



National College of Ireland

BSHC-4

Cyber-Security

2021/2022

Shaurya Kumar

X18138284

X18138284@student.ncirl.ie

Cyber-Vault

Final Report

Contents

Executive Summary.....	4
Introduction	4
Background	4
Technologies	5
Structure	5
Requirement Procurement and Analysis	6
Functional Requirements analysis	6
1. Requirement: Secure Login registration	7
2. Requirement: Password management	11
3. Requirement: Password Generator	19
4. Requirement: Password distinctor	20
5. Requirement: Password strength analyser.....	22
6. Requirement: Facial registration (updated).....	23
Non-Functional Requirements analysis	25
Project Plan and Analysis	27
Design and Architecture	30
ERD.....	32
Implementation	32
Graphical User Interface	34
Final Technology, requirements and Implementation	37
Final Technology stack	37
Final Functional requirement completion	41
More Interesting Code snippets	54
Final Screenshots of application and GUI	58
Security Focused Implementation in the application	65
Security Principles and Pernicious kingdoms incorporation.....	77
Causes of changes in technology stack and implementation	79
Testing & Evaluation	80
Future Development or research.....	87
Conclusion.....	89
References	89
Appendices.....	91
Project Proposal.....	91
Objectives.....	91
Background	91

State of the Art.....	92
Technical Approach.....	93
Technical Details	95
Project Plan	96
Testing.....	98
Journal Entries.....	100
December.....	100
November	101
January	102
February.....	103
March.....	104
April.....	106

Executive Summary

This Mid-point report is a requirement for the final year Software project which is to mainly display our progress and the future plan according to it. One of its goals is also to help us analyse the difficult points of our application; what difficult points have been achieved and how to tackle any other challenging matter ahead. The main purpose of this report is to emphasize on my progress till date, current designs and implementation, deviation from original plans and changes included as well as alternate implementations/solutions utilised, results of current progress etc.

The key points of the report include the current progress which include the implementation of the basic React application implementing Facial recognition login/registration using *openCV's face_recognition* as a part and in combination with the *dlib* library which contains the implementation of deep metric learning used to convert face image sin 128 bit vectors used for comparison and authentication, this is all done in the backend flask server details of which are explained in the project analysis sections. Another major implementation as a part of the current progress includes the implementation of the Have I Been Pwned troy hunt cross-referencing library extension as a part being accessible after a successful login. Which implements a react component implementing the Have I been Pwned API thus helping recognize the if a password has been breached or used before and how many times, thus declaring it safe or unsafe regarding its use. Further important parts of the report include how the implementation of previously planned features has been altered while being developed and why they were changed. The report also importantly includes the use cases for all potential functional requirements with their descriptions and wireframes for the final potential designs of the application also including the non-functional requirements which indicate what the prototype is achieving, satisfying and what its capable of as well as it's though of architectural structure which is all explained below.

Introduction

Background

My plan for my final year Software project is to build CyberVault which shall act as an ultimate secure password managerial all in one application with additional features for customer appeasement and satisfaction by majorly focusing on the population with a strong concern for security through easy management. CyberVault is different from most managerial applications out there since it will implement all the ways to secure an account there are, to cover for the potential obstacles of using master password which could be stolen, hacked or exploited. This is done through implementing multiple alternate login ways which do also include a master password but depend on all secure authentication factors through the implementation of facial recognition authentication collaboratively being implemented with multi-factor authentication.

It is critical such an application exists so as to help the customer, since now customers have a variety of critical passwords and documents pertaining to their everyday life in today's world, and there is no way to secure and manage them all in one location thus to make their life easier kind of like a critical business process solution being provided to a developing company to make its life easier than to have do everything themselves and worry about the security surrounding all their processes. CyberVault is designed to serve as a customer's ultimate third-party security contractor for all of their critical credentials and to satisfy their concern for security surrounding them. The main features the application would include are; Facial recognition authentication using deep learning libraries, Multi-facto/2FA authentication, password strength analyser, customer tailored password generator, Password distinctor (uniqueness analyser), Secure password storage with high level

symmetric encryption (AES) and ease of password managerial structure and an optional secure file storage.

Technologies

1. Current technologies in use: For now the foundational development as the main framework for the application I am using React JS in the front-end (Class component structure) with a python flask server in the back-end holding the face_recognition library of OpenCV build using the state of the art deep learning face recognition of dlib which is being used for the facial authentication and registration with SQL as the main database for now holding the face encodings (also utilising numPy for handling high level mathematical operations) and the user credentials for verification and the webcam extension in python used to utilize the webcam for functionality, all these differ from the original plan of using tensor flow and face net model according to the previous project proposal. On the front-end I am currently using the class component structure to utilize the python back-end and complete the facial login and registration function. The front end also utilizes the implementation of the react have I been pawned API connection to check the authenticity of passwords for my demonstration prototype.
2. Future technologies planned on implementation: further for the application's feature suite and management development I plan on using **express** and **node** in the back-end as well, **crypto JS** for high level AES encryption of all passwords being saved using the application, toastify for system notifications, **SQL**, Fauna or **Mongo** as the database components according to implementation, zxcvbn library for password cracking inspired password guessing strength analysis, JWT (JSON Web Tokens), Auth0, Twilio SMS API for multifactor authentication etc.

Structure

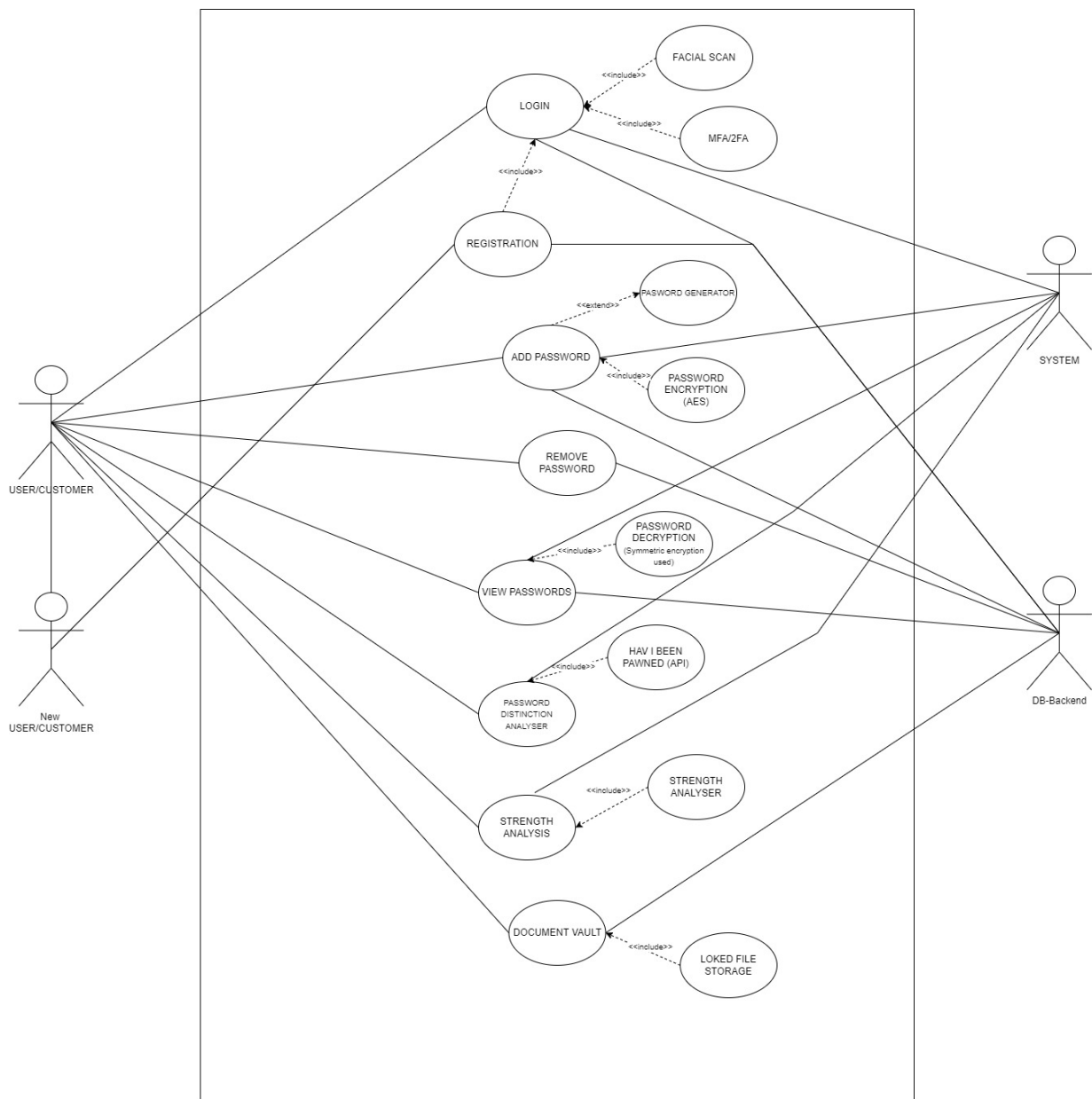
The structure of the whole report includes the executive summary as a gist of the key points for the whole report and the necessary details mentioned as updated from the initial project proposal, further the report includes a brief introduction and background concerning the reason to build this application and its benefits and differences from pre-existing similar software. After this the report divulges into the requirements engineering for this project detailing the functional requirements using basic use case diagrams and their respective descriptions with a brief description and priority of function and its requirement regarding the project. After the functional requirements are defined the report moves on to the non-functional requirements concerning the application which includes screenshots of the prototype handling some of these requirements. Then the report moves on to the project plan and analysis section which includes the screen shots of the plan that has been defined on trello with a partially updated gantt chart of the first quarter which then moves on to the tasks in the original gantt chart. Finally the last sections of the report will include the Architecture and definition using a sample system architecture diagram with definition and wireframes used for description in the GUI section with screenshots from the current prototype and potential future wireframes of application UI. At the end the report would include the original project proposal as well.

Requirement Procurement and Analysis

Requirement gathering is a very important process as a part of the software development lifecycle since it is important to have documented evidence of all the functions the software application will be performing for the customers with an in-depth analysis of all functional requirements through use case diagrams and their respective use case descriptions, as well as non-functional requirement analysis.

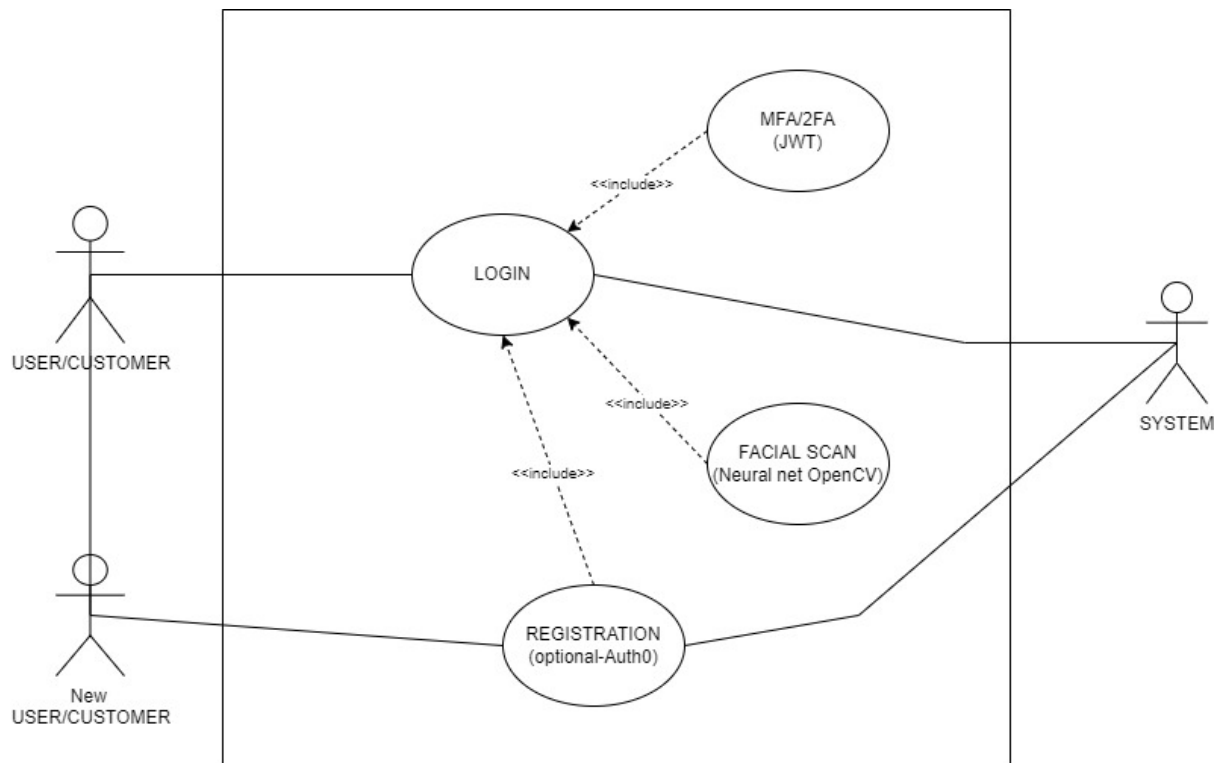
Functional Requirements analysis

Below is an overall Use case diagram of the whole system including most of its functional requirement use cases, which are further separated into individual use case diagrams and their respective descriptions with the requirement description and function as their respective headings.

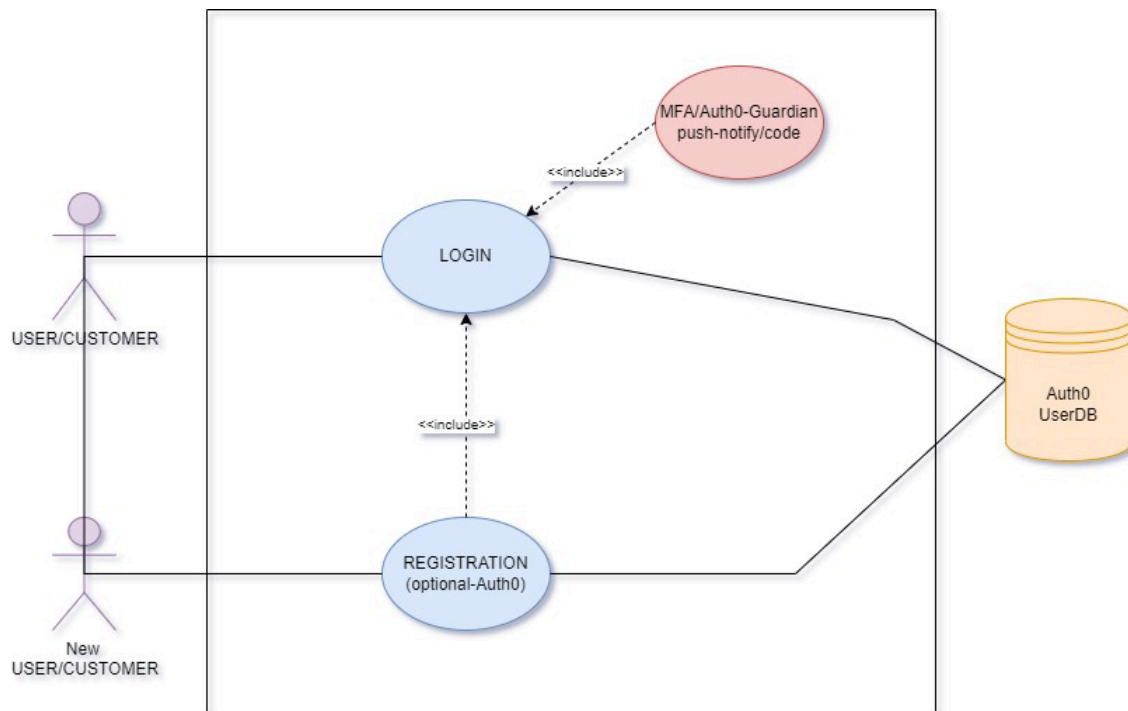


1. Requirement: Secure Login registration

The secure Login and registration is one of the most important high priority functional requirements necessary to be satisfied for this application. This functional requirement is responsible for defining the access control to this web-application. The Login registration feature is to be implemented using facial recognition authentication coupled with multifactor/2-Factor authentication to give priority to a secure and properly authenticated login to manage access control of users to the application profiles and feature suite. Currently the login/registration has been implemented with flask server utilising face-recognition library using the dlib library having deep learning components and neural nets to capture faces specifically and compute 128 bit encoding of the faces which are used for comparison and authentication. Following is the use case diagram and the respective use case descriptions for them.



Updated use-case diagram -



Login Use-case description:

Description:	<p>This use case allows the user to Login into the web application through proper User authentication going through Facial authentication and multifactor verification.</p> <p>Updated – No facial scan in the beginning (Only multifactor authentication)</p>
Actors:	
Primary	User
Secondary	System
Triggers:	<p>User clicks on the Login button after entering their valid credentials (and face scan done through webcam)</p> <p>Updated – (face scan is now in password manager)</p>
Pre-conditions:	<p>The user should be registered with the system with pre-existing credentials and facial neural net in their respective databases</p> <p>Updated - this is a precondition for viewing the manager</p>
Post-Conditions:	User is logged in and has accessibility to the system feature suite
Normal Flow:	<ol style="list-style-type: none"> 1. The user enters personal credentials 2. The user clicks Login button 3. The user face snap is taken through webcam and converted into

	<p>128bit vector by neural net then compared with stored vector through the database (updated – happens during viewing the manager dashboard)</p> <p>4. The user enters either OTP or SMS or whichever second factor is registered</p> <p>5. User is logged in after primary and secondary factor are verified</p>
Alternate Flows:	<p>1a Invalid login credentials</p> <p>1 system notifies to enter valid details</p> <p>2 system resumes to normal flow 2</p> <p>3 system notifies to register first and redirects to registration</p> <p>Use-Case</p> <p>(updated alternate flow belongs to manager use-case)</p> <p>3a Facial snap not registered</p> <p>1 system notifies to register first</p> <p>2 system redirects to register Use-case</p> <p>3b Unavailable webcam</p> <p>1 User chooses alternate login</p> <p>2 system resumes to normal flow 4</p>
Exceptions:	<p>5a Server down/ http request failed system notifies “Server communication failure”</p> <p>1 The System returns to normal flow 1</p>
Includes:	<p>Login includes two factors to authenticate either facial scan and secondary factors or alternate login. Also includes redirection to registration if user not in the system. (updated – only includes multifactor in the beginning)</p>
Extends:	None
Priority:	High
Special Conditions:	<p>1. Secondary factor should be registered too as well</p>
Assumptions:	User is registered in the system with valid existing details

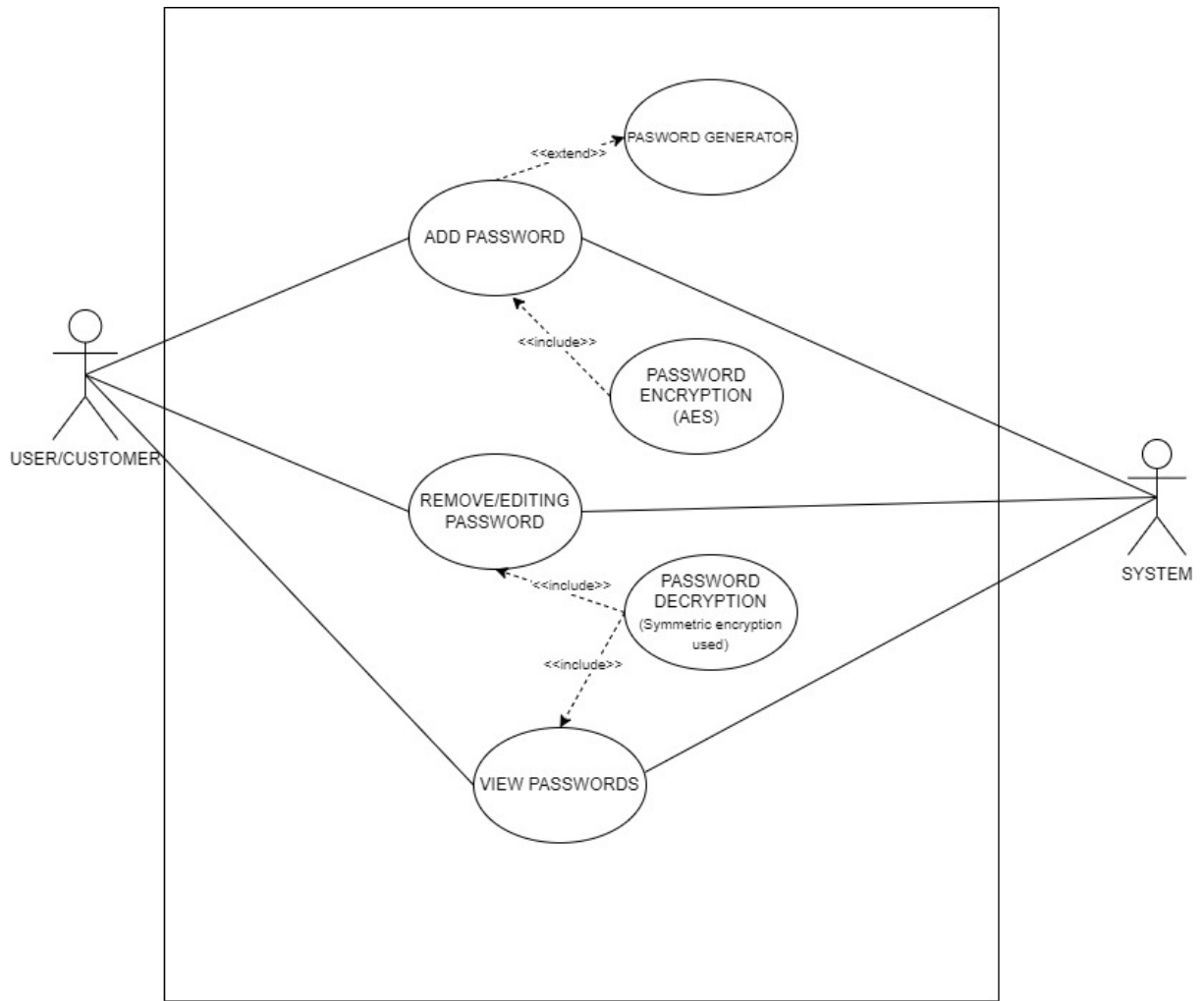
Registration Use-Case description:

Description:	This use case allows the user to register themselves into the web application servers for proper User authentication through Facial authentication (updated - facial registration in inside the application) and multifactor verification.
Actors: Primary Secondary	User System
Triggers:	User clicks on the registration button after entering their valid credentials (and face scanned through webcam and vector saved) (Updated – face registration now inside the feature suite)
Pre-conditions:	The user should either have a webcam or choose alternate method for registration (updated – precondition for feature suite face registration and manager viewing through face authentication)
Post-Conditions:	User is registered in the system and can login successfully to gain accessibility to the system feature suite
Normal Flow:	<ol style="list-style-type: none"> 1. The user enters personal credentials 2. The user clicks register button 3. The user face snap is taken through webcam and converted into 128bit vector by neural net then saved into the database with necessary credentials (updated - done on the face-registration page) 4. The user registers for the secondary factor 5. User is registered in the system respective databases and back-end
Alternate Flows:	<p>3a Unavailable webcam</p> <ol style="list-style-type: none"> 1 User chooses alternate registration 2 system resumes to normal flow 4 <p>(updated – alternate flow for facial registration feature)</p>
Exceptions:	<p>5a Server down/ http request failed or ill-configured database connection system notifies “Backend communication failure”</p> <ol style="list-style-type: none"> 1 The System returns to normal flow 1
Includes:	

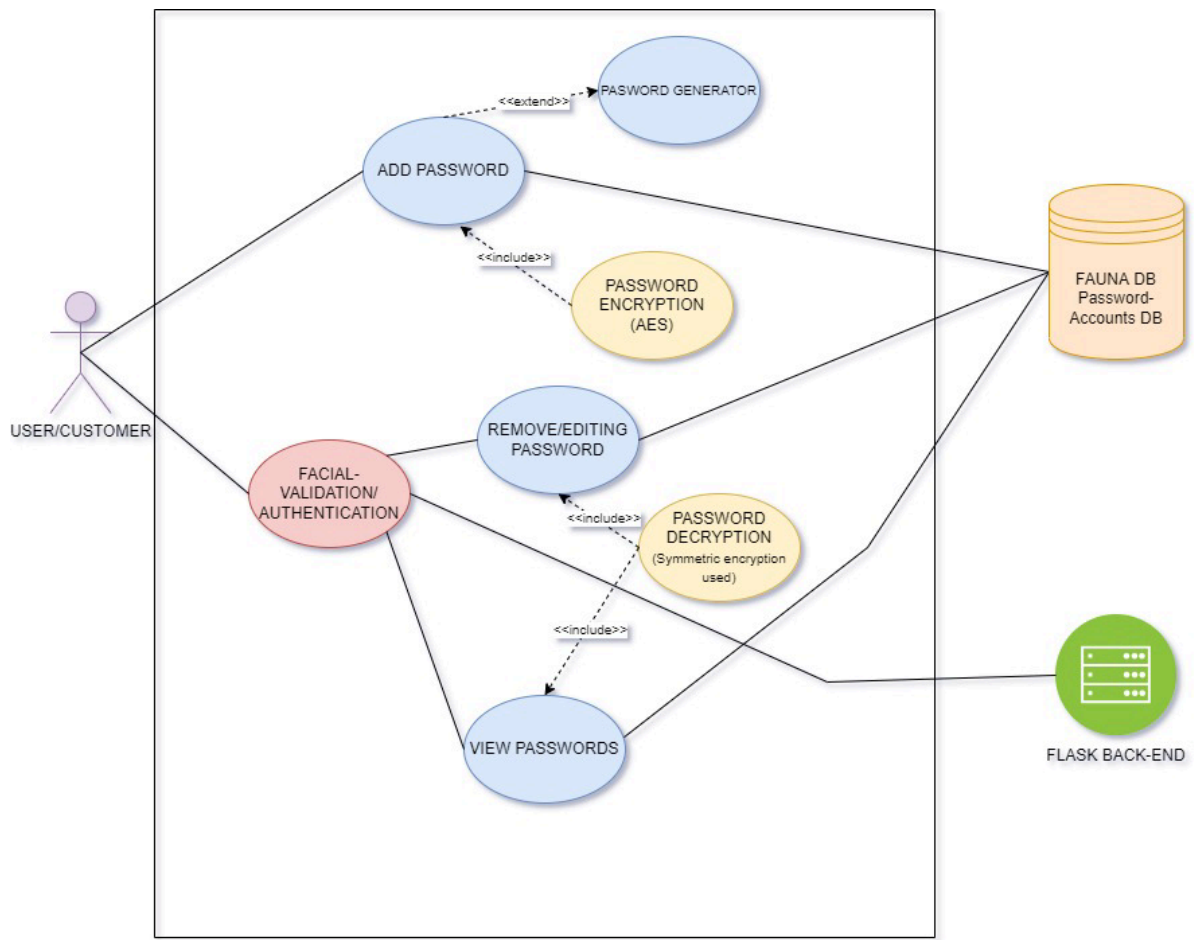
Extends:	None
Priority:	High
Special Conditions:	
Assumptions:	User is providing valid details to be cross-referenced later

2. Requirement: Password management

Another one of the most important functional requirement of the application is its password management and security feature which serves as the main marketing purpose. The password management functionality includes multiple use cases such as adding a password, removing a password, updating and viewing password list. This is a high priority functional requirement for the application which gives the customer the ability to save a plethora of passwords under secure encryption. For this managerial functional requirement I plan on using Crypto JS library which helps encrypting passwords cryptographically and securely and provides multiple options (AES, DES, Triple DES, Rabbit, RC4) for it. All passwords will be saved using symmetric high level AES (Advanced encryption standard) encryption (which include AES-128, AES-192 and AES-256 (passphrase) bit encryption which is chosen depending on the size of the password or security key that is passed in). Each time the user tries to add a password it will be encrypted using crypto JS and then saved into the database and whenever the user tries to receive the password list to either alter or remove a password saved It will first be decrypted using a private key and shown to the user. Following is the Use case diagram regarding this functional requirement and its respective use case descriptions.



Updated Password Manager Use-case diagram -



Adding a password:

Description:	This use case allows the user to utilize the password management functions of the application. Specifically Adding.
Actors:	
Primary	User
Secondary	System
Triggers:	User clicks on the add password button after entering their password for their concerned website or personal use
Pre-conditions:	The user should be successfully logged into their respective profile

Post-Conditions:	Password entered is securely encrypted (AES) and saved into the system passwords database
Normal Flow:	<ol style="list-style-type: none"> 1. The user enters password to be added 2. The user enters the respective application-name for which the password is being saved for. 3. The user clicks add password button 4. The database uses React libraries such as crypto-Js to encrypt the password with AES chosen encryption and further stores it into the database 5. The password is added with its respective application of concern to the system database after being securely encrypted
Alternate Flows:	<p>1a. User decides to generate a password and copies that to clipboard then decides to add that instead of manually entering a password</p> <ol style="list-style-type: none"> 1. System resumes to normal flow 3
Exceptions:	<p>5a Server down/ http request failed or ill-configured database connection system notifies "Backend communication failure, unable to add password"</p> <ol style="list-style-type: none"> 1. The System returns to normal flow 1
Includes:	Password symmetric encryption done using crypto JS and AES encryption by system before adding the password to the respective database with respective details.
Extends:	The user can decide to make use of the secure password generation feature of the application to generate a strong password according to their requirements to be added and used by them for personal use.
Priority:	High
Special Conditions:	Password before adding should be checked with the strength analyser
Assumptions:	User is properly authenticated and logged in

Removing/Altering a password and related credentials:

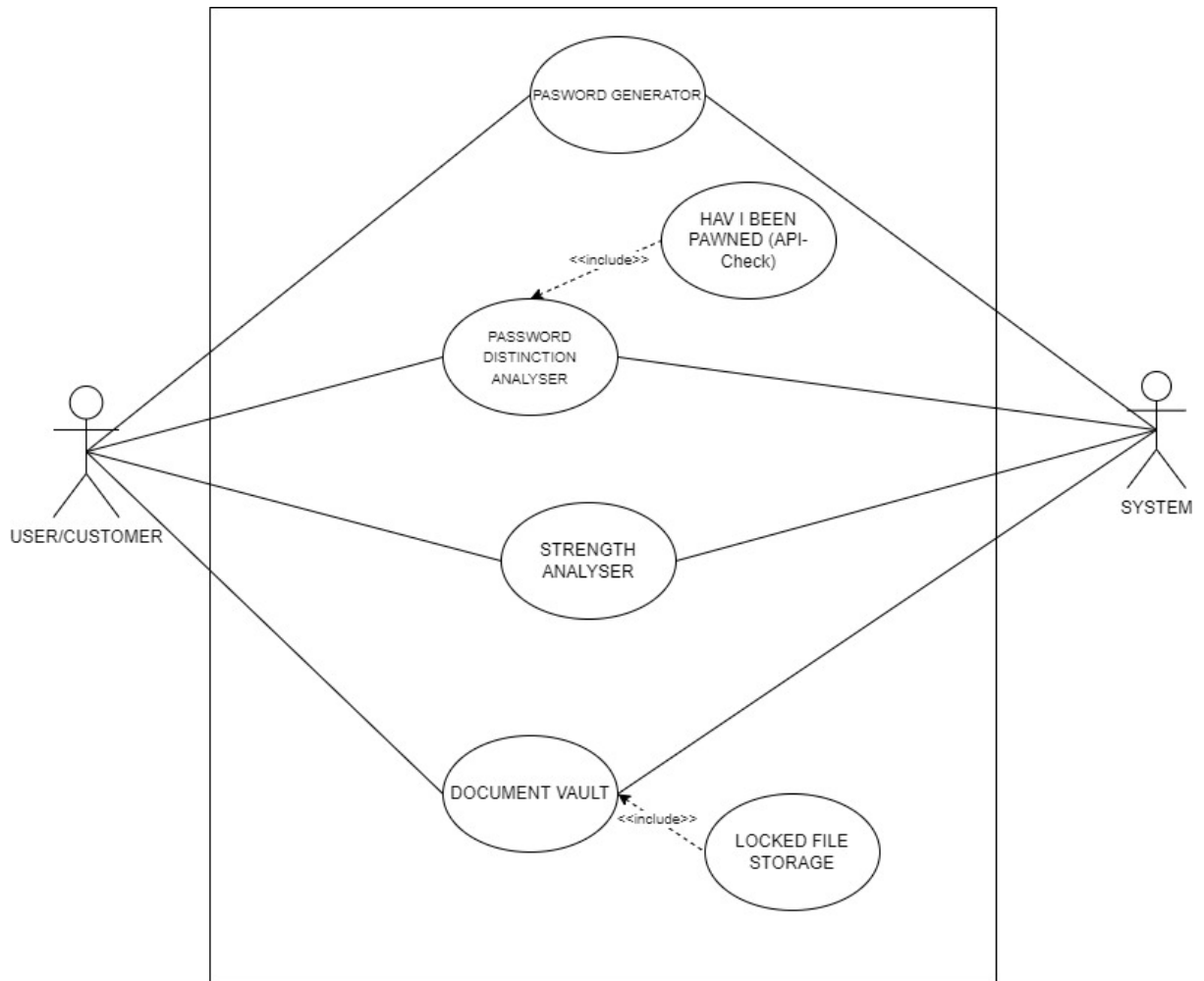
Description:	This use case allows the user to utilize the password management functions of the application. Specifically removing or editing passwords.
Actors: Primary Secondary	User System
Triggers:	User clicks on the remove/edit password button after selecting their added password for their concerned website or personal use
Pre-conditions:	The user should be successfully logged into their respective profile and should have a pre-existing saved password in the system Updated – and should have registered face – face authentication required to edit, view and delete password.
Post-Conditions:	Password entered is removed from the system passwords database or existing password details are altered
Normal Flow:	<ol style="list-style-type: none"> 1. Updated – user validates face scan 2. The user selects password to be removed 3. The user clicks remove password button 4. The database retrieves the selected previously saved password, the system then decrypts it (private decryption key (symmetric encryption used)) and displays it 5. The password is removed/changed with its respective application of concern from/in the system database after being selected
Alternate Flows:	<p>Updated - 1a – Facial ID might not exist (not registered)</p> <p>2a user selects password for editing either the password or related credentials (such as respective application etc.)</p> <ol style="list-style-type: none"> 1 User chooses to edit added password or related credentials 2 system resumes to normal flow 3
Exceptions:	<p>Updated - 1a – Webcam not working, face-auth server offline</p> <p>4a Server down/ http request failed or ill-configured database connection system notifies “Backend communication failure, unable to remove/edit password or password credentials”</p> <ol style="list-style-type: none"> 1 The System returns to normal flow 1
Includes:	Password decryption done by using private decryption key by system before removing or editing the password from/in the respective database with

	respective details.
Extends:	None
Priority:	Medium
Special Conditions:	Updated – mandatory facial scan required
Assumptions:	User is properly authenticated and logged in and has added a password into the system already

Viewing all passwords previously saved by the customer:

Description:	This use case allows the user to utilize the password management functions of the application. Specifically viewing the passwords and deciding CRUD operation further
Actors:	
Primary	User
Secondary	System
Triggers:	User clicks on the view passwords button/tab after logging in
Pre-conditions:	The user should be successfully logged into their respective profile and should have a pre-existing saved password in the system Updated – and should have registered face – face authentication required to edit, view and delete password.
Post-Conditions:	Passwords are decrypted and displayed on the page
Normal Flow:	<ol style="list-style-type: none"> 1. Updated – user validates face scan 2. The user selects or navigates the view passwords endpoint or clicks on the view passwords tab 3. The database identifies the previously saved password, the system then decrypts them (private decryption key (symmetric encryption used)) and displays all of them on the page

Alternate Flows:	<p>Updated - 1a – Facial ID might not exist (not registered)</p> <p>3a deciding CRUD operations for password related credentials (such as respective application etc.)</p> <ol style="list-style-type: none"> 1. User chooses to edit/remove added passwords or related credentials 2. system resumes to normal flow 2
Exceptions:	<p>Updated - 1a – Webcam not working, face-auth server offline</p> <p>3b Server down/ http request failed or ill-configured database</p> <p>connection system notifies “Backend communication failure, unable to retrieve passwords”</p> <ol style="list-style-type: none"> 1. The System returns to normal flow 1
Includes:	<p>Password decryption done using private decryption key by system before displaying the password from the respective database with respective details.</p>
Extends:	<p>None</p>
Priority:	<p>High</p>
Special Conditions:	<p>Strong password should be added</p> <p>Updated – mandatory facial scan required</p>
Assumptions:	<p>User is properly authenticated and logged in and has added passwords into the system already</p>



Updated feature suite use-case diagram.

3. Requirement: Password Generator

Another major functional requirement adhering to this application is the password generator feature of the application with medium priority as it is for the user's added comfort. The feature is an added bonus to make the customer's life easier since it contains multiple password complication options for the customer to choose from to generate a strong complex and random password for their use, tailored to their specific requirements. The uniqueness and complexity of these passwords can be tested with the have I been pwned password distincter feature and the strength analyser to verify that the password is strong and complex in terms of global requirements and that it has not been breached before. The feature also gives the option to the customer to copy their generated passwords to the clipboard to be used anywhere ahead and includes general notification using toastify which is a vanilla JS toast notification library.

Description:	This use case allows the user to access and utilize the CyberVault feature suite of the application. Specifically password generation
Actors:	
Primary	User
Secondary	System
Triggers:	User selects password generation options; length, uppercase, lowercase character, numbers, special characters etc. and then clicks on the password generate button
Pre-conditions:	The user should be successfully logged into their respective profile and should choose conditions regarding the password they want to generate.
Post-Conditions:	A randomly strong password is generated tailored user requirements and is copied to the clipboard
Normal Flow:	<ol style="list-style-type: none"> 1. The user navigates to the password generator 2. The user individually selects components to make the password stronger or to suit their needs; component options include length, uppercase, lowercase character, numbers, and special characters. 3. Then the user clicks the password generate button 4. A randomly strong password is generated for the user; it can be copied to clipboard and used further.
Alternate Flows:	<p>2a User decides to either choose or not to choose either of the complication options for the password generation</p> <ol style="list-style-type: none"> 1. system resumes to normal flow 3
Exceptions:	1a Router failure, system prompts "invalid endpoint" or "Http

	Request failed” 1. The System returns to normal flow 1
Includes:	None
Extends:	None
Priority:	Medium
Special Conditions:	
Assumptions:	User is properly authenticated and logged in to their respective profile

4. Requirement: Password distinctior

Another major functional requirement adhering to this application is the password distinctior or password uniqueness analyser which makes use of the *troy Hunt's Have I been Powned API* which makes use of the react component for it to validate passwords through cross-referencing them with the troy hunt database of passwords through the API and determine how many times they have been breached and if they are or are not safe to use. This leads to an increase in security by notifying the user about the uniqueness and usability of their passwords before they are actually implemented. The security is further added on by also testing the password with the strength analyser as explained ahead. In the application the react front end takes in a password typed in by the customer and cross-references it with the Have I Been Powned password checker react component script connected with the API.

Description:	This use case allows the user to access and utilize the CyberVault feature suite of the application. Specifically Distinctior or uniqueness analyser
Actors:	
Primary	User
Secondary	System

Triggers:	User clicks submit after entering their chosen password into the analyser label
Pre-conditions:	The user should be successfully logged into their respective profile and should navigate to the distinctior endpoint.
Post-Conditions:	System returns with if the password has been breached or not and notifies the user if the password is safe to use.
Normal Flow:	<ol style="list-style-type: none"> 1. The user navigates to the password distinctior endpoint 2. The user enters the password to be checked into the label space of the endpoint 3. The user clicks the submit button 4. System cross-references the password with have I been pawnd database and returns with how many times the password has been breached if it has and whether it is safe to use or not.
Alternate Flows:	
Exceptions:	<p>1a Router failure, system prompts "invalid endpoint" or "Http Request failed"</p> <ol style="list-style-type: none"> 1. The System returns to Home page <p>4a Failed to import "invalid import requested" or "fetch api failed"</p> <ol style="list-style-type: none"> 1. System returns to normal flow 1
Includes:	This use Case includes cross-referencing the password entered by the user in the label by the Have I been pawnd (Troy hunt database)
Extends:	None
Priority:	Medium
Special Conditions:	
Assumptions:	User is properly authenticated and logged in to their respective profile

5. Requirement: Password strength analyser

Another major functional requirement adhering to this application is the password strength and complexity analyser which is a very important functional requirement since it determines the strength and complexity of a password to clearly indicate how strong the password would be against brute force cracking. It is done by utilizing zxcvbn library, which includes a strength estimation algorithm developed with inspiration from password cracking algorithms. It utilises pattern matching to recognize and compare the password with a wide variety of common passwords and common password patterns sequences to determine the complexity of a password is strong or not through a color coded bar representation when a password is entered. It is highly secure and allows greater flexibility in terms of password acceptance depending on their complexity by measuring all its complex components which are also included in the password generator.

Another functional optional requirement for the application includes the file locker capability which would allow users to upload documents for safe keep in this application thus being saved under a lock for security and ease of accessibility purposes. This is an optional functional requirement for now and depending on the completion of all other requirements and purposes of the application will be further developed depending on complexity and benefit and is also demonstrated through its own use case description ahead.

Description:	This use case allows the user to access and utilize the CyberVault feature suite of the application. Specifically Strength analyser
Actors: Primary Secondary	User System
Triggers:	User enters the password in the analyser label or while adding a new password in the system
Pre-conditions:	The user should be successfully logged into their respective profile and should navigate to the analyser endpoint or it is also shown while adding new passwords.
Post-Conditions:	System displays a bar of colors referring to the strength of the password the user is currently entering.
Normal Flow:	<ol style="list-style-type: none">1. The user navigates to the password strength analyser endpoint2. The user starts typing a password in the label section3. System displays a bar of colors referring to the strength of the password the user is currently entering from red being weak to green being strong and acceptable thus notifying the user.
Alternate Flows:	1a The user tries to add in a new password to the managerial database of passwords of the application and the strength analyser is used

	during the process 1. System returns to normal flow 2
Exceptions:	1a Router failure, system prompts “invalid endpoint” or “Http Request failed” 2. The System returns to Home page
Includes:	None
Extends:	None
Priority:	High
Special Conditions:	
Assumptions:	User is properly authenticated and logged in to their respective profile

6. Requirement: Facial registration (updated)

The user has to necessarily register their facial ID in the back-end flask server to view the password in the password manager.

Description:	This use case allows the user to access and utilize the CyberVault feature suite of the application. Specifically Face registration
Actors:	
Primary	User
Secondary	System (flask back-end)
Triggers:	User enters name and clicks register face
Pre-conditions:	The user must be authenticated and logged in
Post-Conditions:	Application notifies user their face has been registered.

Normal Flow:	<ol style="list-style-type: none"> 1. The user navigates to the face registration endpoint 2. The user types in their name (user-account name) 3. User clicks on the register face button
Alternate Flows:	
Exceptions:	<ol style="list-style-type: none"> 3a. The facial scanner server might be offline 1. The application shows notification server offline
Includes:	None
Extends:	The user can redirect to the manager since a link is shown in the dynamic message after successful face registration
Priority:	High
Special Conditions:	
Assumptions:	User is properly authenticated and logged in to their respective profile

File locker requirement (Optional):

Description:	This optional use case (optional implementation in the application) allows the user to store files under lock when the file vault is accessed by the user by uploading documents
Actors:	
Primary	User
Secondary	System
Triggers:	User uploads a file
Pre-conditions:	The user should be successfully logged into their respective profile and should navigate to the file vault and enter their pin to access it.

Post-Conditions:	System gives access to a file vault holding important documents and lets user upload a file (limited size: images etc.) to it.
Normal Flow:	<ol style="list-style-type: none"> 1. The user navigates to the file locker 2. The user starts clicks the upload button and chooses a file 3. System saves file to database and redirect to home page thus locking the vault again
Alternate Flows:	
Exceptions:	<p>1a Router failure, system prompts “invalid endpoint” or “Http Request failed”</p> <ol style="list-style-type: none"> 1. The System returns to Home page
Includes:	None
Extends:	None
Priority:	Medium
Special Conditions:	
Assumptions:	User is properly authenticated and logged in to their respective profile

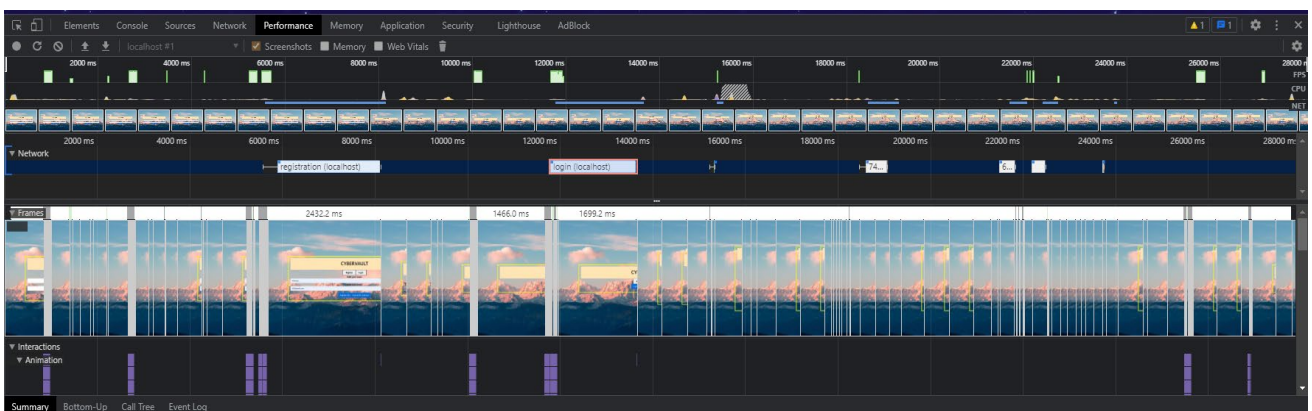
Non-Functional Requirements analysis

This section of the report measure the important non-functional requirements mentioned in the report against the current prototype and plans regarding the future development.

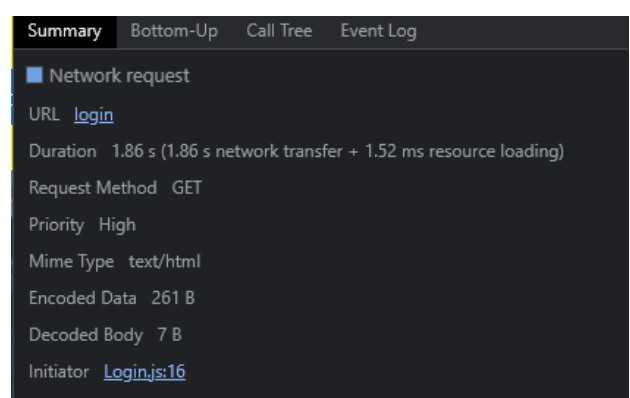
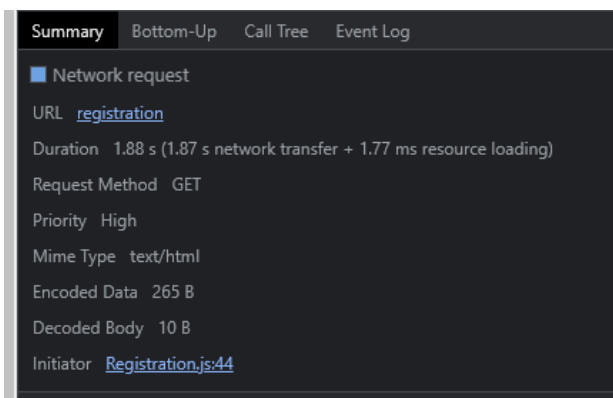
1. Security – One of the most important non-functional requirements which this application should be in accordance with is security which includes the security regarding customer login and registration credentials, security regarding passwords the application saves for all customers and security around all the additional features the application provides to avoid leaks and exploitation. For the future development security requirements are to be satisfied to avoid any exploitation through input fields such has SQL injections or XSS cross side scripting etc., through input sanitation, secure design principle development and design

pattern implementation. The current application prototype also includes security measures to satisfy the security requirements at a basic stage these are indicated by the additional feature of password distinctior not being accessible until successful login and login is disallowed until the user is properly registered into the system with their face-Ids and credentials.

2. Performance efficiency – The performance efficiency of the current prototype is measured to observe and conclude that the prototype is performing its functions with minimum possible response times and required resources and without putting too much load on the processing thus giving the desired output to the customer within a good time frame and performing all its required functions. Optimum performance directly relates to the future capabilities of an application which includes network load times, API request complete times, overall execution duration concluding as overall performance analysis of the application. For the future of this application the performance efficiency is considered as



one of the major requirements to follow through for maintaining application performance. For the current prototype the application performance has been measured and analysed using chrome web developer tools which can be observed by the screenshots provided below:



Network throttling 150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)

The above images represent the current performance requirements that are above average including network through put, CPU throttling, request and response duration, CPU usage etc.

3. **Data integrity:** Data integrity is also a major non-functional requirement that ties in with the applications features and processes since it is utilising a lot of customer oriented data including their credential, facial encodings for recognition and authentication, passwords saved, passwords used with application feature suite functionalities etc. regarding future development I plan to encrypt these the credentials for security and data integrity maintenance purposes. Even the additional features the application provides would include security implementations to avoid any data breaches or exploitation. The password distinctor utilises the Have I Been pwned API which utilises the React component script for Troy Hunt database which actually checks passwords when they are encoded so they don't even reveal the actual passwords being entered to avoid exploitation.
4. **Reliability:** For the current prototype the reliability as a non-functional requirement can also be measured since it is visible through the images above of how the request and communication between the front-end and the respective servers is actually completed and all the functions mentioned are successfully performed with respective incoming response etc. thus receiving the desired outputs and performing the required function at the same time.
5. **Scalability:** Scalability is a very important non-functional requirement according to which the application is being built, it directly ties to how the application would perform under changing or heavier loads of information with less available resources for data processing and that it would be able to perform its necessary function with changing environments and resource conditions. This also ties with the current prototype since the application is usable in other browsers and the prototype also performs its functions with as many users as required while recognising each user every time they login thus working under increased load.
6. **Usability:** Usability as a functional requirement is important for an application to be built according to since it concludes that an application is usable by any and every kind of customer with no added complexity. The customer does not have to figure out how the application performs the functions that it's meant to perform; they can simply do it without prior training. The current prototype also demonstrates it since it includes simple procedures for performing the mentioned function with all complexities being taken care of in the back-end and thus being readily usable by and for any customer.

Project Plan and Analysis

This section of the report contains an updated plan of tasks from the proposal with a partially updated Gantt chart and screenshots from planned tasks, milestones and updates from Trello:

Board SoftwareProject Final Software project Private Invite

ToDo

- Building the password manager vault UI as a component for the React UI
Jan 17, 2022 - Jan 19, 2022 1
0% Jan 19
- Coding password vault back-end logic
Jan 20, 2022 - Jan 25, 2022 1
0% Jan 25
- Implementing and integrating back-end logic with respective database (SQL or Fauna)
Jan 26, 2022 - Jan 31, 2022 1
0% Jan 31
- Coding in encryption and decryption logic in respective to be used models
Feb 1, 2022 - Feb 4, 2022 1
0% Feb 4
- Completing credential vault logic
Feb 7, 2022 - Feb 8, 2022 1
0% Feb 8
- Finalising the UI and endpoints regarding vault functionality with the UI
Feb 9, 2022 - Feb 9, 2022 1

In-Progress

- Proper secure alternate login registration setup
Dec 28 - Jan 3, 2022 1 10%
Jan 3
- Multifactor research and integration into login-registration
Jan 4, 2022 - Jan 5, 2022 1
0% Jan 5
- Building a basic UI for the main application to include different sections of the application according to functional requirements
Jan 4, 2022 - Jan 7, 2022 1
0% Jan 7
- Setting up feature suite back-end (Node. express)
Jan 10, 2022 - Jan 12, 2022 1
0% Jan 12
- Integrating necessary databases (SQL Mongo, Fauna)
Jan 12, 2022 - Jan 14, 2022 1
0% Jan 14

Unit testing

- Face_recognition library integration with flask server
Nov 24 - Nov 25 1 100%
Nov 25
- Registration-login setup only for facial recognition
Dec 8 - Dec 8 1 100%
Dec 8
- Unit testing the API and distinctor feature logic by running the react server on local host
Dec 22 - Dec 22 1 0%
Dec 22

Completed

- Research Wireframes and Design Ideas For UI design and structure
Nov 13 - Nov 14 1 100%
Nov 14
- Setup front-end basic react app
Nov 14 - Nov 15 1 100%
Nov 15
- Python back-end setup
Nov 18 - Nov 19 1 100%
Nov 19
- Flask server creation
Nov 19 - Nov 19 1 100%
Nov 19
- Importing necessary flask dependencies in the back-end
Nov 19 - Nov 19 1 100%
Nov 19
- Integrating flask server to work with react app
Nov 22 - Nov 22 1 100%
Nov 22
- Setting up front-end for flask server back representation

Board SoftwareProject Final Software project Private Invite

ToDo

- trying to generate multiple strong passwords and later on checking with the distinctor and strength analyser
Feb 28, 2022 - Mar 1, 2022 1
0% Mar 1
- Building the password strength analyser userfriendly UI (strength meter etc) section in the designated section of the mainUI
Mar 2, 2022 - Mar 7, 2022 1
0% Mar 7
- Developing the logic code behind testing the strengths of passwords with building strength meters in the front - end
Mar 8, 2022 - Mar 15, 2022 1
0% Mar 15
- Checking for and building the notificationability of the strength analyser
Mar 16, 2022 - Mar 18, 2022 1
0% Mar 18
- Unit testing the password strength analyser with multiple password types and passwords generated from the user tailored password generator
Mar 21, 2022 - Mar 22, 2022 1

In-Progress

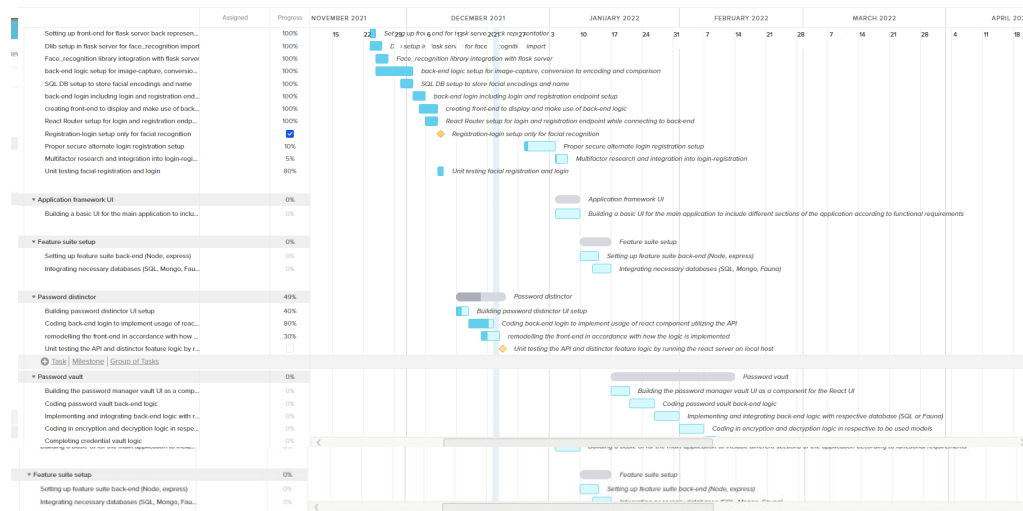
- Proper secure alternate login registration setup
Dec 28 - Jan 3, 2022 1 10%
Jan 3
- Multifactor research and integration into login-registration
Jan 4, 2022 - Jan 5, 2022 1
0% Jan 5
- Building a basic UI for the main application to include different sections of the application according to functional requirements
Jan 4, 2022 - Jan 7, 2022 1
0% Jan 7
- Setting up feature suite back-end (Node. express)
Jan 10, 2022 - Jan 12, 2022 1
0% Jan 12
- Integrating necessary databases (SQL Mongo, Fauna)
Jan 12, 2022 - Jan 14, 2022 1
0% Jan 14

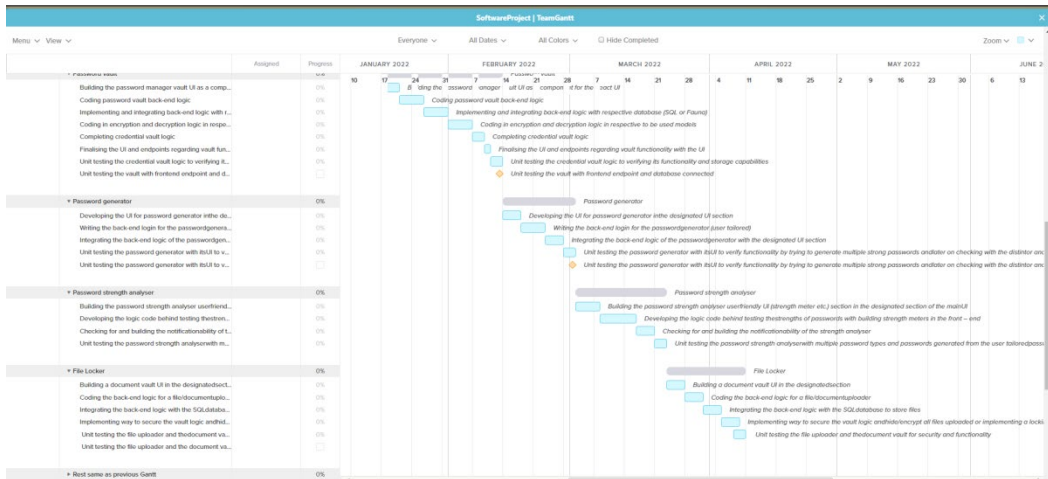
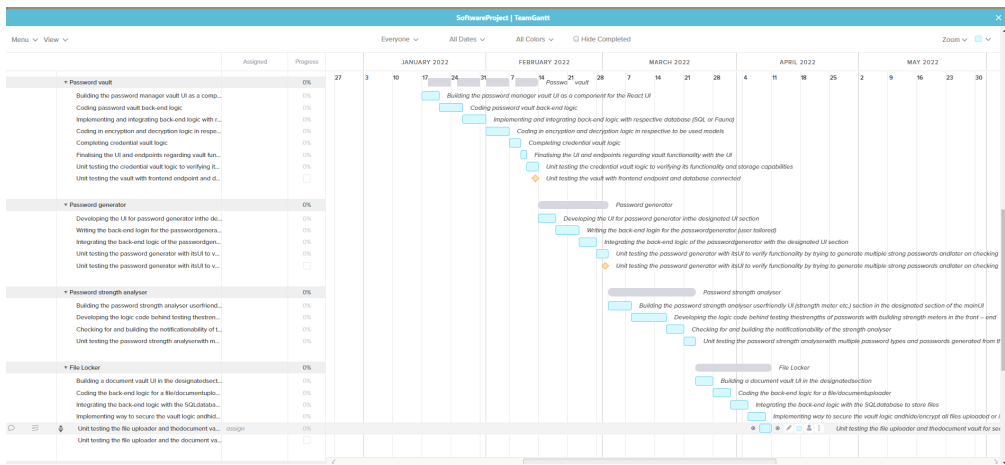
Unit testing

- Face_recognition library integration with flask server
Nov 24 - Nov 25 1 100%
Nov 25
- Registration-login setup only for facial recognition
Dec 8 - Dec 8 1 100%
Dec 8
- Unit testing the API and distinctor feature logic by running the react server on local host
Dec 22 - Dec 22 1 0%
Dec 22

Completed

- face_recognition import
Nov 23 - Nov 24 1 100%
Nov 24
- back-end logic setup for image-capture, conversion to encoding and comparison
Nov 24 - Dec 1 1 100%
Dec 1
- back-end login including login and registration endpoint setup
Dec 2 - Dec 3 1 100%
Dec 3
- creating front-end to display and make use of back-end logic
Dec 3 - Dec 7 1 100%
Dec 7
- React Router setup for login and registration endpoint while connecting to back-end
Dec 6 - Dec 7 1 100%
Dec 7
- SQL DB setup to store facial encodings and name
Nov 30 - Dec 1 1 100%
Dec 1





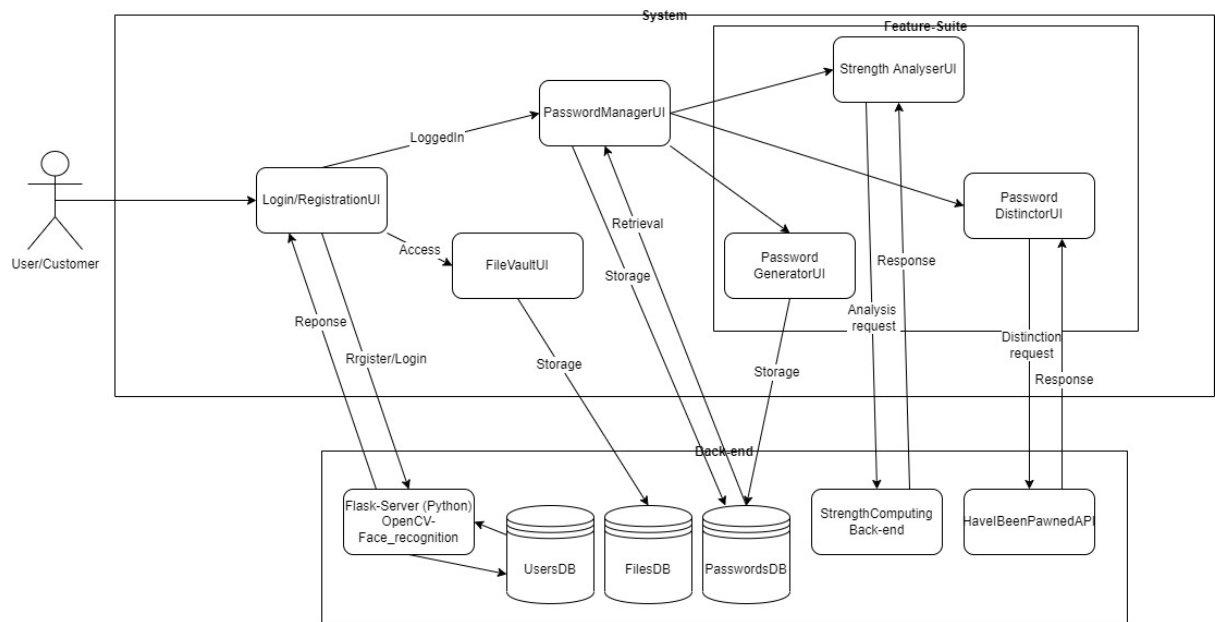
A list of the partial altered task list from the original task list mentioned in project proposal, the tasks in the trello task plan includes:

1. Research Wireframes and Design Ideas For UI design and structure
2. Setup front-end basic react app
3. Python back-end setup
4. Flask server creation
5. Importing necessary flask dependencies in the back-end
6. Integrating flask server to work with react app
7. Setting up front-end for flask server back representation
8. Dlib setup in flask server for face_recognition import
9. Face_recognition library integration with flask server
10. back-end logic setup for image-capture, conversion to encoding and comparison
11. SQL DB setup to store facial encodings and name
12. back-end login including login and registration endpoint setup
13. creating front-end to display and make use of back-end logic
14. React Router setup for login and registration endpoint while connecting to back-end
15. Registration-login setup only for facial recognition
16. Proper secure alternate login registration setup

17. Multifactor research and integration into login-registration
18. Unit testing facial registration and login
19. After this continued on the same task list as the original

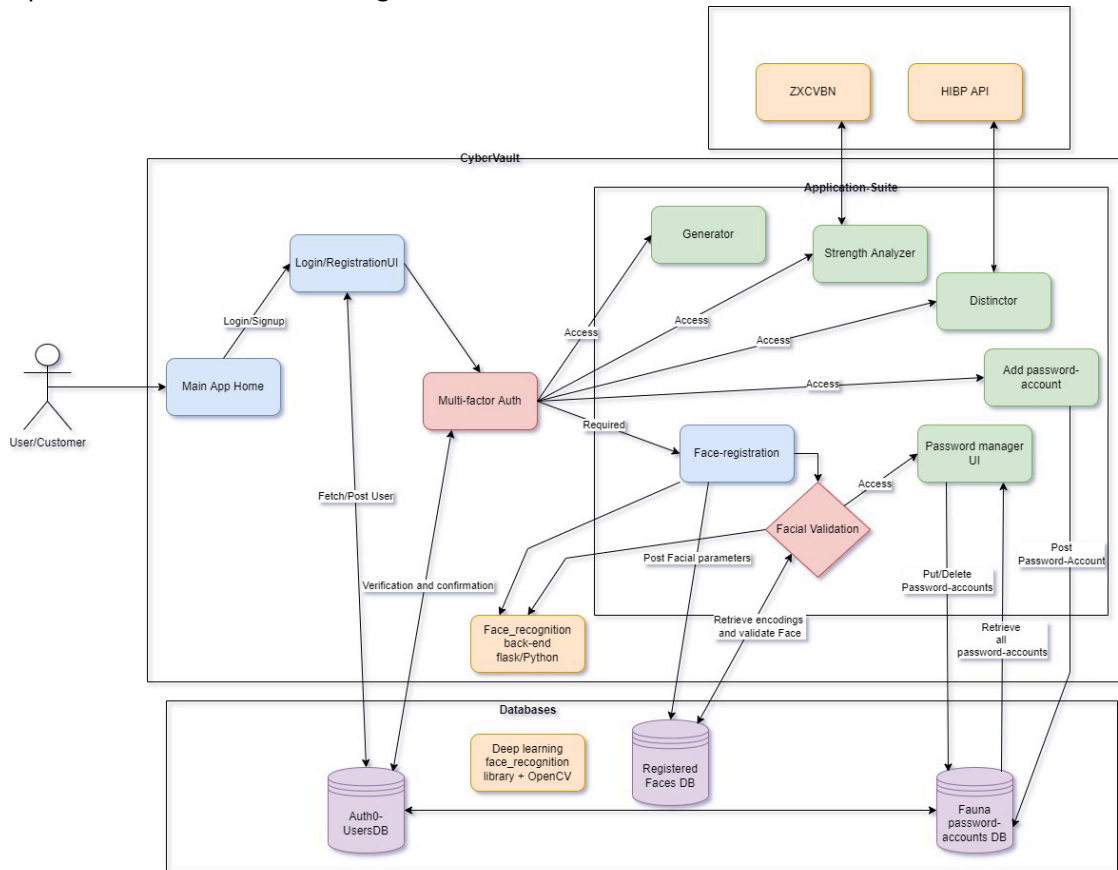
These tasks represent the difference of tasks detail and the order in which certain tasks were performed from the original task plan in the beginning stages and are hence indicated in the updated Gantt chart as well as the task list on trello. The tasks after these tasks are in the same order as the original task list for now until further development. All changes and implementations are briefly explained in the implementation section of the report as well.

Design and Architecture



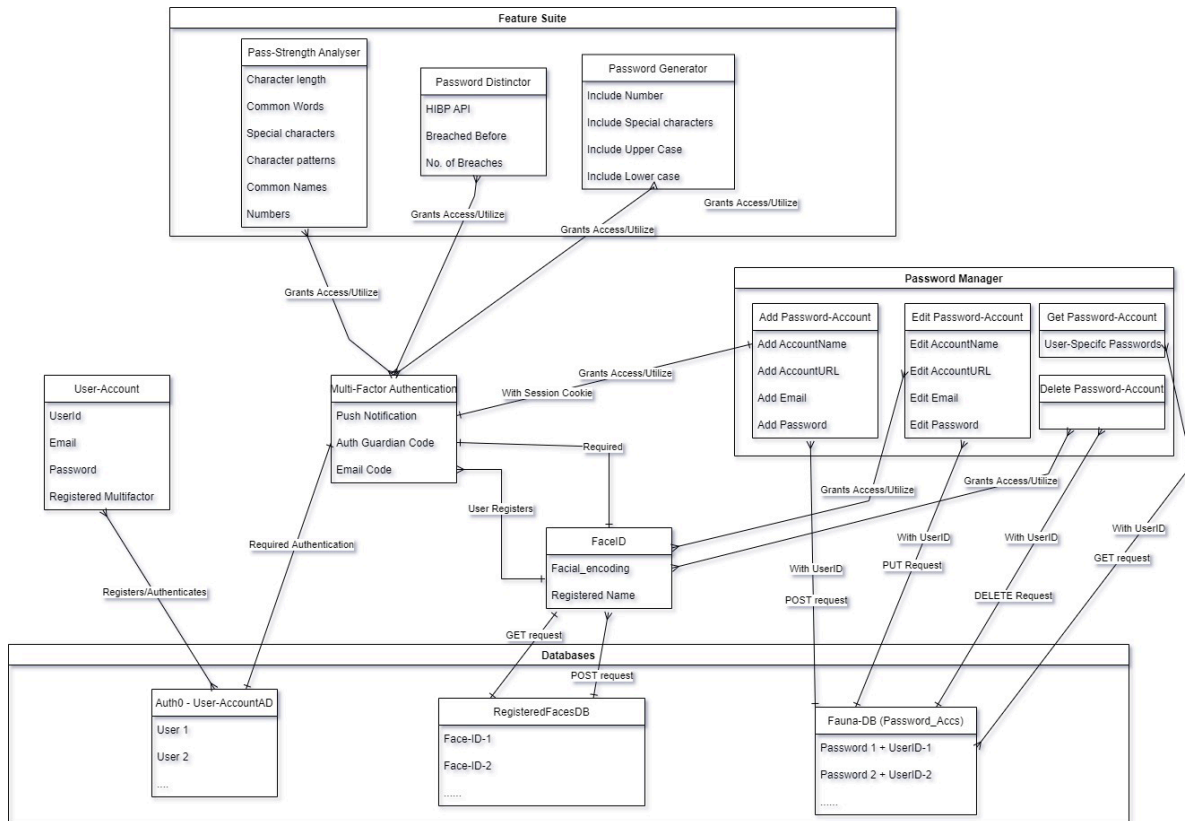
The above image represents the current planned system architecture of the application. The architecture diagram refers to the procedure of a customer interacting with the system depending on what the customer would like to achieve and access regarding what the application provides. Everything follows after the customer is successfully able to login to the application this includes passing through the facial recognition authenticator and providing the right access details for the multifactor authentication which are referenced with the user database and the face recognition python flask backend server then further being able to access the password management UI where the user would be able to manage their passwords which would include adding, removing, altering and viewing all the passwords all these operations include cross referencing with the passwords DB that will be used to save all passwords and their related credentials. After this the user also has access to the feature suite of the application which includes the password generator, the password distictor and the password strength analyser functional features. These entire features suite refers to their respective back-end logics for successful operation and to provide the customer with the desired results regarding the operation they choose to perform. The diagram also represents all the relations between the components of the system architecture; these also include how passwords can be saved from the generator to the database etc. The diagram clearly represents all operational requests and concerning responses from where the requests are being sent which could be a back-end server or back-end system logic algorithms.

Updated Final Architecture Diagram –



The Final architecture diagram describes the flow of the application and includes a display of all requests that are made to and from databases, libraries and API's and the authentication levels that need to be passed to access certain functionality. It also demonstrates the feature suite of the application and access control that has been implemented between them. All the changes including Database updates security framework implementation, integration of facial authentication for the password manager have all been shown in the diagram with the correct flow which an everyday end-user would follow.

ERD



Implementation

This section of the report marks my progress to date and slight changes I made from the original plan of implementation.

My current implementation involves a Login registration panel which uses the customer's face encodings for authentication. For this I first created a back-end flask server in python to store the logic for the face_recognition. The back-end server implements face_recognition library as an import which is a part of the dlib library. Dlib library is originally an open source suite of applications written in C++ which offer a massive range of functionality regarding machine learning including image and live stream processing tools, dlib contains state of the art face recognition libraries built with deep learning which is why I decided to use it. But to import and install the face recognition library I first had to install the dlib library which was not a simple process since it required the right version and had to be individually compiled using visual studio's CMake tools instead of an easy pip installation, then further face recognition imports were installed. After I had all the necessary imports which included os import (used for making python script interact with the operating system in use, used by me to store recognized and unrecognized images in the folders on my local system), sql connector import which was used to connect to the SQL database to store the user credentials and facial encodings, then the Open CV import which was used for performing image processing tasks (capturing images, resizing frames etc.), numPy for mathematical computations required in the script and finally the flask and flask CORS import required for setting up the flask server. After this I wrote the functions for retrieving all the data from the database and functions for the registration and login endpoints. The registration endpoint function when called in the front-end takes an image

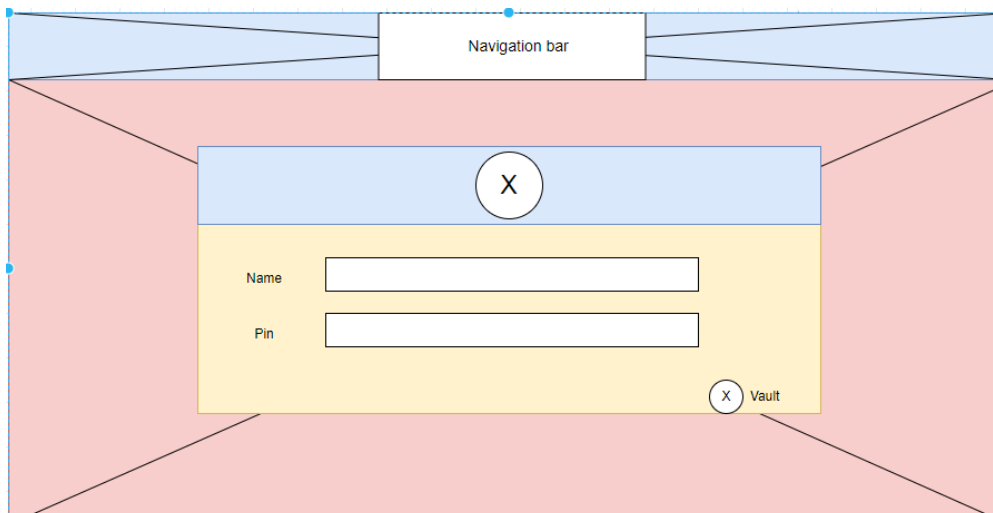
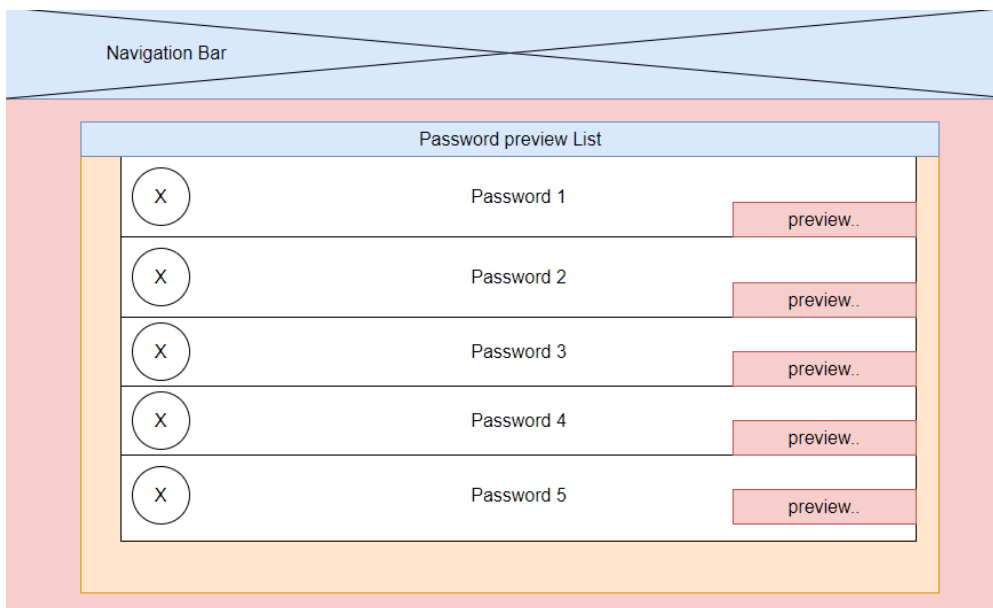
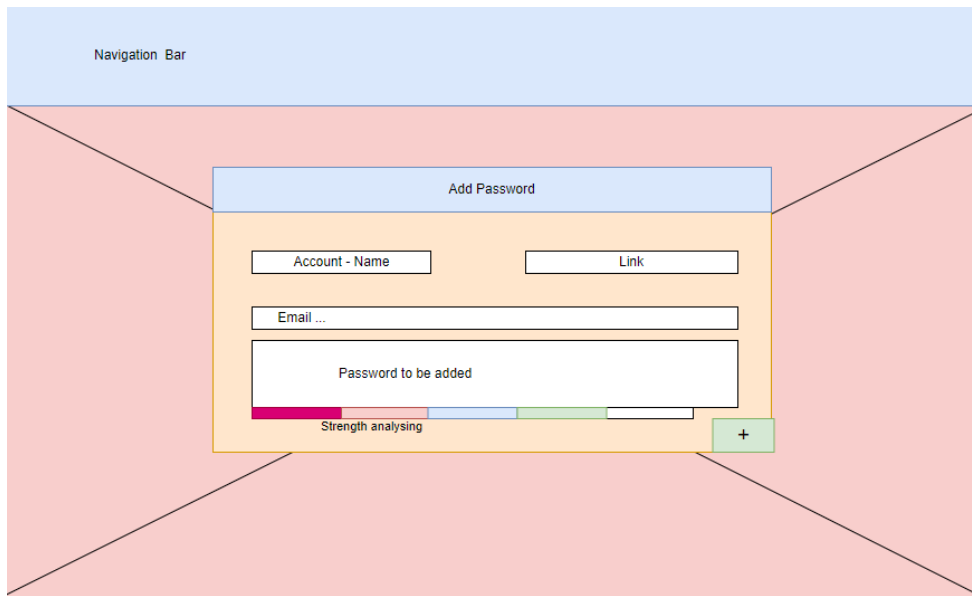
of the customer through the webcam as well as the customer's credentials entered, then the face-recognition library is used to recognize and separate the face from the image taken through the webcam and then retrieve the facial encodings (converted to 128 bit vectors) from the image by passing the face through the deep neural net embedded in this library which computes the vectors thus quantifying the facial landmarks that were recognized which are then saved with the credential into the respective SQL database, the image capture is also saved into a folder on the local machine into the identified folder with the name of the customer as the name of the respective image folder and finally a string registered as the final output to denote that the customer has been registered is returned. Similarly the login endpoint function the database is checked first if it is empty, it means there are no registered customers and shows a message saying not register in the system please register first, then this function is mainly responsible for clicking a picture of the user through the webcam get its encodings. The function then receives all encoding from the database and loops through all these encodings and tries to compare the encodings with the encoding of the current image taken, if a match is found then the function denotes that specific encoding with the name for the encoding that was saved in the database thus recognising the user and returns this name as the output of the function. Then I created a react front-end app to be connected to this back-end flask server and through routing made the login and registration class components which then connect to the flask server with their own respective registration and login functions which call upon these endpoint functions in the back-end thus setting up a prototype registration and login page.

Further I created a class for implementing the Have been pwned API to create the distinctior feature of the app which tells how many times a password has been breached and whether it is safe to use or not. For this I import the react-HIBP Password checker from the React Have I been Pwned component and use it on the front-end to check passwords that the customers enter regarding how many times they are breached since they are cross referenced with the Troy Hunt database and return the output thus declaring whether the password entered by the customer is safe or breached and if breached how many times thus creating a prototype of the distinctior feature. I have done this by creating a react class component in the frontend to display the output that is returned from the API once the typed in password is cross referenced with the troy hunt database. The front end has an input space where the user can enter a password to be checked and the output according to the API returns the amount of times the given password is breached by the cross-referencing of results. This feature can only be accessed after a successful login which can only be done when a user is successfully registered in the system database.

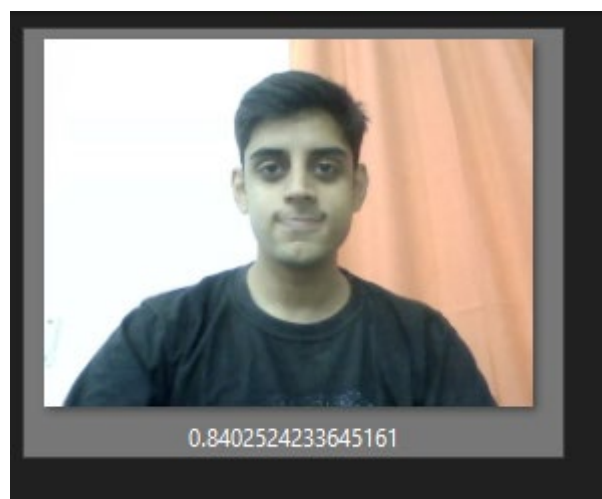
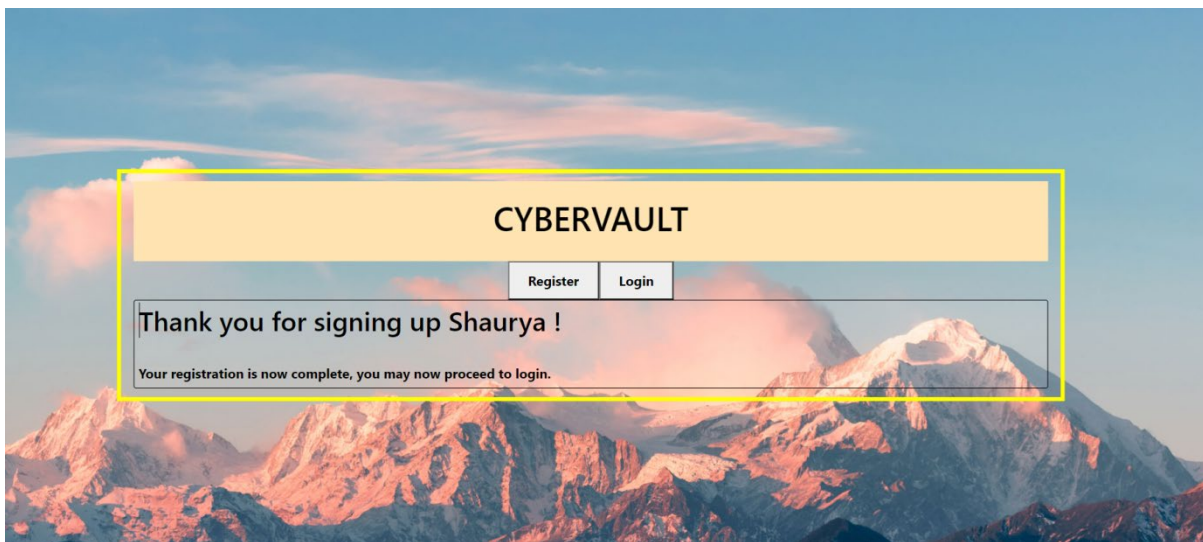
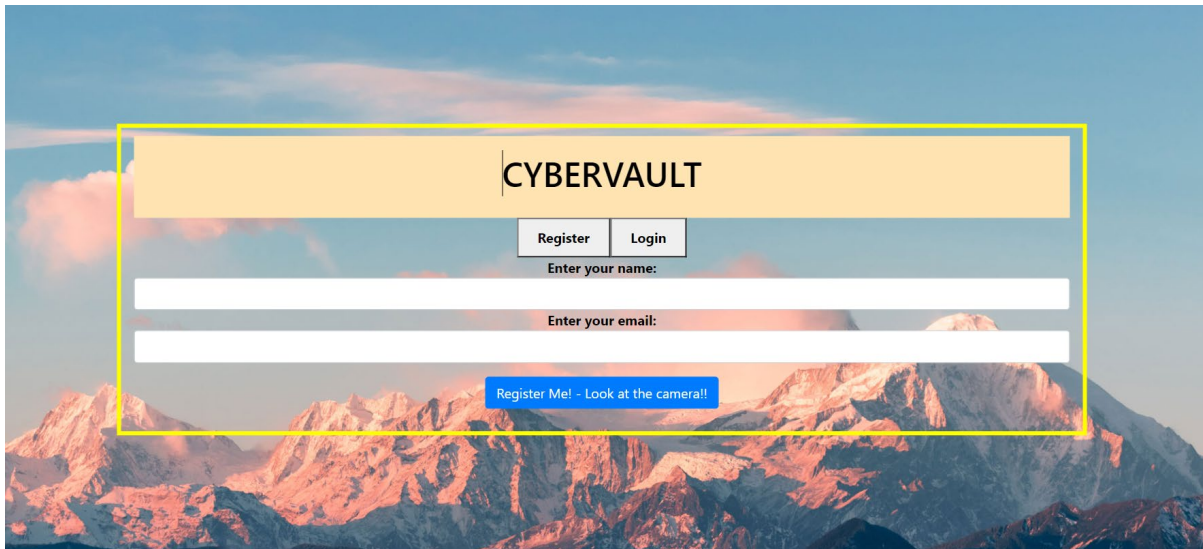
Initially I had planned on using tensor flow and a default deep learning keras model from keras face-models to implement the face recognition login and registration. However, as my research and implementation progressed regarding this functional requirement, I decided to take a different approach to implementing this feature because the previous approach's constraints were unclear. I decided to use the face recognition extension of the OpenCV extension for python in my back-end, which is also stored in a flask server and performed the respective functional requirement, but through a slightly different process and with better defined constraints and clearly defined attributes.

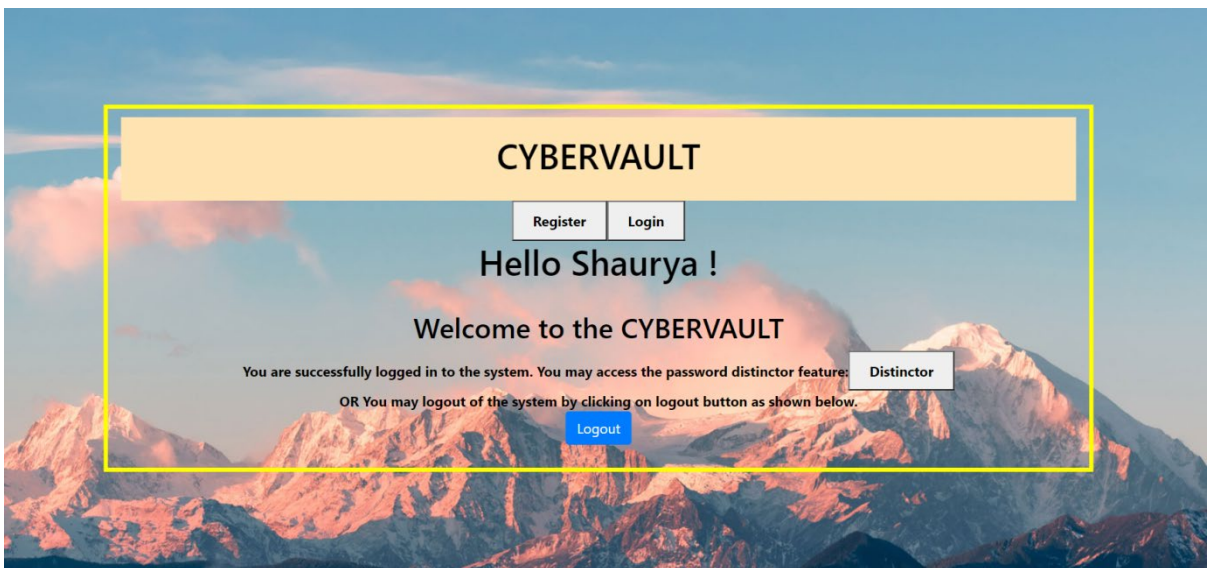
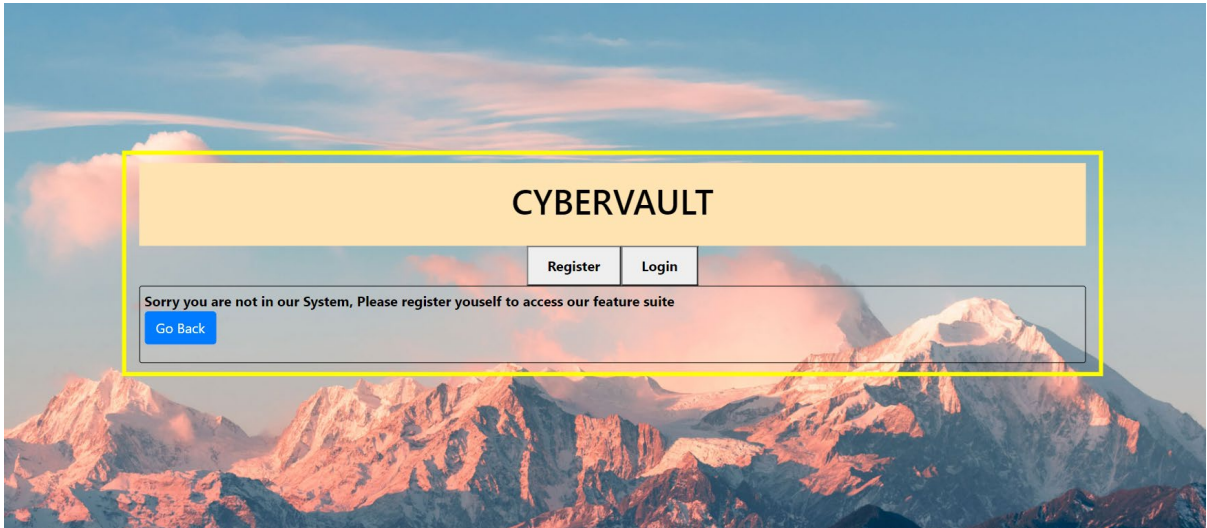
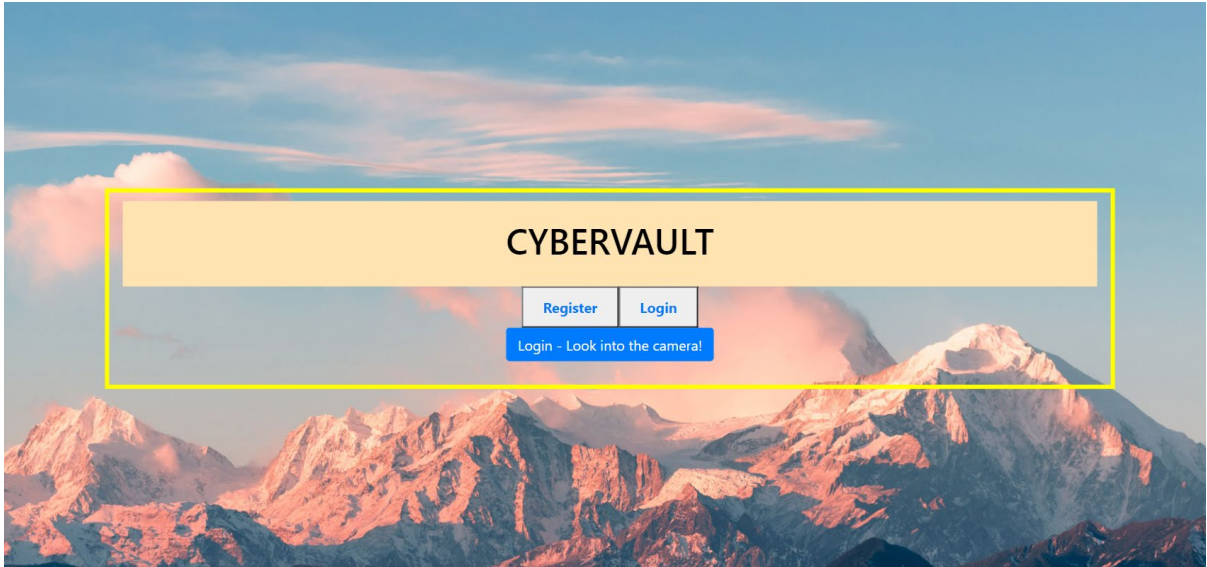
Graphical User Interface

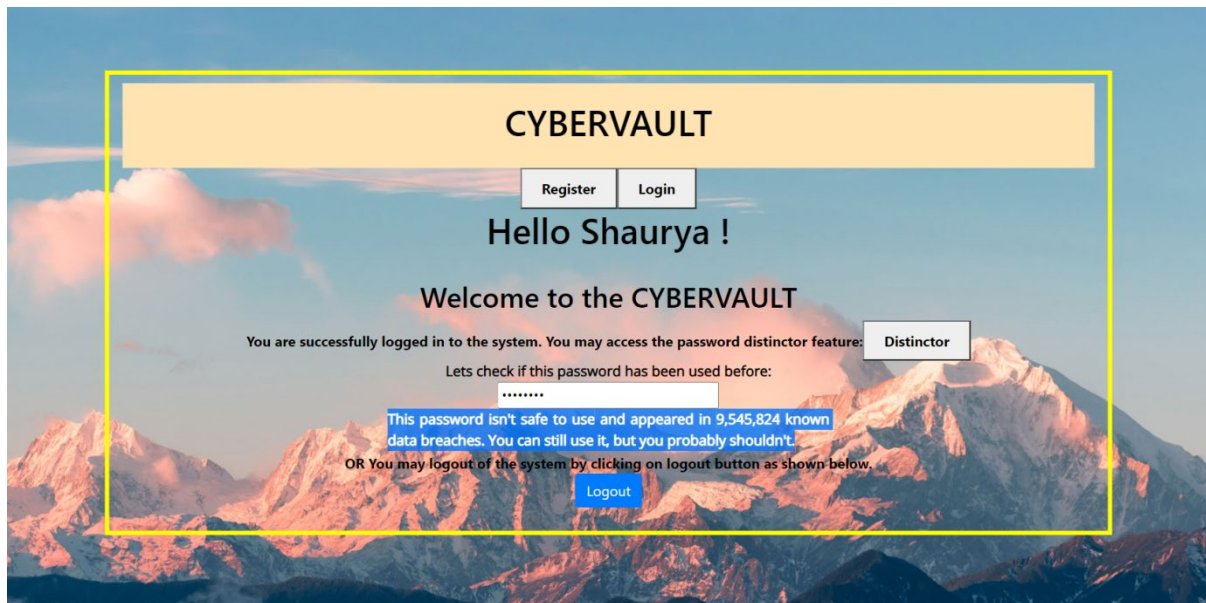
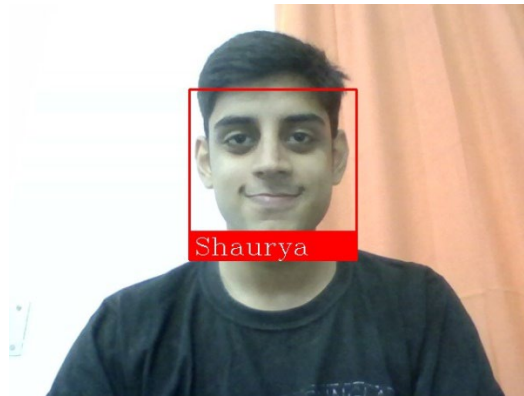
Sample wireframes for the project to be developed further include:



Some screen shots of the current prototype include:







Final Technology, requirements and Implementation

Final Technology stack

Further explained is the final stack of all technologies including modules and libraries that were used to develop the CyberVault web-application. These technologies include:

Back-End: The backend of the application was built to handle and process facial authentication and validation and used the technologies as follows.

- **Python:** Python was used as the main language for back-end flask server development where the deep metric learning is implemented for facial registration and recognition + Authentication; it is further used in the application to validate users to view their passwords. Python was used since it is much more beneficial to be used for Machine-Learning, AI and deep learning driven projects such as CyberVault. This is due to its consistency, simplicity, platform independence and its large variety of access to strong and efficient machine

learning and AI modules, libraries and frameworks (Such as *OpenCV*, *dlib* & *face_recognition*).

- **Flask:** Flask was used as the main back-end micro-framework web application server that is provided in python. It is an efficient and useful back-end framework which comes in use when a JavaScript based front-end is developed such as Next.js and React.js in this application. Since the connection between a flask back-end and JavaScript based Front-end is very smooth and fast. In This application it was connected with the React.js and Next.js based front-end using CORS for developing the API to share Facial Images which were then processed in the python back-end to register a face and to validate the face when accessing the passwords.
- **Flask_cors:** Flask CORS was used to configure the python flask back-end server to work with the Next.js & React.js front-end through Cross Origin Resource sharing i.e., allowing the sharing and use of resources from and between different domains for example in this project resources are being shared between *localhost:3000* server and the flask *127.0.0.1:5000* server (facial images and naming data).
- **OpenCV:** *OpenCV* or *cv2* is a very popular python library/module which is used for live video and image (Webcam) processing and analysis which include face **detection** in a video or image and image capture and editing through the webcam. *OpenCV* library/module uses machine learning and AI algorithms to detect faces in video/images. In this project *OpenCV* is used for capturing the video image through the webcam after detecting a face, then resizing the frames according to the face and finally naming the captures (both on the canvas as well as the name of the image file) during facial registration and validation.
- **Dlib:** *dlib* is a very important python library/module which is required for installing the main *face_recognition* (based on *dlib*) library/module. The *dlib* library/module contains the deep metric learning algorithms that are utilised through the *facial_recognition* module to construct and process the facial embedding/encodings which are used for the recognition process for the faces. Its installation for windows systems was complicated since it required individual compilation using visual studio and *Cmake* tool as it was originally a C++ library/module.
- **Sqlite3 + DB-Browser(SQLite):** The database technology used in the back-end to store user specific facial data required for facial registration and validation is *Sqlite3*. The *sqlite3* library/module in python searches for a database file and creates one in case it is not already present. This database file is then used as the primary database to store all facial encodings along with the user names which are then fetched and processed for validation. To manipulate the database files created by *sqlite3* (.db files) the DB-Browser application for *SQLite* is used.
- **Face_recognition:** The main library/module used in the python flask back-end server is the *face_recognition* library/module which is a *dlib* and deep learning based face recognition library/module. It can be used for detection and recognition of faces and facial features/landmarks through a webcam. In this project it is used to first detect the face locations from the image captured through the webcam, then compute the facial encodings of the face that is detected. Finally the *face-compare* and *face-distance* methods from the library/module are used to compare and validate a face (through its encoding values; 128 bit vectors) and the neural net keeps getting trained accordingly to validate the face faster every time it is scanned.

- **NumPy:** *NumPy* is a universal numerical open source library/module in python popularly used for working with numerical data. In this project *NumPy* was used to generate numerical names for the image captures during facial registration and validation. It is also used in the comparison of encodings when returning minimum indices from the face distance array.
- **os library/module:** the *os* library/module in python is mainly used to interact with the folders present on the OS of the machine/system. In this project it is used to store the facial images when registered (with registration names for folder names) and during validation (with identified names on image) in folders present inside the project folder.

Front-End: The front-end was used to develop the client side of the application i.e., the main Web-application GUI to be used by the user (also including the Next.js server rendering)

- **Next.js:** Initially as mentioned in the proposal and mid-point documentation I mentioned using React.js for the front end along with Express server on the back-end connected to the flask server. I decided to replace the implementation of another back-end server (express) and a separate front-end using React with simply using the Next.js i.e., React based framework which does the job of express.js since it lets me code API endpoints (acting like middleware communication) in the API folder and then use the pages folder to create React components (with Next.js tweaks) which can fetch and send data to these API's using technologies like *axios*, thus combining both their individual advantageous aspects. All functional components and their function code was also written and stored in a components folder within the project folder.
- **React.js** (Purely frontend functional components and design): Since Next.js is a React.js based framework for the front-end with its own tweaks. All the front end pages were created using React **functional** components which also integrated the components (code files) holding Modal popup and logic code and code for all functional requirements etc. (In the components folder).
- **React Hooks** (*useState()*, *useUser(Auth0)*, *useEffect()*, *useRef()*): One of the most important aspects of React.js that were used in the Next.js application were the React hooks including *useState()*, *useRef()*(for coding logic using components that referred to user action), *useEffect()*(for rendering logic as the page renders) and *useUser()*. These hooks are one of the most efficient capabilities of React.js that were used in this application to manage states of variables as they were dynamically updated i.e., managing the stateful logic of all functional components. Special hooks that work in tandem with third party application like *useUser()* were used to manage and track user sessions. Hooks were only usable with React.js functional components.
- **Auth0 + auth0-Guardian App** – Auth0 was used to setup the secure login and registration (including social logins (Google and Facebook)) for CyberVault including multifactor authentication either through the auth0 Guardian mobile application (using push notification or one-time codes) or through one-time codes sent through email to valid email addresses used. Why and how Auth0 was used is explained further in the final functional requirement section.
- **Tippy.js:** *Tippy* is a popular tool tip, pop over, drop down solution Java script library/module that was used in this application to show useful information to the end-user using tool tips i.e., information icons which show information when hovered over with the mouse cursor.

This is additional information that helps the user either understand functionality or directs them on what is to be done to access it or make the most of it.

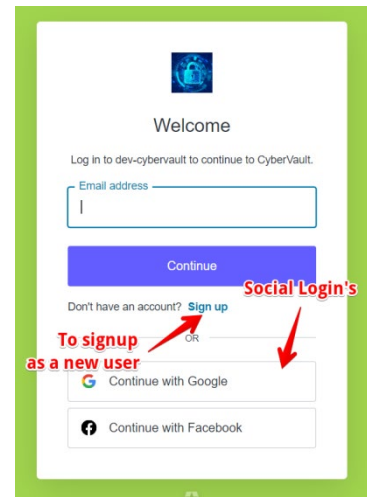
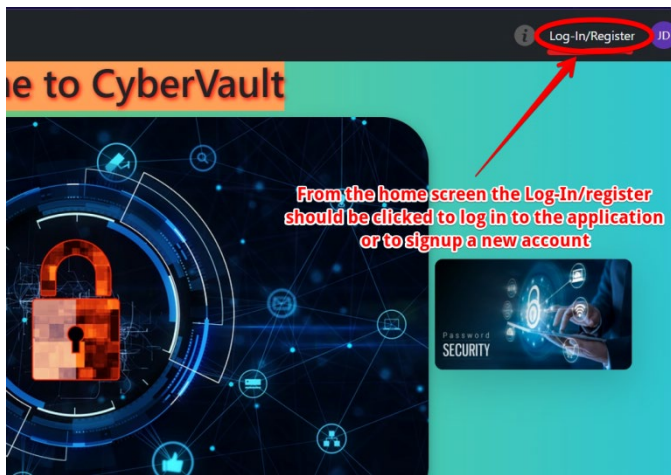
- **Axios:** *Axios* is a popular library/module that has been used in this application to send HTTP requests to the API endpoint middleware of Next.js. *Axios* would also have been used to make these endpoint requests in case another back-end JavaScript based framework such as express would have been used. Its main function is communication with different API endpoints. In this application *axios* sends requests (either POST payloads or GET data for user) from the react front-end components to the API endpoints defined in the API folder of the application which further connect to the database (Fauna DB) and send and receive data from it (including User session specific data).
- **Crypto module:** As initially mentioned in the proposal the Crypto.js library/module was going to be used for password encryption and decryption instead the Node.js inbuilt *Crypto* module has been utilised to encrypt (before they are sent to the DB) and decrypt user passwords being stored. The crypto module has been used with the strongest encryption algorithm i.e., AES-256 bit encryption using specially generated random 40 bit keys (salt) stored as environment variables which are used to encrypt and decrypt the data.
- **Bootstrap + React-bootstrap:** Bootstrap and React-Bootstrap library components were used to beautify the React/Next front-end pages; bootstrap components like – navbar, nav, Modals, forms and form-controls, buttons, rows, columns containers etc.
- **Font-awesome-icons:** The font-awesome library was used to integrate different kinds of icons in the application like the copy-to-clipboard, view or mask icons etc.
- **CSS:** Next.js built in CSS paged (Global Module and Home Module) were altered to suit and include different CSS styling to different components of all the different functionality pages and some styling for all pages in common like background and containers etc.
- **React-have-I-been pawned:** The React-Have-I-Been-Pawned library which connects and pulls data from the Have-I-Been-Pawned API from the Troy-Hunt has been utilised. More specifically the *HIBPPasswordChecker* component has been integrated with a react functional component to develop a page to detect and analyse the uniqueness of different passwords and whether they have been breached before with the number of times they have been. The functional component utilising the API also shows dynamic messages of whether the password is safe to use or not.
- **React-idle-timer:** The React Idle timer component has been utilised in a React functional page component to implement an Idle-timer which automatically logs a user out after 3 minutes of inactivity on the page. The Idle-Timer component scans for mouse movement on the page, including scroll, cursor movement or any clicking etc. to determine whether the user has been idle.
- **React-toastify:** The React *Toastify* library/module has been used to implement notifications in the application which show up to display different messages depending on the situation for a very short period of time, such as “password copied to clipboard”, “password added” etc.
- **React-webcam:** The react-webcam component is used to display the live webcam footage on the screen so the user can look at the camera during facial registration or validation. It has also been extended to show facial landmarks as a part of future development explained ahead.

- **Yup:** Yup is a very important library which has been utilised in the application for assisting in server side data validation in the API folder, the application already involves strict input validation, but using the Yup validation schemas server side (Next.js API); data validation has also been conducted in accordance with defence in depth principle.
- **ZXCVBN:** The ZXCVBN library has been utilised for strict and accurate password strength estimation with dynamic and colour coded progress bar and a colour coded dynamic message that depends on the strength estimation result. The ZXCVBN is a strength and entropy estimator for passwords which was developed on the basis of password cracking algorithms, It involves searching for all different kinds of patterns in a password to estimate its strength including common English words or names, movie names, common dates, very commonly used passwords, common character sequence patterns including keyboard patterns or repeats etc. to give an accurate strength estimation.
- **Fauna-DB:** The main database technology used for the whole application including storing passwords and all other password related credentials is the Fauna Database which is a document relational and distributed database API. It provides amazing scalability and flexibility to all applications while also assisting as a server-less database. It comes built in with a modern security infrastructure and can be used as an OLTP database with ACID transactions that are distributed. It acts as a stateless API (native to cloud) with built in security.
- **Cypress** (cypress real events, cypress next.js auth0 and cypress testing library/module): Cypress has been used as the main testing framework for testing this application. The Cypress JavaScript testing framework has been used to perform automated end to end tests, integration tests and unit tests for all individual functionality and whole pages along the way which it does in an artificial environment as can be seen in the testing section ahead.
- **ESLint:** ESLint has been used as the main static analysis tool for the application which shows potential coding mistakes, bad practises or bugs in the code when run in the command line. Fixing these bugs and potential changes leads to best practises being used.
- **GitHub:** Github has been used as the main version control system for this application to track the code upload and changes. It is also used to submit the source code for the application.

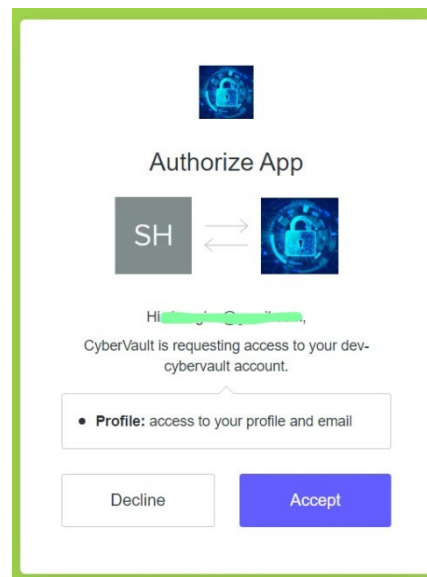
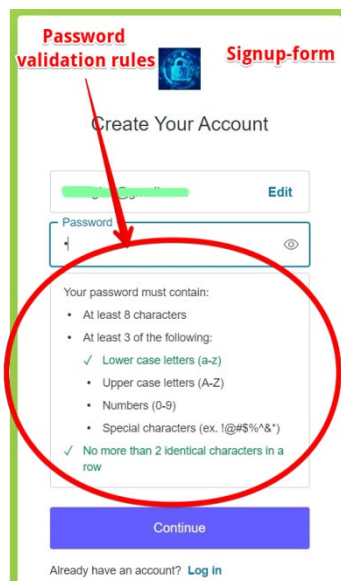
Final Functional requirement completion

With reference to all the functional requirements that were mentioned during the midpoint and the project proposal; further is a comprehensive list of all functional requirements completed with the optional requirements being explained in the future development section along with other future prospect ideas and implementation. Current final functional requirements are as follows -

1. **Secure Login/Registration with multifactor authentication:** A secure login and signup/registration has been developed in the application with the help of Auth0. Initially to login or register the link on the main screen should be clicked as can be seen below. The login also offers an option to sign-in through social accounts which include the options for a Google or Facebook account login. The social login works when the end-user selects a social account; a request is sent to the social network provider (redirected to social login) and once that is completed and the user identification is confirmed, they are logged in after they allow/deny access to their profile information.



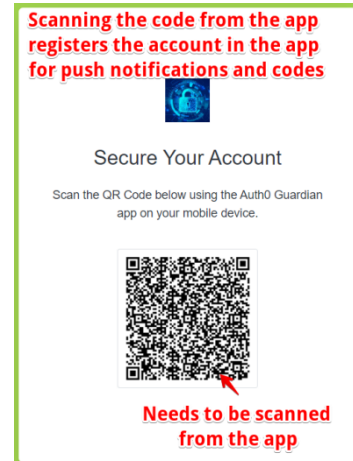
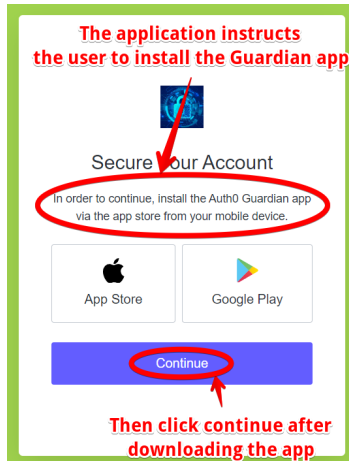
- a. There is sign up option that allows the user to register for the first time using an email and a password which must abide by the rules that are created for a strong secure password to be used. Rules including; No more than 2 identical characters, at least 8 characters long, should include special character, a lowercase, upper-case and a number and lastly cannot be submitted empty with no password. After the user signs up, their password is hashed and salted using the bcrypt algorithm before the user information is stored in the database and each user also gets an additional UID or User-ID which is unique to them. Auth0 additionally ensures the data is being encrypted using AES 128 bit encryption during transit (using TLS) as well. The sign-up form can be observed below.



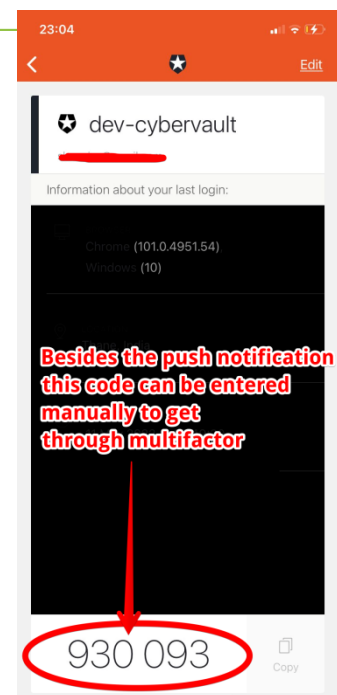
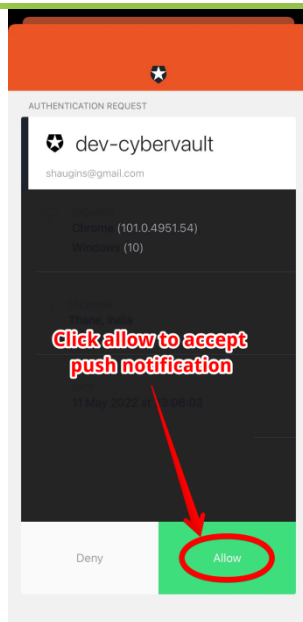
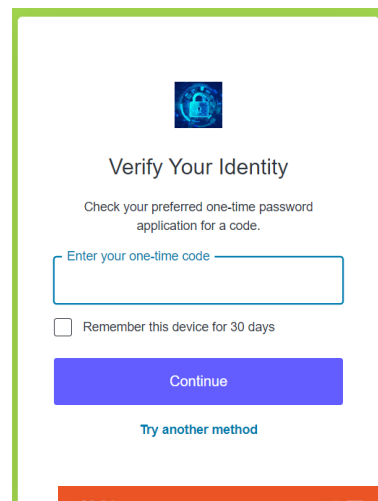
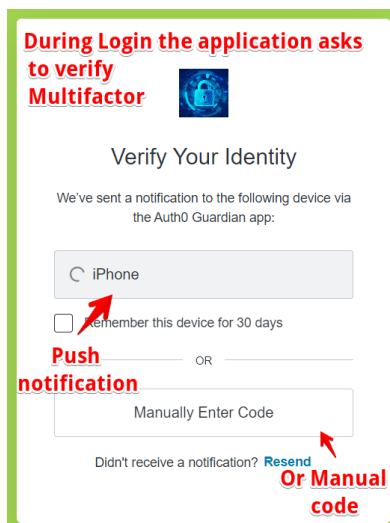
- b. Further Multifactor authentication has also been implemented with this login/registration. When The user tries to login again for the first time after they signed up or during the signup process, the user is prompted to download the Auth0 Guardian mobile application (on their mobile device) and are then shown a QR code which needs to be scanned to attach that account to the guardian app. Then multifactor authentication is activated, to log in the user, who either needs to click 'allow' on the push notification on their guardian app or manually enter the code generated by the app, in case the user does not have access to their device at the

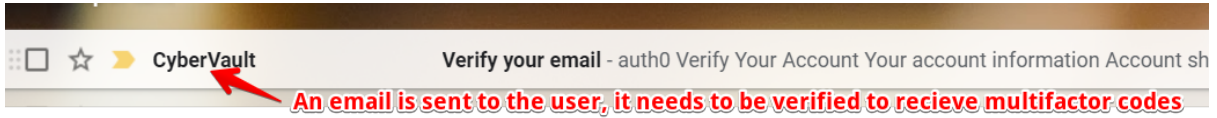
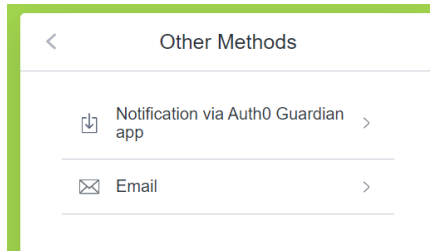
time they can also request for the code to be sent to their email and can use the code from there to successfully log in to the application. Additionally suspicious IP throttling has also been implemented which identifies when multiple failed login attempts for an account are detected from a single IP address, in such a case the login is blocked and an email is sent to the rightful user regarding attempt of forced login.

During the sign up process -

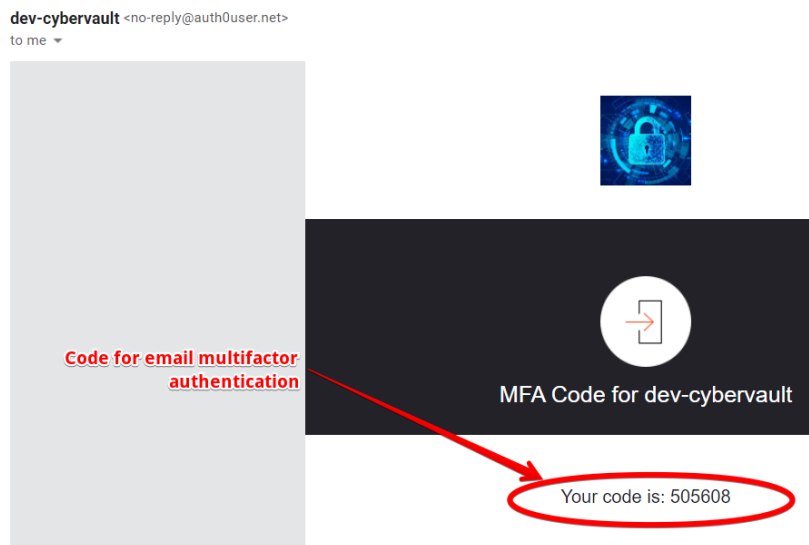
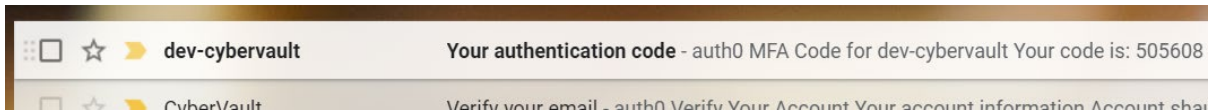
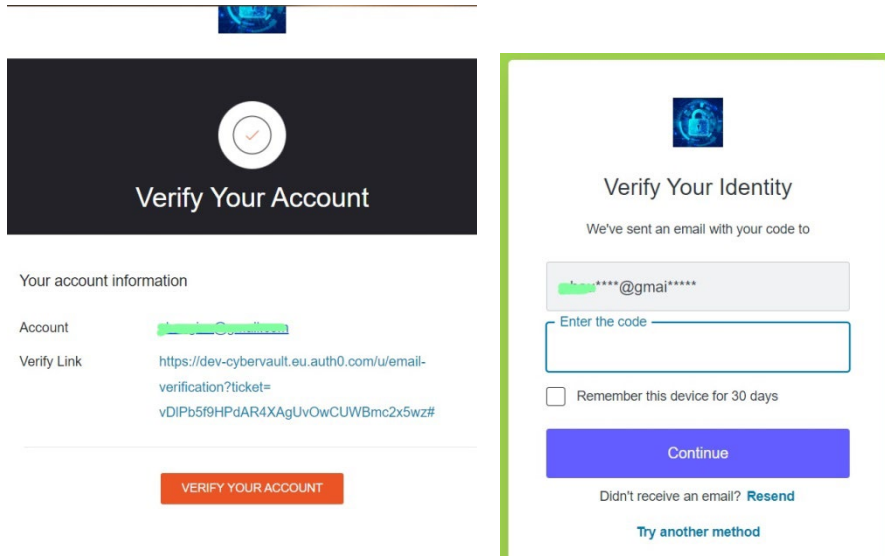


Then During Login -

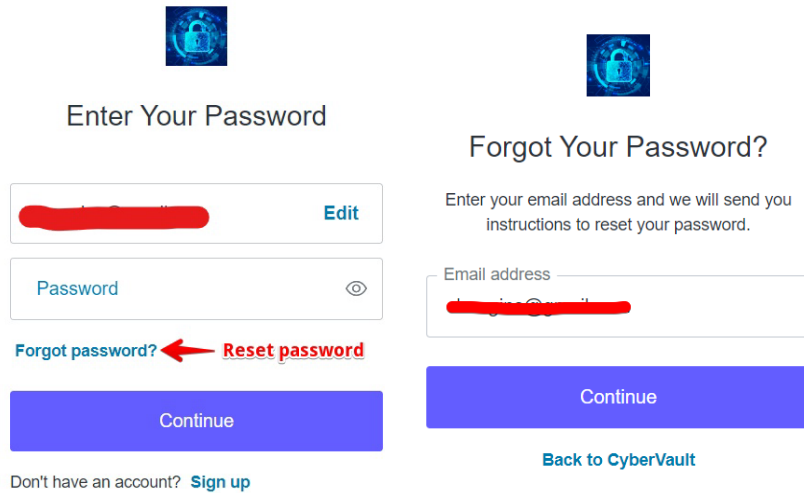




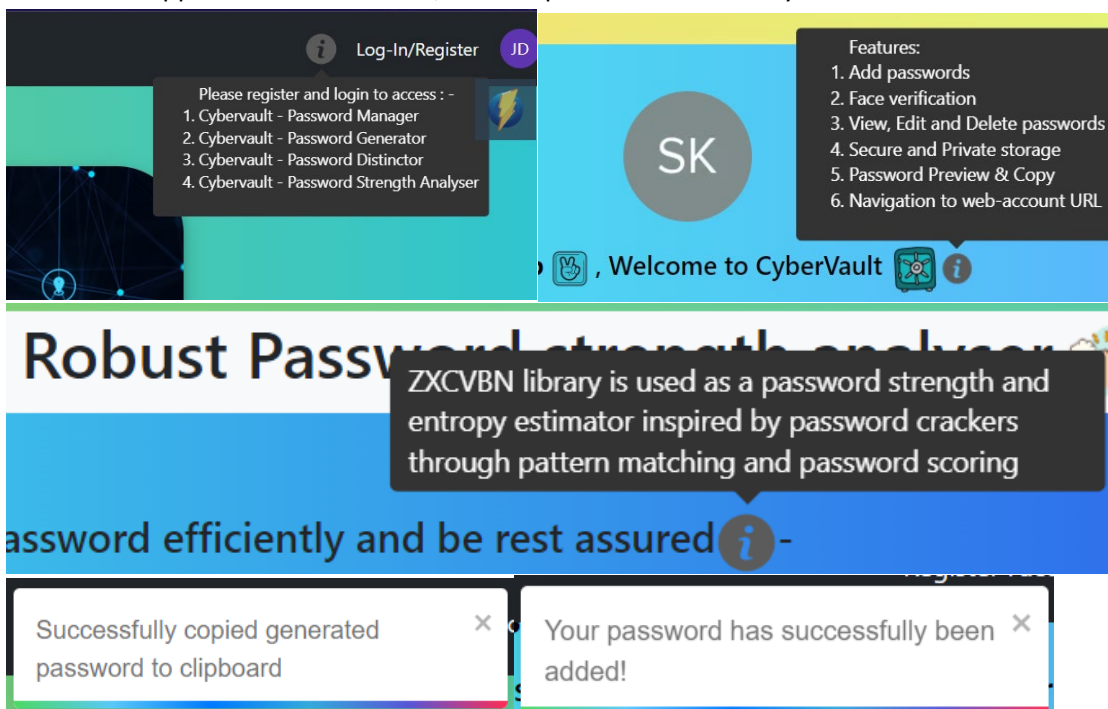
Multifactor can also be done through a code sent through the email, for that the email needs to be verified first.

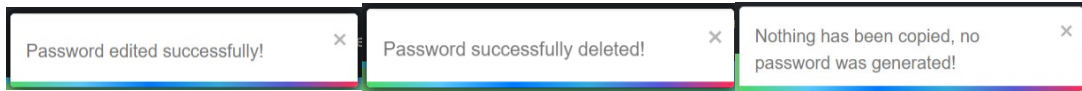


- c. Finally the login registration process also includes the function to reset a password for an account which when clicked after entering the email sends an email to the user with the link which they can follow to reset their password.

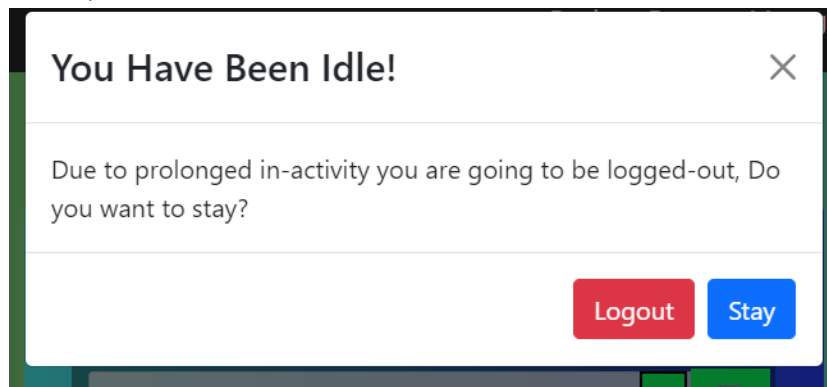


2. **Notifications & Info-tips/Tool-tips:** Some additional functional components included in the application were to improve UX/UI i.e., improve the user experience of the web application and beautify the User Interface, this was done using Notification which were developed using Toastify and were made to be shown on the screen for a very short period of time when a function was accessed and completed such as; copying a password to the clipboard, adding a password to the account, editing a password, deleting a password from the account, generating a password etc. Another additional functional requirement was the use of tool tips or info-tips that were implemented using Tippy.js, these show additional useful information to the user as the hover their mouse over the tool tip. This information could be about the function (how it works what it does) the user is utilising or could be about what the user is supposed to do to access/utilise a piece of functionality.

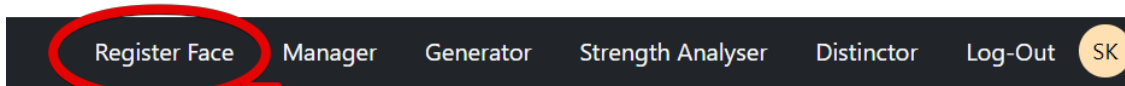




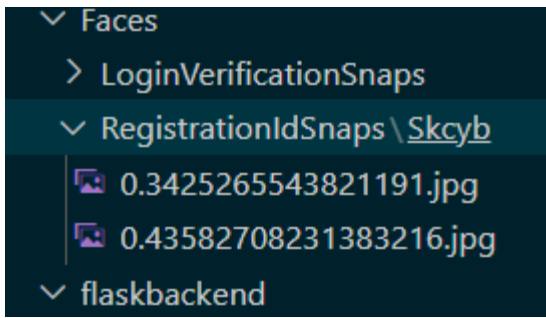
- Idle Timer:** Idle timer was also built as a functional component for the application due to a focus on security as explained in the security feature section. This function basically monitors the whole application after the user logs in and scans the application for user activity such as mouse movement, scrolling, clicking etc. In the event it discovers no movement from the user for 3 minutes straight i.e., the user being Idle it shows a modal popup warning asking the user whether they want to stay or logout. Another 3 minute timer starts when the modal popup is displayed and waits for the user's action. Even after that if the user does not click on the stay button the application automatically logs the user out of the application to prevent misuse.



- Facial registration with flask and python backend** (with front-end and server side data validation): Another functional requirement from the initial documentation that has been implemented in the application is the facial registration and validation. Once the user passes the multifactor authentication and is able to log-in to the application they can access all other features except the password manager itself until they validate their facial encodings. The user is able to utilise all other features i.e., they can generate random passwords according to personal preference, they can check the strength of a password or its uniqueness (hacked/breached before or not), they can even add a password. But the user cannot view or preview their passwords i.e., the manager itself. This includes the inability to edit or delete the passwords until the user has validated their facial ID. Immediately after the login the user can register their facial ID or face encodings to their accounts by viewing the face registration page and registering their facial ID, once this has been completed, the user can then visit the manager page and validate their existing facial ID to preview the passwords and further move on to view, edit or delete them. In case the face does not match or does not exist in the back-end facial database the page would dynamically show a message regarding this with an info tool tip suggesting to register their face first with a redirection link to the page.



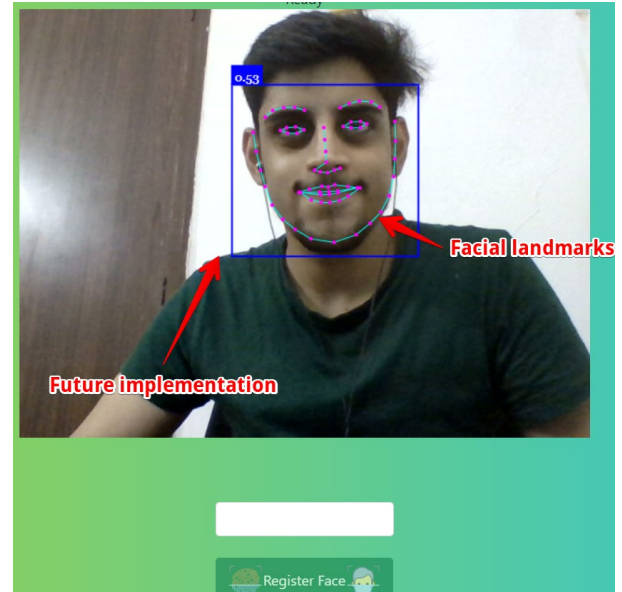
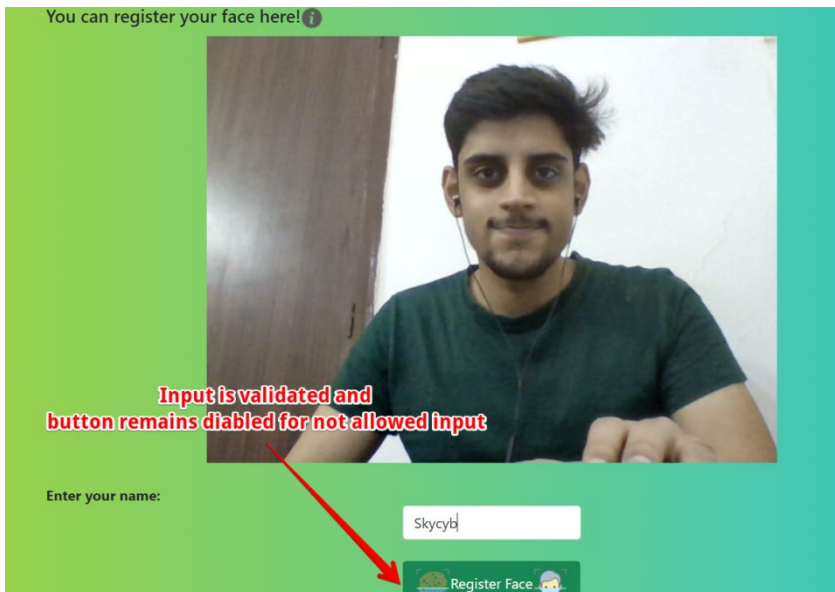
Face registration page link in the Navbar



registrationName	encodedVal
Filter	Filter
Skcyb	[-0.04195816 0.02718065 0.032298...



Saved in the DB



5. **Face authenticated password manager:** The password manager is the main functional component described for the project. Contrary to the proposal it has been built with different technologies from what was mentioned before. Initially it was declared that the manager would be built using react, SQL and express, instead the password manager has now been built with Next.js + React.js for the front-end and with Fauna as the main database system also including the facial validation which is required to access the manager itself.
 - a. To view the manager; a user is successfully supposed to log in to the application after passing the multifactor authentication; doing so will create a session for the user using an Http only session-cookie containing the information which is also used to track the session and record changes to it. This cookie is also used to recognize and fetch all the passwords for a specific user from the fauna Database. Once the user is logged in they can navigate to the manager dashboard page where they are

Face not found , Register your face if you haven't done so yet! 

 Retry!

Password-Accounts are displayed after face is validated

Add Password-Account 

9 passwords present and face verified, List of your passwords and accounts are as follows:




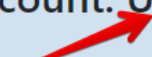
Social/Business/Personal Account : Udemy




Social/Business/Personal Account : Youtube

- b. In the table only the account and email for which the password has been saved are visible with a dynamically updating icon for the account (for example YouTube icon for YouTube account-password). These rows can then be clicked which pops up a password preview modal where the password along with the email is saved. The password there is character-masked by default but can be unmasked by clicking the eye icon and viewed. The password from this preview component can also be copied to the clipboard and the website or account for which the password has been saved can be navigated to by clicking the account name which also acts as a link for the website whose account has been added (this link is made from the account URL entered in the add password component earlier).


Account: Udemy 


Can redirect to page 



Inputs here are disabled 

Email added for Udemy account:
skCybe@gmail.com

Password added for Udemy account:
Ydemy@!123

Unmask button 

Copy Button 

- c. For adding an account-password a user needs to click the add password button and enter all relevant details such as; account name, its URL (Like URL of YouTube or GitHub etc.), an email associated with the account that is being added and finally the password. There are a few strict input validation conditions that are required to be passed to successfully add a password (same conditions exist for editing the password as well). Input validation has been conducted for each relevant detail that has been asked in the popup add password modal form for example; the account name should be a normal string (no special characters or SQL keywords) the email should be valid (checked against email regex), the URL should valid (checked against URL regex) and finally the password conditions necessary for a strong password including; 8 character length, necessary upper and lower case, a number and a special character. A strength meter has also been provided below the password input box which demonstrates the strength of the password being typed in dynamically through colour coding and a dynamic message. Until all validation conditions are passed the create button to add the password remains disabled. This data for the password-account is also validated server side (Next.js API's) so needs to account for all validation conditions. In case the user is not sure what password they would want to use, the add password modal also contains a redirection link to the password generator which opens in a new tab as to not disturb the current session or flow the user was in while adding a new password-account.

The screenshot shows a modal form titled 'Social/Business/Personal Account and its URL:'. It contains three main input sections: 1. Account details: 'Account name' (input: 'NCIRL') and 'URL' (input: 'ncirl.ie'). Below these are three green validation bullet points: 'Account normal String', 'Account No special chars', and 'Url should be valid'. 2. Email: 'Email used for mentioned account:' (input: 'x18138284@student'). Below it is a red validation bullet point: 'Email should be valid'. 3. Password: 'Respective Password to be used:' (input: masked with dots). Below it is a blue link 'Generate' and the text 'Or go to Generate a new password which you can paste here!'. At the bottom, there is a 'Strength meter' with a green bar and the text 'Strong enough'. Below the bar are four green validation bullet points: 'Should be greater than 8 characters', 'Should contain an upper case letter', 'Should contain a lower case letter', 'Should contain a number', and 'Should contain a special character'. At the very bottom are two buttons: 'Close' (red) and 'Create' (green). Annotations in red text with arrows point to various elements: 'Input validation conditions are mentioned' points to the account and email validation lists; 'Can go to generator' points to the 'Generate' link; 'Disabled until input validation is passed' points to the 'Create' button; and 'Strength meter' points to the strength bar and its associated validation list.

Social/Business/Personal Account and its URL:

NCIRL ncirl.ie

- Account normal String
- Account No special chars
- Url should be valid

Email used for mentioned account:

x18138284@student

- Email should be valid

Respective Password to be used:

.....

Or go to [Generate](#) a new password which you can paste here!

Strong enough

- Should be greater than 8 characters
- Should contain an upper case letter
- Should contain a lower case letter
- Should contain a number
- Should contain a special character

Close Create

Input validation conditions are mentioned

Can go to generator

Disabled until input validation is passed

Strength meter

- d. After the face validation when the user opens the password preview modal popup they can also choose to either edit the relevant account-password details by clicking the edit button or delete the account-password entirely. On clicking the edit button the edit password modal pops up which has the same input validation conditions from the add password modal along with the disabled edit button (which only enables after all input validation conditions are passed).

Udemy udemy.com

- Account normal String
- Account No special chars
- Url should be valid

skCybe@gmail.com

- Email should be valid

.....

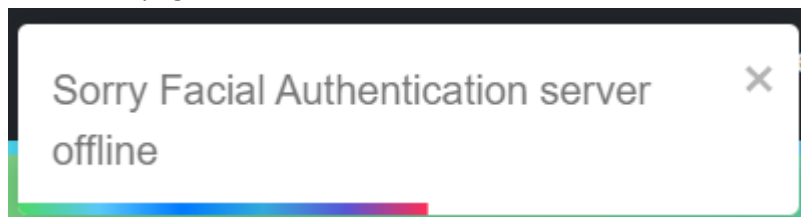
Better

- Should be greater than 8 chars
- Should contain an upper case letter
- Should contain a lower case letter
- Should contain a number
- Should contain a special character

Close Edit

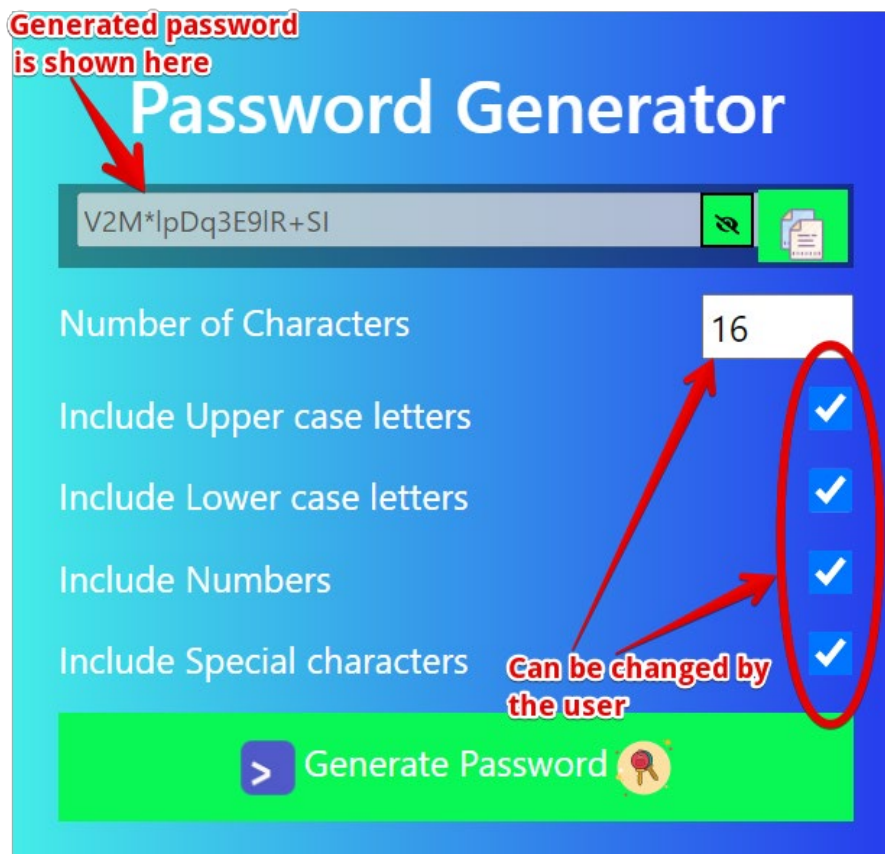
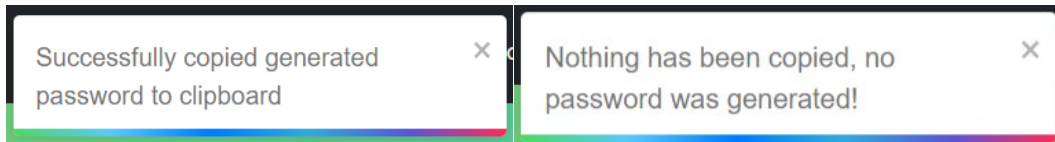
Only enables when validation is passed

- e. Finally notifications are a part all functionalities i.e., a notification will be shown on the screen as a password is added, edited or deleted. Notifications are used in some cases to handle errors, which in this case is done if the front-end is unable to connect to the facial authentication server (flask back-end) when the either the register face button (register face page) or the validate/verify face button is clicked (manager dashboard page).



6. **Customer tailored complex Password generator:** Another one of the main functional components completed in this application is the strong and complex user tailored password generator. The generator function offers multiple options to the user at the front-end to choose what kind of a password would the user want generated, these options include whether the password should include an upper case letter, a lower case letter, a number and a special character. These can all be either checked or unchecked by the user as per their desire. The length of the password can also be set by the user, by default it is set to 16 which is a very good length for strong and complex passwords, but using the number scroll the length can be set according to the user's choice. All check boxed are checked by default in accordance with secure by default principle. Each and every password that is generated is character-masked by default which can be viewed by clicking on the eye icon to unmask them. Further the generator also offers the options for copying the generated password to

clip board with a button click. Notification are integrated into this functionality as well, in case no check box is checked and the generate button is clicked a notification pops up notifying the user to check at least one box to generate a password. Notifications also popup notifying the user regarding a password being copied to the clipboard. The password generator can also be reached from add password modal.



7. **Strength Analyser:** Another main functional component completed is the Password strength analyser functionality. This functionality utilises the ZXCVCBN library to accurately test the strength and entropy of the password. The ZXCVCBN library and evaluation algorithm were developed according to popular password cracking algorithms worldwide which conduct pattern matching to estimate the strength according to common passwords, common patterns like keyboard patterns, common English words and common names, repeating sequences etc. This functionality is completely dynamic and shows responses according to user state and password input. The strength of the password that the user types in is indicated by colour coded progression and colour coded dynamic updated text responses. The *score* from the ZXCVCBN is extracted to dynamically and colour wise update the response and the progression bar to display the strength. This progression bar has also been integrated in the Add/Edit popup modals to display strength while adding the password-account. An Info-tool-tip is also integrated to give information about the library used for the

strength analysis on the front-end for the users. The input typed in the form field for password strength estimation is also character-masked using the type as password.

Robust Password strength analyser

Analyse the strength of your password efficiently and be rest assured

Dynamic strength resultant message

Strength progress indicator

Its Okay-ish

Or go to [Generate](#) a new password which you can paste here!

Decent

- 8. Distinctor** – The final functional component implemented in the application is the password *distinctor* or uniqueness analyser which is a very useful functionality to have, it lets the user check whether their password has been hacked or breached. This functionality also includes a tool-tip which gives information about the API used. The functionality utilises the Have-I-Been-Pawnd API from Troy Hunt to cross-reference the input password (also character masked) with the troy hunt database of passwords that have been recorded in any past data breaches. The database also contains how many times a password (account) may have been breached. The functionality dynamically responds with a message indicating whether the password entered by the user has been hacked or breached before, whether it is safe to use or not and how many times it has been breached or hacked. The dynamic message response is also colour coded according to the response type. All passwords that are typed in are SHA-1 hashed before they are cross-referenced with the database and no password is stored by the Troy-Hunt website in any way and no user action conducted is logged, hence there is no risk of any password leaking from this trusted and security certified website that provides the API.

Password Distinctor

This function will help you analyse the uniqueness of your chosen password

- It cross-references the Troy Hunt (Have-I-Been-Pawnd) database
- It determines whether the password entered has been compromised/breached before.
- It indicates whether a password is safe to utilise or not!

The whole message is dynamic and colour coded

All passwords the user enters are character masked (and encrypted when sent through the API)

Number of times the password has been breached

Check if your password has been breached or hacked before over here:

This password isn't safe to use and appeared in 9,545,824 known data breaches. You can still use it, but you probably shouldn't.

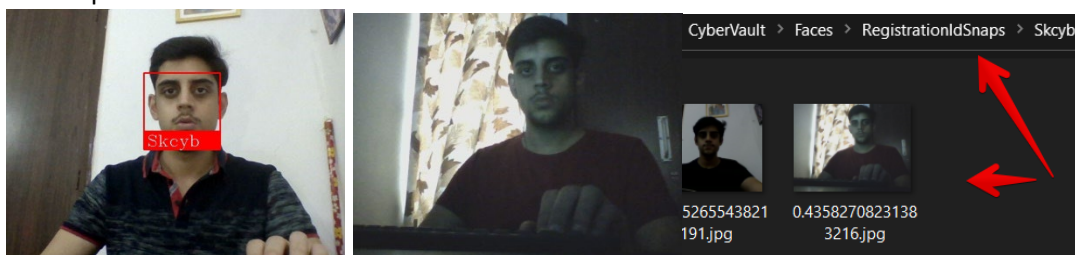
More Interesting Code snippets

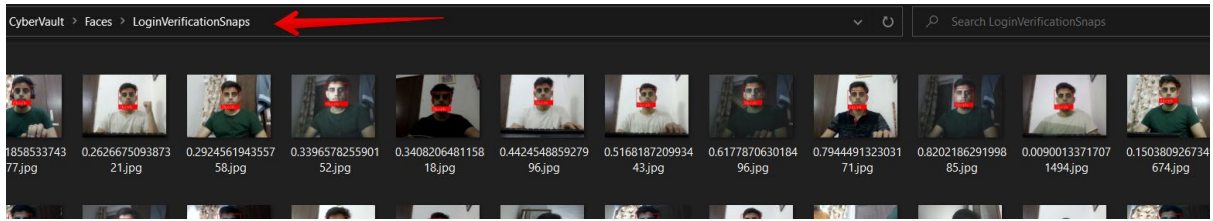
1. Facial scan and validation – The most interesting part of CyberVault is how the Facial ID authentication works. The Facial authentication logic has been setup in the flask server python back-end of the project. The main algorithm method that is utilised to implement and integrate the dlib based face recognition library is deep metric learning. The face-registration function as can be seen below is responsible for accepting a single input image that is captured from the webcam and resized using OpenCV. In theory then the face recognition module is used to return a real-valued 128-d vector which is a comprehensive list of 128 numbers that are real valued used to quantify a face called the facial encodings. The neural net of the Deep learning metric modal (from dlib) is trained to develop and return a 128-d vector when provided an image capture.

In the face registration function; when the call is made to register a face from the client side consequently OpenCV is used to initially grab a single frame from the webcam video feed. The frame is then resized to $\frac{1}{4}$ of its original size for faster processing of the face recognition. Then since OpenCV uses the BGR colour frame by default it needs to be changed to RGB which is the acceptable format for face-recognition library. Then the face recognition library is used to detect the face in the frame that was captured and converted to RGB using ***face_locations()*** method then the facial encoding are calculated using the ***face_encoding()*** which factors in the facial location. Then some exception handling is done to make sure the face is scanned and encoded. Finally then the image with the registration name that was also sent with it are stored in the local directory of the project. Finally the facial encoding with the name is then saved into the SQLite DB with a prepared INSERT query.

In the facial validation function then; first all the values from the database are retrieved suing the ***retrieveAllData()*** function defined earlier. The registered face encoding and name is set into variables. The process of capturing a frame resizing it and converting it into RGB and calculating its facial location and encoding is repeated. Then this time instead of saving it into the database the encoding is looped through and compared (using ***compare_faces()***) i.e., saved into array variable - 'matches') with the registered facial encodings that were retrieved earlier. The facial distance between these encodings is also calculated and the smallest distance to the new face is extracted using ***py.argmax()*** (from NumPy) and saved into the ***best-match*** variable. Then it is checked if the best match variable value exists in the matches array then the name for the frame is set to the name that was retrieved. Finally OpenCV is used yet again to manipulate this new frame that was retrieved to create a rectangular boundary around the detected face and then set text (label) for it with the registration name that has been confirmed for this new face frame as well, thus identifying the person. This frame with the rectangular boundary and the name label is then saved in the local directory present in the project folder.

Both functions saved a response in a message variable after the processing is completed successfully which is then returned to the client side, and according to which then the client side is manipulated.





```

# registration route for the page which will save the face encoding with the name into the sql database
app.route('/faceRegistration', methods=['GET'])
def faceRegistration():
    # using prepared-parameterized query statements to avoid SQL injection added with input validation on front-end (next)
    query=""INSERT INTO registeredFaces VALUES(?,?)""
    registrationName=request.args.get("name")
    # Grabbing a single frame of video from webcam
    faceCapture=cam.VideoCapture(0)
    ret,frame=faceCapture.read()
    # Resizing the frame of video to 1/4th size for faster face recognition processing
    smFrame=cam.resize(frame,(0,0),fx=0.25,fy=0.25)
    # Convert the frame image from BGR color (opencv compatible) to RGB color (face_recognition compatible)
    rgbSmFrame=sfFrame[:,::-1]
    # detecting all the faces and computing face encodings in the current frame
    faceLocations=face.face_locations(rgbSmFrame)
    faceEncodingValues=face.face_encodings(rgbSmFrame,faceLocations)
    if(faceEncodingValues==[]):
        message="Face not detected"
    else:
        directory=os.path.join(registrationIdentificationSnaps,registrationName)
        if(not os.path.isdir(directory)):
            | os.mkdir(directory)
            | os.chdir(directory)
            randomNum=py.random.random_sample()
            cam.imwrite(str(randomNum)+".jpg",frame)
            faceCapture.release()
            cam.destroyAllWindows()
            encodedVal=""
            for i in FaceEncodingValues:
                encodedVal=encodedVal+str(i)+","
            if registrationName.replace(" ", "").isalpha(): # server side data validation - valid string
                checkQuery=""SELECT registrationName FROM registeredFaces WHERE registrationName=?""
                try:
                    sqlCursor.execute(checkQuery,(registrationName,))
                    exists=sqlCursor.fetchall()
                    if not exists:
                        list=[registrationName,encodedVal]
                        valForDB=tuple(list)
                        sqlCursor.execute(query,valForDB) # value prepared and added to db
                        connection.commit()
                        message="Registered"
                    else:
                        message="Name already used"
                except sqlite3.Error:
                    | message="SQL server error, Could not register FaceID"
            else:
                message="Registration name used is not valid"
    return message

```

```

def faceValidation():
    retrieveAllData()
    global database
    if(database==[]):
        message="Not in the system"
    else:
        IdentifiedFaceEncodings=[i[1] for i in database]
        IdentifiedFaceNames=[i[0] for i in database]
        faceLocations=[]
        faceEncodingValues=[]
        faceNames=[]
        faceCapture=cam.VideoCapture(0)
        ret,frame=faceCapture.read()
        smFrame=cam.resize(frame,(0,0),fx=0.25,fy=0.25)
        rgbSmFrame=sfFrame[:,::-1]
        faceLocations=face.face_locations(rgbSmFrame)
        faceEncodingValues=face.face_encodings(rgbSmFrame,faceLocations)
        faceNames=[]
        if(faceEncodingValues==[]):
            message="Not in the system"
        else:
            for faceEncodedValue in faceEncodingValues:
                # Comparing the encodings
                matches=face.compare_faces(IdentifiedFaceEncodings,faceEncodedValue)
                name="Unknown"
                faceDistances=face.face_distance(IdentifiedFaceEncodings,faceEncodedValue)
                # using the known face with the smallest distance to the new face
                bestMatch=py.argmin(faceDistances)
                if matches[bestMatch]:
                    name=IdentifiedFaceNames[bestMatch]
                if(name=="Unknown"):
                    | message="Not in the system"
                else:
                    | message=name
                    | faceNames.append(name)
            #Creating a rectangular boundary and setting its text label and color according to the name recognized for the frame
            for(top,right,bottom,left), name in zip(faceLocations,faceNames):
                top=top*4
                right=right*4
                bottom=bottom*4
                left=left*4
                cam.rectangle(frame,(left,top),(right,bottom),(0,0,255),2)
                cam.rectangle(frame,(left,bottom-35),(right,bottom),(0,0,255),cam.FILLED)
                font=cam.FONT_HERSHEY_COMPLEX
                cam.putText(frame,name,(left+6,bottom-6),font,1.0,(255,255,255),1)
            os.chdir(loginVerificationSnaps)
            randomNum=py.random.random_sample()
            cam.imwrite(str(randomNum)+".jpg",frame)

```

2. Distinctor - Another interesting integration was the use of the HIBP API component provided by react-have-I-been-pawnd module/library. Unlike how its implementation was experimented with before it could not be directly imported into a Next.js application and hence required the use of next/dynamic extension which is a part of the next.js framework. Using dynamic the HIBP react component was imported and was then utilised in the application. A react client side functional component with client render was developed first to accommodate the HIBP component and then according to its documentation the HIBP component was utilised. UseState() was used to set the state of the input password to a dynamic state depending on user input. Then the password variable which was set using useState() was used as a parameter for the HIBP password checked component. According to the HIBP documentation the error pawnd and count of breaches was imported and set according to the password parameter and finally the parameter was passed to this API component to get the results as follows.


```

{faceValidated == false ?
<div className="p-3 my-5 bordered" style={{alignItems:'center', display:'flex', FlexDirection:'column', justifyContent:'center'}}>
  <Webcam className="webcam"/>
  <br/></br>
  <div className="form-group">
    <Button onClick={()=>{validateFace()}} className="btn btn-primary" style={{display:'block', margin:'0 auto',height:'50px',width:'500px'}}>
      
      Validate Face
    
    </Button>
  </div>
</div>
</>
:
<div
  {valName=="Not in the system"?
  <div className="p-3 my-5 bordered text-center">
    <h4><strong>Face not found ,
    <a href="/faceReg" className="text-danger">Register</a> your face if you haven't done so yet!</strong>
    <Tippy content="Visit register face page">
      
    </Tippy>
    </h4>
    <div className="form-group">
      <Button onClick={()=>{setFaceValidated(false), setValName(null)}} className="btn btn-primary" style={{display:'block', margin:'0 auto'}}>
        
        Retry!
      </Button>
    </div>
  </div>
  </>
  :
  <div
    {valName=="Unknown"?
    <div className="p-3 my-5 bordered text-center">
      <h4><strong>Face not found ,
      <a href="/faceReg" className="text-danger">Register</a> your face if you haven't done so yet!</strong>
      <Tippy content="Visit the register face page">
        
      </Tippy>
      </h4>
      <div className="form-group">
        <Button onClick={()=>{setFaceValidated(false), setValName(null)}} className="btn btn-primary" style={{display:'block', margin:'0 auto'}}>

```

Conditional rendering

On-click trigger

An optional link to the register page will also be shown in case the face is not recognized or is not in the system

Failure of validation would set the state to false again

```

const validateFace = () =>{
  stopCapture()
  fetch("http://127.0.0.1:5000/faceValidation")
  .then(res =>res.text())
  .then((res)=>{
    setValName(res)
    setFaceValidated(true)
  })
  .catch(error=>{
    notification(notify)
  })
}

```

Only set true once face is validated

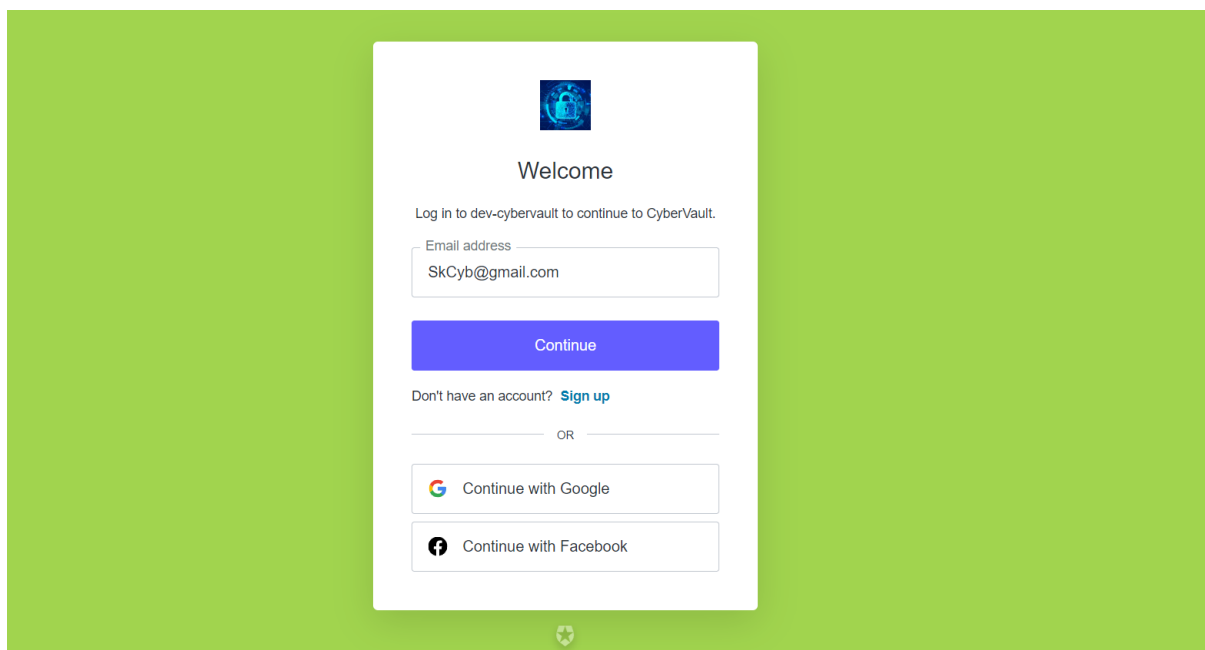
```

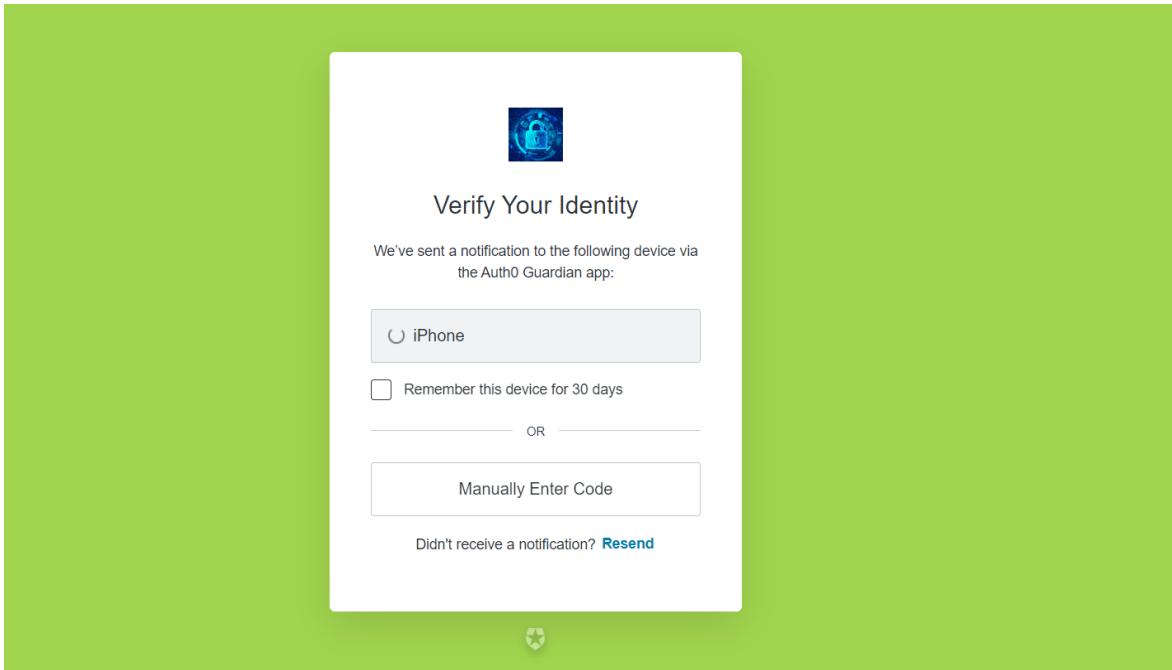
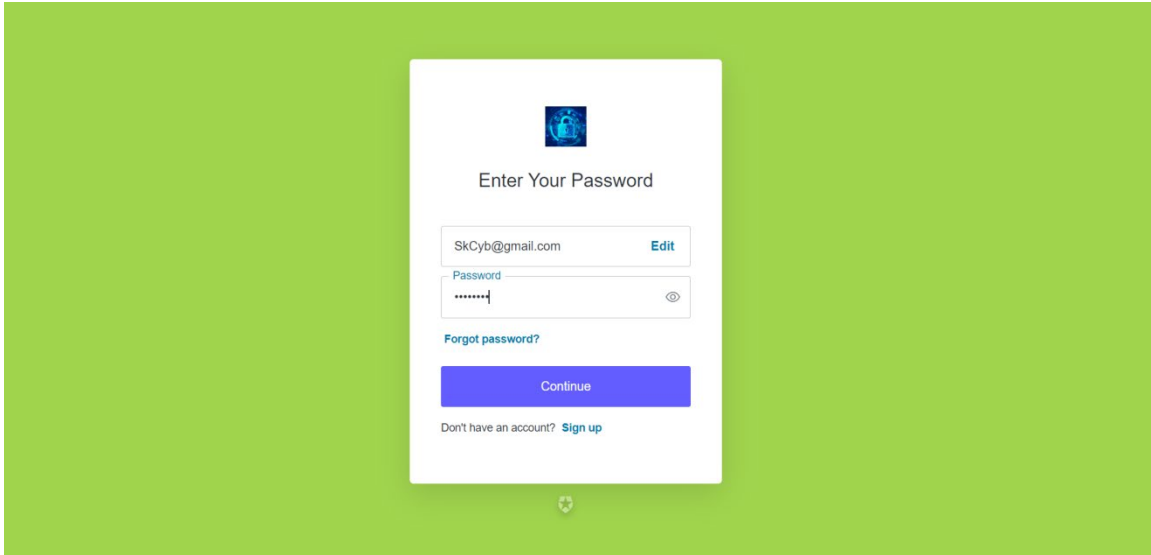
:
<>
<div className={styles.manager}>
  <PasswordsDisplayComp
    passwords={passwords}
    handleEdit={(payload) => updatePasswordAccount(payload)}
    handleDelete={(id) => deletePasswordAccount(id)}
  />
</div>

```

Only displayed as when face validation is accepted

Final Screenshots of application and GUI






CyberVault Register Face Manager Generator Strength Analyser Distinctor Log-Out SK

Face Registration

You can register your face here! ⓘ



Enter your name:

Register Face ⓘ

CyberVault Register Face Manager Generator Strength Analyser Distinctor Log-Out SK


SK

Hello Skcyb ⓘ, Welcome to CyberVault ⓘ ⓘ


This is your own personal protected password managerial space, Enjoy! 🎉

Your account is not verified, To verify your account please find a verification link in the email you have used to signup and sign in!

Add Password-Account 🔒




CyberVault Register Face Manager Generator Strength Analyser Distinctor Log-Out SK



Validate Face ⓘ

CyberVault Register Face Manager Generator Strength Analyser Distinctor Log-Out SK





Hello Skcyb 🇮🇳, Welcome to CyberVault 🇮🇳 📧

This is your own personal protected password managerial space, Enjoy! 🎉

Your account is not verified, To verify your account please find a verification link in the email you have used to signup and sign in!

[Add Password-Account](#) 🗝️








7 passwords present and **face verified**. List of your passwords and accounts are as follows:

	Social/Business/Personal Account : Udemy	Associated email : skCybe@gmail.com
	Social/Business/Personal Account : Thinkable	Associated email : testUser@gmail.com

CyberVault Register Face Manager Generator Strength Analyser Distinctor Log-Out SK

[ADD PASSWORD-ACCOUNT](#) 🗝️

7 passwords present and **face verified**. List of your passwords and accounts are as follows:

	Social/Business/Personal Account : Udemy	Associated email : skCybe@gmail.com
	Social/Business/Personal Account : Thinkable	Associated email : testUser@gmail.com
	Social/Business/Personal Account : Spotify	Associated email : skCyb@gmail.com
	Social/Business/Personal Account : Facebook	Associated email : skCyb@gmail.com
	Social/Business/Personal Account : Youtube	Associated email : skCyb@gmail.com
	Social/Business/Personal Account : Github	Associated email : skCyb@gmail.com
	Social/Business/Personal Account : Amazon	Associated email : skCyb@gmail.com

localhost:3000/dashboard CyberVault Generator Strength Analyser Distinctor Log-Out SK

You Have Been Idle! ✕

Due to prolonged in-activity you are going to be logged-out, Do you want to stay?




[Logout](#) [Stay](#)

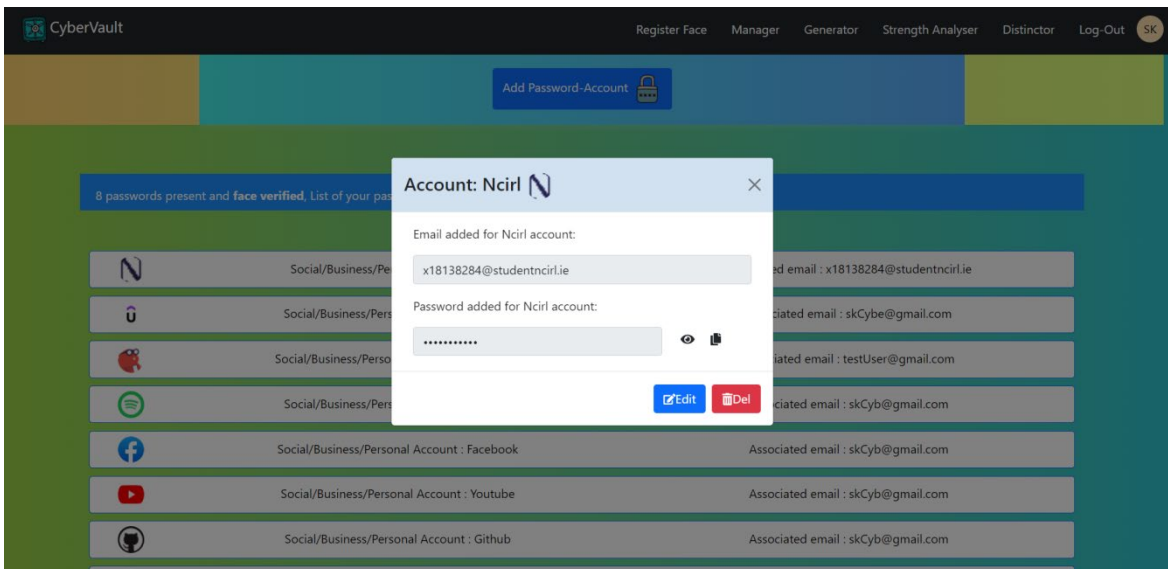
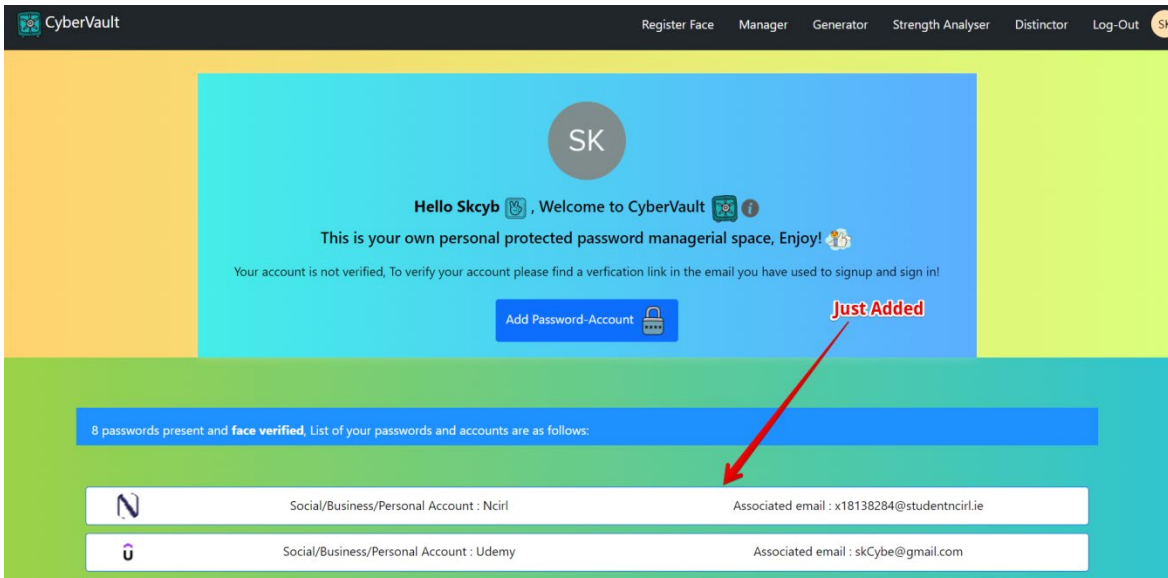
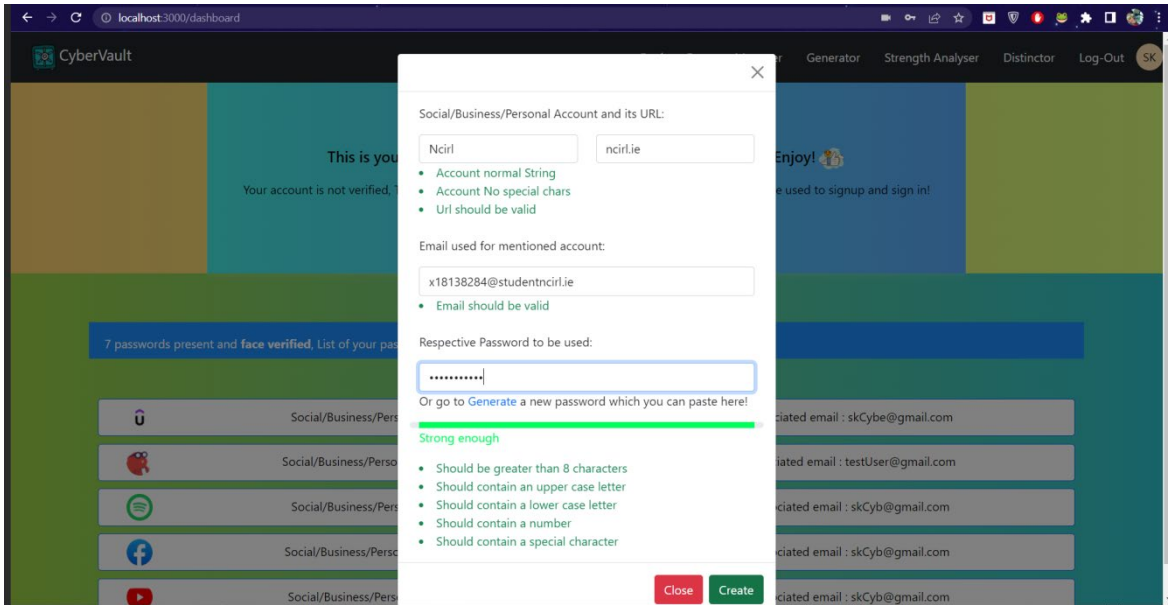
This is your own personal protected password managerial space, Enjoy! 🎉

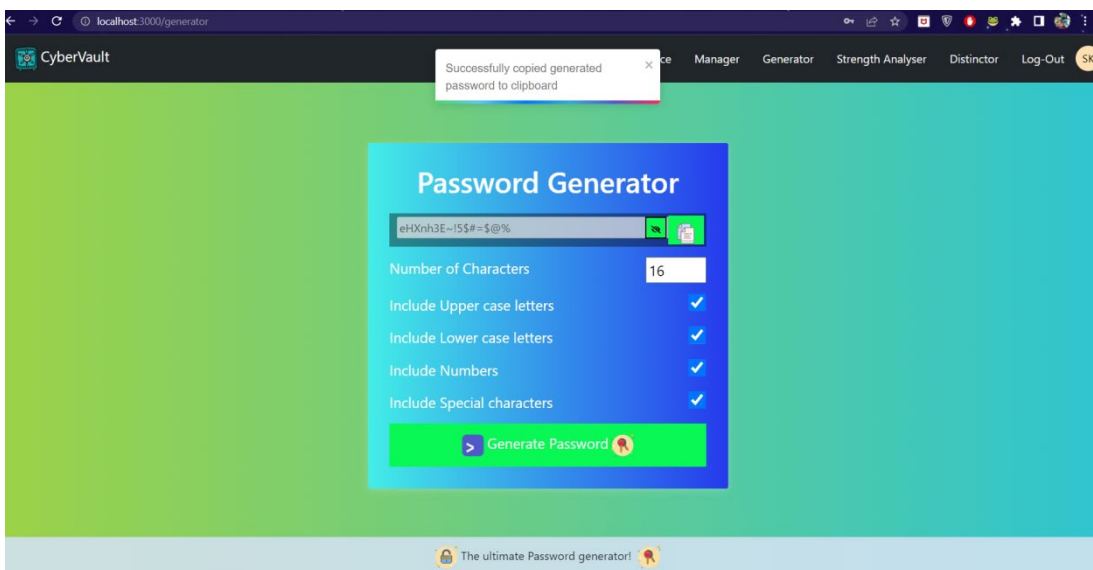
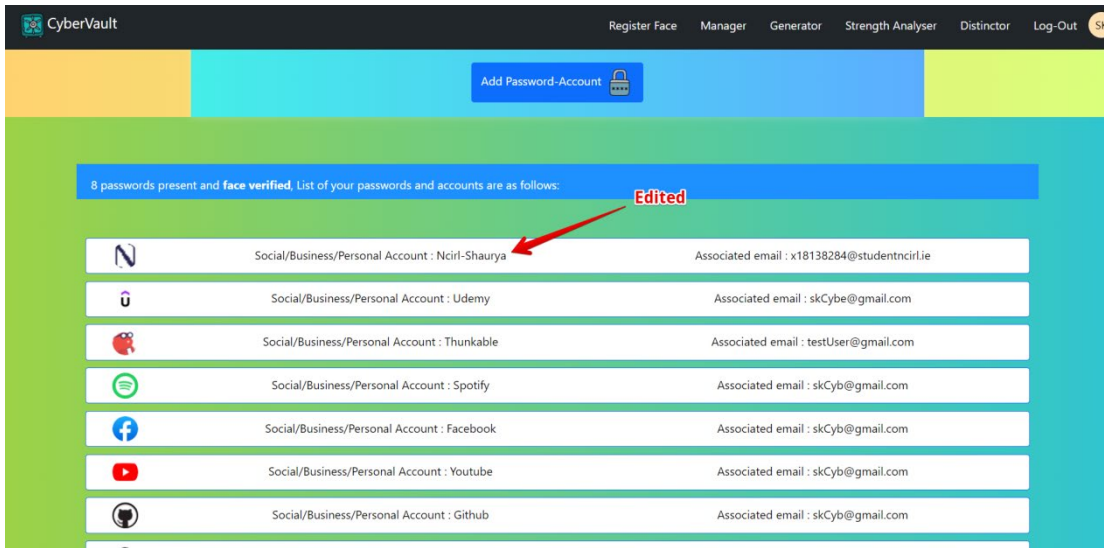
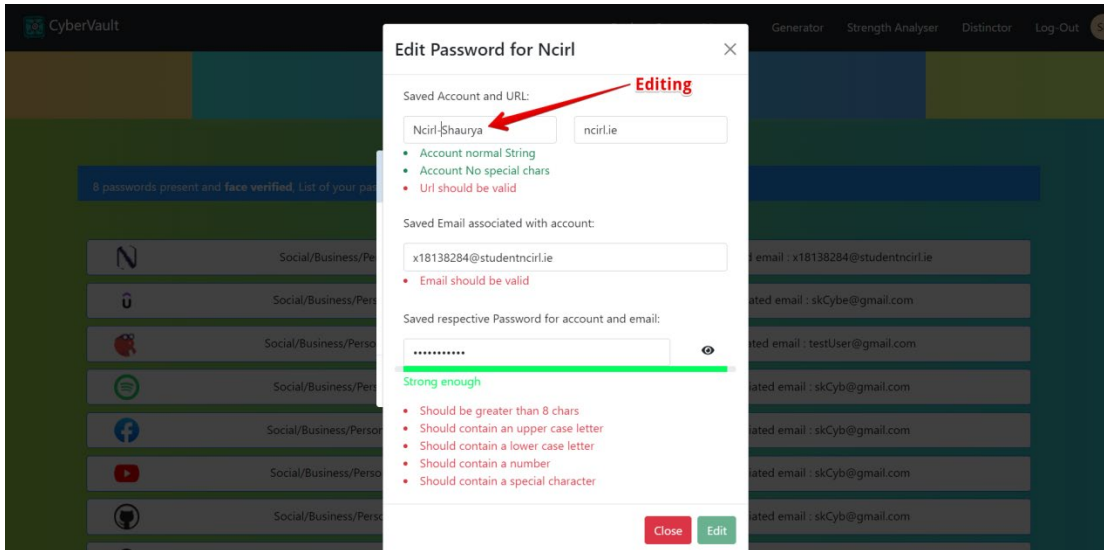
Your account is not verified, To verify your account please find a verification link in the email you have used to signup and sign in!

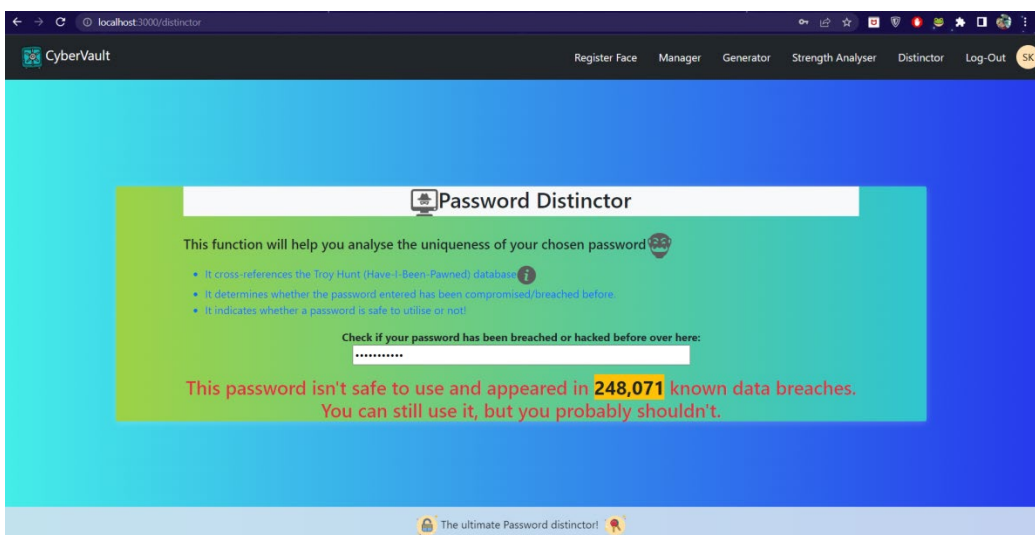
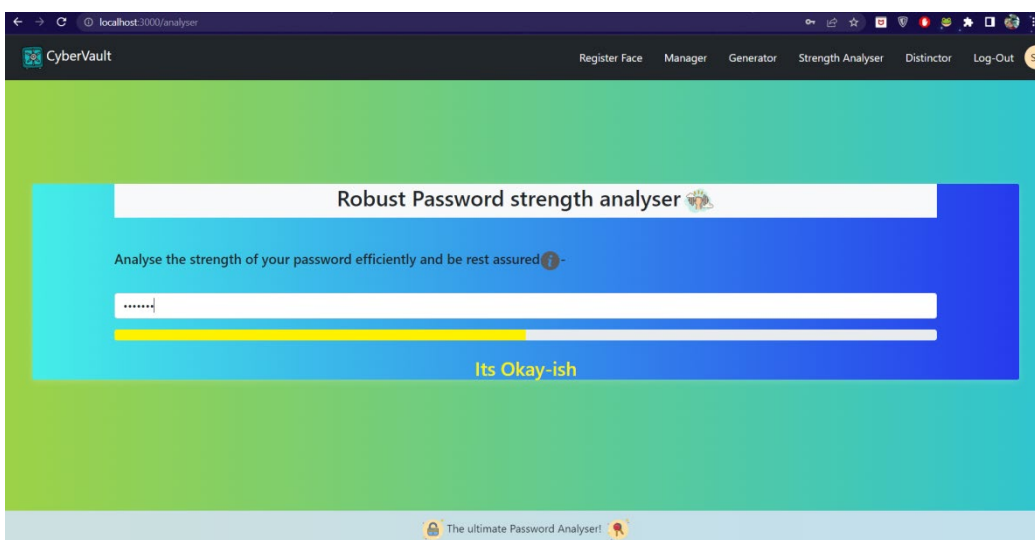
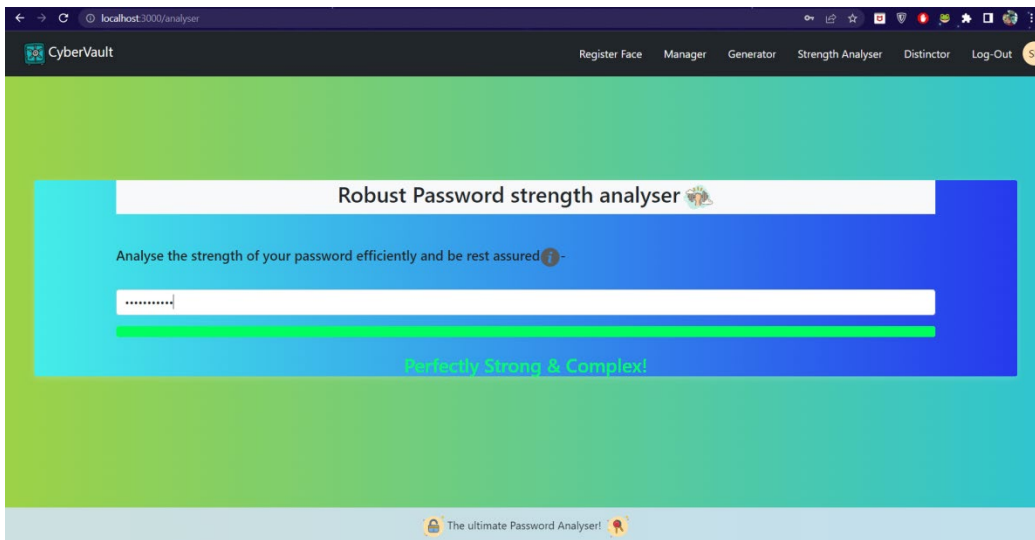
[Add Password-Account](#) 🗝️

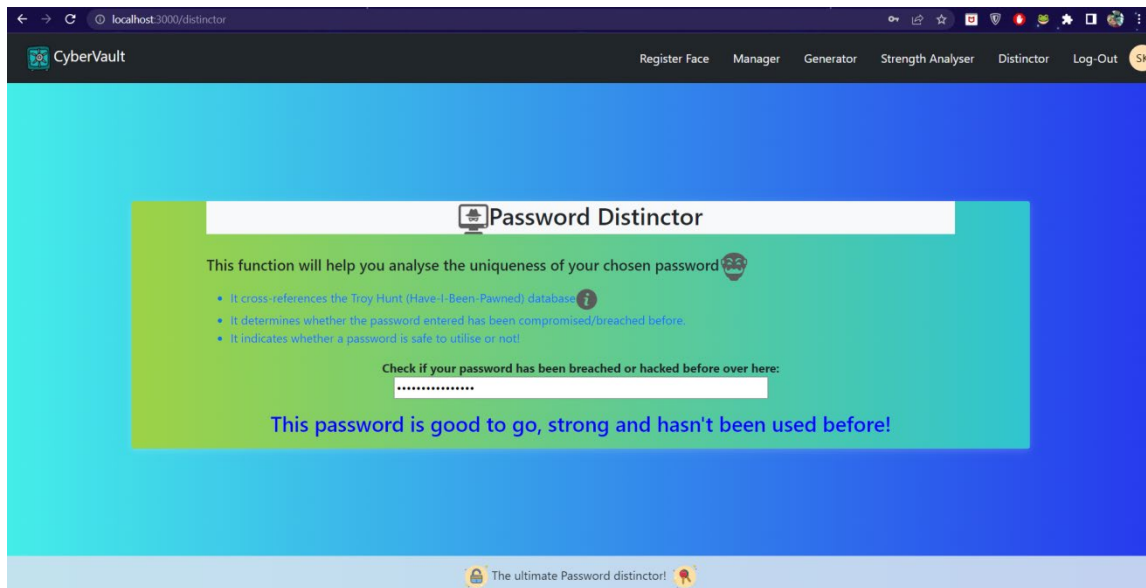
7 passwords present and **face verified**. List of your passwords and accounts are as follows:

	Social/Business/Personal Account : Udemy	Associated email : skCybe@gmail.com
	Social/Business/Personal Account : Thinkable	Associated email : testUser@gmail.com
	Social/Business/Personal Account : Spotify	Associated email : skCyb@gmail.com









Security Focused Implementation in the application

The application is developed with a ton of security focused code-bits or features that help protect the application from any kind of attempted breach, hack or data leak of anykind

- **Route Security, API security** (Client Side and Server side authentication required): A very important security feature of the application that has been implemented is route security i.e., no functionality or page route can be accessed by any user that does not exist in the user database and is not authenticated. The main condition that denies a user from accessing a page or functionality is whether they are authenticated/logged in or not i.e., checking if there is a user session with session cookie created for them (created when a user successfully signs up and logs in). If an unauthenticated user tries to access any page or functionality they are automatically redirected to the log-in page. The routes are protected using *GetPageAuthRequired()* hooks in conjunction with *GetServerSideProps()*. The server side props hook is used in next JS for live/dynamic data retrieval and for rendering HTML pages from JavaScript files. First the pages are rendered server side and are then displayed to the user (i.e., for Dynamic data rendering and authentication). Server side props involve data being requested every time the page is requested or rendered. This hook combined with the authentication required hook stops the pages from rendering if a user (session and session cookie) is not detected on the application (meaning user is not authenticated); it invokes a call back to the login page straight away and does not let the user access anything until they login. Even the server side created API that are used for communication with Fauna-DB are protected with the *getAPIAuthRequired()* hook which necessitates the requirement of user authentication for the API requests to be made. For each page request and API request to be successful and according to the logged in user; the session cookie is validated to retrieve user specific data and to target all post requests only for the specific authenticated user's account.

API requests have been secured using strict authentication.

```
export default withApiAuthRequired(async (req, res) => {
  const user = getSession(req, res).user
  if (req.method === 'POST') {
    let {
      accountName, accountUrl, email,
      encryptedPassword, avatar
    } = req.body
    if(schema.validate(req.body)){
      let newPassword = await addNewPassword(
        accountName, accountUrl,
        email, encryptedPassword,
        user,
        avatar
      )
    }
  }
})
```

The request and response body of the API request require the user session and User Object

All PUT/POST etc. requests are also secured by strict authentication

```
export default withApiAuthRequired(async (req, res) => {
  const user = getSession(req, res).user
  if (req.method === 'PUT') {
    let password = await updatePassword(
      req.body, req.query.id
    )
    res.status(201).json({
      message: 'Successfully updated password'
    })
  }
})
```

They also require an authenticated user with an authenticated User session

```
export const getServerSideProps = withPageAuthRequired()
export default Dashboard
```

The server-side props are set to require the Auth0 authentication thus securing the page rendering (Only render when authenticated, redirect otherwise)

```
export const getServerSideProps = withPageAuthRequired()
export default distinctor
```

Each page is secured by this authentication required hook

```
export const getServerSideProps = withPageAuthRequired()
export default faceReg
```

```
export const getServerSideProps = withPageAuthRequired()
export default generator
```

```
export const getServerSideProps = withPageAuthRequired()
export default Analyser
```

- **Secure Authentication and User session management:** The Server side (Next.js API side) has been setup with the Auth0 configurations and the client side (since authentication takes place server side) is setup with the wrapping of the *UserProvider* component around the app by including it in the *_app.js* file which lets the client side know when the user is authenticated and when the user is not authenticated thus providing access to the application accordingly and

also helps accessing user details on the client side once authenticated. To setup a user session and keep it secure; once the user logs in and is redirected to the application after passing through authentication (including multifactor), the user is redirected with an authorization code which is exchanged for an `id_token` by the server-less call-back function (responsible for the redirection) by Auth0. Only after the `id_token` is validated that a user session is created and is stored in an encrypted Http only cookie (`appSession`). The session cookie is then sent each time a 'page render' is requested to recognize the authenticated user and their session. Then external API calls are made to the Fauna-DB through the Next.js API section (proxying) which also contains the `userSession` (through `getAPIAuthRequired()` hook). This is done to tie each add/edit/delete/fetch request made for the passwords to the user's session (retrieving User specific password-accounts, and posting password accounts with User id to make the password-accounts specific to the user who created them).

```
export default function App({ Component, pageProps }) {
  return (
    <AuthProvider>
      <NavigationComponent />
      <InactivityTimerComponent />
      <Component {...pageProps} />
    </AuthProvider>
  )
}
```

Whole application is wrapped with the user component To track a user's session throughout the application

Session Cookie

Name	Value	[P..	Expir...	S.	HttpOnly
appSession	eyJhbGciOiJIaX...	l...	/	2022...	3...	✓

Cookie Value Show URL decoded

eyJhbGciOiJIaX...

```
export default withApiAuthRequired(async (req, res) => {
  const user = getSession(req, res).user
  if (req.method === 'POST') {
    let {
      accountName, accountUrl, email,
      encryptedPassword, avatar
    } = req.body
    if(schema.validate(req.body)){
      let newPassword = await addNewPassword(
        accountName, accountUrl,
        email, encryptedPassword,
        user,
        avatar
      )
    }
  }
})
```

API requests have been secured using strict authentication.

The request and response body of the API request require the user session and User Object

- **DDOS protection:** An advantage of using auth0 was also to be able to track if authentication through a particular user's credentials is tried multiple times and failed again and again, if this crosses the threshold of failed login attempts (in a very short span of time) then the IP address is noted (IP-throttling) and a warning email is sent to the actual user regarding unauthorized authentication attempts
- **Secure password management encrypted Passwords** (strongest AES-256): Password protection and security is the primary purpose of the application itself and hence there is a major focus on the password protection. The user account password cannot be seen by the admin, in case the user forgets their password; all they can do is reset their passwords to get back into their accounts. The User account's password is always hashed and salted using bcrypt before storage. Auth0 even helps in encrypting data during transit using AES 128 bit encryption (TLS) to avoid any possible leak of data. The user account password standards are also high, so during signup the user has to abide by the validation conditions for setting their user account password. As for the password manager all passwords are strictly input validated i.e., users are forced to save strong and complex passwords (by default) (Multitude of features are provided in the application as mentioned above to assist the user in choosing the best possible password for their accounts and to help them choose new passwords in case their original one's are not up to date with the modern strong password standards). All passwords are not only input validated at client side but also on the server side (Next.js API). The passwords are encrypted using the Node.js's inbuilt Crypto module using the strongest and most efficient encryption algorithm out there i.e., AES-256 bit encryption using randomly generated 40 bit complex hex keys (which are stored as environment variables). The passwords are encrypted immediately as they are received from the user before they are added to the payload to be sent through API requests to the Fauna-DB. The viewing of the password manager itself containing all the user's saved password-accounts is also protected by facial validation so they are only viewable by the right Facial ID holders. All password and passwords assistance functionality involve character masking so there is no leak of data.

Very Weak

- Should be greater than 8 characters
- Should contain an upper case letter
- Should contain a lower case letter
- Should contain a number
- Should contain a special character

Forces users to create and store strong and complex passwords

Password validation rules

Create Your Account

Your password must contain:

- At least 8 characters
- At least 3 of the following:
 - ✓ Lower case letters (a-z)
 - Upper case letters (A-Z)
 - Numbers (0-9)
 - Special characters (ex. !@#%*&)
- ✓ No more than 2 identical characters in a row

Continue

Already have an account? [Log In](#)

```
const handleAccountPasswordOnChange=(e)->{
  setPassword(e.target.value)
  const isLengthy=e.target.value.length>8
  const hasUpperCase=/[A-Z]/.test(e.target.value)
  const hasLowerCase=/[a-z]/.test(e.target.value)
  const hasNum=/[0-9]/.test(e.target.value)
  const hasSpecialChar=/[!@,##$%&*,>,,<`,`~?`=,+,-]/.test(e.target.value)
  setAccountPassErr({isLengthy, hasUpperCase, hasLowerCase, hasNum, hasSpecialChar})
}
```

Client-side Password validation parameters

```
const handleCreate = async () => {
  const encryptedPassword = encryptString(password)
  const payload = {
    accountName,
    accountUrl,
    email,
    encryptedPassword
  }
  props.onCreate(payload)
  setAccountName('')
  setAccountUrl('')
  setEmail('')
  setPassword('')
  notification(notify)
}
```

Password is encrypted before it is added to the payload

```
const schema=Yup.object({
  accountName:Yup.string().matches(/^[aA-zz\s]+$/).required(),
  accountUrl:Yup.string().matches(/^(https?):\/\/(www.)?[a-z0-9]+(\.[a-z0-9]+)+$/).required(),
  email:Yup.string().email().required(),
  encryptedPassword:Yup.string().required()
});

export default withApiAuthRequired(async (req, res) => {
  const user = getSession(req, res).user
  if (req.method === 'POST') {
    let {
      accountName, accountUrl, email,
      encryptedPassword, avatar
    } = req.body
    if(schema.validate(req.body)){
      let newPassword = await addNewPassword(
        accountName, accountUrl,
        email, encryptedPassword,
        avatar
      )
    }
  }
})
```

Server side data validation is done using Yup schema

Before password-add request is sent, the request body is data-validated

```
{
  "ref": "Ref(Col1652527666580000)",
  "ts": "1652527666580000",
  "data": {
    "accountName": "Amazon",
    "accountUrl": "https://amazon.com",
    "email": "skCyb@gmail.com",
    "encryptedPassword": "7f99f79b1095cc917c96a110e884a4284b65ec978c90d29a39912bacd1d5c9e4",
  }
}
```

Password is strictly encrypted in the DB

```
import crypto from 'crypto'

export const encryptString = (text) => {
  const cipher = crypto.createCipher('aes256', 'process.env.REACT_APP_SECRET_KEY')
  const encryptedData = cipher.update(`${text}`, 'utf8', 'hex') + cipher.final('hex')
  return encryptedData
}

export const decryptString = (encryptedText) => {
  const decipher = crypto.createDecipher('aes256', 'process.env.REACT_APP_SECRET_KEY')
  const decryptedData = decipher.update(encryptedText, 'hex', 'utf8') + decipher.final('utf8')
  return decryptedData
}
```

Encryption/Decryption code

Key is saved in the environment variables and is referenced here

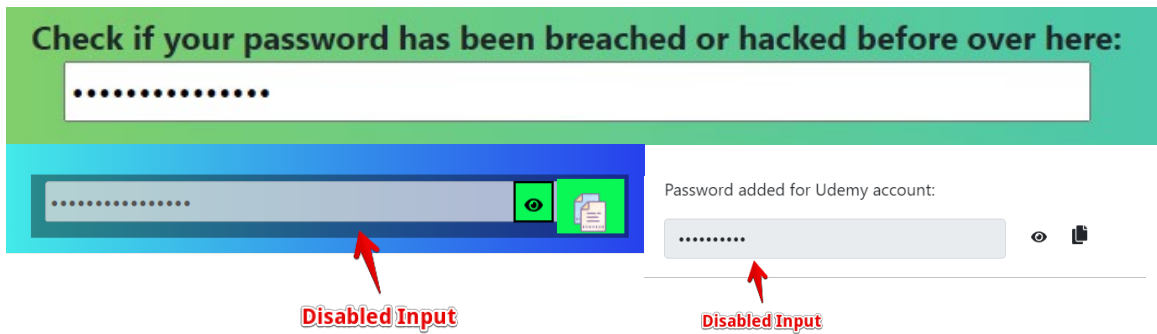
- **Character Masking:** Another security focus that has been implemented in the application is strict character masking i.e., each and every password related functionality and component involves character masking of the password field. Whether it is the password generator, distincter, analyser or the manager itself; each password related field is character masked and where there is no input required from the user such as the generator the fields have also been disabled from receiving any user input. The passwords in these components can be viewed by clicking the eye icon to unmask even then without the copy to clipboard functions it is impossible to copy these password strings (like using Ctrl+C and Ctrl+V). In the manager just to the preview the passwords facial ID validation is required.

.....

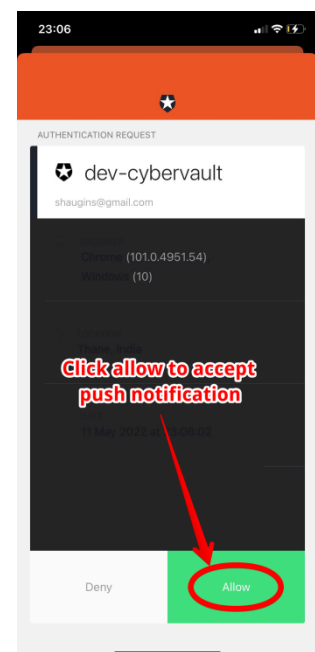
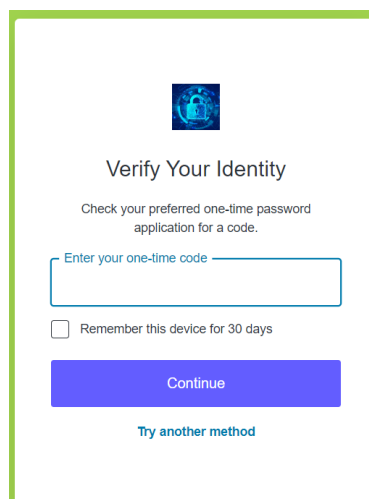
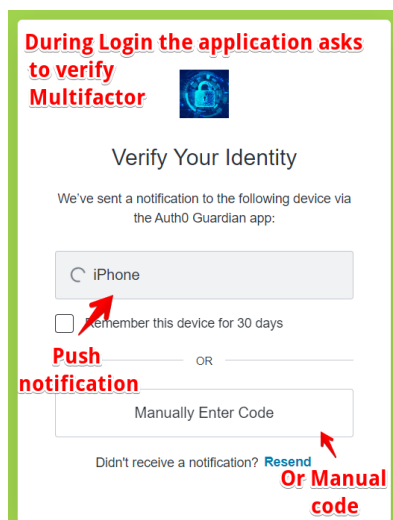
Or go to [Generate](#) a new password which you can paste here!

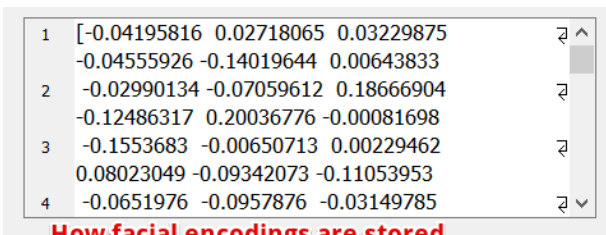
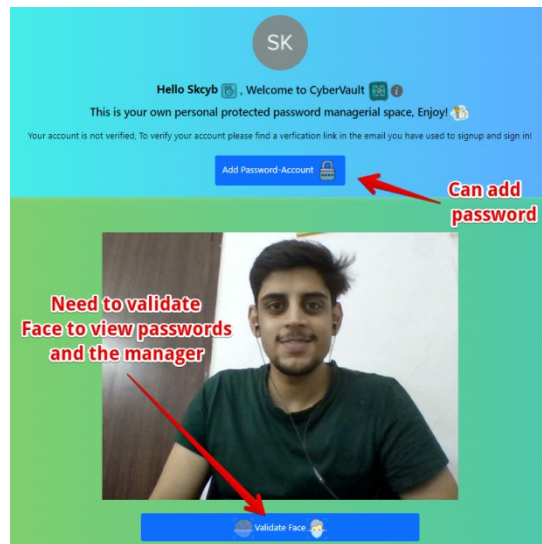
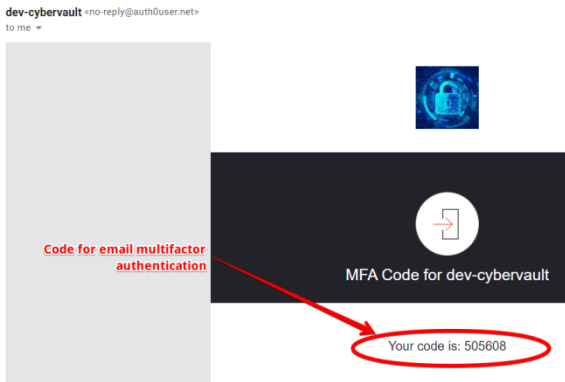
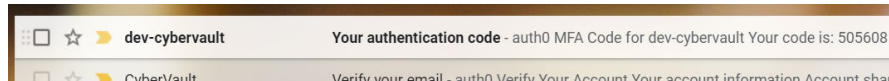
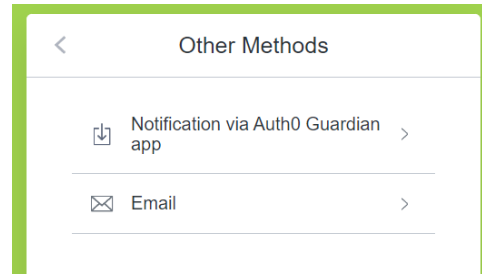
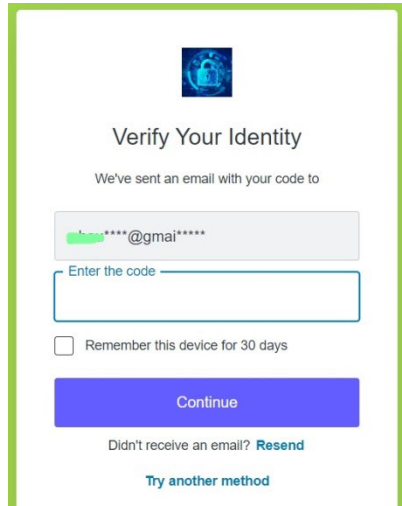
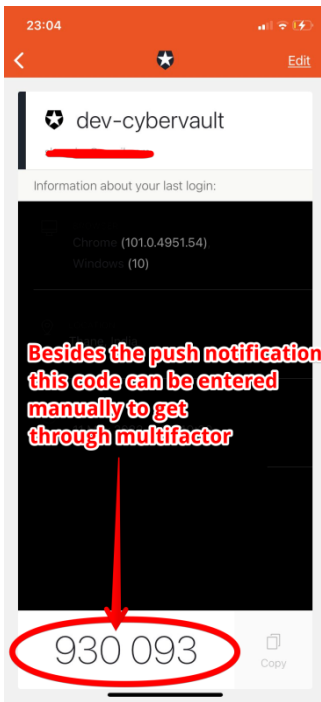
Decent

.....

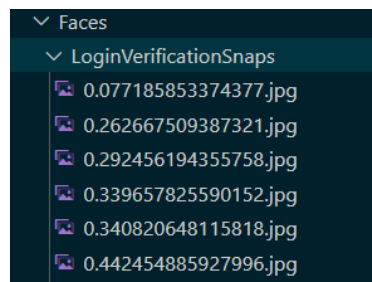


- Multifactor-authentication + Facial validation:** A very important security focused implementation that has been integrated in this application is the use of multifactor authentication. As explained above multifactor authentication has been implemented with auth0 guardian application which includes either push notification to confirm identity, code from the guardian application or a code sent to the verified email ('Something you have' multi-factor), during signup a user is prompted to download the guardian application on their mobile devices after which they can scan the QR code displayed on the screen to link their user account with the guardian application, in case the user does not wish to do so they can opt for the email code verification method. Then there is the facial ID authentication (Something you are) which has been included once the user is authenticated and inside the application, it acts as the third factor ('something you are' factor). This is included to protect the password manager individually, so the user has to validate their registered face ID to view their manager and all their saved passwords that are fetched from the database. This is done using python face_recognition dlib based library which helps compare and validate facial encodings that are computed using deep learning metrics (128-d vectors). A snap is taken each time; it's computed encodings are saved in the SQLite database along with the user's name (also input validated) and the facial image is locally stored in the application directory.// CAN SEEN SSCREENSHOTS FROM ABOVE//



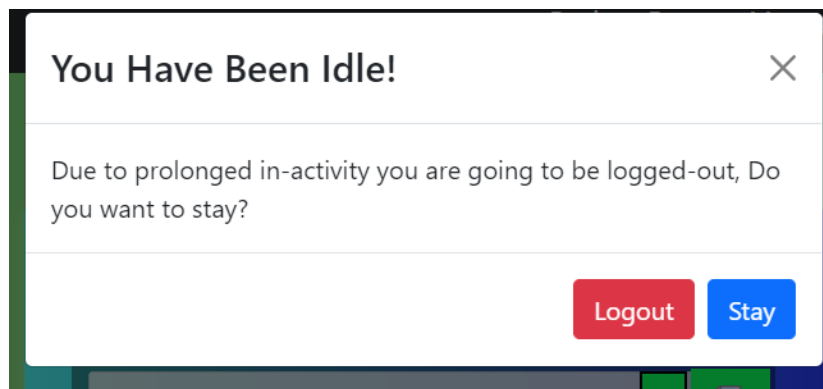


How facial encodings are stored



- Strict Input validation including data validation on both client and server side:** A very important security focused implementation is the integration of strict input validation on both the client as well as the server side. The front-end client side in the Next.js + React.js includes the functionality that takes the user input to save in the database is the password manager which has heavy and strict input data validation that is done using regular expressions or regex for all of the relevant details including account name (normal string without special characters or SQL keywords), account URL (abides by regex URL expression), account email (abides by regex email expression) and password (has conditions such as requirement of lower and upper case, number and special character). These input validations are integrated in both add/edit

to access it in a public location and in case they leave their computer for a short period of time, there is a major risk regarding leaking of their sensitive data i.e., a passing third party might notice their passwords being open on the screen and try to exploit the user's accounts. To avoid such a scenario and many similar scenarios the idle timer function was created using the react-idle-timer library/module. This helps the front-end detect and scan for user movements on the application pages only when the user is authenticated; movements includes cursor moving, clicking scrolling etc. If none of the user movements are detected in less than 3 minutes, the user is considered Idle even though they are authenticated and logged-in. In such a case a prompt is popped up on the screen letting the user know they have been idle for a while and would be logged out automatically unless they decide to click the stay button which resets the timer. They could also choose to logout by clicking the logout button in the popup. If no user movement is detected when the popup appears the user is automatically logged out in the next 3 minutes and would have to log in again to access the application and their account-passwords.



- **Proper and thorough testing in artificial environment:** Another security concern of an application is to look for bugs and problems that might lead to a breach or reveal something about the application that was not apparent at the time of development. Such things can be found out during thorough testing and with this in mind thorough testing has been conducted with the application in an artificial isolated environment. The testing is all round including integration, end to end and unit testing (also trial user testing) all functionality of the application, rendering capabilities and outcomes of the application. The testing has been explained in more detail under the testing heading.
- **Error & Exception handling:** Error handling has been given major importance during the development of this application since front-end errors are actually shown on screen with stack traces in application developed with JavaScript frameworks which is bad practise. Error handling has been done at the server level with specific error messages in try-catch and if-else error prone functions (code for communicating with servers, code for connecting to the database etc.) and has been done in the front-end with either window alerts or notifications depending on the situation to avoid any stack trace data being leaked. Further proper error handling has also been conducted at the server level since proper error messages with http codes have been formed as error responses for failed API requests etc. Finally even at the back-end i.e., the python flask facial authentication server which also connects and makes request to the SQLite database is properly error handled and possible calls or connection statements are all forcefully put into try-except statements which manually created error responses to handle any SQLite3 related errors specific to the function they are being used in. Then If-Else statements all over

the program are used to handle any type of general errors that can be noticed in case a function does not perform in a particular required way and to handle checks and conditions (done both in the front-end, server side as well as the back-end).

```

query='SELECT * FROM registeredFaces' # database table name
try:
    sqlCursor.execute(query)
    resultSet=sqlCursor.fetchall()
    for i in resultSet:
        j=[]
        j.append(i[0])
        string=i[1][1:-2]
        numbers=[]
        for k in string.split():
            numbers.append(float(k.strip()))
        j.append(numbers)
        database.append(j)
except sqlite3.Error:
    print('SQL server issue, unable to fetch all data!')

```

Manual error handling

```

checkQuery=""
try:
    sqlCursor.execute(checkQuery,(registrationName,))
    exists=sqlCursor.fetchall()
    if not exists:
        list=[registrationName,encodedVal]
        valForDB=tuple(list)
        sqlCursor.execute(query,valForDB) # value prepared and added to db
        connection.commit()
        message="Registered"
    else:
        message="Name already used"
except sqlite3.Error:
    message='SQL server error, Could not register FaceID'

```

Manual handling

```

const user = getSession(req, res).user
if (req.method === 'POST') {
    let {
        accountName, accountUrl, email,
        encryptedPassword, avatar
    } = req.body
    if(schema.validate(req.body)){
        let newPassword = await addNewPassword(
            accountName, accountUrl,
            email, encryptedPassword,
            user,
            avatar
        )
        res.status(201).json({
            message: 'Successfully created password',
            data: newPassword,
            status: 'ok'
        })
    } else{
        return res.status(400).json({
            message: 'Data validation failed',
            data: null,
            status: false
        })
    }
}

```

Error Handling in axios requests done using Http error codes and manual message response

```

const { error, isLoading } = useUser()

if (isLoading) return <div>Loading...</div>
if (error) return <div>{error.message}</div>

```

Handling error when User authentication is being detected

```

const notify='Sorry Facial Authentication server offline'
const validateFace = () =>{
    stopCapture()
    fetch("http://127.0.0.1:5000/faceValidation")
    .then(res =>res.text())
    .then((res)=>{
        setValName(res)
        setFaceValidated(true)
    })
    .catch(error=>{
        notification(notify)
    })
}

```

Specific Error notification

Handled error with notification

```

export const getPassword = async id => {
    try{
        let password = await faunaClient.query(
            q.Get(q.Ref(q.Collection('passwords'), id))
        )
        if (password.name === 'NotFound') {
            return 'Passwords for the user were not found!'
        }
        if (password.name === 'BadRequest') {
            return 'Sorry, there was a problem with the request, Database Server issue!'
        }
        password.data.id = password.ref.value.id
        return password.data
    }catch(error){
        return error.message
    }
}

```

DB Query

Individual problem errors with the DB query have also been handled with if-else

Try Catch error handled

```

<HIBPPasswordChecker password={inputPassword}>
    {({ initial, loading, error, pwncd, count }) => {
        if (initial) {
            return null;
        }
        if (loading) {
            return "Checking the uniqueness of this password...";
        }
        if (error) {
            return `error: ${error}`;
        }
    }
}

```

Error handling when using the HIBP API

- **Environment variables setup for sensitive data:** Another important major security focused development strategy that has been implemented is not hardcoding any sensitive data like sensitive keys, URL's, connection domains, Issuer URL's etc. All this sensitive data has been saved in a separate ENV file (as environment variables) which is referenced inside the source

code. This helps in keeping sensitive data isolated in a different environment so that there is no chance of exploiting this data in the extremely rare case of a breach, since it contains decryption keys, keys to connect to the database and much more important sensitive data that could help a third party in disrupting services the application provides etc. These environment variables are referenced using the `process.env` property which is inbuilt as an API of the process module; it is used to retrieve the user environment and its variables (In this project is used to reference all environment variables/keys from the `env.local` file).

```
APP_URL=http://localhost:3000
REACT_APP_FAUNA_KEY=
AUTH0_SECRET=
AUTH0_BASE_URL=http://localhost:3000
AUTH0_ISSUER_BASE_URL=
AUTH0_CLIENT_ID=
AUTH0_CLIENT_SECRET=
REACT_APP_SECRET_KEY=
AUTH0_AUDIENCE=
```

- **Secure Databases and querying** and connection requests: All database connection and queries that have been coded in the application are programmed keeping in mind the vulnerability of using a database i.e., they might be prone to injection attacks or data breaches which could lead to exploitation of sensitive user data. To protect from this; First the input being sent from the client side to the flask back-end is input validated at the client side and the python back-end (back-end server side as mentioned above) then parameterized and prepared SQL queries are used in the python code when they are called by the functions from the client side (triggered by an end-user) in the flask back-end server. The SQL queries are prepared and parameterized in python using placeholders ('?') instead of using direct data values in the queries itself i.e., a query with a place holder is setup first and then the actual data (validated input) is saved into a separate variable and is added to the query as a replacement for its placeholder at a later stage when the query is being executed. A major advantage of doing so is the parameter binding and character escaping is handled itself instead of having to manually escape characters (like backslashes, quotes etc.), character escaping helps to avoid any kind of injection attacks that might be targeted at this SQLite database. Further at the front-end, server less Next.js application is connecting to the Fauna Database and queries are written using FQL (Fauna Query Language). Since Fauna is a cloud native database and is used to create server less applications (No back-end framework required connecting to a database and making queries like done with Next.js), it has its own security models in place. A secret Fauna encrypted access key is required to connect to the database which strictly controls and defines access to the database. Fauna itself was designed while keeping in mind the secure by default principle, the database API does not assume trusting any kind of access, any application that connects to the database will connect over secure HTTP connection and with an encrypted access key (that defines access control). Fauna Database API (using FQL) is also type-safe by default thus making it impossible to be breached by Injection attacks. Finally Fauna also offers strict serializability which leads to ultimate isolation of transactions (queries).

```
checkQuery="""SELECT registrationName FROM registeredFaces WHERE registrationName=?"""
try:
    sqlCursor.execute(checkQuery,(registrationName,))
```

Value to be put in

Place holder

```

let newAddedPassword = await faunaClient.query(
  q.Create(
    q.Collection('passwords'),
    {
      data: {
        accountName,
        accountUrl,
        email,
        encryptedPassword,
        avatar,
        created_at: `${months[date.getMonth()]} ${date.getDate()}, ${date.getFullYear()}`,
        user: {
          id: user.sub
        }
      }
    }
  )
)

```

Fauna Query using FQL

```

let passwordsForUser = await faunaClient.query(
  q.Paginate(
    q.Match(q.Index('user_passwords'), id)
  )
)

```

FQL

```

# using prepared-parameterized query statements to avoid SQL injection added with input validation on front-end (next)
query="""INSERT INTO registeredFaces VALUES(?,?)""
registrationName=request.args.get("name")
list=[registrationName,encodedVal]
valForDB=tuple(list)
sqlCursor.execute(query,valForDB) # value prepared and added to db

```

Place holders

Values are prepared and then added

- Access Control implementation:** Maintaining and regulating access control is a must for a secure application and hence access to the application and all of its security assisted features is strictly regulated by authentication, even the connection to the Fauna Database API is strictly access controlled using a secret Fauna key used to connect to the database. The whole application cannot be accessed by any unauthenticated personnel that are not permitted to access the application. The trust boundaries are clearly and strictly defined as well i.e., Users can not view any other user's data ever all data is strictly separated to avoid any kind of data leaks. This is done both on the client side (facial validation) as well as the server and back-end (strict authentication and access regulated database queries) so the data cannot leak from any layer of the application.
- Static analysis Conducted:** Finally to have the best possible coding practises in accordance with Next.js standards and to avoid syntactical bugs that might lead to opening up security vulnerabilities in the application; these are fixed after conducting static analysis on the application using static analysis tools such as ESLint which show potentially helpful changes that are supposed to be made in the application (with warning, syntactical and error descriptions).

```

PS S:\CyberVault\cybervaultnext> npm run lint

```

Command to run static analysis

```

> lint
> next lint

info - Loaded env from S:\CyberVault\cybervaultnext\.env.local

./pages/analyser.js
58:25 Warning: Do not use <img>. Use Image from 'next/image' instead. See: https://nextjs.org/docs/messages/no-img-element @next/next/no-img-element
64:31 Warning: Do not use <img>. Use Image from 'next/image' instead. See: https://nextjs.org/docs/messages/no-img-element @next/next/no-img-element

./pages/dashboard.js
54:15 Warning: Effect callbacks are synchronous to prevent race conditions. Put the async function inside:

useEffect(() => {
  async function fetchData() {
    // You can await here
    const response = await MyAPI.getData(someId);
    // ...
  }
  fetchData();
}, [someId]); // Or [] if effect doesn't need props or state

```

Examples

Security Principles and Pernicious kingdoms incorporation

The code base of the application was programmed keeping in mind **secure coding principles** and **pernicious kingdom** vulnerabilities explained as follows:

- **CIA integration:** The application focuses on maintaining the security triad i.e., Confidentiality, integrity and availability of all resources and data. All data stored by the user is always kept confidential within strict trust boundaries and cannot be leaked, sensitive data like passwords are all encrypted and character masked. All user account data is confidential as well cannot be accessed by other user's (also involving IP throttling). Even accessing data saved by the user in the fauna DB (password-accounts) from the data requires an encrypted access key. The integrity of all data is maintained even during communication transit by the Database API (Secure HTTP connection requests) and the integrity of the authentication process and user account data (Using AES 128 bit TLS encryption). Finally all resources and data is rightly available to the authenticated and authorized user only.(owner of the password-accounts) DDoS protection is enabled by Auth0 as well to stop from overdoing unauthorized logins to disrupt authentication services.
- **Principle of least privileges** – The principle of least principles has been accounted for during the development of this project-application since no default admin privilege is given to any of the users, even the admin themselves cannot view user accounts and their sensitively saved data in the databases. There is no admin functionality available to be modified on the application itself hence no leaks of admin functionality are possible. No Unauthenticated user has any privileges regarding the application functionality since all routes and API requests have also been secured to detect and require a user sessions (http only Session-Cookie). During Signup even the user is prompted to grant the application access to their profile details (like username/email etc.).
- **Defence in Depth** – Defence in depth has been deeply incorporated within various parts of the application. This includes multiple data validation checks both on the client as well as the server and back-end side to protect against any kind of database targeted attacks such as injection attacks. Defence in depth also refers to the API requests being secured to only be made by authenticated users on top of authentication secured routes. Adding facial validation after a user is authenticated to view their passwords and manage them is also an example of defence in depth. Finally secure querying of databases and the use of Fauna Database API which has Defence in depth incorporated by default i.e., the requirement of the access-key to access the database (no default trust access to any client of any kind) is also an example of defence in depth being incorporated in the application.
- **Securing weak links** – Most commonly the weakest link in software is the end-user who might expose their sensitive data or application data by human error. This has been taken care off in the application by implementing default character masking, automatic Idle-timer logout (protects user data in public places), deep password encryption, use of environment variable and not exposing any important keys, even secured authentication (with choices for multifactor), forced creation of complexly strong passwords (user-account and password-accounts). No security concern has been left to the end user to

take care off and multiple features have been provided to the user to improve security if they are still concerned.

- **Secure by default** – Secure by default has been implemented in various code bits; for example – All the check boxes for the password generator are checked by default to generate a complexly strong password from the get go unless the user decides to choose otherwise (like some password requirement might be exactly 8 characters instead of 8 and more). The idle timer function by default automatically logs out the user after the prompt if it detects that the user is not responding to the prompt either. String input validation has been implemented by default for all possible scenarios where user input is required (including no blank inputs allowed to submit and by default disabled create buttons) to force the user to create strong and complex passwords with no incorrect format of data being submitted. The inputs that are only for display are disabled by default so the user cannot type in them and finally a password strength indicator is included by default in the add/edit password modals. The users are forced to create and store strong and complex passwords by default (even during signup).
- **Keep It Simple** – The application also follows the Keep It Simple principle since all the navigation and functionality of the application is straight forward and easy to follow by the end user. To maximize the user experience everything is straight forward and explained to the user along the way using tool tips and notifications. Appropriate redirect links are also provided throughout the application for ease of use.
- **Privacy** – Finally the application maintains privacy at every level and tries to keep all user-account and password-account information private at all times and secures the application to make it full proof against any kind of data leak. Thus protecting the users that decide to utilise the application to its maximum capability.

The application has also accounted for various security vulnerabilities that have been emphasized in the seven pernicious kingdoms, these include –

- **Strict Input validation** – Strict Input validation has been integrated throughout the application as has been mentioned above in the client-side, server-side and the back-end of the application which contribute to protecting the application from Injection attacks. Further measures to protect against SQL injection attacks have been taken as well including prepared and parameterized statements in the back-end (SQLite using placeholders) and the use of Fauna Database API which is also type-safe along with FQL to make queries. Further the use of Auth0 also pertains to the protection from common XSS attacks due to the involvement of the session cookie being *HttpOnly* (when a session for user is created, the cookie is used to recognize the session and help fetch and post user specific data).
- **Security features** – A wide variety of security features have been added in the application which has been mentioned in detail in the sections above which account for strict access control, secure password management and input validation including no intake of empty strings, powerful and different encryption depending where the encryption is used. Most features of this application automatically contribute to privacy and confidentiality as well.
- **Error handling** – The implementation of proper Error and exception handling has been mentioned in detail above along with how the error handling has been implemented and

in which sections catching errors/exceptions was very important such as database query execution code or connecting to a server to either receive or send data etc. The error/exception handling has been done in a proper way i.e., no catch or except blocks have been left empty and proper, appropriate and limited error messages have been manually declared within these exception/catch blocks.

- **Encapsulation** – Finally encapsulation has been maintained i.e., strict trust boundaries have been declared, with no intermixing of authenticated with unauthenticated data. Processing and access to and of only authenticated users and their authenticated data. Exception handling and trust boundaries and been securely and properly setup to avoid any kind of potential data leaks or breaches that might either open up security vulnerabilities or harm any user by sensitive data exploitation.

Causes of changes in technology stack and implementation

From the initial technology stack there were a few core technologies I decided to replace with a better more scalable and flexible technology which gave me a more variety and expansion on the way features could be implemented and this influenced the changes regarding the method and process of implementation. Some of these major changes include –

- i. Use of Express.js back-end replaced with use of Next.js – Initially it was planned and explained that for the back-end of the application; Express.js framework with React.js in the front-end would be utilised. This plan was altered to use the Next.js framework instead of Express and eliminate the need of two back-end engines. Further since the implementation of Fauna as the main database was decided the opportunity of making the application (at least the manager part) could be made server-less with the utilisation and better match-up of Next.js and Fauna DB. Since Next.js framework can act as a front-end middleware combination by setting up the *Axios* and API connection in the Next.js provided API folder which could then with the use of the Fauna Database key be able to query the fauna DB directly. This would also make the use of a python flask back-end more efficient and increase the speed of processing and improve the communication (between front-end, acting server and back-end) throughout the application.
- ii. Use of the Crypto.js Module – Initially it was decided a separate JavaScript module known as Crypto.js would be used for handling password encryption etc. But through thorough testing of the password *encryptor*; bugs were found in the module's latest version and a few recent versions before it. The use of inbuilt crypto module was decided due to its better efficiency and clearly described and easy to understand code and implementation which resulted with the same required outcomes.
- iii. Fauna Database instead of simple SQL – For the password manager and session management Fauna DB API was used instead of simple SQL due to Fauna's ability to help render server-less applications and it's amazing suite of in built security infrastructure. It was also due to its amazing pairing with Auth0 to improve the security surrounding user and their data even further. The security elevation along with non-functional requirements such as robustness, Speed of query request execution, pairing with Auth0 (API request authentication security). Further factors such as better control over database, Database individual security such as transit encryption, fauna key requirement to make DB requests etc.

- iv. Use of SQLite 3 in the back-end instead of simple SQL – Lastly another difference was the change in database from SQL to SQLite 3 in python back-end was done due to the ease of use that comes along with SQLite 3 including its self-management of Database files and easy of creating parameterized and prepared statements to protect against potential injection attacks.
- v. Using React functional components instead of class components – Initially during the midpoint demonstration the facial registration and validation code were coded as class components, But when the facial code and flask back-end were integrated into the main project they were converted into functional components following the same logic, due to the wider flexibility provided by react functional components like the use of hooks which helped improve these functions and how they look on the main GUI.

Testing & Evaluation

Very thorough testing has been conducted in the application with various methods of testing that are utilised. The main software used for testing this application was **Cypress** which is a testing framework for JavaScript based applications like applications built with Next.js or React.js (or any other JavaScript based framework), even applications containing a back-end developed with the Express.js framework etc. Cypress helps in testing an application in an artificially created and isolated environment which is a chrome tab that is controlled and regulated by the cypress framework and is a copy of chrome but with faked and mocked settings etc. They help in performing automated tests with sample data to conduct all kinds of testing. In this case a test user is used and is logged in before each test (Is logged in the home page test during testing). Different methods of testing that has been utilised in this application involve **Unit testing, Integration testing, End to End testing** and finally **manual user testing cases**. In total there are **32 tests -> 27 Unit + Integration test & 5 End to End tests**.

The main sequence and process that was followed during testing was as follows; starting off with testing the **rendering** of each component in isolation for a single page or component at a time. This involved multiple assertions as the page was rendered to confirm the display of components is as intended and contains all respective JavaScript components. An example would be; testing the rendering of the password manager dashboard page. When the page is rendered for the first time it should include 3 sections i.e., an introduction section; which should contain HTML headings (h2) containing some text then a button for add password-account should exist beneath the introduction section with the text 'add account password'. Finally a webcam component should exist in the next section with a button beneath it and so on.

After the rendering test of the page was passed, each of the main functions present on the page were **unit tested** one by one. It is to be noted that each test that is conducted in cypress is isolated unless the test code is grouped together under one test (Tests start with **"it ('should description of test), () => {test code}"** after the context). Unit testing was done after the render tests. Unit tests involved testing a functionality component individually for example in continuation to the previous example; testing the add password functionality (after its render test is done) would include redirecting to the URL, clicking the add button then confirming the opening of a popup modal for adding password. Then individual tests would be conducted to show the create button would be disabled when the input validation conditions are not passed after entering the details (each input condition was tested). Then a unit test is written to show what happens when all conditions pass, this would include the colour of the conditions (statements) changing to green (success text) and the

enabling of the create button and clicking that should close the modal and show a notification on the screen for a small time notifying the user that their password-account was added. Later during the testing of the manager (which is displayed after face validation) the added password would be displayed in the table and its display render would be asserted to confirm it was added. (All user input is sample data that is entered automatically during the test run, this is after it is included in the test code using `.type(input)`).

After multiple of such unit tests are conducted for all functionalities one by one on the page files individually and in isolation, **integration testing** is conducted. Integration testing involves a combination of functionalities being performed and tested together under one test. An example of continuing from the last example would be to test the Edit and delete password account functionality in one test with their rendering also being tested at the same time. The sequence of this test would include the rendering test of the preview modal popup (all its components and the account and password name exist and are visible in it as well) and the edit modal popup so it is confirmed all parts of their modal are rendered (such as input validation conditions are rendered on the screen along with the disabled edit button, all input fields are rendered and are empty with placeholder texts, button only enabled when the validation conditions are passed). Immediately after the render tests; the test code is written to automatically enter data that passes validation and the edit button is clicked after it and an assertion for a notification popping up is also conducted. Continuing in the same test the preview modal is opened again and this time the delete button is clicked after which the notification display for that is tested and immediately after the display of the page is tested to confirm no more passwords exist since one password was created, edited and deleted. Thus completing the integration test and confirming the functional components work together.

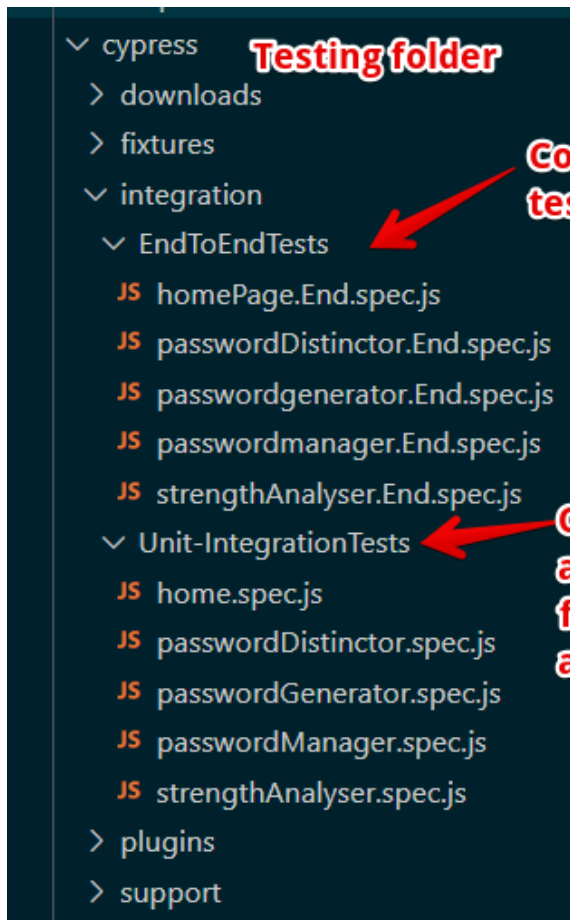
Finally End to End tests are conducted. These are tests designed in such a way that they mimic user movement and flow on a page while trying out all functionalities after the page renders. This test demonstrates how a user would go through a page and utilise all of its functionality inside one test by mimicking user movement for the whole page. It's a combination of all integration tests for a page under one test to maintain a continuous mimicked user flow and testing the application in this way. It will include the assertion of the render then all the functionalities within one end to end test.

The overall structure of the folder is two testing folders inside the integration folder of cypress. One folder contains all end to end tests for each main functionality page and the second folder contains test files that include the unit and integration tests in the files for each page.

```
PS S:\CyberVault\cybervaultnext> npm run cypress
```

```
> cypress  
> cypress open
```

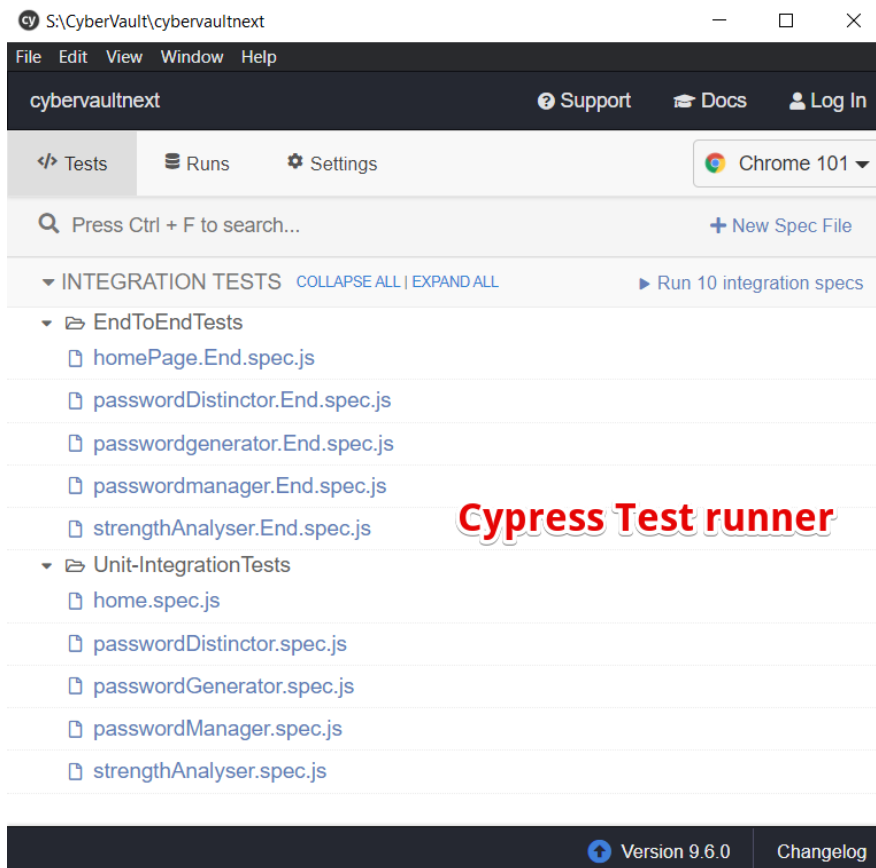
Command to run test runner



Testing folder

Contains all End-End tests for all pages

Contains Both Unit and Integration tests for all functionality and rendering



Cypress Test runner

Chrome is being controlled by automated test software.

Tests: **32** ✓ -- ✗ -- 116.70

All Specs

- Password generator page UI render test
 - ✓ Should render all of generator page's components
- Password generator functionality test
 - ✓ Should be able to generate a password which can be copied and should notify according to the situation
- The password manager link redirection test
 - ✓ Should visit the password manager dashboard page
- Password manager introduction section UI render test
 - ✓ It should render all of the password manager's introduction components as intended
- The password manager should contain another section which should have the webcam component before face is scanned
 - ✓ Should contain the webcam component with the scan face button
- Adding a password test
 - ✓ Should render the create password modal popup with all its components as the add password button is clicked

All End to End, Unit and Integration tests were run together

Chrome is being controlled by automated test software.

Tests: **5** ✓ -- ✗ -- 41.06

All Specs

- End to End test for the home page including the login process and the dynamic updating of the navbar consequently
 - ✓ Should conduct an End to End test on the home page before login and after login to test the dynamic navbar update and all of its components including testing the navigation bar's all navigation links
- Password Distinctor End to End test
 - ✓ Should test the whole Password distinctor page and conduct an End to End test
- End to End test for the password generator page
 - ✓ Should test the whole Password generator page and conduct an End to End test
- End to End test of the password manager page
 - ✓ Should test the whole Password Manager page and conduct an End to End test
- Strength analyser End to End test
 - ✓ Should test the whole Password Strength Analyser page and conduct an End to End test

All End to End tests were run

Each End to End test contains multiple assertions inside

localhost:3000/_/#/te **All Unit & Integration Tests were run**

Chrome is being controlled by automated test software.

Tests **27** 74.93 http://localhost:3000/dashboard

All Specs

- Before Login, Navigation bar should have Login/Register link and Home page should render with all components
 - It should have the Login/Register Link
- After Login, Navigation bar should update and Home page should render with all components
 - It should Login and the navigation bar should be updated according to user session
 - Should have working navigational links for all functionality pages
- The password distinctior link redirection test
 - Should visit the password strength analyser page
- Password distinctior component UI render test
 - should render all the password distinctior page's components
- Password distinction/uniqueness test
 - Should test the password entered and respond with a dynamic message of whether the password has been breached before and if it has then how many times using the Troy Hunt API
 - Should show a massive number of breaches for a very simple password like 12345678
- The Password generator link navigation test
 - Should visit the password generator page
- Password generator page UI render test
 - Should render all of generator page's components
- Password generator functionality test
 - Should be able to generate a password which can be copied and should notify according to the situation
- The password manager link redirection test

These are all Unit and integration test

cybervaultnext localhost:3000/_/#/tests/_all

Chrome is being controlled by automated test software.

Tests **27** 77.06 http://localhost:3000/dashboard

All Specs

Adding a password test **One Unit test**

- Should render the create password modal popup with all its components as the add password button is clicked
 - BEFORE EACH (1)
 - BEFORE EACH (2)
 - BEFORE EACH (3)
 - BEFORE EACH (4)
 - BEFORE EACH (5)
 - TEST BODY
 - 1 visit /dashboard
 - (xhr) GET /api/passwords
 - 2 findByRole button, {name: /Add Password-Account/1}
 - (fetch) GET 200 /api/auth/me
 - 3 -click {force: true}
 - 4 get .show-grid
 - 5 -assert expected <div.show-grid.modal-body> to be visible
 - 6 get form > :nth-child(1)
 - 7 -assert expected <div.row> to contain text Social/Business/Personal Account and its URL
 - 8 get #validationFormik01
 - 9 -assert expected <input#validationFormik01.form-control.form-control> to be enabled
 - 10 -assert expected <input#validationFormik01.form-control.form-control> not to have value undefined

Each Unit & Integration test contains multiple Assertions

Each test includes multiple assertions

Each test runs automatically

```

JS strengthAnalyser.End.spec.js ×
cybervaultnext > cypress > integration > EndToEndTests > JS strengthAnalyser.End.spec.js > context('Strength analyser End to End test') callback > it('Should test
1  /// <reference types="cypress" />
2
3  context('Strength analyser End to End test',()=>{
4    it('Should test the whole Password Strength Analyser page and conduct an End to End test',()=>{
5      cy.viewport(1920,1080)
6      cy.visit('/analyser')
7      //Page rendering tests
8      cy.get('.navbar').should('be.visible')
9      cy.get('.Home_analyser_UZXIM').should('be.visible')
10     cy.findByRole('heading', { name: /robust password strength analyser/i})
11     cy.get('.my-5').should('contain.text','Robust Password strength analyser')
12     cy.findByRole('heading', { name: /analyse the strength of your password efficiently and be rest assured-/i})
13     cy.get('h5').should('contain.text','Analyse the strength of your password efficiently and be rest assured')
14     cy.get('h5 > .mr-2').realHover().should('be.visible')
15     cy.get('.form-control').should('be.enabled').should('not.have.value')
16     cy.get('.progress-bar').should('not.be.visible')
17     cy.get('h3.text-center').should('contain.text','Very Weak, easily crackable')
18     //Testing functionality
19     cy.get('.form-control').type('Password')
20     cy.get('h3.text-center').should('have.css','color','rgb(0, 0, 0)')
21     cy.get('.progress-bar').should('not.be.visible')
22     cy.get('h3.text-center').should('contain.text','Very Weak, easily crackable')
23     cy.get('.form-control').clear().type('Password123')
24     cy.get('h3.text-center').should('have.css','color','rgb(234, 17, 17)')
25     cy.get('.progress-bar').should('be.visible')
26     cy.get('h3.text-center').should('contain.text','Weak-ish still')
27     cy.get('.form-control').clear().type('Sk1jmn@')
28     cy.get('h3.text-center').should('have.css','color','rgb(255, 242, 0)')
29     cy.get('.progress-bar').should('be.visible')
30     cy.get('h3.text-center').should('contain.text','Its okay-ish')
31     cy.get('.form-control').clear().type('Sk1jmn@12')
32     cy.get('h3.text-center').should('have.css','color','rgb(102, 255, 0)')
33     cy.get('.progress-bar').should('be.visible')
34     cy.get('h3.text-center').should('contain.text','Better now...')
35     cy.get('.form-control').clear().type('Sk1jmn@12@as')
36     cy.get('h3.text-center').should('have.css','color','rgb(0, 255, 94)')
37     cy.get('.progress-bar').should('be.visible')
38     cy.get('h3.text-center').should('contain.text','Perfectly Strong & Complex!')
39   })
40 })
  
```

End to End Example

Multiple assertions throughout

Automated typing input throughout

```

context('The password manager\'s password manager section should show after face has been verified, they can then be Edited or Deleted',
  it('Should display the password manager component', ()=>{
    cy.visit('/dashboard')
    cy.wait(10000) //verify face during
    cy.get('.p-3').should('be.visible')
    cy.get('.pt-2').should('be.visible').should('contain.text','face verified, List of your passwords and accounts are as follows:')
    cy.get('.col-sm-1 > .rounded-circle').should('be.visible')
    cy.get('.col-sm-12 > .btn').should('be.visible').should('contain','Spotify')
    //open PASSWORD PREVIEW
    cy.get('.col-sm-12 > .btn').click()
    cy.get('.modal-header').should('contain','Account: Spotify')
    cy.get('.container > :nth-child(1) > .col')
    //verify added email
    cy.get(':nth-child(2) > .my-1').should('contain.value','testUser@gmail.com')
    //view password from PREVIEW
    cy.get('.text-left > span > .svg-inline--fa').click()
    cy.get('.col-md-9 > .form-control').should('contain.value','SpotsTest@1423')
    //COPY password
    cy.get('.text-right > span > .svg-inline--fa').click()
    //EDIT password test
    cy.get('.modal-footer > .btn-primary').click()
    cy.findByText(/edit password for spotify/i).should('contain','Edit Password for Spotify')
    cy.get('.btn-success').should('contain','Edit').should('be.disabled')
    cy.get('form > :nth-child(1) > :nth-child(2) > .form-control').clear().type('Spotify123')
    cy.get(':nth-child(3) > .form-control').clear().type('amazon.com')
    cy.get(':nth-child(3) > .col > .form-control').clear().type('testUser123@gmail.com')
    cy.get(':nth-child(2) > .col > .form-control').clear().type('MySpotifyTest@1234')
    cy.get('.btn-success').click()
    cy.get('.Toastify__toast-body').should('contain.text','Password edited successfully!')
    cy.get('.col-sm-12 > .btn').should('be.visible').should('contain','Spotify123')
    //DELETE the password
    cy.get('.col-sm-12 > .btn').click()
    cy.get('.modal-header').should('contain','Account: Spotify123')
    cy.get('.btn-danger').should('contain','Del').click()
    cy.get('.Toastify__toast-body').should('contain.text','Password successfully deleted!')
    cy.get(':nth-child(4) > .my-5').should('contain.text','You do not have any passwords added!')
  })

```

Integration Test example

```

it('Should add the password if all input validation is passed', ()=>{
  cy.visit('/dashboard')
  cy.get('.Home_welcome_FFAPx > :nth-child(3) > .btn').click({force:true})
  cy.get('#validationFormik01').type('Spotify')
  cy.get('form > :nth-child(1) > :nth-child(2) > :nth-child(2)').should('have.css','color','rgb(25, 135, 84)').should('contain','Account normal String')
  cy.get('form > :nth-child(1) > :nth-child(2) > :nth-child(3)').should('have.css','color','rgb(25, 135, 84)').should('contain','Account No special chars')
  cy.get('#validationFormik02').type('spotify.com')
  cy.get(':nth-child(1) > .text-success').should('have.css','color','rgb(25, 135, 84)').should('contain','Url should be valid')
  cy.get('#validationFormik03').type('testUser@gmail.com')
  cy.get(':nth-child(3) > .text-success').should('have.css','color','rgb(25, 135, 84)').should('contain','Email should be valid')
  cy.get('#validationFormik04').type('SpotsTest@1423')
  cy.get(':nth-child(5) > :nth-child(6)').should('have.css','color','rgb(25, 135, 84)').should('contain','Should be greater than 8 characters')
  cy.get(':nth-child(5) > :nth-child(7)').should('have.css','color','rgb(25, 135, 84)').should('contain','Should contain an upper case letter')
  cy.get(':nth-child(5) > :nth-child(8)').should('have.css','color','rgb(25, 135, 84)').should('contain','Should contain a lower case letter')
  cy.get(':nth-child(5) > :nth-child(9)').should('have.css','color','rgb(25, 135, 84)').should('contain','Should contain a number')
  cy.get(':nth-child(5) > :nth-child(10)').should('have.css','color','rgb(25, 135, 84)').should('contain','Should contain a special character')
  cy.get('.btn-success').should('contain.text','Create').click()
  cy.get('.Toastify__toast-body').should('contain.text','Your password has successfully been added!')
})

```

Unit Test example

Further some manual user testing was also conducted in the application. This was done by letting my fellow classmates try and interact with the application to make use of the feature suite in a way a normal end user or in some case a pen tested would. This revealed some bugs initially that were fixed along the way, this way of testing came to be very useful. Some bugs that were found with how they were fixed are as follows –

- i. Missing Exception handling in regards to the facial authentication server. This bug was revealed as a result of forgetting to run the flask server before actually conducting facial authentication which resulted in a 404 default error that is given by the java script framework which incidentally also reveals stack trace information. This was fixed by implementing proper error handling (using notifications) in the fetch requests that were included in the dashboard code.

```

const notify='Sorry Facial Authentication server offline'
const validateFace = () =>{
  stopCapture()
  fetch("http://127.0.0.1:5000/faceValidation")
  .then(res =>res.text())
  .then((res)=>{
    setValName(res)
    setFaceValidated(true)
  })
  .catch(error=>{
    notification(notify)
  })
}

```

Specific Error notification

Handled error with notification

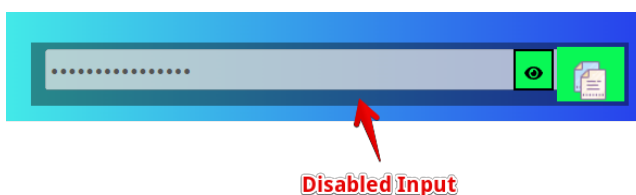
- ii. Input validation not conducted for the account name variable – During manual User testing when one of my classmates tried to put bogus information in the account name it got saved in the database. The account name was validated on the server side to ensure it was a string but was not input validated on the client side. This was then implemented as soon as the bug was discovered. The client side validation parameters were made very specific as to not allow any account name with special characters or any type of SQL DDL or DML commands in it. This secured the account name input as well.

```

const handleAccountNameOnChange=(e)=>{
  setAccountName(e.target.value)
  const hasSpecialChar=/[!,@,#,$,%,&,*,>,<`,`~>?=,+,-,','"/.test(e.target.value)
  const regex = /\bSELECT|FROM|UPDATE|DELETE|INSERT|CREATE|DROP|ALTER|select|from|upd
  const hasSQLWords=regex.test(e.target.value)
  setAccountNameErrs({hasSpecialChar, hasSQLWords})
}

```

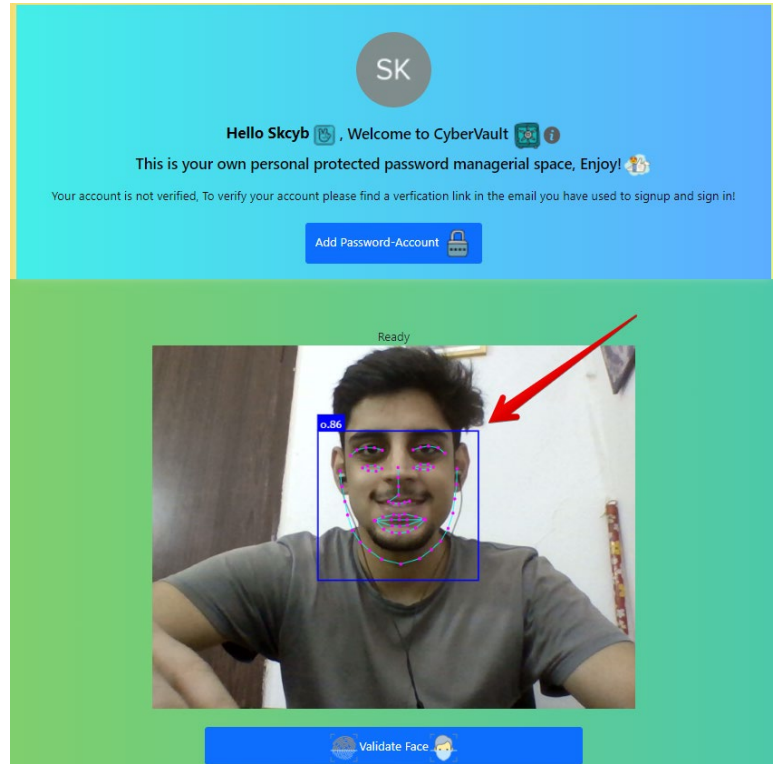
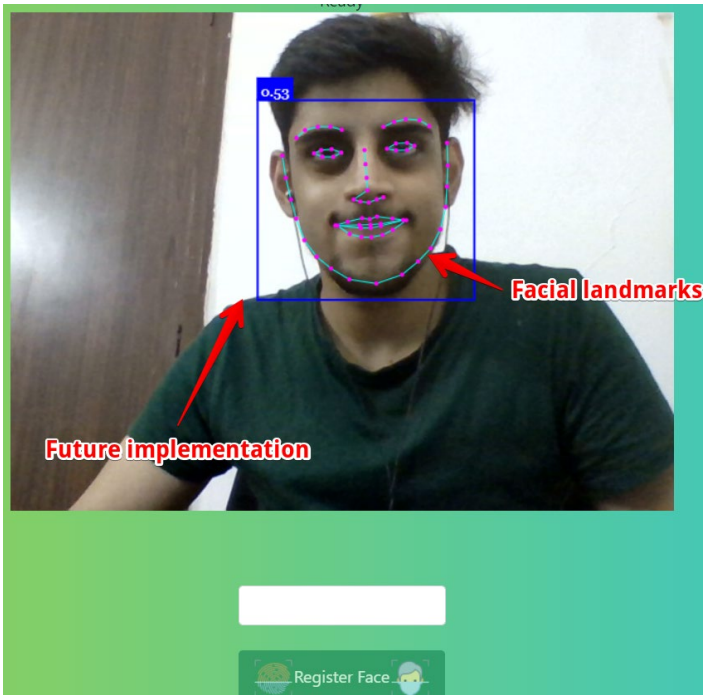
- iii. One more security loop hole found was the missing character masking in the password generator. It generated the proper passwords but they were not character masked and were displayed in the input column in plain text. This went against the principle of secure by default and was hence rectified.



Future Development or research

The future opportunities for this application are limitless; the security around the application could be increased a few times like offline implementation and execution etc. Further the facial ID could be utilised with even more technologies and sensors that detect eye movement and are adapted to masks, different options for multifactor using voice recognition, retina scans etc. Another thing could be to improve User experience even more, throughout the flow of the application as well. Some Future planned improvements include -

- FaceAPI.js related development - During the time of this application's development, future development aspects were also tried and tested but not finalised and hence have been added in the issue section of GitHub such as utilising facial landmarks to render them while the user is trying to validate and register their face so even the end-user can see how facial scans are displayed and scan the face. These could be done using Artificial intelligence integrated libraries like FaceAPI.js to show these landmarks and facial recognition borders on live webcam as can also be seen below.



- File uploading and security – Another potentially useful feature for the future of the application would be to develop a universal documentation format vault including storage for important financial cards and their details that are protected using top grade transit encryption and official security certification from respective banking institutions. This could also lead to a payment service that could be utilised directly from this manager application (which would also be protected by higher grade facial identification).
- Introduction into the Mobile market – Finally another future development option for the application would be to introduce it into mobile markets i.e., android and eventually IOS. These would expand the authentication and verification options to include fingerprint scanning and would include technologies like flutter and React Native for mobile development. This would help give the application an even wider audience base. Since end users might prefer to save and retrieve all this important data from the palm of their hands instead of having to log on to a computer every time. This application would then be available in the android play-store and the IOS app-store.

Conclusion

One of the biggest challenges faced during the development was integrating the facial authentication validation functions that I had developed before the midpoint submission with the final implementation of the password manager built using Next.js, since these were purely developed with a react – front-end in mind (which was the decision at the time) they used older components like React class components (which were not compatible with hooks) and were tailored to fit only react style classes. Even the flask python server back-end was configured to connect with the react front-end. This issue was solved gradually by converting the pure react class component to a functional component and instead of proxying the requests through the *package.json* file they were directly sent using the flask host link. The react functional component was then included in the Next.js developed manager dashboard and the facial registration page. The Back-end was then slowly improved as well to include input validation, efficiently integrate with the Next.js front-end, include proper exception handling etc. Another issue was the implementation of secure authentication while also tracking user sessions. This was initially done by developing a input validated login registration page which also accounted for the flask back-end and was using the API folder in Next.js to send requests to the fauna DB. Then Auth0 with its superior security infrastructure was discovered and researched, and due its impeccable pairing with fauna-DB it was integrated into the application and a better security was incorporated in surrounding the application. One last challenge that persisted throughout the development was the challenge of using a newer technology like Next.js which had a new way of implementation even slightly different than its predecessor (React). But due to its seamless flexibility and the ability to serve server-less applications when paired with a database like fauna and its compatibility with newer and older libraries it was concluded to be right choice for developing this application.

References

- [1]. Anderson, C., 2018. *Data Security in the Age of Serverless Apps*. [online] Fauna. Available at: <<https://fauna.com/blog/data-security-in-the-age-of-cloud-native-apps>>.
- [2]. Bachina, B., 2019. *How to Implement Idle Timeout in React*. [online] Available at: <<https://blog.bitsrc.io/how-to-implement-idle-timeout-in-react-830d21c32942>>.
- [3]. Casagrande, A., 2020. *Two-factor authentication flow with Node and React*. [online] Medium. Available at: <<https://medium.com/onfrontiers-engineering/two-factor-authentication-flow-with-node-and-react-7cbdf249f13>>.
- [4]. De Rooms, B., 2020. *What Is Fauna, and How Does It Work With Auth0?*. [online] Auth0 - Blog. Available at: <<https://auth0.com/blog/what-is-fauna-and-how-does-it-work-with-auth0/#How-Does-Fauna-Fit-in-My-Application->>.
- [5]. Emadamerho Atori, N., 2020. *Authenticating React Apps With Auth0 — Smashing Magazine*. [online] Smashing Magazine. Available at: <<https://www.smashingmagazine.com/2020/11/authenticating-react-apps-auth0/#:~:text=You%20can%20connect%20any%20app,data%20back%20to%20your%20app.>>>.
- [6]. Geitgey, A., 2017. *Face Recognition — Face Recognition 1.4.0 documentation*. [online] Face-recognition.readthedocs.io. Available at: <<https://face-recognition.readthedocs.io/en/latest/readme.html>>.
- [7]. Guevara, H., 2021. *Username and Password Authentication*. [online] Auth0 - Blog. Available at: <<https://auth0.com/blog/username-password-authentication/>>.
- [8]. Patel, H., 2020. *Create your Own Face Recognition Authentication System using Python, Computer Vision, and Machine....* [online] Medium. Available at: <<https://becominghuman.ai/create-your-own-face-recognition-authentication-system-using-python-computer-vision-and-machine-7bcf3aea6c70>>.
- [9]. Raboy, N., 2020. *Test Password Strength with RegEx in a React Application*. [online] The Polyglot Developer. Available at: <<https://www.thepolyglotdeveloper.com/2020/02/test-password-strength-regex-react-application/>>.

- [10]. Rosebrock, A., 2018. *Face recognition with OpenCV, Python, and deep learning - PyImageSearch*. [online] PyImageSearch. Available at: <<https://pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/>>.
- [11]. Simplilearn, 2021. *What is SQLite And When to Use SQLite*. [online] Simplilearn. Available at: <<https://www.simplilearn.com/tutorials/sql-tutorial/what-is-sqlite>>.
- [12]. Stillson, B., 2018. *Let's Encrypt Files With Node*. [online] Medium. Available at: <<https://medium.com/@brandonstilson/lets-encrypt-files-with-node-85037bea8c0e>>.
- [13]. Weaver, E., 2021. *Postgres vs Fauna: Terminology and features*. [online] Fauna. Available at: <<https://fauna.com/blog/compare-fauna-vs-postgres>>.
- [14]. Cypress Documentation. 2022. *Writing and Organizing Tests | Cypress Documentation*. [online] Available at: <<https://docs.cypress.io/guides/core-concepts/writing-and-organizing-tests#What-you-ll-learn>>.
- [15]. Chinda, G., 2019. *How To Build a Password Strength Meter in React | DigitalOcean*. [online] Digitalocean.com. Available at: <<https://www.digitalocean.com/community/tutorials/how-to-build-a-password-strength-meter-in-react>>.
- [16]. Goudar, S., 2021. *Build A Password Generator with React JS - react-toastify - Beginners Tutorial*. [online] DEV Community. Available at: <<https://dev.to/somanathgoudar/build-a-password-generator-with-react-js-react-toastify-beginners-tutorial-4a9n>>.
- [17]. Herrera Itie, R., 2021. *Build a Face Recognition Web App with React and Flask*. [online] Medium. Available at: <<https://python.plainenglish.io/how-to-deploy-a-face-recognition-web-app-with-flask-react-from-zero-c4c624654646>>.
- [18]. Tuduo, V., 2021. *Build a Code Snippet Web App with Next.js and FaunaDB*. [online] SitePoint. Available at: <<https://www.sitepoint.com/nextjs-faunadb-build-code-snippet-app/>>.
- [19]. Arias, D., 2021. *Adding Salt to Hashing: A Better Way to Store Passwords*. [online] Auth0 - Blog. Available at: <<https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/#Simplifying-Password-Management-with-Auth0>>.
- [20]. Di Mattia, S., 2021. *How to Authenticate with Next.js and Auth0: A Guide for Every Deployment Model*. [online] Auth0 - Blog. Available at: <<https://auth0.com/blog/ultimate-guide-nextjs-authentication-auth0/#Next-js-Serverless-Deployment-Model>>.

Appendices

Project Proposal

Objectives

I plan on developing the ultimate one stop security vault web application for my project; *CyberVault* which will help customers/ users safely secure all their important assets (files and documents) and credentials (plethora of online passwords) in one place under optimum security without having to worry about being exploited or the credentials getting stolen.

The existence of such an application is paramount for customer security, appeasement and satisfaction, since in the modern day and age, users have a plethora of important passwords and documents pertaining to their daily lives with no way of securing and managing them all in one place. They are never really comfortable with the amount of security and protection surrounding these assets and credentials. The application will be focused on the non-technical populace with a serious concern for security and ease of management of all their important documents and passwords.

This project sets out to achieve and provide:

1. Maximum customer satisfaction: By protecting all important customer assets in one location under high level security parameters like multiple encryption techniques and facial recognition authentication with multi factor authentication for login and registration etc. Including helpful features for customers like automatic password generation and strength analysis for their ease.
2. Customer Appeasement and reassurance: Since the application would act as a third party personal security contractor for customers by keeping their entire collection of important stuff safe and helping them by providing extra information about all the things they decide to store
3. Sense of security: Providing the customer's with a sense of security regarding their personal credentials and important documents since all their assets will be under heavy security features and they won't have to worry securing and managing all their passwords by themselves.
4. Ease of management: Since the application will have full managerial functionality regarding all the important assets and credentials stored in them to help the user manage them with ease.
5. Avoiding unnecessary stress: The application would help get rid of the unnecessary extra stress of remembering and managing all password and worrying about the security of all credentials and documents by doing it all for the customer.

Background

Nowadays each and every website present on the internet requires a user to properly create a profile and register themselves before they can read or make use of the content of said website/web application. This in turn creates an excess of pressure on the user to come up with yet another complex password to remember for every new website they visit whether to use its data or too

access its functionality. This situation adds another burden on the user to remember the whole plethora of passwords they use for multiple website/web-applications, which is why user's then turn to excel sheets or write them down on paper to remember this vast majority.

These methods of remembering password are not secure at all, since anybody could access an excel sheet full of important passwords, or a piece of paper containing the same and completely exploit the user, whether it be for ransom or stealing information etc. This is a major threat to a customer's personal security which arises due to such a situation. I plan on solving this problem that the user's face by building this application i.e., the ultimate solution to satisfy their concern for security and protect them against exploitation and cybercrime with the best possible security features for storage and management. To meet all the objectives described in the first section the application will include and provide:

- A high level (Vault) security storage solution for all their password and important documents i.e., a fully functioning security password and document managerial application
- Improved and high level security features that none other similar application provides like:
 - Facial recognition authentication for login and registration administration added on multifactor authentication using deep learning.
 - High level encryption techniques like crypto graphical encryptions; AES-128, AES-192, and AES-256 etc., blowfish encryption, Hashing done using external encryption modules for credential security on the back-end and front-end.
 - Options for document storage under encrypted locks for asset security with individual text file encryption option.
- Other general features to ease the customer's security and satisfactory concerns such as:
 - Password distinctior to check the uniqueness and vulnerability of passwords by cross referencing them through the *have I been pawned* website database using their API
 - A password strength analyser to determine the strength and complexity of each password that is being stored in the vault. This would notify a user if a weak password is being saved.
 - Automatic password generator tailored according to user specifications to generate strong complex passwords.

State of the Art

There are multiple password managing applications that already exist like LastPass, Keeper, BitWarden etc. This web-application stands out completely since none other web applications go the extra mile to ensure and guarantee the security and customer satisfaction that this application will provide with its added security and safety attributes. This application completely focuses on trying to provide every possible high security features aiming for maximum customer satisfaction and reassurance regarding their sense of security.

Most password managers in existence have the same basic idea to store all passwords under a master password login, this application takes a different approach to ensure its purpose since it is combined with deep-learning to include a facial recognition authenticaton added on its multifactor authentication to secure the login process completely with no risk of exploitation due to the master login password being stolen which is the major disadvantage of any password manager that already exists.

It takes the extra mile by providing all the features that could help ease the management process for the User, even these extra features that the application provides have their own security criteria regarding the functionality they provide which is different from all other password managers for example the automatic password generator will contain each and every condition required to generate a highly complex password that would satisfy the conditions of any web application that requires complex passwords for their security and features like the password distinctior to warn or notify the user if a password has been compromised and is unsafe to use etc. General password managers also differ from secure file/document managers but this application combines and provides that functionality in one place with a lock over the document vault with multifactor authentication to even help users secure all their important documents.

Technical Approach

The scope of the project is to implement high level security features to ensure the safe storage of all important user passwords and documents in one place with the addition of multiple features to help user manage their assets and credentials with ease.

Regarding the development approach for this project I plan on taking the Waterfall approach or following the waterfall model. The waterfall model of project development includes:

1. **Requirement procurement and analysis:** The requirement procurement and analysis requires identifying all types of requirements before developing the project so the project can be developed in accordance to these requirements. The requirements include functional and non-functional requirements such as:
 - a. Main functional requirements pertaining to this project:
 - i. **Secure Login and registration:** A fully secure login and registration feature for each user including facial recognition authentication using deep learning to cover for the disadvantage of losing master key or the master key getting stolen etc. The login would also include multifactor authentication for covering up any faults that might lead to insecure logins.
 - ii. **Fully secure password storage:** So a vault to store all passwords that the user has under different and high level encryptions (AES-128, AES-192, and AES-256, blowfish encryption etc.) utilising external modules provided by node js, to encrypt passwords using cryptographic encryption techniques with hashing techniques utilised in the back-end for login to keep login/registration passwords secure in the database as well.
 - iii. **Password management functionality:** Interactive UI made using ReactJs to ease password management for the user including functions like password generator containing all password requirement complications to generate complex passwords that can be used by the user for multiple other applications that require complex and secure passwords.
 - iv. **Password distinctior and strength analyser:** For ensuring and reassuring the user about their passwords, These feature would help analyse existing stored password, The distinctior would analyse a password and cross-reference it with multiple other passwords that have already been compromised by utilising the have I been pwned website API to check all the passwords vulnerability. The strength analyser would determine if a pre-existing password is complex enough to not be broken by brute force cracking attempts or any kind of authentication bypassing attempts.

- v. **Secure File storage and management:** Finally an option for a secure file storage, where the user can upload files and store them under a lock and key, if enough time I would also like to implement individual text file encryption to safely secure all text documents.
- b. Non-functional requirements or attributes pertaining to this project:
 - i. **Security:** The main idea of this application is to provide a sense of security by protecting all important credentials and assets. Hence, this is one of the most important non-functional requirement that my project is meant to satisfy by keeping all inputs sanitised, by ensuring all the features mentioned for login and registration administration, by development through secure design principles and according to certain design patterns to fit the application during development.
 - ii. **Scalability:** The application is being built using React and node JS in the express environment i.e., SERN stack (for wide and accessible range of versatility and scalability), since these Libraries and runtime environments ensure ease of scalability in regards to future distribution and development of the application.
 - iii. **Reliability:** Since the application's main focus is security is meant to be highly reliable to give maximum customer satisfaction and ease of management. To make the users want to use it more as an integral part of their lives.
 - iv. **Data integrity:** The main purpose of the application is password management and security so the data integrity is a must, all user information as well all their important credentials will be encrypted and non-accessible by any exterior source, not until the person trying to access them is properly verified and authenticated.
- 2. **System design and architecture:** The application will be designed in accordance to meeting all the requirements mentioned above, the front-end which will be visible to users will be developed using React (Highly scalable JavaScript library), and will be focused on ease of management and interaction with the user thus making it easily accessible.
- 3. **Implementation & development:** The application will be coded using the SERN stack i.e., React in the front-end and Node-JS/Express server in the back-end using visual studio code and implementing external modules and libraries like BCrypt, CryptoJS and the flask server (utilising python scripts) for deep learning implementation.
- 4. **Integration and testing:** Unit testing will be implemented during and after the development to ensure each functional requirement mentioned, works according to its purpose and that the application meets the non-functional requirements as well. Integration testing will also be implemented after successful unit testing is done to check the application's scalability, working after deployment and functional as a whole with all working individual components.
- 5. **Deployment and Maintenance:** For the deployment purposes the application will be deployed using either Heroku, digital-ocean or netlify which are the most compatible deployment options for React/Node applications, and to ensure all the non-functional requirements it will be further re-monitored and maintained.

All functional requirements are basically the functions that the application is supposed to perform as to fulfil its purpose of satisfying the customer and acting as their third party security contractors, these functional requirements will be broken down into individual tasks and milestones will be set with fixed timelines to complete the project in time such as;

- Facial authenticator setup

- Login/registration setup with hashing the passwords being saved in the back-end DB
- Wireframes leading to general UI development and configuration
- Testing of full login setup up to reaching the UI after a successful login
- Proper UI setup and development based on the wireframes
- UI section development as per the features to be created
- Password vault development
- Document vault development
- Securing the password vault
- Securing the document vault
- Testing both vaults for working functionality with the DB
- Individual feature development and integration into the UI
- Testing each feature after being developed
- Final testing of the full application
- Deployment

Technical Details

Following are some of the Technologies and libraries that I plan to implement during development to achieve the desired purposes through the functional requirements, further details and changes will be made in this technological specification up to the midpoint documentation:

1. **Development:** First starting with the development process I plan on using the **SERN** stack for high scalability etc. This stack includes MySQL for databases, Expressjs as server framework for Nodejs (Runtime Environment) implementation for the back-end development and server setup and Reactjs library to develop the front end of the application using bootstrap themes to beautify the front-end and make it more accessible and easier to use for the user.
2. **Login & registration:** For Login and registration purposes I plan on building a facial recognition authenticator coupled with multifactor authentication. For the facial authentication I plan on utilising a facial recognition model that will implement deep learning. I will be using a flask server to contain the model and for communication between the react login and the deep learning model. For building said flask server I will be using important imports such as; tensorflowjs, NumPy(library for python programming – since flask is a web-framework for python), JSON, also p5.js for capturing face images which I will later use with the model for verification etc. For coupling this with multifactor authentication I plan on utilising Auth0 guardian to implement multifactor authentication which might include OTP's, QR scans, push notifications and secure the login completely. For securing the registration forms I plan on using formik and Yup, yup is a js schema builder for value parsing and validation contained inside formik (js form library for React).
3. **Password Encryption:** For password management and encryption inside the application I plan on using Cryptojs which is an external js library used to implement high level cryptographic encryptions up to AES 256 bit encryption (Highest encryption bit level provided by cryptojs) with initialisation vectors to separate (avoid same keys being used) and encrypt & decrypt multiple password stored in the application (also considering bcrypt for blowfish type encryption which has limited range (17 bytes)). This is optional and if time I would look into parameterized encryption algorithms which provide the user with an option to choose the type of encryption.

4. **Document security:** For document security I would be developing a file vault to simply store documents under locking authentication and according to time and development requirements I am also considering whether to store files through compression and encryption, for which I plan on utilising Node streams(read and write streams) to take in documents and compress them using external libraries like zlib as a part of gzip software which can be used with the streams (file compression and decompression) and encrypting them after using Cryptojs Or simply utilising JSONFile encryption packages available in Node for file encryptions etc. For file uploading I will be using ReactDropzone.
5. **Password analyzation:** For password analyzation i.e., strength testing and distinctor for checking uniqueness and password vulnerability I plan on using the Troy Hunt's pwned Passwords API to cross-reference whether an existing saved password has been compromised before or not and notify the user regarding its results. Then using RegEx testing to test password strengths and determine their complexity etc. For the automatic password generator I will be developing a password generator capable for options to develop passwords tailored to user requirements. I plan on using react toastify for all notifications regarding password analysing features.

Project Plan

Project plan including the Task list with the timeline mentioned in the Gantt chart provided:

1. Research wireframes and design ideas on how to structure the application UI to make it easy and user friendly
2. Setup the react app in visual studio code
3. Build react login & registration page (front-end) demo using bootstrap
4. Build flask server using imports and try to implement the flask server and store facial recognition model
5. Setting up the SQL database and integrating it with the front-end for login & registration
6. Integrating and developing the facial recognition authentication for login and registration
7. Adding 2FA on the login system to couple with facial authorisation and verifying its function
8. Unit testing the login & registration system to verify its purpose and functioning
9. Building a basic UI for the main application to include different sections of the application according to functional requirements.
10. Setting up node and express for the back-end
11. Integrating the SQL database for node and express as well
12. Building the password vault/manager UI as a section of the application UI
13. Coding password vault logic and integrating it with the SQL database
14. Coding in the encryption and decryption logic
15. Completing the credential vault logic
16. Improving on credential vault UI and finalising its function
17. Unit testing the credential vault logic to verifying its functionality and storage capabilities
18. Developing the UI for password generator in the designated UI section
19. Writing the back-end login for the password generator (user tailored)
20. Integrating the back-end logic of the password generator with the designated UI section
21. Unit testing the password generator with its UI to verify functionality by trying to generate multiple strong passwords and later on checking with the distinctor and strength analyser
22. Building the password strength analyser user friendly UI (strength meter etc.) section in the designated section of the main UI

23. Developing the logic code behind testing the strengths of passwords with building strength meters in the front – end
24. Checking for and building the notification ability of the strength analyser
25. Integrating the full logic with the front-end user friendly UI
26. Unit testing the password strength analyser with multiple password types and passwords generated from the user tailored password generator
27. Building the password distinctior UI section in the designated section of the main UI
28. Coding the back-end login utilising the Troy hunt API in the back-end to check password compromised or not. With notification alert
29. Integrating the back-end login with the front-end UI section for the distinctior
30. Unit testing the distinctior for functionality verification using compromised and password generator generated passwords.
31. Building a document vault UI in the designated section
32. Coding the back-end logic for a file/document uploader
33. Integrating the back-end logic with the SQL database to store files
34. Implementing way to secure the vault logic and hide/encrypt all files uploaded or implementing a locking mechanism for the vault only to be opened with proper authentication for security of documents
35. Integrating the back-end logic for file vault with the UI
36. Unit testing the file uploader and the document vault for security and functionality
37. Unit testing all developed functionality once again to make sure each function does as intended and gives the desired outputs or stores with security
38. Integration testing the application to check once the application is fully integrated that all functions work together
39. Verifying all non-functional and functional requirements accounted for in the project
40. Uploading project files to github for backup and reference
41. Finally researching the deployment platforms in depth to determine the best and most suited deployment platform (under consideration for now – heroku, digital ocean and netlify)
42. Successfully integrating the whole project and deploying it to the most suited platform according to the steps of deployment
43. Testing deployed application for all functionality working according to intended purposes and checking the deployment does not hinder any non-functional requirements of the application
44. Creating full technical documentation report
45. Testing and Creating demo videos for the application
46. Project submission. May 8th

1. Login testing: Once the whole login system is setup it will be tested with multiple bogus user accounts to verify its functioning, that includes testing the face recognition that will be implemented in the login and register, the face recognition will be tested with actual users and images to test out all possible vulnerabilities it might contain and to verify its function according to its purpose and same will be done for the registration process. After the face authentication is verified, the system will also be tested for multi factor authentication to verify that the app does not allow a successful login until all external factors required for login are properly authenticated which might include OTP's, push notifications or QR code verification.
2. Password encryption testing: After the password vault is manually coded, it will be tested properly to check all saved passwords are highly encrypted and cannot be easily viewed, accessed or exploited with simple brute force cracking algorithms or authentication exploitation etc. The server/database side will also be check and verified to see none of the passwords being saved can be viewed easily i.e., they are all encrypted in the back-end as well.
3. Password generator testing: The password generator will be used to generate highly complex passwords that satisfy most conditions required for a strong password to be generated, these passwords will be tested across multiple strength requirements of other websites and applications to determine all passwords generated are strong, complex and secure to use i.e., they haven't been compromised. They will be checked on the passwords analyser being built with the application to check their uniqueness against the troy hunt database and determine their strength using the strength analyser.
4. Password analyser testing: Once the password distinctior and strength analysers are coded, they will be tested using the password generator generated random passwords which are supposed to be unique and complex, simple passwords will also be used for testing to verify the system does catch passwords that are too weak or have already been compromised in the have I been pawned database.
5. Document vault testing: After successfully building the document vault it will be tested to verify its security and accessibility to only properly authenticated users. All possible scenarios of accessing the documents will be implemented to ensure the document cannot be viewed or accessed until the user is properly authenticated and possesses the right credentials for the vault as well.
6. Integration testing: After all unit testing methods have been accounted for and it is confirmed that all functions are secure and properly perform their duties the application as a whole will be tested together in sequence of all functions to verify that once integrated the application performs all its functions and that too while maintaining its security. The same integrations tests will also be performed once the application is deployed and it is confirmed that even after deployment the application does not get buggy or none of its functions are affected by its deployment.

Finally after all testing, the application will be handed to friends and family for usage and review, to discover any vulnerability or possible threat that can be used for exploitation and will be reviewed and fixed.

Journal Entries

December

Supervision & Reflection Template

Student Name	Shaurya
Student Number	X18138284
Course	BSHC-4 (Cyber-security)

Month:

What?

Reflect on what has happened in your project this month?

For this month of the project I started on with changing the styles and layout of the existing application and moved on to conducting research on how the Have I been Pwned API can be utilised, what its exact function is and evaluating its actual worth for the application. Then I moved onto its implementation testing and integration with the current existing application. After Which I was responsible for making the midpoint presentation and preparing the mid-point documentation for the project.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

This was a major feat for the project progress since I had a functioning Facial login which could be utilised to access an important feature of the application and it all worked in conjunction i.e., each part of the application was integrated with each other to work as one application. Implementing the Have I been Pwned API was also a challenge since I had to figure out a method of implementation that could easily be integrated with the rest of the application. Once a method of implementation was found it was necessary to be tested and verify all API calls working. It required a basic front-end and finally integration with the current application so it could only be accessed after successfully registering and logging into the application. As a result I got a working prototype of the application which had a successful facial scanner registration and login, and one core feature of the application which could only be accessed after the login was successful after

which I prepared the Mid-point documentation and presentation with the existing prototype. Challenges still remain as to how I plan to integrate this prototype with the main feature suite of the application.

Now What?

What can you do to address outstanding challenges?

There are major outstanding challenges that still remain that include a secure login registration separate from this which also require the building of the main feature suite and further integration of this prototype with the main feature suite. The multifactor research and implementation and the integration of the planned parts ahead with this working prototype is going to be a challenge that I will be addressing ahead. For this I first plan on developing the feature suite and focusing on its integration after which I plan on securing the login and making use of the prototype already built.

Apprentice Signature

Shaurya

November

Supervision & Reflection Template

Student Name	Shaurya Kumar
Student Number	X18138284
Course	Bsc(Hons) in Computing

Month:

What?

Reflect on what has happened in your project this month?

This month I was working on implementing the back-end for Facial recognition login and registration feature for my application.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Initially I had to research all technologies required to implement this feature and had to re-examine my approach towards this feature i.e., change it. Initially I planned on utilising a Face-net

machine learning model in the back-end with python and tensorflow with keras extensions. But through the research and implementation progress I decided to use another approach to implement this feature since there weren't clear defined constraints regarding databases for the previous approach, I decided to use the face_recognition extension of the OpenCV extension for python in my back-end which is also stored in a flask server in the back-end and through that, click a picture of the user's face, convert/Encode it into 128 bit measurement vector and then store this measurement vector in a SQL database with the name entered by the user for successful registration and then implement the same extension and implementation to compare user face measurements and name from the database to confirm the user's trying to login. Through rigorous implementation and debugging practises I found it wasn't entirely easy to implement this feature since it required extensions and import which had to be individually compiled using visual studio separately to be installed on the system these included the dlib library and the face_recognition library import, all other imports required direct installations in the system to be used like numpy, cv2 from opencv etc. But I have got the back-end system server running and plan to build a React JS front-end to connect this back-end too and test it out on a browser as an integrated feature. I also plan on configuring the back server to show the camera frame while taking a picture of the user for making it visually transparent for the user while they try to register since the camera frame for now doesn't show while taking the user's picture for registration etc.

Now What?

What can you do to address outstanding challenges?

The outstanding challenges regarding this feature are to implement a front-end using React (creating react web app front-end) to be integrated with this back-end and connected to the flask server so the feature can be tested on a web browser with complete functioning by setting up all the connecting and navigating routes accordingly and can be further debugged accordingly. The back-end further requires configuration to display the camera frame while taking a picture of the user for registration and then shown during login to verify the user's registration.

Student Signature

Shaurya

January

Supervision & Reflection Template

Student Name	Shaurya
Student Number	X18138284
Course	BSHC-4 (Cyber-security)

Month:

What?

Reflect on what has happened in your project this month?

For this month of the project I have focused on working with the password managerial part of the application which includes research on the rest of the login registration required, methods of implementation and back-end options including node, express or server less backends and database options. Further implementation techniques regarding the password manager feature

itself using crypto js for encryption and decryption.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

The project progress included successful research material and actual implementation experimentation of various techniques to determine and finalise which technique and implementation is to be utilised. Currently experimenting with a server-less back-end to serve all api requests automatically and using third party application to pass on the login registration to and implement more secure ways ahead. Then try and utilise these login sessions to utilise and connect to the application dashboard. Progress also included basic UI development including its structure etc., and structuring routes and pages according to other features that are to be developed. The final progress update for the month included experimental implementations of third party login processes.

Now What?

What can you do to address outstanding challenges?

Since the current progress is still underway, the outstanding challenges include finalising the managing feature according to user sessions and database implementation and connection regarding the feature itself. Then researching ways of securing the login registration process for secure authentication. And finalising the managing feature with implementing security for the passwords.

Apprentice Signature

Shaurya

February

Supervision & Reflection Template

Student Name	Shaurya Kumar
Student Number	X18138284
Course	Bsc(Hons) in Computing

Month:

What?

Reflect on what has happened in your project this month?

February was one of the most Important months for the development of the project. It involved th development of the password manager itself, each functionality of the password manager including Add password, Edit Password, Delete Password, Retrieve Password By User, retrieve All passwords. This also included the implementation of the Fauna Database API which was chosen over GraphQL and SQL, due to its amazing feature and security suite. Fauna DB was also chosen due its compatibility with Auth0 and Next.js at the same time was very efficient. Fauna Database with the password manager was finished this month. The password manager also included the setup of the Fauna DB models file which was the database connecting file, it also included the development of the password encryptor file which was used for password encryption utilising the crypto module and AES 256 bit encryption.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Since the Password manager is one of the main features of the application, this month's development was a milestone completed. A lot of details were required to be developed along with the password manager as well. These included the dynamic Icon updating and integration with Auth0 as well. This has setup a foundation for all other features to be developed around it. Some major challenges still remain such as integrating the facial login authenticator with this application. Testing, implementation of a multifactor login etc.

Now What?

What can you do to address outstanding challenges?

The plan for the next month is to improve this password manager first, since it still requires input sanitation, proper authentication protection, testing of the functionality. User session development with auth0 and its integration with the password manager, routing for all other features from the feature suite of the application etc. To address some of these challenges; research regarding potential integration methods, different ways and means of multifactor implementation, libraries that could be used for data validation, secure routing through authentication and session management etc. are under way.

Student Signature

Shaurya

March

Supervision & Reflection Template

Student Name

Shaurya Kumar

Student Number	X18138284
Course	Bsc(Hons) in Computing

Month:

What?

Reflect on what has happened in your project this month?

This month, the project involved implementation and testing of the main password manager UI interface integrated with the Fauna DB and the third party secure login registration. The basic layout of the application foundation was dynamically built depending on the states, built with a proper securely routed Login registration process tracking user session with cookies with third party application security and leading to the actual password manager UI. The password manager UI itself is dynamic and tracks the user, it also tracks whether the user verified the email their using to login. The password extracts the user details from the Fauna DB and keeps a track of all user actions. Password can securely be saved using this UI. Each password detects the website for which the account password is being saved and dynamically updates the index icons which make it easier to recognize which password is saved. The routes for all other features have been set but are under development. Each modal form component used in the manager UI is properly Input sanitised and validated to maximise security and avoid forgery and unbound errors. The manager provides full functionality at the moment as well including CRUD and peek preview.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Major progress regarding the project was done since the foundation of the UI with the main password manager was setup, security surrounding the manager as well as the UI access is also underway. And routes for consequent functionality development are ready. Successes were the development of the main manager itself, Input sanitation and validation setup, proper secure login registration routes etc. Challenges faced during development included integrating Fauna with third party security coverage and access, linking fauna, the application and third party security, dynamic routing, Securing routes, dynamic user tracking and proper input validation and sanitation. Further challenges include multi factor implementation integration, further functionality development, TDD style of development practise, further feature integration, UI

cleaning etc.

Now What?

What can you do to address outstanding challenges?

Addressing the outstanding challenges requires further research and testing the on-going development with bogus information, user review and improvement, other functionality integration and unit testing, for addressing multifactor setup; other factors and their feasibility will be experimented with etc.

Student Signature

Shaurya

April

Supervision & Reflection Template

Student Name	Shaurya Kumar
Student Number	X18138284
Course	Bsc(Hons) in Computing

Month:

What?

Reflect on what has happened in your project this month?

The Development in the month of April has pulled through with the integration for the whole application. Such as completed user authentication, multifactor implementation, Server side data validation in the flask back-end as well in addition to client side input data validation, API and route security. Some technologies were also experimented with. The back-end database implementation for facial authentication was changed to SQLite3 from simple SQL due to efficient flexibility. An Idle timer feature was developed in the application as well. All other features were developed in the consequent routes. Security principles and pernicious kingdoms were incorporated into the application to improve on security. The application is in its final stages due to the integration of facial authentication and validation being successful. The development during this month has been drastic as to the previous month's progress due to the integration of all parts of the application coming together. Also due to the improvements in security because of the incorporation of security principles and measures against the pernicious kingdom vulnerabilities in the application.

So What?

Consider what that meant for your project progress. What were your successes? What challenges

still remain?

All the progress during this month has contributed towards bringing the application into its final stages. There are still tweaks required with all other features in the feature suite, there is also plan to alter the password manager for security improvement benefit in motion. The facial authentication validation has been integrated into the application but still needs to be adjusted and fixed. Quite a few challenges still remain. The application has also been experimented with newer technologies like FaceAPI.js to render live facial imprints on Webcam footage.

Now What?

What can you do to address outstanding challenges?

Outstanding challenges that remain now include proper implementation of error and exception handling in the client server and back-end. Further thorough testing of the application remains. Proper implementation of the facial ID authentication remains, It has been integrated but needs to be implemented correctly. The testing framework that has been decided is Cypress which is a fully automated and wide flexibility inclusive testing framework for java script applications which involves a feature suite to conduct end to end, integration and unit testing in the application. Research into a better security model which does not require code alteration is also underway. The application should be production ready before the deadline with a strongly integrated security framework.

Student Signature

Shaurya