National College of Ireland

Bachelor of Science (Honours) in Computing Information

Cyber Security

2023/2024

Luke Banahan

20116471

x20116471@student.ncirl.ie

HashHub

Technical Report

Contents

Executive Summary

The purpose of this report is to document the planning and development of the HashHub application. HashHub is a web application that uses applied cryptographic techniques to generate digital certificates. The application acts as a platform that allows users to upload documents so that they can be signed with digital signatures. The application will allow users to share documents with other users, who can use the application to verify the digital signature with the click of a button.

The following report shows the documentation of the development process, justification for development decisions and technologies, requirement definitions and testing processes as well as results.

1 Introduction

1.1. Background

As the world continues to move further and further into the digital world, and physical documentation becomes less prevalent, the need for digital document management is increasing. This, combined with the fact that daily business activities are being done remotely, means that reliable methods of ensuring the authenticity and integrity of documents is becoming more important.

As a cyber security student, the area of applied cryptography always interested me. I undertook this project to attempt to solve the problem of digital document management, while using digital signatures to ensure their authenticity and integrity.

During my work experience with the Revenue Commissioners [1], I gained some valuable experience working on large, robust, and highly scalable web applications developed within the Spring Boot Framework [2]. I also had the opportunity to attend some seminars and gain some valuable knowledge about developing web and mobile applications with accessibility in mind.

I chose to develop this project to combine my interest in applied cryptography with my experience with Java, Spring Boot, and accessibility development to solve the problem of digital document management and document integrity.

1.2. Aims

This project aims to create a platform where users can create an account, log in and sign or verify documents digitally. I undertook this project to showcase one of many strong use-cases of applied cryptography. Digital Signatures provide superior security to paper signatures as they have a built-in anti-tamper mechanism which is takes place in the document verification process. Digital signatures also help improve the efficiency and productivity of the typical document signing process, which traditionally involves both parties having to be present in the same room for non-repudiation purposes. By allowing users to sign documents remotely, not only are costs of transport and storage of physical documents cut, but it is also more eco-friendly as it is paperless. Moreover, HashHub will also store these documents, allowing for the streamlining of document management.

1.3 Market Analysis & Innovation/Novelty

During my market analysis of other products that provide similar services such as DocuSign [3], PandaDoc [4] and SignNow [5], I noticed that their web applications were not accessible to users with disabilities who are required to use screen readers or other similar devices.

During my work placement with Revenue Commissioners, I attended multiple seminars regarding website and mobile application accessibility, including people with disabilities. As Revenue Commissioners are a public body, they are subject to the EU Web Accessibility Directive, [6] which aims to ensure that all public sector websites and mobile applications are accessible to people with disabilities. This allowed me to gain an insight to the importance of accessibility in websites and mobile applications, as well as gain some valuable experience in developing accessible applications.

My application aims to separate itself from the competitors such as DocuSign, PandaDoc and signNow, but ensuring that all aspects of the application are developed with accessibility features that allow people with disabilities to use the application. My application will **innovate** by providing enhanced accessibility features.

1.4 Technology

The table below is a brief outline of the technologies used to develop this application.

Technology	Purpose
Spring Boot	Framework
IntelliJ IDEA	Integrated Development Environment (IDE)
PostgreSQL	Database
Java	Primary Programming Language
HTML (Hyper Text Mark-Up Language)	Creating Web Pages
CSS (Cascading Style Sheets), Bootstrap **[7]**	Styling Web Pages
JavaScript	Adding functionality and interactivity to web pages.
Thymeleaf	Dynamically displaying live data from database.
SHA-256	Hash Function
RSA Algorithm	Encryption and Key-Pair Generation

1.1.1 Spring Boot

Spring Boot is the Framework I chose for to develop this application. Spring Boot is an Open-Source Java Framework that allows for rapid development because of its emphasis on convention-over-configuration. [2] This allowed me to reduce time setting up the prerequisites for the application and get straight into the development of requirements.

Spring security comes pre-packaged with Spring Boot. Spring Security will be used for user authentication, authorization, access control and the safe storage of sensitive user information such as passwords.

Spring Boot allows for extensive library support by adding dependencies to the pom file. This will allow me to integrate key functionalities in the application such as dynamically displaying information from database to the user and calling in hashing functions and the RSA algorithm to generate key pairs.

Spring Boot's robust security features and vast library support, as well as my prior experience working with the framework during my work placement with the Revenue Commissioner, made it the most suitable choice for developing the HashHub application.

1.1.2 IntelliJ IDEA

IntelliJ IDEA is an IDE developed by JetBrains [8]. It has excellent support and built-in tools for both Java and Spring Boot. The built in developer tools make it easy to test, debug and run applications. The database tools also make database management, manipulation, and navigation very intuitive. I also have some experience with this IDEA from my work

placement at the Revenue Commissioner, so I am familiar with the tool. For the reasons mentioned above, IntelliJ IDEA will be used to develop this application.

1.1.3 PostgreSQL [9]

In the initial proposal, SQLite3 was planned to be the database used for this application. However, after further research and development, I decided to switch to PostgreSQL. PostgreSQL, like SQLite3 is a relational database management system.

Upon further research, I discovered that PostgreSQL is more reliable and can handle larger volumes of data more effectively. This will be a very important component of my application as it scales, PostgreSQL will allow for securely storing larger amounts of data.

PostgreSQL also integrates seamlessly with Spring Boot and IntelliJ and there is a built-in feature in the IntelliJ IDEA that allows for database management and visualisation straight from the IDE.

1.1.4 Java

Java was chosen for the primary programming language to develop this application due to its versatility, scalability, and robustness. Java and Spring Boot have an extensive selection of libraries that can be called upon, which make it suitable for developing the HashHub application. This, combined with the fact that Java is the language that I have the most experience developing with are the reasons Java was chosen for this application.

1.1.5 HTML, CSS, Bootstrap and JavaScript

HTML will be used to create the templates and web pages for this application. A combination of custom CSS and imported Bootstrap will be used to style these pages. JavaScript will be used to add functionality and make these pages interactive.

1.1.6 Thymeleaf

Thymeleaf is a server-side Java template engine. This will be employed to generate and display information from the database to the user dynamically. [10]

1.1.7 SHA-256

SHA-256 hashing algorithm will be used to create a hash of the documents for the digital signature generation and verification process. SHA-256 was chosen as it is cryptographically secure, irreversible, and not prone to collision. While other hashing algorithms such as SHA-512, may provide greater security, I chose SHA-256 as the trade-off for performance and memory was too great for somewhat negligible enhancements in security. As I will already be using Spring Security to authenticate and authorise users before they sign or verify documents. SHA-256 hashing algorithm combined with authentication and authorisation implementations will be enough to ensure non-repudiation for the digital signatures. SHA-

256 will be imported into the application along with Spring Security, which is a dependency for my application that comes installed with Spring Boot.

1.1.8 RSA

RSA will be used to generate the key pair during for encryption during the digital signature generation/verification process. RSA was chosen because of its robust security, performance efficiency and the ability to change the key length to suit the performance needs of my application. RSA will also be imported into the application along with Spring Security.

1.5 Structure

The structure of the remainder of this document is as follows.

Section 2: System

This section will describe all requirements definitions of the application. It will also outline the system architecture, describe the implantation, explain what can be seen in on each of the screens displayed by the application's graphical user interface (GUI). The end of this section will document the testing and evaluation of the system. It will describe the types of tests carried out and show evidence of test performance and results.

Section 3: Conclusion

This section will summarise the development of the application and evaluate the strengths and weaknesses of the system.

Section 4: Further Development or Research

This section will address the direction that this application could take if given more time and resources.

Section 5: References

Any references in the documentation will be shown here.

Section 6: Appendices

Appendices and supporting documentation will be attached here.

2 System

2.1 Requirements

2.1.1 Functional Requirements

2.1.1.1 Use Case Diagram

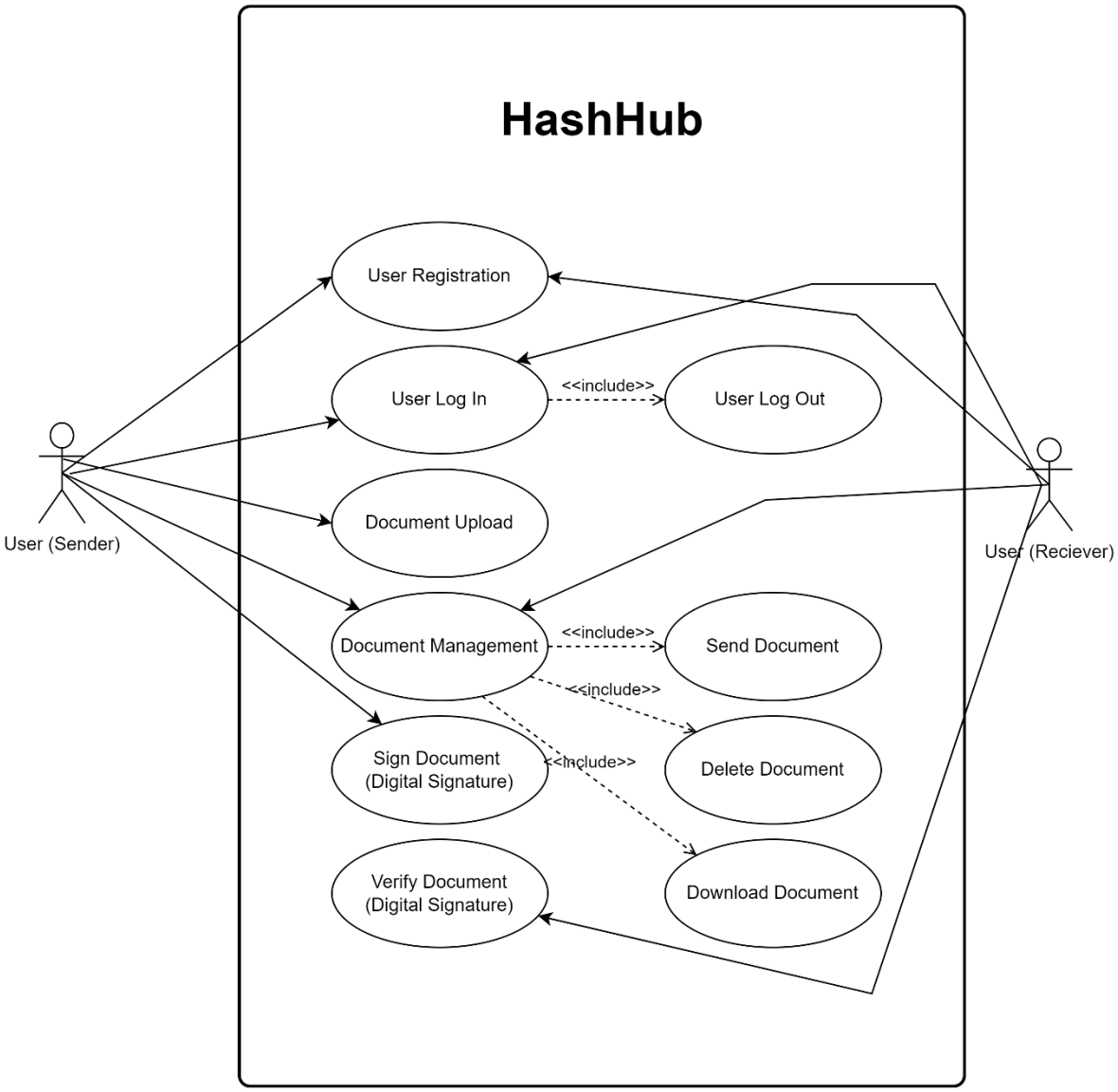


Figure 1: Use Case Diagram of Hashub System

Functional Requirements

FR1: User Registration

FR2: User Log In

FR3: User Log Out

FR4: Document Upload

FR5: Send Document

FR6: Delete Document

FR7: Download Document

FR8: Sign Document

FR9: Verify Document

2.1.1.2 FR1 User Registration

- Description & Priority

This requirement allows the user to register an account with the application. Users must register with a Username and Email Address that are unique. Passwords must be at least 8 characters long and contain at least one letter and one digit. Passwords will be stored securely in database. Bcrypt algorithm will be used to hash and salt passwords. This requirement is of critical priority as having an account is required to log in and access the application's functionality.

Description/Scope	User can register an account with application. This will allow user to log in to the application and access the core functionality of the application.
Pre-Condition	The user navigates to the web application address and is redirected to the login page.
Activation	The user navigates to the Register page via the "Register Here" button.
Main Flow	1. User enters unique username. 2. User enters email address. 3. User enters a password that is at least 8 characters long containing at least 1 letter and 1 digit. 4. User enters matching password in the "Confirm Password" field.

	5. User is informed of successful registration and is redirected to the log in page.
Alternate Flow 1: Required Field Missing	1. User leaves a required field empty when registering. 2. System notifies user which field is missing. 3. System picks up from step 1 of the main flow.
Alternate Flow 2: Username or Email is already registered	1. User attempts to register with a username or email that is already registered in the database. 2. System notifies user. 3. System picks up from step number 1 of main flow.
Alternate Flow 3: Passwords do not match or incorrect format	1. User enters a password that does not pass the validation. 2. System notifies user that password must contain at least 8 characters with at least 1 letter and 1 digit. 3. System picks up from step 1 of main flow.
Termination	1. System redirects user to log in page.
Post Condition	User logs in with registered details.

2.1.1.3 FR2: User Login

- Description and Priority

This requirement allows the user to log in to the application with their registered details. Only logged in users can access the other pages where the key functionality of the application is carried out. Therefore, this is of critical priority to the application.

Description/Scope	The user logs into the application with their own personal registered details.
Pre-Condition	The user has previously registered an account.
Activation	The user navigates to the login page.
Main Flow	1. User enters correct username. 2. User enters correct password. 3. User clicks login button. 4. System checks that username is correct.

	5. System checks that password is correct. 6. System notifies user that login was successful. 7. System redirects user to dashboard page.
Alternate Flow 1: Incorrect Username or Password	1. User enters incorrect username or password. 2. System checks that username and/or password is correct. 3. System notifies user that username or password is incorrect. 4. System picks up from step 1 of the main flow.
Termination	User is logged in.
Post Condition	System redirects user to log in page.

2.1.1.4 FR3: User Log Out

- Description and Priority

This requirement allows users to log out of their current session. This is of crucial priority for the application.

Description/Scope	The user logs out of their current session with the application.
Pre-Condition	The user is registered and logged into their account.
Activation	The user clicks the log out button.
Main Flow	1. User clicks the log out button. 2. User is signed out and notified that log out was successful. 3. User is redirected to the login page and is locked out from accessing any other pages until they sign back in.
Termination	User is logged out.
Post Condition	System redirects user to log in page and notifies them that they have been logged out.

2.1.1.5 FR2: Document Upload.

- Description and Priority

This requirement allows the user to upload a document from their computer to be stored in the database. This requirement is a prerequisite for the document signing and verifying requirements, it is therefore of critical importance to the application.

Description/Scope	A user uploads a document/file from their machine to be stored on the application database.
Pre-Condition	The user is registered and logged into their account.
Activation	The user navigates to the 'Document Upload' page.
Main Flow	1. User navigates to the document upload page. 2. User clicks 'Upload' button. 3. User selects file to be uploaded. 4. Document is uploaded and stored in database. 5. System notifies user that document was successfully uploaded.
Alternate Flow 1: Document upload fail.	1. User navigates to document upload page. 2. User clicks 'Upload' button. 3. User selects file to be uploaded. 4. An error occurs and the document could not be uploaded. 5. System notifies the user that the document uploaded failed. 6. System picks up from step 1 of the main flow.
Termination	The document is successfully uploaded.
Post Condition	The user navigates away from the document upload page.

2.1.1.6 FR5: Send Document

- Description and Priority

This requirement will allow the user to share a document that they have uploaded to the system with another user. This requirement is of critical importance to the application as sharing the document is a necessary step in the signing and verification process of the document.

Description/Scope	This use-case allows a user to share a document that they have uploaded with another user.
Pre-Condition	The user has registered an account, is currently logged in and have at least one document already uploaded into the system.

Activation	The selects the document they wish to send, clicks the “Send Document” button, and enters the email address of the user they would like to send the document to.
Main Flow	1. User navigates to the “My Documents” page. 2. User selects the document they would like to send via radio button. 3. User clicks “send document”. 4. User is prompted with relevant information to the document they want to send in a pop-up screen and is prompted to enter the recipient’s email address. 5. User enters recipient email address. 6. User clicks “Send Document” button on prompt page. 7. System shares the document with the recipient. 8. System sends an email notification to the recipient to let them know they have received a new document. 9. System prompts user that the document has been sent successfully. 10. System redirects to the “My Documents” page.
Alternate Flow: User chooses not to send document.	1. User navigates to the “My Documents” page. 2. User selects the document they would like to send via radio button. 3. User clicks “send document”. 4. User is prompted with relevant information to the document they want to send in a pop-up screen and is prompted to enter the recipient’s email address. 5. User decides not to send the document and clicks “close button” instead. 6. User is redirected to “My Documents” page.
Termination	System redirects user to the “My-Documents” page.
Post Condition	System awaits user input.

2.1.1.7 FR6: Delete Document

- Description and Priority

This requirement allows the user to deleted documents that they have previously uploaded from the system. This requirement is of high priority as it allows the user to delete documents they do not want stored in the system, which helps declutter their UI, while also keeping unwanted files from filling up the database.

Description/Scope	User can delete documents they have previously uploaded.
Pre-Condition	User has registered, is currently logged in, and has already uploaded at least one document.
Activation	User clicks “Delete Document” button.
Main Flow	1. User navigates to the “My-Documents” page. 2. User selects document they wish to delete via radio button. 3. User clicks “Delete Document” button. 4. User is prompted with a pop-up screen containing relevant information to the document they have selected and are asked “Are you sure you want to delete this document?” 5. User clicks “Delete Document” button on prompt page. 6. System deletes document from database. 7. System notifies user that document was deleted successfully. 8. User is redirected to “My Documents” page.
Alternate Flow 1: User chooses not to delete document	1. User navigates to the “My-Documents” page. 2. User selects document they wish to delete via radio button. 3. User clicks “Delete Document” button. 4. User is prompted with a pop-up screen containing relevant information to the document they have selected and are asked “Are you sure you want to delete this document?” 5. User clicks “No” button on prompt page. 6. User is redirected to “My-Documents” page.

Termination	User is redirected to “My-Documents” page.
Post Condition	System awaits further user input.

2.1.1.8 FR7: Download Document

- Description and Priority

This requirement allows the user to download a document that they have uploaded previously or download a document that has been shared with them. This requirement is of medium importance to the application. It is a nice feature to have, however it is not critical to the core functionality of the application.

Description/Scope	The user downloads a document from the application.
Pre-Condition	The user has registered an account, is currently logged in, and has as least one document uploaded to the system or at least one document shared with them.
Activation	User clicks “Download Document” button.
Main Flow	1. User navigates to “My-Documents” page. 2. User selects document they wish to download. 3. User clicks “Download Document” button. 4. Pop up screen appears with relevant information about the document and asks the user “Are you sure you wish to download this document?” 5. User clicks “Download Document”. 6. System retrieves document from the database. 7. User downloads document from system. 8. User is redirected to the “My-Documents” page.
Alternate Flow 1: Download a shared document.	1. User navigates to the “Documents Shared with Me” page. 2. User selects document they wish to download. 3. User clicks “Download Document” button. 4. Pop up screen appears with relevant information about the document and asks

	the user “Are you sure you wish to download this document?” 5. User clicks “Download Document”. 6. System retrieves document from the database. 7. User downloads document from system. 8. User is redirected to the “Documents Shared with Me” page.
Termination	User is redirected to respective page.
Post Condition	System awaits further input from the user.

2.1.1.9 FR8: Sign Document

- Description and Priority

This requirement allows a user to sign a document digitally. This is of utmost importance to the application and is of critical priority. This is a core functionality of the application.

SHA-256 will hash the original document the user wants to sign, then RSA algorithm will generate a key pair and encrypt the hashed document with the signer’s private key. The hashed and encrypted document is used as a digital signature. Hashing the document and encrypting it with the signer’s private key ensures security and non-repudiation. This is because the hashed value of the document represents the unique value of the content in the document. The encrypted hash value of the document can then be used as a secure digital signature and is attached to the original document to produce a digitally signed document.

Description/Scope	The user can sign an uploaded document with a digital signature.
Pre-Condition	The user has registered an account, is currently logged in, and has at least one document uploaded.
Activation	The user selects the document they wish to sign and clicks the “Sign Document” button.
Main Flow	1. User navigates to the “My Documents” page. 2. User selects document they wish to sign. 3. User clicks “Sign Document” button. 4. The system generates a digital signature for the document and stores it in the database with the document.

	5. The system notifies the user that the document was signed successfully.
Exceptional Flow 1: Digital Signature generation failure	1. User navigates to the “My Documents” page. 2. User selects document the wish to sign. 3. User clicks “Sign Document” button. 4. The system fails to sign the document. 5. The system notifies the user that “The document could not be signed at this time, please try again later”.
Termination	The user is redirected to the “My-Documents” page.
Post Condition	The system awaits further input from the user.

2.1.1.10 FR9: Verify Document

- Description and Priority

This requirement allows a user to securely verify a digitally signed document that has been shared and signed by another user. This requirement is of critical importance to the application and is a core functionality.

Description/Scope	The user can securely verify a digitally signed document.
Pre-Condition	The user is registered, is currently logged in, and has at least one signed document shared with them.
Activation	The user clicks the “Verify Document” button.
Main Flow	1. The user navigates to the “Documents Shared with Me” page. 2. The user selects the document they wish to verify via the radio button. 3. The user clicks “Verify Document” button. 4. The system securely verifies the digitally signed document. 5. The system notifies the user that the document has been verified successfully.
Exceptional Flow 1: Could not verify document.	1. The user navigates to the “Documents Shared with Me” page. 2. The user selects the document they wish to verify via the radio button. 3. The user clicks “Verify Document” button. 4. The system cannot verify the document.

	5. The system notifies the user that the document could not be verified.
Termination	The user is redirected to the "Documents Shared with Me" page.
Post Condition	The system waits for further user input.

2.1.2 Data Requirements

The system will use the database to handle data from different parts of the system. Firstly, it will handle user data from the login/register functionality of the application. It will also store data on the documents that are being signed and verified. Below are specified the user data and document data entities, attributes, and constraints.

User Table

Attribute Name	Data Type	Constraints	Key Type
user_id	bigint	Unique, Not Null	Primary
username	VARCHAR (255)	Unique, Not Null	-
email	VARCHAR (255)	Unique, Not Null	-
password	VARCHAR (255	Not Null	-

Document Table

Attribute Name	Data Type	Constraints	Key Type
id	bigint	Unique	Primary
name	varchar (255)	-	-
content	oid	-	-
email	Varchar (255)	-	-
Shared_with	Varchar (255)	-	-
Is_verfied	Boolean	-	-
Is_signed	Boolean	-	-
Size	Bigint	-	-
Public_key	Oid	-	-
Signed_on	Timestamp	-	-
Verified_on	Timestamp	-	-
Uploaded_on	Timestamp	-	-

Roles Table

Attribute Name	Data Type	Constraints	Key Type
Id	Bigint	-	Primary
Name	Varchar(255)	-	-

Users_Roles Table

Attribute Name	Data Type	Constraints	Key Type
user_id	Bigint	-	Foreign Key
Role_id	Bigint	-	Foreign key

2.1.3 Accessibility Requirements

To make this application usable for visually impaired users, all pages should be implemented with tab-index ids and titles. This is to allow users who rely on screen readers or otherwise visually impaired/disabled users to be able to use this application. Correct Tab-Indexes are extremely useful important for ensuring accessibility for users that require screen readers as it will ensure the system allows the user to navigate the fields in the correct order to use the application without a mouse. Proper descriptions and html titles to allow the screen reader describe what is in the current field, or where abouts the user is on the current screen.

2.1.4 Security Requirements

Only login/registration pages should be accessible to unauthorized users. All other parts of application should be locked off without authorization.

Passwords must be at least 8 characters long and contain at least 1 letter and 1 digit.

Passwords will hashed and salted by Bcrypt algorithm during the registration process and stored in this format.

2.2 Design & Architecture

2.2.1 High Level Overview

Spring Boot, Java with Maven are the main programming technologies and frameworks being used to develop this application. PostgreSQL is the database that will be used in this system. Below I will provide diagrams explaining how these systems will interact with each other.

2.2.2 System Architecture

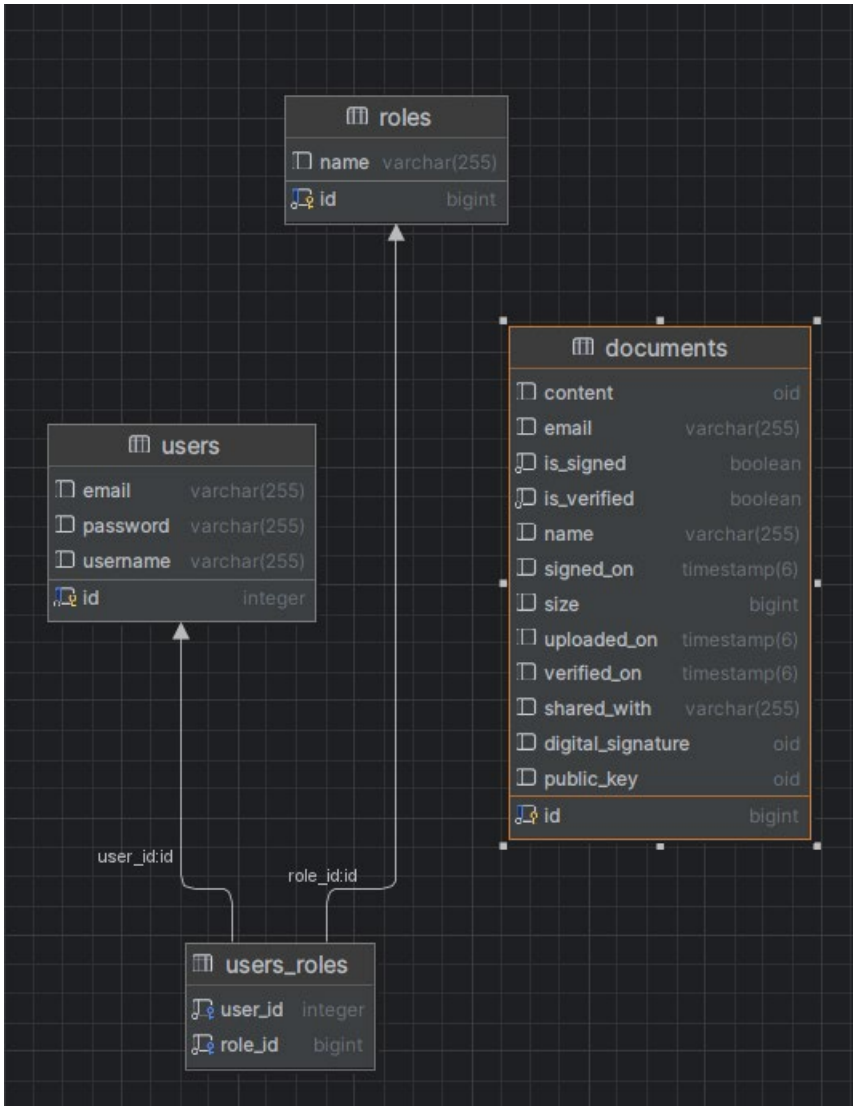


Figure 2: Database Entity Relationship Diagram

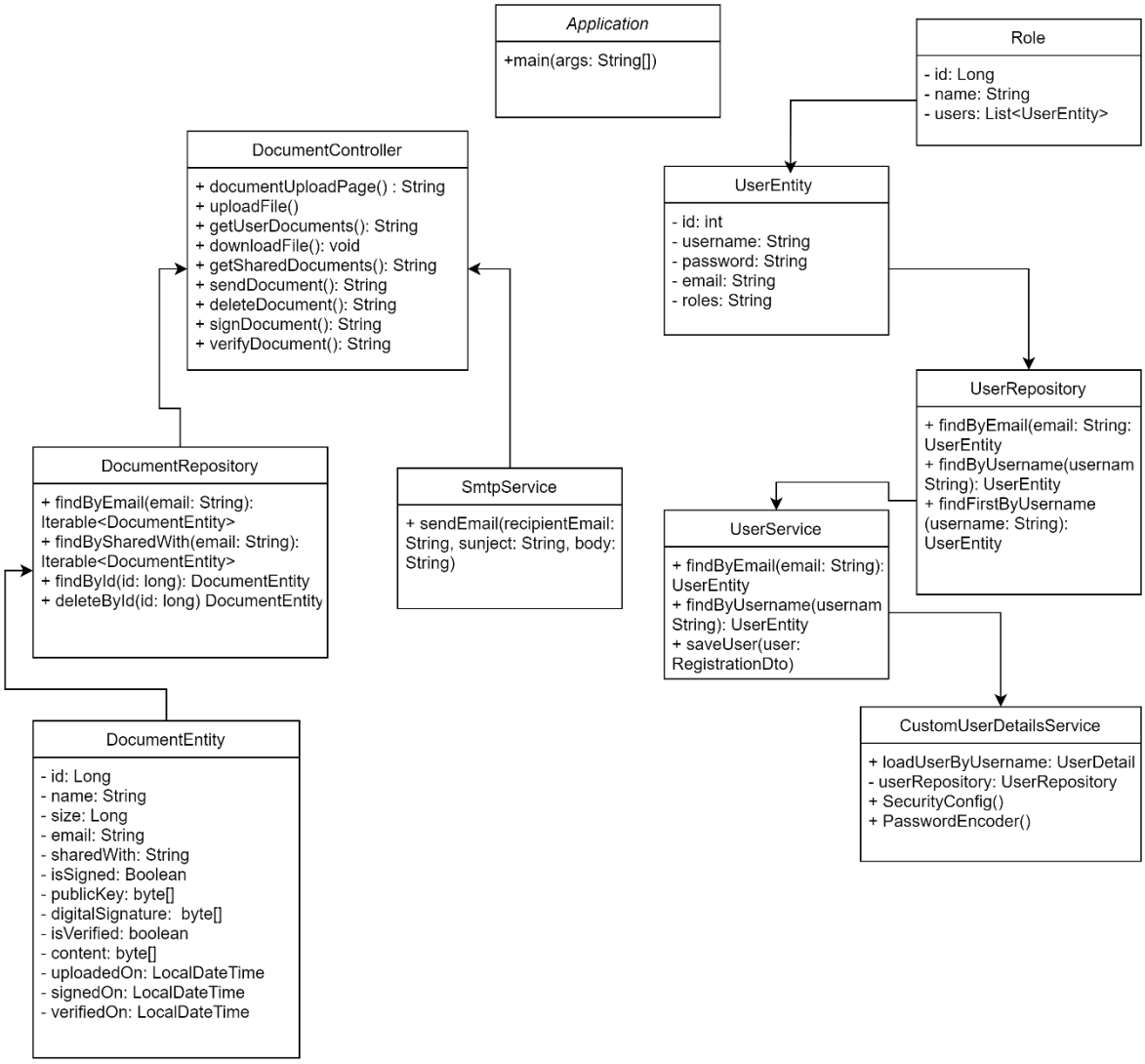


Figure 3: HashHub Class Diagram

The above class diagram shows the main classes of this application and how they are connected to each other.

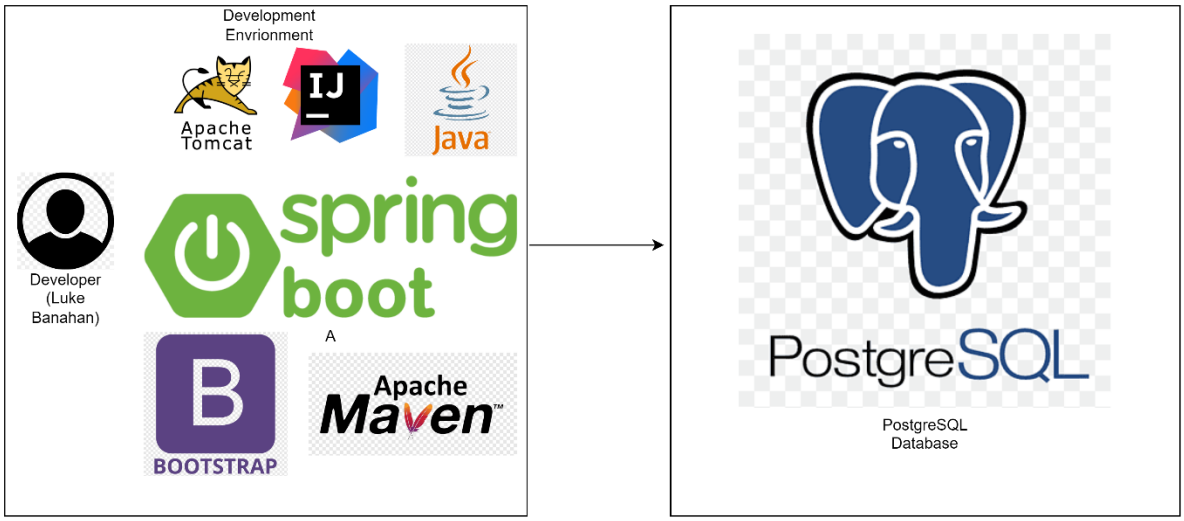


Figure 4: Technology Architecture Diagram

The above diagram shows the technologies used during the development of this application.

2.3 Implementation

```
[Luke Banahan] *
@PostMapping("/upload")
public String uploadFile(@RequestParam("document") MultipartFile multipartFile,
                         RedirectAttributes ra) throws IOException {
    String fileName = StringUtils.cleanPath(multipartFile.getOriginalFilename());


    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    if (authentication != null && !authentication.getName().equals("anonymousUser") &&  authentication.isAuthenticated()) {
        //gets authenticated email from spring security
        String userEmail = authentication.getName();

        DocumentEntity document = new DocumentEntity();
        document.setName(fileName);
        document.setContent(multipartFile.getBytes());
        document.setSize(multipartFile.getSize());
        document.setSigned(false);
        document.setVerified(false);
        document.setUploadedOn(LocalDateTime.now());

        //populates email field with current authenticated user's email
        document.setEmail(userEmail);

        repo.save(document);
        ra.addFlashAttribute( attributeName: "documentSuccessMsg",  attributeValue: "Document upload was successful.");

    return "redirect:/document-upload";
} else {
        return "redirect:/login?authFail";
    }
```

Figure 5: Document Upload Code Snippet

Document upload.

This is the method in my controller class for the document upload functionality. I request the parameter “document” from the html form that corresponds to this /upload path. I then declare a variable fileName which will be the same name as the file that is uploaded from the user’s machine.

Then I declare the authentication object from Spring security. I do this so that I can check if the user is authenticated, if the user is not authenticated there is a check that will redirect the user to the login screen with a message to tell them they must be signed in to use this page. I also do it so that I can get the current user’s email address. I store the current user’s email address in a String called userEmail. MultipartFile is an interface that is included in Spring Boot. This will handle the file upload process. I then relevant information of the document in the document entity and save it to the database. The email field in the document entity is set as the current user email. This is important for how the rest of the application will work.

```
[Luke Banahan]
@GetMapping("/user-documents")
public String getUserDocuments(Model model, RedirectAttributes redirectAttributes) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    // Check if the user is authenticated
    if (authentication != null && !authentication.getName().equals("anonymousUser") &&  authentication.isAuthenticated()) {
        // Get current user email from spring security
        String userEmail = authentication.getName();
        // Creates an iterable collection of documents and displays documents uploaded by current user email
        Iterable<DocumentEntity> userDocuments =  repo.findByEmail(userEmail);

        model.addAttribute( attributeName: "userDocuments", userDocuments);

        return "user-documents";
    } else {
        // If user is not authenticated, redirect to login page with error message
        redirectAttributes.addFlashAttribute( attributeName: "authFail",  attributeValue: "Please log in to access this page.");
        return "redirect:/login?authFail";
    }
}
```

Figure 6: Get User-Documents Code Snippet

Load “My-Documents” Page

The My Documents page shows all documents uploaded by the current user who is signed in. To achieve this, I declare the authentication object from Spring Security. I do this to make sure that unauthenticated users can not reach this page. I store the current user email address in a String variable called userEmail. I then use the Iterable interface to iterate through all of the document entities with the current user’s email address. On the front-end I use the Thymeleaf dependency to allow me to dynamically show information from the database.

```
[Luke Banahan]
@GetMapping("/shared-documents")
public String getSharedDocuments(Model model) {
    //get current user email from spring security
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    if (authentication != null && !authentication.getName().equals("anonymousUser") &&  authentication.isAuthenticated()) {
    String userEmail = authentication.getName();

    // creates an iterable collection of documents and filters by current user email
    Iterable<DocumentEntity> sharedDocuments =  repo.findBySharedWith(userEmail);

    model.addAttribute( attributeName: "sharedDocuments", sharedDocuments);

    return "shared-documents";
} else // If user is not authenticated, redirect to login page
        return "redirect:/login?authFail";
    }
```

Figure 7: Get Documents Shared with Me snippet.

Documents Shared with Me Page

The documents shared with me page works much the same as the My Documents page, but instead of showing documents uploaded by the current user, it shows documents that have been shared with that current user.

```
[Luke Banahan] *                                                                                    6  3
@PostMapping("/send-document")
public String sendDocument(@RequestParam("selectedDocumentIdValue") String id,
                           @RequestParam("recipientEmail") String recipientEmail, RedirectAttributes ra){
    long documentId = Long.parseLong(id);
    DocumentEntity documentEntity = repo.findById(documentId);

    if (documentEntity != null) {
        documentEntity.setSharedWith(recipientEmail);
        repo.save(documentEntity);
    }

    String subject = "New Document Shared With You.";
    String body = "Dear " +recipientEmail+ "," +"\n\nA new document has been shared with you on HashHub. \nPlease sign in to review/sign documents shared with
     you. \n\nMany Thanks, \nHashHub Team";

    smtpService.sendEmail(recipientEmail, subject, body);

    if (documentEntity != null) {
        ra.addAttribute( attributeName: "sendSuccess",  attributeValue: true);
        return "redirect:/user-documents";
    } else {
        ra.addAttribute( attributeName: "sendFail",  attributeValue: true);
        return "redirect:/user-documents";
    }
}
```

Figure 8: Send Document Code Snippet

Send Document

This method handles the Send Document use case. First I request the parameters selectedDocumentIdValue and recipientEmail from the front end. This selectedDocumentIdValue is what the user has selected with the radio button on the

documents table. As this ID is being parsed from the front end it must be parsed from a string to a long, as in the database the documentId is a long. I then use this id to find the selected document id in the database.

I then do a null check to make sure that the document has indeed been retrieved from the database, I then set the recipientEmail variable in the database as the email address that the user has entered to send the document to. This is important for how the document will be shared with the user in the Documents Shared with me screen.

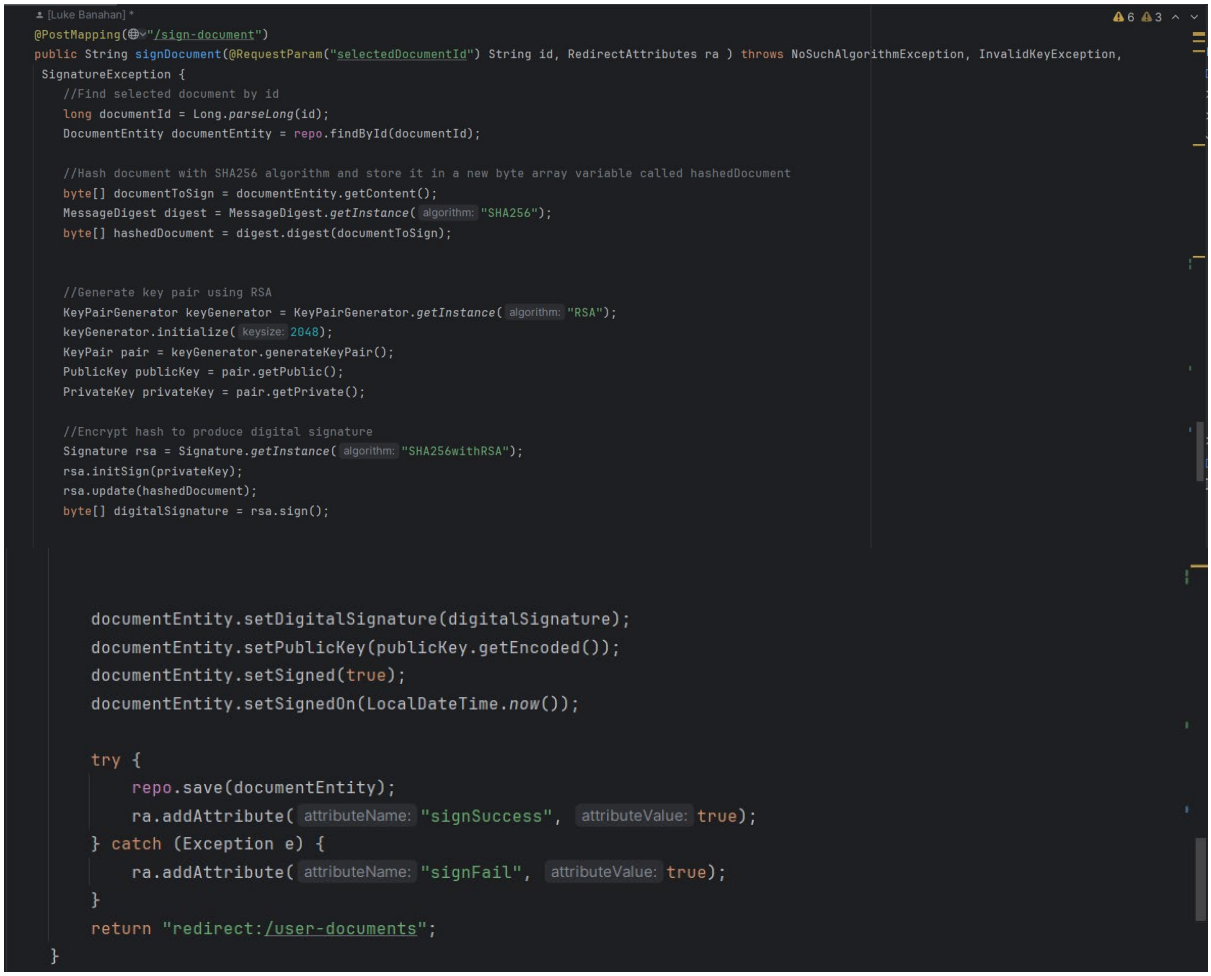
I then call on the SMPT service that I have created in my services package. I set the subject, body, and use the recipient email to call on the sendEmail class in the smptService class. If this is successful, the user is redirected to the same page with a success method. If it fails, the same is true with an error message.

```
2 usages  [Luke Banahan] *
@Service
public class SmtpService {
    @Autowired
    private JavaMailSender mailSender;

    1 usage  [Luke Banahan]
    public void sendEmail(String toEmail,
                          String subject,
                          String body){
        SimpleMailMessage message=new SimpleMailMessage();
        message.setFrom("hashhubsmtpservice@gmail.com");
        message.setTo(toEmail);
        message.setText(body);
        message.setSubject(subject);

        mailSender.send(message);

        System.out.println("Mail sent successfully");


    }
}
```

Figure 9: SMTP Service class snippet

Email Service

There is a dependency included in java and spring boot that handles email services. I added this JavaMailSender dependency to my pom.xml file and implemented it to be called later in my services package.



```
[Luke Banahan] *
@PostMapping("/sign-document")
public String signDocument(@RequestParam("selectedDocumentId") String id, RedirectAttributes ra ) throws NoSuchAlgorithmException, InvalidKeyException,
SignatureException {
    //Find selected document by id
    long documentId = Long.parseLong(id);
    DocumentEntity documentEntity = repo.findById(documentId);

    //Hash document with SHA256 algorithm and store it in a new byte array variable called hashedDocument
    byte[] documentToSign = documentEntity.getContent();
    MessageDigest digest = MessageDigest.getInstance( algorithm: "SHA256");
    byte[] hashedDocument = digest.digest(documentToSign);


    //Generate key pair using RSA
    KeyPairGenerator keyGenerator = KeyPairGenerator.getInstance( algorithm: "RSA");
    keyGenerator.initialize( keysize: 2048);
    KeyPair pair = keyGenerator.generateKeyPair();
    PublicKey publicKey = pair.getPublic();
    PrivateKey privateKey = pair.getPrivate();

    //Encrypt hash to produce digital signature
    Signature rsa = Signature.getInstance( algorithm: "SHA256withRSA");
    rsa.initSign(privateKey);
    rsa.update(hashedDocument);
    byte[] digitalSignature = rsa.sign();
```

```
    documentEntity.setDigitalSignature(digitalSignature);
    documentEntity.setPublicKey(publicKey.getEncoded());
    documentEntity.setSigned(true);
    documentEntity.setSignedOn(LocalDateTime.now());

    try {
        repo.save(documentEntity);
        ra.addAttribute( attributeName: "signSuccess",  attributeValue: true);
    } catch (Exception e) {
        ra.addAttribute( attributeName: "signFail",  attributeValue: true);
    }
    return "redirect:/user-documents";
}
```

Figure 9 & 10: Sign Document Code Snippets

Sign Document

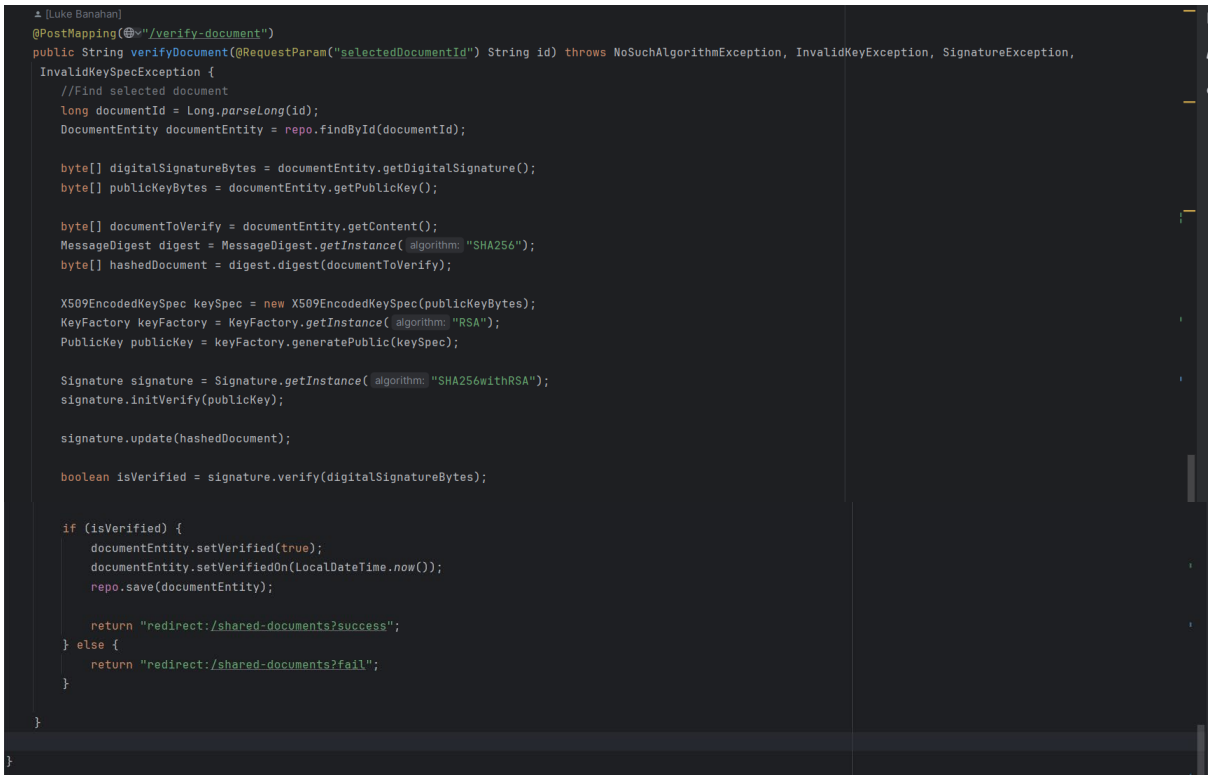
This is the method that handles the digital signature generation. First I request the selectedDocumentId paramater from the front end and parse it from String to a long. I then call on my document repository class to use the method findById. This allows me to retrieve the document selected by the user from the database.

I then declare a document entity with the selected document. I then create a new byte array variable called hashed document. I call on Message Digest and use SHA-256 to create a hash of the selected document.

RSA alrogithm is used to generate a key pair. An RSA signature object is then declared, signed and updated with the document hash. A new byte array called digitalSignature is declared as the signed- hash of the selected document.

The digital signature and the public key are now saved to the document entity. The signed boolean is set to true and the time of the signature is stored too. If the proccess was

successful the system redirects the user to the My Documents page with a success message. If it fails the same is true but with a fail message.



```
[Luke Banahan]
@PostMapping("/verify-document")
public String verifyDocument(@RequestParam("selectedDocumentId") String id) throws NoSuchAlgorithmException, InvalidKeyException, SignatureException,
 InvalidKeySpecException {
    //Find selected document
    long documentId = Long.parseLong(id);
    DocumentEntity documentEntity = repo.findById(documentId);

    byte[] digitalSignatureBytes = documentEntity.getDigitalSignature();
    byte[] publicKeyBytes = documentEntity.getPublicKey();

    byte[] documentToVerify = documentEntity.getContent();
    MessageDigest digest = MessageDigest.getInstance( algorithm: "SHA256");
    byte[] hashedDocument = digest.digest(documentToVerify);

    X509EncodedKeySpec keySpec = new X509EncodedKeySpec(publicKeyBytes);
    KeyFactory keyFactory = KeyFactory.getInstance( algorithm: "RSA");
    PublicKey publicKey = keyFactory.generatePublic(keySpec);

    Signature signature = Signature.getInstance( algorithm: "SHA256withRSA");
    signature.initVerify(publicKey);

    signature.update(hashedDocument);

    boolean isVerified = signature.verify(digitalSignatureBytes);


    if (isVerified) {
        documentEntity.setVerified(true);
        documentEntity.setVerifiedOn(LocalDateTime.now());
        repo.save(documentEntity);

        return "redirect:/shared-documents?success";
    } else {
        return "redirect:/shared-documents?fail";
    }

}
}
```

Figure 11 & 12: Verify Document Code Snippets

Verify Document

This is the method that handles the verification of digital signatures. The method requests the selectedDocumentId parameter from the front end and parses it from a String and places the value in a long variable called documentId. The documentEntity class is then called and the reposotory class findById is called to find the selected document.

The digital signature is then called from the database and placed in a new byte array variable called digitalSignatureBytes. The same happens for the public key that is stored in the database for that digtially signed document. The same also occurs for the document content, which is stored in a byte array variable called documentToVerify.

The document content is then hashed and placed in a variable called hashedDocument. The document is then verified by the RSA algorithm and the public key. If the process is successful the user is redriected to the shared documents screen with a success message. The same is true if it fails excpet with a failure message.

2.4 Graphical User Interface (GUI)

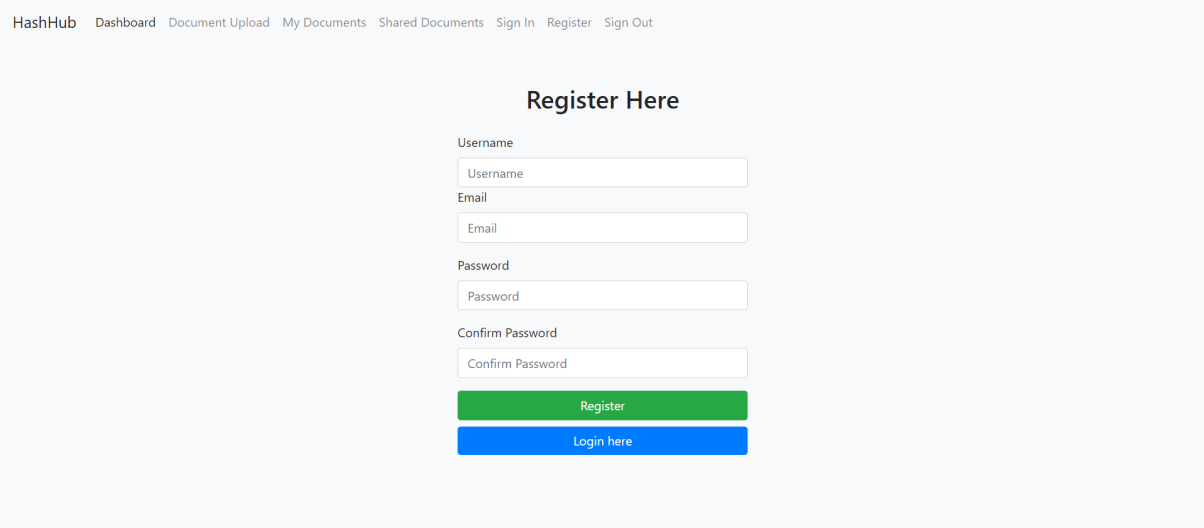


Figure 13: Register Screen

Register Screen

This is the register page. This is where the user will enter their details to create an account. All fields are required to be filled out. Username and email fields must be unique. Back-end validation will check if the username is unique. There is front end validation to check if the email field follows the format of an email address. (E.g., example@email.com). There is back-end validation to check if the email address is unique (i.e. the first entry of that email address in the database). There is front end validation to ensure that the password entered is at least 8 characters long and contains at least one alphabetic character and at least one numeric character (as per current OWASP recommendations). There is also front-end validation to ensure that both the password and confirm password fields match. Once all of this has been achieved, the user can successfully register an account.

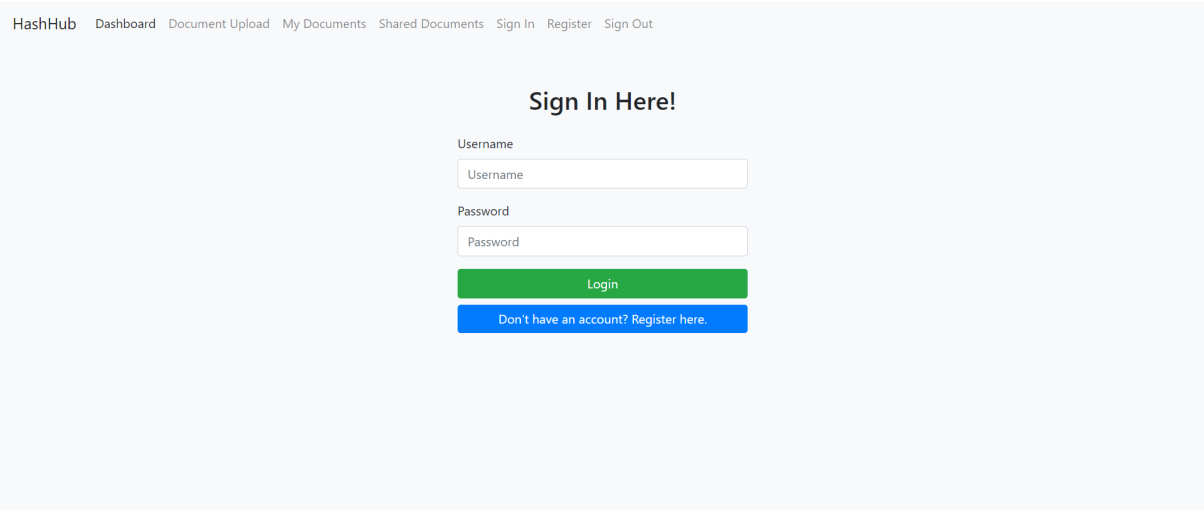


Figure 14: Log-In Screen

Log-In Screen

Here the user can sign into their account and access the functionality of the application. The user must enter a correct username and password that currently is registered in the database. If incorrect details are entered, the system will notify the user that they have entered incorrect details.

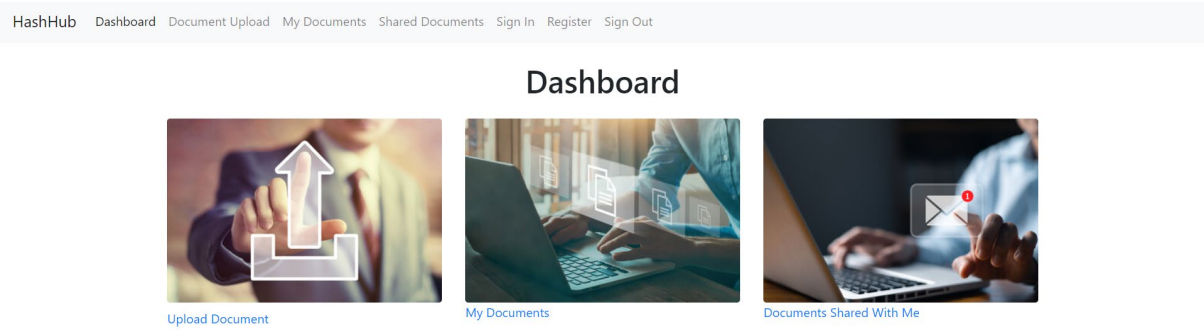


Figure 15: Dashboard Screen

Dashboard:

When a user successfully logs in, they will be redirected to the dashboard. The dashboard shows images relating to the functionality of the pages they are linked to. When a user clicks any of these links they will be redirected to the corresponding page where they can access the respective functionality.

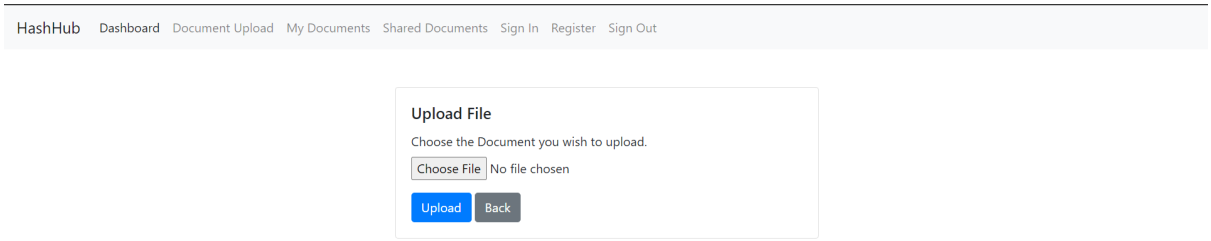


Figure 16: Upload Screen

Upload Page:

Here the user can upload a file from their computer directly to the system database.

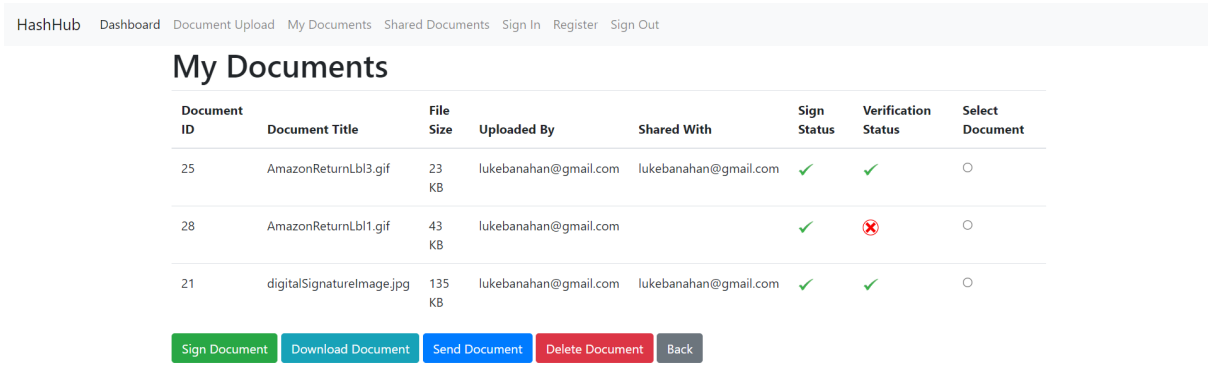


Figure 17: My Documents Screen

My Documents:

The My Documents page contains information about all of the documents that have been uploaded by the current user. Here the user can access document management functionality such as Sign Document, Download Document, Send Document and Delete Document. The back button will redirect the user to the Dashboard. Relevant information about the documents is displayed here too such as Document ID, Document Title, File Size (KB), who the document was uploaded by, users that the document has been shared with. A red x or a green tick will be used to communicate to the user whether the document has been signed or verified. (Green tick means yes, Red X means no). There is a radio button on the table to select a document.

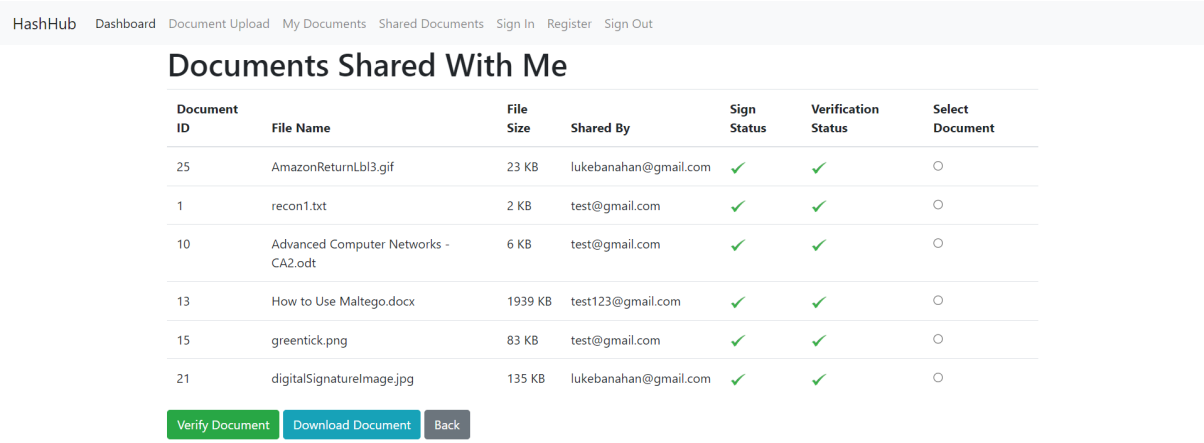


Figure 19: Documents Shared with Me Screen

Documents Shared with Me:

This page shows the user all the documents that have been sent to the user by another user. Relevant information about the documents is displayed here too such as Document ID, Document Title, File Size (KB), who the document was uploaded by, users that the document has been shared with. A red x or a green tick will be used to communicate to the user whether the document has been signed or verified. (Green tick means yes, Red X means no). There is a button to Verify the document and another to Download the Document.

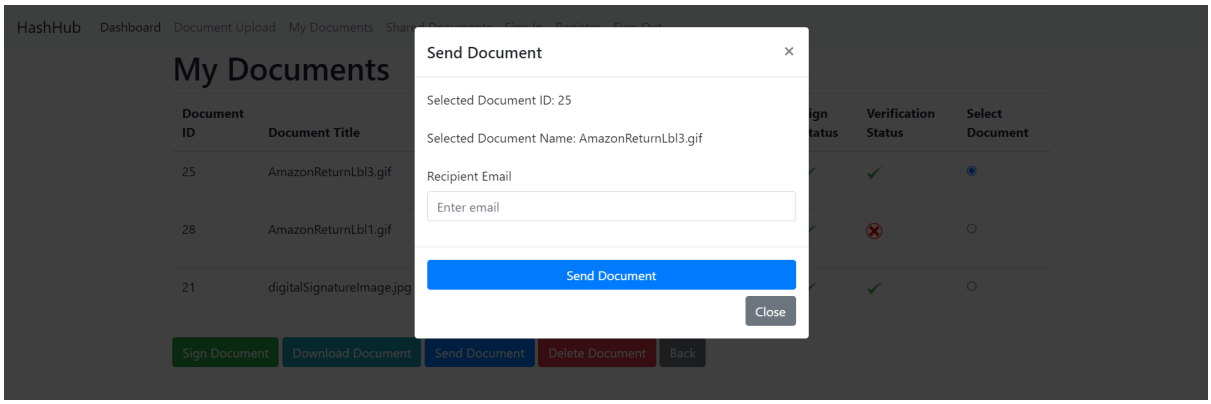


Figure 20: Send Document Pop-up

Send Document Pop-Up

When a Document has been selected to be sent to another user, a pop-up screen will prompt the user to enter the recipient's email address. The pop-up screen will also contain relevant information about the document to make sure that the user knows they are sending the correct document. When the user enters the recipient email address and clicks

“Send Document” the document will be shared with the recipient, and they will receive an email notification.

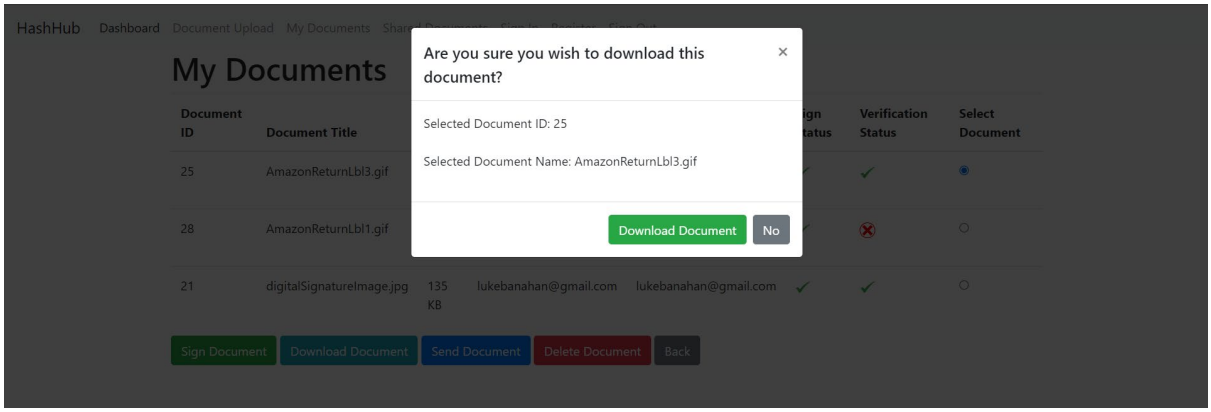


Figure 21: Download Document Pop-up

Download Document Pop-up:

When selects a document and clicks download document, a pop-up screen will appear, asking the user are they sure this is the correct document they wish to download. The pop-up will contain information relevant to the document they selected. When they click “Download Document” here the download will begin.

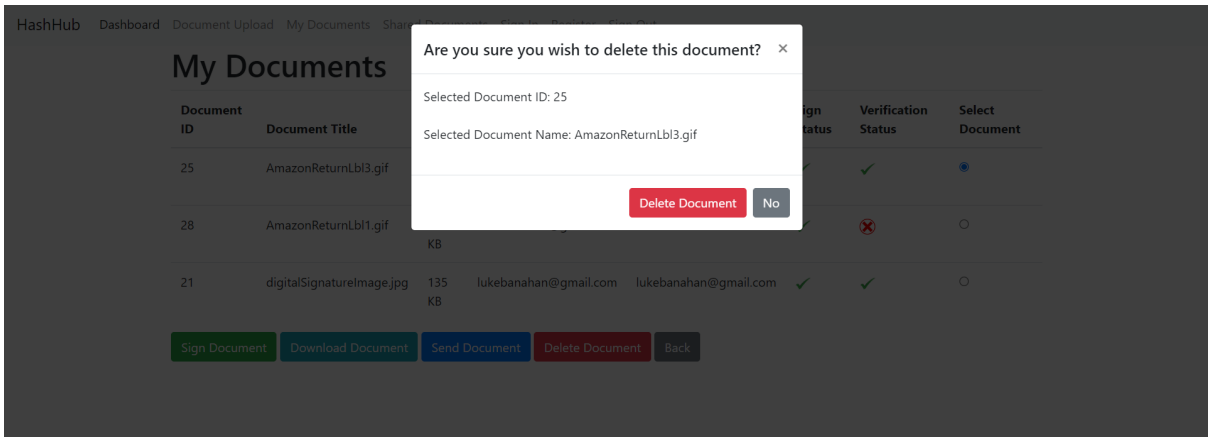


Figure 22: Delete Document Pop-up

Delete Document Pop-up

When a document has been selected for deletion, the user will be prompted with a pop-up screen which asks the user do they wish to delete this document. The pop-up screen will

contain relevant information to the document selected for deletion. When the user clicks delete document, the document will be removed from the database.

2.5 Testing & Evaluation

Three different types of testing were conducted:

- Black box testing including functional, accessibility, and security testing.
- White box testing including unit testing.

2.5.1 Testing Summary

Table 1 Functional Testing Results Summary

Test Case 1.1: Register Missing Fields	Results
Test 1.1.1: Username field	Pass
Test 1.1.2: Email Address field	Pass
Test 1.1.3: Password field	Pass
Test 1.1.4: Confirm password field	Pass
Test Case 1.2: Register Non-unique Credentials	Results
Test 1.2.1: Username field.	Pass
Test 1.2.2: Email Address field.	Pass
Test 1.2.3: Username and Email address field.	Pass
Test 1.3 Incorrect Password Format	Results
Test 1.3.1 Password too short.	Pass
Test 1.3.2 Password does not contain any numbers	Pass
Test 1.3.3 Password does not contain any letters	Pass
Test 1.3.4 Passwords do not match	Pass
Test Case 1.4: Register Success	Results
Test 1.4: Correct credentials	Pass
Test Case 1.5 Log-In	Results
Test 1.5.1: Incorrect credentials	Pass

Test 1.5.2: Successful Log-In	Pass
Test Case 1.6: Log-Out	Results
Test 1.6 Log-Out Success	Pass
Test Case 1.7: Document Upload	Results
Test 1.7: Document Upload Success	Pass
Test Case 1.8: Send Document	Results
Test 1.8 Document Sent Successfully	Pass
Test Case 1.9: Delete Document	Results
Test 1.9: Delete Document Success	Pass
Test Case 1.10: Document Download	Results
Test 1.10.1: Document Download Success (My-Documents)	Pass
Test 1.10.2 Document Download Success (Documents Shared with You)	Pass
Test Case 1.11: Sign Document	Results
Test 1.11: Sign Document Success	Pass
Test Case 1.12: Verify Document	Results
Test 1.12: Document Verification Success	Pass

Table 2 Security Testing Results Summary

Test Case 2.1: Security Tests	Results
Test 2.1.1: Access Control	Pass
Test 2.1.2 Password Hashing	Pass
Test 2.1.3 Password Salting	Pass

Table 3 Accessibility Testing Results Summary

Test Case 3.1: Accessibility Tests	Results
Test 3.1.1: Register	Pass
Test 3.1.2: Log-In	Pass
Test 3.1.3: Log-Out	Pass
Test 3.1.4: Document Upload	Pass
Test 3.1.5: Send Document	Pass
Test 3.1.6: Delete Document	Pass
Test 3.1.7: Document Download	Pass
Test 3.1.8: Sign Document	Pass
Test 3.1.9: Verify Document	Pass
Test 3.1.10: Dashboard Navigation	Pass
Test 3.1.11: Nav Bar Navigation	Pass

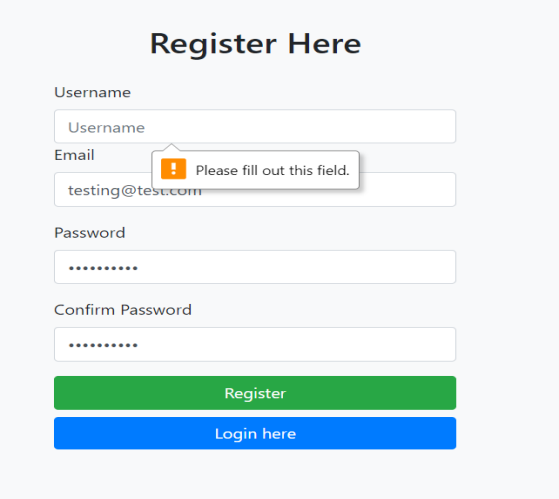
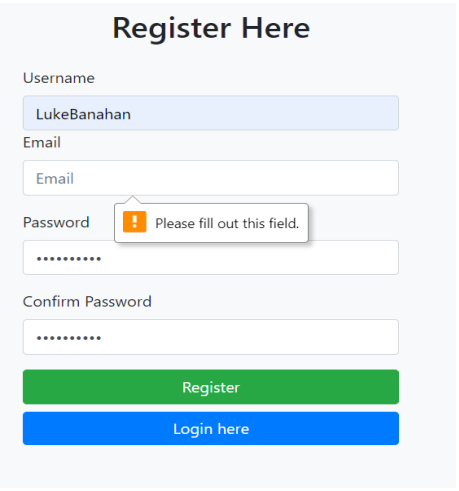
Table 4 Unit Testing Results Summary

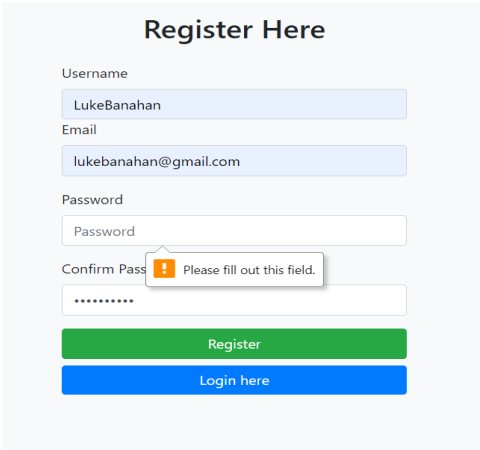
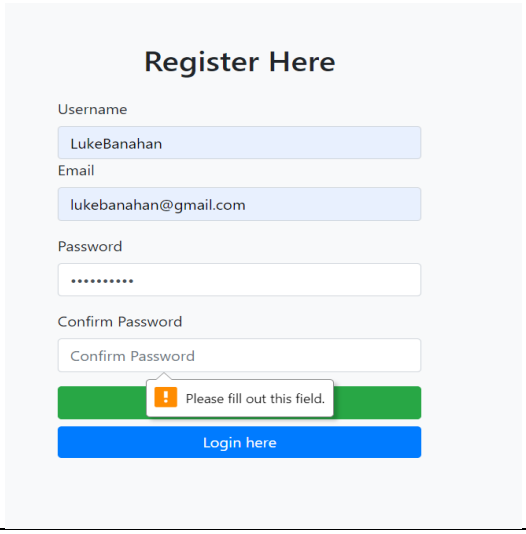
Unit Tests: Document Repository	Results
Save Document	Pass
Find Document by Id	Pass
Delete Document by Id	Pass
Unit Tests: User Repository	Results
Find By Username	Pass
Find By Email	Pass

2.5.2 Functional Testing

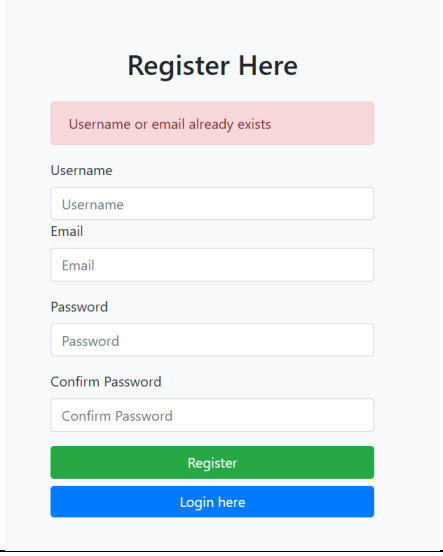
2.5.2.1 Register Tests

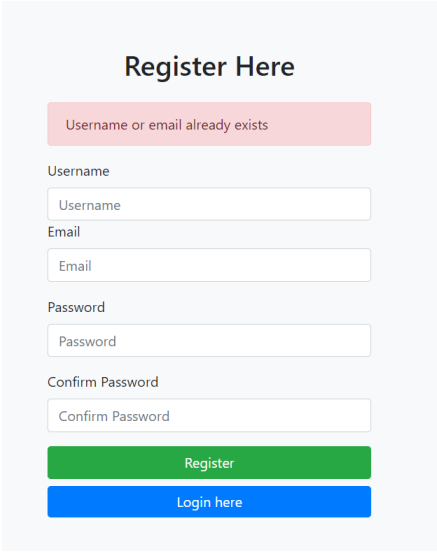
Test Case 1.1:	Missing Field:
Description:	This test case will test the behaviour of the application when a user tries to register to the application but leaves an input field empty in the register form.
Expected Behaviour:	The system will produce an error message notifying the user of which field was left empty.
Test 1.1.1: Username Field Steps:	1. Leave the username field blank 2. Enter a unique email address. 3. Enter a password that is at least 8 characters long and contains at least one letter and one digit. 4. Enter the same password in the 'Confirm Password' field.
Test 1.1.1: Outcome:	The system produces an alert, notifying the user which field needs to be filled in.

	
Test 1.1.2: Email Address Field Steps:	1. Enter a unique username. 2. Leave the email address field blank. 3. Enter a password that is at least 8 characters long and contains at least one letter and one digit. 4. Enter the same password in the 'Confirm Password' field.
Test 1.1.2: Outcome	The system produces an alert, notifying the user which field needs to be filled in. 
Test 1.1.3: Password Field Steps:	1. Enter a unique username. 2. Enter a unique email address. 3. Leave the password field blank. 4. Enter the same password in the 'Confirm Password' field.

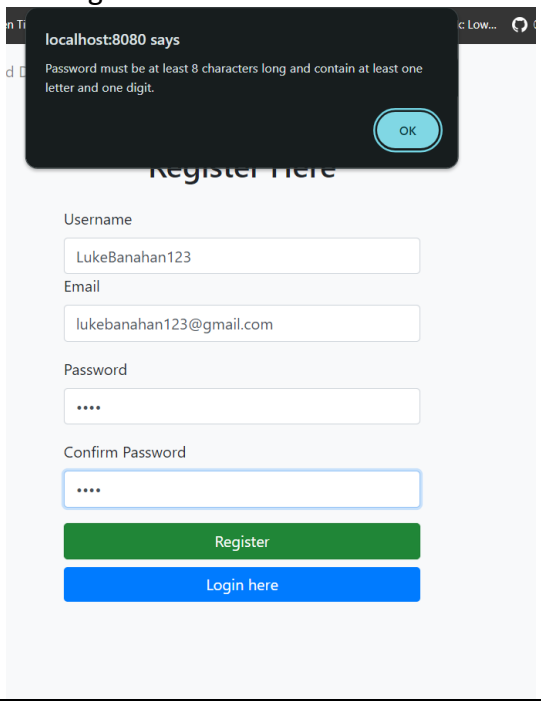
Test 1.1.3: Outcome	The system produces an alert, notifying the user which field needs to be filled in. 
Test 1.1.4: Confirm Password Field Steps:	1. Enter a unique username. 2. Enter a unique email address. 3. Enter a password that is at least 8 characters long and contains at least one letter and one digit. 4. Leave the confirm password field blank.
Test 1.1.4: Outcome:	The system produces an alert, notifying the user of which field needs to be filled in. 

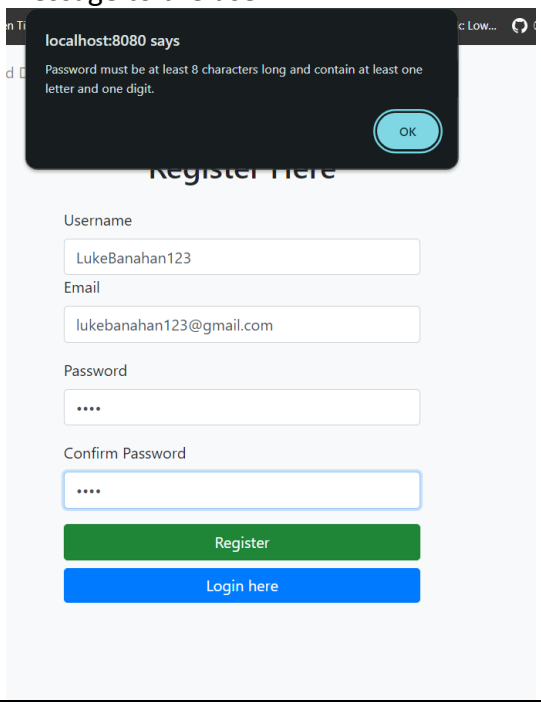
Test Case 1.2:	Register Non-unique credentials.
Description:	This test case will test the system behaviour when a user enters a username or email

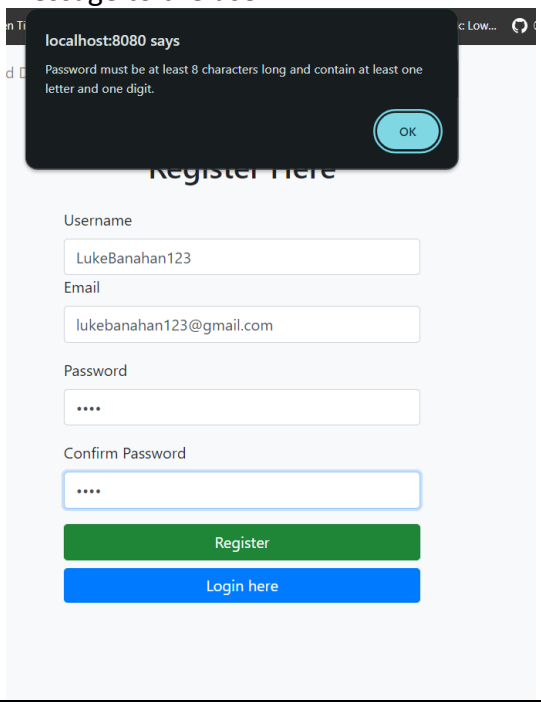
	address that already exists in the system database.
Expected Behaviour	The system will produce an error message to notify the user that either the username or email address is already taken. As per OWASP recommendations the system will not specify which of these is taken.
Test Pre-conditions:	The tester already has an account registered and they know the credentials.
Test 1.2.1: Username Field Steps:	1. Enter a username that is already registered in the system into the username field. 2. Enter a unique email address in the email field. 3. Enter a password that is at least 8 characters long and contains at least one letter and one digit. 4. Enter the same password in the confirm password field.
Outcome:	The system produces an error that says, “username or email address already exists”. 
Test 1.2.2: Email Address Field Steps:	1. Enter a unique username in the username field. 2. Enter an email address that is already registered in the system. 3. Enter a password that is at least 8 characters long and contains at least one letter and one digit.

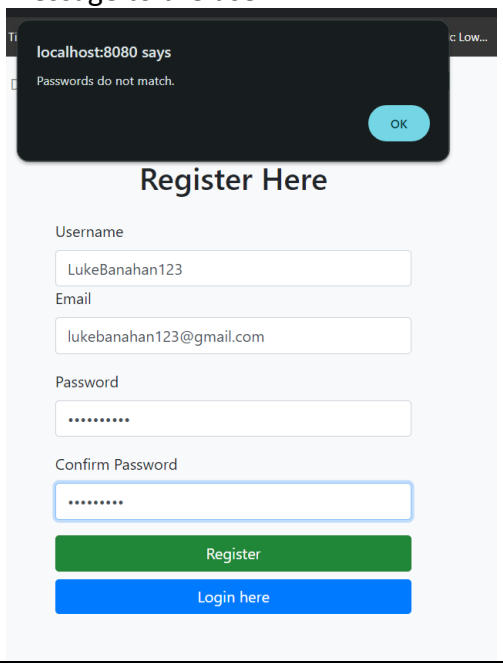
	4. Enter the same password in the confirm password field.
Outcome:	The system produces an error that says, “username or email address already exists”. 
Test 1.2.3: Username and Email Address fields Steps	1. Enter a username that is already registered with the system. 2. Enter an email address that is already registered with the system. 3. Enter a password that is at least 8 characters long and contains at least one letter and one digit. 4. Enter the same password in the confirm password field.
Outcome:	The system produces an error that says, “username or email address already exists”.

Test Case 1.3:	Incorrect password format.
Description:	This test case will test the system behaviour when a user enters a password that does not pass the system password validation. Passwords must be at least characters long, contain at least 1 letter and at least 1 digit.
Expected behaviour:	The system will display the relevant alert message to the user.
Test 1.3.1: Password too short	1. Enter a unique username.

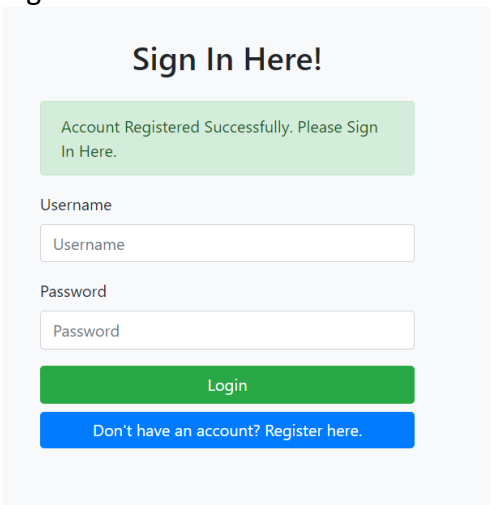
Steps:	2. Enter a unique email address. 3. Enter a password that is less than 8 characters. 4. Enter the same password in the confirm password field.
Test 1.3.1: Outcome	The system displays the relevant alert message to the user. 
Test 1.3.2: Password does not contain any numbers. Steps:	1. Enter a unique username. 2. Enter a unique email address. 3. Enter a password that is 8 characters long but contains only letters. 4. Enter the same password in the confirm password field.

Test 1.3.2: Outcome	The system displays the relevant alert message to the user. 
Test 1.3.3: Password does not contain any letters. Steps:	1. Enter a unique username. 2. Enter a unique email address. 3. Enter a password that is 8 characters long but contains only digits (e.g., 12345678) 4. Enter the same password in the confirm password field.

Test 1.3.3: Outcome	The system displays the relevant alert message to the user. 
Test 1.3.4: Passwords do not match. Steps:	1. Enter a unique username. 2. Enter a unique email address. 3. Enter a password that is 8 characters long and contains at least one letter and one digit. 4. Enter a different password in the password field.

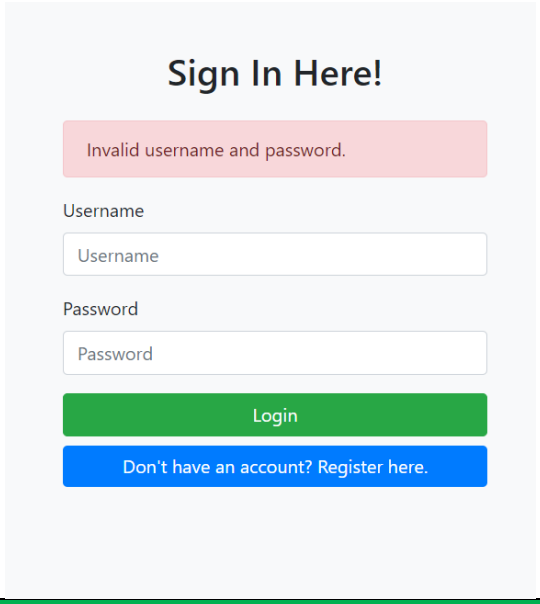
Test 1.3.4: Outcome	The system displays the relevant alert message to the user. 

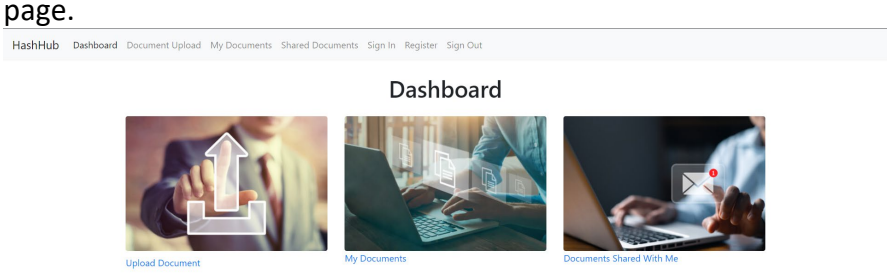
Test Case 1.4:	Registration Success
Description:	The user successfully registers an account with the application.
Expected Behaviour:	The system saves the user's credentials and redirects to the log in screen and displays a message to tell the user the registration was successful.
Test Steps:	1. Enter a unique username in the username field. 2. Enter a unique email address in the email field. 3. Enter a password that is at least 8 characters long and contains at least one letter and at least one digit. 4. Enter the same password in the confirm password field. 5. Click register.
Outcome:	System notifies user that account was registered successfully and redirects to the

	log in screen. 
Result:	Pass

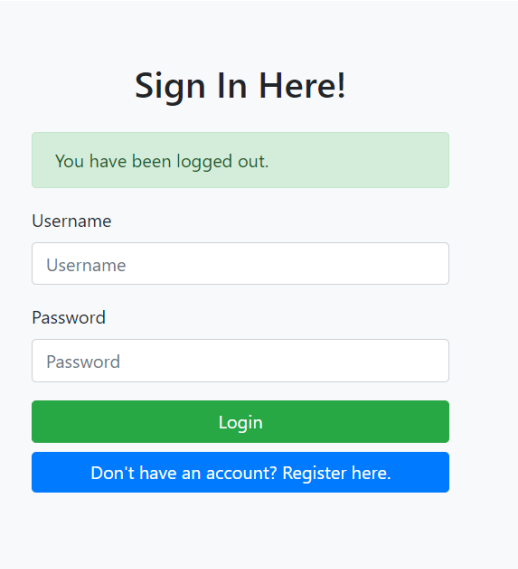
2.5.2.2 Login Tests

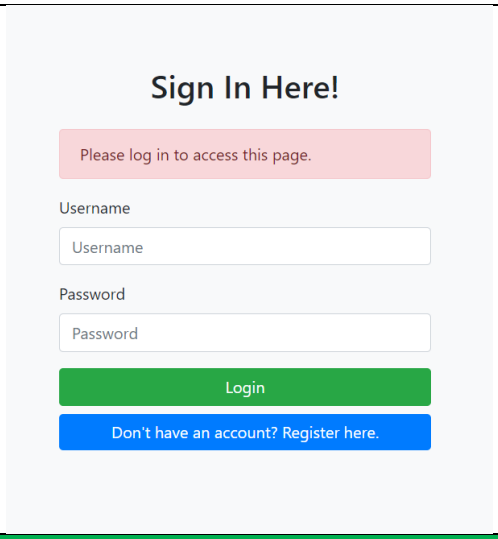
Test Case 1.5.2:	Incorrect Username or Password
Description:	The user enters an incorrect username or password into the log in screen.
Expected Behaviour:	The system notifies the user that the username or password is incorrect.
Test Pre-Conditions:	Tester already has an account registered in the system and is on the login page.
Steps:	1. Enter incorrect username or password (or both) 2. Click login.

Outcome:	The system notifies the user that the username or password is incorrect. 
Result	Pass

Test Case 1.5.2:	Successful Login
Description:	This test case describes the system behaviour when a user successfully logs into their account.
Expected Behaviour:	The system logs the user into their account and redirects to the 'Dashboard' page.
Test Pre-Conditions:	Tester already has an account registered in the system and is on the login page.
Steps:	1. Enter correct username. 2. Enter correct password. 3. Click login button.
Outcome:	User is logged in successfully and is redirected to the 'Dashboard' page. 
Result:	Pass

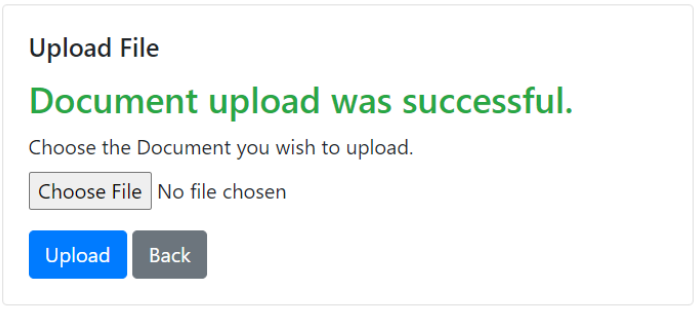
2.5.2.3 Log-Out Tests:

Test Case 1.6:	Log-Out
Description:	The user can log out of the application.
Expected Behaviour:	When the user clicks the “Sign Out” button on the Nav-Bar, the system logs the user out of their current session.
Test Pre-Conditions:	The tester has an account registered.
Steps:	1. Sign into account. 2. Click Sign-Out button. 3. Try to navigate to Dashboard page.
Outcome:	The system logs the user out of their current session. The user is redirected to the Login Screen and an alert message is displayed to the user to notify them they are signed out. When the user tries to navigate to the Dashboard page, the system will notify the user that they must be signed in to access this page. This is confirmation that the user was successfully signed out. 

	
Result:	Pass

2.5.2.4 Document Upload Tests

Test Case 1.7:	Document Upload Success
Description:	This test case will check to see that a user can upload a document to the application successfully.
Expected Behaviour:	When user uploads a document, system will notify the user with a success message. System will store document in database.
Pre-Conditions	Tester has already registered an account and is logged in.
Steps:	1. Navigate to “Document Upload” page. 2. Click “Choose File” button. 3. Select a file that is smaller than 50mb. 4. Click “Upload”.

Outcome:	System notifies user that document upload was successful. 
Result:	Pass.

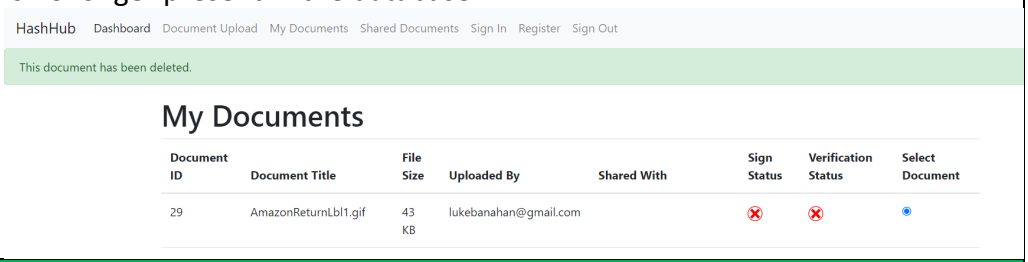
2.5.2.5 Send Document Tests

Test Case 1.8:	Document sent successfully
Description:	This test will check that a user can send a document to another user successfully.
Expected Behaviour:	When the tester sends the document, the system sends a notification email to the recipient, allows the recipient to see that document on the 'Documents Shared with You' screen and the system will display a success message.
Test Pre-Conditions	The tester has registered 2 separate accounts with the system. The tester has access to the email address of one of the accounts registered. This account will be used to test the recipient side. The other account will be used to test the sender. The sender account must be logged in and have at least one document previously uploaded.
Steps:	1. Navigate to the 'My Documents' page. 2. Select document you wish to send via radio button. 3. Click 'Send Document' 4. Enter recipient email in pop-up screen.

	5. Click 'Send Document' again (in pop up screen). 6. Check success message. 7. Check email address of recipient email for email notification titled "New Document Shared with You". 8. Log into recipient account in system. 9. Navigate to 'Documents Shared with You' page.
Outcome:	Recipient account should receive email notification and document should now be visible in 'Documents Shared with Me' page.
Result:	Pass

2.5.2.6 Delete Document Tests

Test Case 1.9:	Delete Document Success
Description:	This test will check the functionality that allows a user to delete a document they have previously uploaded.
Expected Behaviour:	The system will notify the user with a success message and the document will be deleted from the database.
Test Pre-Conditions	The tester has an account registered, is currently logged in and has at least one document uploaded into the system.
Steps:	1. Navigate to "My-Documents" page. 2. Select document you wish to delete via radio button. 3. Click "Delete Document" button. 4. Confirm and click "Delete Document" again in pop up window. 5. Confirm deletion success message. 6. Check system database to ensure the document is no longer present in database.

Outcome:	The system notifies the user the document has been deleted. The document is no longer present in the database. 
Result:	Pass

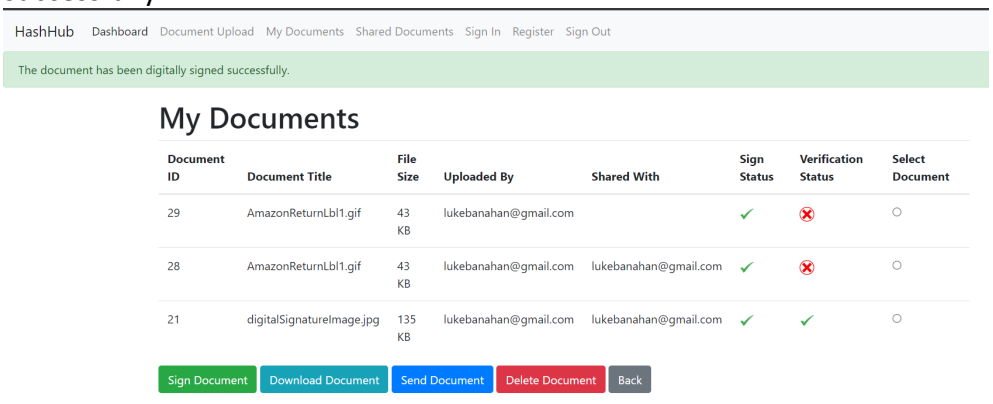
2.5.2.7 Document Download Tests

Test Case 1.10.1:	Document Download Success (My-Documents)
Description:	Test that a user can successfully download a document from the 'My Documents' page.
Test Pre-Conditions:	The tester has an account registered, is logged in and has at least one document uploaded.
Expected Behaviour:	The system allows the user to successfully download the selected document.
Steps:	1. Navigate to 'My Documents' page. 2. Select document you wish to download via radio button. 3. Click 'Download Document'
Outcome:	Document is successfully downloaded.
Result:	Pass

Test Case 1.10.2:	Document Download Success (Documents Shared with You)
Description:	Test that a user can successfully download a document from the 'Documents Shared with you' page.
Test Pre-Conditions:	The tester has an account registered, is logged in and has at least one document shared with them.

Expected Behaviour:	The system allows the user to successfully download the selected document.
Steps:	1. Navigate to 'Documents Shared with You' page. 2. Select document you wish to download via radio button. 3. Click 'Download Document' button.
Outcome:	Document is successfully downloaded.
Result:	Pass

2.5.2.8 Sign Document Tests

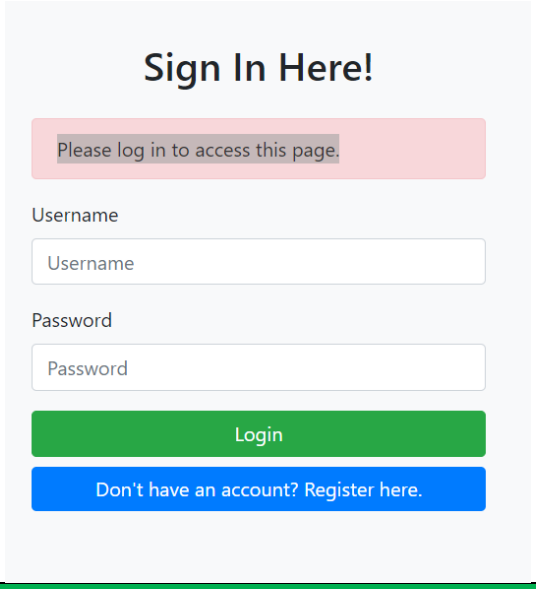
Test Case 1.11:	Sign Document Success
Description:	This test case checks that a user can successfully sign a document with a digital signature.
Expected Behaviour:	The system displays a success message, and the "Sign Status" column of the document now has a green tick to indicate that it has been signed.
Test Pre-Conditions	The tester has an account registered, is currently signed in and has at least one document uploaded to the system.
Steps:	1. Navigate to "My Documents" page. 2. Select the document you want to sign by the radio button. 3. Click "Sign Document" button. 4. Check success message.
Outcome:	The document is signed, the system displays a success message and the red x changes to a green tick to indicate the document has been signed successfully. 
Result:	Pass

2.5.2.9 Verify Document Tests

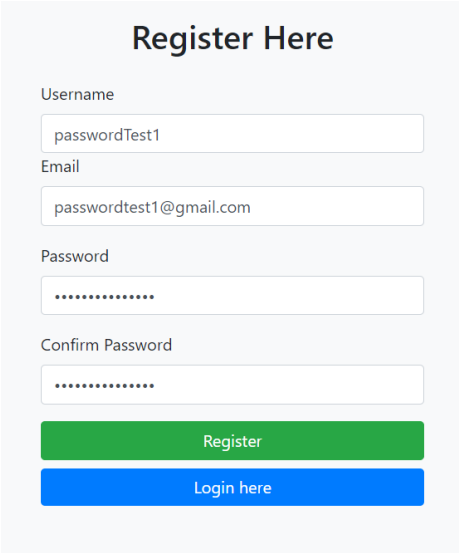
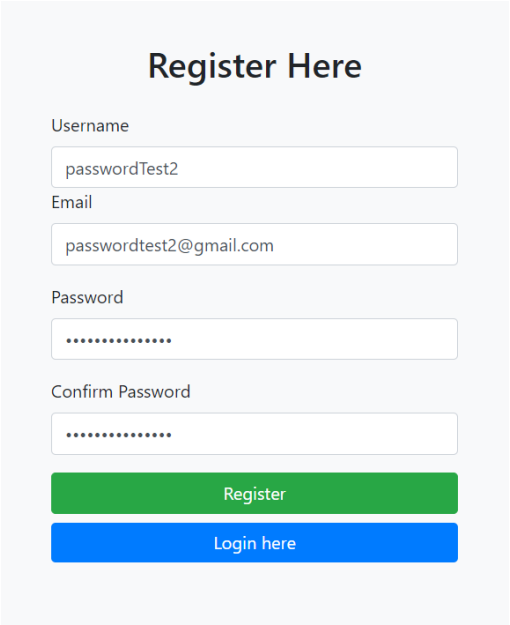
Test Case 1.12:	Document Verification Success
Description:	This test checks that a user can securely verify that a document has been signed securely.
Expected Behaviour:	The system will display a success message to the user and the red x will change to a green tick in the document column to indicate that this document has now been verified.
Test Pre-Conditions	The tester has an account registered, is currently logged in and at least one digitally signed document has been shared with them.
Steps:	1. Navigate to 'Documents Shared with Me page'. 2. Select document to be verified via radio button. Click "Verify Document" button. 3. Check success message.
Outcome:	The system verifies the document and displays a success message to the user. The red x, changes to a green tick in the document column to indicate the document has been verified.
Result:	Pass

2.5.3 Security Tests

Test Case 2.1.1:	Access Control
Description	Test that all pages except Register/Login pages deny entry to unauthorized/unauthenticated users.
Expected Behaviour:	When an unauthorized/unauthenticated user tries to access any part of the application other than Register/Login, the system will redirect the user to the Login page and will alert the user that they must log in to access this page.
Steps:	1. From Register/Login page, try to navigate to Dashboard,

	2. Try to navigate to Upload Document page. 3. Try to navigate to My Documents page. 4. Try to navigate to Documents Shared with Me page.
Outcome:	System redirects user to Login page with alert that says, “Please log in to access this page.” 
Result:	Pass

Test Case 2.1.2:	Password Hashing
Description:	Check to see if passwords are being hashed during the registration and password storage process.
Expected Behaviour:	When an account is created and the tester checks the password in the database, the password should not be stored as plain text, should be a 60-character, hash of that password.
Steps:	1. Register a new account. 2. Use “Password123” for password. 3. Check password in database.
Outcome:	Password is not stored as plain text in database. Password is a 60-character password hash.
Result:	Pass

Test Case 2.1.3:	Password Salting
Description	This is a test to check that the password salting functionality is working as expected.
Expected Behaviour:	When the system stores the passwords in the system, and the tester checks the database, both password hashes should be different. This is because of the salt added to the password during the registration process.
Steps:	1. Register an account with the application. Use “PasswordTest123” for this test.  2. Register a second account with the application but use the same password.  3. Check that the user can log into both accounts. 4. Check the passwords stored in the database.

	5. Compare both passwords, they should both be stored as different hashes, even though the same password was used for both accounts during registration.
Outcome:	 When comparing passwords stored in the database, we can see that both passwords begin with '$2a$10$', this is the Bcrypt version identifier. Bcrypt is the algorithm being used to hash and salt these passwords. However, the rest of the password hashes are completely different due to the salting process.
Result:	Pass

2.5.4 Accessibility Tests

To test that this application is accessible to people visually impaired people and people with disabilities, who use screen readers, all functional requirements of this application should be able to be performed without the use of a mouse. All pages should be able to be navigated using tab and enter functions. The same is true for the functionality. All buttons and information should be readable by a screen reader (should have a descriptive title in the html tags). To test this all-functional requirements and page navigations were tested using only tab and enter buttons.

Requirement	Pass Condition	Result
Register	User can register using only keyboard and each field and button has titles that can be read by screen readers.	Pass
Log-In	User can log-in using only keyboard and each field and button has titles that can be read by screen readers.	Pass
Log-Out	User can log-out using only keyboard and each field and button has titles that can be read by screen readers.	Pass
Upload Document	User can upload a document using only keyboard and each field and button has titles that can be read by screen readers.	Pass
Download Document	User can download a document using only keyboard and each field	Pass

	and button has titles that can be read by screen readers.	
Send Document	User can send document using only keyboard and each field and button has titles that can be read by screen readers.	Pass
Delete Document	User can delete document using only keyboard and each field and button has titles that can be read by screen readers.	Pass
Sign Document	User can sign document using only keyboard and each field and button has titles that can be read by screen readers.	Pass
Verify Document	User can verify document using only keyboard and each field and button has titles that can be read by screen readers.	Pass
Navigation	User can navigate the application using only keyboard and each field and button has titles that can be read by screen readers.	Pass

2.5.5 Unit Testing

Repository Tests

The following tests were written in Spring Boot using Spring Data JPA and Junit. These tests were written to test the data access layer of my application to ensure that the CRUD functionalities of the application were behaving as expected.

```
private DocumentRepository documentRepository;

//test that save function is working in DocumentRepository
[Luke Banahan]
@Test
public void DocumentRepository_Save_ReturnSavedDocuments() {

    //Arrange
    DocumentEntity documentEntity = DocumentEntity.builder()
            .id(1L)
            .name("testDocument")
            .size(1024L)
            .email("test@test.com")
            .sharedWith("shareTest")
            .isSigned(true)
            .isVerified(true)
            .build();

    //Act
    DocumentEntity savedDocumentEntity = documentRepository.save(documentEntity);


    //Assert
    Assertions.assertNotNull(savedDocumentEntity);
}
```

Figure 24: Document Repository – Save Unit Test

Save Test.

This test was written to test the save functionality of the application. It creates a document entity builds it and stores it in a variable called savedDocumentEntity. The test then asserts that savedDocumentEntity is not null to check that the save functionality worked.

```
[Luke Banahan]
@Test
public void DocumentRepository_DeleteById_ReturnNullDocument(){
    //Arrange
    DocumentEntity documentEntity = DocumentEntity.builder()
            .id(1L)
            .name("testDocument")
            .size(1024L)
            .email("test@test.com")
            .sharedWith("shareTest")
            .isSigned(true)
            .isVerified(true)
            .build();

    DocumentEntity documentEntity2 = DocumentEntity.builder()
            .id(2L)
            .name("testDocument2")
            .size(1024L)
            .email("test2@test.com")
            .sharedWith("shareTest2")
            .isSigned(true)
            .isVerified(true)
            .build();

//Act
documentRepository.save(documentEntity);
documentRepository.save(documentEntity2);

//check that both documents saved
Assertions.assertNotNull(documentEntity);
Assertions.assertNotNull(documentEntity2);

//delete document by id
documentRepository.deleteById(documentEntity2.getId());

//assert that the correct document was deleted
Assertions.assertNotNull(documentEntity);

//attempt to retrieve deleted document and assert that it is deleted.
DocumentEntity deletedDocument = documentRepository.findById(documentEntity2.getId()).orElse( other: null);
Assertions.assertNull(deletedDocument);

// check that remaining document is still present, to confirm only specified document was deleted.
DocumentEntity remainingDocument = documentRepository.findById(documentEntity.getId()).orElse( other: null);
Assertions.assertNotNull(remainingDocument);
```

Figure 25 & 26: Document Repository – DeleteById Unit Test

DeleteById Test

The above tests that the DeleteById functionality is behaving as expected. Here I created to Document Entities, saved them both, and checked that they were not null. Then deleted one and stored the deleted one in a deletedDocument entity variable. I then asserted that the Deleted Document was null and the remaining document was not null. This will ensure the delete by id function is working as expected.

```
[Luke Banahan]
@Test //test that findById function works as expected.
public void DocumentRepository_findDocumentById_returnDocumentById() {
    //Arrange
    DocumentEntity documentEntity = DocumentEntity.builder()
            .id(1L)
            .name("testDocument")
            .size(1024L)
            .email("test@test.com")
            .sharedWith("shareTest")
            .isSigned(true)
            .isVerified(true)
            .build();

    //Act
    documentRepository.save(documentEntity);
    DocumentEntity foundDocument = documentRepository.findById(1L);


    //Assert
    Assertions.assertEquals(documentEntity, foundDocument);


}
```

Figure 27: Document Repository – findDocumentById Unit Test

Find By Id

This Junit test tests that the findById is performing as expecting. Firstly I create document entity, then I save it. I create a variable called foundDocument and store the saved document entity above into it by searching for it by id in the document repository. I then

assert that the saved document entity is equal to the found document (searched document. If this passes then the findById is working.

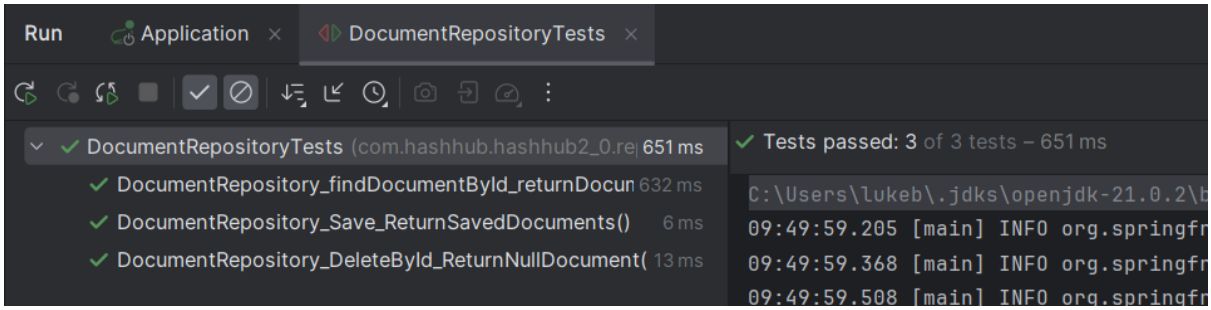


Figure 28: Document Repository – Unit Tests Passed

Tests passed.

All Document Repository Tests passed so I know these are working as expected.

User Repository Tests

```java
@DataJpaTest
@AutoConfigureTestDatabase(connection = EmbeddedDatabaseConnection.H2)

public class UserRepositoryTests {
    @Autowired
    private UserRepository userRepository;

    [Luke Banahan]
    @Test //Test that findByUsername method works.
    public void UserRepository_FindByUser_ReturnUser() {
        //Arrange
        UserEntity userEntity = UserEntity.builder()
                .id(1)
                .username("testUser")
                .password("testPass")
                .email("test@test.com")
                .build();

        //Act
        userRepository.save(userEntity);
        UserEntity foundUser = userRepository.findByUsername("testUser");

        //Assert
        Assertions.assertEquals(userEntity, foundUser);
    }
```

Figure 29: User Repository – FindByUser Unit Test

findByUsername

This is a test to check that the FindByUsername function works in the User Repository. I build a user entity and save it. I search for that user by username and store the found user in a UserEntity variable called testUser. I then assert that the saved userEntity is equal to the found user entity. If this passes I know the find by username function is working as expected.

```
    [Luke Banahan]
    @Test //Test that findByEmail method works
    public void UserRepository_FindByEmail_ReturnUser() {
        //Arrange
        UserEntity userEntity = UserEntity.builder()
                .id(1)
                .username("testUser")
                .password("testPass")
                .email("test@test.com")
                .build();

        //Act
        userRepository.save(userEntity);
        UserEntity foundUser = userRepository.findByEmail("test@test.com");

        //Assert
        Assertions.assertEquals(userEntity, foundUser);
    }
}
```

Figure 29: User Repository – FindByEmail Unit Test

findByEmail

This is a test to check that the findByEmail function works as expected. The test is carried out the same way as the findByUsername test. But searching for email instead.

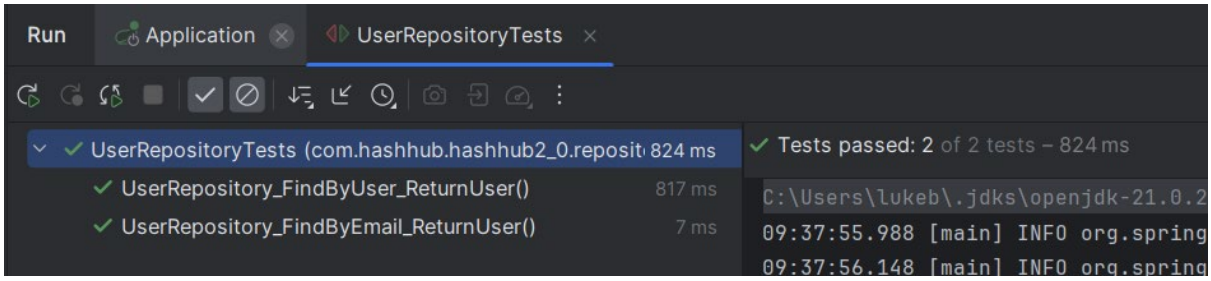


Figure 30: User Repository – Unit Tests Passed

User Repository Tests passed.

3 Conclusions

To conclude, HashHub is a document management system with enhanced accessibility features that provides users with a digital signature generation and verification functionality to guarantee authenticity, integrity, and non-repudiation through the use of applied cryptography techniques such as hashing, key pair generation and encryption. Below I will describe some of the advantages and disadvantages of this application in its current form.

Advantages	
1	Allows for streamlined document management.
2	Allows users to securely sign, send and verify documents.
3	Robust Security features that ensure user data is stored safely.
3	HashHub provides enhanced accessibility features when compared with competitors who provide a similar service.

Disadvantages	
1	Digital signatures are generated within the application are not accompanied by a Digital certificate which has been issued by an established certificate authority. This may lead to lack of trust in the cryptographic functions being used to generate the digital signature.
2	In its current form, application does not allow for direct editing of uploaded documents within the application.
3	Can only upload one document at a time.
3	Can only share documents with one other user at a time.

4 Further Development or Research

With more time and resources I would implement digital certificates from a certificate authority. This would enhance the credibility of the strength of the encryption being used to generate the digital signatures.

With more time, HashHub would also allow for the editing of documents from within the application, for example allowing users to inscribe their initials or personal signatures onto their documents, for other users to see. The application would then use the byte array from the edited document to generate the digital signature which would add another layer of authentication and document integrity.

I would also edit the document upload functionality of the application to allow for multiple document uploads at a time. This would enhance the document management aspect of the application.

Furthermore, in relation to enhanced document management, I would also allow for users to share their documents with multiple other users at a time, to streamline the experience of sending documents.

5 References

[1] Revenue Commissioners, “Revenue,” 2000. [Online]. Available: https://www.revenue.ie/en/home.aspx. [Accessed 10 January 2024].

[2] V. Tanzu, “Spring Boot Framework,” 2005. [Online]. Available: https://spring.io/projects/spring-boot. [Accessed 01 January 2024].

[3] DocuSign, “DocuSign,” 2024. [Online]. Available: https://apps.docusign.com. [Accessed 01 January 2024].

[4] PandaDoc, “PandaDoc,” 2024. [Online]. Available: https://apps.docusign.com/send/home. [Accessed 10 May 2024].

[5] SignNow, “SignNow,” 2024. [Online]. Available: https://www.signnow.com/. [Accessed 01 January 2024].

[6] National Disability Authority, “EU Web Accessibility Directive,” National Disability Authority, 23 September 2020. [Online]. Available: https://nda.ie/monitoring/eu-web-accessibility-directive. [Accessed 15 01 2024].

[7] Bootstrap, “Build fast, responsive sites with Bootstrap,” Bootstrap, 2011. [Online]. Available: https://getbootstrap.com/. [Accessed 25 January 2024].

[8] JetBrains, “Intellij IDEA (Integrated Development Environment,” 2001. [Online]. Available: https://www.jetbrains.com/idea/. [Accessed 01 January 2024].

[9] PostgreSQL, “PostgreSQL: The World's Most Advanced Open Source Relational Database,” Open Source, 1996. [Online]. Available: https://www.postgresql.org/?gad_source=1&gclid=Cj0KCQjw3ZayBhDRARIsAPWzx8oinps7MPsLBPr6jAM7cWkEQjq08IpsmVW5DH_ok5mKH2E3QojWkuwaApUmEALw_wcB. [Accessed 20 January 2024].

[10] Open Source, “Thymeleaf,” Apache, [Online]. Available: https://www.thymeleaf.org/. [Accessed 21 January 2024].

6 Appendices

1.1 Project Proposal

National College of Ireland

Project Proposal

HashHub

09/11/2023

Bachelor of Science (Honours) in Computing Information Cyber Security

Academic Year 2023/24

Luke Banahan

20116471

X20116471@student.ncirl.ie

1. Objectives

This application aims to create a platform that allows for several different use cases of applied cryptography. This platform will offer hash-based and encryption solutions. The main function of this platform is to allow users to sign documents securely, with the use of digital signatures, to verify the authenticity of electronic documents or messages. Another feature of this platform is a tool that will use hashing functions for file integrity verification, the purpose of this will be to ensure the integrity of files during downloads and file transfers. The application will also have a feature that uses hash functions to deduplicate

data, by comparing hashes of files to eliminate data for the purpose of storage optimisation.

2. Background

I chose to undertake this project to showcase the many use cases for applied cryptography. Applied cryptography is an exciting and fast evolving field and many new use cases are brought forward each year. By undertaking this project, I can showcase some of the amazing use cases, while also challenging myself to learn some new skills and concepts as I undertake this complex software project. This project will bring together all my prior experience and learning of the Java language and Software Engineering over the course of my studies at NCI, as well as my experience gained with Spring Boot while on work experience with Revenue Commissioner. I will dedicate much of my time and resources to this project over the course of the next 6 months. The project scope will be readdressed as necessary throughout the development process. A Software Requirement Specification (SRS) will be created to set out the requirements and objectives of the application. Project management tools such as Kanban boards will be used to keep track of requirements and their development. Ethical and legal considerations will be revised monthly to ensure that the project is compliant with GDPR and other relevant Ethical obligations. Testing and quality assurance will be crucial to ensure the application is secure and the cryptographic functions are behaving correctly. GitHub will be used for version control.

3. State of the Art

Adobe Sign and DocuSign are two of the most popular applications on the market now, with regards to secure document signing platforms. They both allow users to sign and send documents securely. EMC Data Domain and Dell EMC Avamar are some examples of tools that use hash functions to identify and eliminate duplicate data. What makes my project stand out is that it is a platform that employs all these use cases in one place.

4. Technical Approach

4.1 Agile Development

The technical approach I will take towards this application is the Agile Development approach. An agile approach will allow me to be more flexible with my development. I will focus on delivering small pieces of functionality at a time. While Agile Development usually focuses on team collaboration and customer feedback to drive development, instead I will take feedback and collaborate with my supervisor, allowing for iterations and small changes to the application as necessary. Regular meetings with my supervisor will be scheduled to assess the progress of the project and address any issues.

4.2 Requirements Identification

To identify requirements, firstly I will need to identify the main use cases of my application. The primary use cases for this application are secure document signing, file integrity

verification and data deduplication. Once the use cases are clearly defined, I will then begin to specify my functional requirements from non-functional requirements. When the requirements are clearly defined, they will be broken down into project tasks, set into a list in order of importance and necessity and I will work on them accordingly.

4.3 Project Planning

A Software Requirements Specification will be created, this will outline the requirements, use cases and system architecture. Here I will also specify the cryptographic algorithms I will use for my project, and security measures undertaken to ensure that this is a secure application.

I will use the Software Requirements Specification to identify project tasks. I will break these tasks down into smaller, more manageable tasks for easier tracking. I will use Kanban boards to visualize the project tasks and their progress. By using Kanban boards, I can effectively track my progress with each task and the project as a whole and how much more development needs to still be done. I will also establish milestones to track progress, such as the completion of key features etc. GitHub will be used for a version control system, and I will regularly commit changes to document development.

5. Technical Details

Java 11 is the programming language that will be implemented in this project. **Spring Boot** is the Framework I will employ. Intellij IDEA will be used for developing this project. Sqlite3 will be used as the Database.

Bouncy Castle Library is a Java library that I will utilise for API calls to perform cryptographic functions. Bouncy Castle Library will also provide me with access to Java Cryptography Arctitecture (JCA) and Java Cryptography Extension (JCE). These are also APIs that are instrumental in implementing Java security features.

Apache Commons Crypto is a library that allows for the use of an API for high performance Advanced Encryption Standard (AES) encryption. This library will allow me to utilize secure cryptographic keys.

RSA Algorithm is currently under consideration as a method of employing digital signatures, however if the high computational cost becomes an issue regarding performance of the application, **Elliptic Curve Digital Signature Algorithm (ECDSA)** may be used instead. Will revise this document pending implementation and testing of performance of said algorithms.

Message Digest Hash Functions, in particular, SHA-256. SHA-256 provides a high level of security and this will allow me to perform hash functions for file integrity verification.

6. Special Resources Required

N/A

7. Project Plan

Project Scope

HashHub is a web application that performs as an applied cryptography platform designed to provide secure solutions for various cryptographic and hashing applications. The

platform's primary function is to allow users to utilize a secure document signing platform. The application will allow users to verify the authenticity and integrity of electronic documents. While also providing a data deduplication tool, for storage optimization purposes.

Research

Before development begins, there is a lot of research to do on the most suitable cryptographic algorithms and libraries to utilise.

Supervisor Check In

Supervisor check ins will be scheduled on a weekly basis to keep track of progress and relay any necessary feedback on the development process. This will play a crucial role in tracking the progress and ensuring work is being done on an incremental basis. Supervisor feedback will also play an important role in iterating and improving code that has already been worked on.

Software Requirements Specification

A software requirements specification will be created. This will define the objectives of the project while also clearly defining each of the use cases and their expected behaviours. This will detail the functional and non-functional requirements of the system. Wireframes will be drawn up for application. Once these use-cases and requirements are clearly defined, an initial test plan can be derived from this information.

Set up Environment and Design Wireframes

Developer Environment will be set up and application will be connected to database. Wireframes will be designed and linked together. Login page and User Authentication and authorization will be set up.

Implement Digital Signature Functionality First Milestone will be to set up digital signature functionality. This is the primary use case and most important feature.

Mid-Point Presentation

Mid-Point Implementation, Documentation & Video Presentation

Implement File Integrity Verification Functionality

Second milestone will be to set up File Integrity Verification functionality.

Implement Data Deduplication Functionality

Third milestone will be to set up Data Deduplication Functionality

Security Enhancements

Ensure application is robust and follows secure programming principles.

1.2 Monthly Reflective Journals

Supervision & Reflection Template

Student Name	Luke Banahan
Student Number	20116471
Course	BSHCYB4
Supervisor	Vanessa Ayala-Rivera

Month: October

What?

This month I pitched 2 different ideas for my software project. The first was an NCI student companion application. However, upon review with my supervisor, we decided that this was not the best choice of application to develop as the bulk of the work is Software Development focused, rather than my specialisation, which is cybersecurity.

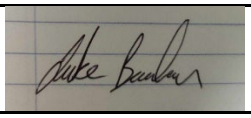
So What?

After some thought, and discussion with my supervisor, we came up with the idea of creating a web application that serves as a platform for Secure Document signing and File Integrity Verification. This is more in line with my specialisation.

While I am happy to have decided on the application I want to make, I am yet to meet with my supervisor to discuss how best to proceed.

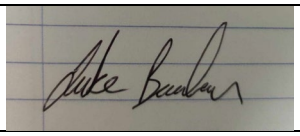
Now What?

I have reached out to my supervisor and scheduled a meeting for the 16th of November. In the meantime I will research the problem and the technologies needed to solve it to the best of my abilities. I hope to proceed further with my planning once I have met with my supervisor, who can offer guidance and what I should do next.

Student Signature	

Month: November

What? I made my initial Project Proposal, which included an early version of my Project Plan and Testing Plan. I then met with my supervisor to talk about my proposal, plan and how to proceed further.
So What? I found some of the technologies and libraries I believe will be most effective in allowing me to solve this problem. My supervisor acknowledges that the technologies I researched are suitable for solving the problem, but believes it is best if I continue researching in order to gain a proper understanding of the process of the problem.
Now What? I will now draw up design diagram to gain a better understanding of the problem at hand. I will identify inputs, processes, and outputs. I will also clearly identify and understand use-cases and requirements.

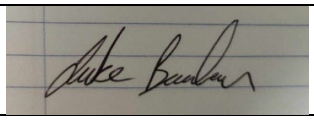
Student Signature	

Supervision & Reflection Template

Student Name	Luke Banahan
Student Number	20116471
Course	BSHCYB4
Supervisor	-

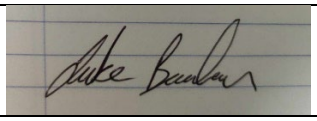
Month: December

What? Reflect on what has happened in your project this month? This month I developed the prototype for the front end of my application. I also completed my midpoint documentation and presentation.
So What? Consider what that meant for your project progress. What were your successes? What challenges remain? Now, I need to implement functionality for the login and registration. I will then work on the functionality of the digital signature generation and verification.
Now What? What can you do to address outstanding challenges? To address these challenges, I will continue to research and develop these functionalities.

Student Signature	

Month: January

What? Reflect on what has happened in your project this month? I have developed some of the login and registration functionality.
So What? Consider what that meant for your project progress. What were your successes? What challenges remain? This means I have made progress in one of the functional requirements of my project. I will continue to develop this part of the project to add salting to the registration to ensure that my application is secure.
Now What? What can you do to address outstanding challenges? I will continue to research proper salting implementation according to industry standards.

Student Signature	

Supervision & Reflection Template

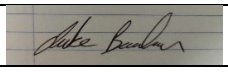
Student Name	Luke Banahan
Student Number	X20116471
Course	BSHCYB4
Supervisor	Arghir Moldovan

Month: February

What?
This month I got to work on implementing the core features of my application. I set up a project in Spring Boot and set up my database with all the necessary tables and columns. Once this was done, I got to work on implementing security into my application. Spring Boot comes with security features built into it however there is quite a bit of configuration and customisation to be done in order to get it to work to this application's specific needs. I have currently implemented the registration and login functionality of the application; however, some work is still needed in order to have proper session handling.

So What?
Next, I will continue to implement proper session handling as it will be an important feature of how the application handles the data that is shown to each user. Then I will proceed to work on document uploads, storage and retrieval, and then I implement the signing and verification of these documents.

Now What?
For now, I will continue to research session handling in Spring MVC projects and figure out how to customize it to work in my application. I will do this by reading spring documentation, looking through other similar projects on github and watching relevant tutorials online.

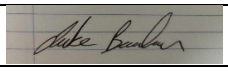
Student Signature	

Supervision & Reflection Template

Student Name	Luke Banahan
Student Number	X20116471
Course	BSHCYB4
Supervisor	Arghir Moldovan

Month: March

What? Implemented session handling that knows when a user is authenticated and allows user to sign out from current session.
So What? This is extremely important to how the application will run as it will only display information from the database that is relevant to the user who is currently signed in.
Now What? Now I can continue working on the rest of the functional requirements. The first and most important functional requirements are document upload, sign document and verify document. I will begin work on these straight away.

Student Signature	

Supervision & Reflection Template

Student Name	Luke Banahan
Student Number	X20116471
Course	BSHCYB4
Supervisor	Arghir Moldovan

Month: April

What? Implemented all functional requirements. The only thing left to add is an email service, which is an added functionality to 'Send Document' requirement.
So What? I can begin work on the email service. Once this is done, I can begin testing the application.
Now What? I will continue working on adding the email service to the application. Once this is done I will begin testing all of the functional and nonfunctional requirements of the application.

Student Signature