National College of Ireland

Bachelor of Science (Honours) in Computing

Cyber Security - BSHCYB4

2021/2022

Gavin Corr

x18382836

x18382836@student.ncirl.ie

Dragon Pass – Password Manager

Final Report

Contents

Executive Summary 2

1.0 Introduction 3

1.1. Background 3

1.2. Aims 3

1.3. Technology 4

1.4. Report Structure 4

2.0 System 5

2.1. Requirements 5

2.1.1. Functional Requirements 5

2.1.1.1. Requirement 1: Login and User Security 5

2.1.1.2. Description & Priority 5

2.1.1.3. Use Case 6

2.1.1.4. Requirement 2: Secure Registration 7

2.1.1.5. Description & Priority 7

2.1.1.6. Use Case 7

2.1.1.7. Requirement 3: Secure Password Manager 8

2.1.1.8. Description & Priority 8

2.1.1.9. Use Case 9

2.1.1.10. Requirement 4: Auto-Change feature 10

2.1.1.11. Description & Priority 10

2.1.1.12. Use Case 10

2.1.2. Data Requirements 12

2.1.3. User Requirements 12

2.1.4. Environmental Requirements 13

2.1.5. Usability Requirements 13

2.2. Design & Architecture 13

2.3. Implementation 16

2.4. Graphical User Interface (GUI) 23

2.5. Testing 27

2.6. Evaluation 36

3.0 Conclusions 37

4.0 Further Development or Research 37

5.0 References 38

6.0 Appendices 38

6.1. Project Proposal 38

6.2. Reflective Journals 42
6.3. Invention Disclosure Form 43

Executive Summary

The purpose of this report is to breakdown and demonstrate how I developed a password manager for my project and why I decided to do so. The purpose of the Dragon Pass password manager is to dynamically change user passwords on a selected time interval to protect the user's passwords from company leaks, company breaches, phishing attacks, and password cracking methods. The reason I decided to create this application was due to my work experience in Beaumont as part of my 3rd year in my Computer Science degree. During the HSE cyber attack in 2021, to protect user accounts and stop the virus from spreading, our team changed all user passwords within the hospital manually on a 24-hour basis. This was very time consuming but ensured that the hospital IT systems would remain protected and secure. This gave me an idea to securely automate this process and create a password manager that would dynamically change any chosen passwords by the user. DragonPass functions like any other password manager in its convenience, however, it adds an extra layer of security by dynamically and securely changing the user's chosen passwords. This dynamic extra layer of security is called 'Auto-Change'. Auto-Change is what sets this password manager apart from any other. In this report I describe in depth how I developed, tested, and designed this project and show the relevant results of the development. The main goal and purpose of this project was to offer users a more secure and convenient means of protecting their passwords for various applications.

1.0 Introduction

1.1. Background

In 2021, during my work experience in Beaumont Hospital IT for my 3rd year in Computer Science, the hospital became a victim of the HSE cyber-attack. The hospital was able to successfully protect and secure its systems and data due to our team implementing relevant and important secure practices within the hospital and IT environment. One of the keyways we protected the hospital, staff and patients was by changing all account passwords every 24 hours. This meant that any accounts that were infected or at risk would be isolated and secured and as a result, stopping the virus from spreading. This implementation was extremely successful; however, it was also very time consuming and required a lot of IT staff to coordinate. As part of the team that coordinated this effort, I came up with an idea that would automate this process. If users within a company or even a household environment only needed to remember one password, pin and use two-factor authentication to access their password manager and then the password manager dynamically changed all other account passwords, it would lower the risk of company password leaks, phishing attacks, database breaches and other various malicious activities from affecting the user. If any passwords are accessed by a malicious user, they would have a very short time frame to take advantage of it as the password would be constantly changing.

1.2. Aims

The goal of this project is to create a secure and dynamic password manager that has all the same convenient features as any other password manager but also has the added and unique ability of dynamically changing user passwords based on a set time interval. For example, if a user sets the time interval on a selected account to 30 seconds, the password manager should then change that password to a new secure password every 30 seconds. The purpose of this feature is to add an extra layer of security to password management. Current password managers on the market do not fully protect users from data breaches, password leaks and phishing attacks. A user is sometimes notified of potential security threats on other password managers, but my project aims to act in these scenarios by constantly changing the user's password for them. A user can't leak or reveal their password as they do not know the password themselves until they check the password manager. Companies that fall victim to data breaches and password leaks will also be countered as any leaked user passwords would be change before a malicious user could take advantage of it. The project also aims to implement and provide reliable cyber security practices and technology to offer the best security possible for the user. Security and confidentiality are key aspects of the development and implementation of this project.

1.3. Technology

The main technologies used in the development and implementation of this project are Java, JavaFX, IntelliJ IDEA, MySQL, SQLite, and Google Authenticator. JavaFX allows for smooth and effective GUI design, which improves overall use and user experience. MySQL is used for database structure and development. SQLite has been used for error handling and monitoring the application. Google Authenticator has been used to add an extra layer of security to the login system. As the application is a password manager, it is of utmost importance that the login process is extremely secure. Two factor authentication with Google Authenticator ensures that the users use their mobile device to complete the login process, meaning a malicious user would also need access to their mobile device additionally with the password and pin. Also included in the registration and login process is the SHA-256 hashing algorithm. This cryptographic hashing method creates an irreversible unique hashed string which allows for user verification but also protects the integrity of the user's actual password. For password changing and retrieval within the password manager, AES-GCM encryption has been implemented. AES-GCM is a block cipher operation that allows for authenticated users to encrypt and decrypt their data at high speed while maintain password integrity. The type of encryption used for this project is password-based encryption, which uses the users account password as the encryption key. All these technologies ensure that Dragon Pass accounts are protected at a high level and ensures the integrity and confidentiality of user password data.

1.4. Report Structure

This report will be discussing and demonstrating how Dragon Pass works and functions. System requirements and use cases will be showcased to explain how the application should be used and how's its intended functionality works. An in depth look at the architecture of various algorithm used within the project will also be reported and accompanied by relevant diagrams and snippets. Each important feature that makes up Dragon Pass will be explained in depth with coded examples and text descriptions. The GUI and all viewable features of the application will also be presented in this report and accompanied by a description of how they function. This report will also detail how testing was performed, how the results were evaluated and the results themselves presented. Finally, a conclusion that describes the overall advantages and disadvantages of the project, what future potential the project holds and a short summary of the report. An appendix can also be found at the end of the report that includes further information and resources regarding the Dragon Pass project. All report sections can be easily navigated via the Contents list on page one.

2.0 System

2.1. Requirements

Requirements involve the following:

- a secure login and registration.
- A secure password management system
- A secure dynamic password changer
- Confidentiality
- Integrity
- Usability
- Adaptability
- Accessibility
- Overall security

2.1.1. Functional Requirements

The functional requirements for this project are ranked starting from the highest to lowest requirement. The ranking of these requirements is based on the prioritization of user security, integrity, and confidentiality.

1. Login and user verification security (Highest priority)
2. Secure registration
3. Secure password manager for storing passwords
4. Auto-Change feature for dynamic password changing

2.1.1.1. Requirement 1: Login and User Security

2.1.1.2. Description & Priority

Due to the sensitive nature of a password manager and the data that it stores, the login and user verification must be of the highest priority. If any malicious user gains unwarranted access to a user account, it would defeat the main goal and purpose of the application. For the rest of the application to function at a high level, this requirement must be fulfilled. SQL injection and brute forcing must be countered by the login system to protect its users and the overall integrity of the system.

2.1.1.3. Use Case

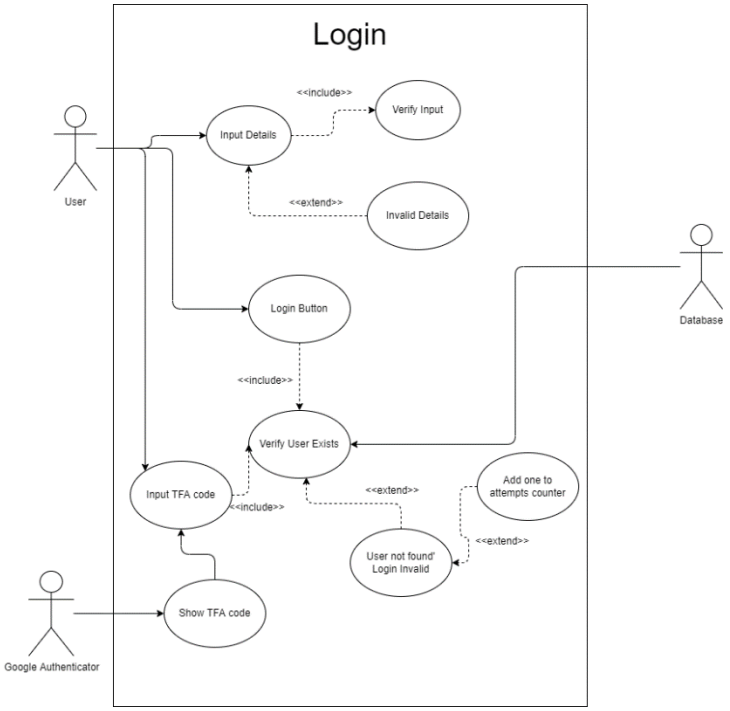
Scope

The scope of this use case is to provide a user with the knowledge of how to use the login system and show how the login system verifies and validates the necessary information required.

Description

This use case describes the relationship between the login system, the user, and the database. It shows the various security methods implemented to safely secure user account information and prevent malicious users from gaining unwanted access to the password manager.

Use Case Diagram



Flow Description

The user inputs details and the application verify the input. The user receives a TFA code from Google Authenticator and inputs along with the verified credentials. They select the login button, and the database verifies if the user exists based on the provided details. If a user is not found, the login attempt counter increases by one. If the user is found in the database, the use case terminates, and the next process begins.

Precondition

The system is in initialisation mode when the user enters the Login Controller.

Activation

This use case starts when a user inputs a string into a text field.

Termination

The system terminates when the user verification has been accepted or denied.

Post condition

The system goes into a wait state for further user input.

2.1.1.4. Requirement 2: Secure Registration

2.1.1.5. Description & Priority

Like with the secure login requirement, the registration requirement is just as important regarding securing user account information for the password manager. To stop users from obtaining user account information, it is vital that user sensitive data such as their account password and pin is hashed. Storing user sensitive data and preventing security vulnerabilities such as buffer overflow, SQL injection and bot account creation, this requirement is necessary.

2.1.1.6. Use Case

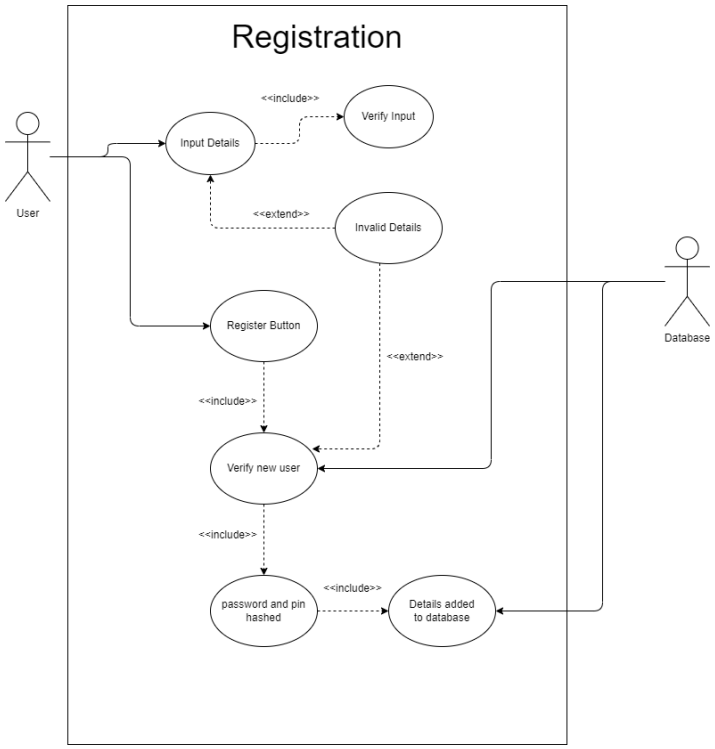
Scope

The scope of this use case is to provide a user with the knowledge of how to use the registration system and safely create an account using the provided resources such as two factor authentication.

Description

This use case describes the relationship between the registration system, the database, and the application. It describes how sensitive data is hashed and secured.

Use Case Diagram



Flow Description

The user inputs details, the user selects the register button, the system verifies the inputs, the system also checks the database to see if the user account already exists, the user password and pin is hashed, the details are stored to the database.

Precondition

The system is in initialisation mode when the user enters the Registration Controller.

Activation

This use case starts when a user inputs a string into a text field.

Termination

The system terminates once user registration is verified, and user details are added to the database.

Post condition

The system goes into a wait state until another user accesses the register controller.

2.1.1.7. Requirement 3: Secure Password Manager

2.1.1.8. Description & Priority

The password manager is the backbone of the application as it stores and manages all user account passwords. The encryption and storing of this sensitive data are essential for the user security and the main Auto-Change feature to work effectively. Ensuring that only the verified user can access their own passwords is essential.

2.1.1.9. Use Case

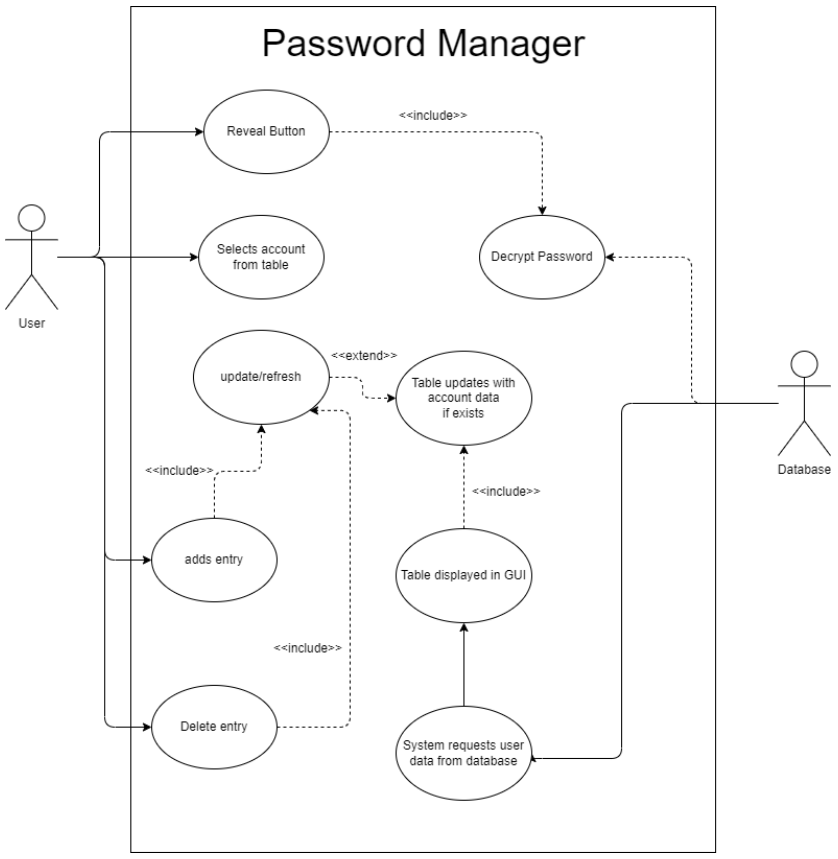
Scope

The scope of this use case is to provide a user with the knowledge of how to use the Password manager and understand the functionality behind it. It also describes how the manager effectively secures their account passwords.

Description

This use case describes the relationship between the password manager, the database, and the user. The use case also shows how the password manager secures the information put into it.

Use Case Diagram



Flow Description

The system calls the database to fill in the table view for the user, the table is then displayed in the GUI, the table then updates the data in the table automatically, the user can add and delete entries from the table, the user can also select the reveal button to decrypt the selected account password. Any passwords added to the table are encrypted automatically.

Precondition

The system is in initialisation mode when the user enters the Manager Controller.

Activation

This use case starts when a user selects an item or button within the password manager.

Termination

The system terminates when the user exits the controller.

Post condition

The system goes into a wait state until the user re-enters the controller

2.1.1.10. Requirement 4: Auto-Change feature

2.1.1.11. Description & Priority

The Auto-Change feature dynamically changes user passwords securely and conveniently for any chosen user account. This feature is offering a high level of password security for the user. It is essential for the main goal of the application, however, is ranked lower than the other requirements as it offers the user a high level of protection when using 3rd party applications but offers less to the Dragon Pass system itself. Regardless, it is still an essential requirement for the user's overall security when using other applications. Overall, this is the most important feature of the application, however, it requires all the other requirements to be fully implemented and functional before it can work as intended.

2.1.1.12. Use Case

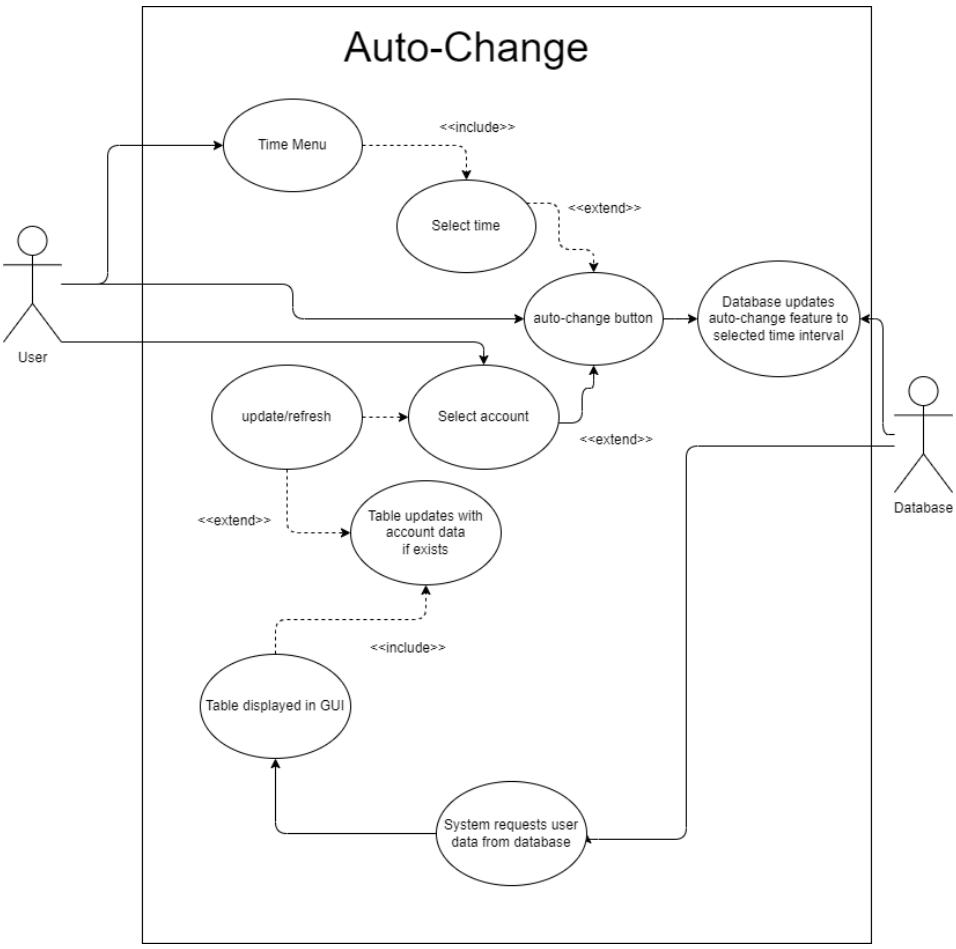
Scope

The scope of this use case is to provide a user with the knowledge of how to use the Auto-Change feature. In doing so, they will be equipped with a unique and secure password management solution by dynamically changing their passwords on selected time intervals.

Description

This use case describes how the Auto-Change feature works and how a user would interact with the feature. Furtherly, it shows the relationship between the user, the database, and the application itself.

Use Case Diagram



Flow Description

The user accesses the time menu and selects a time, the database loads up the table view and its related data, the user selects an account loaded from the table view, the user selects the auto-change button, the database updates the auto-change feature based on the selected time interval for the selected account. This means that the selected password will now change every time the selected time interval loops.

Precondition

The system is in initialisation mode when the user enters the Auto-Change controller.

Activation

This use case starts when a user selects an account or time interval from the time menu for Auto-Change.

Termination

The system terminates when the user leaves the controller.

Post condition

The system goes into a wait state until the user re-enters the controller.

2.1.2. Data Requirements

For the password manager to work, a user will need to provide a password, pin, username, email and generate a secret key upon registration. The login system requires hashed data of the user's password from registration to verify the user. The login system will additionally require the users two factor authentication secret key. When encrypting and decrypting data, the password manager will also require the user's password to use as a secret key. This password will be required again upon using the auto-change feature to encrypt and decrypt the information again using AES-GCM encryption.

2.1.3. User Requirements

The user requirements for this project include providing the following for the end user:

- Reliability
- Security
- Confidentiality
- Integrity
- Functionality
- Usability
- Convenience
- Adaptability
- Availability
- Consistency

2.1.4. Environmental Requirements

This application has been designed with no audible requirements or features; therefore, no external sounds or interruptions will affect user experience. Alerts are clear, precise and the application design is colourful, making visuals much easier to process regardless of visual environmental effects such as low brightness.

The main environmental requirements for this project are the following:

- Visually appealing and colourful design
- Designed without the use of audible cues
- Clear direction of application use
- Designed to work securely in any type of environment with secure validation and authentication methods implemented

2.1.5. Usability Requirements

The user interface has been designed in a way that offers ease of use and similar patterns that makes usability familiar and easier for users such as repeated use of text fields and buttons. Notes and labels have been provided for user convenience. Only necessary features are implemented as too many features or information can overcomplicate design and confuse or intimidate the user.

The main usability requirements for this project are the following:

- Efficiency and Ease of use
- Low Error encounters
- Simple UI design with not intimidating or complex factors. In this case, the less the better
- Intuitiveness of the UI by using similar patterns and designs such as buttons, text fields and simple table

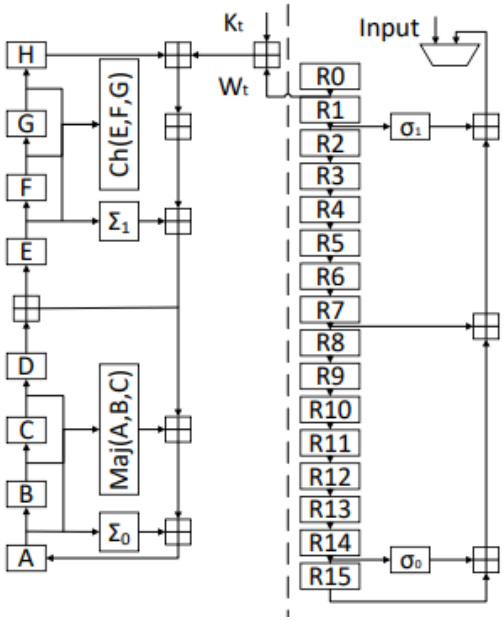
2.2. Design & Architecture

Dragon Pass follows a secure design pattern by using various encryption, hashing and security features to ensure no malicious users can access non-authorized user data. Due to the project being a password manager, the encryption and hashing of passwords is a vital and key aspect of the design and architecture of the application to ensure user integrity and confidentiality while still offering a convenient and useful product. The three key algorithms used in this project are SHA-256 hashing for user authentication, AES-GCM encryption for the password manager and the TOTP algorithm for two-factor authentication during the login process. Both the hashing and encryption algorithms are used for the Auto-Change feature, which is a key aspect of this project.

SHA-256 Architecture:

To verify the user while maintaining password confidentiality and integrity, the SHA-256 algorithm is used to hash the user password. The hashed password stored in the database upon registration can be matched with the hashed user input upon login to verify the user without breaking user confidentiality by protecting and hiding the user's actual password. SHA-256 hashing pre-processes the users input by taking the plain text, padding it (appending bits based on rules until the total length of the integer reaches 512-bit) and expanding the text (expanding each 512 bits to 64x32) for the round computation [1]. The round computation process can be seen left of the dotted line in the diagram below [1]:

[1]



The parallel hardware design used in SHA-256 hashing architecture is ideal for reaching high processing speeds while also maintaining a low power consumption in the computer hardware [1]. In this project implementation, the user's password and pin are hashed for user authentication. The algorithm is also used for third-party applications (test applications) to access newly changed passwords from the Auto-Change feature in this project.

AES-GCM Architecture:

AES-GCM is the primary component used for the encryption and decryption of user passwords within the database of this project. The key length used in this project is 128-bits. AES encrypts a 128-bit of a string or decrypts 128-bit of a ciphertext by applying the same round of transformation on repeat depending on the key size specified [2].

The standard number of rounds for 128-bit key length is 10. The larger the key length, the longer deciphering and encryption takes.

The efficiency and overall performance of this algorithm can be accurately calculated from the time it takes for a block cipher to enter decryption or encryption to the time it leaves this process [2].

The performance of this algorithm can be reviewed by its latency. To calculate the latency of the algorithm, the following mathematic notation can be used [2]:

Latency = Cycles per Encrypted Block (C) x Time period (T)

'C' can be described as the clock per byte for the encryption of a cipher block while 'T' can be described as the reciprocal of the frequency of the clock. The latency of the AES-GCM algorithm is 7.6ns [2].

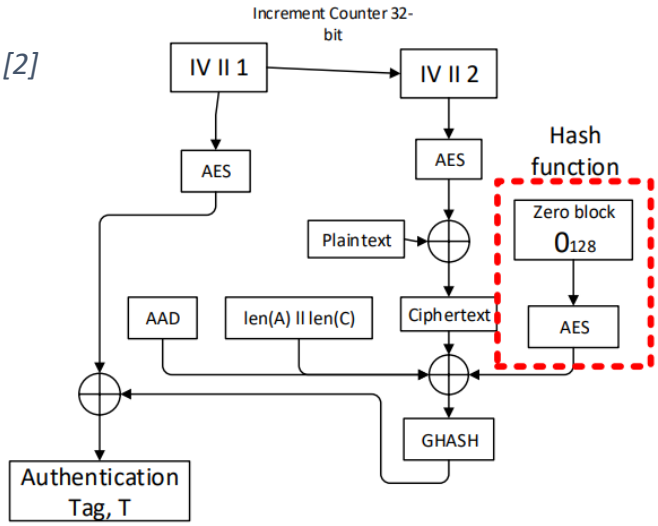
The throughput shows the speed of the algorithm's decryption and encryption processes. The throughput is obtained by calculating in bit per second(bps). The calculation is shown below [2]:

Throughput = 128 bits/latency

= 128 / (7.6×10^{-9})

= 16.84 Gbps

The AES-GCM algorithm is designed to output high speed encryption and decryption while still maintaining plaintext integrity and confidentiality. The below diagram shows the architectural structure of the AES-GCM algorithm [2]:



[2]

The implementation of AES-GCM uses a password based key solution, requiring the users account password to encrypt and decrypt data. This is used for the creation (encryption) and retrieval (decryption) of the user's stored passwords in the password manager.

TOTP Architecture:

For two-factor authentication, the TOTP (Time-based one-time password) algorithm was used in conjunction with Google Authenticator, which helps implement the TOTP algorithm for mobile devices. The TOTP algorithm is variant of the HOTP algorithm that

specifies the calculation of a one-time password value which is based on a representation of a counter as a time factor [3]. X represents a time step in seconds with the default value of X being 30 seconds, meaning the code for two factor authentication will change every 30 seconds by default. X is defined as a system parameter in the TOTP algorithm [3]. T0 can be described as the Unix time to start counting the time steps with the default value being 0, the Unix epoch, and is also defined as a system parameter for the TOTP algorithm [3]. By implementing the TOTP algorithm within this project, it requires the user to store a secret key on their Google Authenticator app on their mobile phone. The Dragon Pass application stores their key and can then match the two codes generated every 30 seconds to verify the identity of the user. This adds an extra layer of security to the login process as a malicious entity would require the user's physical phone additionally with the user's password and pin.

2.3. Implementation

Login and Registration:

For the login and registration implementation, user integrity and confidentiality are top priority. To achieve a high level of security for this area various methods have been used. These methods include prepared SQL statements, secure input parameters, SHA-256 password and pin hashing, limited user interaction, a login attempt counter and two-factor authentication. When a user logs in their name will be stored by RAM memory in the 'UserSession' class to store any relevant information required such as database password retreival from the user specific account.

Prepared SQL statement for user login verification:

```
//Prepared sql statement to prevent SQL injection
try{
    monitor.setLog(new Logger( level: "info",  message: "Verification status requested"));
    String query = "select * from user_account where username = ? and password = ? and pin = ?";

    DatabaseConnection connect = new DatabaseConnection();
    conn = connect.getVerification();
    stmt = conn.prepareStatement(query);

    //Parameters
    stmt.setString( parameterIndex: 1, username);
    stmt.setString( parameterIndex: 2, password);
    stmt.setString( parameterIndex: 3, pin);

    rs = stmt.executeQuery();

    //If account match is found
    if(rs.next()) {

        monitor.setLog(new Logger( level: "info",  message: "Account verified"));
```

In this snippet, the user inputs are being stored to String variables and being inserted into a prepared SQL statement. This stops malicious users from injecting their own code by treating their input as a string and strictly checking to see if the string matches the inputs made by the user. If the inputted string text doesn't match a user within the database, the request is simply denied, and the user must try again. This is one of many prepared statements within the project.

Security input parameter for user registration:

```
else if(pinTF.getText().equals(confirmPinTF.getText()) && !pinTF.getText().isBlank()) {

    String input = pinTF.getText();
    boolean flag = true;
    for (int a = 0; a < input.length(); a++) {
        if (a == 0 && input.charAt(a) == '-')
            continue;
        if (!Character.isDigit(input.charAt(a)))
            flag = false;
        pinMatch.setText("Pin must contain 4 digits!");
    }
    if (flag) {
        pinMatch.setText("✓");
        validPin = true;
    }
}
```

In this snippet, it shows an example of a security parameter set up for pin inputs in the registration controller. The 'IF' statement grabs the text inputted into the pin password field and the confirm pin text field to see if both pins match. It also checks to see if the password field is blank. Both must be satisfied before the statement can continue. When these arguments have been satisfied, the application then runs a loop to look for digits in the input and checks if the pin is a length of 4 digits. If there are any inputs other than a digit or the length does not have 4 digits in total, the input will be flagged as false and will not be accepted. If the arguments are arguments are satisfied, the pin will be seen as valid and will wait for all other registration fields to also be valid before the account can be registered.

Two-Factor Authentication (TOTP):

```
import org.apache.commons.codec.binary.Base32;
import org.apache.commons.codec.binary.Hex;

public class TwoFactorAuthentication {

    //Obtaining a TOTP code using the Time based one time password algorithm
    public static String getTOTPCode(String secretKey) {

        Base32 base32 = new Base32();
        byte[] bytes = base32.decode(secretKey);

        String hexKey = Hex.encodeHexString(bytes);
        return de.taimos.totp.TOTP.getOTP(hexKey);

    }
```

This above snippet shows the TOTP algorithm being implemented to run the algorithm based on the user's secret key. A 6-digit code can then be created to match with the user input for

verification as they should have the same code generated on their Google Authenticator mobile app as they were required to input the secret key upon registration.

```
public static String generateSecretKey() {
    SecureRandom random = new SecureRandom();
    byte[] bytes = new byte[20];

    random.nextBytes(bytes);

    Base32 base32 = new Base32();
    return base32.encodeToString(bytes);

}
```

This 'generateSecretKey' method is called to generate the secret key for the user. The key is then stored within the database to verify the user's identity. This allows for further security in the login process, where a malicious user would require the users secret key additionally with their password and pin.

SHA-256 Hashing:

```
//SHA-256 hashing
public static byte[] getSHA(String input) throws NoSuchAlgorithmException {

    MessageDigest md = MessageDigest.getInstance("SHA-256");

    return md.digest(input.getBytes(StandardCharsets.UTF_8));

}

public static String toHexString(byte[] hash){

    BigInteger number = new BigInteger( signum: 1, hash);

    StringBuilder hexString = new StringBuilder(number.toString( radix: 16));

    while(hexString.length() < 32){

        hexString.insert( offset: 0,  c: '0');

    }

    return hexString.toString();

}
```

This snippet shows the implementation of the SHA-256 hashing algorithm where a hashed string based on the algorithm is returned.

```
String password = Hash.toHexString(Hash.getSHA(passTF.getText()));
String pin = Hash.toHexString(Hash.getSHA(pinTF.getText()));
String key = secretKey.getText();


String query = "INSERT INTO user_account(username, email, password, pin, tfaKey, session) VALUES(?,?,?,?,?,0)";
```

The above snippet shows the Hash class being called and hashing the users input for the password and pin fields. These are then inserted into the database via prepared statement once all parameter arguments are satisfied.

Login and Lock attempt counter:

```
else {

    counter = counter + 1;

    int result = 3 - counter;


    invalidLabel.setText("Incorrect. Login attempts remaining: " + result);

    monitor.setLog(new Logger( level: "info",  message: "Invalid account details"));
```

The above snippet shows the counter that gives the user a total of 3 attempts to login. If they input invalid details 3 times, the application will close. This is helpful against bots and other scripts used by malicious users to brute force attack the app to access a user account. This same counter method is applied to the application lock function, that allows the user to lock the app but keep it running in the background. Similarly, to the login controller, the lock requires the users pin and 2FA key to unlock the application. If the details are invalid in the lock a total of 3 times in a row, the lock will close the application.

Password Manager/Auto-Change:

For the password manager implementation, similar coding methods such as secure parameters and SQL prepared statements were used, however due to the confidential nature of password storing, it was important to use a reliable and secure encryption methods where the user could encrypt and decrypt their chosen passwords within the manager on demand. This requires AES-GCM encryption. For the Auto-Change feature, SHA-256 hashing was also used to update any account passwords within the manager and the encryption methods was used again to encrypt changed passwords that were store within the manager itself so that the user can securely decrypt their passwords in the list on demand.

AES-GCM Encryption:

```java
//Return a base64 encoded AES encrypted text
public static String encrypt(byte[] pText, String password) throws Exception {

    //16 byte salt
    byte[] salt = CryptoUtils.getRandomNonce(SALT_LENGTH_BYTE);

    //GCM recommended 12 bytes iv
    byte[] iv = CryptoUtils.getRandomNonce(IV_LENGTH_BYTE);

    //Secret key from given password
    SecretKey aesKeyFromPassword = CryptoUtils.getAESKeyFromPassword(password.toCharArray(), salt);

    Cipher cipher = Cipher.getInstance(ENCRYPT_ALGO);

    //AES-GCM needs GCMParameterSpec
    cipher.init(Cipher.ENCRYPT_MODE, aesKeyFromPassword, new GCMParameterSpec(TAG_LENGTH_BIT, iv));

    byte[] cipherText = cipher.doFinal(pText);

    //Prefix IV and Salt to cipher text
    byte[] cipherTextWithIvSalt = ByteBuffer.allocate(iv.length + salt.length + cipherText.length)
            .put(iv)
            .put(salt)
            .put(cipherText)
            .array();

    //String representation, base64, send this string to other for decryption.
    return Base64.getEncoder().encodeToString(cipherTextWithIvSalt);

}
```

This snippet shows the implementation of the encryption method using the AES-GCM algorithm. This method takes the specified password as the secret key and will encrypt the password string.

```java
//The same password, salt and iv are required to decrypt the string
public static String decrypt(String cText, String password) throws Exception {

    byte[] decode = Base64.getDecoder().decode(cText.getBytes(UTF_8));

    //Retrieve the iv and salt from the cipher text
    ByteBuffer bb = ByteBuffer.wrap(decode);

    byte[] iv = new byte[IV_LENGTH_BYTE];
    bb.get(iv);

    byte[] salt = new byte[SALT_LENGTH_BYTE];
    bb.get(salt);

    byte[] cipherText = new byte[bb.remaining()];
    bb.get(cipherText);

    //Retrieve the AES key from the same password and salt
    SecretKey aesKeyFromPassword = CryptoUtils.getAESKeyFromPassword(password.toCharArray(), salt);

    Cipher cipher = Cipher.getInstance(ENCRYPT_ALGO);

    cipher.init(Cipher.DECRYPT_MODE, aesKeyFromPassword, new GCMParameterSpec(TAG_LENGTH_BIT, iv));

    byte[] plainText = cipher.doFinal(cipherText);

    return new String(plainText, UTF_8);

}
```

This snippet shows the decryption method. Decryption requires the same password, salt and iv to decrypt the string. This method is used when the user wants to view a selected password in their password manager.

```
String PASSWORD = UserSession.instance.getPasswordHash();


String app = appNameTF.getText();
String username = usernameTF.getText();
String hashPass = Hash.toHexString(Hash.getSHA(passwordTF.getText()));

//calls encryption class to encrypt user password
String password = Cypher.encrypt(passwordTF.getText().getBytes(StandardCharsets.UTF_8), PASSWORD);

Connection conn;

String query = "INSERT INTO " + UserSession.instance.getUserName() + " (app, username, password, passHashed) VALUES (?,?,?,?);";
```

The above snippet shows the password manager add password function. The users account password is being retrieved via RAM memory and being used as the encryption key by calling the Cypher class. The now encrypted password is then stored to database under the user's account.

```
public void revealTableAction(ActionEvent event) throws Exception {


    try {
        String PASSWORD = UserSession.instance.getPasswordHash();
        String pass = tableManager.getSelectionModel().getSelectedItem().getPassword();

        String decryptedPass = Cypher.decrypt(pass, PASSWORD);

        revealText.setText(decryptedPass); //Reveals decrypted password to verified user

    }
    catch(Exception e){
        monitor.setLog(new Logger( level: "warn", e.getMessage()));
    }
```

The above snippet here shows the Cypher class being called again when the password reveal button is clicked in the password manager. The user's password is once again used to decrypt the password and it is then revealed in the manager for the user to view.

```
//Encrypts generated string and sets it to a variable using the users account password as the encryption key
String password = Cypher.encrypt(generatedString.getBytes(StandardCharsets.UTF_8), PASSWORD);

//Hashes password for linked applications
String passHashed = Hash.toHexString(Hash.getSHA(generatedString));
```

Encryption, decryption and hashing methods are also applied to any passwords changed with the Auto-Change feature as seen in the snippet above. The encryption algorithm is used for the user to access newly changed passwords within the password manager. Hashing is used for any linked applications that the user has accounts for where the hash can be used to validate the user with their newly generated password.

Logging and Monitoring:

A logger class has been implemented in this project for security, error handling and testing purposes. The class monitors any activity specified within the application and reports back errors, user specific interactions and logs this information within a log file. This log file can be used to track malicious and suspicious activity or to track and report any errors found or specified. SQLite is used to in conjunction with the logger to provide relevant results.

```
@Override
public void update(Logger log) {

    System.out.println("File " + log.toString());
    try (FileWriter fw = new FileWriter( fileName: "log.txt",  append: true);
         BufferedWriter bw = new BufferedWriter(fw);
         PrintWriter out = new PrintWriter(bw)) {
        out.println(log.toString());
    } catch (IOException e) {
        monitor.setLog(new Logger( level: "warn", e.getCause().getMessage()));
    }
```

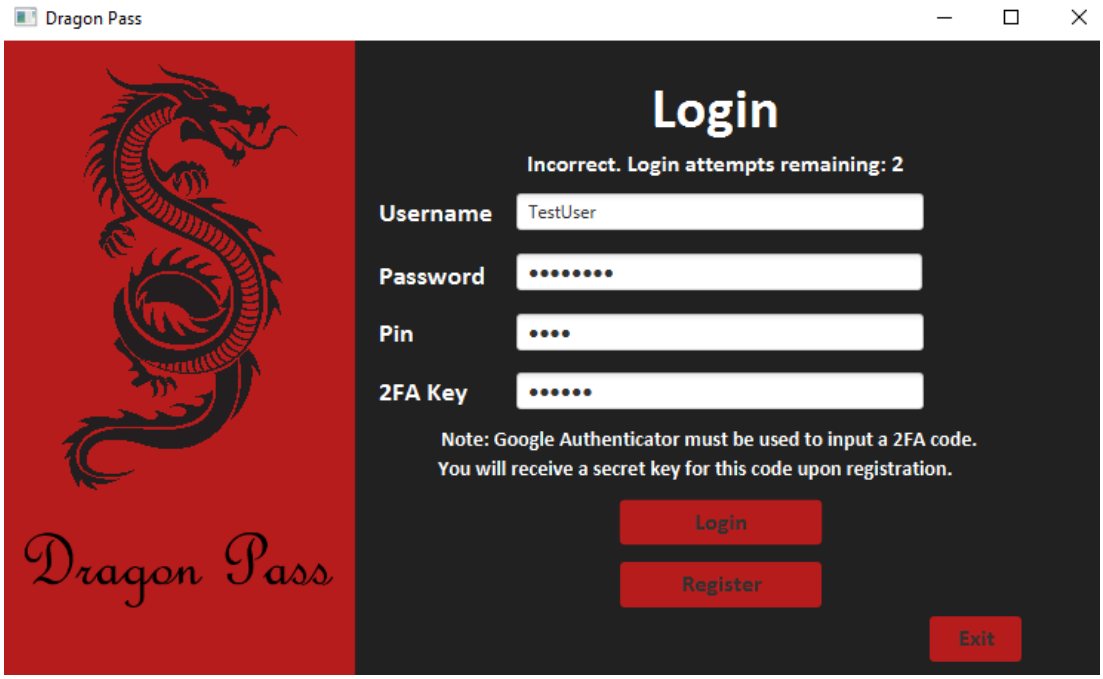
The above snippet shows the 'FileObserver' class updating the log information and shows an example of the monitor setting the log by obtaining the error cause. The monitor can be used in all try-catch implementations to display relevant error information.

```
public static void insert(Logger log) {
    String sql = "INSERT INTO monitor(level,message, occuranceTime) VALUES(?,?,?)";

    try (Connection conn = DriverManager.getConnection(url);
         PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString( parameterIndex: 1, log.getLevel());
        ps.setString( parameterIndex: 2, log.getMessage());
        ps.setString( parameterIndex: 3, log.getOccurenceTime().toString());
        ps.executeUpdate();
    } catch (SQLException e) {
        monitor.setLog(new Logger( level: "warn", e.getCause().getMessage()));
    }
```

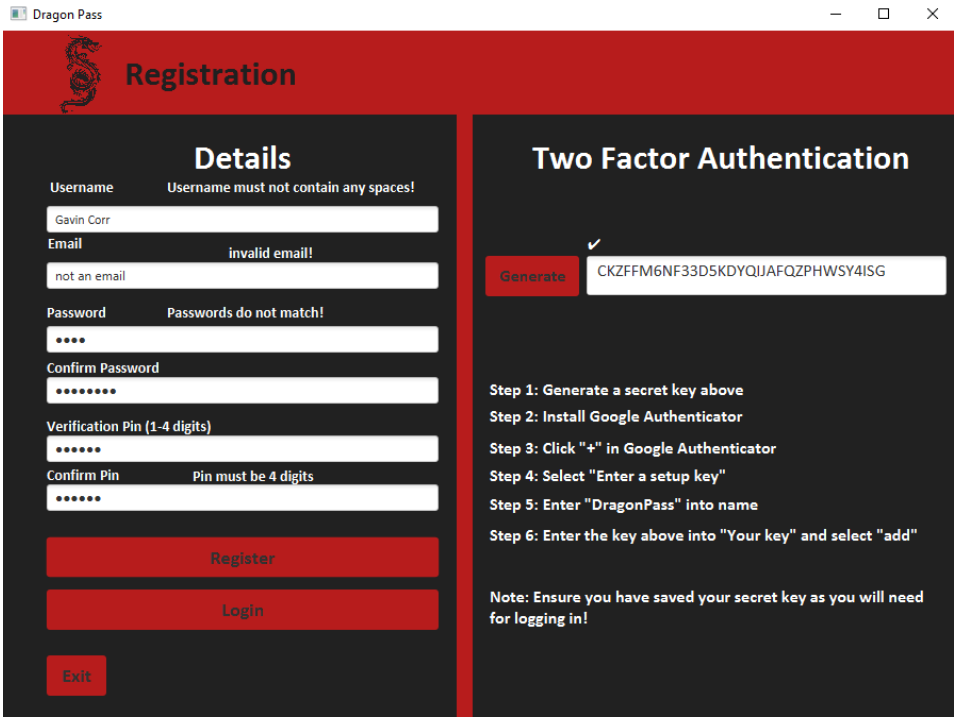
The snippet above here shows the SQLite class inserting a prepared statement of the monitor message into the database. This can then be stored to the log file and displayed during testing.

2.4. Graphical User Interface (GUI)

Login:



The above screenshot shows the login screen. The user has filled the login form and has received an alert via a label notifying them that they have input the wrong details, as there is no user within the database under these details. If they continue to input incorrect user details, the application will close after two more attempts



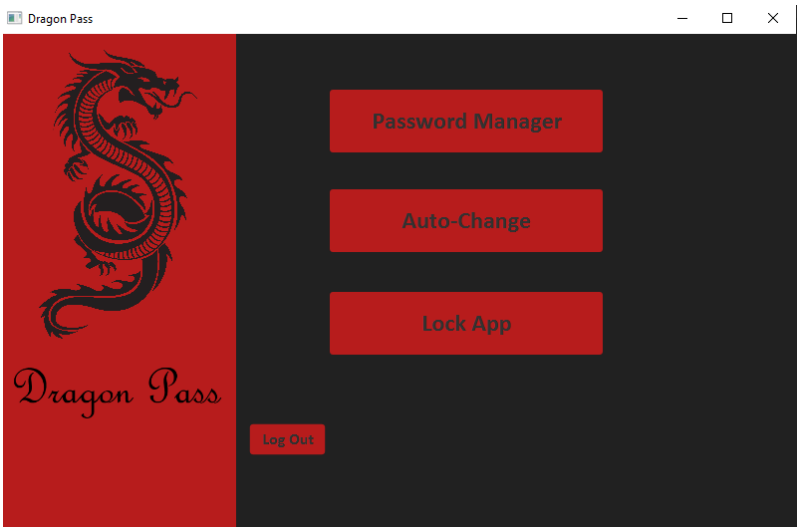
Register

This screenshot shows the registration GUI. Several alerts have been thrown and the registration request was denied. None of the input fields have the correct parameters except for the secret key generation text area. The username has a space which is not allowed, the email does not have the makeup of an actual email, for example: arealemail@example.com. If any of these features are missing, it will not be accepted.

The password and pin also do not follow the proper parameters as seen above. These are just a few examples of the registration parameter alerts, for example, another one would be that the username cannot be the same as an existing account within the database. A tick like the one seen above the secret key generation field notifies the user of correct input. Once all inputs are correct, registration is accepted, and an account is created. The Two Factor Authentication field to the right of the registration tab allows a user to generate a TOTP key that they can input into Google Authenticator. The text area is not editable to prevent user interaction with the secret key. Steps of how to set up two factor authentication with their received key is displayed below for the user's convenience.

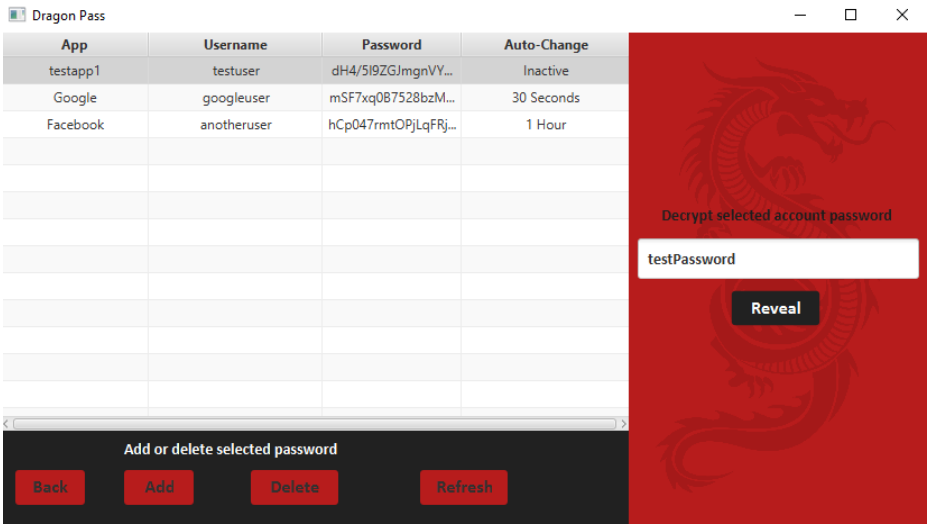
Secure parameters such as the ones shown above prevent buffer overflow and help counter SQL injection and bot account creation. Additionally, there is character limit on each text field. Every input within the registration page and the rest of the application uses prepared statements to prevent SQL injection and prioritize secure inputs.

Menu:



This screenshot shows the menu of the application which allows users to navigate the app via buttons after successful login. The logout button clears the user session. Note: the user session is also cleared upon application start-up and all user session data is stored via RAM.

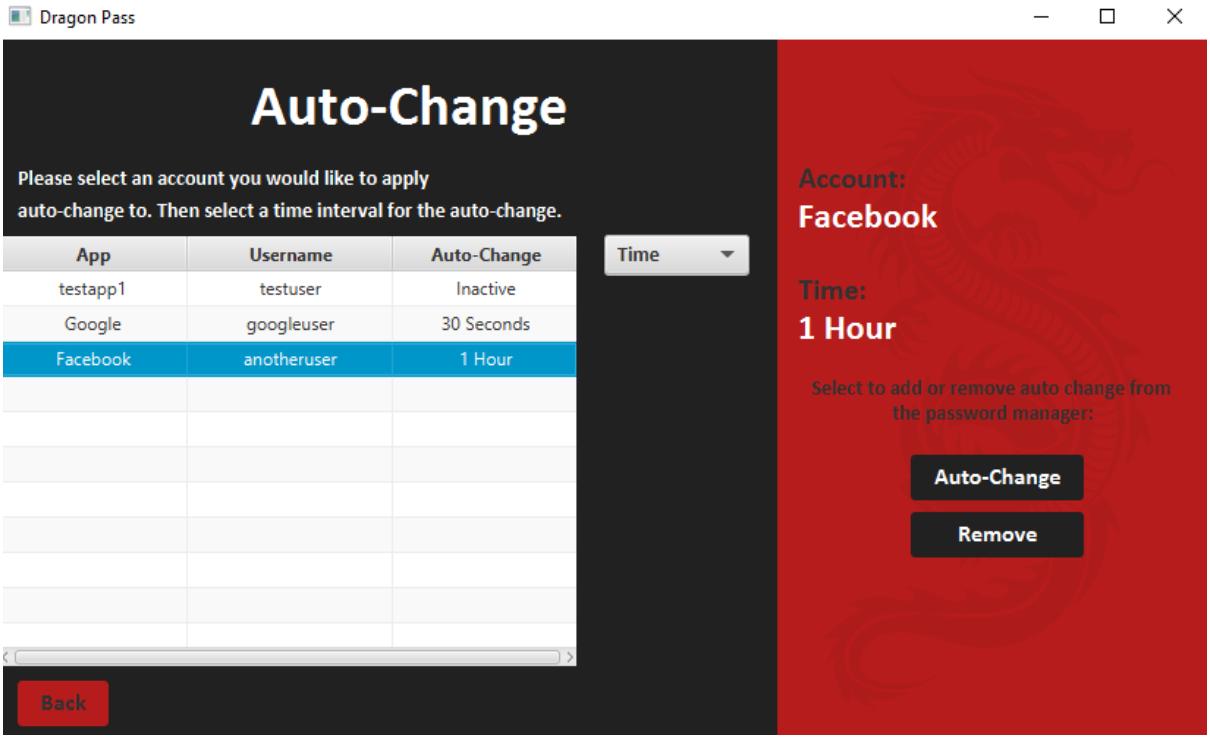
Password Manager:



This screenshot shows the password manager GUI. Using a table view, user password account information is taken from the database and displayed to the user. In the example above, the user has selected the "testapp1" account and has selected to reveal it to the right of the GUI. The table shows the encrypted password but when the 'Reveal' button is clicked, it will show the decrypted password.

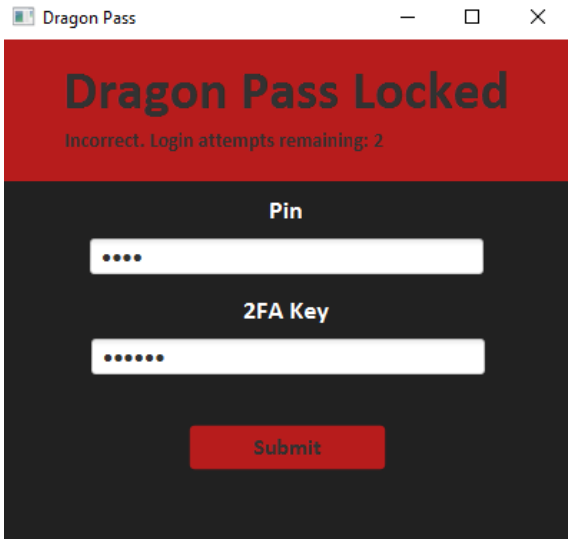
The user can use the add button to add more entries and the delete button to remove entries. The refresh button is used for any accounts that have Auto-Change enabled. Whenever a password is changed, the user can refresh to see the new password. In the above example, the Google and Facebook accounts have been set to change every 30 seconds and 1 hour respectively. The testapp1 account password will not change as the Auto-Change feature on it has been set to inactive.

Auto-Change:

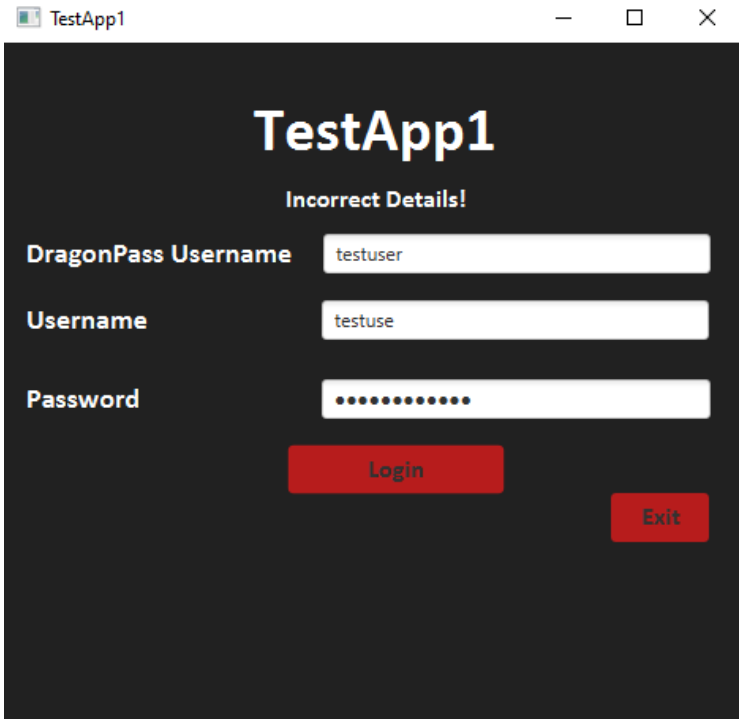


This screenshot shows the Auto-Change GUI. In this example, the user has selected the Facebook account from the password manager and chosen the 1-hour option from the time drop-down menu. By selecting the 'Auto-Change' button, the Auto-Change feature will be set to 1-hour, meaning that the password for that account will be changed to a randomly generated complex password every 1-hour. The time options that are available include: 30 Seconds, 5 Minutes, 15 Minutes, 30 Minutes and 1 hour. These are displayed via the 'Time' dropdown-menu. The user can remove the auto-change feature from a selected account within the password manager by using the 'Remove' button. This will cause the chosen account password to remain the same until the user re-applies the Auto-Change feature again.

Lock:

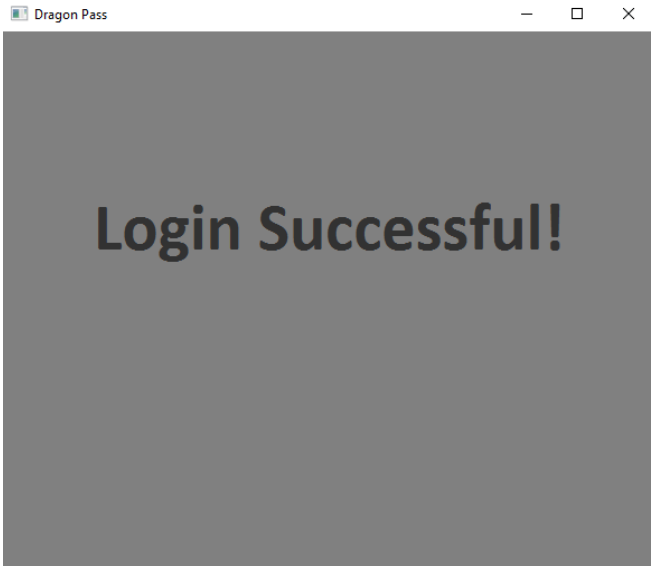


This screenshot shows the Dragon Pass Lock GUI. The lock allows the user to lock the application and allow it to securely run in the background. To unlock it, the user pin and 2FA key are required. Similarly, to the login GUI, the lock GUI has a login attempt counter and will close the application after 3 failed login attempts. This feature is useful if a user is temporarily absent from their device.



Test Applications:

This screenshot shows one of the test applications used to work in conjunction with the Dragon Pass application to demonstrate how the auto-change feature would work in a commercial environment. Using the same account details shown earlier in the password manager GUI, the username has been intentionally typed incorrectly to demonstrate the validation system in place. An alert notifying the user of incorrect details appear as the user Is missing ‘r’ at the end of the username field.



When the user successfully logs in using the correct details, they will receive an alert show in this screenshot. This test application verifies that the password manager is working correctly.

2.5. Testing

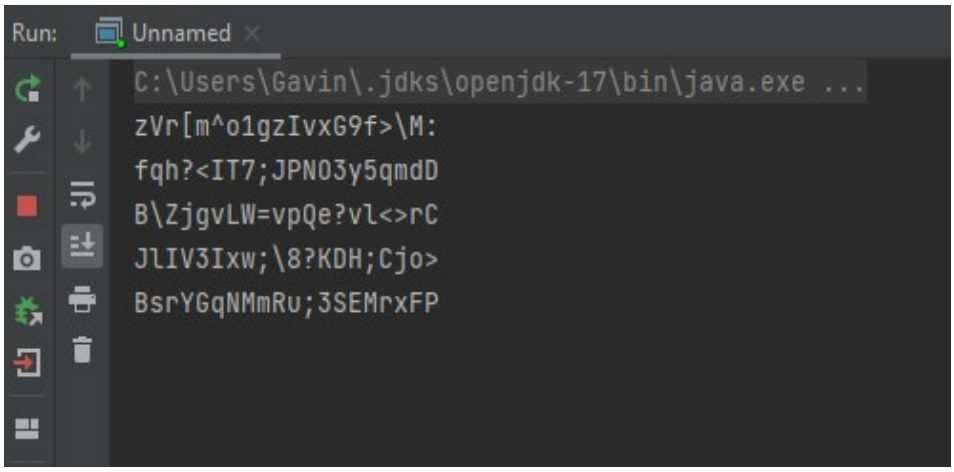
Unit Testing:

During development of this project various testing techniques were used. One of these is unit testing. By creating standalone applications to test small cells of feature within the application, I could design, and fix errors appropriately based on received results. The main methods used throughout the application is the Auto-Change feature, the AES-GCM encryption algorithm and the SHA-256 hashing. Due to the repeated use of these methods, it is vital that they work reliably and consistently. The following shows the unit testing carried out to ensure that each method worked as intended before integration and implementation.

Auto-Change Method Test:

```
//Random String Generator that loops at a specified time

public static void main(String[] args) throws Exception {

        try {
            while (true) {
                int leftLimit = 48; // numeral '0'
                int rightLimit = 122; // letter 'z'
                int targetStringLength = 20;
                Random random = new Random();

                String generatedString = random.ints(leftLimit,  randomNumberBound: rightLimit + 1)
                        //.filter(i -> (i <= 1 || i >= 1) && (i <= 1 || i >= 1)) Option to filter generated string
                        .limit(targetStringLength)
                        .collect(StringBuilder::new, StringBuilder::appendCodePoint, StringBuilder::append)
                        .toString();

                System.out.println(generatedString);
                Thread.sleep( millis: 3 * 10000);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

This snippet shows the unit responsible for generating a random String with a length of 20 characters. The generator creates a random string every 30 seconds.



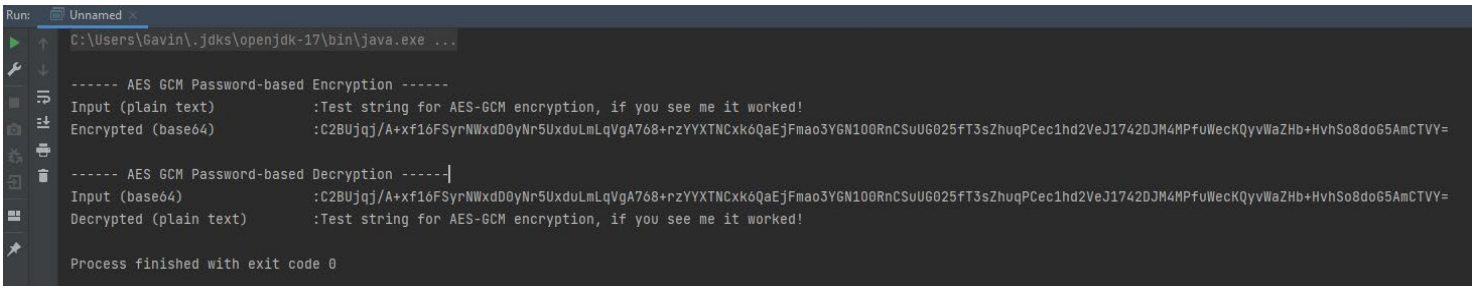
This screenshot shows the result of this unit test. The test was successful, and a random 20-character string was generated every 30 seconds. The purpose of this piece of code is to apply the generated string to create a new password with the Auto-Change feature.

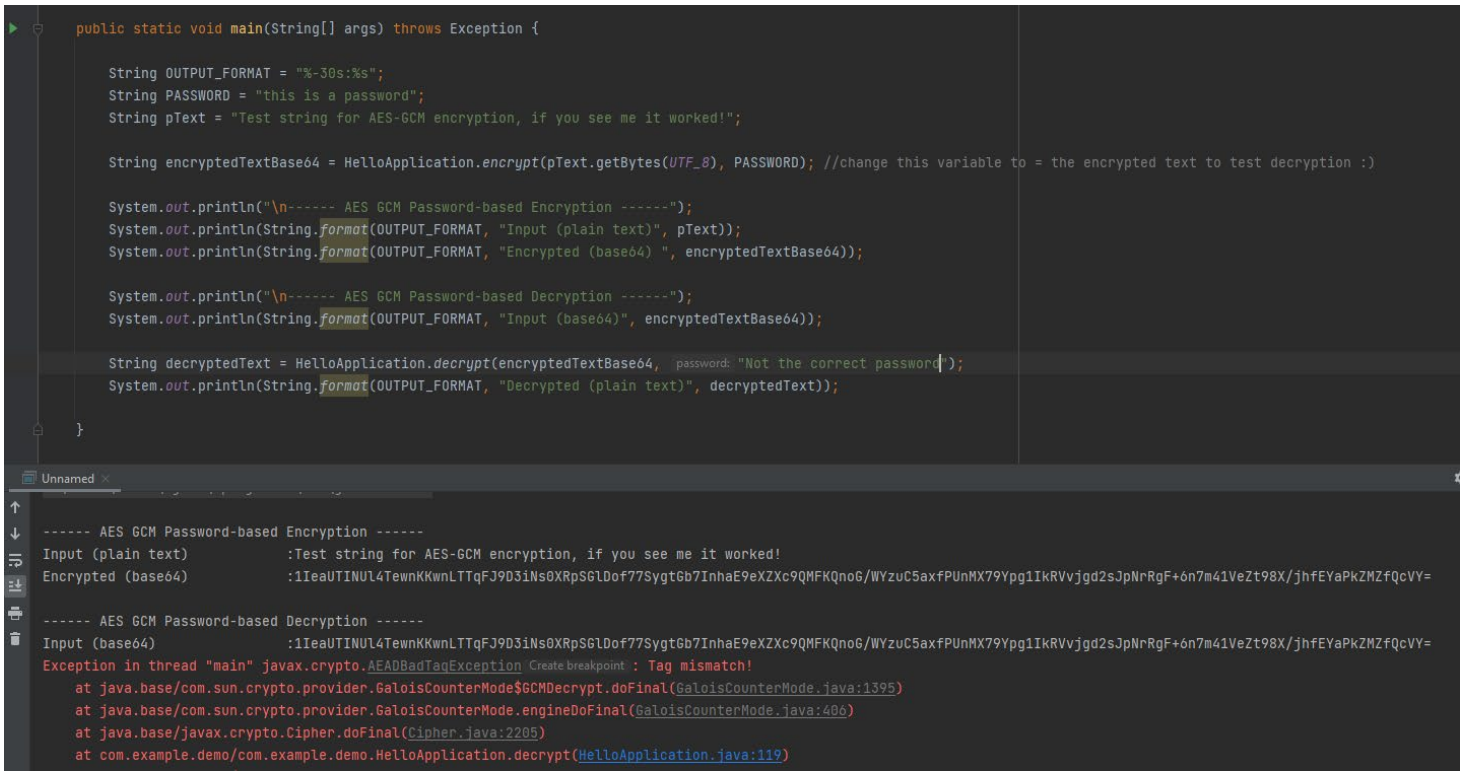
AES-GCM Algorithm Test:

```
public static void main(String[] args) throws Exception {

    String OUTPUT_FORMAT = "%-30s:%s";
    String PASSWORD = "this is a password";
    String pText = "Test string for AES-GCM encryption, if you see me it worked!";

    String encryptedTextBase64 = HelloApplication.encrypt(pText.getBytes(UTF_8), PASSWORD); //change

    System.out.println("\n------ AES GCM Password-based Encryption ------");
    System.out.println(String.format(OUTPUT_FORMAT, "Input (plain text)", pText));
    System.out.println(String.format(OUTPUT_FORMAT, "Encrypted (base64) ", encryptedTextBase64));

    System.out.println("\n------ AES GCM Password-based Decryption ------");
    System.out.println(String.format(OUTPUT_FORMAT, "Input (base64)", encryptedTextBase64));

    String decryptedText = HelloApplication.decrypt(encryptedTextBase64, PASSWORD);
    System.out.println(String.format(OUTPUT_FORMAT, "Decrypted (plain text)", decryptedText));

}
```

This screenshot shows the main class of the encryption test application. This is a standalone test application used for encryption and decrypting plain text using the AES-GCM algorithm. The plain text gets encrypted using the ‘PASSWORD’ variable.

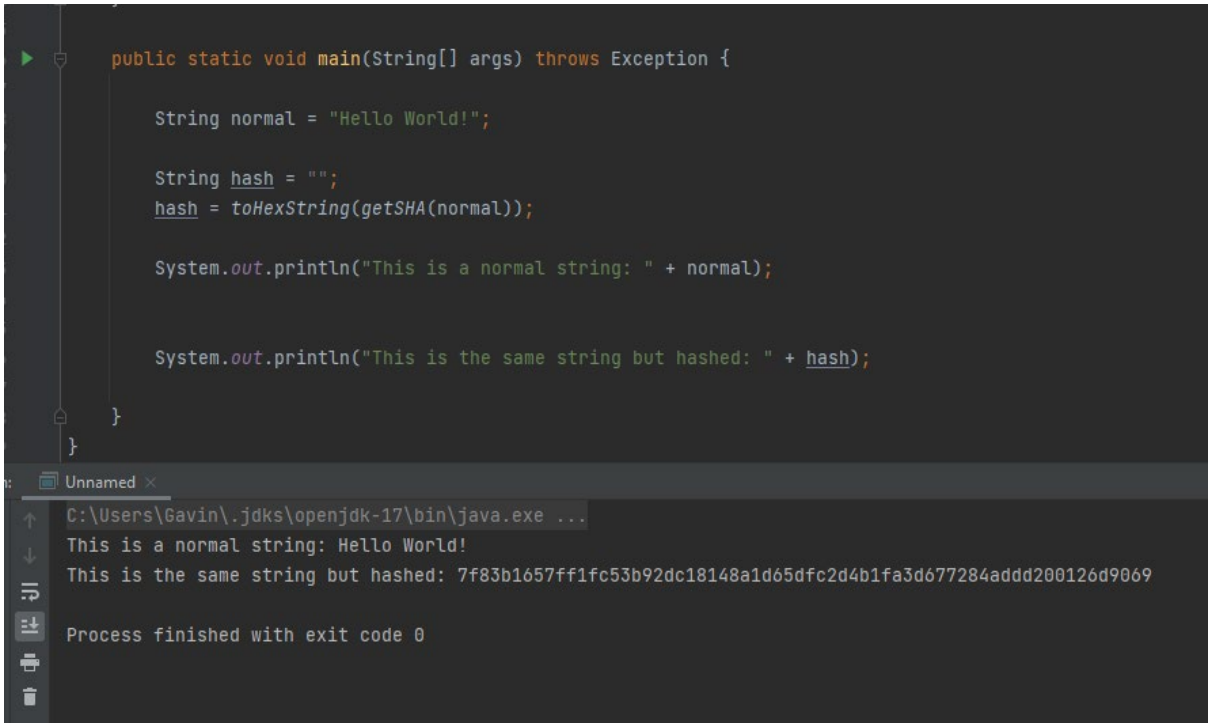
This is the result of the test. The string is successfully encrypted and decrypted using the algorithm:





The screenshot above shows what happens when the password does not match the original secret key. The text was successfully encrypted; however, the decryption returned an error due to the password not matching. This unit test verifies that the AES-GCM encryption and decryption is working as intended.

SHA-256 Hashing Test:



In this unit test, a normal string is printed, and the hashed version of that same string is also printed using the SHA-256 hashing. This screenshot shows the main method running and calling the ‘toHexString’ method and the ‘getSha’ method to hex the string. The result can be seen at the bottom of the image.

Integration Testing:

After unit testing on the vital elements of the application, it was time to integrate these methods into the basic application framework of the login page, registration page and password manager/Auto-Change.

```
try {
    Thread thread = new Thread(() -> {

        while (Objects.equals(AutoChange.instance.getAuto(),  b: "on")) { //Only active when auto-change is set to 'on' from Model.AutoChange
            String PASSWORD = UserSession.instance.getPasswordHash(); //calls user password by RAM memory for encryption
            try {

                int leftLimit = 48; // numeral '0'
                int rightLimit = 122; // letter 'z'
                int targetStringLength = 20; // length of generated string
                Random random = new Random();

                //Generates random string based on above parameters
                String generatedString = random.ints(leftLimit,  randomNumberBound: rightLimit + 1)
                        //.filter(i -> (i <= 1 || i >= 1) && (i <= 1 || i >= 1)) Option to filter generated string
                        .limit(targetStringLength)
                        .collect(StringBuilder::new, StringBuilder::appendCodePoint, StringBuilder::append)
                        .toString();


                //Encrypts generated string and sets it to a variable using the users account password as the encryption key
                String password = Cypher.encrypt(generatedString.getBytes(StandardCharsets.UTF_8), PASSWORD);

                //Hashes password for linked applications
                String passHashed = Hash.toHexString(Hash.getSHA(generatedString));

                //Updates users password in the database with the encrypted random string
                String query1 = "UPDATE  " + UserSession.instance.getUserName() + " SET password = " + "'" + password + "'" + ", passHashed = " + "'" + "
```

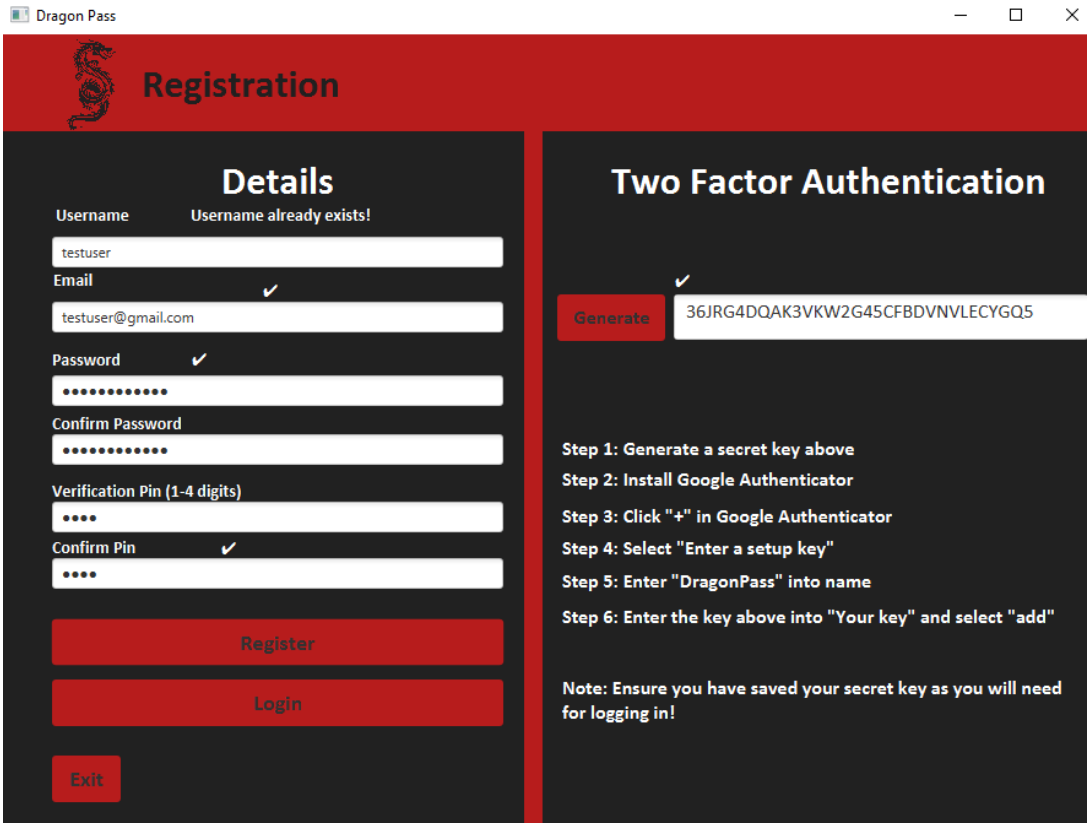
This snippet shows both the integration of the 30 second string generator and the AES-GCM encryption algorithm. It also uses the SHA-256 hashing algorithm for external application password changing. A string is generated like before and is then encrypted using the AES-GCM algorithm. A newly encrypted password is then updated into the database every 30 seconds. This is the best example of the integration testing that was carried out as it shows all three vital elements of the project working in one thread.

password_ID	app	username	password	passHashed	AutoChange
1	testapp1	gav	jzP4qBMdJL2Fkv5nG17twn+q7yC9A21ZgYTgPQ...	1c4bf19b58e4208e4b307e19db716ad846762f0...	Inactive
3	Google.com	BestUserEver100	UKAB/FDVsto5wyRH/ZgW0Apalt9K89lrt/zh/yHa...	a0cf3ad79e6a45604be9f0348b0412d573a1f26...	30 Seconds
4	Facebook	John Doe	jdfMPniIB0MAjUX8Kz1Mooc8kLEVMkEhvPPmjStF...	5994471abb01112afcc18159f6cc74b4f511b998...	Inactive
5	Twitter	tweetuser	jtQnCsd8Wdm+LD/h8IMv9fBFrypODNJRhzDDO...	45302038ee557753d2cc44d5e63483c187ecfbe...	Inactive
6	Steam	iLoveGames99	BVey63aOf/+6thCctZJJy+inoeKvr5KZSsRNWPG...	f3bcbfa6f3a2033f137eccb903e017fd064fd899...	Inactive
NULL	NULL	NULL	NULL	NULL	NULL

This database query shows all the accounts stored inside a test users account. The password inputted into the password manager encrypts and hashes the password. This result shows the integration of AES-GCM encryption, and SHA-256 hashing are working as intended. The Auto-Change feature can also be seen working as the ‘Google.com’ account is set to 30 seconds for auto-change.

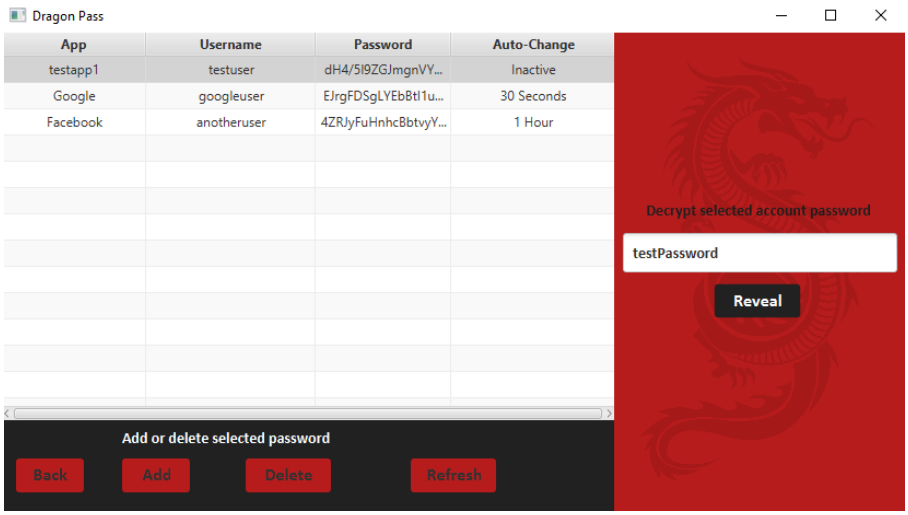
End User Testing:

For end user testing I performed every possible task within the application to ensure all elements were working as intended.

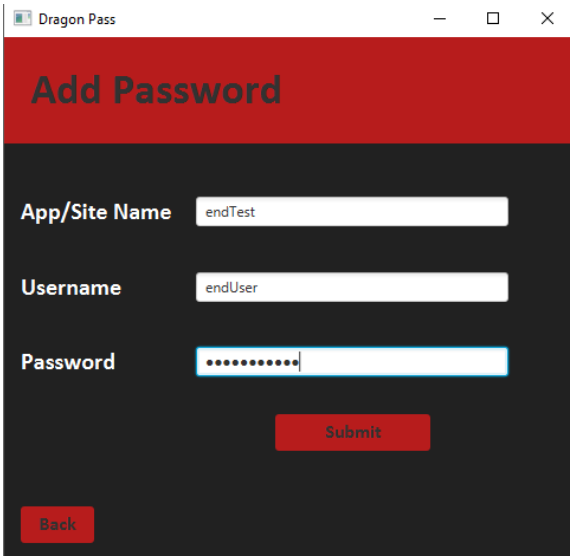


In the above snipped I created an account successfully as indicated by the tick symbols. For the purpose of demonstration, the screenshot shows an account that was already created, this is the account that would be used for the end user test.

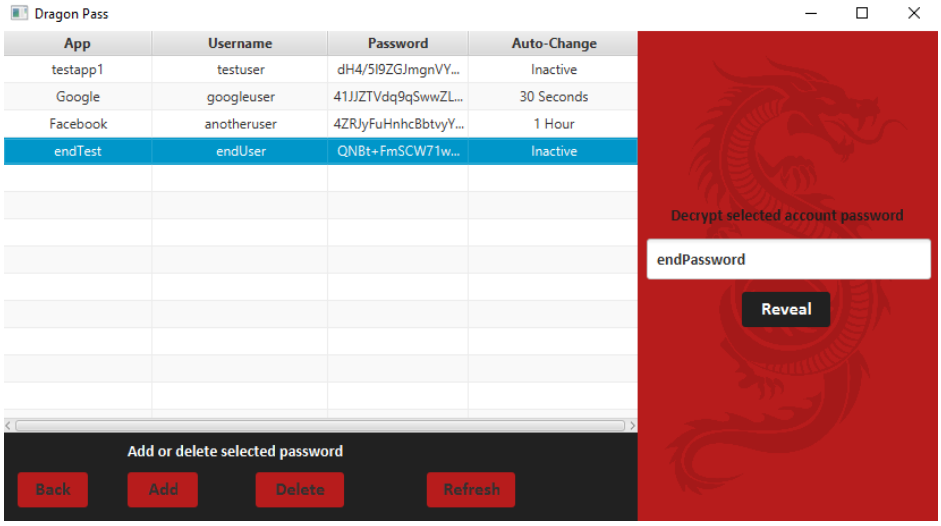
After successful login using these details, I accessed the password manager.



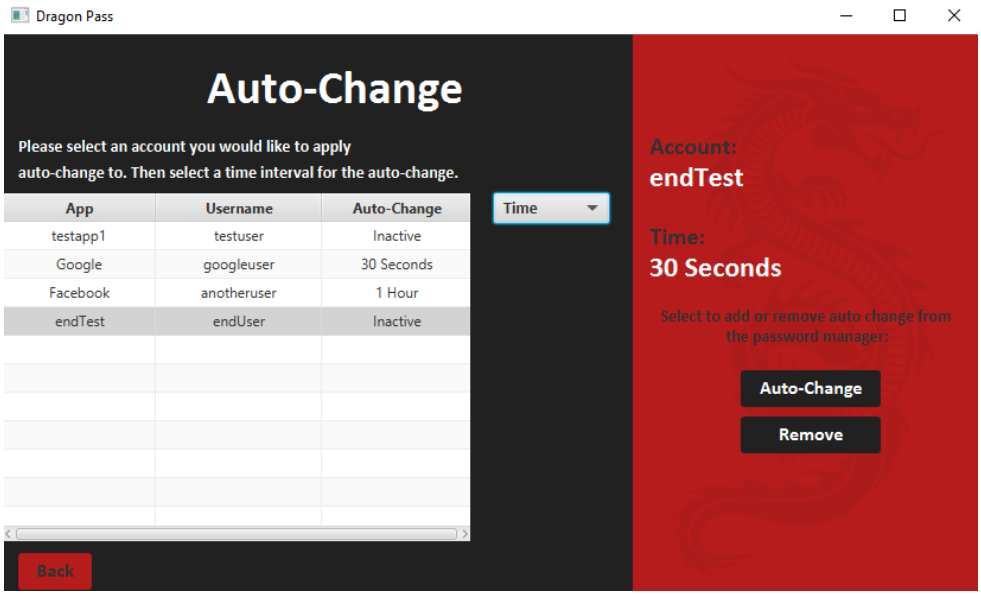
This snippet shows the encryption and decryption working successfully.



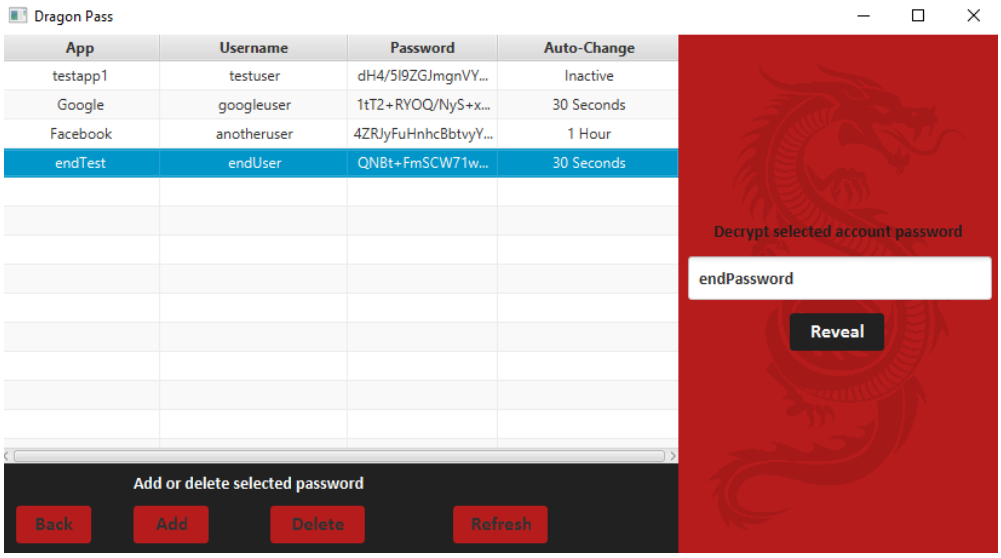
I added another password to the list for the auto-change test.



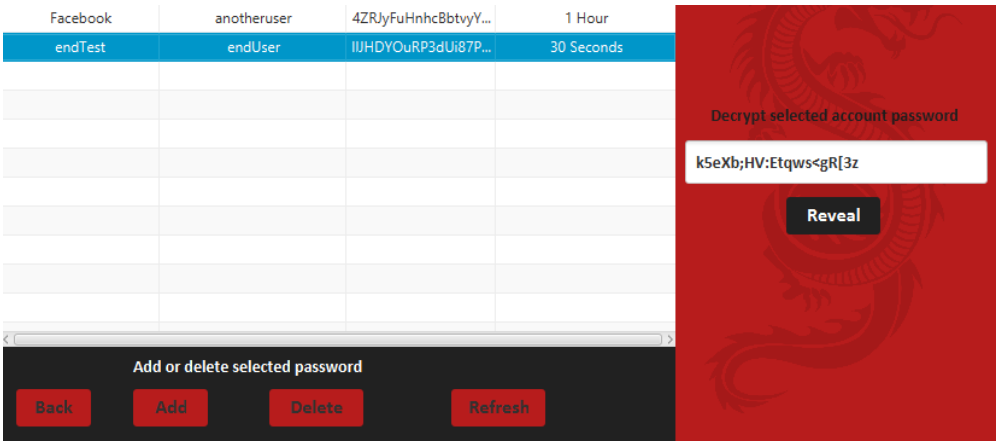
This was added successfully as seen in this screenshot.



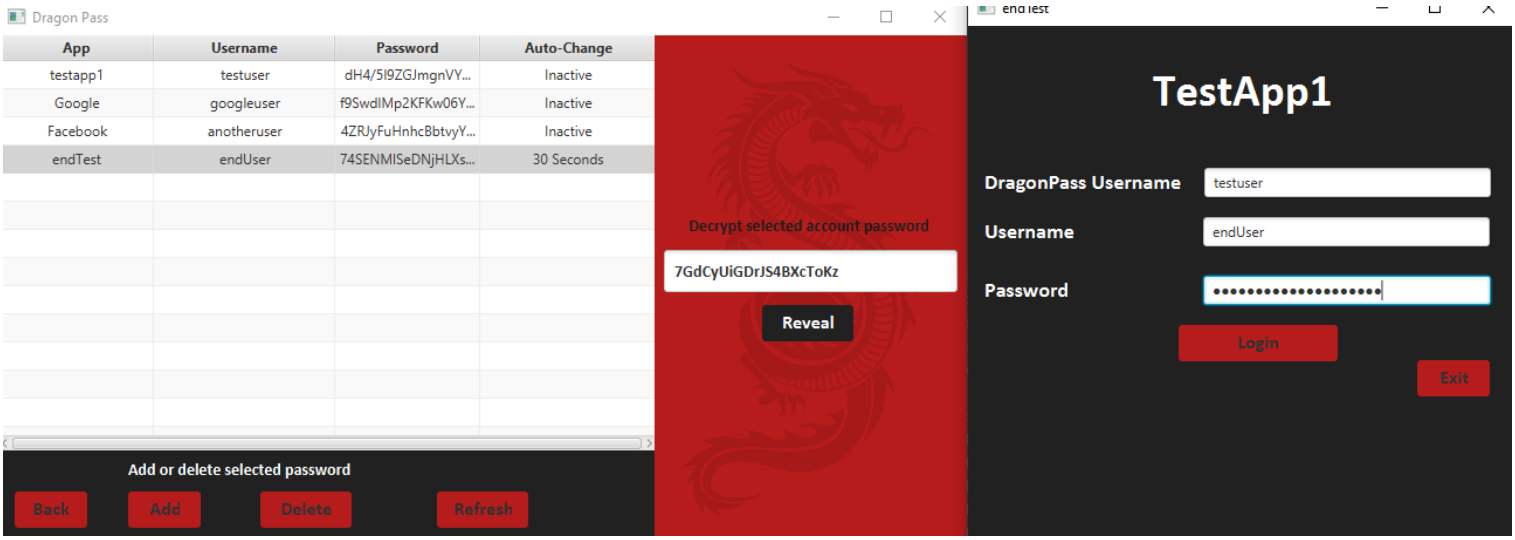
This screenshot shows me applying the 30 seconds auto change to the new password entry.



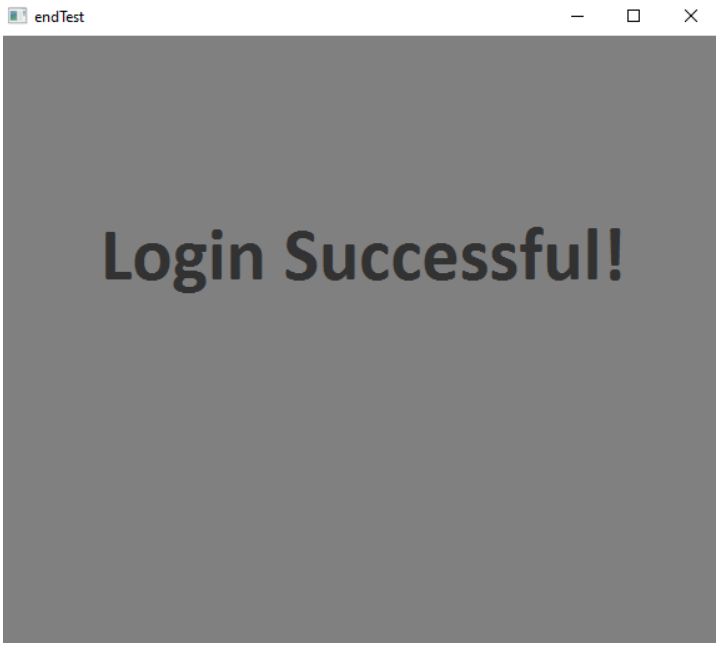
This snippet shows the account immediately after applying auto-change to 30 seconds.



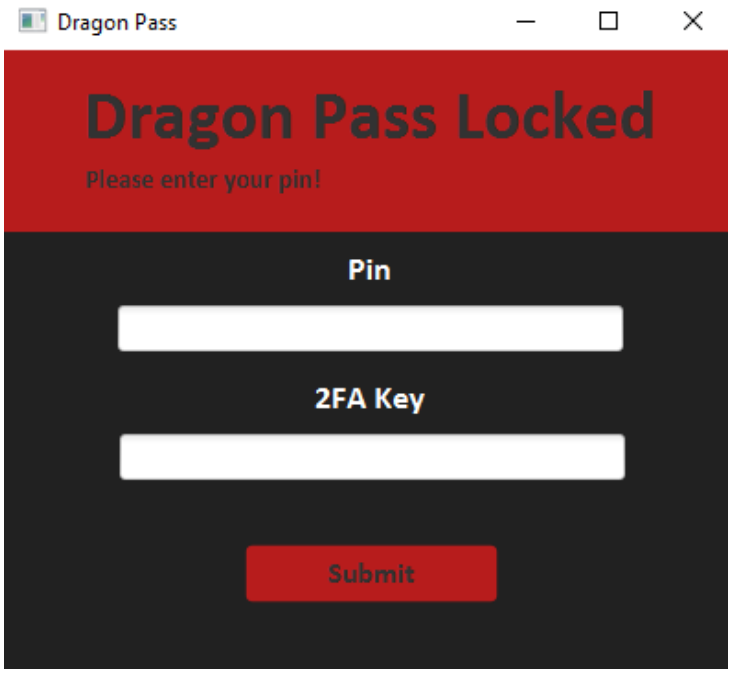
This snippet shows 30 seconds later. The password has changed to a complex string.



In the above screenshot, the test app and the password manager are side by side. When the password changed again after another 30 seconds I input the new password into the test app login.



This was the result of inputting the new password. The verifies that all the main elements of the application are working correctly.



The application lock also functioned as intended.

This end user test was repeated a total of 10 times with 3 different test apps and was successful every time.

LoginController.java | log.txt | RegisterController.java | Logger.java | FileObserver.java

```
info : New submission added : 04:43:54.691444800
info : New submission added : 05:11:12.006833200
info : Verification status requested : 05:11:30.451970900
info : Submission verified : 05:11:30.533576800
info : Verification status requested : 05:11:44.357289900
info : Invalid form details : 05:11:44.407162900
info : Exit application : 05:11:48.573255600
info : Status controller accessed : 05:12:07.316745500
info : Submission controller accessed : 05:12:12.880099900
info : Submission details invalid : 17:27:56.597611300
info : Status controller accessed : 17:28:01.479946800
info : Submission controller accessed : 17:28:06.308481
info : New submission added : 21:08:33.282885800
info : New submission added : 21:33:09.838993800
info : Submission details invalid : 21:58:34.870823300
info : Submission details invalid : 21:58:41.345682800
info : Submission details invalid : 21:58:57.439813
info : Submission details invalid : 21:59:00.333733500
info : Submission details invalid : 21:59:08.691076200
info : New submission added : 21:59:21.205131
info : Verification status requested : 21:59:31.043311800
info : Submission verified : 21:59:31.126092300
info : Verification status requested : 21:59:36.493790
info : Invalid form details : 21:59:36.531654500
info : Submission controller accessed : 21:59:51.892340200
info : Submission details invalid : 21:59:55.744624100
info : Submission details invalid : 21:59:59.997273600
info : Submission details invalid : 22:00:04.415410300
info : Submission details invalid : 22:04:55.576777400
info : Submission details invalid : 22:05:17.033065900
info : New submission added : 22:05:31.533757600
info : Verification status requested : 22:05:39.686402900
info : Invalid form details : 22:05:39.754657400
info : Verification status requested : 22:05:44.187381500
info : Invalid form details : 22:05:44.226282300
info : Verification status requested : 22:05:50.272638700
info : Submission verified : 22:05:50.311533200
info : Exit application : 22:05:56.348545
```

This snippet shows a working example of the logger being used, tracking user activity, and reporting any possible errors.

2.6. Evaluation

Name	Status	6% CPU	45% Memory	0% Disk	0% Network	28% GPU	GPU engine	Power usage	Power usage t...
OpenJDK Platform binary (2)		1,2%	489,9 MB	0,1 MB/s	0 Mbps	0%		Low	Very low
Console Window Host		0%	5,7 MB	0 MB/s	0 Mbps	0%		Very low	Very low
OpenJDK Platform binary		1,2%	484,2 MB	0,1 MB/s	0 Mbps	0%		Low	Very low

Upon observing the task manager, the CPU, memory usage and power usage for Dragon Pass is very low.

After analysing the results of the unit, integration and user tests the overall completeness and correctness of the application was as expected and worked as intended. Any errors found through the development process were fixed upon discovery. This meant that there were very little errors upon user testing and for the final user test rounds as reported, there were no errors found.

Based on the test results and performance recorded, Dragon Pass works as initially intended and provides a secure platform to secure and change passwords.

3.0 Conclusions

The main advantages of Dragon Pass include an increased level of password security, high level data integrity and confidentiality, user convenience and overall cyber protection from various malicious attacks. A disadvantage of the application is the lack of increased user convenience such as an autofill feature that other password managers use such as Google's web password manager. However, a further developed Dragon Pass application in a commercial environment could potentially use this feature. The main strength of this project is to offer a solution to data breaches, password leaks, phishing scams, brute force attacks, SQL injections and buffer overflow by providing a high level of security method implementations. The application uses the auto-change feature which gives a whole new level of security to the user which no other password manager or security tool currently does on a commercial level. While providing a strong security solution, however, the project is limited to the fact that it is not developed at a commercial level and currently does not provide a practical usage in real world examples. In its current state, it acts more as a model or prototype for what it could become. The application has good future potential, however, would require further development to make it commercially and practically viable. Regardless of this, however, Dragon Pass lays the foundation for a new possible security implementation for password managers to utilize and to further protect their user's sensitive data.

4.0 Further Development or Research

I believe with further development and additional time and resources; this application could become a commercially used application to help home users and companies securely store user password data. To become a fully developed and commercial application there are two possible routes the project could take. The first possible route would be the cooperation of various companies to implement the idea within their own systems and to create a password manager that works for all and any application/web app in the commercial sphere, such as google and Facebook. The reason for this is that many companies and applications have different password changing requirements and rules, therefore, cooperation and implementation into already exiting applications would allow the Auto-Change feature of Dragon Pass to work as intended. The second possible route would be to develop the application in a way that it could navigate through all application password requirements and rules. This may include artificial intelligence technology or various API usage. If this were to be implemented, the application could be sold as a standalone product and compete with other various password manager applications. Unfortunately, this would require much more time, research, and development to implement, however, the foundation for this possible future has already been set by this project.

5.0 References

[1] R. Wu, X. Zhang, M. Wang and L. Wang, "A High-Performance Parallel Hardware Architecture of SHA-256 Hash in ASIC," 2020 22nd International Conference on Advanced Communication Technology (ICACT), 2020, pp. 1242-1247, doi: 10.23919/ICACT48636.2020.9061457.

[2] N. Ahmad, L. Wei and M. Hairol Jabbar, "Advanced Encryption Standard with Galois Counter Mode using Field Programmable Gate Array.", Journal of Physics: Conference Series, vol. 1019, p. 012008, 2018. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1019/1/012008/pdf.

[3] D. M'Raihi, S. Machani, M. Pei and J. Rydell, "RFC 6238 - TOTP: Time-Based One-Time Password Algorithm", Datatracker.ietf.org, 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6238.

6.0 Appendices

6.1. Project Proposal

1.0 Objectives

For my 4th year cyber security software project, I have decided to create a password manager. This password manager will automatically manage and change the users' passwords on a 24-hour basis. The passwords will be randomly generated 16-character passwords with all the usual parameter's passwords are required to have. The user will only need one password and a pin to access their password list. The manager will also automatically input the users' desired passwords into whatever app or website they are trying to access.

With the pin and password, the password manager app will also have two factor authentication to further secure the user's account. The project aims to make password management more secure and convenient for users. The user will be able to choose whether they would like certain passwords to be changed on a 24-hour basis.

The essential goal of this project is to allow users to access their various apps and websites by using the app. Meaning the user will only ever need to remember one password and one pin, rather than hundreds of different passwords for hundreds of different websites. The other feature of the app that automatically changes their passwords will stop malicious users from easily gaining access to the user's specific accounts. For example, if a user's password is leaked on a website, the manager will change that password within 24 hours, meaning malicious users have a limited amount of time to access it. I may implement a feature to choose how many hours it takes for a password to change.

2.0 Background

The reason I decided to undertake this project is due to the work experience I did in Beaumont Hospital during my 3rd Year of this course. I worked in Beaumont during the HSE attack in 2021. One of the main ways Beaumont defended against the attack was by changing the passwords of every user in the hospital every 24 hours. This stopped the attack from spreading as once it gained access to one password, it only had 24 hours to use the account. This is one of the main reasons Beaumont, unlike other hospitals linked to the HSE, was able to remain untouched by the virus. We changed users' passwords manually on a 24-hour basis and this required a lot of work and time. This is what made me come up with the idea of a password manager that would do the work automatically. In a digital world where passwords are being gained through various methods such as brute force attacks or company leaks, a manager that would automatically generate and change passwords would help defend against these. It is also convenient for users as it is very difficult to remember hundreds of different and secure passwords.

I will use various coding languages and techniques to complete this project. I will also use the cyber security techniques I have learned from my various modules this year to ensure the app is secure. I will also use the different planning techniques I learned throughout my studies in this computing course for the last 3 years.

3.0 State of the Art

There are similar applications that already exist to my project such as Dashlane, NordPass and Chrome password manager. The difference between this project and them is that my application will be able to automatically change and store unique randomized passwords rather than keeping one password the same forever. The named password managers alert the user about data breaches and how to protect their data against them i.e., changing password, however, my app will do this automatically all the time, meaning data breaches will have no effect on users that use this app. It will also include two factor authentication to ensure that malicious users can't access the users account if they somehow gain access to the users pin and password. The examples mentioned above only require one password meaning if someone gains them, they will have access to all the users' passwords.

Just like the mentioned examples, my projects aim is to create an app that allows users to securely access their various accounts carefree of any security issues, as the app will take care of that for them. Data breaches and various hacks to gain peoples passwords aren't being protected by other password managers as well as they could be. For example, users are only notified of data breaches on chrome password manager if the user goes into their password list, they then must manually follow the password managers instructions. For more security and convenience, automatic password changing defends more effectively against this. If a malicious user also gains access to a specific password through various methods such as brute force or malicious software, they will only have limited time to use it, other password managers don't defend against this like my project.

4.0 Technical Approach

For my project I have chosen to use the agile approach for its development. This will allow me to plan my project in incremental phases that can adapt and change depending on different variables. I will outline various requirements by researching the topic related to my project and by using various methods such as use case modelling, mind mapping and scope management.

I will also create a list of goals that need to be reached and will mark off each goal when their conditions have been met.

When the requirements have been identified I will outline tasks which will each have a deadline. Specific features of the application will all have their own requirements and respective deadlines. To list these tasks and organize the project requirements I will use Microsoft software such as Access and Project. Using this software will allow me to efficiently keep track of tasks and what times should be allocated to performing these tasks. The agile approach in software development allows for adaptive strategies meaning it will give more freedom to change various tasks/deadlines to create a better product for the user. It also includes fast and productive incremental steps which can easily be kept track of by using the software mentioned before.

5.0 Technical Details

I intend to use java as the primary language for this project. Due to the nature of the project, it will need to be a web-based application. To do this I will need to use java servlet technology and JSP. This will then allow me to create a java-based web app. This means I will also be using HTML and CSS. For the server environment, I will be using Apache Tomcat and for the database I will be using MySQL.

Due to the system dealing with passwords and important information, I plan to use the AES (Advanced Encryption Standard) algorithm to securely encrypt and decrypt the data.

I may develop the two-factor authentication feature through email input or by using apis, other external resources, etc. This is still something I am researching to create the best result.

I intend to use IntelliJ IDEA and Atom for development of the project.

6.0 Special Resources Required

As mentioned, before I may need to use APIs and other external resources. Two resources I am currently looking at for two factor authentication is Twilio API for two factor authentication or Sessions mobile app API.

7.0 Project Plan

This plan may be subject to change and is following an agile approach of project management. I plan on creating a list of requirements and tasks. These will then be placed and listed in a Microsoft Access or Microsoft Project document. The main tasks that need to be completed are currently as follows:

- Create a secure login system for application (12th Nov)
- Create two factor authentication for login system (No set date)
- Create a Home page for application site (20th Nov)
- Create random password generation (28th Nov)
- Create system for password storing (8th Dec)
- Create a list of stored passwords (using hashing and other encryption methods) (19th December)

The listed goals above are the main tasks that will need to be completed for my project with their predicted and set dates for completion. Each goal will have a list of sub goals to make development go more smoothly.

Mid-Point implementation is due on December 21st , this means I should have a functional login system, home page, password storing and generation with encryption and decryption methods. I also hope to have two factor authentication implemented by this point as well.

The following list will include goals and tasks that I hope to complete after my mid-point implementation:

- Have app auto fill details into another website
- Have password change for test website automatically every 24 hours
- Two factor authentication functional (if not already)
- Create auto create and password storage from test website
- Polish/clean up website design (CSS)

8.0 Testing

To test my project, I will use the four main stages of testing in software: Unit testing, Integration testing, System testing and Acceptance testing. For unit testing, I will go through units of source code one by one to ensure all units are working based on the goals of the project. I will use various techniques such as Black box and White box testing. There will be a list of functional requirements and goals, such as the mentioned in my project plan. With the use of integration tests such as big bang testing, incremental testing, top-down, and bottom-up testing, etc; I can test the compliance of my systems with the already listed requirements. For system testing I can also use the Black box testing method.

Acceptance testing will be based on:

- Performance
- Data Integrity
- Usability
- Functionality

If the above tests are successful, I will then get an end user to test the project. To do this I will need to complete an ethics form. Once I find a user suitable to test the project, I will have them test all the different features and will base the success of the test on how many functional requirements and goals have been met. These tests will ensure my project works as intended.

6.2. Reflective Journals

Month 1:

This month I began researching ideas for my project. I have decided to create a password manager with a unique difference to existing password managers. My application will change passwords on a daily or hourly basis, constantly changing them to random complex Strings periodically.

Month 2:

This month I began development on my project. I decided to call it Dragon Pass and have used red and black as its colour design. I have created a basic UI and plan to create a login, registration system by the end of next month.

Month 3:

This month I have completed the applications login/registration system, created a navigation menu, a mock website for testing and have also finished the prototypes for my password manager UI.

Month 4:

This month I decided to create a mock application in place of the previous mock website and have implemented it to work with the application. I have also finished the password manager with addition, deleting and the refreshing of passwords for user accounts. I have also added password hashing to my login and registration system for user verification. This month has been my most productive so far.

Month 5:

This month I have implemented two-factor authentication for the user login verification using the TOTP algorithm and Google authenticator. I have also implemented AES-GCM encryption for the password manager to secure any stored passwords within the application database. I have also begun working on the Auto-Change feature.

Month 6:

This month I have finished the Auto-Change feature, allowing the user to select a stored password from within the manager and then apply a time interval to the password. The password will then change based on the time interval chosen while still using encryption for the newly changed passwords. A hash of that same password is then sent to the test login application for verification. I have also polished and fixed any bugs I could find based on testing and during development.

6.3. Invention Disclosure Form

1. Title of Invention

Dragon Pass

2. Inventors

Name	School/Research Institute	Affiliation with Institute (i.e. department, student, staff, visitor)	Address, contact phone no., e-mail	% Contribution to the Invention
Gavin Corr	National College of Ireland (NCI)	Student	Student	100%
			0838730761	
			Gavincorr18@gmail.com	

3. Contribution to the Invention

I am the sole inventor of this project. I intend to use the knowledge I have gained throughout the course of my studies at National College of Ireland to create and effective security application.

12/05/2022

4. Description of Invention

The novelty aspect of this project is the 'Auto-Change' feature. This feature will allow users to select any of their application/website accounts and on a set timed basis, change their desired account password automatically. For example, if a user sets their password to change every 24 hours, the application will generate a new complex password every 24 hours for the chosen account. This is to prevent damage from password leaks or data breaches. The dynamic nature of this feature is what makes the idea unique and is not found in other password managers.

5. Why is this invention more advantageous than present technology?

For more security and convenience, automatic password changing defends more effectively against password leaks, data breaches and other malicious attacks. If a system is making passwords more complex and changing them automatically on a constant basis, this will make it extremely difficult to gain access to material protected by a password. Also due to human constraints and limitations, a computer can store hundreds of complex passwords, whereas a human brain usually cannot. Furtherly, if those complex passwords are constantly being changed, it creates an extremely secure system.

6. What is the current stage of development / testing of the invention?

The application is fully functional and fully developed. As it is a college project and not currently a commercial product, the use of test applications has been used as means of demonstrating how the application would function in real life scenarios. Real life use would require business cooperation for the full implementation of the project in the commercial sphere.

7. List the names of companies which you think would be interested in using, developing or marketing this invention

Google, Microsoft, NordPass, NortonLifeLock and Dashlane.

8. Where was the research carried out?

National College of Ireland

9. What is the potential commercial application of this invention?

To be used as an extension or addition to already existing password manager technologies. Possibly its own standalone password manager application that benefits from the cooperation of other companies within a commercial landcape. The application would offer users an extra layer of security for their password management and overall password usage.

10. Was there transfer of any materials/information to or from other institutions regarding this invention?

No

11. Have any third parties any rights to this invention?

No

12. Are there any existing or planned disclosures regarding this invention?

Please give details.

The project will be showcased from May 30th to June 3rd

13. Has any patent application been made? Yes/No

No

14. Is a model or prototype available? Has the invention been demonstrated practically?

The full application model and prototype is available at: https://github.com/gavinc99/project

I/we acknowledge that I/we have read, understood, and agree with this form and the Institute's *Intellectual Property and Procedures* and that all the information provided in this disclosure is complete and correct.

I/we shall take all reasonable precautions to protect the integrity and confidentiality of the IP in question.

Inventor: Gavin Corr  12/05/2022

Signature Date

_____

Signature