National College of Ireland

Continuous User Authentication for Secured Messaging Using Advanced Machine Learning Models

MSc Research Project
Cyber Security

Kenechukwu Nwokedi
Student ID: x22248153

School of Computing
National College of Ireland

Supervisor: Liam McCabe

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	KENECHUKWU NWOKEDI…… ……
Student ID:	… x22248153…… ……..……
Programme:	CYBERSECURITY…… **Year:** …2023/2024 …… …
Module:	MSC RESEARCH PRACTICUM/INTERNSHIP…… ……..……
Supervisor:	…LIAM MCCABE……..……
Submission Due Date:	……..……
Project Title:	… CONTINUOUS USER AUTHENTICATION FOR SECURED MESSAGING USING ADVANCED MACHINE LEARNING MODELS…… ……
Word Count:	…1516…… **Page Count**…20……..……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	…N.K.C…… …
Date:	…11TH AUGUST 2024……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□

Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Table of Contents

TABLE OF FIGURES

Configuration Manual

1 Introduction

This configuration manual details the process and steps taken in the development of a continuous user authentication system for a custom messaging application. Additionally, it describes the settings and software tools needed to replicate the experimental setup for the project.

2 System Specification

The system configuration used in the project are:

- Operating System: Mac OS Ventura version 13.6.6
- Processor: 2.3 GHz Dual-Core Intel Core i5
- Hard Drive: 250GB
- RAM: 8 GB 2133 MHz LPDDR3

3 Software Tools

Some of the software tools used to implement this project are:

- Python
- Google Colab
- Visual Studio Code 1.88.1

3.1 Software Installation

This presents the processes taken in installing the tools used.

- Download and Installation of Python 3.12.4. The download link is https://www.python.org/downloads/

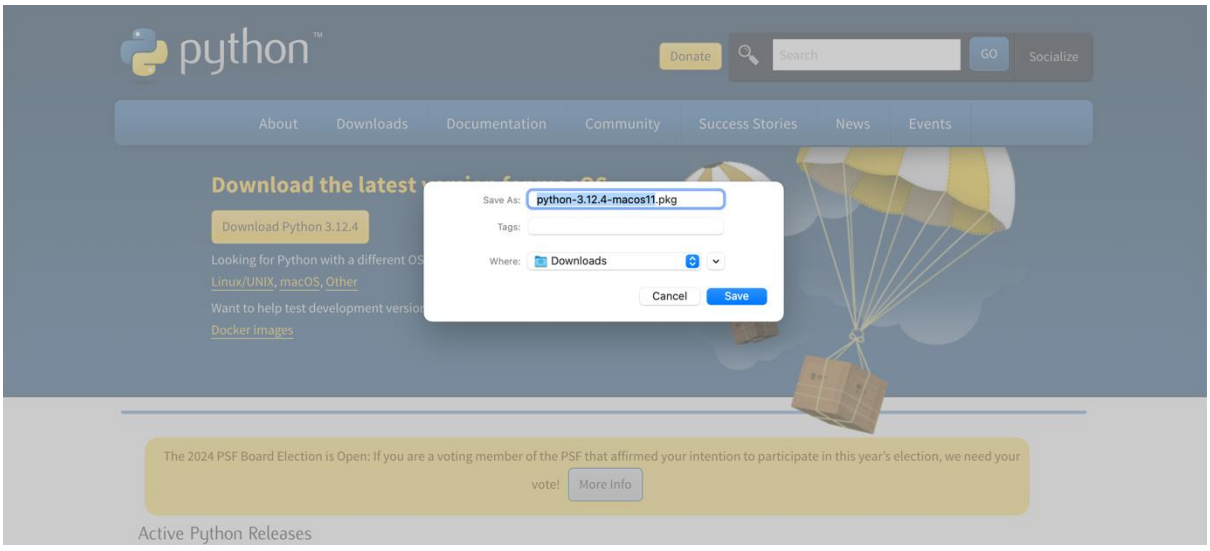


Figure 1. Python Download

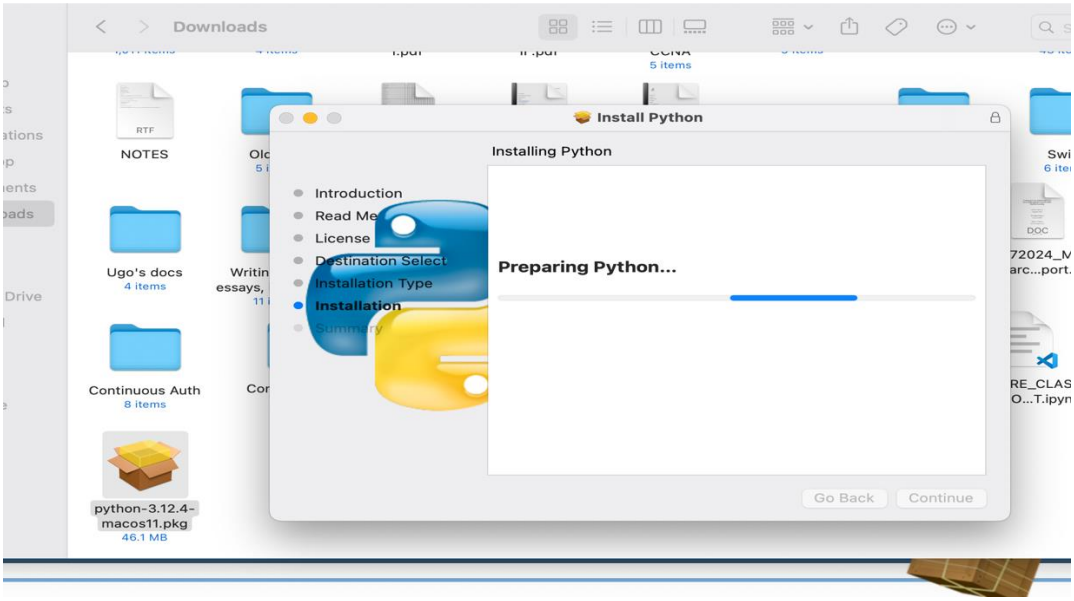


Figure 2. Python Installation

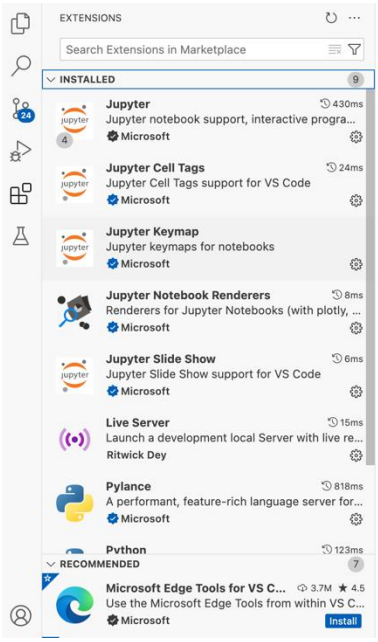


Figure 3. Visual Studio Code installed on Mac OS

4 Implementation

The key libraries from python used in implementing this project:

- Sckit-Learn
- Math
- Pickle
- Pandas
- Numpy
- Matplotlib
- Json

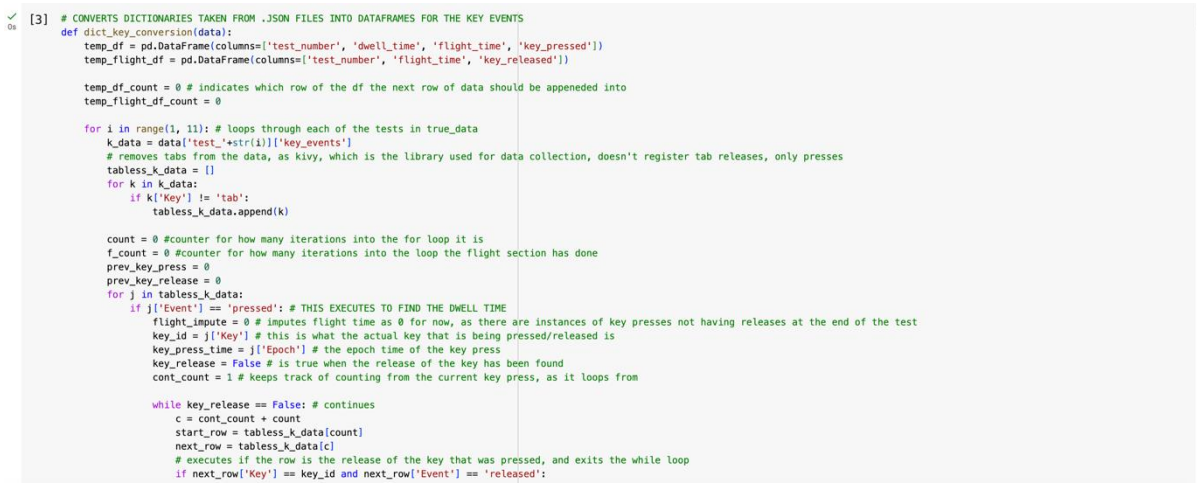
Figure 4. Mounting dataset to google colab

```
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn import metrics
import matplotlib.pyplot as plt
import numpy as np
from google.colab import drive
import json
import pandas as pd
import math
```

Figure 5. Importing libraries

4.1 Data pre-processing

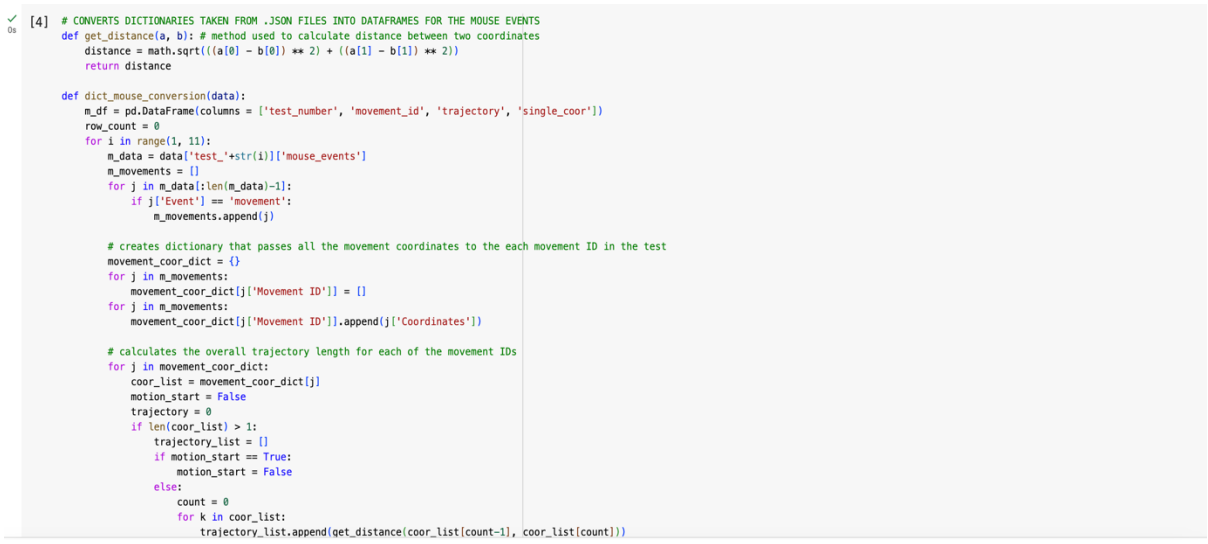
In this chapter, data preparation steps are taken before passing it to the model training and testing. This includes gathering the key and mouse events and computing for the dwell time, flight time, and mouse travel trajectory. The code for computing the key events, mouse events, and the average dwell time, flight time, and mouse trajectory were obtained from the Jupyter notebook file (kmt_feature_classification.ipynb) submitted in the 'Behaviour Biometrics Dataset' on the Mendeley Data platform (Nnamoko et al. 2022)



```
[3]  # CONVERTS DICTIONARIES TAKEN FROM .JSON FILES INTO DATAFRAMES FOR THE KEY EVENTS
     def dict_key_conversion(data):
         temp_df = pd.DataFrame(columns=['test_number', 'dwell_time', 'flight_time', 'key_pressed'])
         temp_flight_df = pd.DataFrame(columns=['test_number', 'flight_time', 'key_released'])

         temp_df_count = 0 # indicates which row of the df the next row of data should be appeneded into
         temp_flight_df_count = 0

         for i in range(1, 11): # loops through each of the tests in true_data
             k_data = data['test_'+str(i)]['key_events']
             # removes tabs from the data, as kivy, which is the library used for data collection, doesn't register tab releases, only presses
             tabless_k_data = []
             for k in k_data:
                 if k['Key'] != 'tab':
                     tabless_k_data.append(k)

             count = 0 #counter for how many iterations into the for loop it is
             f_count = 0 #counter for how many iterations into the loop the flight section has done
             prev_key_press = 0
             prev_key_release = 0
             for j in tabless_k_data:
                 if j['Event'] == 'pressed': # THIS EXECUTES TO FIND THE DWELL TIME
                     flight_impute = 0 # imputes flight time as 0 for now, as there are instances of key presses not having releases at the end of the test
                     key_id = j['Key'] # this is what the actual key that is being pressed/released is
                     key_press_time = j['Epoch'] # the epoch time of the key press
                     key_release = False # is true when the release of the key has been found
                     cont_count = 1 # keeps track of counting from the current key press, as it loops from

                     while key_release == False: # continues
                         c = cont_count + count
                         start_row = tabless_k_data[count]
                         next_row = tabless_k_data[c]
                         # executes if the row is the release of the key that was pressed, and exits the while loop
                         if next_row['Key'] == key_id and next_row['Event'] == 'released':
```

Figure 6. Keystrokes Data Collection

The keystroke dynamics data includes key press and release events. Key metrics such as dwell time (the duration a key is pressed) and flight time (the time between releasing one key and pressing another) are extracted. The mouse movement data includes events such as movements, clicks, and coordinates. The distance traveled during mouse movements is calculated to extract trajectory features. Furthermore, features for each test are generated by calculating the average dwell time, flight time, and trajectory length.



```
[4]  # CONVERTS DICTIONARIES TAKEN FROM .JSON FILES INTO DATAFRAMES FOR THE MOUSE EVENTS
     def get_distance(a, b): # method used to calculate distance between two coordinates
         distance = math.sqrt(((a[0] - b[0]) ** 2) + ((a[1] - b[1]) ** 2))
         return distance

     def dict_mouse_conversion(data):
         m_df = pd.DataFrame(columns = ['test_number', 'movement_id', 'trajectory', 'single_coor'])
         row_count = 0
         for i in range(1, 11):
             m_data = data['test_'+str(i)]['mouse_events']
             m_movements = []
             for j in m_data[:len(m_data)-1]:
                 if j['Event'] == 'movement':
                     m_movements.append(j)

             # creates dictionary that passes all the movement coordinates to the each movement ID in the test
             movement_coor_dict = {}
             for j in m_movements:
                 movement_coor_dict[j['Movement ID']] = []
             for j in m_movements:
                 movement_coor_dict[j['Movement ID']].append(j['Coordinates'])

             # calculates the overall trajectory length for each of the movement IDs
             for j in movement_coor_dict:
                 coor_list = movement_coor_dict[j]
                 motion_start = False
                 trajectory = 0
                 if len(coor_list) > 1:
                     trajectory_list = []
                     if motion_start == True:
                         motion_start = False
                     else:
                         count = 0
                         for k in coor_list:
                             trajectory_list.append(get_distance(coor_list[count-1], coor_list[count]))
```

Figure 7. Mouse Scroll Collection

```python
# GENERATES FEATURES FOR EACH TEST FROM THE DFS GENERATED IN THE PREVIOUS TWO CELLS
def feature_gen(k_data, m_data):
    columns = ['dwell_avg', 'flight_avg', 'traj_avg']

    df = pd.DataFrame(columns=columns)

    # for loop calculates average value for the dwell time, flight time and trajectory for each test
    for i in range(1, 11):
        dwell_list = []
        flight_list = []
        traj_list = []
        for j in k_data.index:
            if k_data.at[j, 'test_number'] == i:
                dwell_list.append(k_data.at[j, 'dwell_time'])
                flight_list.append(k_data.at[j, 'flight_time'])
        for j in m_data.index:
            if m_data.at[j, 'test_number'] == i:
                traj_list.append(m_data.at[j, 'trajectory'])

        dwell_list = [j for j in dwell_list if j != 0]
        flight_list = [j for j in flight_list if j != 0]
        traj_list = [j for j in traj_list if j != 0]

        dwell_avg = sum(dwell_list)/len(dwell_list)
        flight_avg = sum(flight_list)/len(dwell_list)
        traj_avg = sum(traj_list)/len(traj_list)


        agg_data = [dwell_avg, flight_avg, traj_avg]

        df.loc[i] = agg_data
    return df
```

Figure 8. Feature Generation of average dwell time, flight time, and mouse trajectory

5 Machine Model Training and Evaluation

Four machine learning models, Decision Tree Classifier and Random Forest Classifier, Support Vector Machine and Gradient Boosting Classifier are trained and evaluated on the dataset. The models' performance is assessed using accuracy and F-measure metrics.

5.1 Decision Tree Classifier

```python
        clf = tree.DecisionTreeClassifier()
        clf.fit(X_train, y_train)
        predicted_labels = clf.predict(X_test)
        clf_rep = classification_report(y_test, predicted_labels, output_dict=True)
        acc = clf_rep['accuracy']
        fm = clf_rep['weighted avg']['f1-score']
        print('User', i)
        print('Target Labels', y_test)
        print('Predicted Labels', predicted_labels)
        print('Accuracy:', acc)
        print('Fm:', fm)
        print('------------------------------')

        acc_list.append(acc)
        fm_list.append(fm)



    final_acc = sum(acc_list)/len(acc_list)
    final_fm = sum(fm_list)/len(fm_list)


    with open('dtc_model.pkl', 'wb') as file:
        pickle.dump(clf, file)


    print(' ')
    print('############################')
    print('##########RESULTS##########')
    print('############################')
    print('Accuracy:', final_acc)
    print('F-Measure:', final_fm)
```

Figure 9. Decision Tree Model Training

5.2 Random Forest Classifier

```
        rfc = RandomForestClassifier(max_depth = None, min_samples_leaf=8, min_samples_split=3, ccp_alpha=0.0, n_estimators = 150, random_state = 1)
        rfc.fit(X_train, y_train)
        predicted_labels = rfc.predict(X_test)
        clf_rep = classification_report(y_test, predicted_labels, output_dict=True)
        acc = clf_rep['accuracy']
        fm = clf_rep['weighted avg']['f1-score']
        print('User', i)
        print('Target Labels', y_test)
        print('Predicted Labels', predicted_labels)
        print('Accuracy:', acc)
        print('Fm:', fm)
        print('----------------------------')

        acc_list2.append(acc)
        fm_list2.append(fm)

    with open('rfc_model.pkl', 'wb') as file:
        pickle.dump(rfc, file)


    final_acc = sum(acc_list)/len(acc_list)
    final_fm = sum(fm_list)/len(fm_list)


    print(' ')
    print('##########################')
    print('##########RESULTS##########')
    print('##########################')
    print('Accuracy:', final_acc)
    print('F-Measure:', final_fm)
```

Figure 10. Random Forest Model Training

5.3 Support Vector Machine

```
        svm = SVC(kernel='linear', C= 1, gamma = 1)
        svm.fit(X_train, y_train)
        predicted_labels = svm.predict(X_test)
        clf_rep = classification_report(y_test, predicted_labels, output_dict=True)
        acc = clf_rep['accuracy']
        fm = clf_rep['weighted avg']['f1-score']
        print('User', i)
        print('Target Labels', y_test)
        print('Predicted Labels', predicted_labels)
        print('Accuracy:', acc)
        print('Fm:', fm)
        print('----------------------------')

        acc_list3.append(acc)
        fm_list3.append(fm)


    with open('svm4_model.pkl', 'wb') as file:
        pickle.dump(svm, file)

    final_acc = sum(acc_list)/len(acc_list)
    final_fm = sum(fm_list)/len(fm_list)


    print(' ')
    print('##########################')
    print('##########RESULTS##########')
    print('##########################')
    print('Accuracy:', final_acc)
    print('F-Measure:', final_fm)
```

Figure 11. SVM Model Training

5.4 Gradient Boosting Classifier

```
[ ]     #----------------------------------
        y = final_df['label'].tolist() # carries out the train test split and the ML prediction
        X = final_df.drop(['label'], axis=1)
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)


        gbc = GradientBoostingClassifier()
        gbc.fit(X_train, y_train)
        predicted_labels = gbc.predict(X_test)
        clf_rep = classification_report(y_test, predicted_labels, output_dict=True)
        acc = clf_rep['accuracy']
        fm = clf_rep['weighted avg']['f1-score']
        print('User', i)
        print('Target Labels', y_test)
        print('Predicted Labels', predicted_labels)
        print('Accuracy:', acc)
        print('Fm:', fm)
        print('-----------------------------')

        acc_list4.append(acc)
        fm_list4.append(fm)


    with open('gbc_model.pkl', 'wb') as file:
        pickle.dump(gbc, file)

    final_acc = sum(acc_list)/len(acc_list)
    final_fm = sum(fm_list)/len(fm_list)


    print(' ')
    print('##########################')
    print('##########RESULTS##########')
    print('##########################')
```

Figure 12. GBC Model Training

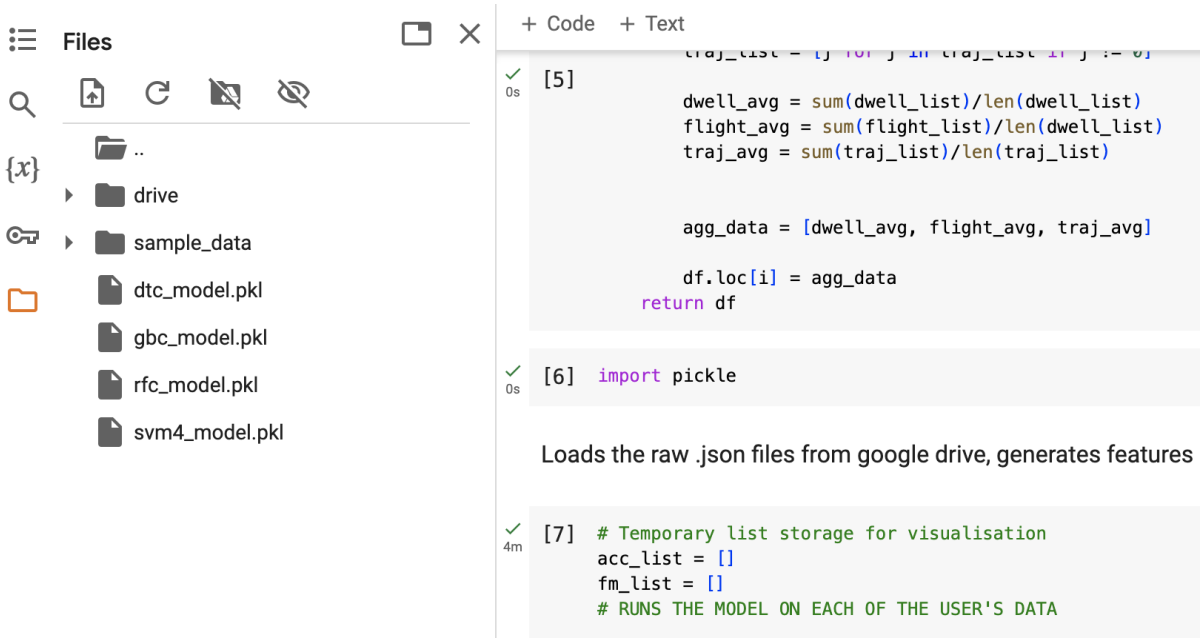


Figure 13. Trained models generated and extracted from the Colab space in the form of a Python Pickle file (PKL) using the pickle module

5.5 Results and Evaluation

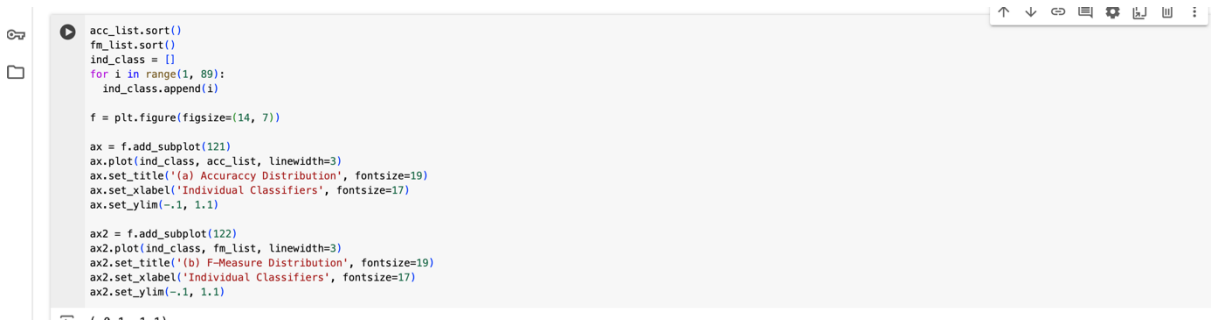
				Metrics
MODELS	Hyperparameters	Minimum and Maximum Accuracy (%)	Minimum and Maximum	Mean Accuracy (%)

			Weighed F-Score (%)	
Decision Tree Classifier (DTC)	Default setting Random state = 0	Min = 25 Max = 100	Min = 20 Max = 100	76.98
Random Forest Classifier (RFC)	n_estimators = 150 Random state = 1	Min = Max = 50	Min = Max = 33	76.98
Support Vector Machine (SVM)	kernel='linear', C= 1, gamma = 1	Min = 25 Max = 100	Min = 20 Max = 100	76.98
Gradient Boosting Classifier (GBC)	learning_rate=0.1, n_estimators=100	Min = 25 Max = 100	Min = 20 Max = 100	76.98

Table 5.1 Describing and Comparing the Selected Machine Learning Models And their Evaluation Metrics

5.6 Visualization/Plotting

Visualization is performed after the models have been trained. The code below aims to visualize the distribution of accuracy and F-measure values for a set of classifiers. Meanwhile, the plot will show how these metrics vary across different classifiers, numbered from 1 to 88. This is done by plotting the list containing accuracy values for different classifiers (acc_list) and the list containing F-measure values for the classifiers (fm_list) against the ind_class values (x-axis).



```
acc_list.sort()
fm_list.sort()
ind_class = []
for i in range(1, 89):
  ind_class.append(i)

f = plt.figure(figsize=(14, 7))

ax = f.add_subplot(121)
ax.plot(ind_class, acc_list, linewidth=3)
ax.set_title('(a) Accuracy Distribution', fontsize=19)
ax.set_xlabel('Individual Classifiers', fontsize=17)
ax.set_ylim(-.1, 1.1)

ax2 = f.add_subplot(122)
ax2.plot(ind_class, fm_list, linewidth=3)
ax2.set_title('(b) F-Measure Distribution', fontsize=19)
ax2.set_xlabel('Individual Classifiers', fontsize=17)
ax2.set_ylim(-.1, 1.1)
```

Figure 14. Code aiming to visualize the distribution of accuracy and F-measure values

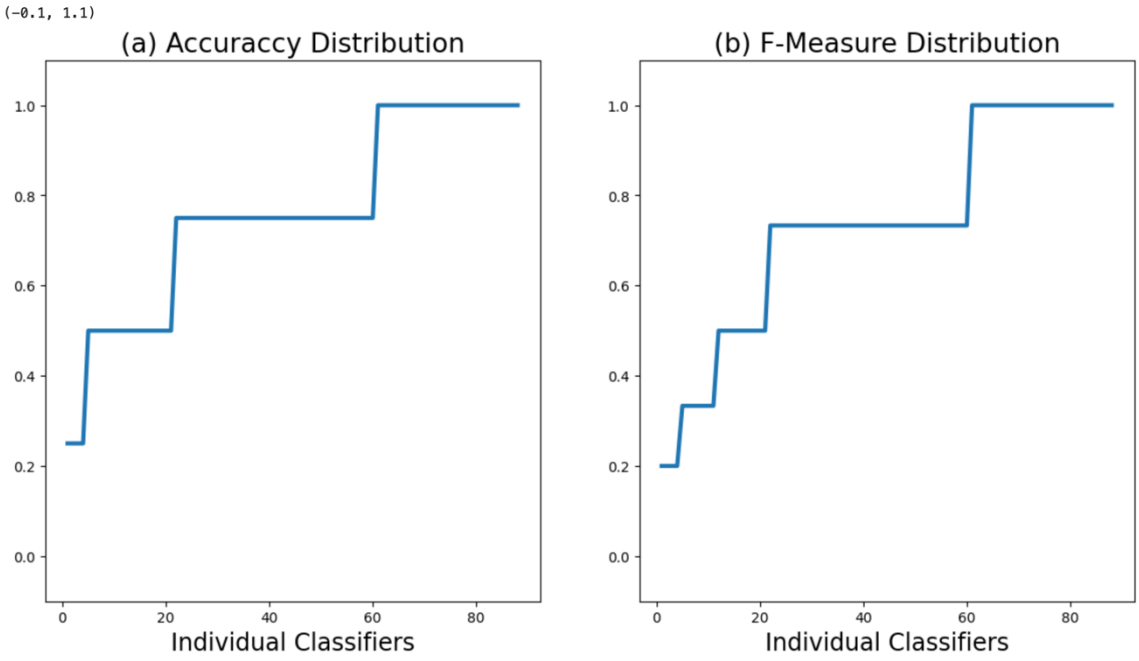


Figure 15. F-Measure values and Accuracy distribution among the classifiers

6 Building the Continuous Authentication Web-Based Messaging Platform

6.1 System Overview

For this project, continuous authentication is integrated into a web-based messaging platform. The primary goal is to utilize keystroke dynamics and mouse scroll patterns to validate the user's identity during their session. The main components of the system include:

Frontend Section:

1. HTML file for user registration and messaging
2. CSS pages for styling the user registration and messaging
3. JavaScript for capturing keystroke and scroll events.

Backend Section:

1. Flask application for handling user management and authentication
2. Machine learning model for predicting user identity.

```
<html lang="en">
<head>
    <script>
        function attachEventListeners() {
            document.getElementById('registerBtn').addEventListener('click', function (event) {
                event.preventDefault();
                sendRegistrationData();
            });
        }

        window.onload = attachEventListeners;
    </script>
</head>
<body>
    <form id="registrationForm">
        <label for="fname">Full Name:</label>
        <input type="text" id="fname" name="fullname" required><br><br>

        <label for="addr">Address:</label>
        <input type="text" id="addr" name="address" required><br><br>

        <label for="pnumber">Phone Number:</label>
        <input type="text" id="pnumber" name="pnumber" required><br><br>

        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required><br><br>

        <label for="edu">Education:</label>
        <input type="text" id="edu" name="education" required><br><br>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required><br><br>

        <button id="registerBtn">Register</button>
    </form>
```

Figure 16. User Registration HTML template

Fig 6.1 is a dummy page for user registration with an in-line JavaScript script with the functionality to collect keystrokes and mouse scroll data as the user is inputting their details to use it for future authentication. Meanwhile, Fig 6.2 below is a code snippet; a messaging dummy page with the functionality to extract user keystrokes and mouse scroll data to be used for continuous authentication.

```
        function sendMessage() {
            const message = document.getElementById('message').value;

            const xhr = new XMLHttpRequest();
            xhr.open('POST', '/send_message', true);
            xhr.setRequestHeader('Content-Type', 'application/json');
            xhr.onreadystatechange = function () {
                if (xhr.readyState === 4 && xhr.status === 200) {
                    const messagesDiv = document.getElementById('messages');
                    messagesDiv.innerHTML += `<p>${message}</p>`;
                    document.getElementById('message').value = '';
                }
            };

            const data = JSON.stringify({ 'message': message });
            xhr.send(data);
        }

        window.onload = attachEventListeners;
    </script>
</head>
<body>
    <div id="messagingContainer">
        <h1>Welcome to the Messaging Platform</h1>
        <form id="messageForm">
            <label for="message">Message:</label>
            <textarea id="message" name="message"></textarea>
            <button type="button" onclick="sendMessage()">Send</button>
        </form>
        <div id="messages"></div>
    </div>
</body>
</html>
```

Figure 17. Messaging HTML template

```css
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    display: flex;
    justify-content: center;
    padding-top: 50px;
    margin: 0;
}

#messagingContainer {
    width: 80%;
    max-width: 800px;
    background-color: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h1 {
    text-align: center;
    color: #333;
}

#messageForm {
    display: flex;
    flex-direction: column;
    align-items: center;
}

#messageForm label {
    width: 100%;
    margin-bottom: 8px;
    font-weight: bold;
}
```

Figure 18. CSS file for the messaging page

```css
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 150vh;
    margin: 0;
}

#loginForm {
    background-color: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    width: 300px;
}

#loginForm label {
    display: block;
    margin-bottom: 8px;
    font-weight: bold;
}

#loginForm input {
    width: 100%;
    padding: 8px;
    margin-bottom: 16px;
    border: 1px solid #ccc;
    border-radius: 4px;
}
```

Figure 19. CSS file for the registration page

6.2 JavaScript Event Logging

Fig 6.5 below displays the JavaScript event logging script. This script captures the following user interactions:

- Keystroke Events: Logs ‘keydown’ and ‘keyup’ events, records the key and timestamp.
- Scroll Events: Logs scroll positions and timestamps.

```
<script>
    let keyEvents = [];
    let scrollEvents = [];

    function logKey(event) {
        keyEvents.push({
            'Event': event.type,
            'Key': event.key,
            'Epoch': event.timeStamp / 1000 // Convert to seconds
        });
    }

    function logScroll(event) {
        scrollEvents.push({
            'Event': 'scroll',
            'ScrollX': window.scrollX,
            'ScrollY': window.scrollY,
            'Epoch': event.timeStamp / 1000 // Convert to seconds
        });
    }

    function authCheck() {
        const xhr = new XMLHttpRequest();
        xhr.open('POST', '/auth_check', true);
        xhr.setRequestHeader('Content-Type', 'application/json');
        xhr.onreadystatechange = function () {
            if (xhr.readyState === 4) {
                if (xhr.status === 200) {
                    console.log('Authentication successful!');
                } else {
                    console.log('Session expired or authentication failed:', xhr.responseText);
                    alert('Session expired due to keystroke and scroll mismatch!');

                    window.location.href = '/'; // Redirect to login page
                }
```

Figure 20. Keystroke and Mouse scroll Collection

6.3 Flask Application

The Flask application handles the main logic. Following the importation of the Flask object from the flask package, the name app is used to create an instance of your Flask application. The system includes user registration endpoint, authentication check endpoint, and the messaging platform endpoint. Lastly, the Flask application is initialized with a secret key for session management.

```python
from flask import Flask, request, jsonify, session, redirect, url_for, render_template
import joblib
import numpy as np
import os
import json

app = Flask(__name__)
app.secret_key = "nmtpllseet34gssce4t"

# Define the file path for the model
model_path = os.path.join(os.path.dirname(__file__), 'server/svm4_model (1).pkl')

# Load the trained model
if os.path.exists(model_path):
    model = joblib.load(model_path)
else:
    raise FileNotFoundError(f"Model file not found: {model_path}")

users_db = {}


def calculate_metrics(key_events, scroll_events):
    dwell_times = []
    flight_times = []
    scroll_deltas = []

    for i, event in enumerate(key_events):
        if event['Event'] == 'pressed':
            key = event['Key']
            press_time = float(event['Epoch'])
            release_time = next((float(e['Epoch']) for e in key_events[i+1:] if e['Key'] == key and e['Event'] ==
            dwell_time = release_time - press_time
            dwell_times.append(dwell_time)

            if i > 0:
                previous_event = key_events[i-1]
                if previous_event['Event'] == 'pressed':
                    flight_time = press_time - float(previous_event['Epoch'])
                    flight_times.append(flight_time)
```

Figure 21. A display of the Flask Web Server

Continuous Auth 2 > app.py > ...

```python
from flask import Flask, request, jsonify, session, redirect, url_for, render_template
import joblib
import numpy as np
import os
import json

app = Flask(__name__)
app.secret_key = "nmtpllseet34gssce4t"

```

Figure 22. Flask Application Setup with a secret key for session management

```python
# Define the file path for the model
model_path = os.path.join(os.path.dirname(__file__), 'server/svm4_model (1).pkl')

# Load the trained model
if os.path.exists(model_path):
    model = joblib.load(model_path)
else:
    raise FileNotFoundError(f"Model file not found: {model_path}")

users_db = {}

```

Figure 23. Model Loading and Database

```python
def calculate_metrics(key_events, scroll_events):
    dwell_times = []
    flight_times = []
    scroll_deltas = []

    for i, event in enumerate(key_events):
        if event['Event'] == 'pressed':
            key = event['Key']
            press_time = float(event['Epoch'])
            release_time = next((float(e['Epoch']) for e in key_events[i+1:] if e['Key'] == key and e['Event'] ==
            dwell_time = release_time - press_time
            dwell_times.append(dwell_time)

            if i > 0:
                previous_event = key_events[i-1]
                if previous_event['Event'] == 'pressed':
                    flight_time = press_time - float(previous_event['Epoch'])
                    flight_times.append(flight_time)

    for i in range(1, len(scroll_events)):
        delta_x = scroll_events[i]['ScrollX'] - scroll_events[i-1]['ScrollX']
        delta_y = scroll_events[i]['ScrollY'] - scroll_events[i-1]['ScrollY']
        scroll_deltas.append((delta_x, delta_y))

    average_trajectory = sum(dwell_times) / len(dwell_times) if dwell_times else 0
    average_scroll_delta = np.mean(scroll_deltas, axis=0) if scroll_deltas else (0, 0)

    return {
        'dwell_times': dwell_times,
        'flight_times': flight_times,
        'average_trajectory': average_trajectory,
        'average_scroll_delta': average_scroll_delta
    }
```

Figure 24. Keystrokes and Mouse Scroll Feature Metrics Computation

```python
def save_user_data(username, key_events):
    user_dir = os.path.join(os.path.dirname(__file__), f'user_data/{username}')
    os.makedirs(user_dir, exist_ok=True)

    key_events_file = os.path.join(user_dir, 'key_events.json')
    #scroll_events_file = os.path.join(user_dir, 'scroll_events.json')

    with open(key_events_file, 'w') as f:
        json.dump(key_events, f, indent=4)

    #with open(scroll_events_file, 'w') as f:
        #json.dump(scroll_events, f, indent=4)


```

Figure 25. Saving of User Data

```python
@app.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')
    key_events = data.get('key_events', [])
    scroll_events = data.get('scroll_events', [])

    # Check if username already exists (prevent duplicate registrations)
    if username in users_db:
        return jsonify({'message': 'Username already exists!'}), 200

    # Calculate keystroke metrics
    metrics = calculate_metrics(key_events, scroll_events)
    feature_vector = metrics['dwell_times'] + metrics['flight_times'] + [metrics['average_trajectory']] + list(me

    users_db[username] = {
        'password': password,
        'keystroke_metrics': feature_vector
    }

    # Save user data
    save_user_data(username, key_events)

    session['username'] = username
    return jsonify({'message': 'Registration successful!'}), 201  # Created status code

```

Figure 26. User Registration Endpoint

```python
@app.route('/auth_check', methods=['POST'])
def auth_check():
    if 'username' not in session:
        return jsonify({'message': 'Not authenticated!'}), 401

    username = session['username']

    data = request.get_json()
    key_events = data.get('key_events', [])
    scroll_events = data.get('scroll_events', [])
    metrics = calculate_metrics(key_events, scroll_events)
    feature_vector = metrics['dwell_times'] + metrics['flight_times'] + [metrics['average_trajectory']] + list(me

    # Save user data
    save_user_data(username, key_events)

    if not key_events:  # No keystrokes, user is idle
        return jsonify({'message': 'User is idle, no authentication required.'}), 200

    registered_metrics = users_db[username]['keystroke_metrics']
    prediction = model.predict([feature_vector])

    if prediction[0] != 1:
        session.pop('username', None)
        return jsonify({'message': 'Session expired due to keystroke and scroll mismatch!'}), 401

    return jsonify({'message': 'Authentication successful!'}), 200

```

Figure 27. Authentication Checkpoint

The authentication endpoint periodically checks user authentication based on keystroke and mouse scroll events. It compares the current user metrics with the registered metrics using the pre-trained model and maintains or terminates the session based on the prediction with a continuous authentication interval of 60 seconds.

Messaging Platform and Message Sending Endpoint

```python
@app.route('/messaging', methods=['GET'])
def messaging():
    if 'username' not in session:
        return redirect(url_for('/login_page'))
    return render_template('chat.html', username=session['username'])


@app.route('/send_message', methods=['POST'])
def send_message():
    data = request.get_json()
    message = data.get('message')
    # Save or process the message as needed
    return jsonify({'message': 'Message sent successfully!'}), 200


```

Figure 28. Messaging Endpoint

The messaging endpoint renders the chat interface if the user is authenticated while the message sending endpoint processes and acknowledges user messages.

```python

@app.route('/', methods=['GET'])
def login_page():
    return render_template('register.html')


if __name__ == '__main__':
    app.run(debug=True)
```

Figure 29. Login Page Endpoint

The root endpoint renders the user registration page, and the application is run in debug mode as seen from running the app.py python script in the terminal.

Figure 30. Running the application python file in VS

Accessing and Testing the Messaging Web Application

The messaging web application is operating locally at http://127.0.0.1:5000/, where :5000 represents the port number and 127.0.0.1 corresponds to the machine's localhost. Therefore, to access the messaging platform, one can browse to http://127.0.0.1:5000/ using any browser of choice.

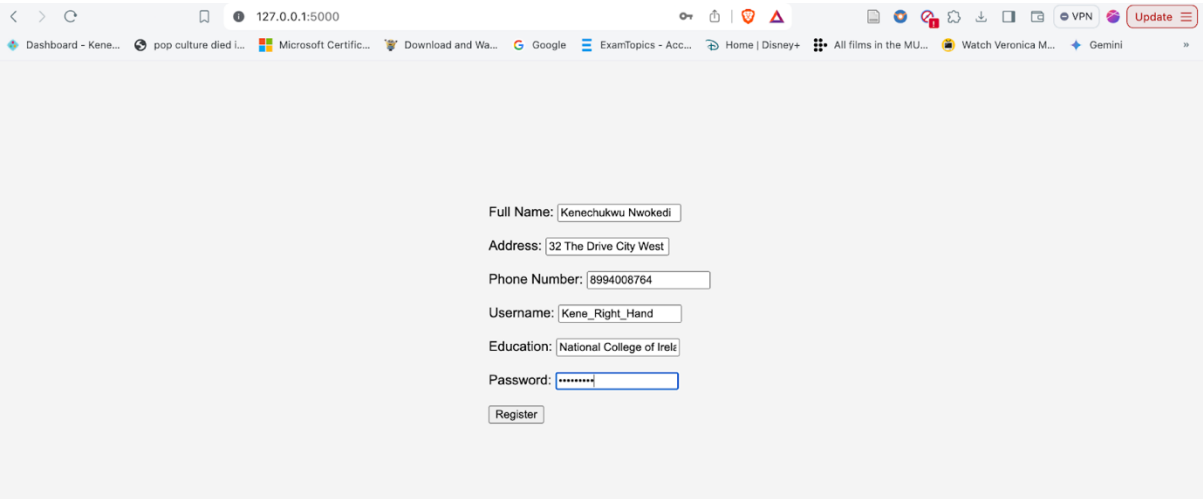


Figure 31. A Display of the Registration Page Where User Inputs Various Details Including Full Name, Address, Phone Number, Education, Username, and Password

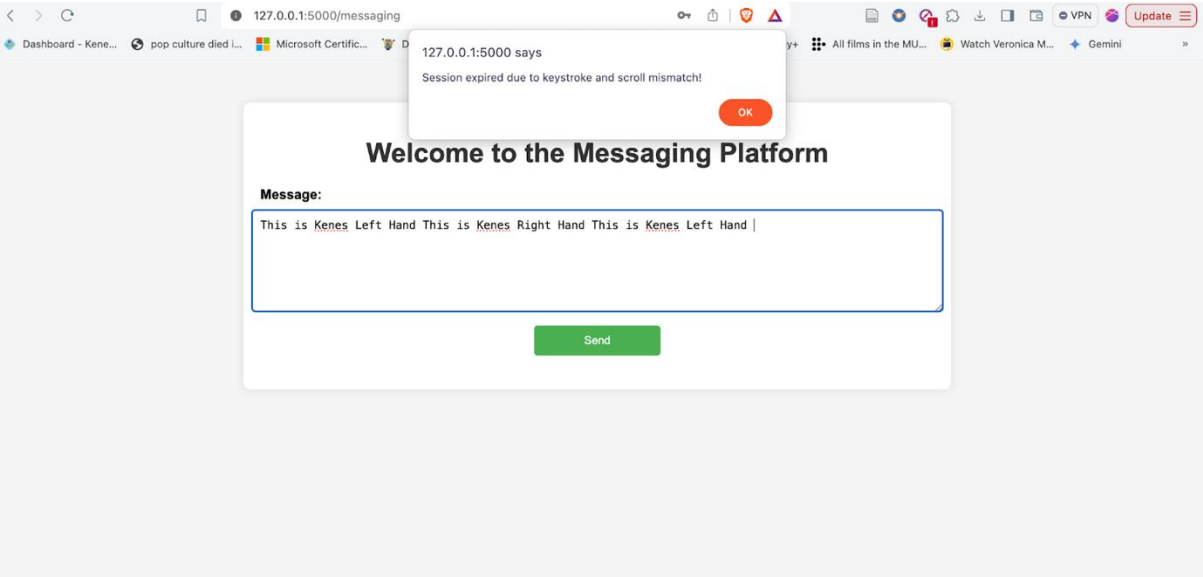


Figure 32. A Display of the Messaging Platform Page Detailing a Scenario where the User has been logged out due to a keystroke and scroll mismatch

7 References

Nnamoko, N., Barrowclough, J., Liptrott, M. and Korkontzelos, I. (2022) *Behaviour Biometrics Dataset.* DOI: 10.17632/fnf8b85kr6.1. Available at: https://data.mendeley.com/datasets/fnf8b85kr6/1 [Accessed 17 July 2024].