CMP315

Secure Software Engineering

Project Presentation

Liam Townsley | Owen Walker | Ewan Taylor | Jack Barnett | Ryan van Ee

What's going to be covered?

- Project Overview and Clean Build by Ewan Taylor
- Front-End Overview by Owen Walker
- Back-End Overview by Liam Townsley
- System Architecture Overview by Ryan van Ee
- Testing Overview by Jack Barnett

Project Overview and Clean Build

Presented By: Ewan Taylor

Project Overview

- The password manager satisfies the aim of developing a "*password manager with appropriate security mitigations and defence-in-depth, to be resilient against a range of attack vectors.*"
- The project development fell into four main sprints
  - Development of an API backend
  - Development of a front-end C# application to interact with the backend
  - Applying security tests to
  - Testing, fixing and optimising the password manager
- Security testing was applied throughout to ensure the code was securely built, with testing become an increased priority from sprint three onwards when more avenues of testing was available.

Build Process

To install this project, the repository must be downloaded, alongside the requirements.txt file.

Git, Python and Visual Studio should also be installed.

To aid these process, instructions are provided in the README.md file to assist the user in setting up the API with the necessary .env and dependencies requires.

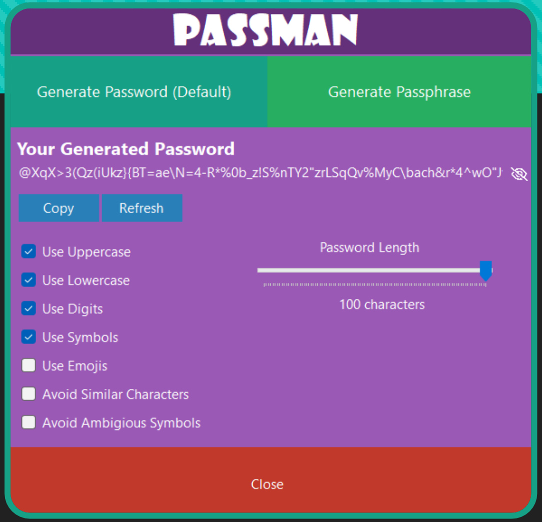
Password Manager Functionality Overview

Perform CRUD operations with Passwords and Secure Notes

Secure credentials, only requiring the user to remember one master password

Generate passwords of up to 100 characters with filters to accommodate according to a range of specifications

Generate Passphrases up to 100 words

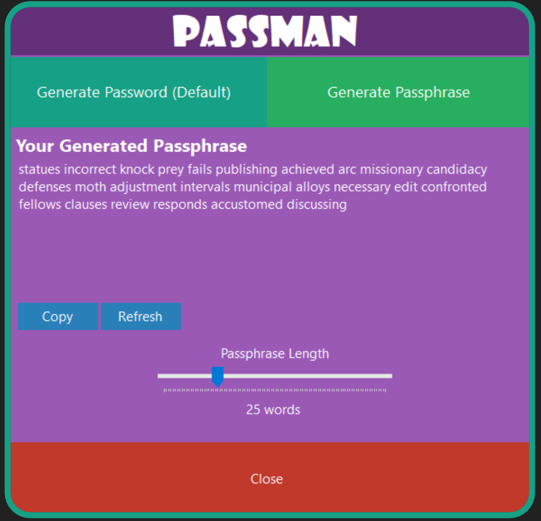


Your Generated Password

@XqX>3(Qz(iUkz}{BT=ae\N=4-R*%0b_z!S%nTY2"zrLSqQv%MyC\bach&r*4^wO"J

- [x] Use Uppercase
- [x] Use Lowercase
- [x] Use Digits
- [x] Use Symbols
- [ ] Use Emojis
- [ ] Avoid Similar Characters
- [ ] Avoid Ambigious Symbols

Password Length: 100 characters



Your Generated Passphrase

statues incorrect knock prey fails publishing achieved arc missionary candidacy defenses moth adjustment intervals municipal alloys necessary edit confronted fellows clauses review responds accustomed discussing

Passphrase Length: 25 words

Build Process Overview

Throughout the build process an agile methodology was used. This allowed for flexibility with project as we could prioritise the following:

- Building an initial stable version by the end of the sprint 2
  - This allowed for more security testing to be conducted, notably dynamic testing
- Patching vulnerabilities found through security testing
- Prioritising progress with new or existing features where they could be made

GitHub was used to house the project and for all team members to make changes, raise issues and pull a clean and stable version of the project for testing.

Front-End Overview

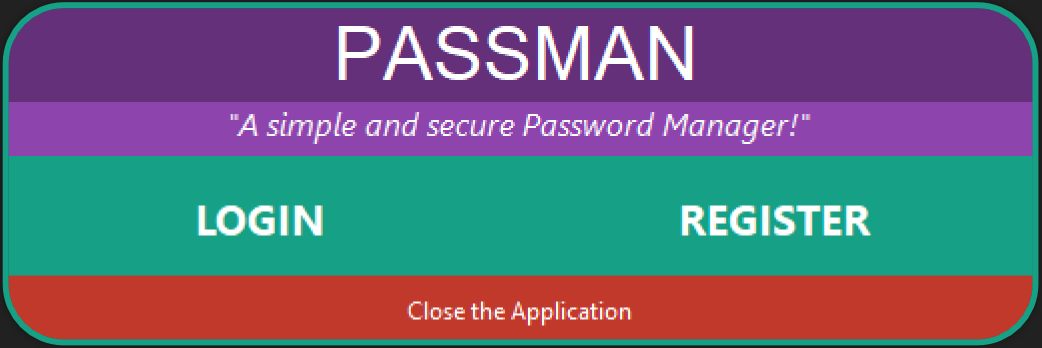
Presented By: Owen Walker

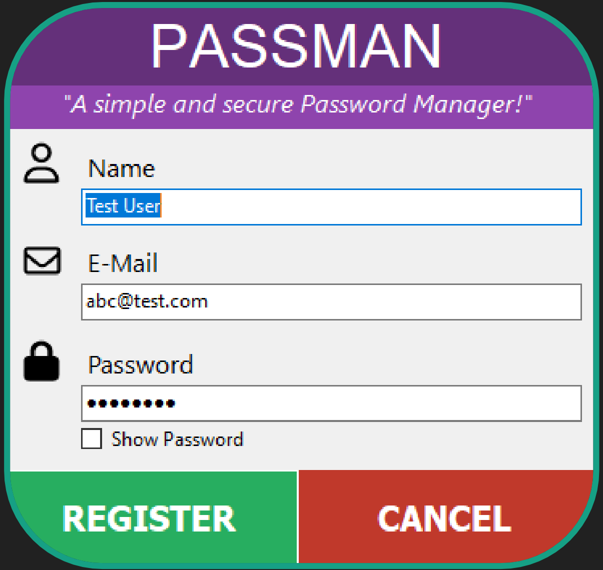
Login & Registration Page

When first opening the program, you are greeted by the login/registration page which takes input and parses it through the PASSMAN API to query the database.

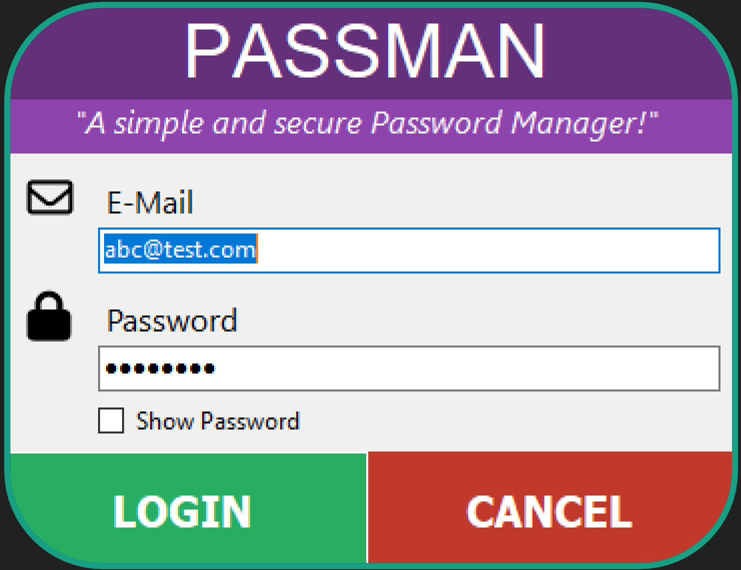
All user inputs are screened for SQL injection techniques using parameterized queries by using placeholders for the parameters rather than injecting them directly.

The Program also validates user input by making sure that the user can only input an email style string.





Name
Test User
E-Mail
abc@test.com
Password
••••••••
☐ Show Password

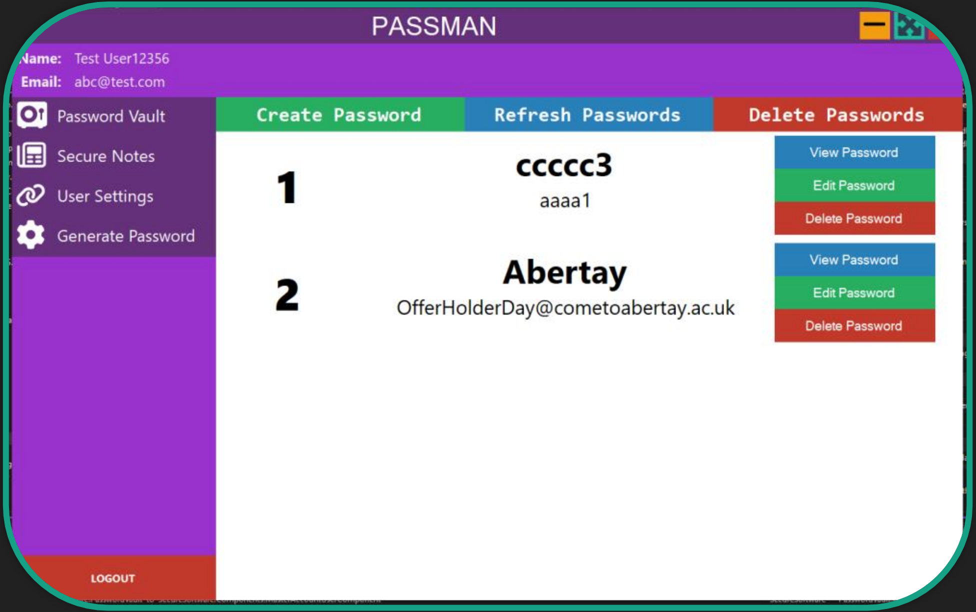


E-Mail
abc@test.com
Password
••••••••
☐ Show Password

Password Vault

After logging into program, you are greeted by several different options the main one being the password vault which shows the users username/email and when you click in further you can view the passwords for individual accounts edit and delete them as needed.

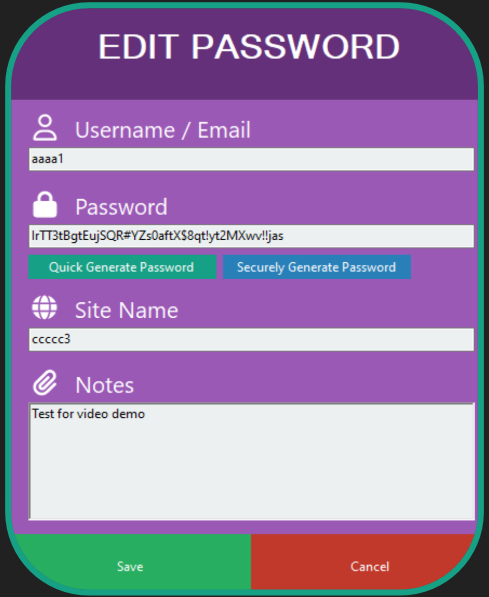
You can also generate a secure password in the edit password and the create password.



Name: Test User12356
Email: abc@test.com

1 ccccc3 aaaa1

2 Abertay OfferHolderDay@cometoabertay.ac.uk

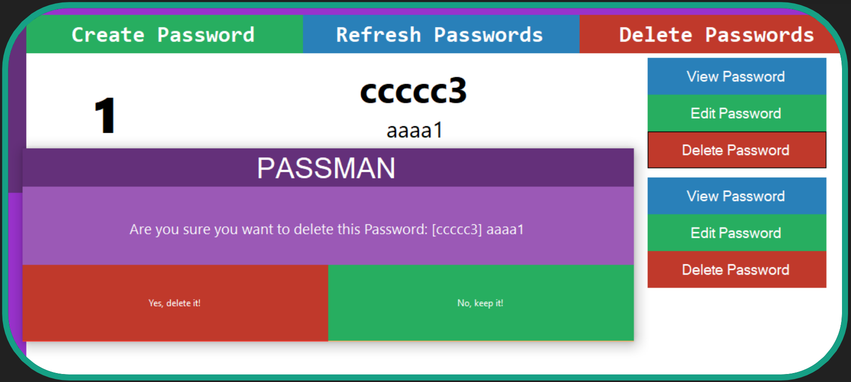


Username / Email: aaaa1

Password: IrTT3tBgtEujSQR#YZs0aftX$8qt!yt2MXwv!!jas

Site Name: ccccc3

Notes: Test for video demo



1 ccccc3 aaaa1

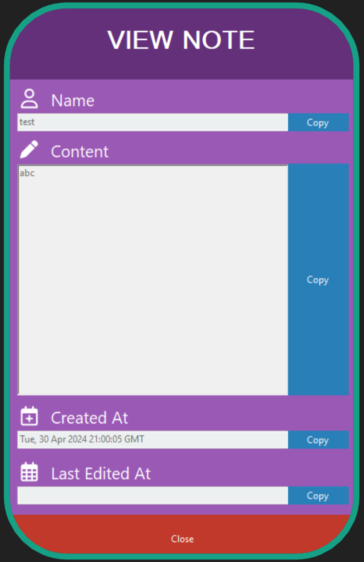
Are you sure you want to delete this Password: [ccccc3] aaaa1

Secure Notes

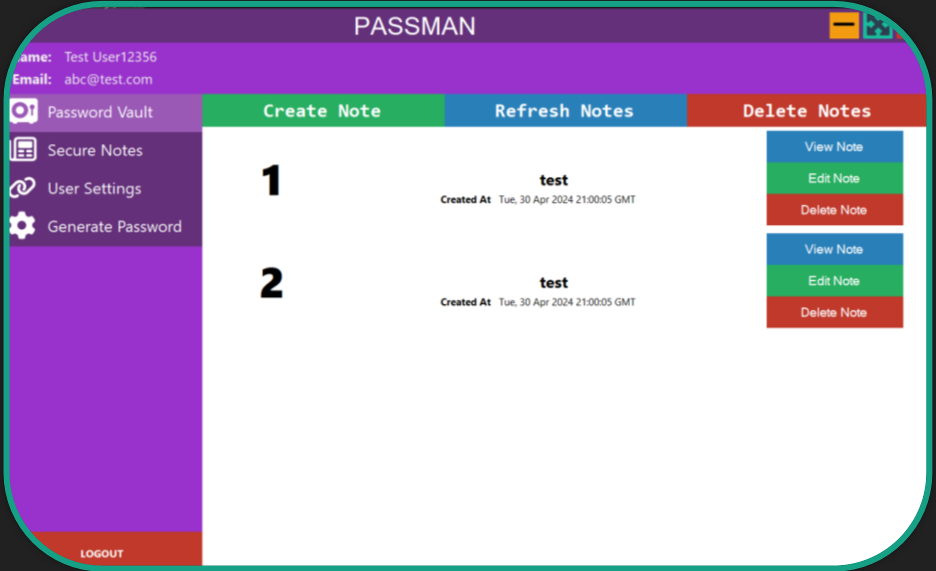
You can also generate secure notes within the application to keep your information secure

The page gives you an overview of the notes with the date created and the last edited timestamp.

You can also delete/edit the notes along with viewing them.



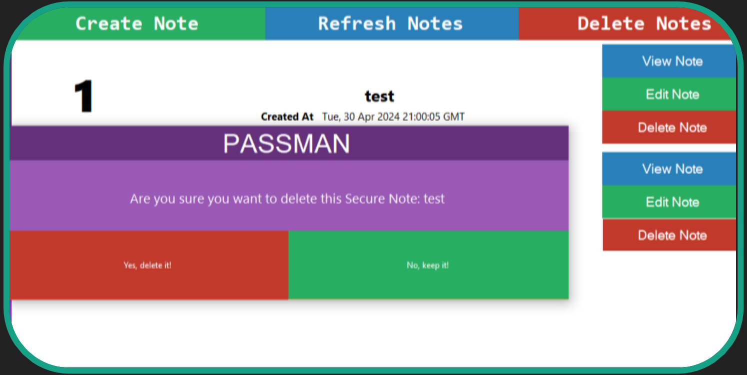
Name: test
Content: abc
Created At: Tue, 30 Apr 2024 21:00:05 GMT
Last Edited At:



ame: Test User12356
Email: abc@test.com

1 test
Created At Tue, 30 Apr 2024 21:00:05 GMT

2 test
Created At Tue, 30 Apr 2024 21:00:05 GMT



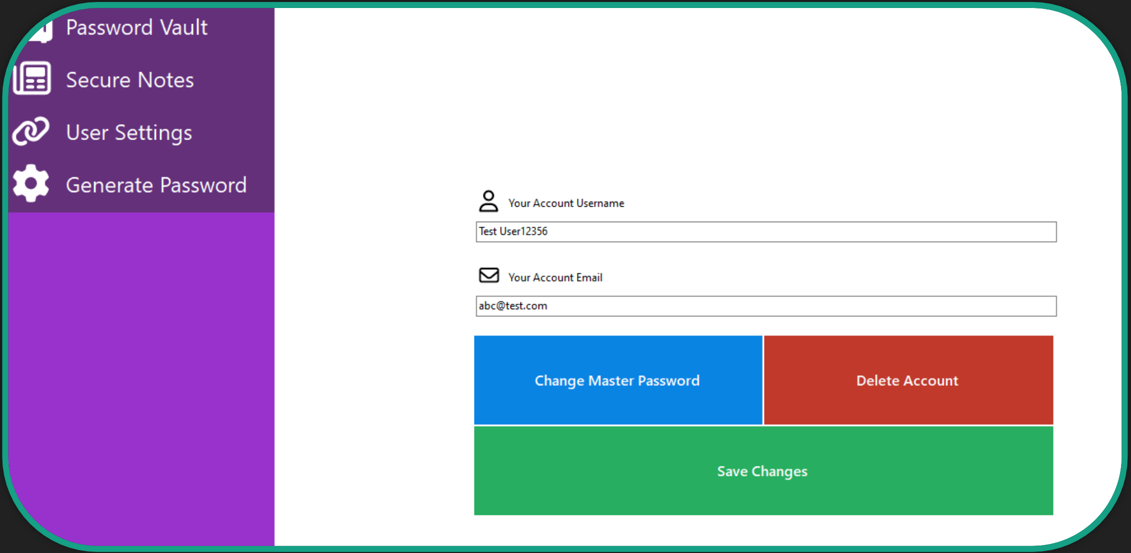
1 test
Created At Tue, 30 Apr 2024 21:00:05 GMT

Are you sure you want to delete this Secure Note: test

User Settings

This section is responsible for the editing of the master account and allows you to delete account and change the master password.

It also includes a warning prompt for the deletion your account so that you understand what you are losing with the deletion of your account.

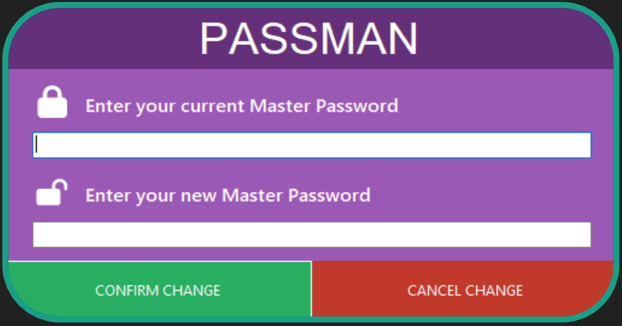


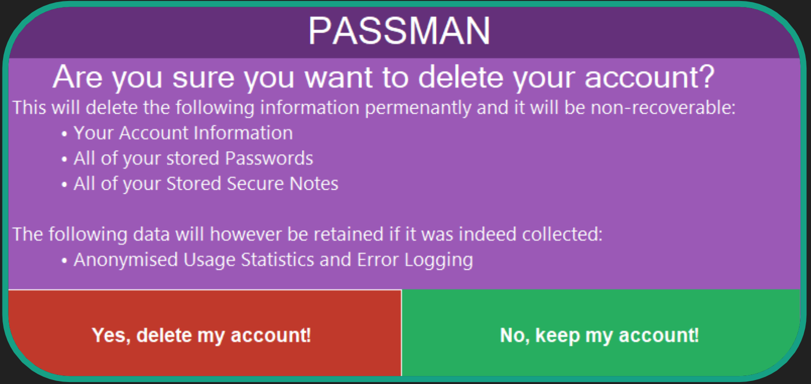
Your Account Username

Test User12356

Your Account Email

abc@test.com


PASSMAN
Enter your current Master Password
Enter your new Master Password
CONFIRM CHANGE
CANCEL CHANGE



Are you sure you want to delete your account?

This will delete the following information permenantly and it will be non-recoverable:

- Your Account Information
- All of your stored Passwords
- All of your Stored Secure Notes

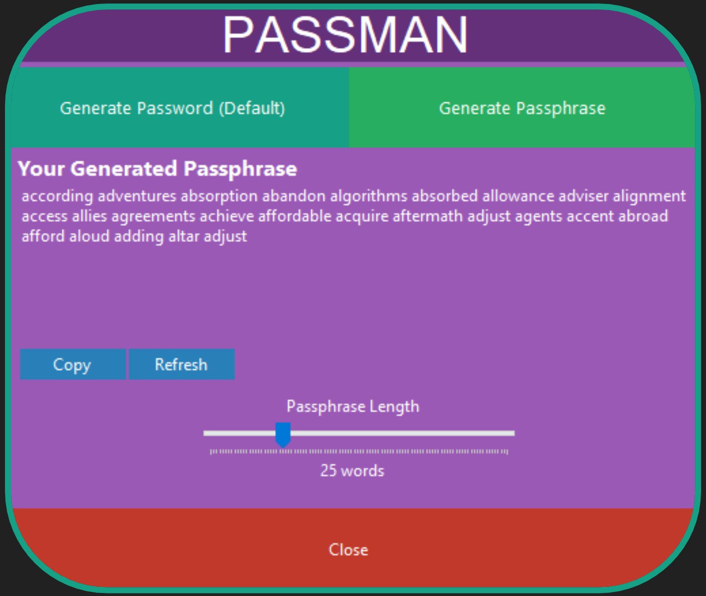
The following data will however be retained if it was indeed collected:

- Anonymised Usage Statistics and Error Logging

Password & Passphrase Generator

The application also comes with a password/passphrase generator which can be used to generate a secure password or passphrase with a few different options to suit a user's needs such as:

- Length
- Use of upper/lowercase characters
- Use of special characters
- Avoiding similar and Ambiguous characters.

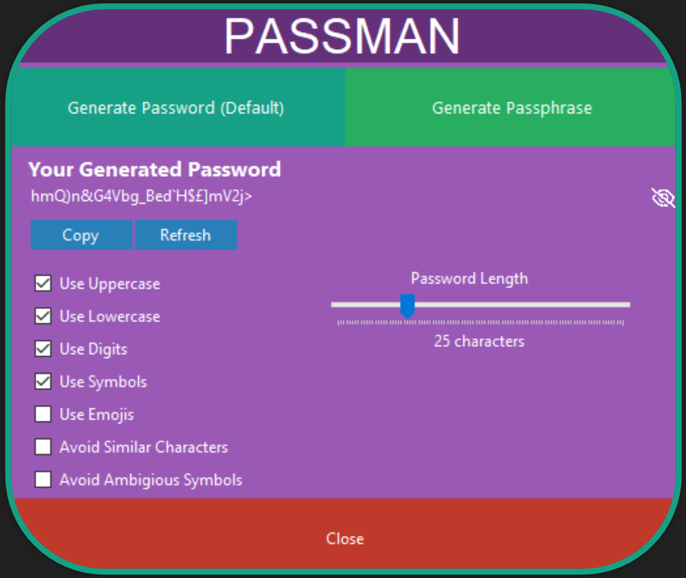


Your Generated Passphrase

according adventures absorption abandon algorithms absorbed allowance adviser alignment access allies agreements achieve affordable acquire aftermath adjust agents accent abroad afford aloud adding altar adjust

Passphrase Length

25 words



Your Generated Password

hmQ)n&G4Vbg_Bed`H$£]mV2j>

- [✓] Use Uppercase
- [✓] Use Lowercase
- [✓] Use Digits
- [✓] Use Symbols
- [ ] Use Emojis
- [ ] Avoid Similar Characters
- [ ] Avoid Ambigious Symbols

Password Length

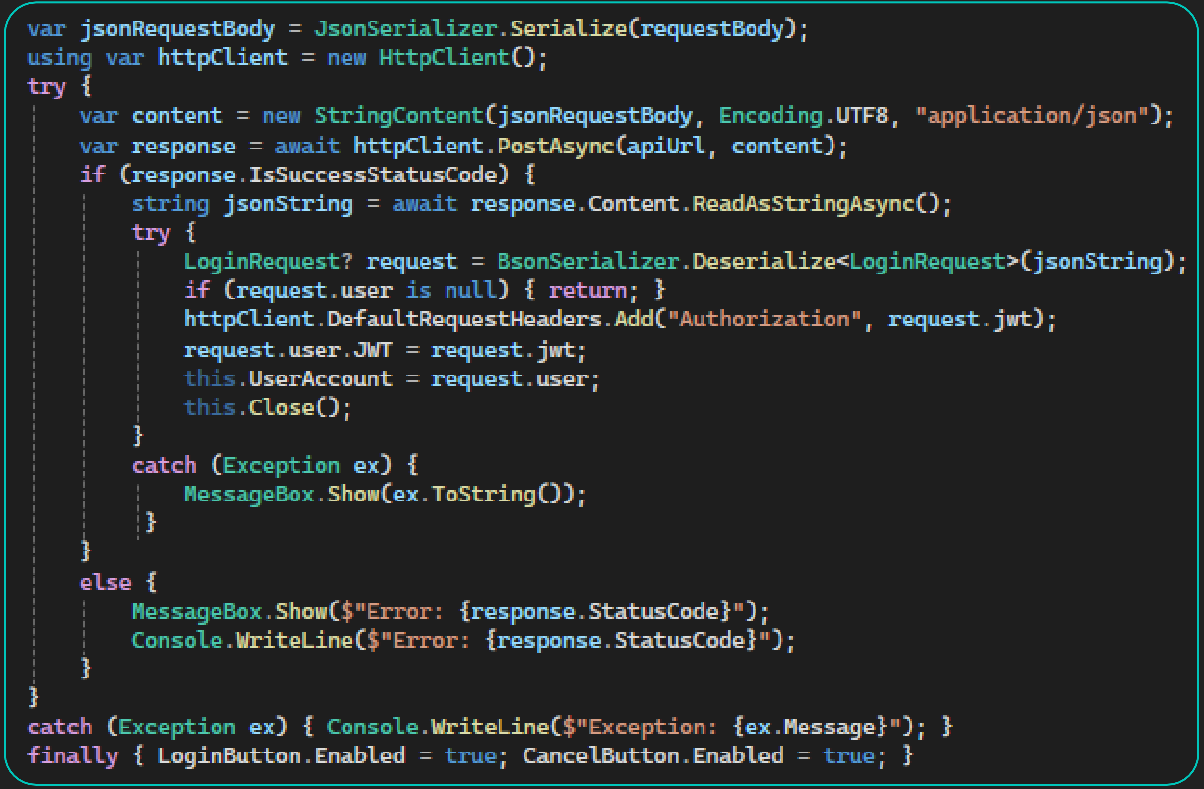
25 characters

Back-End Overview

Presented By: Liam Townsley

How does the app communicate?

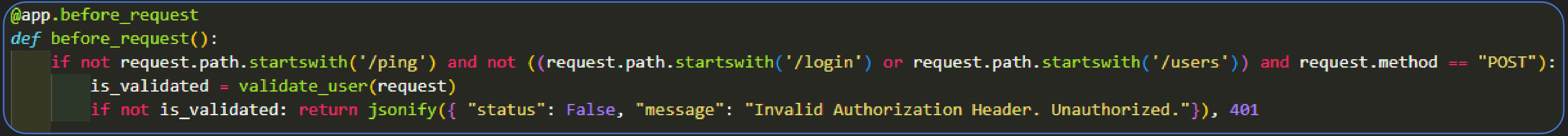
- The application uses a custom API to handle all data, the API then communicates with the Database, this layer of abstraction allows for easy testing of each individual component and allows for changes to be made easily in the application.
- In essence, the Desktop Application has no direct connection with the database, the API is used as the relay.



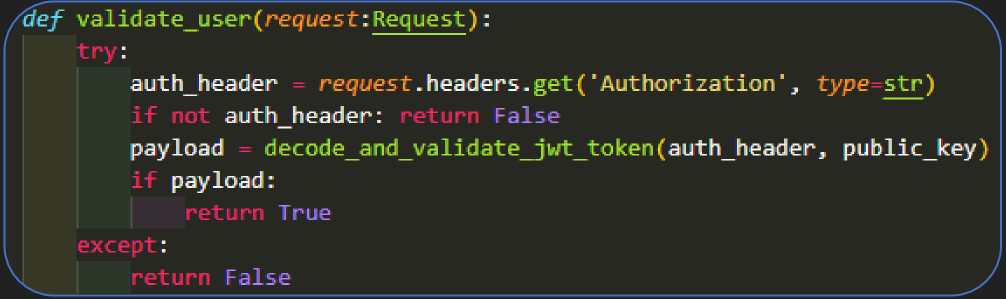
```
var jsonRequestBody = JsonSerializer.Serialize(requestBody);
using var httpClient = new HttpClient();
try {
    var content = new StringContent(jsonRequestBody, Encoding.UTF8, "application/json");
    var response = await httpClient.PostAsync(apiUrl, content);
    if (response.IsSuccessStatusCode) {
        string jsonString = await response.Content.ReadAsStringAsync();
        try {
            LoginRequest? request = BsonSerializer.Deserialize<LoginRequest>(jsonString);
            if (request.user is null) { return; }
            httpClient.DefaultRequestHeaders.Add("Authorization", request.jwt);
            request.user.JWT = request.jwt;
            this.UserAccount = request.user;
            this.Close();
        }
        catch (Exception ex) {
            MessageBox.Show(ex.ToString());
        }
    }
    else {
        MessageBox.Show($"Error: {response.StatusCode}");
        Console.WriteLine($"Error: {response.StatusCode}");
    }
}
catch (Exception ex) { Console.WriteLine($"Exception: {ex.Message}"); }
finally { LoginButton.Enabled = true; CancelButton.Enabled = true; }
```

How is authentication performed on the API?

- On all routes (excluding the Ping, Login or Create User routes) the request is validated for a JSON Web Token. This token includes the users ID and is used to ensure that the user is indeed a valid user, and their credentials are correct.
- The use of this token allows for authentication of the user without requiring the user to send their username and password alongside each request. This is more secure as a man in the middle will not be able to intercept this information as easily.
- The encryption and decryption of this token handled by the 'JWT' (JSON Web Token) module.



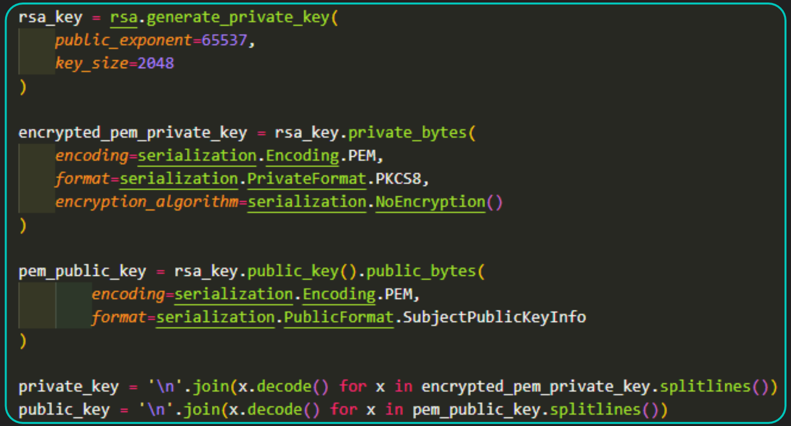
```python
@app.before_request
def before_request():
    if not request.path.startswith('/ping') and not ((request.path.startswith('/login') or request.path.startswith('/users')) and request.method == "POST"):
        is_validated = validate_user(request)
        if not is_validated: return jsonify({ "status": False, "message": "Invalid Authorization Header. Unauthorized."}), 401
```



```python
def validate_user(request:Request):
    try:
        auth_header = request.headers.get('Authorization', type=str)
        if not auth_header: return False
        payload = decode_and_validate_jwt_token(auth_header, public_key)
        if payload:
            return True
    except:
        return False
```

How is authentication performed on the API?

- The JSON Web Token is only as strong as the provided key to encode the data with, to ensure the safety of the key a random RSA keypair will be generated each power cycle of the API.
- This RSA keypair is used for authentication of the JSON Web Token, to ensure that it has not been altered within transit.

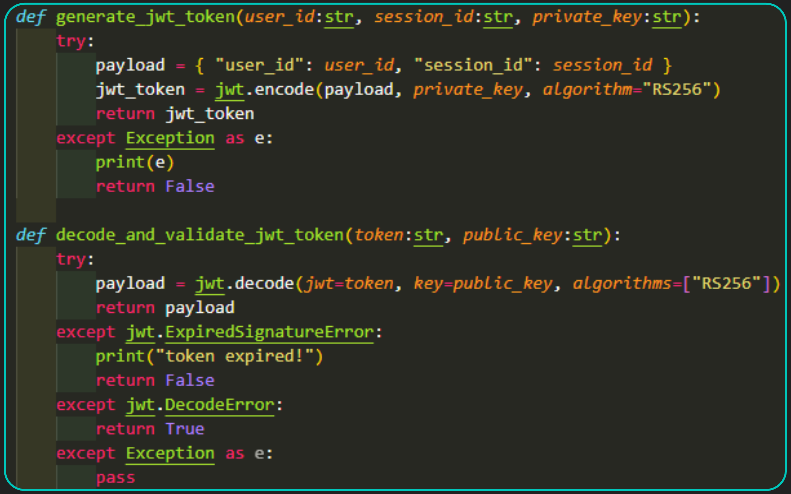


```
rsa_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048
)

encrypted_pem_private_key = rsa_key.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.PKCS8,
    encryption_algorithm=serialization.NoEncryption()
)

pem_public_key = rsa_key.public_key().public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
)

private_key = '\n'.join(x.decode() for x in encrypted_pem_private_key.splitlines())
public_key = '\n'.join(x.decode() for x in pem_public_key.splitlines())
```

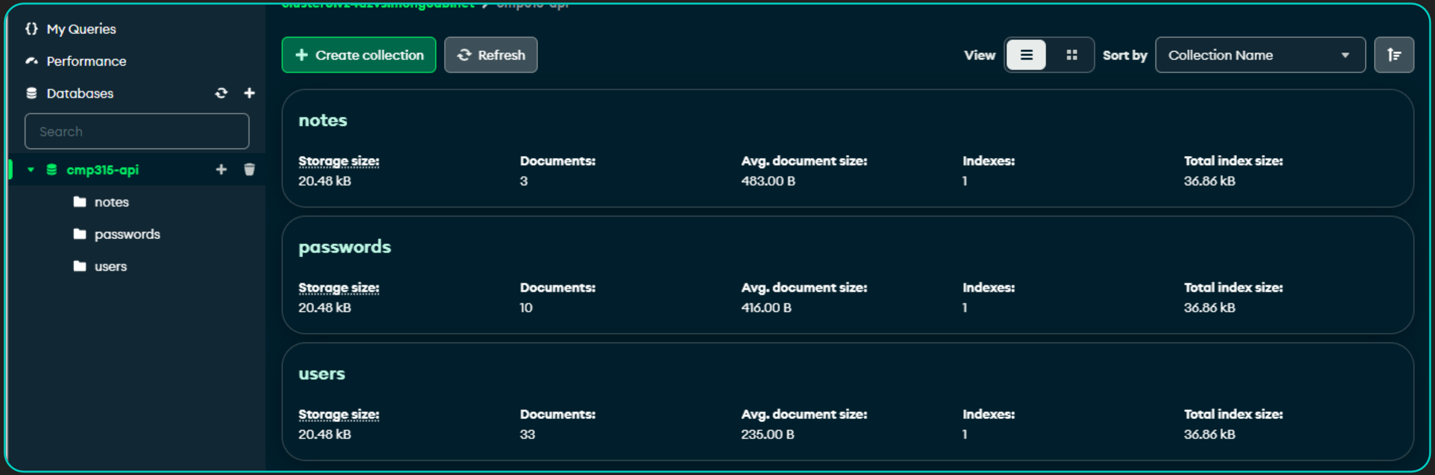


```
def generate_jwt_token(user_id:str, session_id:str, private_key:str):
    try:
        payload = { "user_id": user_id, "session_id": session_id }
        jwt_token = jwt.encode(payload, private_key, algorithm="RS256")
        return jwt_token
    except Exception as e:
        print(e)
        return False

def decode_and_validate_jwt_token(token:str, public_key:str):
    try:
        payload = jwt.decode(jwt=token, key=public_key, algorithms=["RS256"])
        return payload
    except jwt.ExpiredSignatureError:
        print("token expired!")
        return False
    except jwt.DecodeError:
        return True
    except Exception as e:
        pass
```

Where is the data stored?

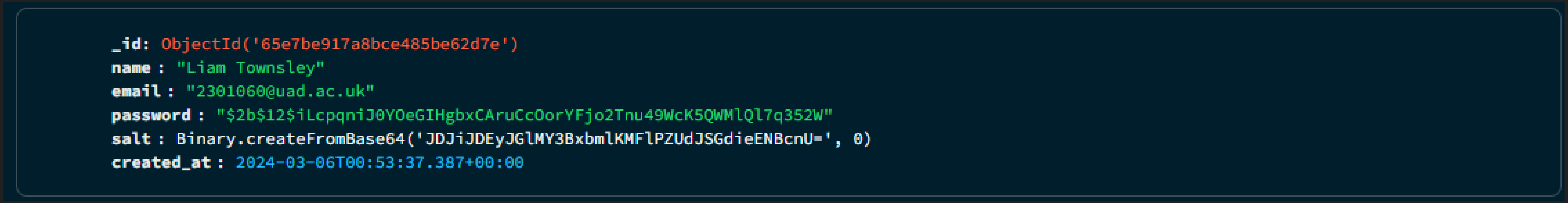
- MongoDB was decided as the Database for this project. As it is a high speed, flexible & simplistic database solution.
- Shown is an overview of the data that is stored within the Demo Database.



	Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
notes	20.48 kB	3	483.00 B	1	36.86 kB
passwords	20.48 kB	10	416.00 B	1	36.86 kB
users	20.48 kB	33	235.00 B	1	36.86 kB

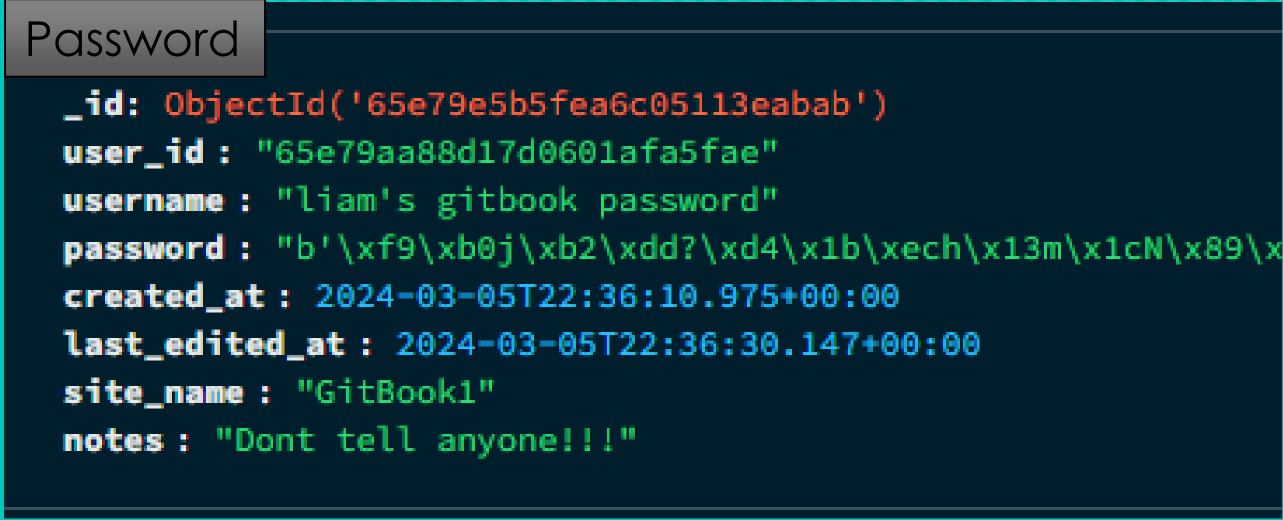
How are user accounts secured?

- To ensure that the user's Master Password is securely stored and that it cannot easily be brute forced and their original password remains safe even in the event of a data leak.
- To do this 'Bcrypt' was implemented as by default it implements Automatic Salting, the password has random characters appended to it to ensure that the hash looks different even if the same two passwords were entered
- This ensures that Rainbow Table Attacks cannot be used against the application's database if it were to be leaked. Each password would need to be brute forced individually, this is computationally expensive and will slow down or even stop an attacker.



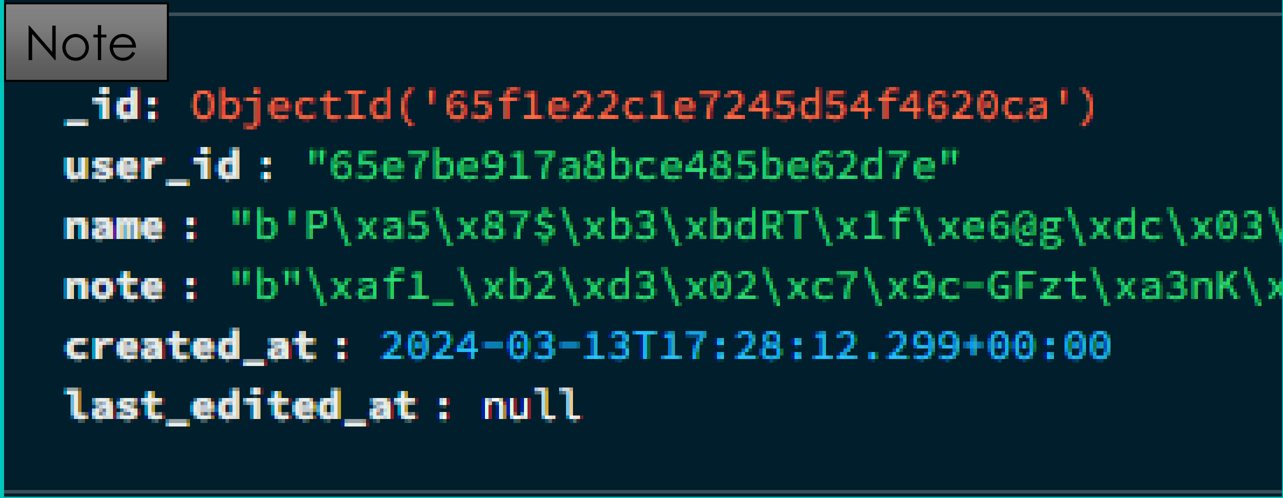
```
_id: ObjectId('65e7be917a8bce485be62d7e')
name : "Liam Townsley"
email : "2301060@uad.ac.uk"
password : "$2b$12$iLcpqniJ0YOeGIHgbxCAruCcOorYFjo2Tnu49WcK5QWMlQl7q352W"
salt : Binary.createFromBase64('JDJiJDEyJGlMY3BxbmlKMFlPZUdJSGdieENBcnU=', 0)
created_at : 2024-03-06T00:53:37.387+00:00
```

How is User Data secured?



Password

```
_id: ObjectId('65e79e5b5fea6c05113eabab')
user_id : "65e79aa88d17d0601afa5fae"
username : "liam's gitbook password"
password : "b'\xf9\xb0j\xb2\xdd?\xd4\x1b\xech\x13m\x1cN\x89\x
created_at : 2024-03-05T22:36:10.975+00:00
last_edited_at : 2024-03-05T22:36:30.147+00:00
site_name : "GitBook1"
notes : "Dont tell anyone!!!"
```



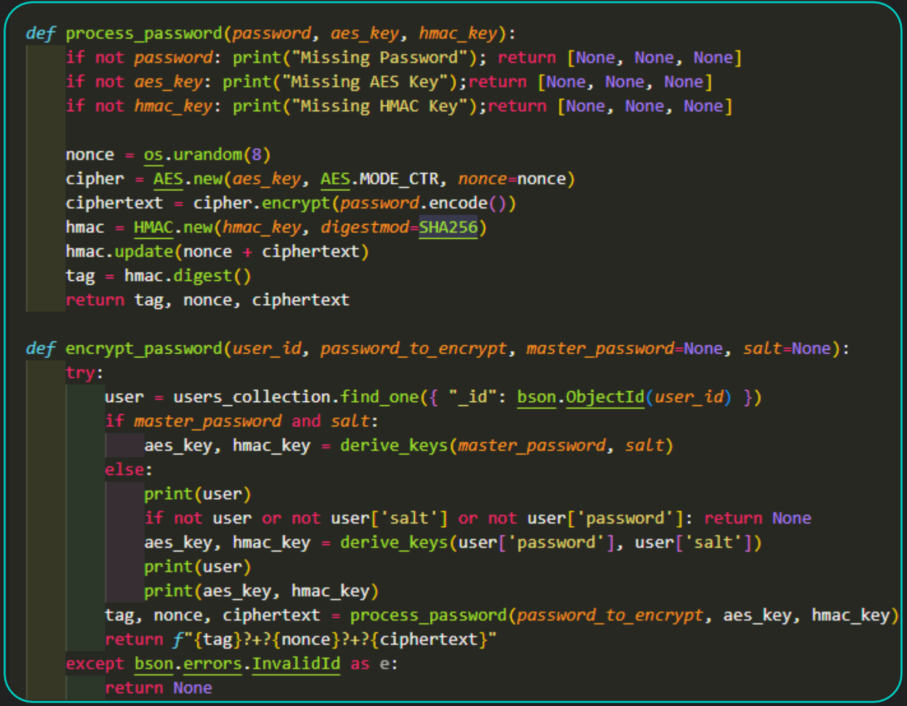
Note

```
_id: ObjectId('65f1e22c1e7245d54f4620ca')
user_id : "65e7be917a8bce485be62d7e"
name : "b'P\xa5\x87$\xb3\xbdRT\x1f\xe6@g\xdc\x03\
note : "b"\xaf1_\xb2\xd3\x02\xc7\x9c-GFzt\xa3nK\x
created_at : 2024-03-13T17:28:12.299+00:00
last_edited_at : null
```

AES Encryption is used to ensure that a large amount of data can be encoded using a shared key, this encrypted using the Master Password of the user.

How is User Data secured? (Encryption)

- As we need to support possibly encrypting a large amount of data (example, Secure Notes), the password is encrypted using AES-256.
- This encryption algorithm requires a shared key is used to allow for encryption & decryption. In this case the Master Password is used.
- To add security to the AES Key, Password Based Key Derivation is used, this uses the password as the input and outputs a cryptographically secure key based on the inputted information (in our case, the password & salt).



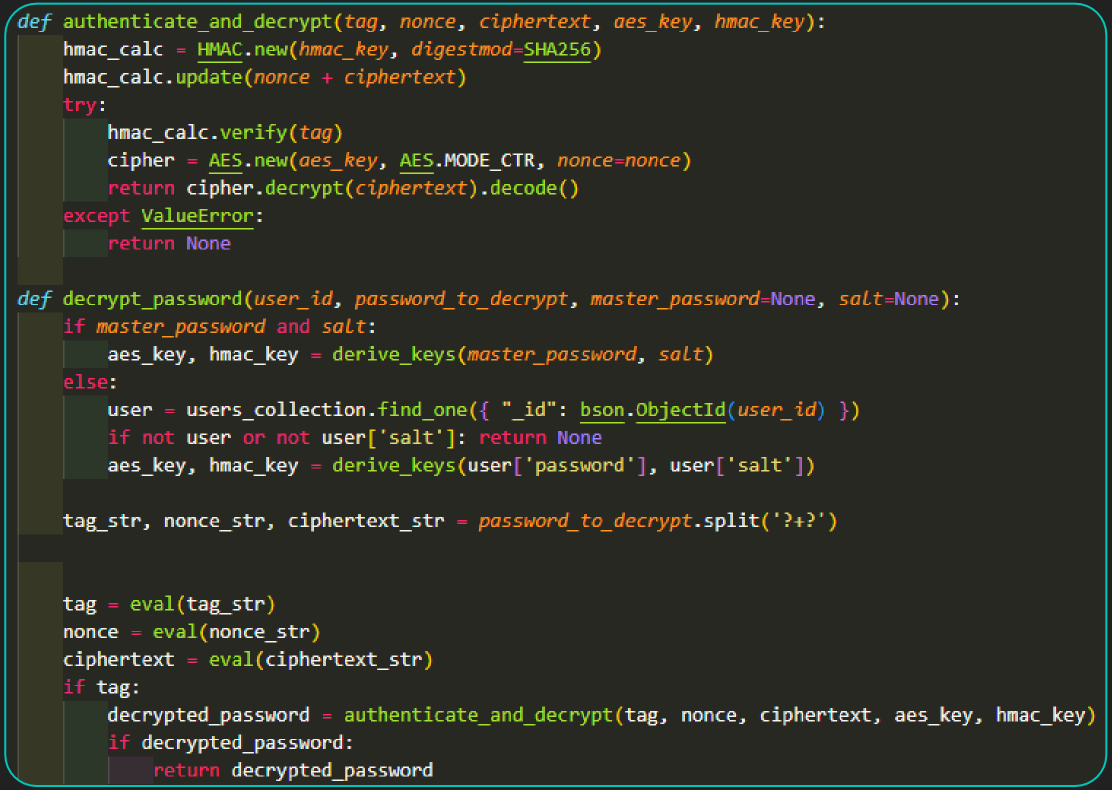
```python
def process_password(password, aes_key, hmac_key):
    if not password: print("Missing Password"); return [None, None, None]
    if not aes_key: print("Missing AES Key");return [None, None, None]
    if not hmac_key: print("Missing HMAC Key");return [None, None, None]

    nonce = os.urandom(8)
    cipher = AES.new(aes_key, AES.MODE_CTR, nonce=nonce)
    ciphertext = cipher.encrypt(password.encode())
    hmac = HMAC.new(hmac_key, digestmod=SHA256)
    hmac.update(nonce + ciphertext)
    tag = hmac.digest()
    return tag, nonce, ciphertext

def encrypt_password(user_id, password_to_encrypt, master_password=None, salt=None):
    try:
        user = users_collection.find_one({ "_id": bson.ObjectId(user_id) })
        if master_password and salt:
            aes_key, hmac_key = derive_keys(master_password, salt)
        else:
            print(user)
            if not user or not user['salt'] or not user['password']: return None
            aes_key, hmac_key = derive_keys(user['password'], user['salt'])
            print(user)
            print(aes_key, hmac_key)
        tag, nonce, ciphertext = process_password(password_to_encrypt, aes_key, hmac_key)
        return f"{tag}?+?{nonce}?+?{ciphertext}"
    except bson.errors.InvalidId as e:
        return None
```

How is User Data secured? (Decryption)

- To allow for the type of encryption we would like to allow with a hybrid of Bcrypt and AES, if the user were to change their Master Password (hashed by Bcrypt) they would lose access to all their data.
- To account for this following a user's Password Change Request the code will decrypt all their passwords using the provided Master Password (the previous one), and then re-encrypt their data using their newly provided password. This information is all done in memory and is not saved to disk.



```python
def authenticate_and_decrypt(tag, nonce, ciphertext, aes_key, hmac_key):
    hmac_calc = HMAC.new(hmac_key, digestmod=SHA256)
    hmac_calc.update(nonce + ciphertext)
    try:
        hmac_calc.verify(tag)
        cipher = AES.new(aes_key, AES.MODE_CTR, nonce=nonce)
        return cipher.decrypt(ciphertext).decode()
    except ValueError:
        return None

def decrypt_password(user_id, password_to_decrypt, master_password=None, salt=None):
    if master_password and salt:
        aes_key, hmac_key = derive_keys(master_password, salt)
    else:
        user = users_collection.find_one({ "_id": bson.ObjectId(user_id) })
        if not user or not user['salt']: return None
        aes_key, hmac_key = derive_keys(user['password'], user['salt'])

    tag_str, nonce_str, ciphertext_str = password_to_decrypt.split('?+?')


    tag = eval(tag_str)
    nonce = eval(nonce_str)
    ciphertext = eval(ciphertext_str)
    if tag:
        decrypted_password = authenticate_and_decrypt(tag, nonce, ciphertext, aes_key, hmac_key)
        if decrypted_password:
            return decrypted_password
```

System Architecture Overview

Presented By: Ryan van Ee

Written By: Ryan van Ee & Liam Townsley

Architecture Breakdown

The architecture of the application is broken down into 3 sections:

- Client Application (A Desktop Application in C#)
- Application Interface (An API coded in Python)
- Database (Using MongoDB)

Client Application

The Client Interface (Windows Desktop App) is responsible for the front end of the application, it is programmed using C# in Visual Studio (VS), the VS Designer was also utilized in the development of the User Interface.

This application gives the user the visual representation and feedback whilst maintaining a communication with the API to facilitate the user's requests.

Application Layer (API)

The Application Layer (API) is responsible for the backend of the application, it is programmed using Python and uses multiple packages provided through pip (the Python Package Management System) to provide utility and security-based functions.

The API oversees all the data that the Desktop Application interacts with, the API facilitates a connection with the Database and is used to ensure that the Desktop Application can request and send the required data.

Database layer

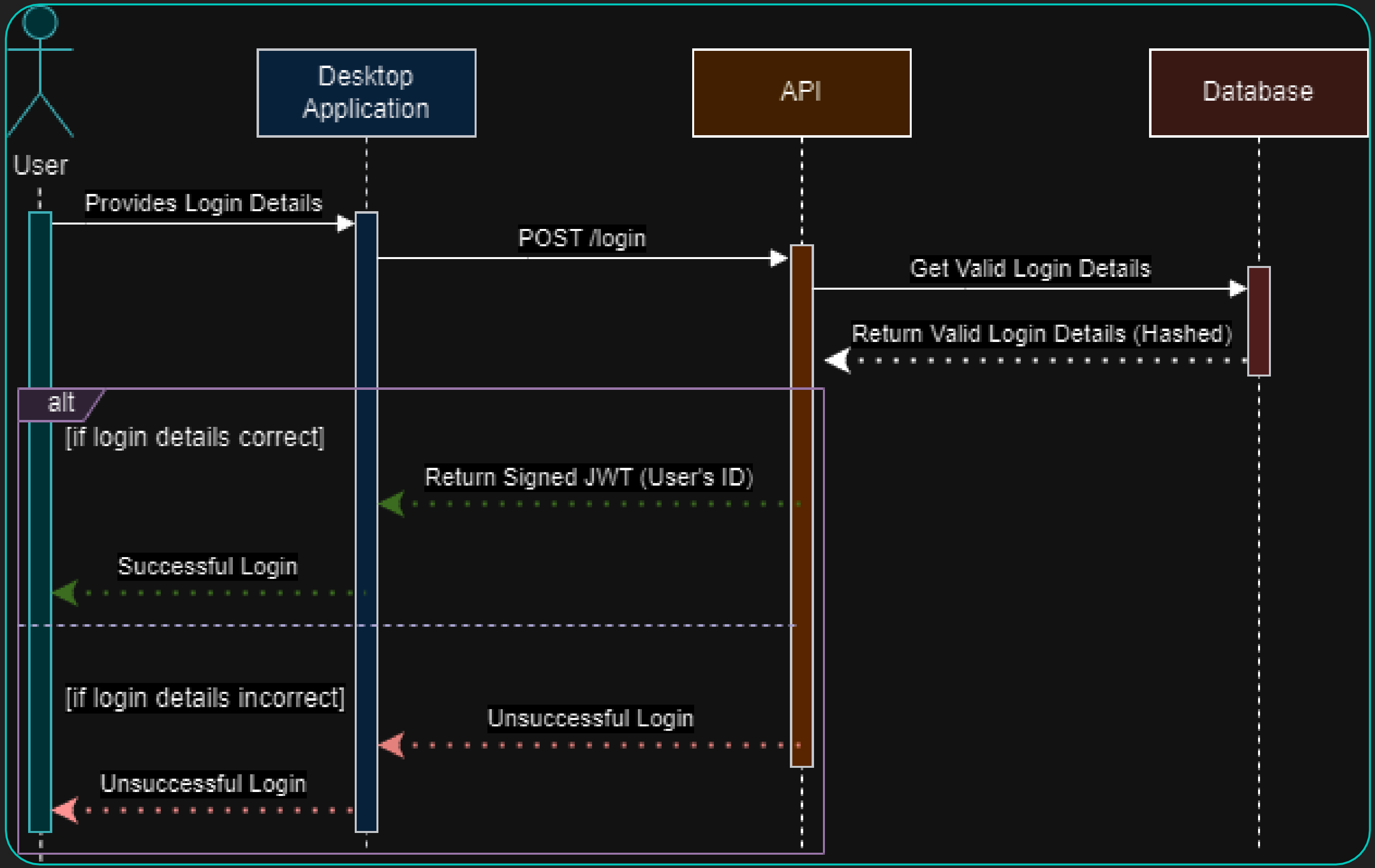
The Database Layer is responsible for storing the data that the system may require, this layer is inaccessible from the Desktop Application and can only be accessed through the API.

The Database Layer currently stores the following information and can transact it with the API:

Master Account User Details (within the 'users' collection)
Saved Passwords (within the 'passwords' collection)
Saved Notes (within the 'notes' collection)

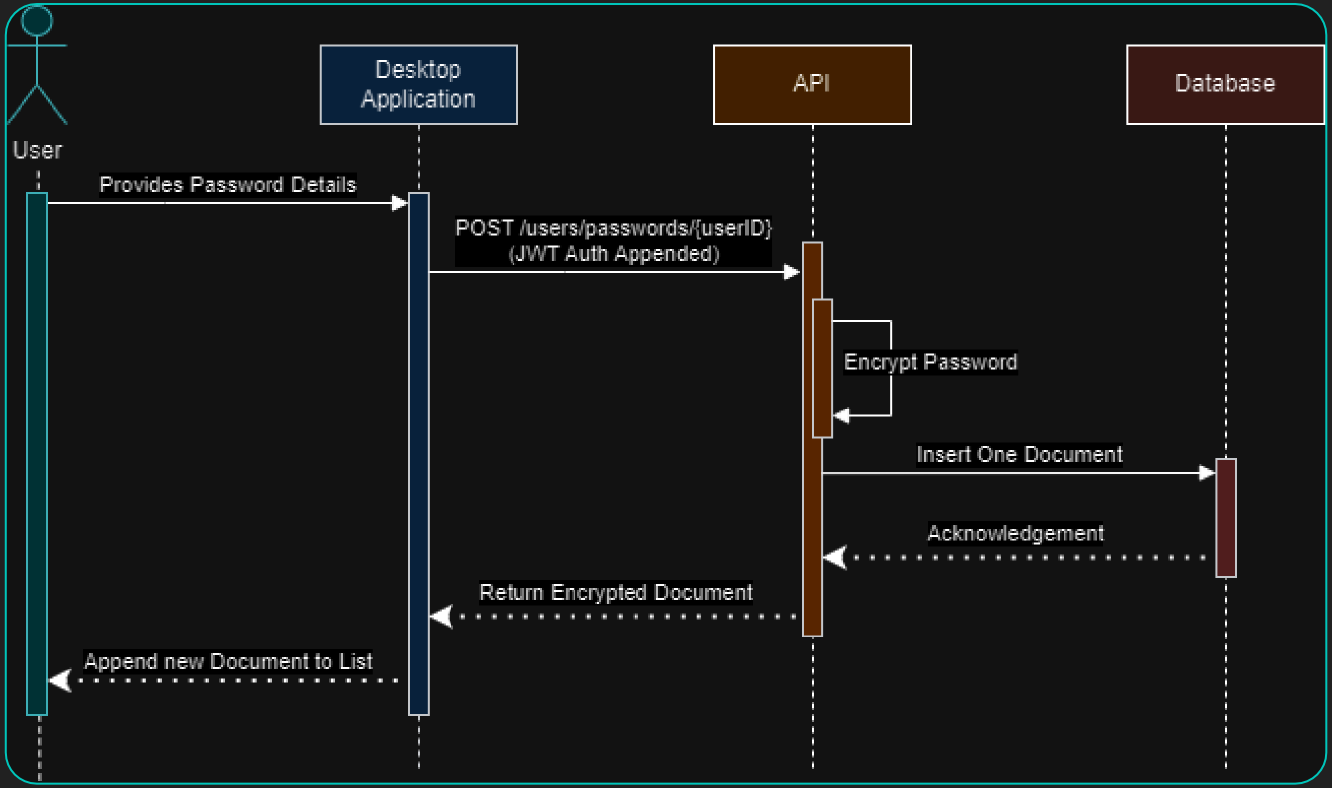
Sequence Diagram – Logging In

This sequence diagram shows a visual representation of the process the system would undertake if a user were to attempt to login.



Sequence Diagram – Saving a Password

This sequence diagram shows a visual representation of the process the system would undertake if a user were to attempt to save a password.


User
Desktop Application
API
Database
Provides Password Details
POST /users/passwords/{userID} (JWT Auth Appended)
Encrypt Password
Insert One Document
Acknowledgement
Return Encrypted Document
Append new Document to List

Testing Overview

Presented By: Jack Barnet

Snyk Testing

- Snyk Vulnerability scanner is a tool that scans for vulnerabilities in your code and automatically fixes them adding it to the github. It supports languages like javascript, python, c# any many others for this case we used it for python & C#

API & Project 'requirements.txt' Vulnerabilities

Within the following vulnerabilities the dependencies were noted to have a an outdated / vulnerable version of the modules installed. This has since been rectified.

- CWE-696 – incorrect Behaviour Order
- CVE-2023-29483 - sending an invalid packet from the expected IP address and source port, aka a "TuDoor" attack
- CVSS 5.9
- SNYK-PYTHON-DNSPYTHON-6241713 – incorrect behaviour order

Sensitive Information Unencrypted

Initially the project stored certain information in plain text but to mitigate this the AES-256 encryption was implemented to ensure that even if the database is compromised the stored information remains secure and unreadable without a decryption key.

Implementation of Authentication

Authentication mechanisms were implemented in the API to ensure secure access to endpoints. This was implemented as it was found there was no authentication, so industry standard authentication protocols were added. The implementation of API tokens was created so that users couldn't impersonate others and obtain access to accounts.

Deletion of accounts

While testing it was discovered that the system allows users to delete their accounts without properly invalidating existing accounts. This left it so that users could delete their accounts but remained still logged in and access their passwords. This is a significant security risk as unauthorized access to sensitive data could occur even after the user intends to terminate the account. Proper session management upon account deletion was implemented.

Questions?