National College of Ireland

Software Project

Cyber Security (BSHCYB4)

2020/2021

Lukas Zubrus

x18107079

x18107079@student.ncirl.ie

SecureX

Technical Report

Contents

Table of Figures

Executive Summary

SecureX is a web application developed using Laravel's Php framework. SecureX helps you create manage and store online login details safely inside vaults that you can create. Sites that are stored in vaults have strong state-of-the-art AES-256-CBC encryption set in place for your login and password.

Compared to other similar applications we offer a very easy and straightforward setup with complimentary user-interface that is easy to navigate for beginners. Sorting your details has never been more user friendly even if you had hundreds of login details saved, they can be organised into folders and found easily using the search bar.

Users can generate strong unique passwords that do not need to be remembered because it is stored on your account that requires one strong master password that's secured with a bcrypt hashing engine. SecureX account can also be set up to have Googles two-factor authentication which would put a peace of mind on users.

System admins are also not forgotten and have been granted their own admin dashboard with its exclusive functionality like creating announcements, managing users, changing internal system settings and more. SecureX goal is to appeal to existing password manager users but to also convince people that have never used such an application to how easy and intuitive your experience can be without being overwhelmed by the user interface of other similar applications.

1.0 Introduction

1.1. Background

I undertook this project because I wanted to learn more on how we can secure our information securely on the web and so I decided to develop my own interactive web page that would let its users store their sensitive information securely.

We as people are not good at remembering many passwords, but machines are and so with this web application you will only need to remember one master password that will grant you access to all your existing passwords.

Another reason why I choose to pursue this project is from the research that I have conducted online I see a strong possibility that password managers will become more widely used in the future especially for organizations. These tools can help employees generate strong passwords without having to remember each one which in turn would add that extra layer of security against possible attacks.

1.2. Aims

SecureX aims to provide its users with an intuitive user-interface for storing and managing all your personal information safely all in one place. Users will have their own vaults where they can store logins and generate strong passwords to improve their existing accounts security. Having unique passwords does not mean you need to remember them; all you will need is your “Master Password”. All your vaults can be locked with an additional password and sorted into folders like Email, Bank, Social etc. to address scaling. Users will also be able to easily find logins by using the search bar. All your sensitive information will be hashed using bcrypt to ensure maximum security.

We also aim to provide SecureX administrators exclusive control to manage users and various system settings.

1.3. Technology

SecureX uses **Laravel's Php web application framework**. I choose this framework because Laravel comes with secure **bcrypt hashing** and **AES-256-CBC encryption** will be used throughout the project when storing and retrieving data. I will also use **Livewire** which is a full stack Php framework for Laravel. Livewire will be used to develop dynamic interface pages without having to use JavaScript frameworks like Vue.JS. Laravel also provides a powerful **blade templating engine** that will be used for the frontend development of the application. **Bootstrap 4** will also be used on top of blade for navbar, sidebar, icons, and an overall user-friendly view.

We will also use various open-source packages like **laravel/sanctum** that allows us to generate multiple API tokens for users. Tokens can be granted abilities like authenticating when user registers, login accessing vaults and various other functions that require user to be authenticated.

techtailor/rpg is used for generating passwords and adjusting generation settings like strength and length. It's also used to generate users pin and access key when the user has registered successfully.

pragmarx/google2fa is used to generate secret keys that will be used to provide users with unique QR codes that can be scanned with a smartphone to enable two-factor authentication when logging into SecureX.

coderello/laraflash is used for providing users with notifications or a form of output to the user to confirm that the action that they took was completed successfully like deleting a vault.

pestphp/pest is a Php testing framework and was used to carry out various test cases of the application.

All the packages are installed using **Composer** which is a dependency Manager for PHP.

SecureX is developed locally using **Xampp** as it comes with Php pre-installed it's the easiest way to manage the applications database and to make backups during development.

1.4. Structure

The purpose of this document is to provide the reader with a clear description and purpose of the application that has been developed. The document describes functional requirements and what processes take place in the backend and frontend of each feature of the application. We will then touch on data, user, environmental and usability requirements of the application. I am also going to describe some of the key implementation aspects of the application and how user data is secured. Finally, we will conduct testing and discuss future developments of the application.

2.0 System

2.1. Requirements

2.1.1. Functional Requirements

User Functional Requirements

Functional Requirement No.	Functional Requirement Description
UFR 1	User can register
UFR 2	Authenticate email and password has not been reused
UFR 3	User can login
UFR 4	User added to database and Initial Vault assigned
UFR 5	User Dashboard
UFR 6	User can edit profile details
UFR 7	User can view IP logs
UFR 8	User can delete his account
UFR 9	User can create new vaults and manage vault settings
UFR 10	User can add login details to vaults
UFR 11	User can enable Two-Factor Authentication (2FA)
UFR 12	User can generate random passwords
UFR 13	User can Logout

Admin Functional Requirements

Functional Requirement No.	Functional Requirement Description
AFR 1	Admin can login
AFR 2	Admin Dashboard
AFR 3	Admin can add and manage announcements
AFR 4	Admin can manage pages
AFR 5	Admin can manage registered users
AFR 6	Admin can access system settings
AFR 7	Admin can manage user vault count
AFR 8	Admin can start maintenance mode
AFR 9	Admin can control access setting
AFR 10	Admin can enable/disable Two-Factor Authentication (2FA)
AFR 11	Admin can logout

2.1.1.1. Use Case Diagram

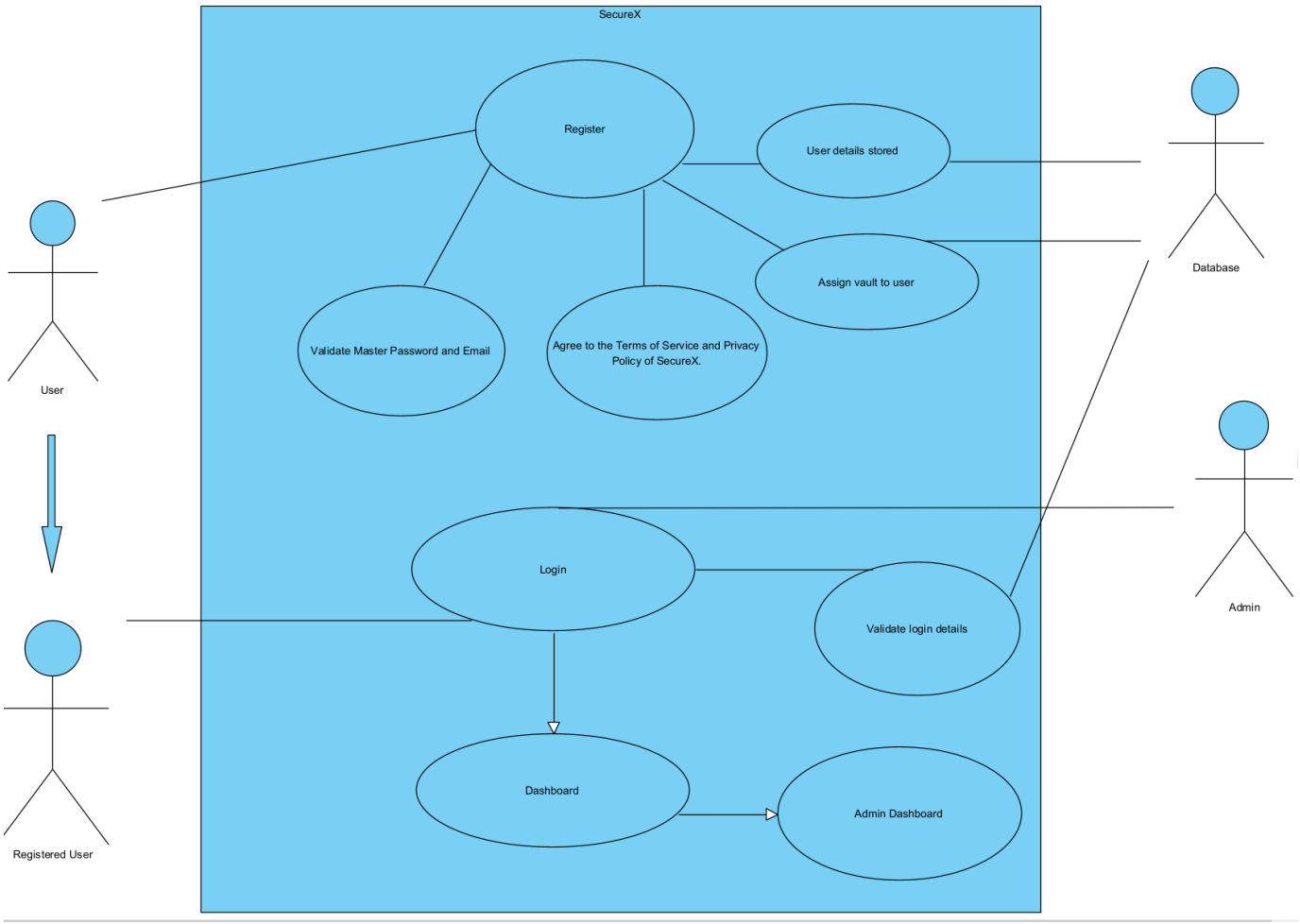

Figure 1

2.1.1.2. Requirement 1: User Registration

2.1.1.3. Description & Priority

User registration is essential in accessing SecureX features without registration user cannot login successfully thus no access to the main dashboard.

Register page has the following fields:

- First Name*
- Last Name*
- Email Address*
- Master Password*
- Confirm Master Password*
- Password Hint (Optional)

Fields marked with "*" must be filled in.

2.1.1.4. Use Case

Scope

The scope of register page is for user to input all required fields so that the application can verify details and validate if master password matches. It also validates if the user has been registered before with the same email if true you will have to enter different email address.

Description

This use case describes the register page and how it operates once the user fills the required fields and clicks “Register”.

Use Case Diagram

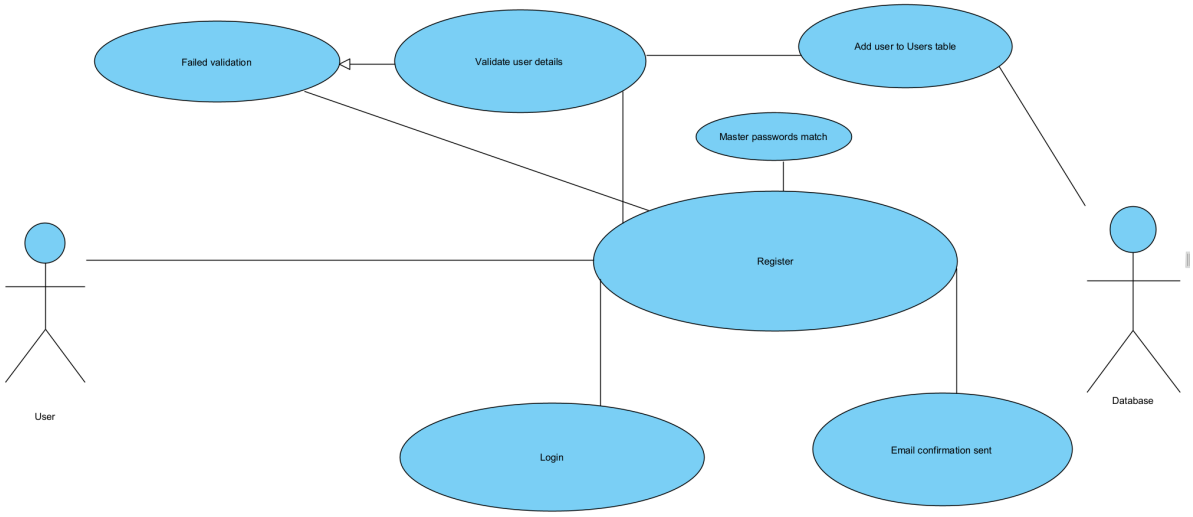

Figure 2

Flow Description

User attempts to register.

Precondition

When the user visits SecureX for the first time he will be in the login page. If user does not have an account, they will need to click “Create One Now” which would bring them to the register page.

Activation

This use case begins when the user is in the register page.

Main flow

1. **User** fills in the required fields first name, last name, email, master password and confirm master password.
2. **System** validates the details successfully.
3. **System** add user to Users table, generates support pin, access key and creates default vault.
4. **System** sends out a confirmation email to the user.
5. **User** confirms registration by clicking the email link.
6. **System** verifies the user.
7. **User** can now login.

Alternate flow

A1: Admin Adds User

1. **Admin** adds user through "All Users" tab
2. **User** logs in with the registered details assigned by the admin.

Exceptional flow

E1: Registration Failed

1. If **user** master password and confirm master password do not match the system display message "Master passwords do not match"
    - **System** validation failed.
    - Use case resumes at main flow 1.

2. If **user** enters a password that is to short "<8" system display "The password must be at least 8 characters."
    - **System** validation failed.
    - Use case resumes at main flow 1.

3. If **user** uses same Email that's already in the users table System display "The email has already been taken."
    - **System** validation failed.
    - Use case resumes at main flow 1.

Termination

1. User registered successfully.
2. User details are invalid registration failed.
3. User exits registration page.

Post condition

New user is added to the database.

2.1.1.5. Requirement 2: User Login

2.1.1.6. Description & Priority

User login is our second functional requirement. Once the user has registered successfully, they will have to login to access all the sites features that are available from the user dashboard.

Login page has the following fields:

- Email Address*
- Master Password*
- Remember Me

Fields marked with **"*"** must be filled in.

2.1.1.7. Use Case

Scope

The scope of login page is for user to input all required fields so that the application can validate user details with the ones in the users table of the database and give user access to the main dashboard.

Description

This use case describes the login page and how it operates once the user fills the required fields and clicks "Login".

Use Case Diagram

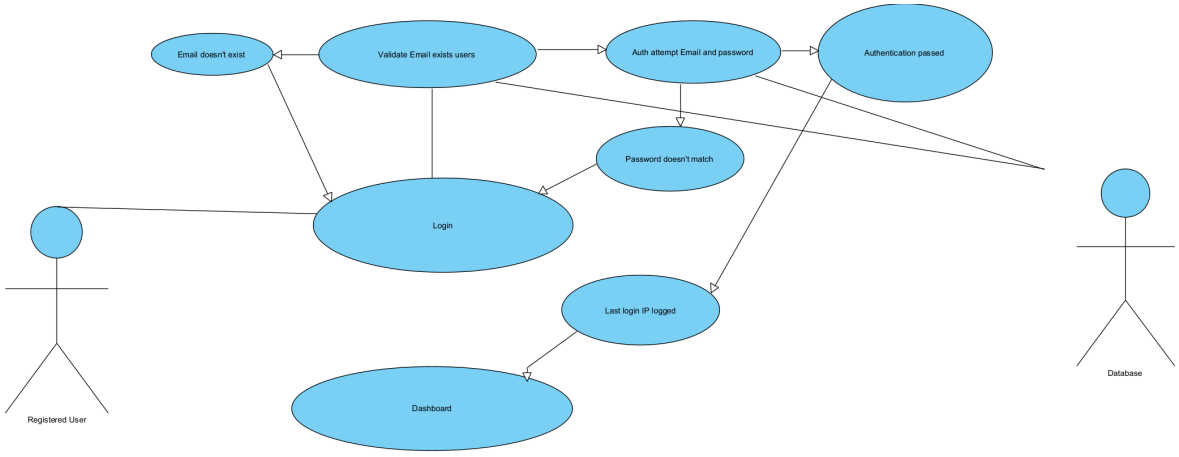

Figure 3

Flow Description

User attempts to login.

Precondition

When the user visits SecureX for the first time he will be in the login page. Here they will need to input their login details email and master password that they created during registration.

Activation

This use case begins when the user is in the login page.

Main flow

1. **User** fills in the required fields email and master password and clicks login.
2. **System** validates email exists in database.
3. **System** authenticates user email and password passed.
4. **System** logs users IP.
5. **System** redirects logged in user to the dashboard.
6. **User** is now in main dashboard page.

Alternate flow

A2: User forgot password.

1. **User** clicks “Forgot Password?”
2. **System** displays password reset page.
3. **User** enters their email address.
4. **System** sends password reset link.
5. **User** sets new password.
6. **User** signed in successfully.

Exceptional flow

E2: Login Failed

1. If **user** master password email does not match system display message “The Master Password does not match our records. Check and try again.”
    - **System** validation failed.
    - Use case resumes at main flow 1.

2. If **user** enters wrong email system display “The selected email is invalid.”
    - **System** validation failed.
    - Use case resumes at main flow 1.

Termination

1. User logged in successfully.
2. User details are invalid login failed.
3. User email does not exist, login failed.

Post condition

User IP logged.

User redirected to the dashboard.

2.1.1.8. Requirement 3: Vaults

2.1.1.9. Description & Priority

User vaults is our third functional requirement. Once the user has logged in successfully, they will be brought to the user dashboard. From the dashboard user can access their vaults from All Vaults option in the sidebar. User can create a new vault and assign it:

- Vault Name*
- Vault Description*
- Select an Icon*
- Preferred Colour*
- Vault Password (optional)

2.1.1.10. Use Case

Scope

The scope of All vaults page is one of the key aspects of SecureX it should let users create a vault add sites, add extra fields, create folders, and secret notes.

Description

This use case describes the “All Vaults” page and what functions it can achieve.

Use Case Diagram

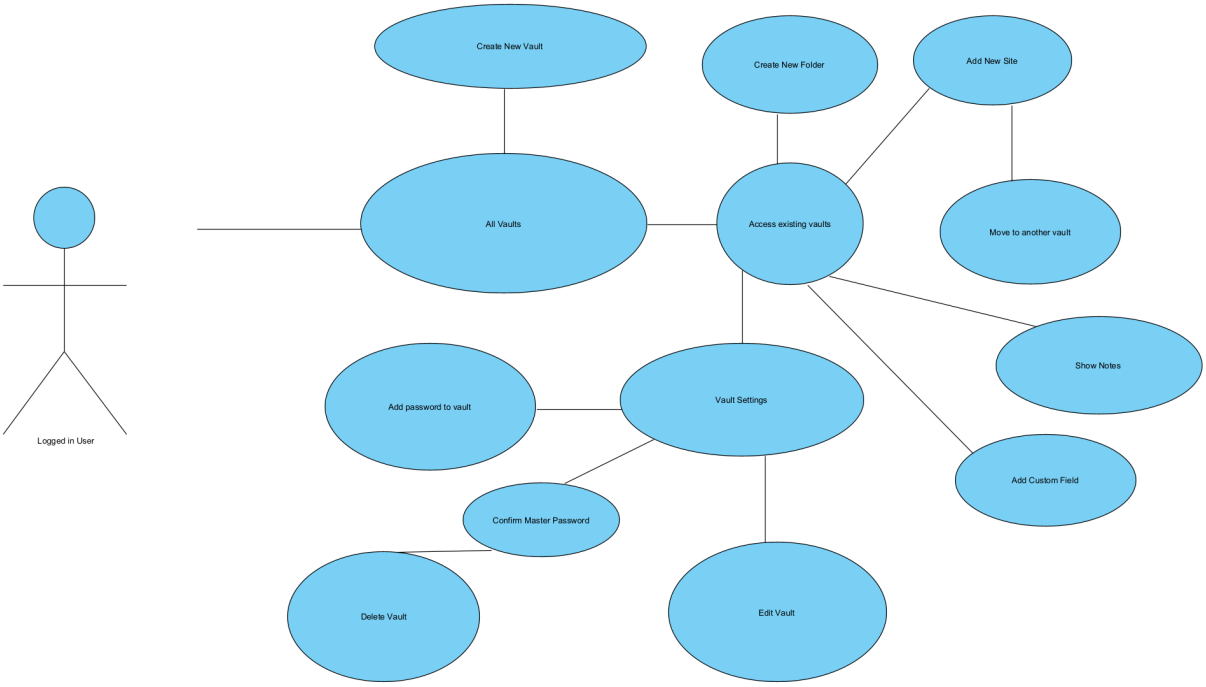

Figure 4

Flow Description

User attempts to create new vault.

Precondition

User needs to be logged in to the system.

Activation

This use case begins when the user wants to add a new vault.

Main flow

1. **User** fills in the required fields Vault Name, Vault Description, Select an Icon, preferred colour and leaves optional password blank.
2. **System** checks if the required fields are filled in.
3. **System** Adds vault and updates the total list of vaults on the server.
4. **User** can now access new vault.

Alternate flow

A3: User creates vault with password.

1. **User** fills in the required fields Vault Name, Vault Description, Select an Icon, preferred colour, and vault password.
2. **System** checks if the required fields are filled in.
3. **System** Adds vault, encrypts the password, and updates the total list of vaults on the server.

4. **User** must unlock the vault with set password to access the vault.
5. **System** validates the password and unlocks the vault for duration of the session.
6. **User** vault access granted; user can now view contents of the vault.

Exceptional flow

E3: Add new vault failed.

1. **User** Vault name is too long, display message "Vault Name can contain maximum 14 characters".
    - **System** validation failed.
    - Use case resumes at main flow 1.
2. **User** maximum number of vaults reached.
    - **System** validation failed.
    - Use case resumes at main flow 1.

Termination

Maximum number of vaults reached.

User exits create new vault.

Post condition

Vault added to vaults table.

Total vault count recorded.

2.1.1.11. Requirement 4: Two-Factor Authentication

2.1.1.12. Description & Priority

Our fourth functional requirement is two-factor authentication. This requirement is important as it allows users to add an extra layer of security and creates a two-step login where you need to verify the login code through your authenticator app on your smartphone.

Following fields are used:

question_1* – Security questions

question_2*

id – Used for identifying user and attaching 2FA.

Answer 1* – Used for removing authenticator without a phone.

Answer 2*

2.1.1.13. Use Case

Scope

The scope of two-factor authentication is that the user can increase their account security by adding two-step process when they are logging in.

Description

This use case describes the process of successfully activating two-factor authentication on your account.

Use Case Diagram

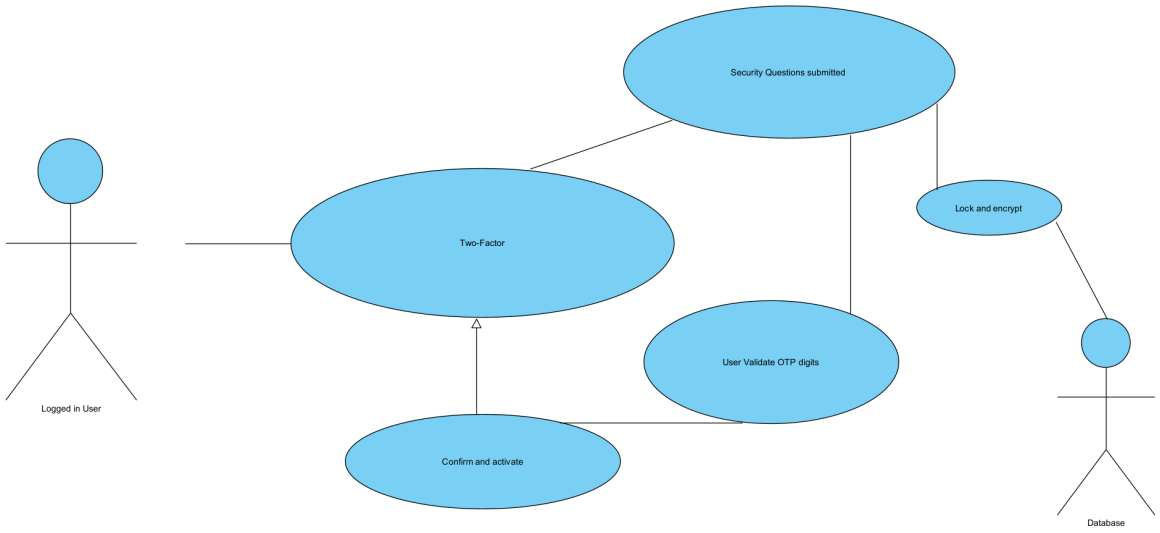

Figure 5

Flow Description

User attempts to activate two-factor authentication.

Precondition

User needs to be logged in to the system and go inside two-factor tab from the dashboard.

Activation

This use case begins when the user wants to add two-factor authentication to their account.

Main flow

1. **User** selects two security questions from the dropdown menu, inputs 2 answers and presses “Submit Security Questions”.

2. System checks if both fields are filled in.
3. System adds security questions to database.
4. System encrypts and lock's security questions.
5. System outputs user with QR code and input field for OTP code.
6. User opens their authenticator app on a smartphone and scans QR code to generate OTP code.
7. User inputs OTP code into empty field and presses "Confirm & Activate" button
8. System verifies OTP code with two factor secret questions.
9. System 2FA enabled display message "success".
10. User now has two-factor enabled.

Alternate flow

A4: Reset 2-Step Key & QR Code

1. User selects two security questions from the dropdown menu, inputs 2 answers and presses "Submit Security Questions".
2. System checks if both fields are filled in.
3. System adds security questions to database.
4. System encrypts and lock's security questions.
5. System outputs user with QR code and input field for OTP code.
6. User opens their authenticator app on a smartphone and scans QR code but its invalid.
7. User Resets QR code by clicking "Reset 2-Step Key & QR Code"
8. System resets secret code and generates new QR code.
9. User inputs OTP code into empty field and presses "Confirm & Activate" button
10. System verifies OTP code with two factor secret questions.
11. System 2FA enabled display message "success".
12. User now has two-factor enabled.

Exceptional flow

E4: Security questions incomplete

1. User only fills in one security question display message "Security questions required".
    - System validation failed.
    - Use case resumes at main flow 1.

Termination

Wrong OTP code

User disables 2FA.

Post condition

Security questions added to security_questions table.

Two-factor authentication enabled.

User notified through email.

2.1.2. Data Requirements

Data requirements are as follows for user registration:

- **First Name**
- **Last Name**
- **Email Address**
- **Master Password**
- **IP address**

Inside “My profile” further data requirements are required for updating profile:

- **Country**
- **Phone number**
- **Date of Birth**
- **Address Line 1**
- **Address Line 2**
- **City**
- **Zip Code**
- **State**

All these extra data requirements that you can see above will be used in future work when adding a payment system to unlock more vaults, provide membership title and for any other premium features that fell out of scope during this development cycle.

2.1.3. User Requirements

- Easy to use UI interface.
- Ease of use when it comes to managing and sorting passwords.
- Security such as password encryption and any other personal data.
- Generate unique passwords.
- Edit existing profile details.
- Able to delete their account easily.
- Strong overall website security resilient to cyber-attacks.
- Two-factor authentication (2FA) through the smartphone.
- 24/7 support and access to the SecureX dashboard.
- Sufficient number of vaults to store website login details.

2.1.4. Environmental Requirements

- Regulatory requirements - such as users have the right to know what data we hold about them. Users can also request their personal data by contacting support and users are able to delete their account if they wish.
- Desktop or a smartphone with access to a stable internet connection is required to run SecureX web application.

2.1.5. Usability Requirements

- Easy to use minimalistic interface.
- Easy first-time setup.
- Fast access to data stored.
- Error prevention.
- Help documentation.
- Scalability.

2.2. Design & Architecture

Database Design Structure

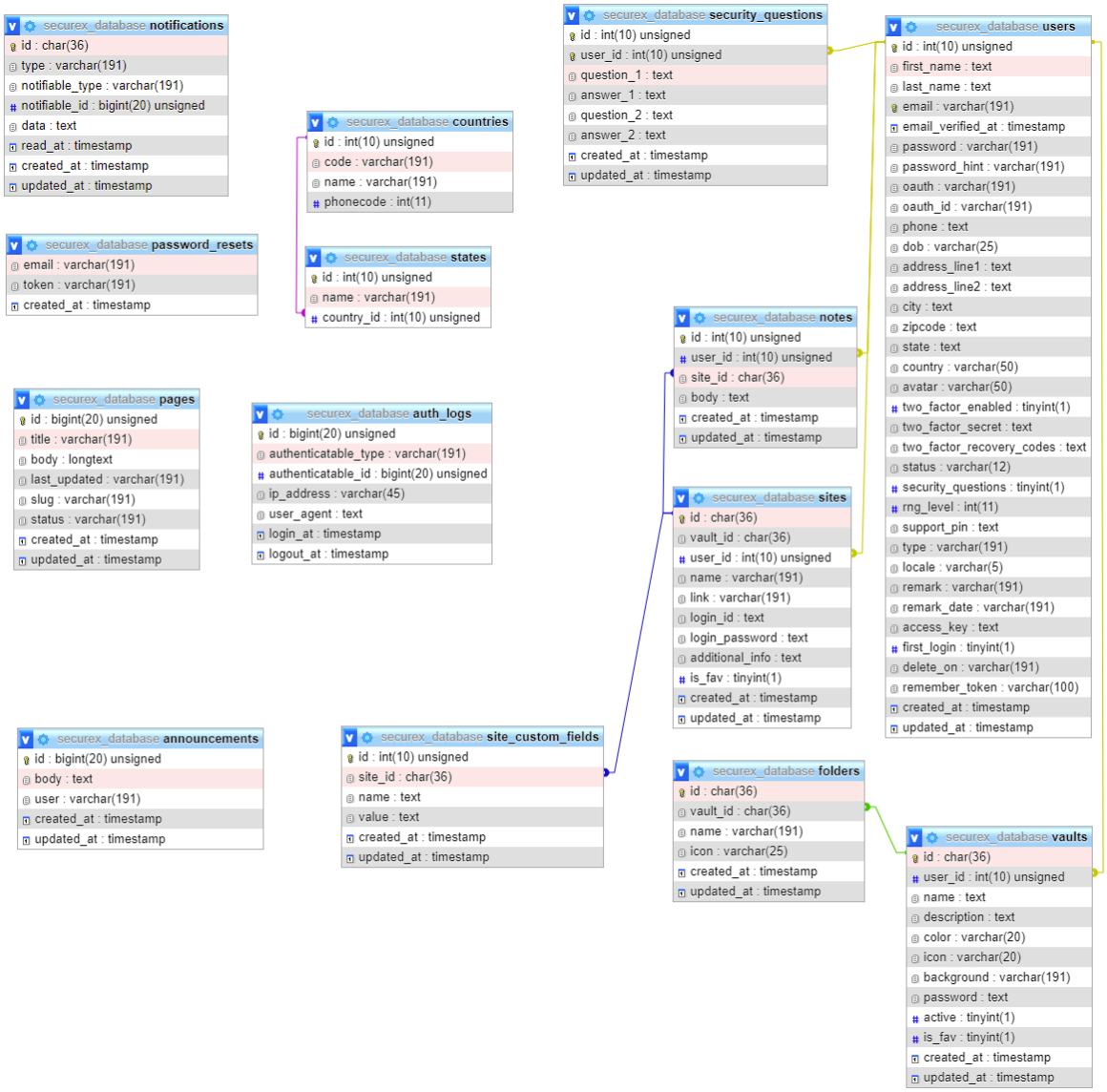

Figure 6

Above is the design structure of the database and its relationships.

When the user registers, they are added to the users table and their password is hashed. Support_pin and access_key is generated for that user. User id is assigned to a default vault. Security questions, vaults, sites, folders, and notes are all assigned to user id.

Sites table is linked with notes and site_custom_fields tables. Vaults are also linked with folders with vault_id but site data is not stored in folders in the database. As you may have noticed most of the tables consist of timestamp fields which record when a certain action was started and ended.

The system follows similar procedures when user inputs information into fields such as:

User input→Validation→Bcrypt Hashing→Database

When for example user is logging in, two bcrypt hashes are compared and if they are identical authentication is successful.

All tables are created using migrations from the command prompt like for example:

php artisan make:migration create_vaults_table

These migrations can then be customized from database/migrations folder inside the project as shown below.

```php
class CreateVaultsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('vaults', function (Blueprint $table) {
            $table->uuid('id')->primary();
            $table->integer('user_id')->unsigned();
            $table->text('name');
            $table->text('description')->nullable();
            $table->string('color', 20)->default('#000000');
            $table->string('icon', 20)->nullable();
            $table->string('background')->nullable();
            $table->text('password')->nullable();
            $table->boolean('active')->default(true);
            $table->boolean('is_fav')->default(false);
            $table->timestamps();

            $table->foreign('user_id')
                  ->references('id')
                  ->on('users')
                  ->onDelete('cascade');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('vaults');
    }
}
```

Figure 7

2.3. Implementation

Register.php

```php
public function updated($field)
{
    $this->validateOnly($field, [
        'email' => 'email|unique:users'
    ]);
}

public function register()
{
    $messages = [
        'first_name.required' => Lang::get('alerts.profile.validation.first_name_required'),
        'last_name.required' => Lang::get('alerts.profile.validation.last_name_required'),
        'email.required' => Lang::get('alerts.profile.validation.email_required'),
        'password.required' => Lang::get('alerts.profile.validation.password_required'),
        'password_confirmation.required' => Lang::get('alerts.profile.validation.password_confirmation_required'),
        'agree.required' => Lang::get('alerts.profile.validation.agree'),
        'captcha.recaptcha' => Lang::get('alerts.profile.validation.captcha_recaptcha')
    ];

    // Validating Form Input
    $this->validate([
        'first_name' => 'required|string|max:255',
        'last_name' => 'required|string|max:255',
        'email' => 'required|string|email|unique:users|max:255',
        'password' => 'required|string|min:8|confirmed',
        'password_confirmation' => 'required',
        'password_hint' => 'nullable|string|max:40',
        'agree' => 'required'
    ], $messages);

    if(setting()->get('recaptcha_enabled') == "true")
        $this->validate([
            'captcha' => 'recaptcha'
        ], $messages);

    // Creating the User
    $user = User::create([
        'first_name' => $this->first_name,
        'last_name' => $this->last_name,
        'email' => $this->email,
        'password' => Hash::make($this->password),
        'password_hint' => $this->password_hint,
        'support_pin' => RPG::Generate('d', 4, 0, 0),
        'access_key' => RPG::Generate('ud', 30, 1, 1),
        'locale' => setting()->get('app_locale'),
        'type' => User::DEFAULT_TYPE,
    ]);
```

Figure 8

Register.php is responsible for user registration and validating the form input. Fields that have “required” attached to them must be completed by the user.

Keyword “unique” checks the email if it is a new email or exists in the database.

When validation is successful, password is hashed using Laravel’s bcrypt engine.

Random password generator (RPG) package is used to generate unique pin and access key for the user.

```php
    // Create the default Personal Vault
    $user->vaults()->create([
        'name' => Lang::get('vault.default_vault_name'),
        'description' => Lang::get('vault.default_vault_description'),
        'icon' => 'hdd',
        'color' => '#000000'
    ]);

    // Loggin in the User
    Auth::login($user);

    // Redirecting to the Dashboard
    return redirect()->route('dashboard');
}
```

Figure 9

Above image shows how the default vault is created for the user.

```php
public function __construct()
{
    $this->middleware(['auth', 'verified', '2fa']);
}
```

Figure 10

Above is a public function that is used throughout the project and is required in most cases. This is to prevent any session takeover and to ensure that the user is who they appear to be.

web.php

```php
/** User Routes **/

// Dashboard Route
Route::get('/', [DashboardController::class, 'redirect']);
Route::get('/dashboard', [DashboardController::class,'index'])->name('dashboard');
Route::get('/dashboard/announcements', [DashboardController::class, 'announcements'])->name('dashboard.announcements');

// Vault Related Routes
Route::get('/vaults', [VaultController::class, 'index'])->name('vaults');
Route::get('/vaults/{vault}', [VaultController::class, 'select'])->name('vaults.select');
Route::post('/vaults/{vault}/authenticate', [VaultController::class, 'authenticate'])->name('vaults.select.authenticate');
Route::get('/vaults/{vault}/lock', [VaultController::class, 'lockVault'])->name('vaults.select.lock');
```

Figure 11

Inside web.php are all the applications routes defined.

Login.php

```php
class Login extends Component
{
    public $email;
    public $password;
    public $remember;

    public function updated($field)
    {
        $this->validateOnly($field, [
            'email' => 'email|exists:users'
        ]);
    }

    public function login()
    {
        $this->validate([
            'email' => 'required|email|exists:users',
            'password' => 'required|string',
            'remember' => 'nullable'
        ]);

        if (Auth::attempt(['email' => $this->email, 'password' => $this->password], $this->remember))
        {
            // Authentication passed...
            $user = Auth::user();

            if ($user->status == 'Active') {
                return redirect()->intended('dashboard');
            }
            else {
                $remark = $user->remark;
                Session::flush();
                Auth::logout();
                session()->flash('banned', $remark);
                return redirect()->route('login');
            }
        }
        else {
            return $this->addError('password', Lang::get('alerts.profile.validation.master_password_password'));
        }
    }

    public function render()
    {
        return view('livewire.auth.login');
    }
}
```

Figure 12

Above lines of code are highlighted how authentication happens when a user is attempting to login. If a user is banned, they are redirected back to login and session is flushed.

TwoStep.php

```php
class TwoStep extends Component
{
    public $otp;

    public function authenticateCode()
    {
        $messages = [
            'otp.required' => Lang::get('alerts.security.validation.otp_required'),
            'otp.digits' => Lang::get('alerts.security.validation.otp_digits')
        ];

        $this->validate([
            'otp' => 'required|numeric|digits:6'
        ], $messages);

        $google2fa = new Google2FA();

        $valid = $google2fa->verifyKey(decrypt(Auth::user()->two_factor_secret), $this->otp, 8);

        if ($valid)
        {
            session()->put('two_factor_authenticated', time());
            return redirect()->route('dashboard');
        }
        else {
            return $this->addError('otp', Lang::get('alerts.security.validation.otp_invalid'));
        }
    }

    public function render()
    {
        return view('livewire.auth.twostep');
    }
}
```

Figure 13

TwoStep class is initiated when the user has 2FA enabled and is attempting to login.

Key needs to be verified by third party google2fa if valid user is brought to dashboard else key is invalid.

DashboardController.php

```php
class DashboardController extends Controller
{
    /**
     * Adding auth middleware to this controller.
     *
     */
    public function __construct()
    {
        $this->middleware(['auth','verified', '2fa']);
    }

    /**
     * Display the main Dashboard page
     *
     */
    public function index(Request $request)
    {
        $user = Auth::user();

        if($user->first_login)
        {
            $user->first_login = false;
            $user->save();
            return view('main.dashboard.first_login');
        }

        $announcement = Announcement::latest()->first();
        $loginIP = $user->previousLoginIp();
        $loginAt = $user->previousLoginAt();

        return view('main.dashboard.index')->with(compact('announcement', 'loginIP', 'loginAt'));
    }

    public function announcements()
    {
        $announcements = Announcement::latest()->get();

        return view('main.dashboard.announcements')->with(compact('announcements'));
    }

    /**
     * Redirect '/' Url to '/dashboard' Url
     *
     */
    public function redirect()
    {
        return redirect('/dashboard');
    }
```

Figure 14

Dashboard controller uses the same middleware as seen before to authenticate the user. Function is used if user is logging in for the first time to display the emergency kit.

CreateVault.php

```php
class CreateVault extends Component
{
    public $name;
    public $description;
    public $icon = 'sun';
    public $color = '#000000';
    public $password;
    public $hashedpass;

    public function createVault()
    {
        $messages = [
            'name.required' => Lang::get('alerts.vault.validation.name_required'),
            'name.min' => Lang::get('alerts.vault.validation.name_min'),
            'name.max' => Lang::get('alerts.vault.validation.name_max'),
            'name.string' => Lang::get('alerts.vault.validation.name_string'),
            'description.required' => Lang::get('alerts.vault.validation.description_required'),
            'description.min' => Lang::get('alerts.vault.validation.description_min'),
            'description.max' => Lang::get('alerts.vault.validation.description_max'),
            'icon.required' => Lang::get('alerts.vault.validation.icon_required'),
            'color.required' => Lang::get('alerts.vault.validation.color_required'),
            'password.min' => Lang::get('alerts.vault.validation.password_min'),
        ];

        $this->validate([
            'name' => 'required|string|min:3|max:14',
            'description' => 'required|string|min:5|max:245',
            'icon' => 'required',
            'color' => 'required',
            'password' => 'nullable|string|min:4'
        ], $messages);

        $this->storeVault();
    }

    public function storeVault()
    {
        $hashedpass = null;

        if ($this->password)
            $hashedpass = Hash::make($this->password);

        $user = Auth::user();
        $vault = $user->vaults()->create([
            'name' => $this->name,
            'description' => $this->description,
            'icon' => $this->icon,
            'color' => $this->color,
            'password' => $hashedpass
        ]);

        if ($user->hasReachedVaultLimit()) {
            $user->notify(new VaultLimitReached());
            if (setting()->get('app_email_alerts') === 'true')
                Mail::to($user->email)->send(new VaultLimitReachedMail($user));
        }

        laraflash($vault->name . ' - ' . Lang::get('alerts.vault.created'), Lang::get('alerts.success'))->success();

        return redirect()->route('vaults');
    }
```

Figure 15

When a user is creating a vault, they can set a password you can see password being hashed highlighted. CreateVault.php also checks if the user has not reached their vault limit.

VaultController.php

```php
/* Authenticate & unlock a Password Protected Vault */
public function authenticate(Request $request, Vault $vault)
{
    $input = $request->all();

    if(Hash::check($request->password, $vault->password))
    {
        $request->session()->put($vault->id.'-pass', true);

        laraflash($vault->name . ' ' . Lang::get('alerts.vault.unlocked'), Lang::get('alerts.congrats'))->info();
        return back();
    }
    else {
        laraflash(Lang::get('alerts.vault.incorrect_password'), Lang::get('alerts.warning'))->danger();
        return back();
    }
}
```

Figure 16

Above is the authentication process that takes place when a user tries to unlock a password protected vault. Hash values are checked of the passwords and if it's a match vault is unlocked.

CSRF Token

```
@csrf
<div class="row">
    <div class="form-group col-md-12 col-12">
```

Figure 17

CSRF tokens are used throughout all of applications views to protect users from Cross-site request forgery attacks.

Site.php

```
/**
 * The attributes that should be encrypted and decrypted on the fly.
 *
 * @var array
 */
protected $encrypted = ['login_id', 'login_password', 'additional_info'];
```

Figure 18

This is one example showing what is encrypted by the AES-256-CBC “protected $encrypted”. Your vault name, folders and authentication codes are encrypted by the same encryption algorithm.

app.php

```
/*
|--------------------------------------------------------------------------
| Encryption Key
|--------------------------------------------------------------------------
|
| This key is used by the Illuminate encrypter service and should be set
| to a random, 32 character string, otherwise these encrypted strings
| will not be safe. Please do this before deploying an application!
|
*/

'key' => env('APP_KEY'),

'cipher' => 'AES-256-CBC',
```

Figure 19

Config/App.php shows the specified 32-character encryption.

RandomGenerator.php

```php
class RandomGenerator extends Component
{

    public $rpg;
    public $size;
    public $level;
    public $dashes;

    public function updated($field)
    {
        $this->validateOnly($field, [
            'size' => 'numeric'
        ]);
    }

    public function mount()
    {
        $this->size = 12;
        $this->level = 'lud';
        $this->dashes = 1;
    }

    public function generate()
    {
        $messages = [
            'level.required' => Lang::get('alerts.dashboard.validation.level_required'),
            'size.required' => Lang::get('alerts.dashboard.validation.size_required'),
            'size.numeric' => Lang::get('alerts.dashboard.validation.size_numeric'),
            'dashes.required' => Lang::get('alerts.dashboard.validation.dashes_required')
        ];

        $this->validate([
            'size' => 'required|numeric',
            'level' => 'required',
            'dashes' => 'required'
        ], $messages);

        $this->rpg = RPG::Generate($this->level, $this->size, $this->dashes);
    }

    public function render()
    {
        return view('livewire.dashboard.random-generator', [
            'rpg' => $this->rpg
        ]);
    }
}
```

Figure 20

Above is the implementation of random password generator seen in the dashboard.

2.4. Graphical User Interface (GUI)

Register

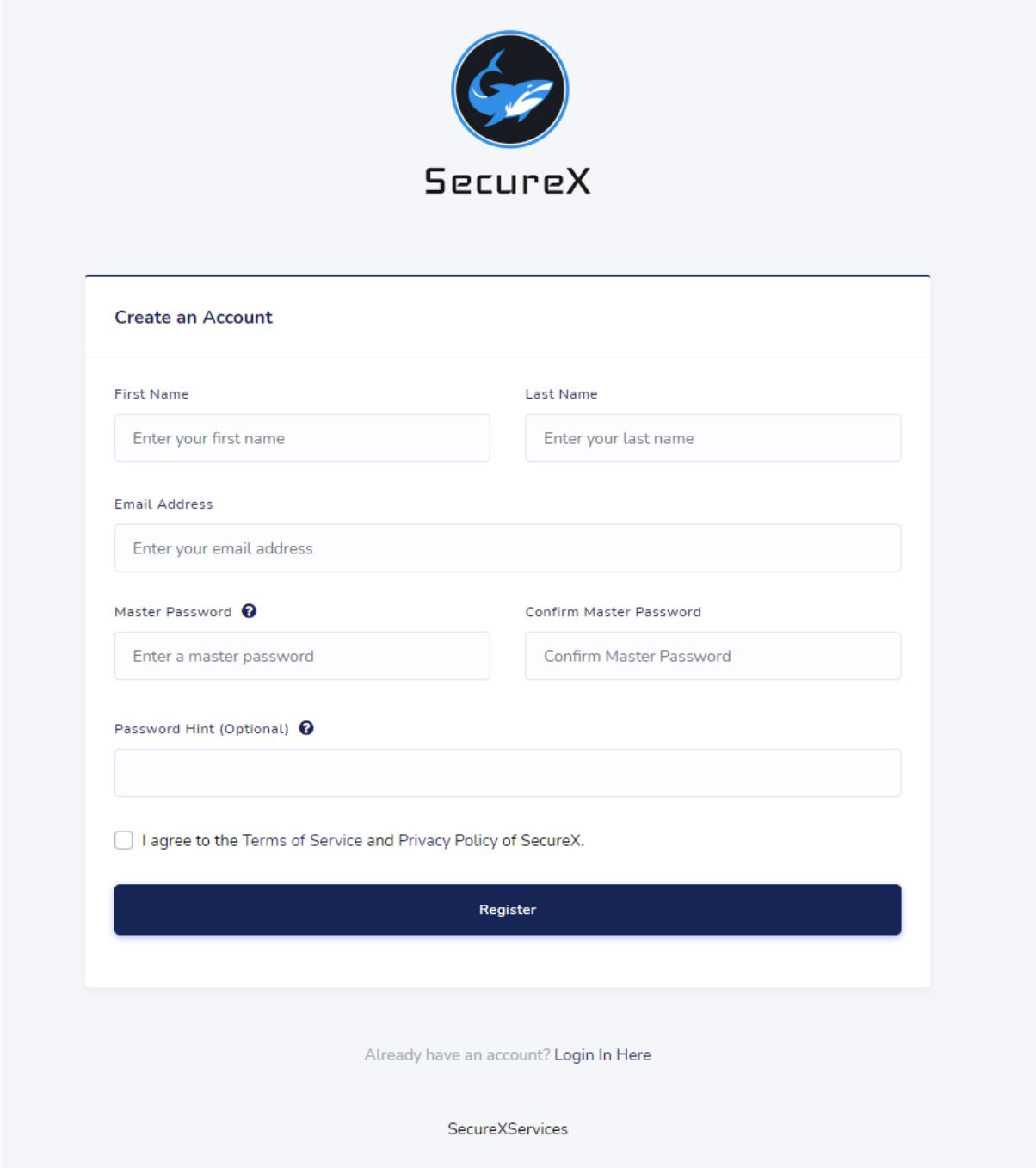

Figure 21

Here is SecureX main register page where a new user has to fill in the following information to register: First Name, Last Name, Email Address, Master password and Confirm Master Password. They can also set a password hint that will give user a hint if they ever forget their password.

Login Page

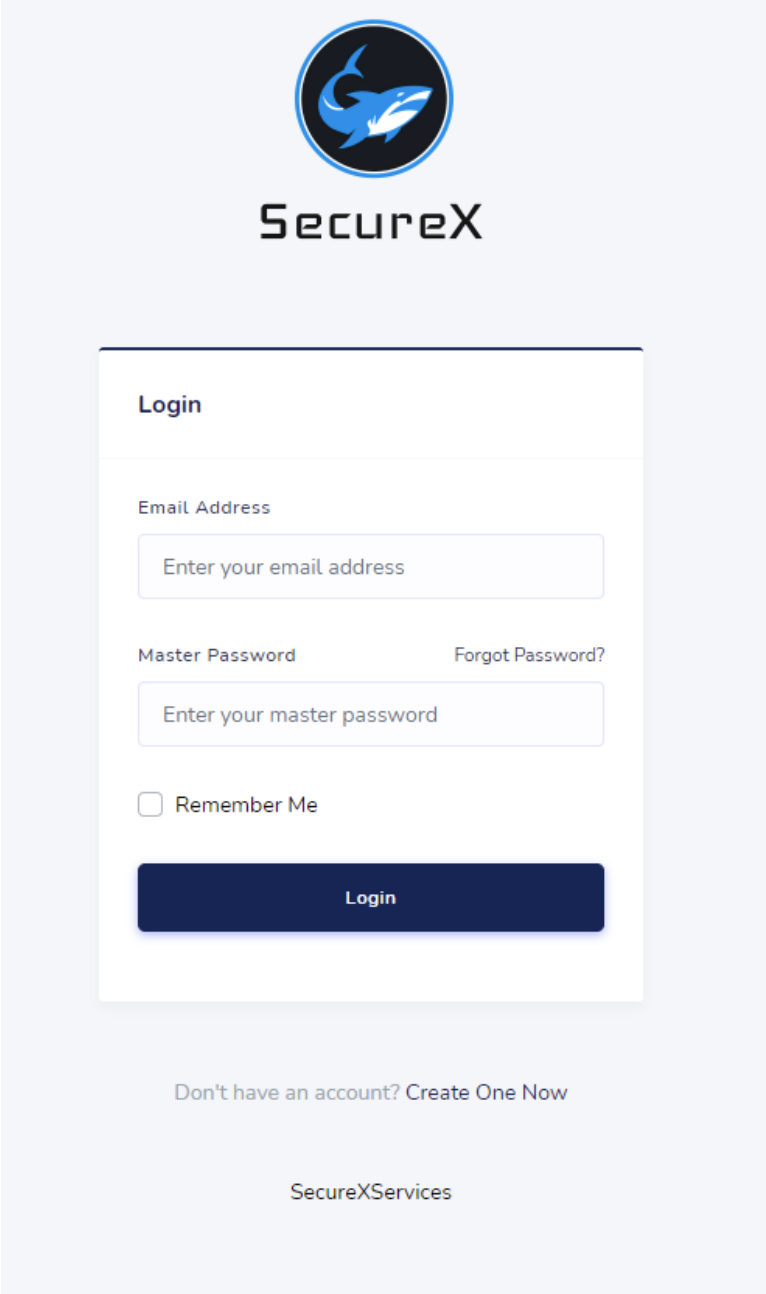

Figure 22

Login page is where you will need to login in order to access main dashboard. Email Address and Master Password is required. You also have options like Forgot Password that will lead to a process of resetting your password. Remember me option will remember your user log in for faster access to the site by creating a remember token. Also, if you do not have an account, you can click Create One Now which would bring you to register page.

Login/Two-Factor Authentication

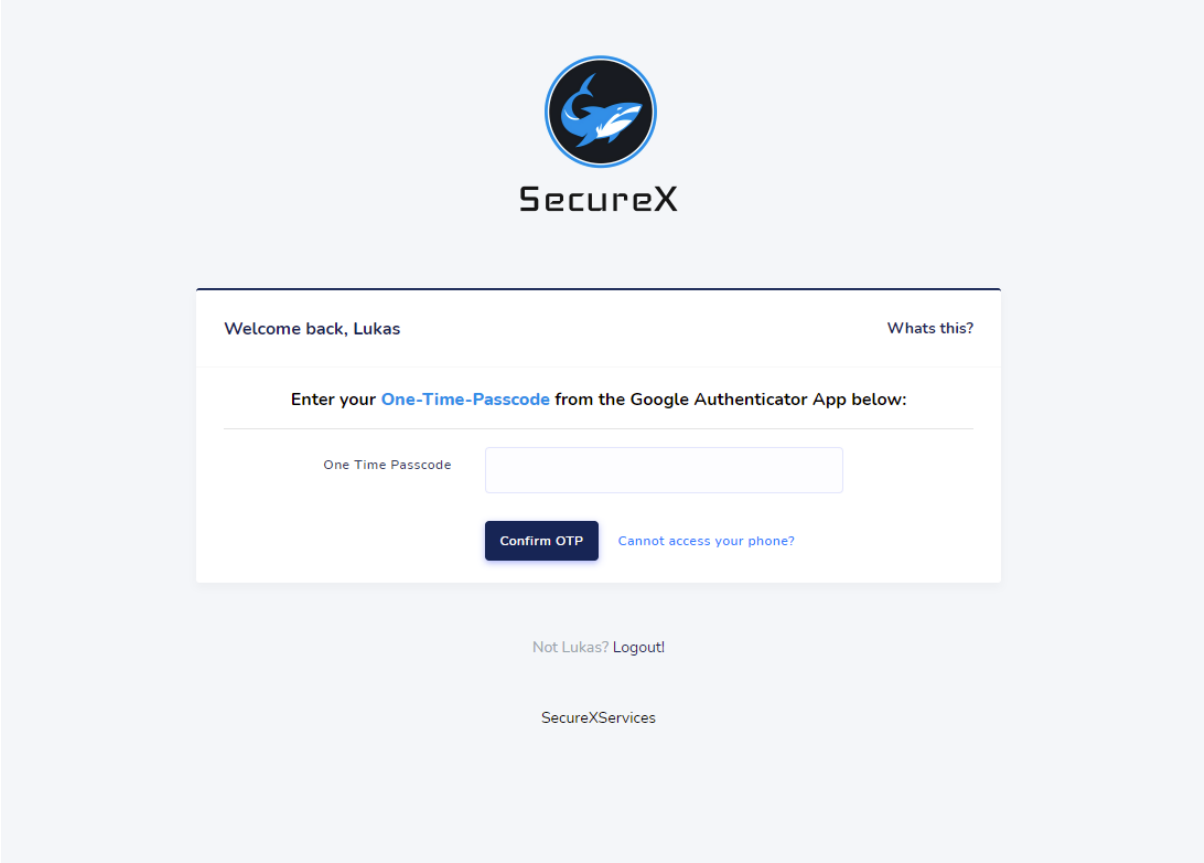

Figure 23

If you have Two-Factor enabled, you will be prompted with a screen to input One-Time-Passcode from Google authenticator app. This is an extra layer of security for your account.

Dashboard/First time logging in message

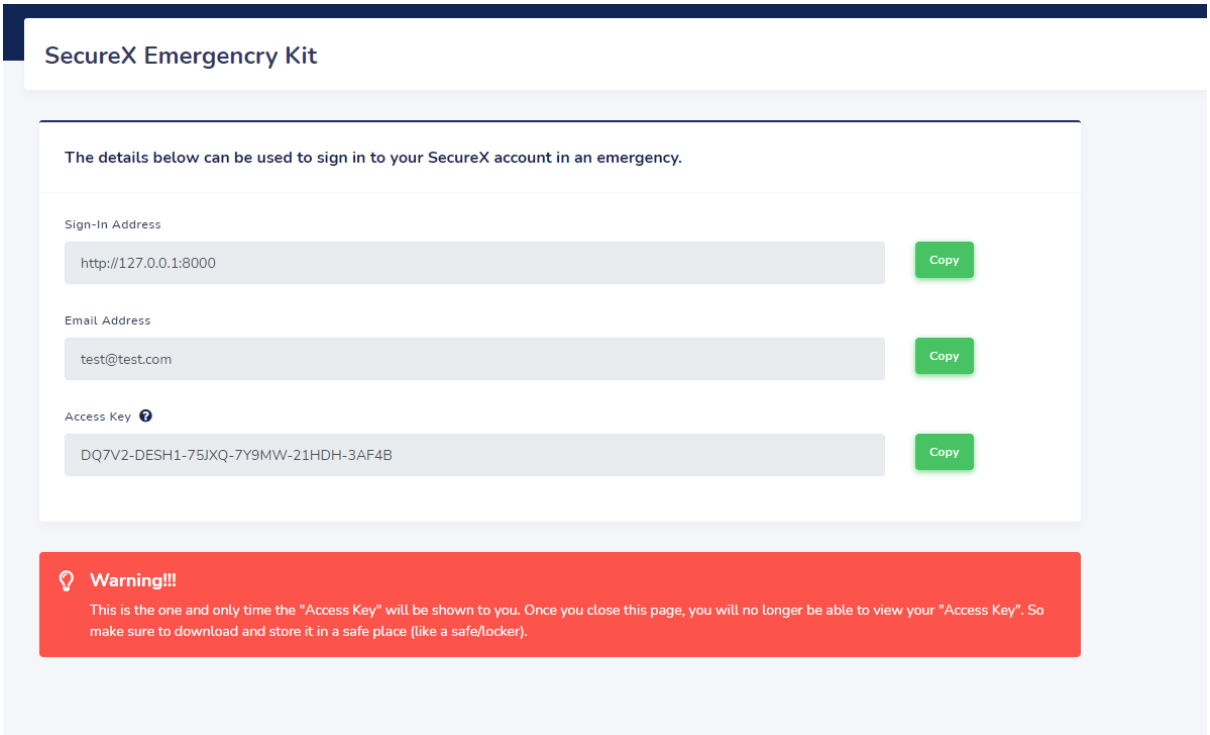

Figure 24

When you first enter the dashboard, you will be provided with an emergency kit. This kit is used for recovering your account. It shows your Sign-in Address Email Address and most importantly your Access Key that you should save somewhere safe. This key is used if you ever lose your two-factor authentication by losing access to your phone you can always use your access key to get back into your account.

User Dashboard

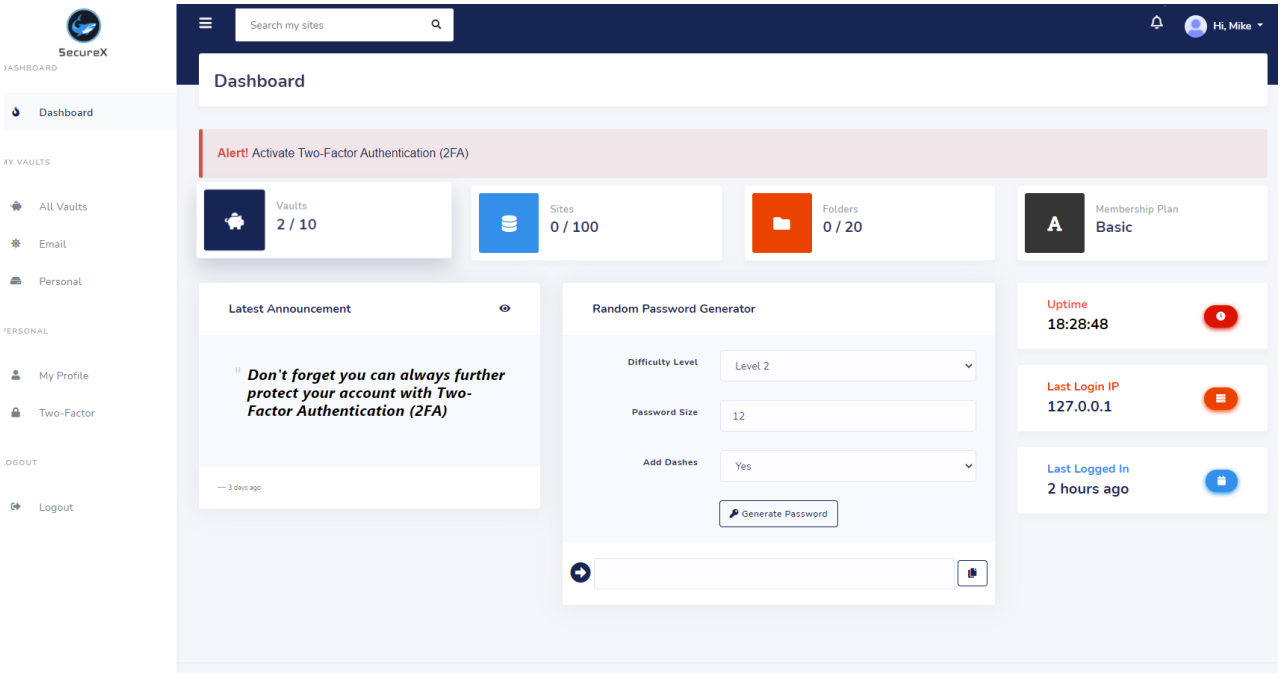

Figure 25

Once the user has logged in successfully, they will be brought to the user dashboard. From here user can access various features like generating passwords, view announcements, view last login IP, Last logged in server uptime, notifications, search bar, total vaults and other statistics related to vaults. Sidebar is used for accessing vaults, my profile, two-factor authentication, and logout for the user.

My Profile

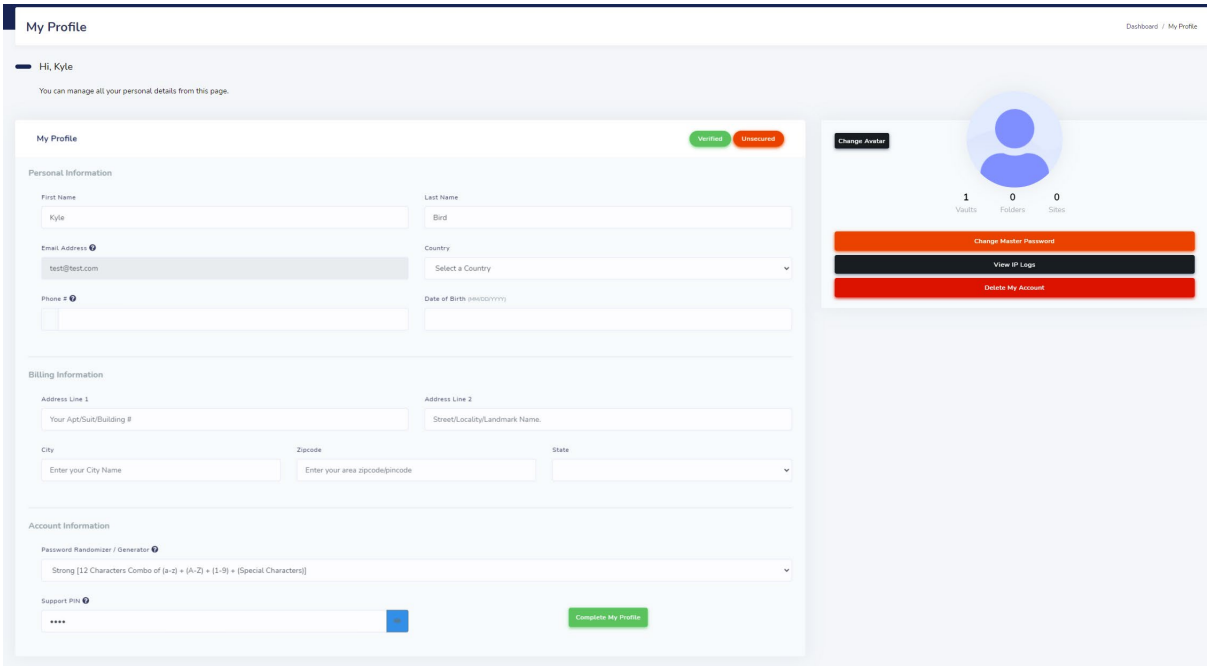

Figure 26

In my profile page you can change all your personal details except email address for security reasons. Email address can only be changed by an admin. Profile page also gives you the ability to change your profile picture, random password generator strength, view support pin, view IP logs and delete your account.

All Vaults

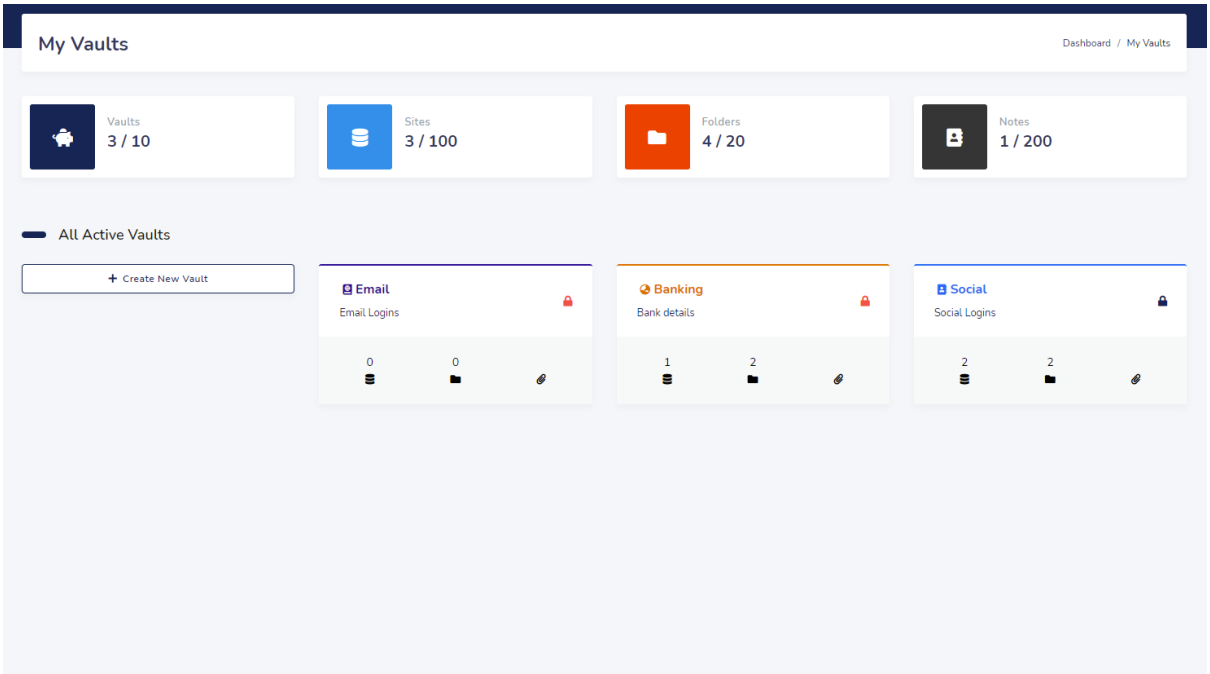

Figure 27

All vaults page lets your create vaults also it will show existing vaults and what is inside them. You can also see locked vaults with a red lock icon, these vaults have a password set on them.

All Vaults/Create New Vault

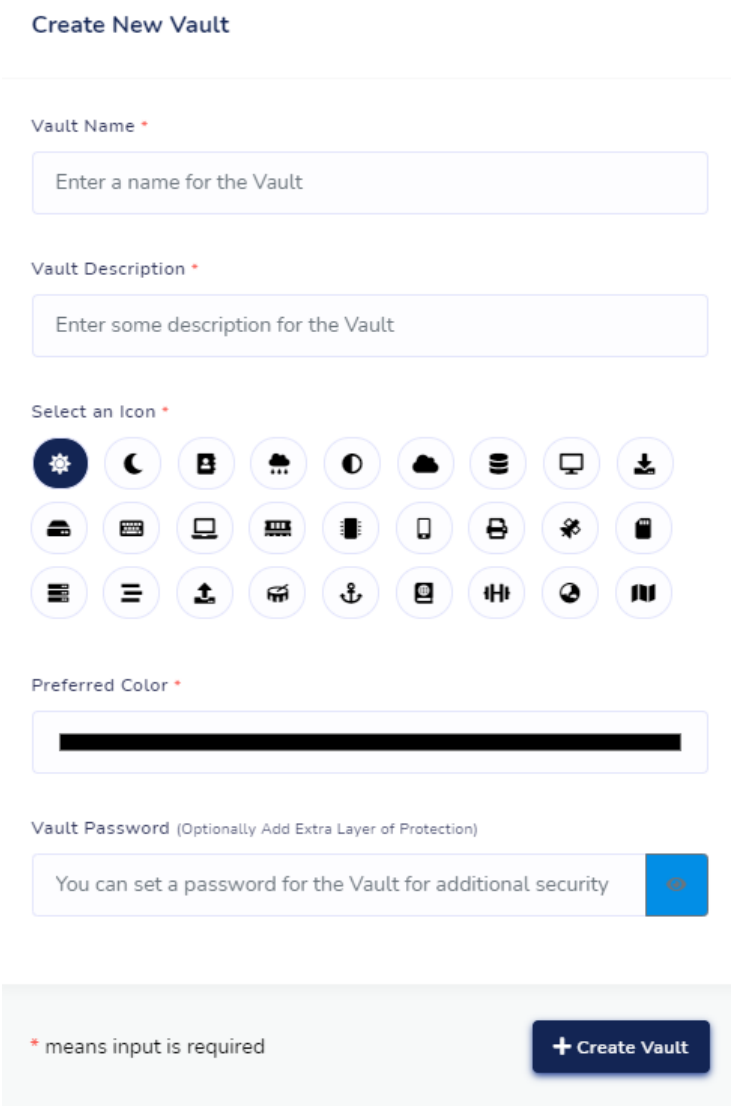

Figure 28

This is the screen that you will get when creating a new vault. You are required to give it a name, description, icon, and preferred colour. Setting a password on a vault means that you will have to unlock that vault with a password after logging in. All details stored in vaults is encrypted.

All Vaults/Add New Site

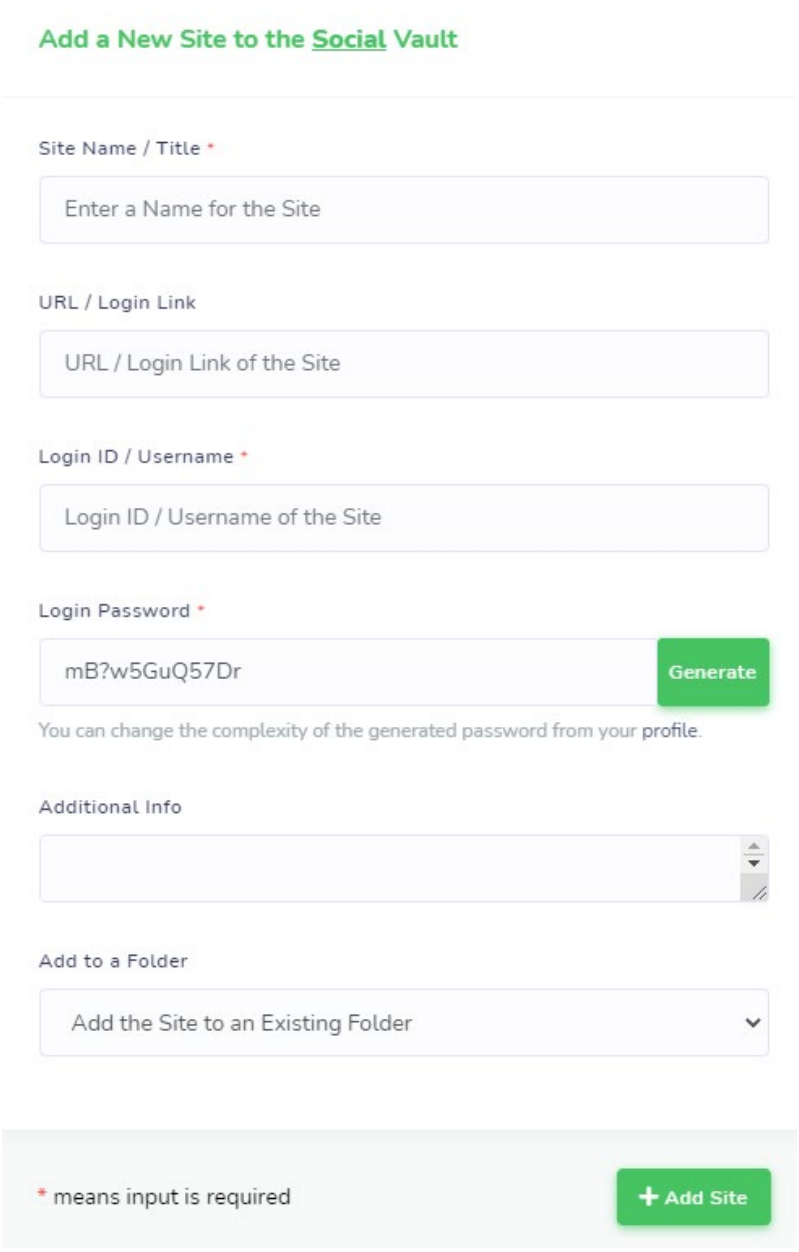

Figure 29

Once you are inside in one of your vaults you can populate it by adding websites with their login details. You can also generate strong passwords from here without needing to remember them. If you wish to be more organised, you can add them to a folder inside a vault.

All Vaults/Create New Folder

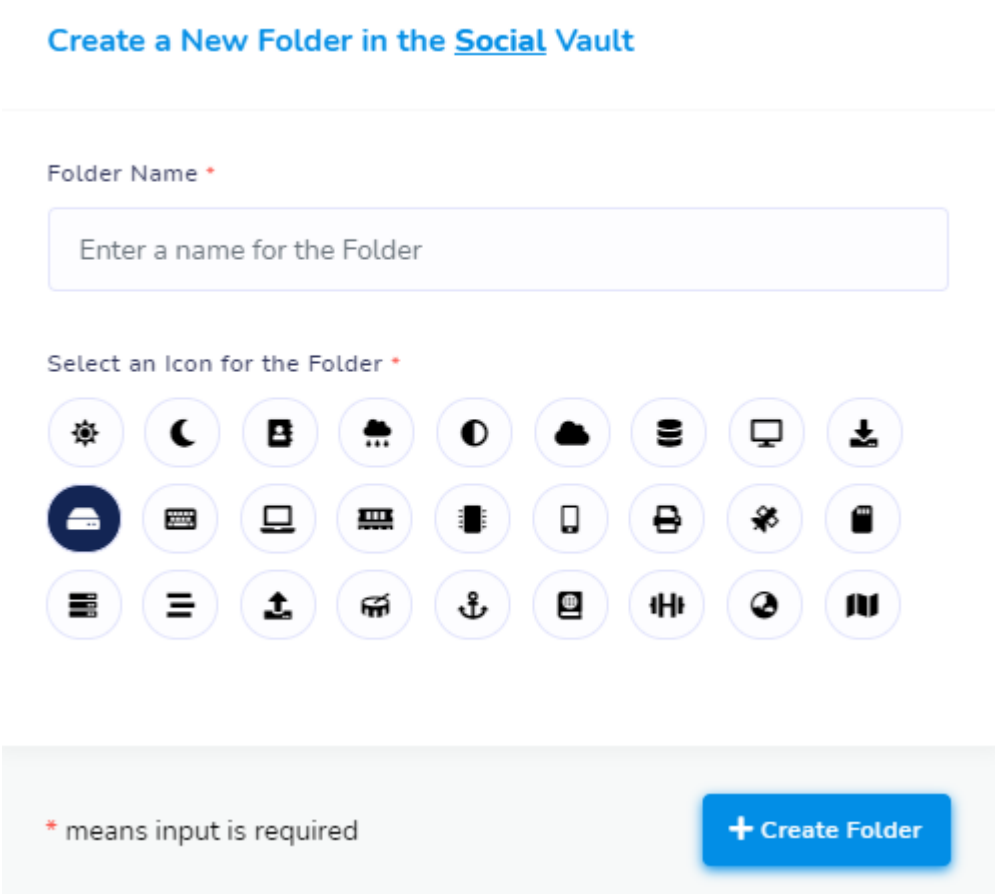

Figure 30

Above screen shows how the create folder GUI form looks like.

All Vaults/Inside Social Vault

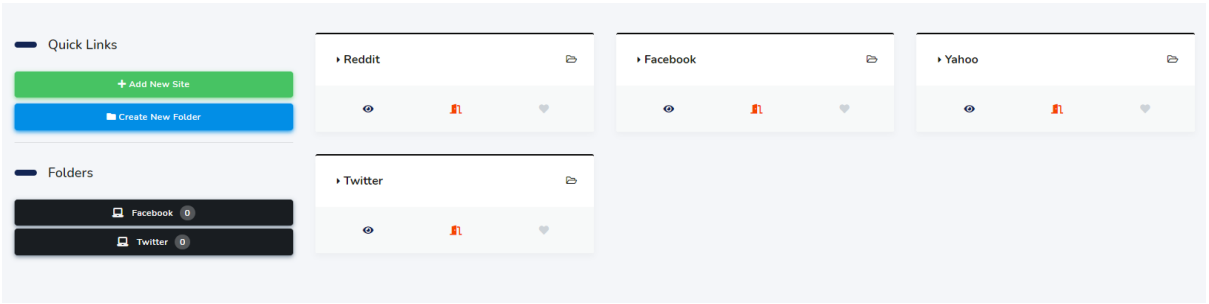

Figure 31

This is how your sites will look once the vault is populated.

All Vaults/Viewing Site

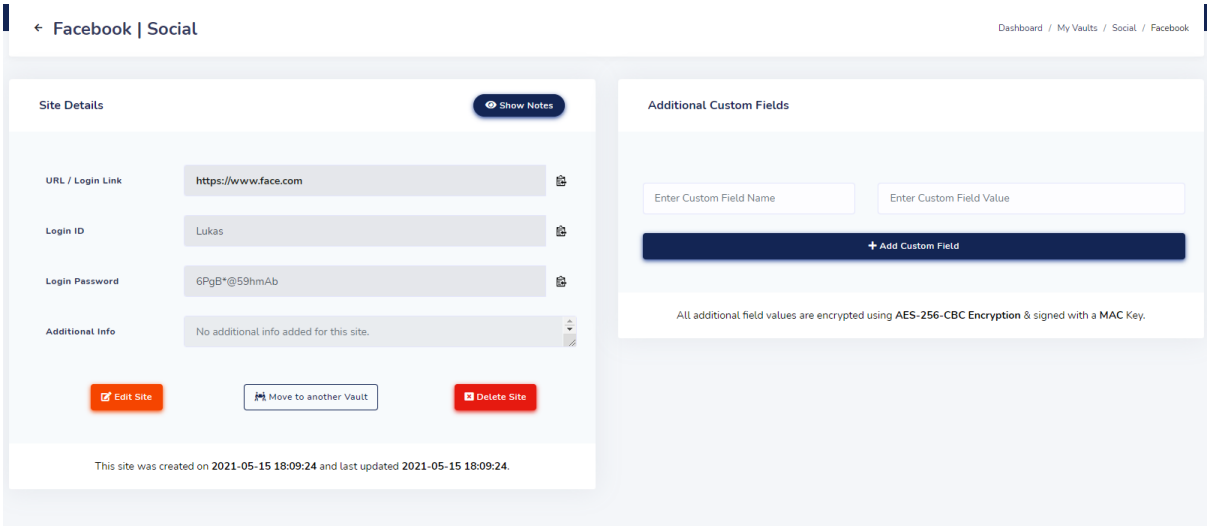

Figure 32

When you successfully added a new site with login details you can access it from your vault, and this is how it looks above. It provides a click to copy function for fast retrieval of data. Here you can also edit site, move it to a different vault or delete it. You can also add custom fields and show secret notes.

Two-Factor

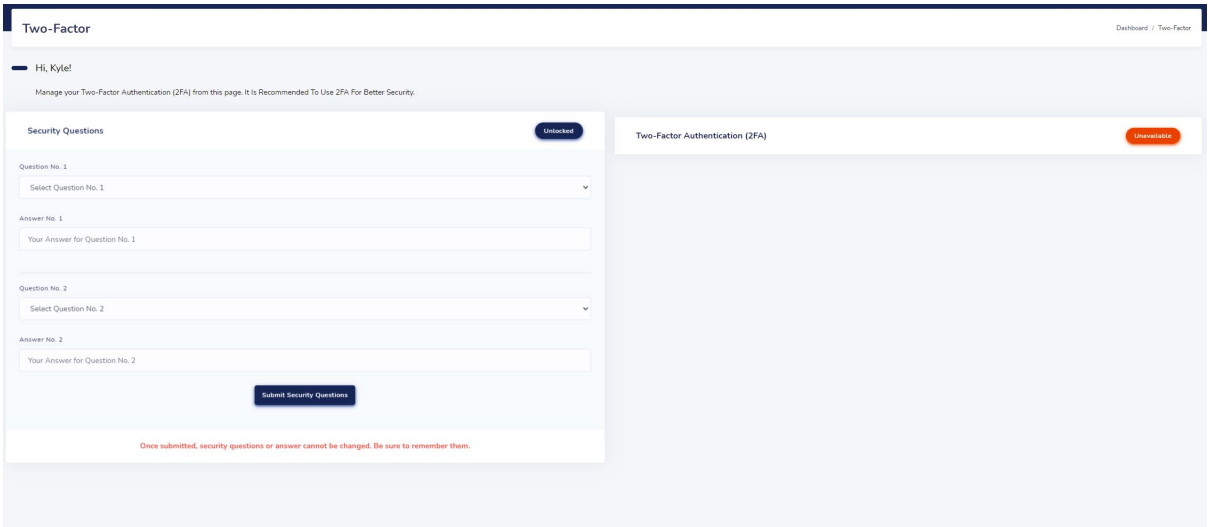

Figure 33

This is the Two-Factor page when you first visit it. You will have to pick two security questions and provide to answers to activate your 2FA authentication with a phone.

Two-Factor/After Submit

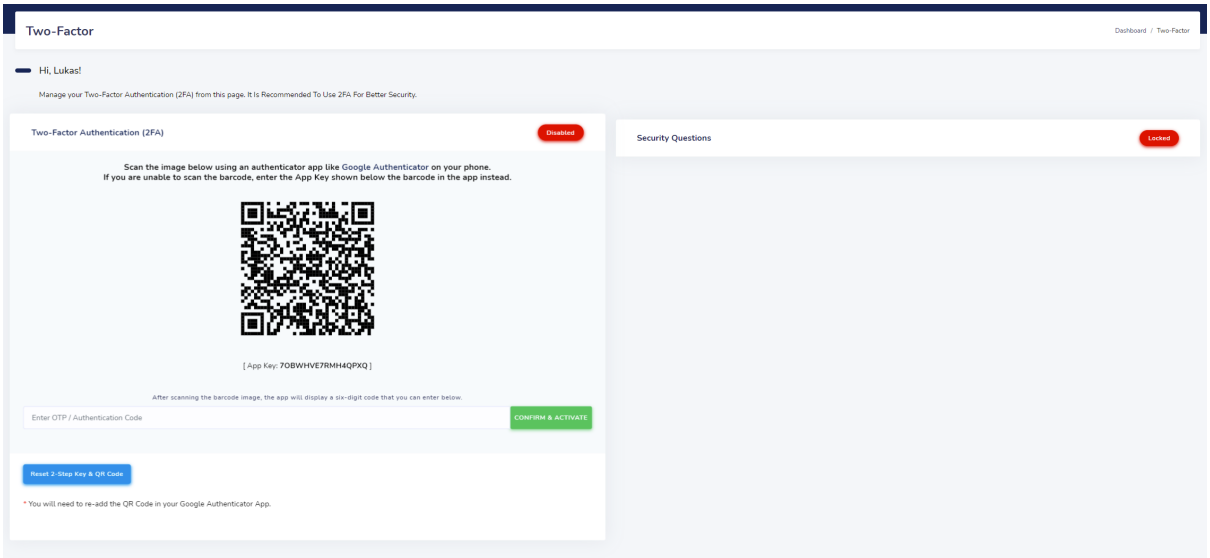

Figure 34

After submitting your answers, they will be locked and the system will generate a unique QR code for the user that can be scanned through your authenticator app to generate an Authentication code to activate your 2FA.

Two-Factor/Activated

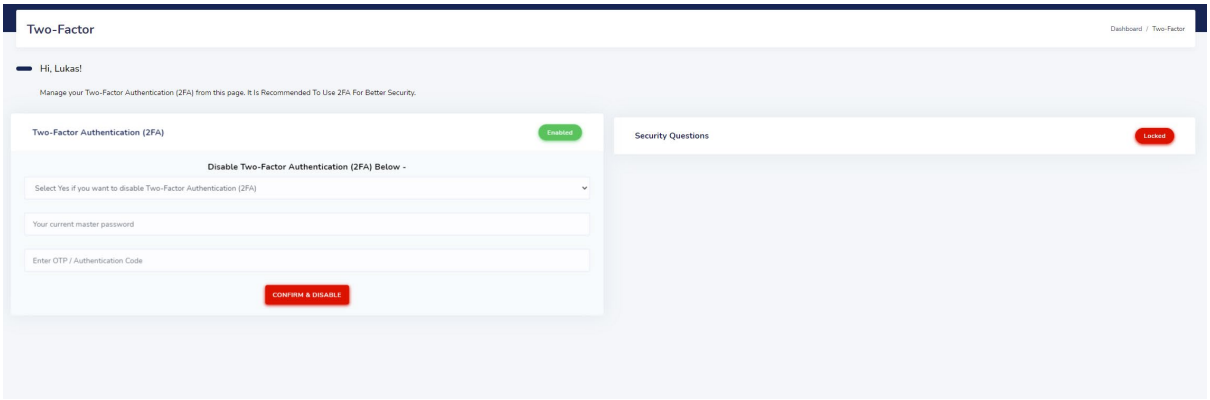

Figure 35

In this screen 2FA is already activated. You can disable 2FA by entering your master password and current authentication code.

Admin Dashboard

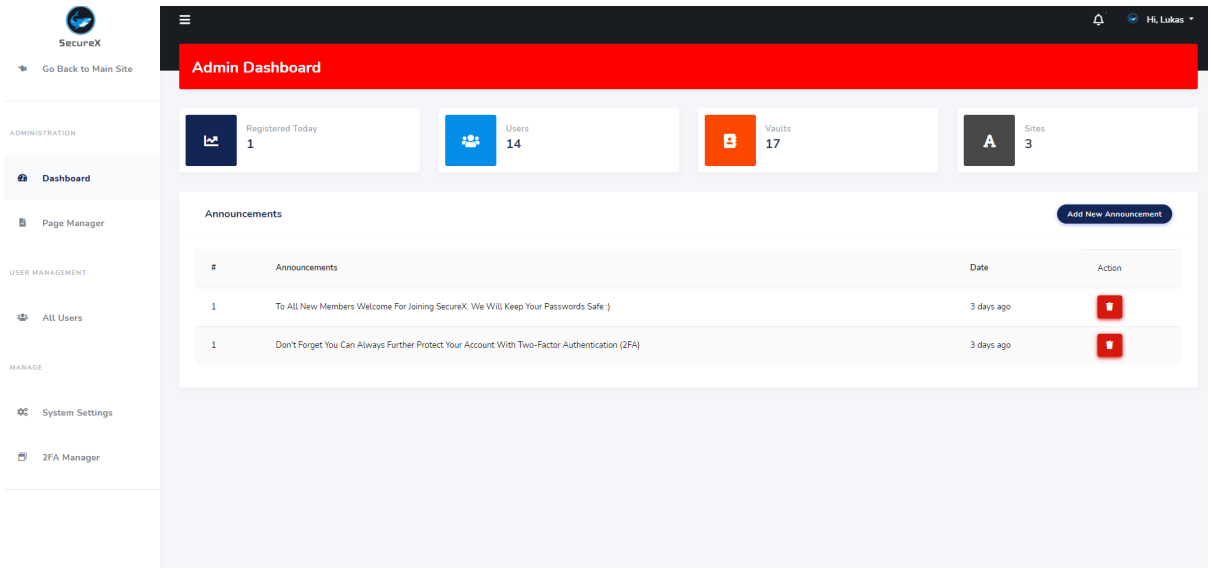

Figure 36

Admin dashboard can only be accessed by admin role. From this dashboard admins can add new announcements that would be visible to everyone from their dashboard. They also have access Page Manager, All Users, System Settings and 2FA Manager.

Page Manager

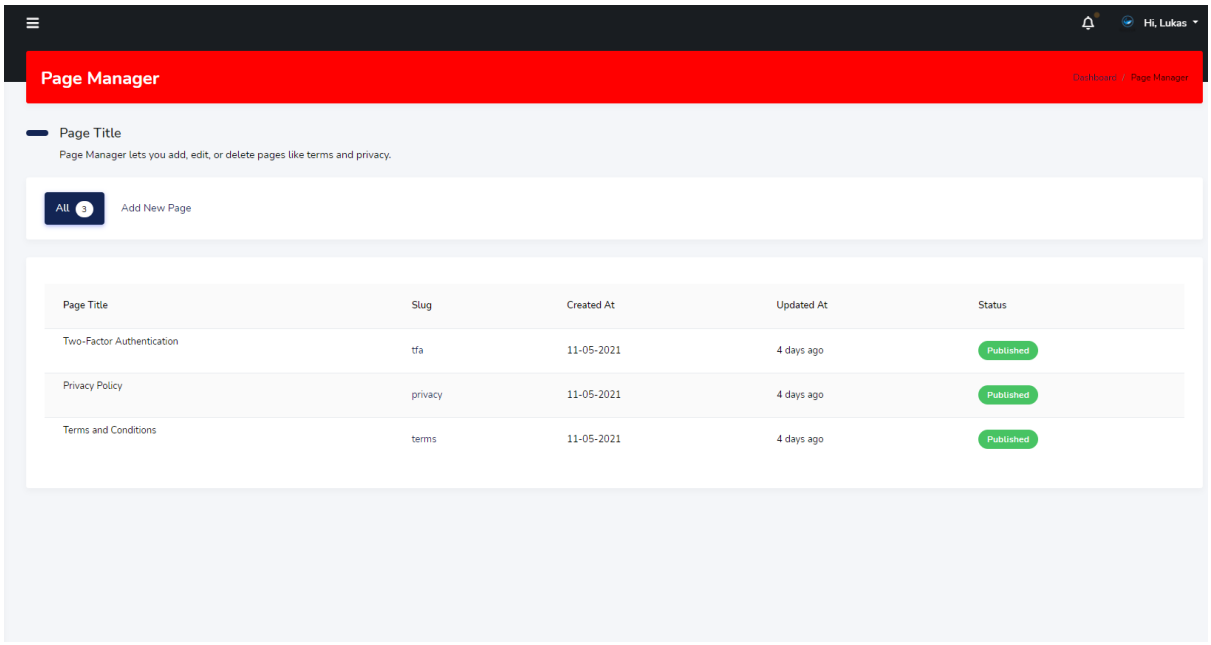

Figure 37

Page manager lets you create new pages or edit existing ones like for example information pages like terms and conditions and privacy policy.

All Users

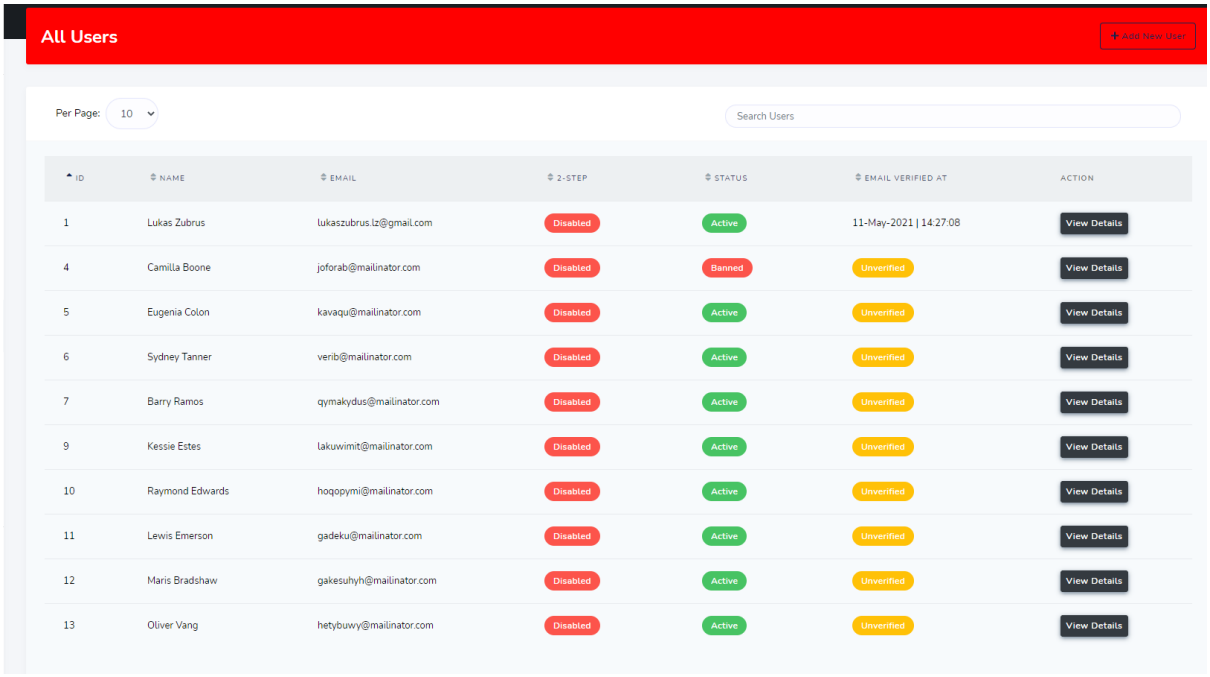

Figure 38

From this page all registered users and their details can be viewed. It also shows accounts with 2FA enabled and accounts that are active, banned, or unverified. Admin can also add new users from here and use the search bar to find a specific user.

All Users/View Details

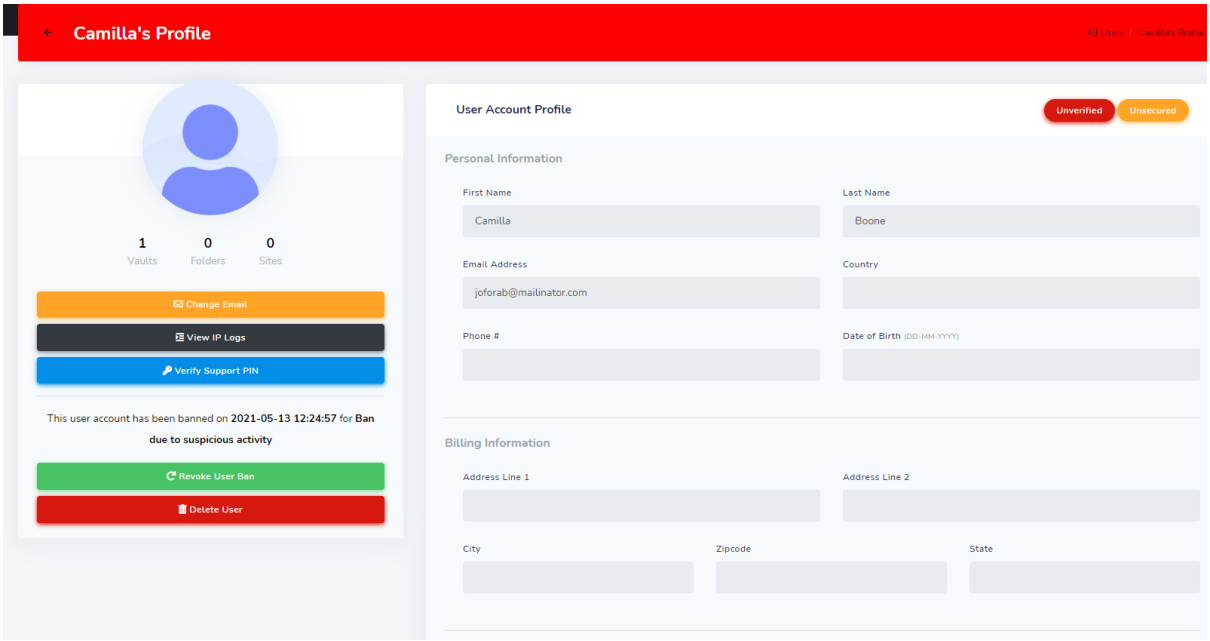

Figure 39

This is an example of user that has been banned due to suspicious activity.

From this page you can change users' email, view IP logs, verify support pin, ban/revoke user and delete user.

Before entering System Settings

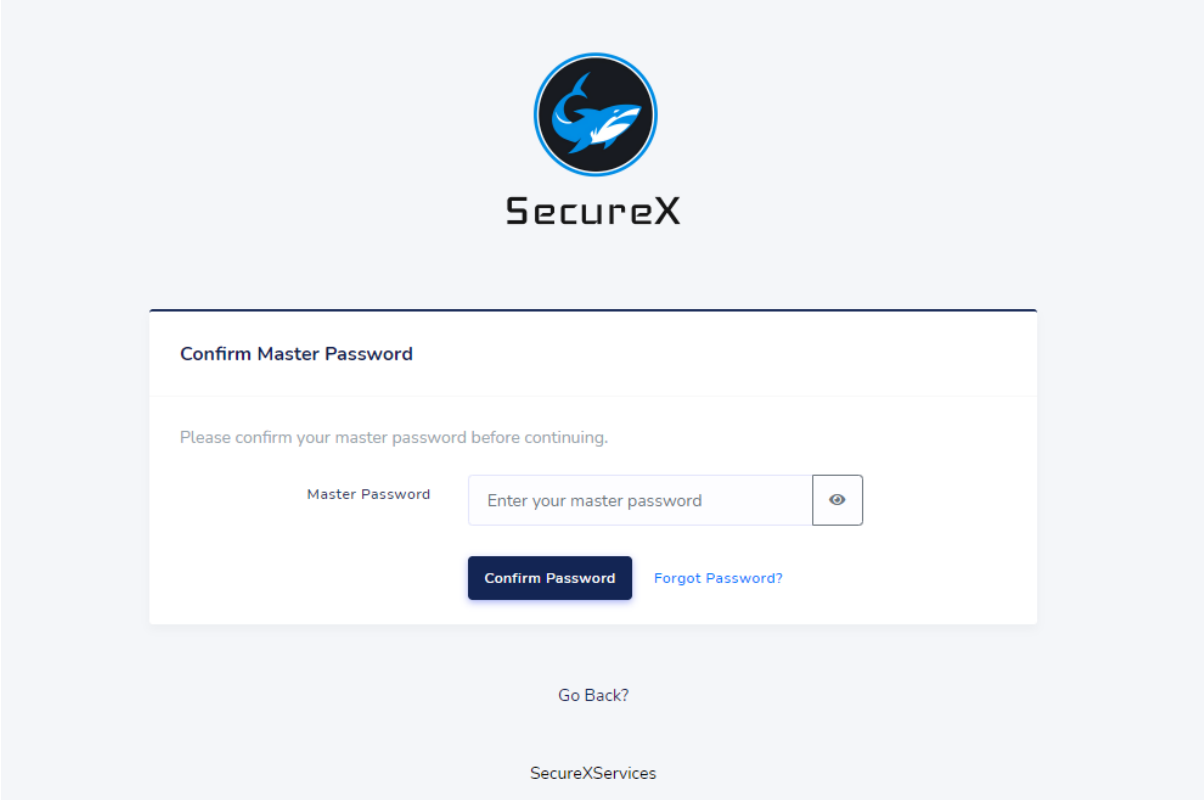

Figure 40

Before entering system settings, you will need to confirm your master password. This is for security reasons.

System Settings/Maintenance Settings

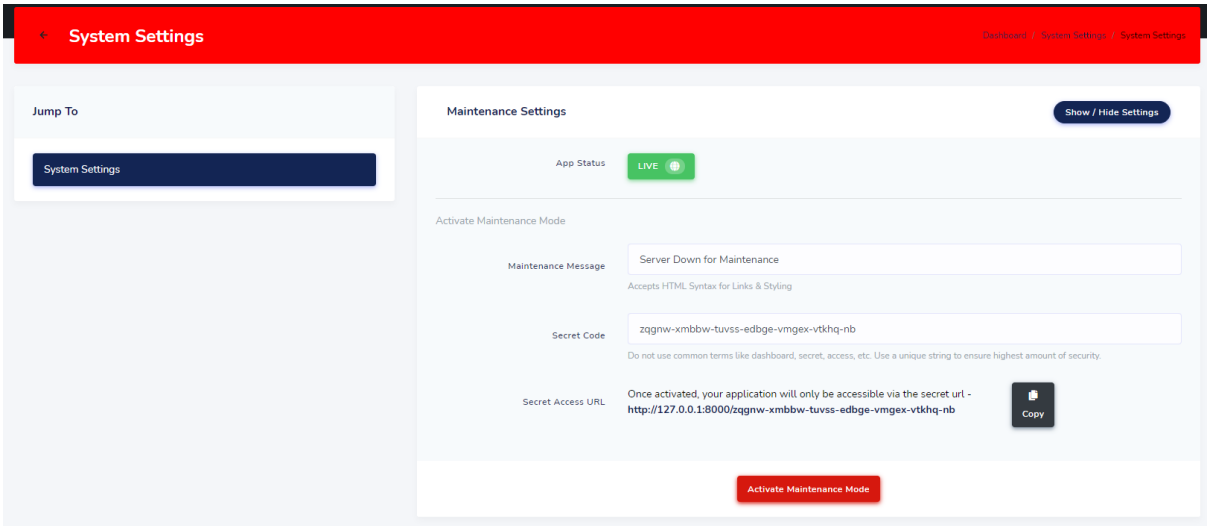

Figure 41

Maintenance mode allows admins to close the website down for maintenance this is especially useful when the website is being hosted online and needs to be updated for whatever reason like a vulnerability was found. Once activated the site can only be accessed through a secret access URL.

System Settings/Application Settings

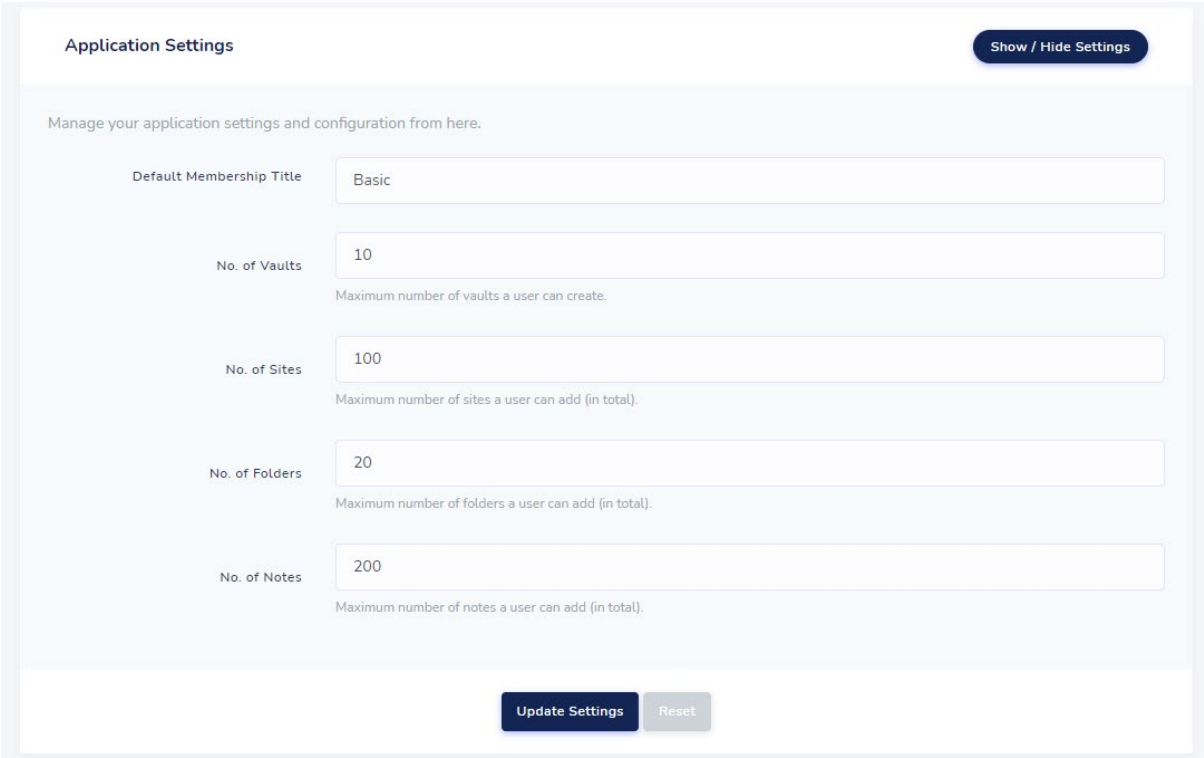

Figure 42

Application settings lets you customize default membership title, number of vaults a user can have, total sites, folders and notes.

System Settings/Access Settings

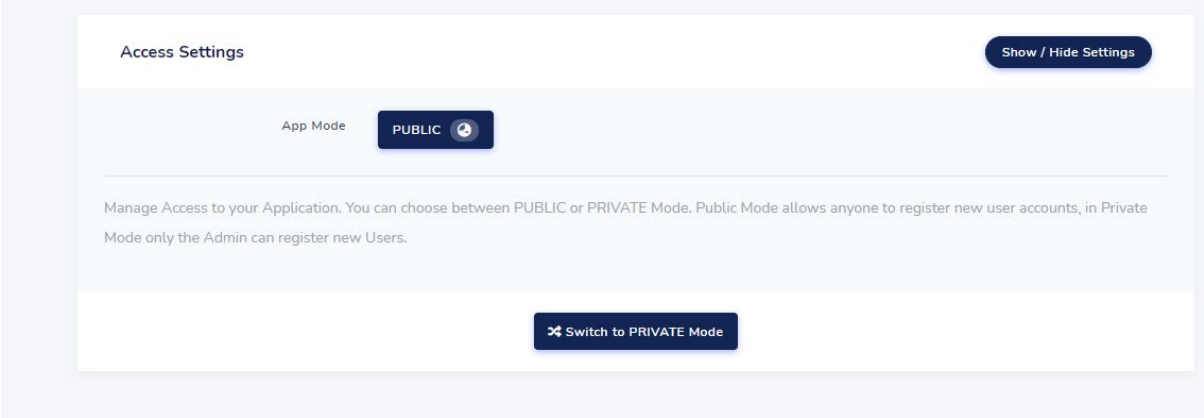

Figure 43

From access settings you can set application to public or private this is also useful for development as it restricts new users registering. Once the application is in private mode only admin can add users from “All Users” page.

2FA Manager

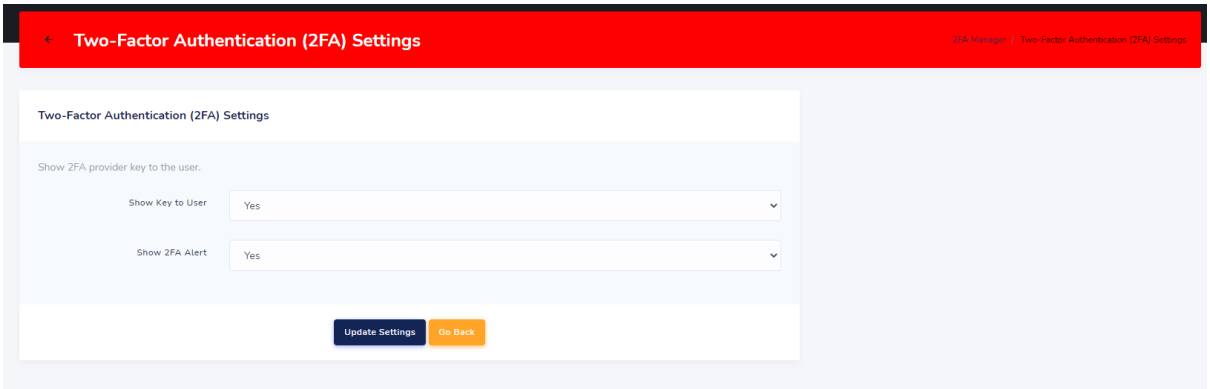

Figure 44

2FA lets you choose whether the user can see the 2FA provider key and if the 2FA alert should be shown to users that don’t have it enabled.

2.5. Testing

The following unit test cases were made.

Authentication Test Case

```
test('users_cannot_authenticate_without_an_email_and_a_password', function() {

    livewire(Login::class)
        ->set(['email' => '', 'password' => ''])
        ->call('login')
        ->assertHasErrors(['email' => 'required', 'password' => 'required']);
});

test('users_cannot_authenticate_with_an_unregistered_email', function() {

    livewire(Login::class)
        ->set(['email' => 'unreg@email.com'])
        ->call('login')
        ->assertHasErrors(['email' => 'exists']);
});

test('users_cannot_authenticate_with_an_invalid_password', function() {

    $user = User::factory()->create();

    livewire(Login::class)
    ->set(['email' => $user->email, 'password' => 'invalidPass'])
    ->call('login')
    ->assertHasErrors('password');
});

test('users_can_authenticate_with_valid_credentials', function() {

    $user = User::factory()->create();

    livewire(Login::class)
    ->set(['email' => $user->email, 'password' => 'password'])
    ->call('login')
    ->assertRedirect(RouteServiceProvider::HOME);

    $this->assertAuthenticated();
});

test('users_without_active_status_cannot_access_dashboard', function() {

    $user = User::factory()->create(['status' => 'Banned', 'remark' => 'Test Ban']);

    livewire(Login::class)
        ->set(['email' => $user->email, 'password' => 'password'])
        ->call('login')
        ->assertRedirect('/login');

    $this->assertGuest();
});
```

Figure 45

Authentication tests to see if user can get authenticated if was banned or with invalid details.

Two-Factor Test Case

```php
beforeEach(function () {
    $this->user = User::factory()->tfa_key()->create([
        'two_factor_enabled' => true
    ]);
});

test('two_factor_challenge_view_can_be_rendered', function() {

    $response = $this->actingAs($this->user)->get('/dashboard');

    $response->assertSee('Two-Factor Authentication (2FA)');
})->group('tfa');

test('two_factor_challenge_contains_livewire_component', function() {

    $response = $this->actingAs($this->user)->get('/dashboard');

    $response->assertSeeLivewire('auth.two-step');
})->group('tfa');

test('two_factor_challenge_fails_without_code', function() {

    livewire(TwoStep::class)
        ->call('authenticateCode')
        ->assertHasErrors(['otp' => 'required']);
})->group('tfa');

test('two_factor_challenge_code_must_be_numeric', function() {

    livewire(TwoStep::class)
        ->set('otp', 'abce')
        ->call('authenticateCode')
        ->assertHasErrors(['otp' => 'digits']);
})->group('tfa');

test('two_factor_challenge_code_must_be_6_digits_exactly', function() {

    livewire(TwoStep::class)
        ->set('otp', '12345')
        ->call('authenticateCode')
        ->assertHasErrors(['otp' => 'digits']);
})->group('tfa');

test('two_factor_challenge_is_not_asked_for_authenticated_user', function() {

    session()->put('two_factor_authenticated', time());

    $response = $this->actingAs($this->user)->get('/dashboard');

    $response->assertSee('Dashboard');

    $response->assertStatus(200);
})->group('tfa');
```

Figure 46

This test case represents activating 2FA and its exceptional flows.

User Registration Form Test Case

```php
test('registration_form_requires_valid_email', function() {

    livewire(Register::class)
        ->set(['email' => 'invalidEmail'])
        ->call('register')
        ->assertHasErrors(['email' => 'email']);
});

test('registration_form_requires_unique_email', function() {

    $user = User::factory()->create();

    livewire(Register::class)
        ->set(['email' => $user->email])
        ->call('register')
        ->assertHasErrors(['email' => 'unique']);
});

test('registration_form_requires_password', function() {

    livewire(Register::class)
        ->set(['password' => ''])
        ->call('register')
        ->assertHasErrors(['password' => 'required']);
});

test('registration_form_requires_password_have_min_8_char', function() {

    livewire(Register::class)
        ->set(['password' => '123456'])
        ->call('register')
        ->assertHasErrors(['password' => 'min']);
});

test('registration_form_requires_password_confirmation', function() {

    livewire(Register::class)
        ->set(['password_confirmation' => ''])
        ->call('register')
        ->assertHasErrors(['password_confirmation' => 'required']);
});

test('registration_form_requires_password_confirmation_to_match_password', function() {

    livewire(Register::class)
        ->set(['password' => 'somePassword', 'password_confirmation' => 'notMatchingPassword'])
        ->call('register')
        ->assertHasErrors(['password' => 'confirmed']);
});
```

Figure 47

Test case for user registration form.

User Registration Valid Data Test Case

```
test('user_is_registered_successfully_if_valid_data_is_provided', function() {

    $initialDispatcher = Event::getFacadeRoot();

    Event::fake();

    Model::setEventDispatcher($initialDispatcher);

    livewire(Register::class)
        ->set([
            'first_name' => faker()->firstName,
            'last_name' => faker()->lastName,
            'email' => faker()->email,
            'password' => 'password',
            'password_confirmation' => 'password',
            'agree' => true
        ])
        ->call('register')
        ->assertRedirect(RouteServiceProvider::HOME);

    Event::assertDispatched(Registered::class);

    $this->assertAuthenticated();

    $this->assertDatabaseCount('users', 1);
    $this->assertDatabaseCount('vaults', 1);
});
```

Figure 48

Test case for successful user registration if data provided is valid.

2.6. Evaluation

System requires to be constantly evaluated by admins of sites usage, timestamps and IP logs.

3.0 Conclusions

The current project has a strong core with many of initially set-out features implemented while having user's security in mind throughout the development. Built on actively maintained Laravel framework ensures that if a vulnerability is found it would be quickly patched by the development team. On the other hand, this may not be as feature packed as other password managers on the market, but it has one of the most clutter free easy to use user interfaces on the market that would appeal to many people. With future implementations and the right marketing, I think this project would have a good chance to compete against the best password managers on the market. If the project would ever go live it would have to be actively maintained by admins to support customers and to monitor IP logs for any suspicious activity.

4.0 Further Development or Research

During the development cycle some things fell out of scope due to external factors like hosting SecureX on a live hosting platform. My original plan was to host the application on Namecheap but when trying to deploy the application I realized the mail providers that Namecheap offers were not compatible with Laravel and thus this plan was scrapped. Extensive research is needed to find a suitable hosting platform for this project. Further development plans were made to create a monthly payment subscription using Stripe that would introduce membership roles that would grant paying users with exclusive features. One of these features was checking if users emails and passwords have been exposed in a data breach by using haveibeenpwned API. Users would be able to manually check for these breaches similar to how password generator works in the dashboard but also users would get notified if one of their sites was in a recent breach and that they should change their details. Another feature that planned is to develop a Google Chrome plugin that would allow registered users to access their login details without having to visit the site, but this may raise security concerns and further research is needed for a secure implementation.

5.0 References

Laravel.com. 2021. *Laravel - The PHP Framework For Web Artisans*. [online] Available at: https://laravel.com/ [Accessed 15 May 2021].

Mordorintelligence.com. 2021. *Password Management Market | Growth, Trends, and Forecasts (2021 - 2026)*.. [online] Available at: https://www.mordorintelligence.com/industry-reports/password-management-market [Accessed 15 May 2021].

ASPER BROTHERS. 2021. *Laravel Security Best Practices - Features to Secure PHP Apps | ASPER BROTHERS*. [online] Available at: https://asperbrothers.com/blog/laravel-security/ [Accessed 15 May 2021].

Pest - An elegant PHP Testing Framework. 2021. *Pest - An elegant PHP Testing Framework*. [online] Available at: https://pestphp.com/ [Accessed 16 May 2021].

6.0 Appendices

6.1. Project Plan

Project Proposal

Objectives

Objective of this software is to provide the user with storage of their sensitive information and ease of access of files and passwords that can be stored safely online.

Background

We all have a lot of online accounts now, between banks, emails, marketplaces, public institutions, games, transport, storage, food delivery… The list goes on.

Vast majority of these accounts that we use today have same or very similar passwords which can make it easy for hackers to gain access to your accounts. Passwords should be very different from one another and not have similarities.

I want to develop an application that would safely store user information like passwords, credit card and identity information that can be accessed easily and quickly while also helping you create unique passwords.

Technical Approach

All the research will be done online on the technical side of the project similar applications will be viewed and analysed. Furthermore, user requirements will be looked at on the forums and then evaluated what features would be added.

Project Plan

Securex
Lukas Zubrus

Project Start:	Sun, 11/8/2020
Display Week:	1

TASK	ASSIGNED TO	PROGRESS	START	END
Review product str	Lukas	100%	11/8/20	11/11/20
Customer research		100%	11/11/20	11/13/20
Feature definition		100%	11/13/20	11/17/20
Design/branding		100%	11/17/20	11/22/20
Register page		10000%	11/12/20	11/14/20
Vault for user logins		25%	1/1/21	1/5/21
Encrypt the vaults		0%	1/3/21	1/8/21
Allow users to edit details			1/8/21	1/11/21
Two factor authentication			1/8/21	1/10/21
Auto generate passwords			1/8/21	1/11/21

Technical Details

Software will be developed in JavaFX, JavaScript and possibly Php if required. Angular framework will also be used for the development of the web application.

Angular is great as it makes it easy to develop applications on multiple platforms. This could be beneficial in the future if I decide to develop a mobile app of the web application. Angular uses TypeScript programming language which is a strict syntactical superset of JavaScript. This will be easier to learn as I am already familiar with HTML, CSS, and JavaScript.

I will integrate the HIBP API into the web app which will let registered users check if their email has been breached on any site that they have registered with the same email.

I am using Namecheap for my domain and server. I am choosing Namecheap as it provides excellent customer support, and it has database tools like PhpMyAdmin which I will need for storing all the data.