Dokumentation der Eigenleistung

Oskar Bartsch / Matrikelnummer 1115334

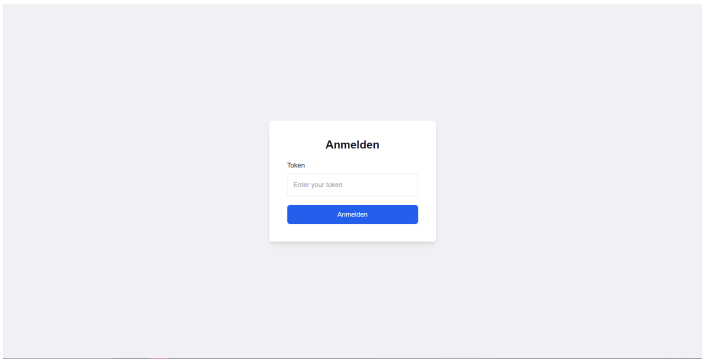
In diesem Kapitel möchte ich den Prozess erläutern, den ich bei der Implementierung des Login-Systems durchlaufen habe (bzw. in Bezug auf die Dateien Anmeldung.html, anmeldung.js und app.py, insbesondere die app.route(/login)).

Zunächst einmal, welches Ziel verfolge ich? Es ging darum, eine Seite zu erstellen, auf der sich Bewerber melden können. Mein erster Ansatz bestand darin, eine E-Mail-Adresse zusammen mit einem Passwort zu verwenden, wobei das Passwort zunächst als Platzhalter diente. Diese Platzhalter-Anmeldung leitete den Benutzer vorerst auf eine Platzhalterseite weiter, in diesem Fall google.com. Dies funktionierte problemlos, zunächst nur mit einer HTML-Datei.

Als nächsten Schritt versuchte ich, dass der User sich erst dann einloggen kann, wenn nur ein Token in der Datenbank registriert ist. Hierbei stieß ich auf einige Schwierigkeiten, da es oftmals Probleme gab, das Token korrekt abzufragen. Trotz korrektem Token führte die fehlerhafte Abfrage zu Problemen. Nach einigen frustrierenden Versuchen fiel mir auf, dass ich die app.py nicht richtig ausgeführt und mein Verzeichnis in der PowerShell zeigte auf einen anderen Ordner, was alle Probleme in der HTML Datei verursachte. As well, I have been advised to create a separate JS file where the whole JavaScript code had to be embedded.

Nachdem ich diese Probleme behoben hatte, funktionierte alles wie vorgesehen. Im letzten Schritt sollte der Benutzer anstelle der Platzhalterseite google.com auf unsere app.route(/home) weitergeleitet werden. Dies übernahm Christoph für mich, nicht weil ich es nicht konnte, sondern weil er dies gerne selbst umsetzen wollte.

Hier sehen Sie ein Bild der Seite am Ende.
Ich habe mir beim Designen bei der Website gesagt, ich wollte es ein bisschen homogener und professioneller haben. Weniger Schnickschnack und dann einfach nur eine Seite, die ihren Zweck erfüllen soll.



Frontend

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Anmelden</title>
    <script src="https://cdn.tailwindcss.com"></script>
    <script>
        tailwind.config = {
            theme:{
                fontFamily:{
                    sans:['Arial']
                }
            }
        }
    </script>
    <style>
        #error-message {
            display: none;
        }
    </style>
</head>
<body class="bg-gray-100 text-gray-900 flex items-center justify-center h-screen">

    <div id="login-page" class="bg-white p-12 rounded-lg shadow-lg w-full max-w-md mx-auto">
        <h2 class="text-3xl font-semibold mb-6 text-center">Anmelden</h2>
        <form id="login-form">
            <div class="mb-6">
                <label class="block text-lg font-medium mb-2" for="token">Token</label>
                <input class="w-full p-4 border rounded-lg text-lg" type="text" id="token" placeholder="Enter your token">
            </div>
            <div id="error-message" class="text-red-500 mb-6 text-center">Invalid token.</div>
            <button class="bg-blue-600 text-white px-6 py-3 rounded-lg w-full text-lg hover:bg-blue-700" type="submit">Anmelden</button>
        </form>
    </div>

    <script src="{{ url_for('static', filename='anmeldung.js' ) }}"></script>

</body>
</html>
```

Ich habe die <!doctype-HTML-Deklaration> verwendet, um das Dokument zu starten und sicherzustellen, dass es im Standardmodus im Browser gerendert wird. Kopfbereich, UTF-8-Zeichenkodierung, Viewport-Meta-Tag und die Anmelde Schaltfläche als Seitentitel

```
<script src="https://cdn.tailwindcss.com"></script>
<script>
    tailwind.config = {
        theme:{
            fontFamily:{
                sans:['Arial']
            }
        }
    }
</script>
```

Um das Styling auf meiner Website zu vereinfachen, nutze ich Tailwind CSS, da es einfacher ist und ohne viel eigenständiges CSS die Website flexibel zu gestalten. Daraufhin habe ich die Schrift auf Arial gesetzt, damit sie einfach schlicht und gut lesbar ist.

```
    <style>
        #error-message {
            display: none;
        }
    </style>
```

Damit eine dynamische Fehlermeldung angezeigt wird, habe ich mit dem Style Tag eine neue Regel hinzugefügt, die mir erlaubt, am Anfang ein Error message zu verstecken und wenn das Token eingegeben wird und falsch ist, wird durch den JavaScript-Teil dies initialisiert.

```
<body class="bg-gray-100 text-gray-900 flex items-center justify-center h-screen">

    <div id="login-page" class="bg-white p-12 rounded-lg shadow-lg w-full max-w-md mx-auto">
        <h2 class="text-3xl font-semibold mb-6 text-center">Anmelden</h2>
        <form id="login-form">
            <div class="mb-6">
                <label class="block text-lg font-medium mb-2" for="token">Token</label>
                <input class="w-full p-4 border rounded-lg text-lg" type="text" id="token" placeholder="Enter your token">
            </div>
            <div id="error-message" class="text-red-500 mb-6 text-center">Invalid token.</div>
            <button class="bg-blue-600 text-white px-6 py-3 rounded-lg w-full text-lg hover:bg-blue-700" type="submit">Anmelden</button>
        </form>
    </div>

    <script src="{{ url_for('static', filename='anmeldung.js' ) }}"></script>

</body>
```

Der Body-Tag enthält den Hauptinhalt von der Seiter. Damit ich eine benutzerfreundliche Oberfläche schaffen kann, habe ich das Layout so gestaltet, dass die Anmeldebox in der Mitte des Bildschirms ist. Damit eine Anmeldebox entsteht, habe ich eine Flexbox benutzt, die sowohl horizontal als auch vertikal

zentriert ist. Der Hintergrund ist in einem schlichten Grau angelegt, damit der Fokus auf dem Formular liegt.

Die Anmeldebox besteht aus einem Eingabebereich um ein gültiges Token einzulegen und einem Button der wenn das token falsch ist die Fehlermeldung anzeigt. Das Eingabefeld ist mit einer eindeutigen ID versehen, damit ich später im Javascript darauf zurückgreifen kann. Der Bereich für die Fehlermeldung ist zunächst verdeckt und wird angezeigt wenn nötig. Der Anmeldebutton ist so gestaltet, dass wenn man darüber hoverte, er seine Farbe ändert, um einfach eine visuelle Rückmeldung zu geben.

```
function handleLogin() {
    const token = document.getElementById('token').value;
    console.log('Attempting login with token:', token);

    fetch('/login', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ token: token }),
    })
    .then(response => response.json())
    .then(data => {
        console.log('Server response:', data);
        if (data.success) {
            localStorage.setItem('user_id', data.user_id);
            console.log('Redirecting to Home');
            window.location.href = '/Home';
        } else {
            const errorMessage = document.getElementById('error-message');
            if (errorMessage) {
                errorMessage.style.display = 'block';
            } else {
                console.error('Error message element not found');
            }
        }
    })
    .catch(error => {
        console.error('Error:', error);
    });
}

document.addEventListener('DOMContentLoaded', function() {
    const form = document.getElementById('login-form');
    form.addEventListener('submit', function(event) {
        event.preventDefault();
        handleLogin();
    });
});
```

```
function handleLogin() {
    const token = document.getElementById('token').value;
    console.log('Attempting login with token:', token);

    fetch('/login', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ token: token }),
    })
```

Die Funktion handle login() ist für die Anmelde Logik der Kern. Sie beginnt damit, dass der Token-Wert, der eingegeben worden ist, abzurufen ist. Dieser Token wird daraufhin mit document.getElementById(´Token´).value überprüft und ermittelt. Nachdem das Token ermittelt wurde wird eine Post-Anfrage an den Server geschickt. Da kommt die fetch-API zum Einsatz, die es mir ermöglicht, eine Anfrage zu stellen, welche asynchron ist. Diese enthält den Token im JSON-Format und wird auch so gekennzeichnet. Sobald die Anfrage bearbeitet wurde, wird diese in ein JSON-Objekt umgewandelt. Die Rückmeldung wird Protokolliert falls Fehler entstehen könnte man diese Abfangen.

```
.then(response => response.json())
.then(data => {
    console.log('Server response:', data);
    if (data.success) {
        localStorage.setItem('user_id', data.user_id);
        console.log('Redirecting to Home');
        window.location.href = '/Home';
    } else {
        const errorMessage = document.getElementById('error-message');
        if (errorMessage) {
            errorMessage.style.display = 'block';
        } else {
            console.error('Error message element not found');
        }
    }
})
.catch(error => {
    console.error('Error:', error);
});
```

Falls die Authentifizierung erfolgreich ist, wird die `user_id`, die der Server zurückgibt, im `localStorage` gespeichert. Dieser Schritt ist entscheidend für das Wiedererkennen des Benutzers bei späteren Besuchen. Der Benutzer wird dann zur

Wenn nun diese Token Authentifizierung erfolgreich war, dann wird die User_id, die vom Server gegeben wird, im LocalStorage gespeichert, um zu erkennen, wer gerade die Fragen beantwortet. Daraufhin wird der Nutzer auf unsere Startseite geleitet, um die Tests zu bearbeiten. Sollte die Authentifizierung fehlschlagen, dann wird die vorhin erwähnte Fehlermeldung sichtbar gemacht. Falls es irgendwelche Probleme mit der Anfrage gibt gibt die Konsole den Fehler aus um ihn zu diagnostizieren.

```
document.addEventListener('DOMContentLoaded', function() {
    const form = document.getElementById('login-form');
    form.addEventListener('submit', function(event) {
        event.preventDefault();
        handleLogin();
    });
});
```

Letzter Teil des JavaScripts wird der Event-Listener, dieser wird beim Absenden des Formulars aufgerufen und wird erst aktiv, nachdem das Dokument geladen ist. Dies ist dazu da, dass das Standardverhalten geändert wird, welches normalerweise einfach nur die Seite neu laden würde und ruft stattdessen die handle login() funktion auf.

Backend

```python
@app.route('/login', methods=['POST'])
def login():
    if request.is_json:
        data = request.get_json()
        token = data.get('token')
        print(f"Received token: {token}")  # Log the received token

        db = getDB()
        cur = db.execute('SELECT ID, Token FROM User')
        all_users = cur.fetchall()
        print(f"All users: {all_users}")  # Log all users and their tokens

        cur = db.execute('SELECT ID FROM User WHERE Token = ?', (token,))
        user_id = cur.fetchone()
        print(f"Query result: {user_id}")  # Log the query result

        close_connection()

        if user_id:
            session['user_id'] = user_id[0]
            return jsonify({'success': True, 'user_id': user_id[0]})
        else:
            return jsonify({'success': False, 'message': 'Invalid token'})
    return jsonify({'success': False, 'message': 'Invalid request'})
```

Das Backend fängt zunächst damit an, dass die Route überprüft wird und es gefragt wird, ob die Anfrage in einem JSON-Format vorliegt. Falls dies so ist, dann wird der Inhalt gelesen(also das Token). Der Empfangene Token wird in der Konsole ausgegeben um sicher zu gehen das der Token richtig empfangen wurde. Damit dieser Token verglichen wird wird mit db = getDB() eine verbindung zur Datenbank hergestellt wird. Und dann werden alle User Tokens abgefragt, um diese mit dem eingegebenen Token zu vergleichen. Falls der Token gleich einem anderen ist, dann wird die Benutzer ID in der Session gespeichert und die Antwort an den Clienten gegeben wird, damit dieser erkennt, dass das Token richtig war. Falls dies aber nicht der Fall ist, wird die Fehlermeldung zurückgegeben und gesagt, dass der Token ungültig ist. Falls es kein JSON Format ist, dann wird ebenfalls ein Fehler ausgegeben, der darauf hinweist, dass die Anfrage ungültig ist. Am Ende wird nun noch einmal die Datenbankverbindung geschlossen, da sie nicht mehr gebraucht wird und Ressourcen freigibt.