One-Time Two-Factor Authentication

CSU33BC1 – Group 4

☰ **Menu**

Introduction to Project

This project is built upon the premise that logging into an account, using only a username and password can be risky. Hackers have discovered many ways to gain unauthorised access to user accounts. Such methods include malware, which is software designed to disrupt, damage or gain unauthorised access to a user account, and key stroking, which involves phishing a user's password by tracking their keys without their consent.

Once a password has been uncovered, the account is now compromised. This may end up becoming an even greater issue if that user reuses passwords and logins for multiple services, for ease-of-use. If an attack has been successful, it is likely that the user is now vulnerable on multiple services, and not only on the compromised computer.

Two factor authentication (2FA) can be defined as *"a security process in which the user provides two different verification factors- their password and a security token, biometric feature etc. to validate themselves in a way which better protects both the user's credentials and the resources the user can access"* (Rouse, 2018) (https://searchsecurity.techtarget.com/definition/two-factor-authentication). 2FA significantly lowers the chance of a user's account becoming compromised as the one-time password changes each time the user logs in. Even if there is malware or key stroking in place, it will be extremely difficult to gain a user's information.

Our Goal

In order to solve the problem outlined in our Background/Problem Statement, we created a one-time two-factor authentication model, 'webAuthV2'. Our webAuth software grants the user the option of logging into their account without entering their full password (preventing key stroking), along with the added security of two-factor authentication.

When signing into an account using webAuthV2, a user is asked to enter their username. A one-time verification code is then sent to the email address/phone number that is registered with the account. This code must be entered correctly for the user to continue to the next stage of login.

The user is then offered the option of signing in via a standard password login, or partial password login. A partial password login involves entering a 4 digit combination of characters derived from the user's full password. This adds an extra layer of protection as potential attackers will not be able to record the complete user password, as only certain characters are entered by the user on any given login attempt. This majorly reduces security risks, even if 2FA gets deactivated by the user.

dervlabrennan . Uncategorized Leave a comment November 20, 2019November 20, 2019 2 Minutes

OT2FA

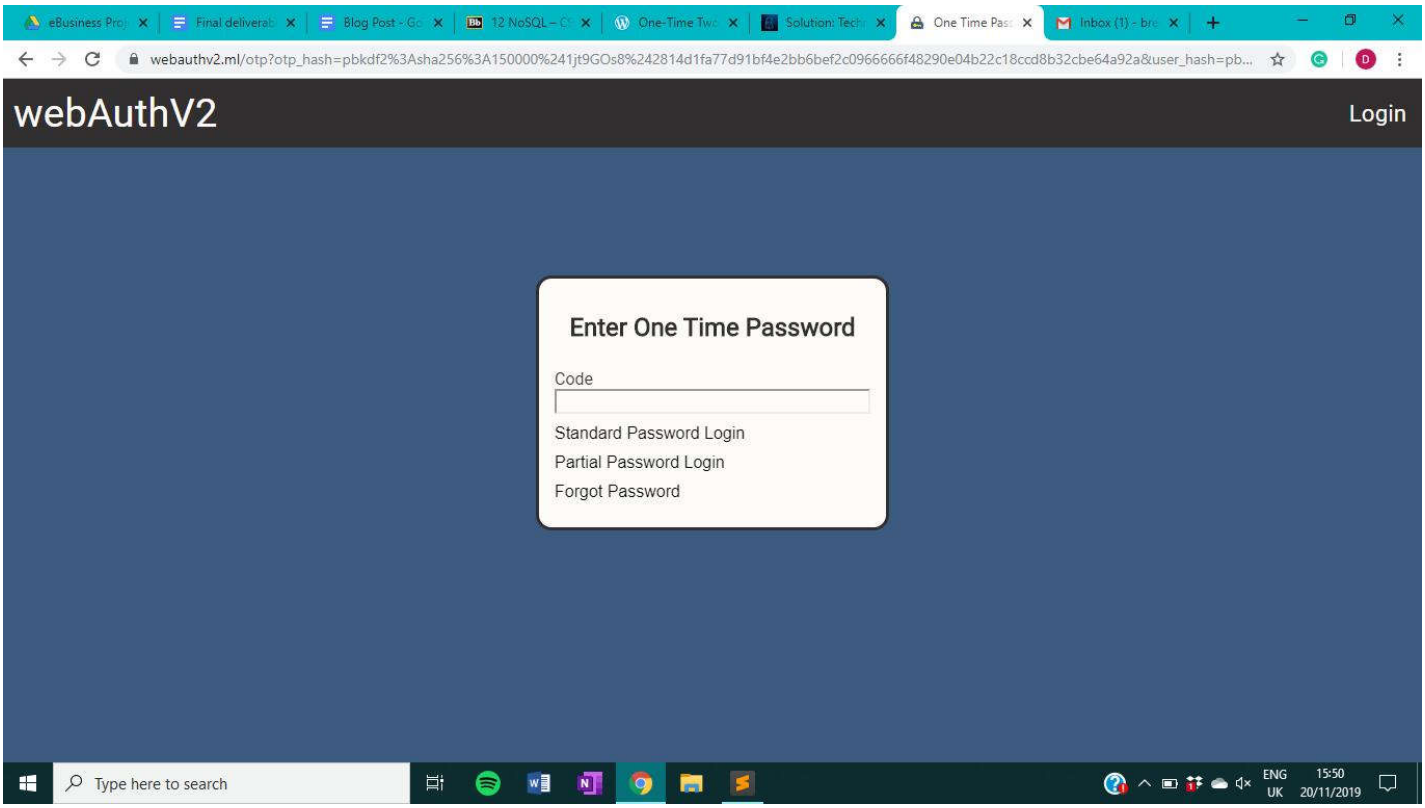

Figure 2: One-Time Password Entry

When the one time password has been entered correctly, the user is then asked would they prefer to enter their password as their standard password or as a partial password for more security.

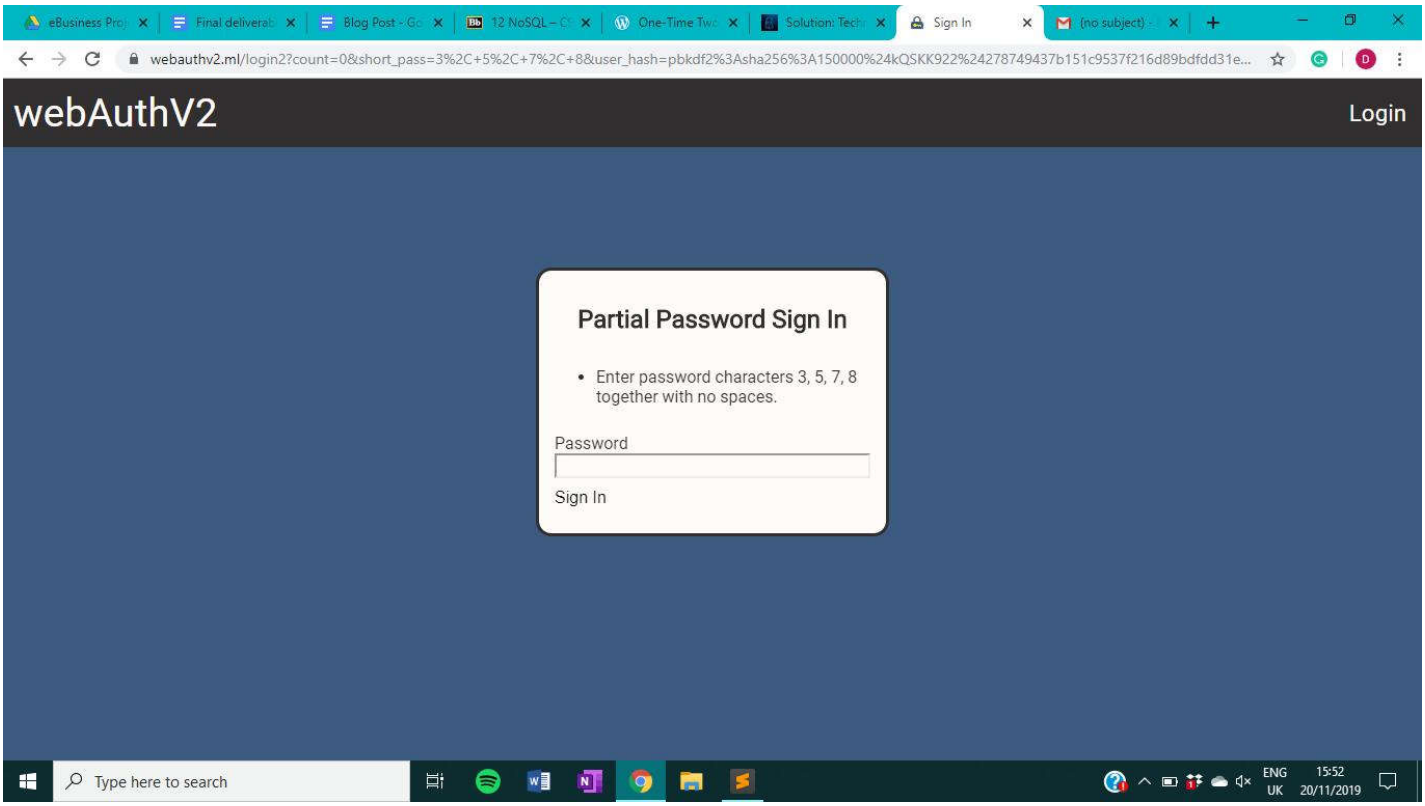

Figure 3: Entering the partial password

Once the partial/standard password is entered correctly, the user is successfully logged in. Here, they can change the css to their liking however if this was to be implemented on an actual website, here the user would be able to access all of the functionality provided.

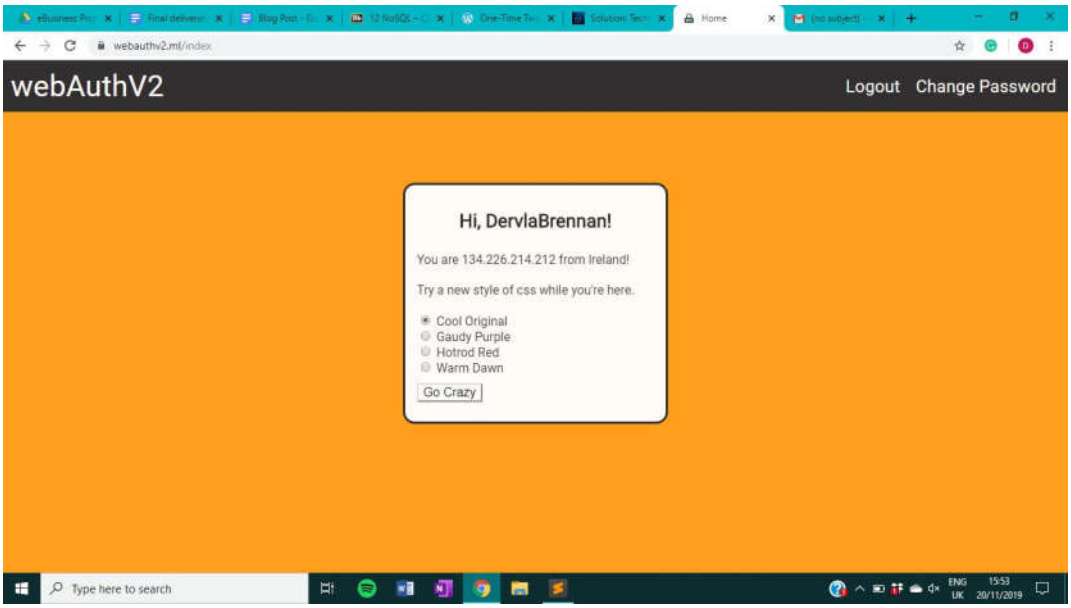

Figure 4: Successful Login

Installation Instructions

The following commands inside the working directory will launch the project with all installed dependencies. it is recommended to set up a virtual environment before using `pip install` .

```
pip install -r requirements.txt
bash fixdb
flask run
```

`fixdb` is a short bash script to generate an SQLite database and do an initial migration, it can be discarded if you wish to run these commands yourself.

A .env file must also be created, with relevant information filled in to allow a production server, SQL support, etc:

```
SECRET_KEY=     // for secure public hosting, can be anything
MAIL_SERVER=        // relative to the email address used
MAIL_PORT=      // server dependent
MAIL_EMAIL=     // owner email address, configured correctly for insecure login
MAIL_PASSWORD=      // relevant password
DB_URL=         // used for a web-hosted SQL database
ENVIRONMENT=        // 0 for local SQLite db, 1 for MySQL with DB_URL
ACCOUNT_SID=        // twilio account
AUTH_TOKEN=     // twilio auth token
USER_PHONE_NUMBER=  // this would be removed upon proper implementation of user phone number in the database
PHONE_NUMBER=       // twilio host phone number
CAPTCHA_PRIVATE=    // provided by Google for ReCaptcha v2 support
CAPTCHA_PUBLIC=     // provided by Google for ReCaptcha v2 support
```

Solution: Technical

Overview of technologies used

When building our one time two factor authentication model, we made use of a number of platforms:

Python

Python was used as our main development language. The source code is tested and compatible with both Python 2 and 3, given certain relevant dependencies are correct.

Flask

We used Flask, an extensible web microframework, to build our web application with Python. One of the main reasons we chose to use this framework was because Flask provides a popular extension called Flask-Mail, which makes the task of sending emails to users very easy. Flask was also aided by a number of libraries which enabled SQL options and intelligent forms for password entry.

SQL Database

We created a database using SQL, which stores all necessary information regarding each user and their account. Random partial passwords are hashed and then stored in this database. When checking if partial passwords are correct, this database is queried, before emailing a one time code to the user.

NGinx:

Nginx is a reverse proxy server that secures and forwards traffic by HTTPS. It is designed for maximum performance and stability which is why we incorporated it into our project (NGINX, 2019) (https://www.nginx.com/resources/glossary/nginx/)

Gunicorn:

Gunicorn is a WSGI server that handles everything between the server and the web application. It translates the multiple requests supplied by NGinx at the same time and works efficiently with our python built front and back end. It also managed multiple instances of a web application which was extremely useful for webauthv2 (vsuplov, 2019) (https://vsupalov.com/what-is-gunicorn/).

Twilio

Twilio is a cloud communications platform that allows software engineers the opportunity the sending and receiving of text messages, calls and voice messages using its web service APIs. (Twilio, 2019) (https://www.twilio.com/) Each Twilio user is given a unique Account SID and Auth Code which is used by the api to facilitate the sending of information. The information is defined by the user in a client function which outlines the receiver of the text (for the purpose of our project we only had the ability to send texts to one number), the body of the message and who the message is from (the Twilio account phone number). When the 'Log in with SMS' button is clicked the Twillio API fires a post request and the message containing the unique OTP to be used by the user is sent to their phone and the hashed OTP is stored in the database exactly as in the Login with Email option. Our Twilio Account information is stored in a .env file to avoid any misuse or unauthorised use of our account.

```
const accountSid = 'AC27429267f69fb92dd092c9d007652ca9';
const authToken = 'your_auth_token';
const client = require('twilio')(accountSid, authToken);

client.messages
      .create({body: 'Hi there!', from: '+15017122661', to:
      .then(message => console.log(message.sid));
```

Architecture Diagram

Figure 5 above sets out the architecture diagram for the webauthV2 system. We used the flask framework for both front-end and back-end with HTML and SQL respectively. We used NGinx to process traffic and requests from users which is translated by Gunicorn.

Flask
web development,
one drop at a time

HTML

NGINX

Reverse Proxy Server

gunicorn

Webserver

Back End

Flask
web development,
one drop at a time

SQL

Figure 5: Architecture Diagram

Entity Diagram

Figure 6 below represents the database that is used to store user information. An ID is created and used as the primary key. User_hash, Password_hash, short_hash1, short_hash2, short_hash3 are depicted as derived attributes as they are derived from Username, password and short_pass1, short_pass2, short_pass3 respectively.

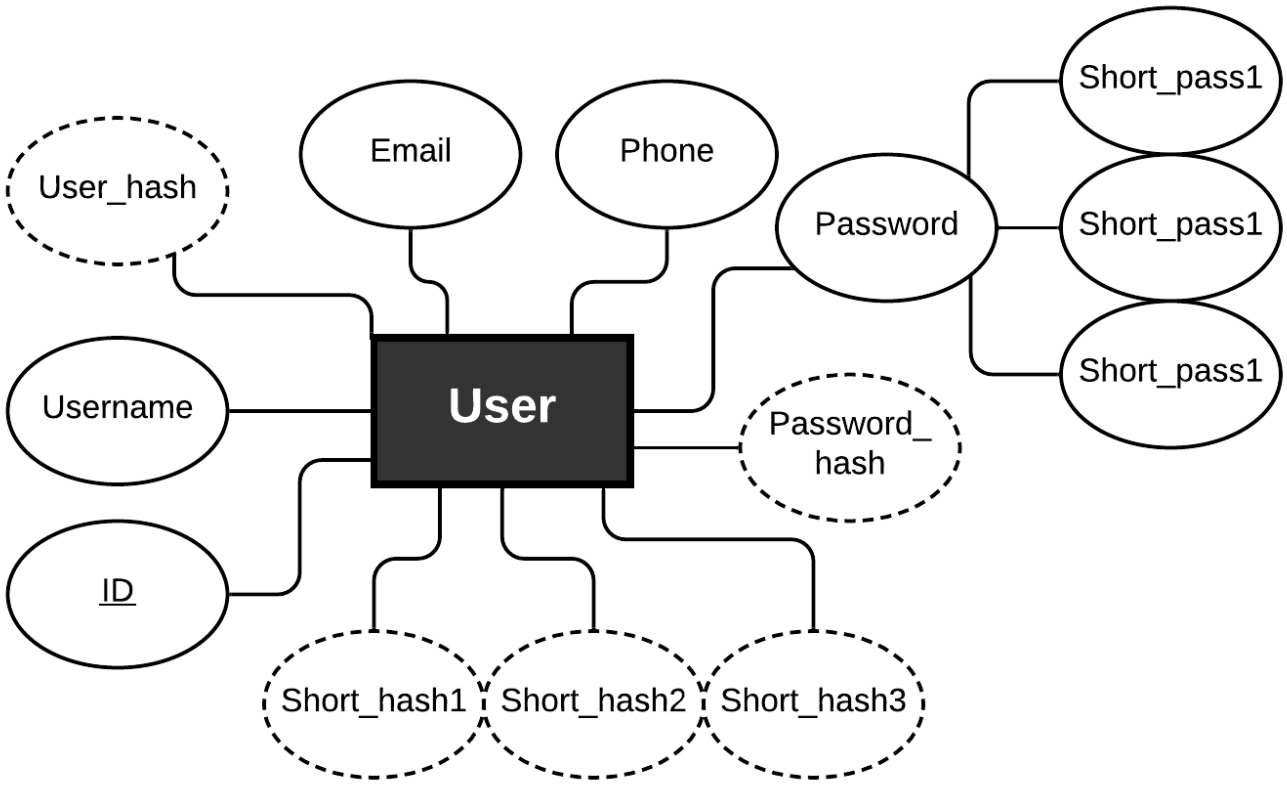

Figure 6: Entity Diagram

Class Diagram

Figure 7 below sets out a UML class Diagram that represents the structure of the webauthV2 system. The user interacts with the webauthv2 site by registering or logging in. webauthv2 stores the user information such as username and password in a database where this password is hashed. Webauthv2 also calls the two api's used in this project, flask-mail api and twilio api. These api's then contact the user with their one time password.

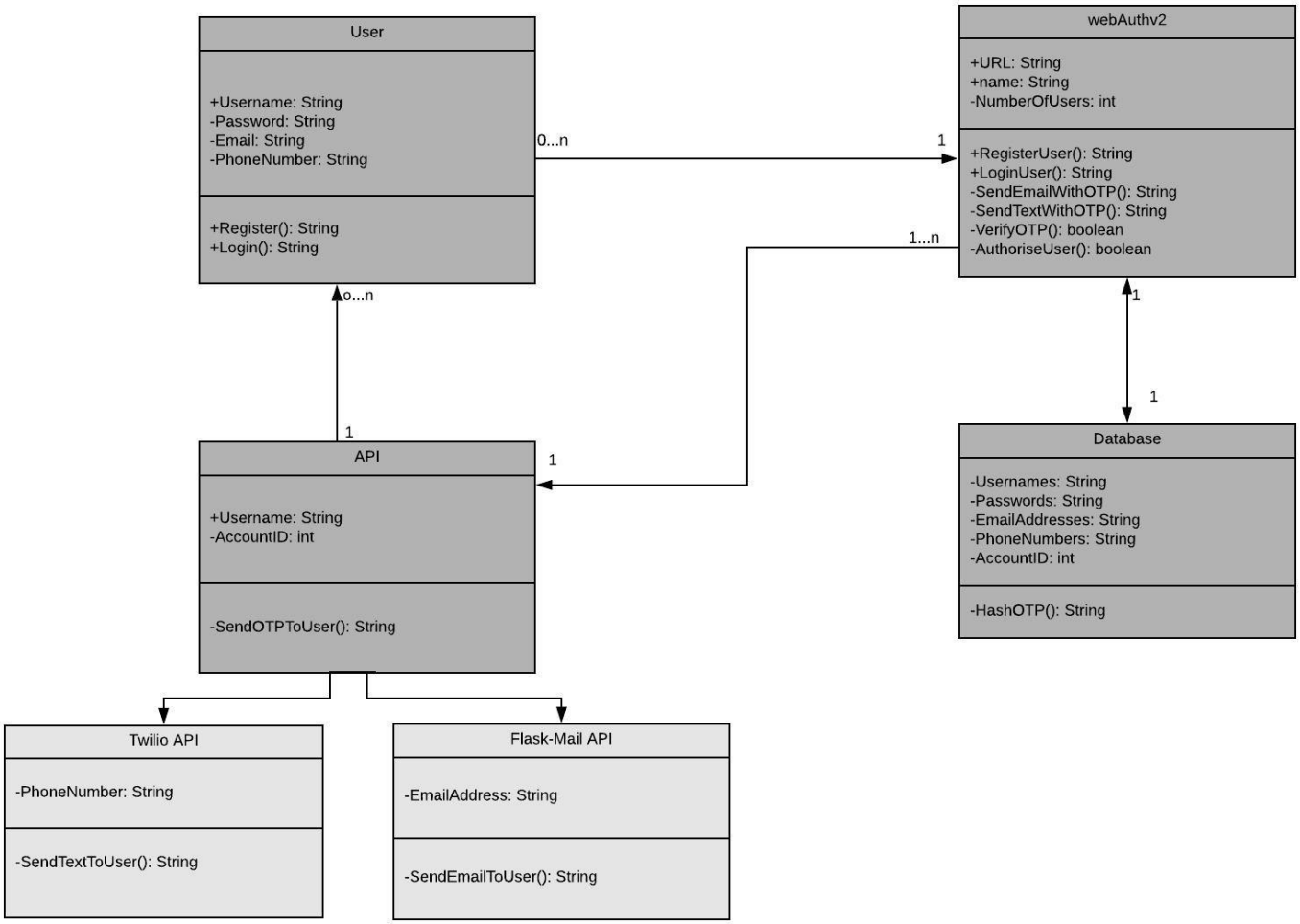

Figure 7: Class Diagram

Activity Diagram

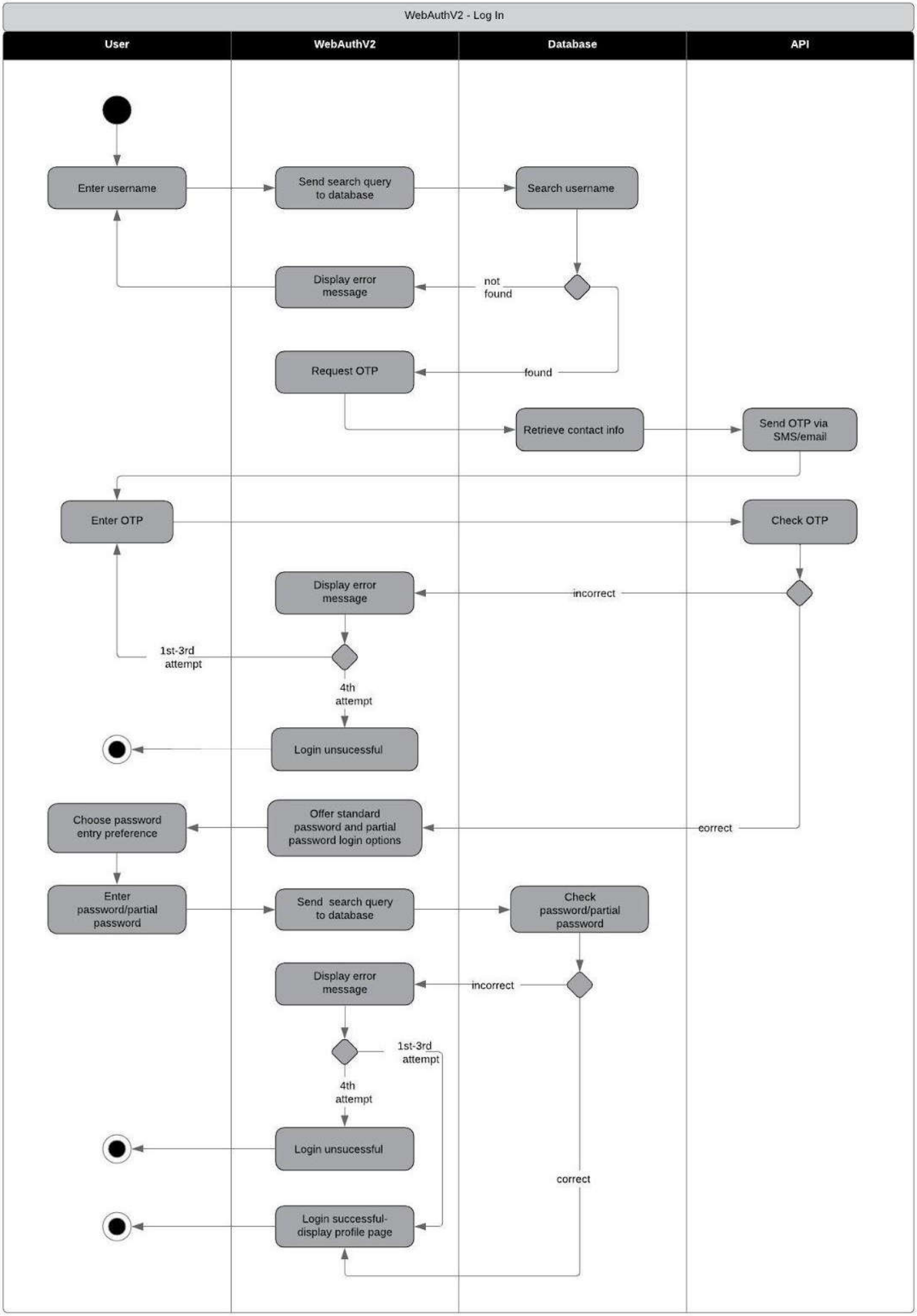

Figure 8: Activity Diagram

Limitations/Outlook

Although we are happy with the prototype that we have produced, given the short time frame and various other commitments involved with a group being 3rd Year college students, there are of course areas that could be improved upon.

- The front end could have been improved overall. We could have added more CSS attributes to the site to make it more visually appealing.
- Although our system helps massively with security and deters hackers, there still remains small vulnerabilities. When logging in, the URL includes the user's hashed username which could be potentially uncovered if a hacker understood this. This could be used to spam someone's email with one time passwords. However, the hacker still would not be able to access the user's account. If we were to continue on with this project, we could implement cookies as an alternative instead of using the POST command which is currently in place. This would counteract the problem with the users hash being revealed.
- At present, the site currently has no support for a user if they forget their password. This function would be implemented if we were to develop this project further.
- The site also only allows for an arbitrary number of developers to work/use the website at the same time. At the moment, this number for webauthv2 is 4. Moving forward, we would increase this number so that more stakeholders could use the site at the same time.
- We also could improve our error handling as although we have some in place, we don't have every case covered
- Due to maintaining this project as 'free to run' the OTA via text functionality is implemented using Twilio's free offer. The database would need to be expanded to include a user phone number, and relevant methods in routes.py would need to be modified to support sending a text to a number other than the listed `PHONE_NUMBER` environment variable.