications and from my reading they appear to be quickly becoming an industry standard, hence why they are integrated into this application.

Once a password is validated it is hashed with the PBKDF2 algorithm to ensure that no raw passwords are stored in the database. Naturally once a password is hashed the original password text cannot be pulled from the database, if I attempt to do this via the Django admin panel, I am simply presented with the hash used to store it:



Username:	Josh Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.
Password:	**algorithm**: pbkdf2_sha256 **iterations**: 216000 **salt**: wmLT2g****** **hash**: YklFb5**************************************

The below screenshot is the process used when the data is returned from the form, found in userMgmt/views.py. Again, we have the check to ensure the response is indeed a secure POST request before saving any data. Once this is done the form will run the check on the respective entry in forms.py and validate the user entries against the required criteria. If this is correct, the function will save the new user into the database. If there are any issues, the user will be redirected back to the registration page and notified which field is incorrect so that they may try again.



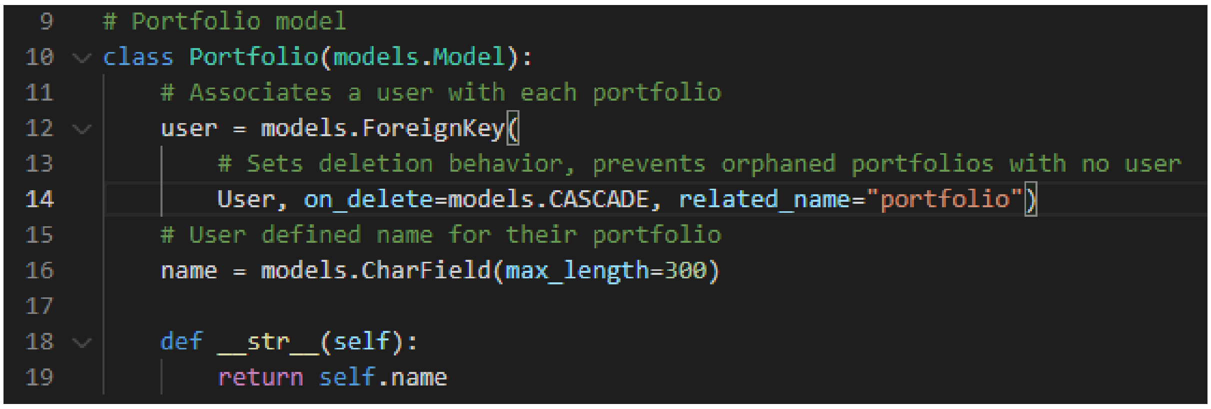
```python
def register(response):
    if response.method == "POST":
        form = RegisterForm(response.POST)
        if form.is_valid():
            user = form.save()
            username = form.cleaned_data.get('username')
        return redirect("/")

    else:
        form = RegisterForm()
    return render(response, "userMgmt/register.html", {"form": form})
```

2.3.4.Managing Page Access

Managing page access in this application is done by following two main methodologies to increase security and also display the innovation of the application. The first is restricting pages based on user. Each portfolio is required to have a user attached to it and therefore information can be restricted based on authenticated users, only portfolios belonging to the authenticated user can be viewed by that user. This is implemented by having a mandatory user field In the portfolio model in models.py, using this ensures that when a user queries the contents of the portfolio.

The code shown for implementing this is below, it can be seen that on line 12 the user is set and inserted as a foreign key. This operation is then continued on line 14, on this line we can see that the deletion behaviour is inserted, meaning that if a user were to choose to permanently delete their account, their portfolio would also be deleted. This is part of a defence in depth strategy to prevent against unauthorized portfolio access:



```
 9  # Portfolio model
10  class Portfolio(models.Model):
11      # Associates a user with each portfolio
12      user = models.ForeignKey(
13          # Sets deletion behavior, prevents orphaned portfolios with no user
14          User, on_delete=models.CASCADE, related_name="portfolio")
15      # User defined name for their portfolio
16      name = models.CharField(max_length=300)
17
18      def __str__(self):
19          return self.name
```

Of course, since a defence in depth strategy is being followed in order to keep to a commercial standard and also display the difficulty of the project, I implemented another method of controlling page access, decorators.

Decorators in this context are called before each view to check parameters to ensure a user is authorized to view the page. In this regard I developed two decorators, one to ensure that the user is actually logged in before they can view certain pages (eg: dashboard, file upload) and another to check the users group to ensure they have privileges to view certain pages (eg: landlord can only create properties) in a dynamic fashion. I will talk about how I implemented these and display the code from the decorators.py file.

Authorized Users check (line 6-23):

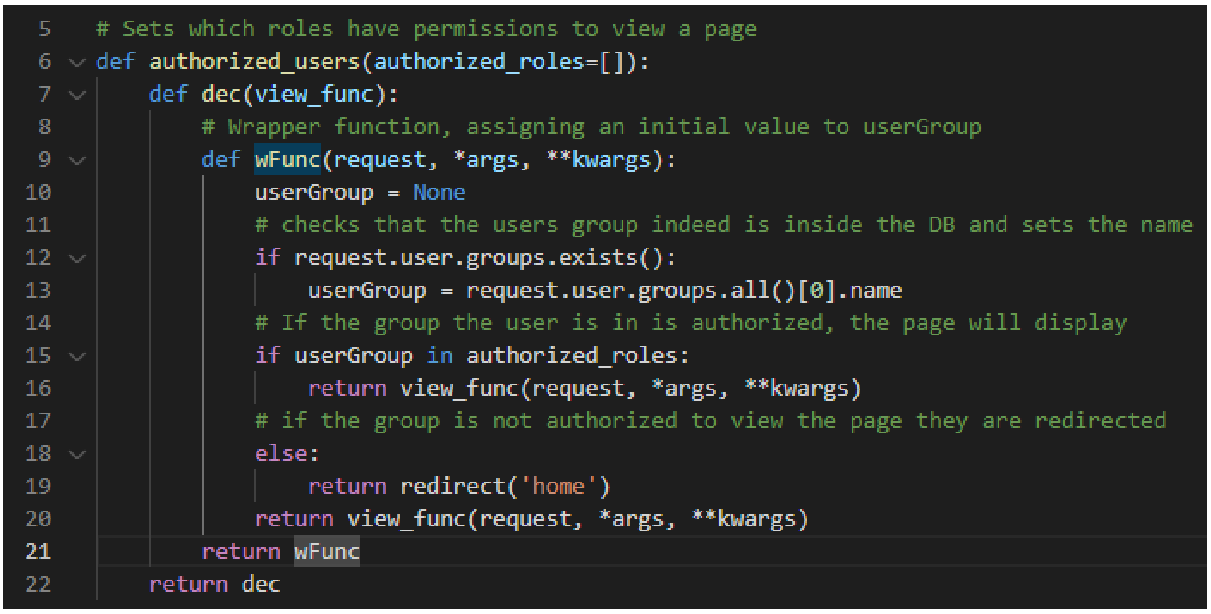
Each user is assigned to a group when they sign up, these groups are then used to assign access to certain pages depending on their permissions level. This is another layer of defence on top of user specific pages to ensure that the clearly defined roles in the application (eg: Landlord, Tenant) can only access functions relevant to their needs.

I implemented this by firstly defining the function in line 6, creating a collection that can be dynamically changed from page to page depending on the restrictions needed.

Following on from this the wrapper function on line 9 which contains the logic is defined. The userGroup variable is then defined on line 10. The user accessing the page is then queried, the system will check which groups they are assigned to and confirm that these groups exist within the database.If the groups do indeed exist then the group is assigned to the userGroup variable for further operations, this occurs on line 13.

On line 15 The userGroup is then compared against the groups that are allowed access, this is dynamic and can change on a page-by-page basis (I will show how these groups are defined depending on the view later). If the userGroup passes the check on line 16, the view and subsequently the page is returned to the user. From my testing this evaluation has little to no impact on user performance as it is all performed on the backend.

The else condition from line 18-20 redirects users back to the page they came from if they are not a member of a group that is authorized to view the page. The described code is shown below:



```
 5  # Sets which roles have permissions to view a page
 6  def authorized_users(authorized_roles=[]):
 7      def dec(view_func):
 8          # Wrapper function, assigning an initial value to userGroup
 9          def wFunc(request, *args, **kwargs):
10              userGroup = None
11              # checks that the users group indeed is inside the DB and sets the name
12              if request.user.groups.exists():
13                  userGroup = request.user.groups.all()[0].name
14              # If the group the user is in is authorized, the page will display
15              if userGroup in authorized_roles:
16                  return view_func(request, *args, **kwargs)
17              # if the group is not authorized to view the page they are redirected
18              else:
19                  return redirect('home')
20              return view_func(request, *args, **kwargs)
21          return wFunc
22      return dec
```

Logged in User check (line 27-36):

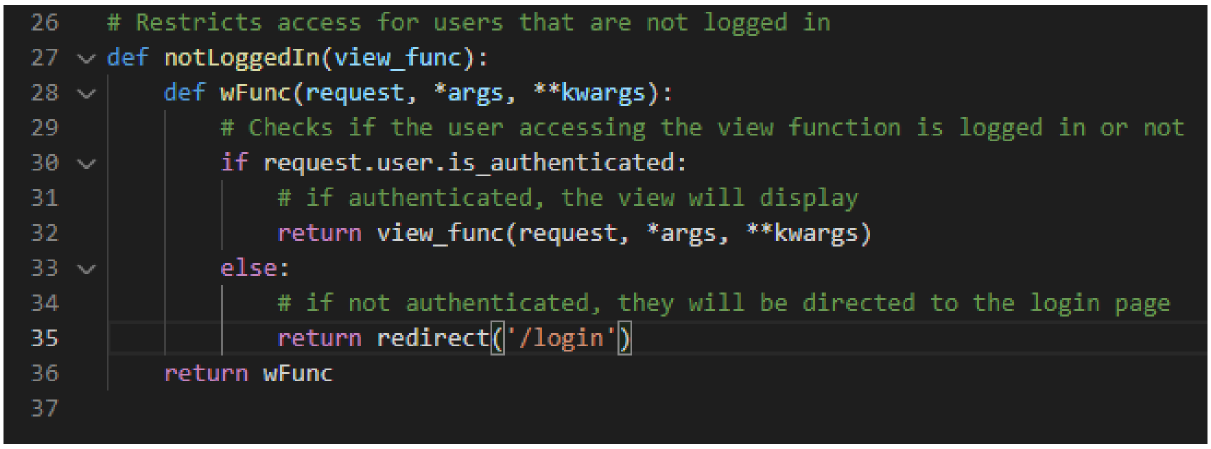
This application is strictly only for use by users that have an account. I discussed earlier how information is filtered on a user-by-user basis using database models to ensure streamlined sessions and to prevent database leakage. To add another layer to this I created a decorator that will deny users access to pages if they are not logged in and authenticated.

On line 27 the function is defined, there is no collection necessary like there was in the previous decorator, pages will not need to add any dynamic requirements into the decorator.

On line 28 the wrapper function is defined; this contains the logic for the operation and is used to pass through parameters.

Line 30 takes in the request and checks if the user making it is authenticated. If they are the request is then passed to line 32, which displays the desired view and subsequent page to the user.

Line 33 handles a situation wherein the user is not logged in, if this is the case line 35 will redirect the user to the login page. The described code is shown below:

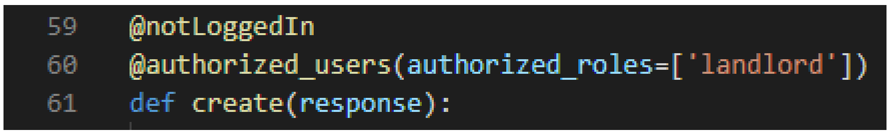


```
26  # Restricts access for users that are not logged in
27  def notLoggedIn(view_func):
28      def wFunc(request, *args, **kwargs):
29          # Checks if the user accessing the view function is logged in or not
30          if request.user.is_authenticated:
31              # if authenticated, the view will display
32              return view_func(request, *args, **kwargs)
33          else:
34              # if not authenticated, they will be directed to the login page
35              return redirect('/login')
36      return wFunc
37
```

I will now show both the Authorized Users check and Logged in User check decorators in use. In views.py

Line 59 is first calling the Logged in User check to ensure that an authenticated user is access the page, if they pass this check they are passed onto the next line.

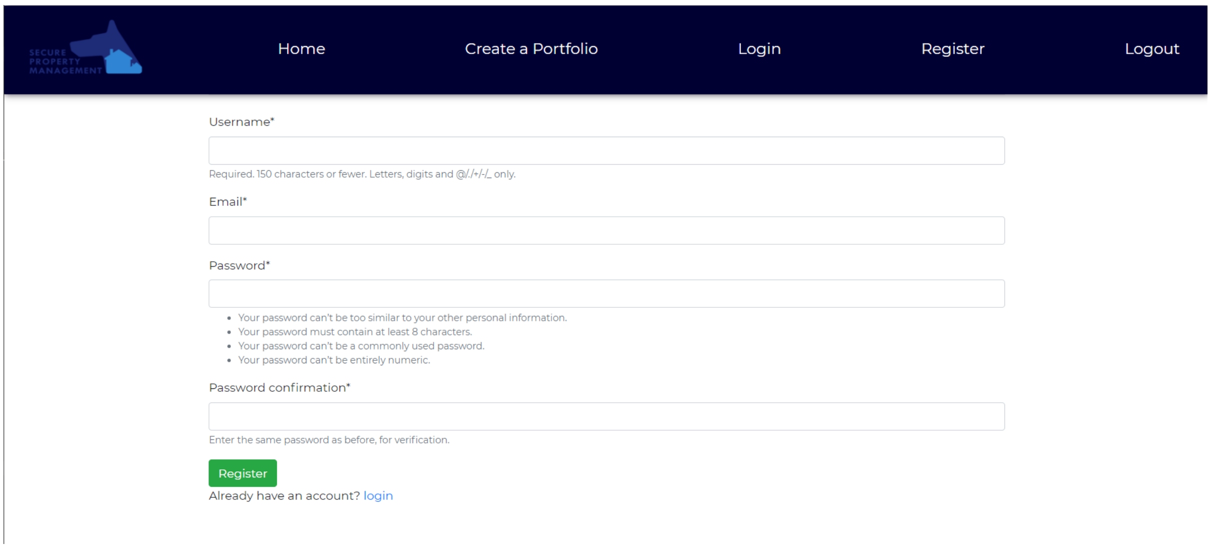
That next line being line 60, here use is being made of the collection that was defined in the decorator to dynamically set allowed groups. In this case we only want users with the ‘landlord’ group to access the page so we insert the group name as an argument. If the decorator returns that the user is indeed a member of the landlord group, the create view is presented to the user. The afforementioned redirects for both decorators will run if the conditions are not satisfied. This adds complexity to the defence in depth system and ensures users can only access pages they are meant to access:



```
59  @notLoggedIn
60  @authorized_users(authorized_roles=['landlord'])
61  def create(response):
```

2.4. Graphical User Interface (GUI)

The first page a user will see is the registration page, here all the password policy I mentioned in implementation is presented to the user and enforced:



Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email*

Password*

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

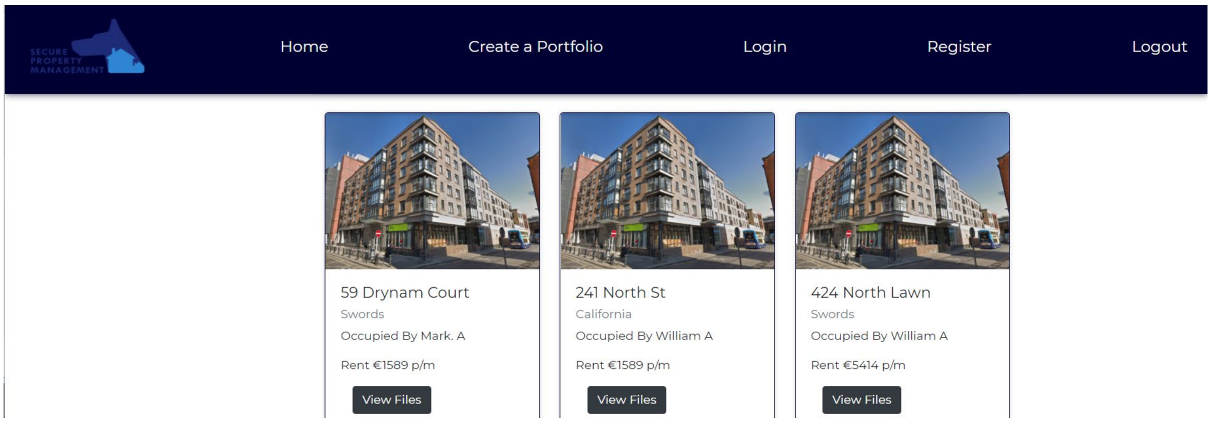
Password confirmation*

Enter the same password as before, for verification.

Register

Already have an account? login

When a user enters their portfolio, they are presented with a list of properties, from here they can view the files associated with them or create a new property:



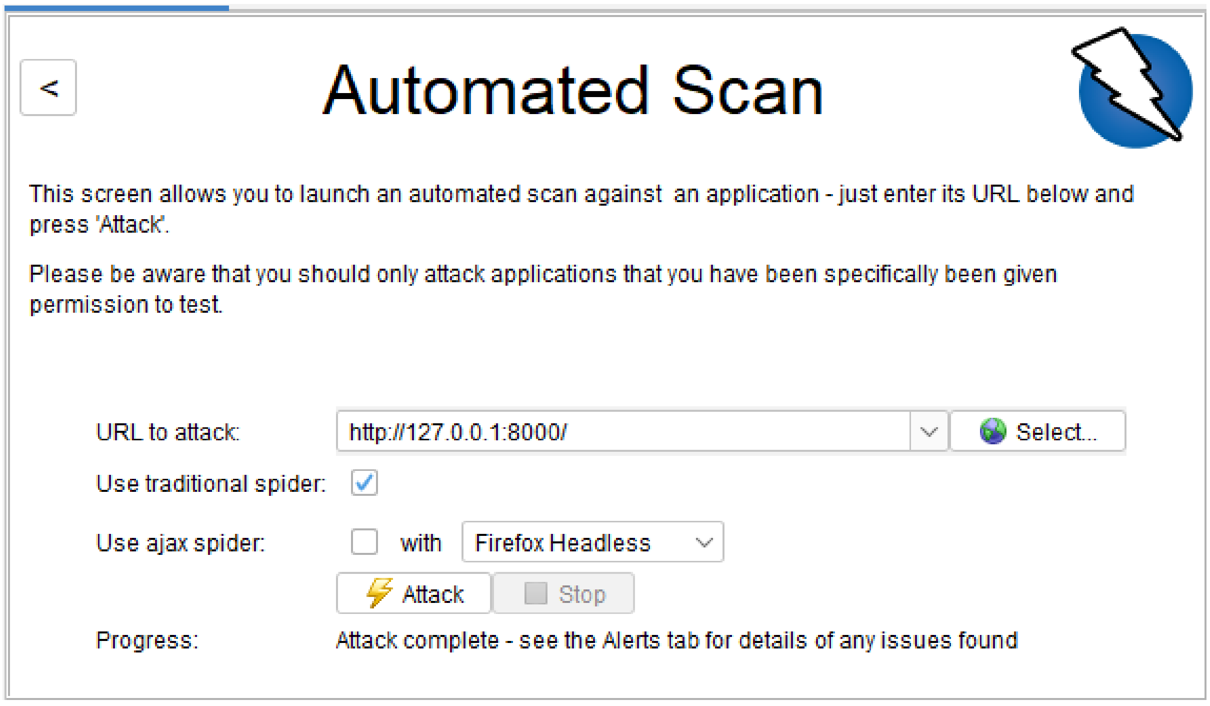
59 Drynam Court
Swords
Occupied By Mark. A
Rent €1589 p/m
View Files

241 North St
California
Occupied By William A
Rent €1589 p/m
View Files

424 North Lawn
Swords
Occupied By William A
Rent €5414 p/m
View Files

2.5. Testing

In order to test the security of the web application I ran every page through an industry leading security scanner, OWASP ZAP. This scanner was ran on each individual page (unit) aswell as on the overall system itself:



Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

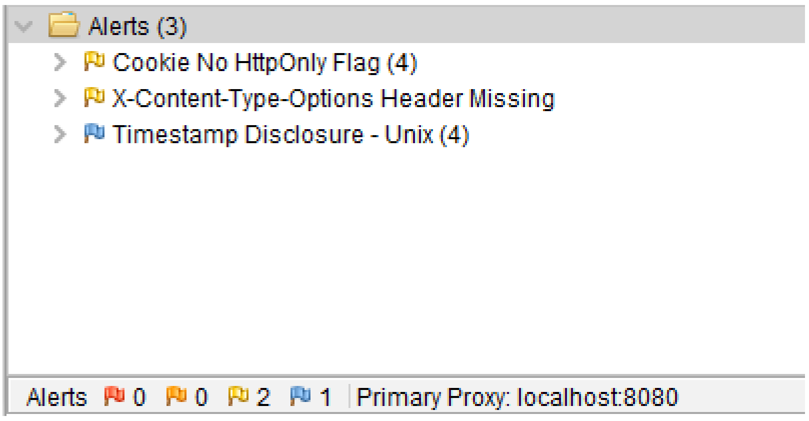
URL to attack: http://127.0.0.1:8000/

Use traditional spider: ✓

Use ajax spider: ☐ with Firefox Headless

Progress: Attack complete - see the Alerts tab for details of any issues found

In the above screenshot we can see the target, in this case the main URL of the website on localhost. As evidence I will provide the results of my final scan:

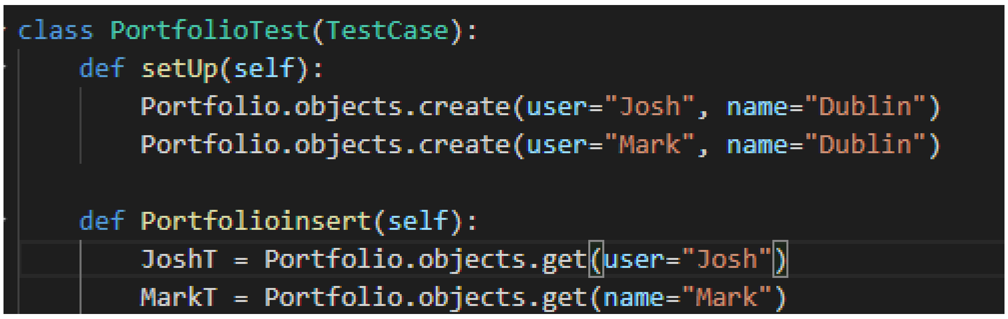


Alerts (3)
- Cookie No HttpOnly Flag (4)
- X-Content-Type-Options Header Missing
- Timestamp Disclosure - Unix (4)

Alerts 0 0 2 1 Primary Proxy: localhost:8080

The application, due to the implementation of innovative features such as CSRF tokens to protect against cross-side scripting is very secure as shown above. The only alerts are informational and very low risk. I achieved this by consistent penetration testing by myself (learned via the penetration testing module) and by reading up on topics such as cross side scripting (PortSwigger, 2020).

In terms of testing the code, I created a unit test for every model and the important classes (Create Property, Portfolio etc) Here is one such test case I created for the Portfolio model:



```
class PortfolioTest(TestCase):
    def setUp(self):
        Portfolio.objects.create(user="Josh", name="Dublin")
        Portfolio.objects.create(user="Mark", name="Dublin")

    def Portfolioinsert(self):
        JoshT = Portfolio.objects.get(user="Josh")
        MarkT = Portfolio.objects.get(name="Mark")
```

By using unit testing both with OWASP ZAP for security and Django for code quality and parameter checks I was able to build a solid foundation for my system testing. The system tests using OWASP ZAP helped me achieve the level of security that was desired since requirements for it were laid out.

2.6. Evaluation

The system was evaluated by comparing its performance on multiple different devices. I tested it on the following:

-Desktop PC (1920x1080 Display), Fast internet connection

-iPhone 11, Fast internet connection

-Samsung phone from circa 2014, slow mobile data connection

On all of these devices I got the main homepage to load within 1 second. The fastest two devices (desktop and iPhone) were within .3 of a second, almost instantaneous. When I began development the slower device struggled greatly and would often take in excess of 2 seconds to load the list of properties, especially if there was an excess number of them.

The security was evaluated by my own penetration testing and by the OWASP ZAP testing shown in the testing section.

Scalability in this application proved to be extremely good. By creating a large array of users with a large number of properties in each portfolio I ran some tests on performance. I was able to push the document storage and querying ability of the backend to the limit via automated queries and requests. Given that the vast majority of work is done on the backend I found the application to be extremely well able to handle multiple requests in a timely manner and only dependant on the resources of the server hosting the application. Session management and the defence in depth strategy ensure no data leakage, even at the most intense of workloads. In order to increase scalability I would in future invest in more hardware and a higher level of bandwidth to handle multiple clients accessing the system at once.

3.0 Conclusion

3.1. Advantages

The most major advantage of Secure Property Management(SPM) is its ease of use for the user while also being much more secure than any legacy system that a firm may have in use. Ransomware and phishing attacks are becoming more and more prevalent meaning that having a password on an email or windows account is not enough to protect the files within. This application follows a defence in depth methodology wherein the first line of defence is the strong, modern and dynamic password requirements defined in the implementation section to prevent any accounts being easily compromised. This is followed however by robust document encryption, ensuring that if an account somehow is breached an attacker would not be able to view any confidential or sensitive files as they are all password encrypted. CSRF tokens are also used to prevent any cross-side scripting attacks while also being lightweight enough to not damage performance (OWASP, 2020) This defence system is a good medium between ease of use and security.

The aforementioned encryption algorithm is another big advantage to this project. It works quickly as not to hinder user experience while also securing the file to a high standard. The fact that it is encrypted by password means that the landlord has great flexibility in distributing the document, they could use the system to allow tenants to view said document or indeed could download the newly encrypted file from the file list and share it with anyone they give the password to. Having this centralised database for properties and having files associated with properties is incredibly useful for landlords both big and small, naturally the bigger the portfolio a user has, the more the more useful the categorisation of documents by property will be.

Another advantage is the lightweight frontend of the application. I made this a priority as the web application was designed to work across multiple devices with varying amounts of processing power. By choosing Django and implementing the vast majority of the complex functions on the back end, the web application can be ran on nearly any device. To confirm this I performed testing on my desktop PC, current mobile phone and even my mobile phone from circa 2014. By using Bootstrap on the frontend, the site is scalable to all display resolutions while not being overly taxing on system resources.

3.2 Disadvantages

There are some disadvantages with this project that I can discuss, mostly due to the lack of monetary investment into payments. Payment gateways are expensive to set up and maintain, this being a final year project with no monetary budget made a complex payment system out of the scope of this project. Another disadvantage would be the compromise on visual fidelity to meet performance requirements on all devices. The project was designed from the start to be accessible on devices of nearly any age and also to run on poor internet connections (rural mobile internet for example) This meant that the design needed to be streamlined with only the functional elements of the UI being displayed without too many images or other attributes that could slow down the browser and harm the user experience. I prioritized functionality over visual fidelity which can be seen as a disadvantage.

4.0 Further Development or Research

Advanced payment gateway:

Given more resources I would add a complex payment gateway such as stripe to the application. This would increase the usability of the application and it would make it more appealing to potential investors and clients.

Increased database capacity:

I developed everything I wanted to develop in the timeframe but given a few team members and no budget restraints I would increase the scale of the database to service a large potential client base. The scalability of the application is there but future developments could see it expanded and capitalised upon. Subscriptions could be charged to clients for larger document storage space.

5.0 References

Cheatsheetseries.owasp.org. 2020. *Authentication - OWASP Cheat Sheet Series*. [online] Available at: <https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html> [Accessed 7 May 2021].

Owasp.org. 2020. *Cross Site Request Forgery (CSRF) | OWASP Foundation*. [online] Available at: <https://owasp.org/www-community/attacks/csrf> [Accessed 7 May 2021].

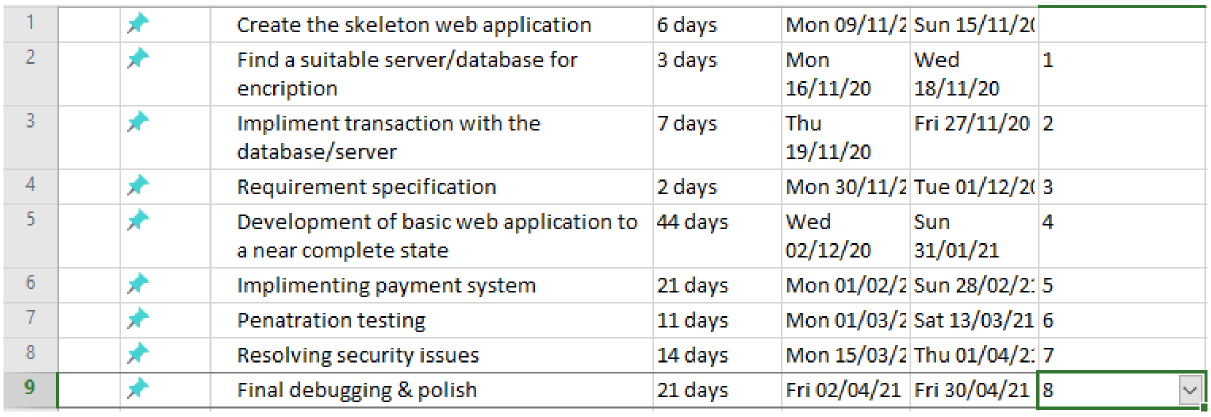
Academy, W., 2020. *CSRF tokens | Web Security Academy*. [online] Portswigger.net. Available at: <https://portswigger.net/web-security/csrf/tokens> [Accessed 6 May 2021].

Academy, W. and scripting, C., 2021. *What is cross-site scripting (XSS) and how to prevent it? | Web Security Academy*. [online] Portswigger.net. Available at: <https://portswigger.net/web-security/cross-site-scripting> [Accessed 5 May 2021].

Docs.djangoproject.com. 2021. *Django documentation | Django documentation | Django*. [online] Available at: <https://docs.djangoproject.com/en/3.2/> [Accessed 6 March 2021].

6.0 Appendices

6.1. Project Plan



1		📌	Create the skeleton web application	6 days	Mon 09/11/2	Sun 15/11/2(	
2		📌	Find a suitable server/database for encription	3 days	Mon 16/11/20	Wed 18/11/20	1
3		📌	Impliment transaction with the database/server	7 days	Thu 19/11/20	Fri 27/11/20	2
4		📌	Requirement specification	2 days	Mon 30/11/2	Tue 01/12/2(	3
5		📌	Development of basic web application to a near complete state	44 days	Wed 02/12/20	Sun 31/01/21	4
6		📌	Implimenting payment system	21 days	Mon 01/02/2	Sun 28/02/2:	5
7		📌	Penatration testing	11 days	Mon 01/03/2	Sat 13/03/21	6
8		📌	Resolving security issues	14 days	Mon 15/03/2	Thu 01/04/2:	7
9		📌	Final debugging & polish	21 days	Fri 02/04/21	Fri 30/04/21	8

6.2 Project Proposal

National College of Ireland

Project Proposal

Secure Property Management (SPM)

06/11/2020

BSc (Honours) in Computing

Cybersecurity

2020/2021

Josh O’Regan

X17521289

X17521289@student.ncirl.ie

1.0 Objectives

The main objective of this project is to provide a secure, efficient and legally compliant system for managing rental properties remotely. collecting and storing data this kind of data remotely and securely is much harder in person due to the ongoing pandemic. It will be focused on landlord and tenant relations and ensuring the communication, documentation and payments are collected and stored securely via encryption.

Firstly, the web application will ensure that the landlord can request and send documentation to the prospective renter. The renter can then return the documentation via the web application. There will be encryption implemented to ensure that the data is only accessible by the two parties. There will also be GDPR standards applied to the files at the time of upload.

Secondly, the web application will allow for the management of current properties and tenants. The main aspect of this will be the secure payments that will be implemented via the application. It will allow the landlord to view the payment history and amount outstanding per tenant and per property. The landlord will be able to make manual adjustments to the amount outstanding if payment is received in cash via a secure superuser login.

Thirdly, the web application will show a record of all files, documentation and payments regarding each tenant to the landlord, this data will be stored securely on a database. When it comes a time where the tenant leaves the property there will be an automated timer started to ensure the data is removed as per GDPR timelines. There will also be a facility for the tenants to request a copy of their data.

2.0 Background

The inspiration for this project came from the effects of the Covid 19 pandemic. With the sudden change in how business was conducted all industries needed to adapt, I witnessed first-hand how different sectors adapted to this change via my internship. One of the most major challenges faced was with industries that would usually conduct business dealings face to face. Engineering and construction firms had some leeway given to them by the government as did healthcare. One industry I seen struggle tremendously was the rental and property management sector. The collection of documents and payments became a huge issue, they could of course be sent through informal means but these methods (eg: email) did not meet any kind of security standards. After an email breach it become apparent that one successful phishing attempt would compromise countless numbers of tenant details and sensitive documentation. It was also a logistical nightmare to organize and categorize files per tenant and per building.

I identified this sector as one that needed a great amount of development in regards to cybersecurity. Renting is something that most people will have to do in their lives, as times change and people would prefer to stay distanced, the technology must change also. Never before has a such a need for a secure document and payment transaction system been so apparent. Government regulation will also make person to person interaction difficult for the foreseeable future, this along with the move towards working from home will make such an application appealing to those who wish to work at home or from different locations.

In a traditional setting, agreements would be signed in person and the tenant could meet the landlord at one of their properties to arrange paperwork. Existing tenants would either pay their rent in cash or arrange a bank transfer. This application will allow for a homogenised, secure solution for onboarding new tenants as well as managing existing tenants. Tenants will be able to pay their rent in cash still if they so wish, the landlord can manually change the rent outstanding via their login. In order to ensure security the landlord will have to use a two factor login method, this is because of the volume of information the landlord will have access to as well as their access to the records when it comes to payments.

Email as a communication method works quite well, however after seeing how easily a compromised email can lead to a data breach it is clear why a secure method for sending and receiving this documentation. This was the main inspiration for this project.

3.0 Technical Approach

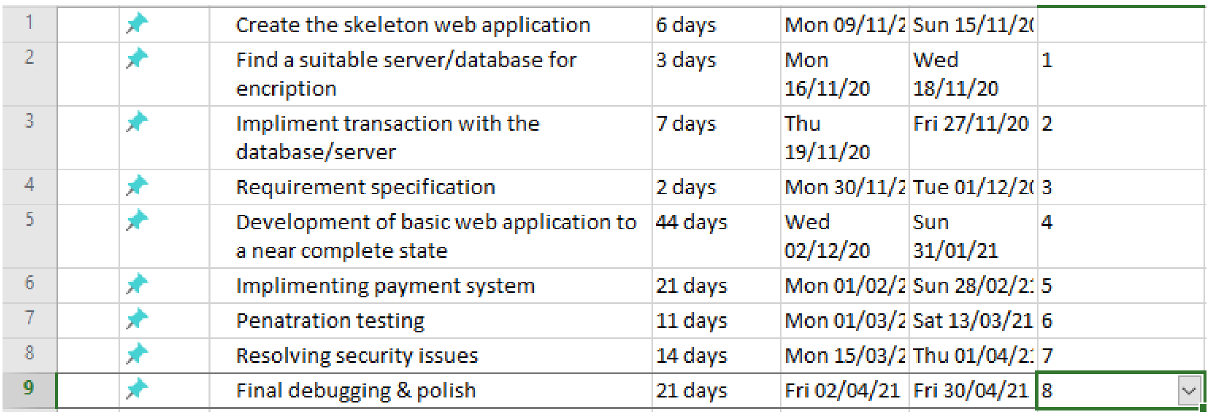
I captured the main requirements for this project from the source. I referred back to the knowledge I gained in my internship about how many of our property and asset management clients dealt with their tenants. I found that while some use more up to date methods, most relied heavily on a legacy pen and paper system, some implementing an excel spreadsheet at times.

The main requirements I gathered were for the application to be secure from end to end, two factor authentication or similar to be implemented for landlord users, ease of access via a web portal and for payment to be able to be taken securely via the application instead of having to wait several days for a bank transfer.

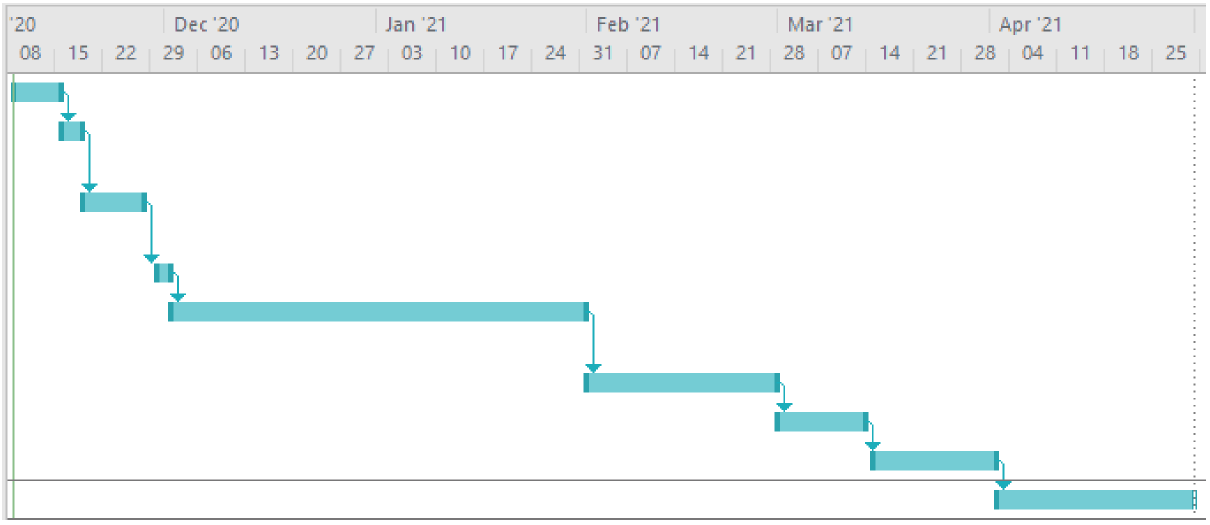
Having only conceived this new idea over the past few days I have had limited time to explore the technical approach for these requirements, but of that papers I have read I must say that encryption via an external server seems the best way to go about the transactions. Ensuring that authentication to this server is kept to a high standard will be the top priority.

In terms of payments I will be looking to develop a secure payment system via the external server. This will cut out any reliance on third party payment services and of course decrease overheads in a business situation.

4.0 Project Plan



1		📌	Create the skeleton web application	6 days	Mon 09/11/2	Sun 15/11/2(	
2		📌	Find a suitable server/database for encription	3 days	Mon 16/11/20	Wed 18/11/20	1
3		📌	Impliment transaction with the database/server	7 days	Thu 19/11/20	Fri 27/11/20	2
4		📌	Requirement specification	2 days	Mon 30/11/2	Tue 01/12/2(	3
5		📌	Development of basic web application to a near complete state	44 days	Wed 02/12/20	Sun 31/01/21	4
6		📌	Implimenting payment system	21 days	Mon 01/02/2	Sun 28/02/2:	5
7		📌	Penatration testing	11 days	Mon 01/03/2	Sat 13/03/21	6
8		📌	Resolving security issues	14 days	Mon 15/03/2	Thu 01/04/2:	7
9		📌	Final debugging & polish	21 days	Fri 02/04/21	Fri 30/04/21	8



5.0 Technical Details

- React – I will be using React as the frontend development language, It allows for the web application to be easily translated to a mobile or desktop counterpart.
- Visual Studio Code – The main development platform I am using for this project, allows for integrations and debugging.
- Python – This will be used for some of the more complex language processing, first time using this language. Will be using the PassLib package.
- Firebase – The current backend database used at this level of the project, could change as project progresses.
- Balsamiq – Used to for the purpose of wireframes and conceptualising code.
- SQL – Will be using the SQL language to allow for interaction between the front and back end.

6.0 Evaluation

I plan to integrate real world data into the application in order to test it. In order to avoid any serious data breaches in the development stage I will use placeholder word documents and PDFs to test the security of the transaction. In terms of payments, I will test the send and receive function via my own funds once it is fully implemented.

Once I feel the application is secure and penetration testing has been conducted, I will transact payments, documentation and user information that is fabricated but closely mirroring the actual information planned. I will follow my use cases closely to ensure that all use cases are accounted for and tested. If the time arises wherein a potential client wishes to use the application, I will consult the ethics form but at the moment I have no intention or no belief that this will arise.

6.3 Monthly Reports

Reflective Journal

October 2020

During the month of October, I took the first steps in finding a project idea, I attended the lecturers relating to how to format the documentation, the requirements in regards to technologies and what is expected in terms of complexity. The next step was composing and submitting a project proposal. For this I wrote up a script of points a wanted to cover. Once I had this composed, I recorded it via a teams call and submitted it via Microsoft Stream.

It was difficult to compose a project idea based on cybersecurity with relatively little experience in the matter, the idea has to be approved or vetoed so I will await this before I start any major work on the project. The plan for the next month is to start the proposal, get the development rolling and set up a Microsoft Project file with all of the objectives over the year. Once I have the roadmap composed and structured I will be able to get the ball rolling on development.

Reflective Journal – November

Josh O’Regan – x17521289

During the month of November, I began practical work on the project itself. I started work on the barebones web application and started testing different backend implementations and features. All of this work was aimed towards my eventual midpoint submission for my final year project. I refreshed my memory in regards to front and back end web development by developing the prototype.

In December I plan to further expand on the development of the prototype and finish it for the mid point demonstration. I also plan on working on developing the technical documentation for the mid point submission. Time management will be key with all the other projects due in the coming month.

I have achieved what I wanted to from last months journal and hope I can continue to stay on track for next month!

Reflective Journal – December

Josh O’Regan – x17521289

The month of December was mostly spent developing my documentation and prototype with the goal of submitting it for the mid-point. The main effort was focused on developing the technical document with all the necessary technical information and diagrams. I felt that this helped greatly when further implementing features. Fleshing out use cases and technical details on paper helped me better conceptualise what I was aiming for and therefore focused me greatly when it came to further developing the prototype.

In January to I plan to further increase the development of the prototype into the proposed goals as per the project plan. I plan to increase my learning on the technologies I plan to use over the break after my exams for this purpose.

I am still on track and have achieved my goals that I set in November, however this was not easy due to the sheer amount of deadlines this month.

Reflective Journal

January 2020

During the month of January I focused on improving the implementation of my project, ensuring I had a solid foundation work on towards continued development of features. To achieve this I continued my learning of the technologies that make up the project and also researched possible methods of integrating all of these together. At the moment I am adding the features one by one, ensuring that they are fully functional and bug free before deciding to implement another. It was hard to find the time to do this between the TABA, and exams we had, aswell as the interviews I partook in over the month. Self-management and timekeeping were essential to achieve the goals I set myself but I the end I managed to achieve my goals for January. In February I hope to implement more features and to reach the point where I have the majority of the application working and in a functional state. There will be issues in integrating the technologies with each other no doubt but I am confident with a unit testing based approach at first, they can be overcome. Looking at the Gantt chart I am still well on track for the deadline.

Reflective Journal

February 2020

The month of February proved the most dynamic month of the project thus far. The implementation of the backend via Python was proving difficult to combine with the React frontend. In order to stop the headaches being caused by the interaction of these two systems I decided to work with a framework (Django) to greatly improve compatibility. Choosing Django is proving an excellent decision for the full stack nature of this application, the security features provided are exceptional and are working very well in combination with my own original security features. Python was always going to need to be the backend in order for the document processing and encryption element of the application. In essence I am now currently adapting my React code to work in tandem with the Django framework. This provides a great opportunity of segmentation of the frontend and the backend web servers, improving overall security of the application. In the month of march, I hope to complete this transition and do full tests on every aspect, from unit to system testing. The change to the Django framework has been a setback but it will prove invaluable in making the project much more complete and much more secure.

Reflective Journal

March 2020

During the month of March, I began to polish the remainder of the application and began testing. I will lay out the full scale of the testing in the technical document but in essence I followed testing methodology to the largest extent possible. This included running tests from unit tests to other types such as class, integration and system testing and checking for any security issues by attempting to penetration test my application. The nature of the environment means that it is naturally quite secure. From the development perspective most of the major work to be done has been done and the documentation of such work has started, I am planning to build upon the template that I used at midpoint and to add much more information in regards to the challenges I have faced thus far in development, especially my pivot to the Django framework and the changes I have made to the original application proposal due to emerging technologies being discovered for implementation. Using python to read and display text in PDFs for example was a massive aspect of the project that evolved along with the research done during the course of development. In April I hope to continue testing and do more work on the documentation.

Reflective Journal

April 2020

During the month of April I continued testing the application and making minor tweaks based on my findings. The application itself is now in a near complete state so the focus was mostly on documentation. Based on feedback I received during my midpoint submission I overhauled my use case diagrams. On all other sections that I had previously completed I am updating with new information as per what has changed and improved during the development process. The documentation itself is taking shape with references from relevant sources and research into the background of the project, the components that are used and why those components were decided to be used. Diagrams naturally now can be expanded upon greatly and much more accurately reflect the current state of the application. Great effort is being put into looking back at previous journals in order to properly document the changes in the application. I expect the document to be completely finished by early May. After this I will revise the document and make any grammatical or editorial changes to fit the specification laid forth in the marking rubric. After this is completed I will move forward with creating the presentation and all other requisite uploads such as the poster.