PROJECT PRESENTATION

Log-in by password and key stroke dynamic

6488045 SUPITHCHA 6488052 SASASUANG 6488226 NISAKORN

Introduction

In today's digital age, securing online accounts and systems is critical. **Traditional password-based authentication systems are vulnerable to various attacks**. To enhance security, biometric authentication methods, such as keystroke dynamics, have been introduced. **Keystroke Patterns** have been used as a biometric since the early 1900s in the days of WWI [2]. Keystroke is the combination of an initial KeyDown event and the corresponding KeyUp event [2]. It analyzes the **unique typing patterns of individuals to verify their identities**, **adding an extra layer of security** to the authentication process.

CONTENTS

TECHNOLOGY USED

PROGRAMMING

HTML/CSS

NODE.JS

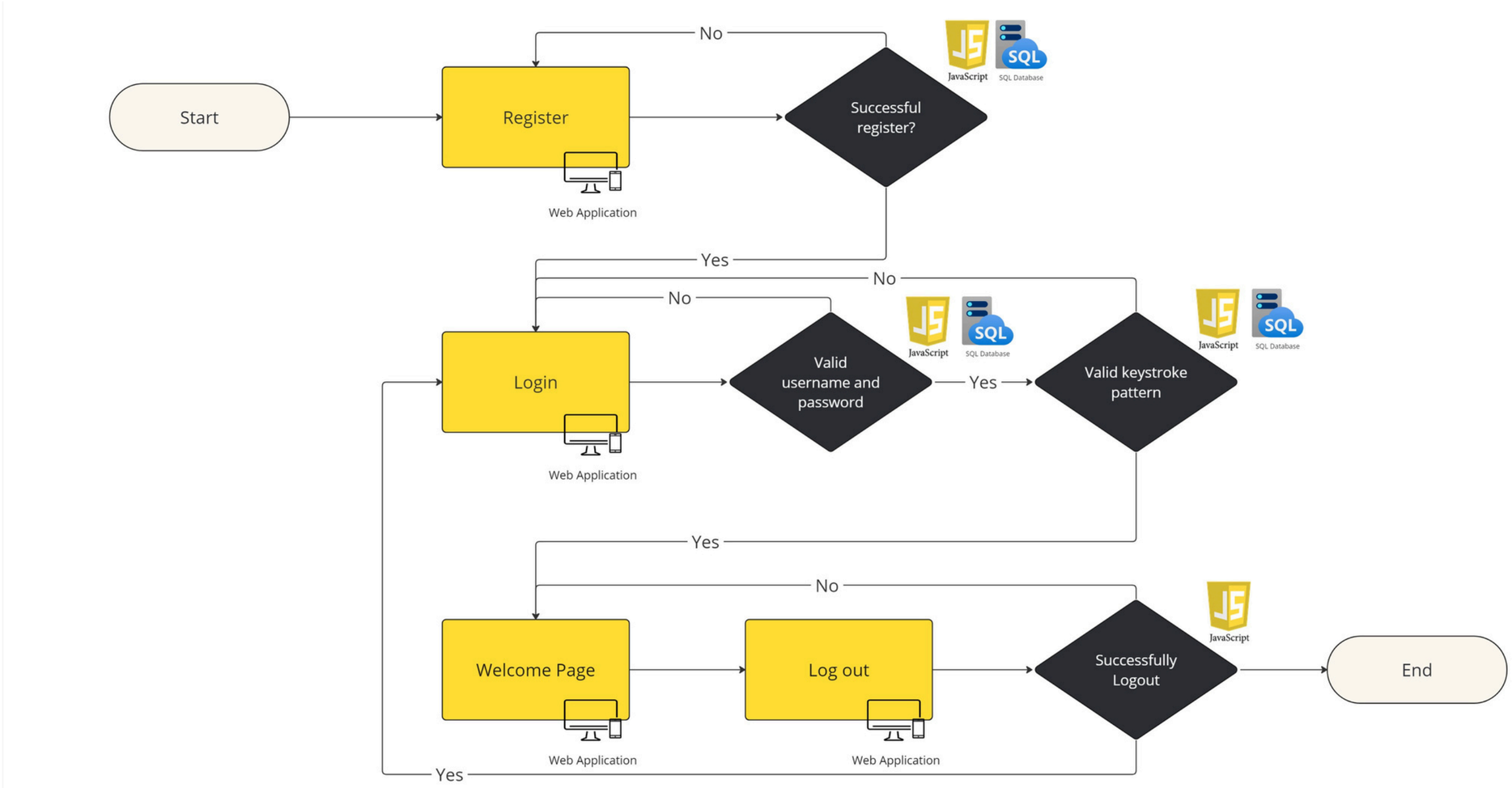
MYSQL DATABASE

JAVASCRIPE

COMPUTER

- Visual Studio Code
- Github

DETAILS OF DESIGN



REGISTER

Comment Code | | Click here to ask Blackbox to help you code faster

```
function addUser(){
    let username = document.getElementById("username").value;
    let password = document.getElementById("password").value;
    if (password.length < 12) {
        alert("Password must be at least 12 characters long");
        return false;
    }

    let data = {
        "username": username,
        "password": password,
        "keystrokes": keystrokes
    };

    // Calculate average CPM, UD, and DU for user input
    const totalKeystrokes = keystrokes.length;
    const totalUpDownTime = calculateTotalUpDownTime(keystrokes);
    const totalDownUpTime = calculateTotalDownUpTime(keystrokes);
    const avg_CPM_user = calculateCPM(totalKeystrokes, totalUpDownTime);
    const avg_UD_user = totalUpDownTime / totalKeystrokes;
    const avg_DU_user = totalDownUpTime / totalKeystrokes;

    // Include average values for user input in the data object
    data.avg_CPM_user = avg_CPM_user;
    data.avg_UD_user = avg_UD_user;
    data.avg_DU_user = avg_DU_user;

    // Calculate average CPM, UD, and DU for password input
    if (password) {
        const passwordKeystrokes = password.split('').map((key, index) => ({
            key,
            timestamp: Date.now() + index * 10 // Use a simple timestamp for demonstration
        }));
        const totalPasswordKeystrokes = passwordKeystrokes.length;
        const totalUpDownTimePassword = calculateTotalUpDownTime(passwordKeystrokes);
```


```
// Calculate average CPM, UD, and DU for user input
const totalKeystrokes = keystrokes.length;
const totalUpDownTime = calculateTotalUpDownTime(keystrokes);
const totalDownUpTime = calculateTotalDownUpTime(keystrokes);
const avg_CPM_user = calculateCPM(totalKeystrokes, totalUpDownTime);
const avg_UD_user = totalUpDownTime / totalKeystrokes;
const avg_DU_user = totalDownUpTime / totalKeystrokes;
```

```
if (password) {
    const passwordKeystrokes = password.split('').map((key, index) => ({
        key,
        timestamp: Date.now() + index * 10 // Use a simple timestamp for demonstration
    }));
    const totalPasswordKeystrokes = passwordKeystrokes.length;
    const totalUpDownTimePassword = calculateTotalUpDownTime(passwordKeystrokes);
    const totalDownUpTimePassword = calculateTotalDownUpTime(passwordKeystrokes);
    const avg_CPM_pass = calculateCPM(totalPasswordKeystrokes, totalUpDownTimePassword);
    const avg_UD_pass = totalUpDownTimePassword / totalPasswordKeystrokes;
    const avg_DU_pass = totalDownUpTimePassword / totalPasswordKeystrokes;

    // Include average values for password input in the data object
    data.avg_CPM_pass = avg_CPM_pass;
    data.avg_UD_pass = avg_UD_pass;
    data.avg_DU_pass = avg_DU_pass;
}
```

REGISTER

```
document.addEventListener('DOMContentLoaded', function() {
    let startTime = null;
    let totalKeystrokes = 0;
    let totalUpDownTime = 0;
    let totalDownUpTime = 0;

    document.addEventListener('keydown', function(event) {
        if (document.getElementById("password").value.length < 12) {
            return; // Don't capture keystrokes if password length is less than 12
        }
        const key = event.key;
        const timestamp = new Date().getTime();
        keystrokes.push({ key, timestamp });
        currentInput += key; // Update the current input value

        if (!startTime) {
            startTime = timestamp;
        } else {
            totalKeystrokes++;
            const elapsedTime = timestamp - startTime;
            totalUpDownTime += elapsedTime;
            startTime = timestamp;
        }
    });

    document.addEventListener('keyup', function(event) {
        const timestamp = new Date().getTime();
        const elapsedTime = timestamp - startTime;
        totalDownUpTime += elapsedTime;
        startTime = timestamp;
    });
```

CALCULATE CHARACTERS PER MINUTE

```
Comment Code
function calculateCPM(totalKeystrokes, totalUpDownTime) {
    const totalTimeMinutes = totalUpDownTime / (1000 * 60); // Convert milliseconds to minutes
    return totalKeystrokes / totalTimeMinutes;
}
```

TOTAL UP DOWN TIME (UD)

```
Comment Code
function calculateTotalUpDownTime(keystrokes) {
    let totalUpDownTime = 0;
    for (let i = 1; i < keystrokes.length; i++) {
        totalUpDownTime += keystrokes[i].timestamp - keystrokes[i - 1].timestamp;
    }
    return totalUpDownTime;
}
```

DOWN UP TIME (DU)

```
Comment Code
function calculateTotalDownUpTime(keystrokes) {
    let totalDownUpTime = 0;
    for (let i = 0; i < keystrokes.length - 1; i++) {
        totalDownUpTime += keystrokes[i + 1].timestamp - keystrokes[i].timestamp;
    }
    return totalDownUpTime;
}
```

AUTHENTICATION

LOGIN

```
// Calculate average CPM, UD, and DU for user input
const totalKeystrokesUser = keystrokes.filter(k => k.key !== 'Enter').length;
const totalUpDownTimeUser = calculateTotalUpDownTime(keystrokes);
const totalDownUpTimeUser = calculateTotalDownUpTime(keystrokes);
const avg_CPM_user = calculateCPM(totalKeystrokesUser, totalUpDownTimeUser);
const avg_UD_user = totalUpDownTimeUser / totalKeystrokesUser;
const avg_DU_user = totalDownUpTimeUser / totalKeystrokesUser;

// Calculate average CPM, UD, and DU for password input
const passwordKeystrokes = keystrokes.filter(k => k.key !== 'Enter' && k.key !== 'Tab');
const totalKeystrokesPass = passwordKeystrokes.length;
const totalUpDownTimePass = calculateTotalUpDownTime(passwordKeystrokes);
const totalDownUpTimePass = calculateTotalDownUpTime(passwordKeystrokes);
const avg_CPM_pass = calculateCPM(totalKeystrokesPass, totalUpDownTimePass);
const avg_UD_pass = totalUpDownTimePass / totalKeystrokesPass;
const avg_DU_pass = totalDownUpTimePass / totalKeystrokesPass;
```

```
// Calculate similarity using some metric (e.g., cosine similarity)
let similarity_user = calculateSimilarityUser({ avg_CPM_user, avg_UD_user, avg_DU_user },
    { avg_CPM_user: stored_avg_CPM_user, avg_UD_user: stored_avg_UD_user, avg_DU_user: stored_avg_DU_user });
let similarity_pass = calculateSimilarityPass({ avg_CPM_pass, avg_UD_pass, avg_DU_pass },
    { avg_CPM_pass: stored_avg_CPM_pass, avg_UD_pass: stored_avg_UD_pass, avg_DU_pass: stored_avg_DU_pass });
```

```
let similarityThreshold = 0.4;
let tokens = jwt.sign({ username }, 'secretkey', { expiresIn: '1h' });
if (similarity_user >= similarityThreshold && similarity_pass >= similarityThreshold) {
    // Authentication successful
    res.json({ status: 'success', token: tokens, message: 'Login Success' })

} else {
    // Authentication failed
    res.json({ 'status': 'error', 'message': 'Invalid keystroke pattern' })
}
```

Comment Code

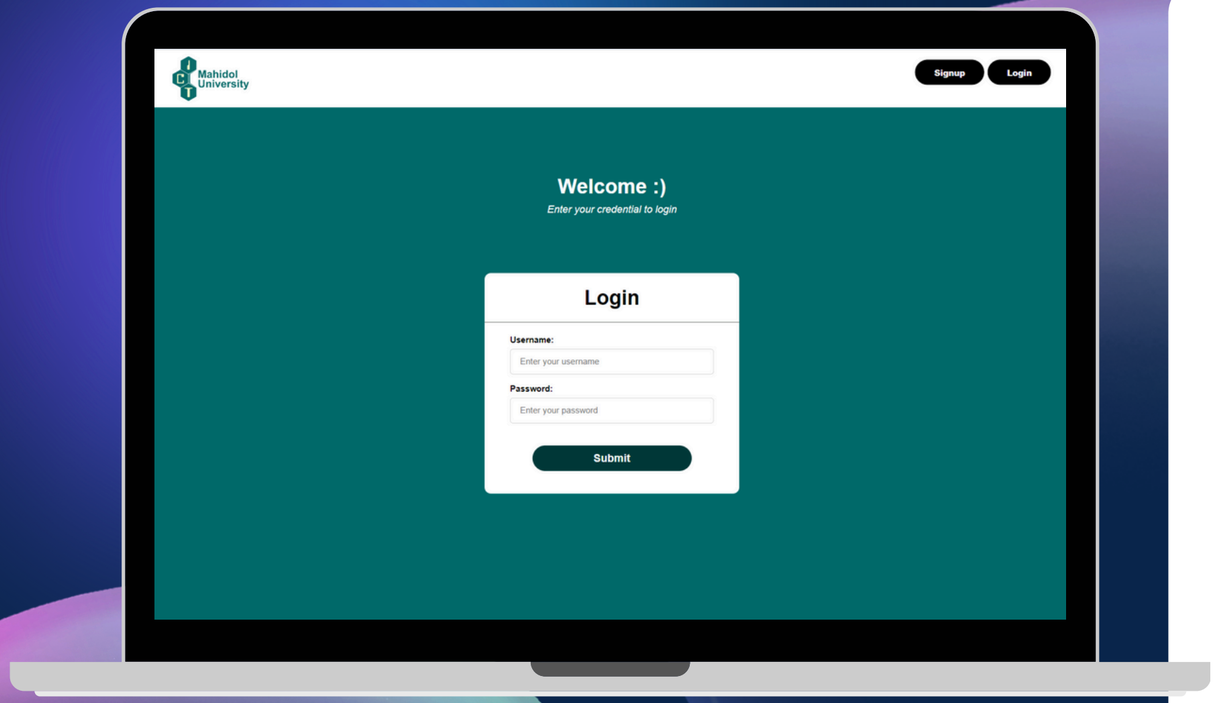
```
router.post('/db_login', async function (req, res) {
    let username = req.body.username;
    let password = req.body.password;
    let keystrokes = req.body.keystrokes;

    // Calculate average CPM, UD, and DU for user input
    const totalKeystrokesUser = keystrokes.filter(k => k.key !== 'Enter').length;
    const totalUpDownTimeUser = calculateTotalUpDownTime(keystrokes);
    const totalDownUpTimeUser = calculateTotalDownUpTime(keystrokes);
    const avg_CPM_user = calculateCPM(totalKeystrokesUser, totalUpDownTimeUser);
    const avg_UD_user = totalUpDownTimeUser / totalKeystrokesUser;
    const avg_DU_user = totalDownUpTimeUser / totalKeystrokesUser;

    // Calculate average CPM, UD, and DU for password input
    const passwordKeystrokes = keystrokes.filter(k => k.key !== 'Enter' && k.key !== 'Tab');
    const totalKeystrokesPass = passwordKeystrokes.length;
    const totalUpDownTimePass = calculateTotalUpDownTime(passwordKeystrokes);
    const totalDownUpTimePass = calculateTotalDownUpTime(passwordKeystrokes);
    const avg_CPM_pass = calculateCPM(totalKeystrokesPass, totalUpDownTimePass);
    const avg_UD_pass = totalUpDownTimePass / totalKeystrokesPass;
    const avg_DU_pass = totalDownUpTimePass / totalKeystrokesPass;

    console.log(keystrokes);
    try {
        dbConn.query('SELECT * FROM user_account WHERE username = ?', [req.body.username],
            async function (error, results, fields) {
                if (error) {
                    res.json({ 'status': 'error', 'message': "error" })
                } else if (results.length == 0) {
                    res.json({ 'status': 'error', 'message': 'Invalid username or password' })
                } else {
                    const similarUser = results[0];
                    let storedPassword = similarUser.password;
```

AUTHENTICATION

CALCULATE SIMILARITY

```
function calculateSimilarity(storedValues, currentValues) {
    if (!storedValues || !currentValues || storedValues.length !== currentValues.length) {
        return 0; // Return 0 if either storedValues or currentValues is null or their lengths don't match
    }

    let dotProduct = 0;
    let storedMagnitude = 0;
    let currentMagnitude = 0;

    // cosine similarity
    for (let i = 0; i < storedValues.length; i++) {
        dotProduct += storedValues[i] * currentValues[i];
        storedMagnitude += Math.pow(storedValues[i], 2);
        currentMagnitude += Math.pow(currentValues[i], 2);
    }

    storedMagnitude = Math.sqrt(storedMagnitude);
    currentMagnitude = Math.sqrt(currentMagnitude);

    // Check for zero magnitude to avoid division by zero
    if (storedMagnitude === 0 || currentMagnitude === 0) {
        return 0; // Return 0 if one of the magnitudes is zero
    } else {
        return dotProduct / (storedMagnitude * currentMagnitude);
    }
}
```

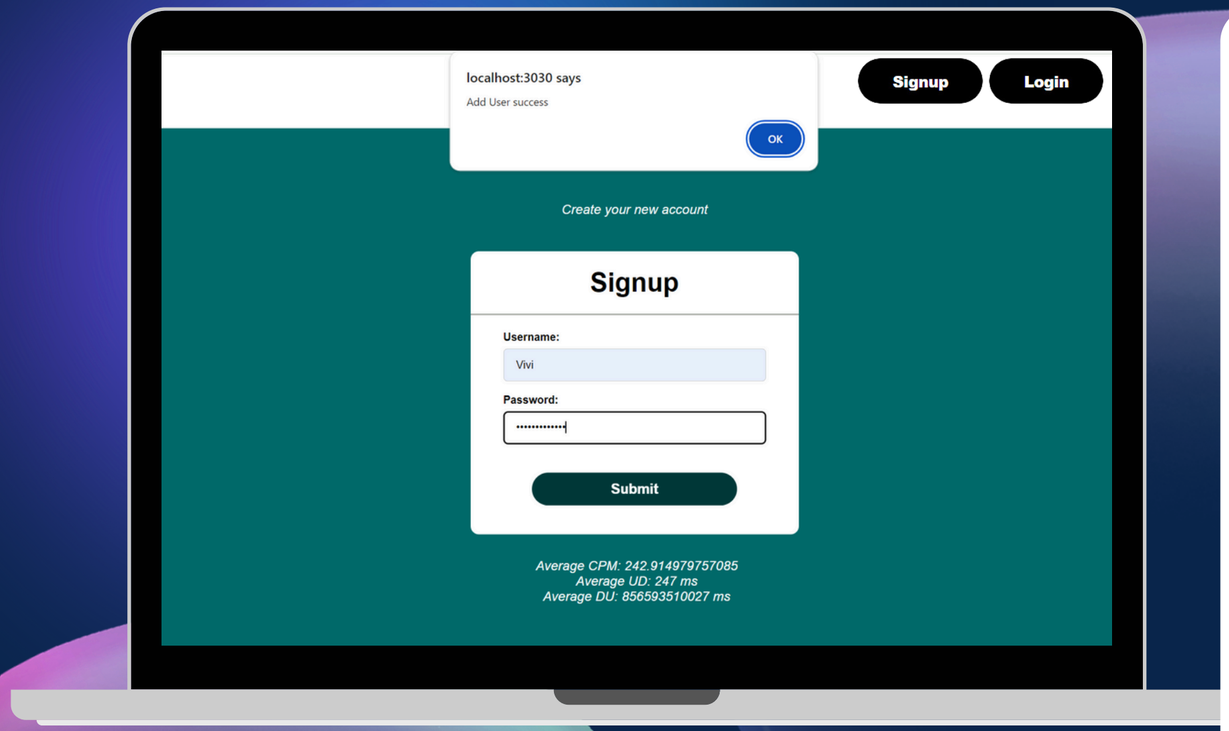
DETAILS OF DESIGN



LOGIN PAGE

After running the program, the user can access the login page first. The new user is required to register before login, so the user can click on the “Sign up” button on the right side header to access to sign-up page.

DETAILS OF DESIGN

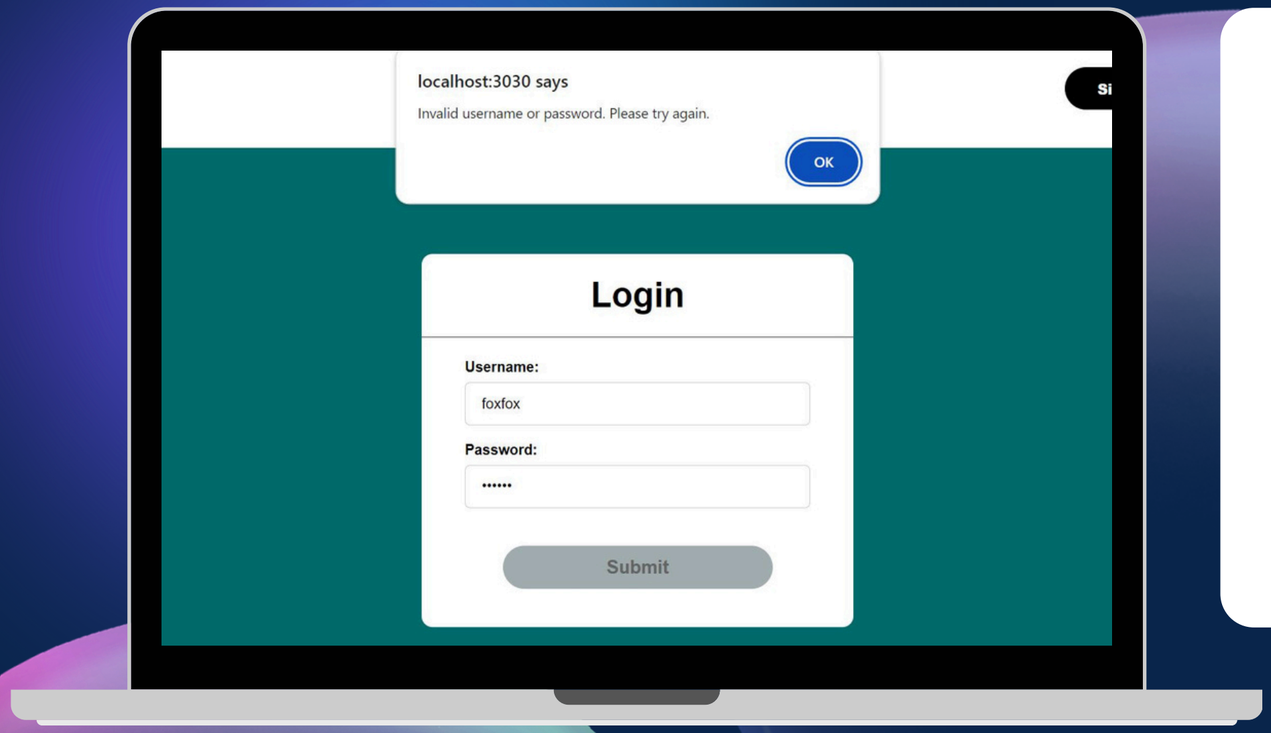


REGISTER PAGE

The user needs to register the account by entering a username and password. After successfully registering, the calculated averages CPM, UD, and DU will display below. Additionally, the user can go back to the login page via the “Login” button

Signup Login

DETAILS OF DESIGN

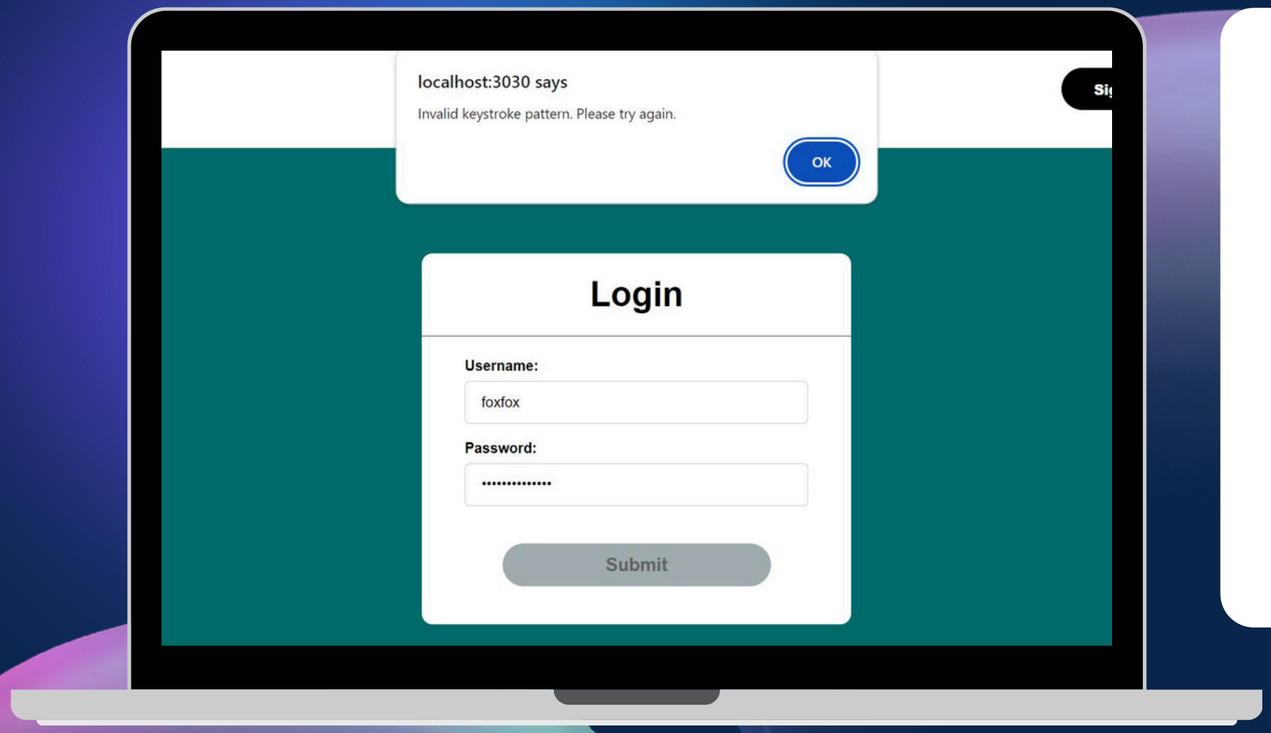


LOGIN PAGE

If the user tries to log in with an invalid account username or password, the pop-up message will notify to try again.

“Invalid username or password, Please try again.”

DETAILS OF DESIGN

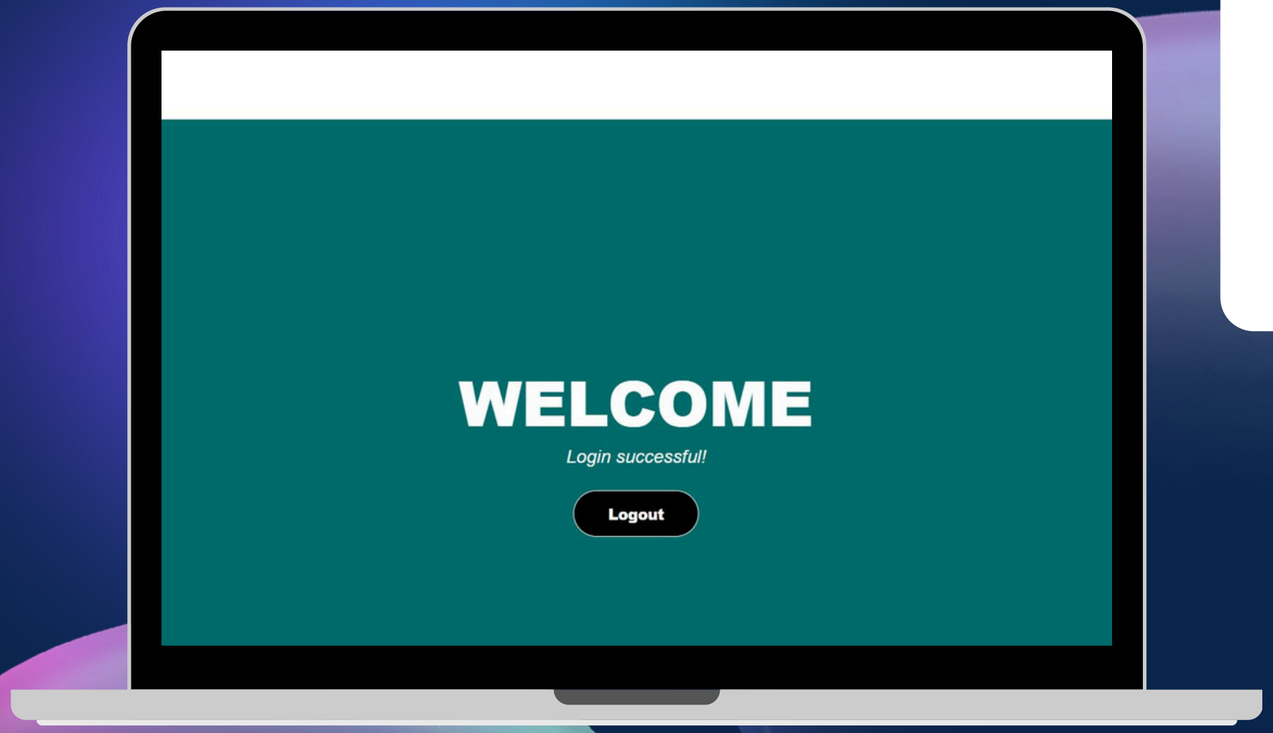


LOGIN PAGE

If the keystroke pattern of the user isn't similar to the averages throughout the sign-up process, the user must try to log in again unit the keystroke is similar.

"Invalid keystroke pattern, Please try again."

DETAILS OF DESIGN

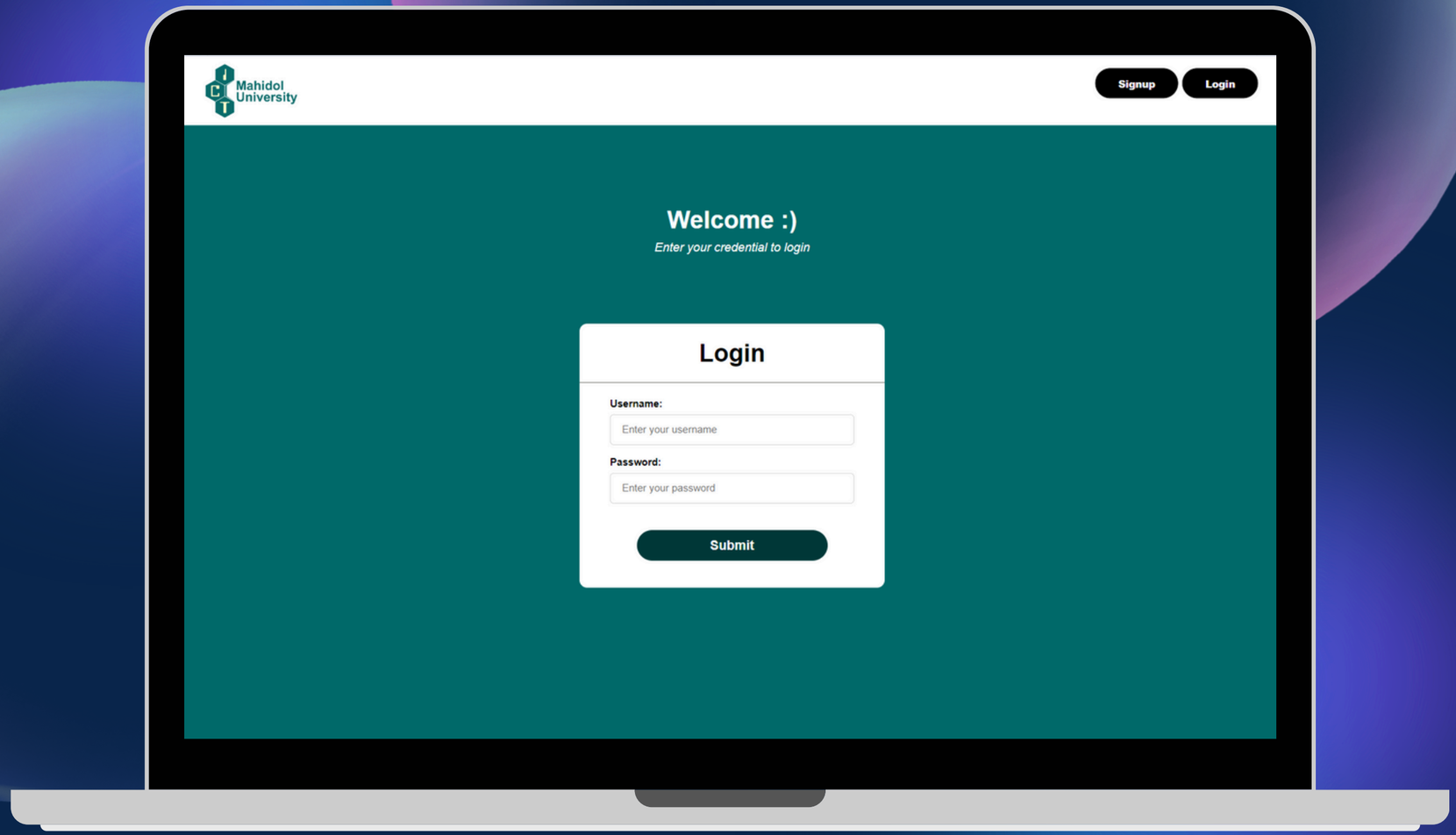


WELCOME PAGE

After login was successful, the user will access the welcome page and can logout through the “Logout” button.

```
ion authen() {
const token = localStorage.getItem('token');
fetch('http://localhost:3000/db_authen', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
        'Authorization': 'Bearer ' + token
    },
})
    .then(response => response.json())
    .then(data_authen => {
        if (data_authen.message == 'Token is valid') {
            //pass
        } else {
            localStorage.removeItem('token');
            alert('Please login first!')
            window.location.href = '/login';
        }
        console.log('Success:', data_authen);
    })
    .catch((error) => {
        console.error('Error:', error);
    });
```

DEMO SHOWING OUR IMPLEMENTATION



REFERENCES

01 A STUDY AND ANALYSIS OF KEYSTROKE DYNAMICS AND ITS ENHANCEMENT FOR PROFICIENT USER AUTHENTICATION

https://www.researchgate.net/publication/273382241_A_Study_And_Analysis_of_Keystroke_Dynamics_And_Its_Enhancement_For_Proficient_User_Authentication

02 USERNAME AND PASSWORD VERIFICATION THROUGH KEYSTROKE DYNAMICS

https://core.ac.uk/download/pdf/230452244.pdf

03 SALT AND HASH PASSWORDS WITH BCRYPT

https://heynode.com/blog/2020-04/salt-and-hash-passwords-bcrypt/

04 PLUEMTHNN/KEY-STROKE-DYNAMIC-LOGIN

https://github.com/pluemthnn/key-stroke-dynamic-login?tab=readme-ov-file

05 COSINE SIMILARITY: HOW DOES IT MEASURE THE SIMILARITY, MATHS BEHIND AND USAGE IN PYTHON

https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db

THANK YOU FOR ATTENDTION

FEEL FREE TO ASK ANY QUESTION

Members:

Miss Supithcha	*Jongphoemwatthanaphon*	*6488045*
Miss Sasasuang	*Pattanakitjaroenchai*	*6488052*
Miss Nisakorn	*Ngaosri*	*6488226*