Konzept: Nutzerfeedback- und News-Feature in der Gym Progress Tracking App

Gym Progress Tracking App

February 8, 2025

Einleitung

Neben den bisherigen Funktionen der App – dem Tracking von Trainingseinheiten, Reporting und Leaderboard-Features – soll nun ein Nutzerfeedback-Feature integriert werden. Dieses ermöglicht es den Nutzern, zu jedem Trainingsgerät Feedback (z.B. bei Beschädigungen oder sonstigen Anmerkungen) abzugeben. Gleichzeitig kann der Studiobetreiber diese Feedbacks einsehen und gezielt auch Rückfragen stellen, etwa bei wenig genutzten Geräten. Darüber hinaus soll die App auch einen Bereich für aktuelle News und Werbeaktionen enthalten, um Informationen wie Sonderangebote, Aktionen oder wichtige Mitteilungen direkt in der App bereitzustellen.

1 Idee und Nutzen des Features

- **Feedback für Geräte:**
  - Nutzer können direkt in der App Feedback zu einzelnen Geräten abgeben, z.B. wenn etwas defekt ist oder Verbesserungsvorschläge bestehen.
  - Studiobetreiber erhalten diese Rückmeldungen, um Probleme schnell zu erkennen und zu beheben.
- **Gezielte Feedback-Aufforderung:**
  - Im Reporting-Dashboard kann der Betreiber sehen, welche Geräte selten genutzt werden.
  - Auf Basis dieser Daten kann automatisch oder manuell eine Aufforderung an die Nutzer gesendet werden, Feedback zu dem betreffenden Gerät zu geben (z.B. „Nutzen Sie Gerät 102 weniger? Hinterlassen Sie uns Ihr Feedback!“).
- **News und Informationen:**
  - Der News-Bereich liefert aktuelle Informationen, Gym-News, und Werbeaktionen (z.B. Merch-Aktionen).
  - Dies verbessert das Nutzererlebnis und kann zusätzliche Einnahmequellen eröffnen.

2 Umsetzung: Schritt-für-Schritt Anleitung

2.1 1. Datenmodellierung

1. **Erstellen einer `feedback`-Tabelle:**
   Führe in deiner PostgreSQL-Datenbank folgendes SQL-Skript aus:

```
DROP TABLE IF EXISTS feedback;
CREATE TABLE feedback (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL,
  device_id INTEGER NOT NULL,
  feedback_text TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2. **Optional: Erstellen einer `news`-Tabelle:**
   Falls du einen News-Bereich implementieren möchtest, führe folgendes Skript aus:

```
DROP TABLE IF EXISTS news;
CREATE TABLE news (
  id SERIAL PRIMARY KEY,
  title TEXT NOT NULL,
  content TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2.2 2. Backend-Endpunkte erweitern

1. **Feedback einreichen (POST):**
   Füge in deiner `server.js` folgenden Endpunkt ein:

```
app.post('/api/feedback', async (req, res) => {
  const { userId, deviceId, feedbackText } = req.body;
  if (!userId || !deviceId || !feedbackText) {
    return res.status(400).json({ error: 'Ungültige Eingabedaten' });
  }
  try {
    const result = await pool.query(
      'INSERT INTO feedback (user_id, device_id, feedback_text) VALUES ($1, $2, $
      [userId, deviceId, feedbackText]
    );
    res.json({ message: 'Feedback erfolgreich gespeichert', data: result.rows[0]
  } catch (error) {
    console.error('Fehler beim Speichern des Feedbacks:', error.message);
    res.status(500).json({ error: 'Serverfehler beim Speichern des Feedbacks' });
  }
```

```
});
```

2. **Feedback abrufen (GET):**
   Erstelle einen Endpunkt, der alle Feedbacks für ein bestimmtes Gerät zurückgibt:

```
app.get('/api/feedback/:deviceId', async (req, res) => {
  const { deviceId } = req.params;
  if (!deviceId) {
    return res.status(400).json({ error: 'Ungültige Geräte-ID' });
  }
  try {
    const result = await pool.query(
      'SELECT * FROM feedback WHERE device_id = $1 ORDER BY created_at DESC',
      [deviceId]
    );
    res.json({ message: 'Feedback erfolgreich abgerufen', data: result.rows });
  } catch (error) {
    console.error('Fehler beim Abrufen des Feedbacks:', error.message);
    res.status(500).json({ error: 'Serverfehler beim Abrufen des Feedbacks' });
  }
});
```

3. **News abrufen (GET):**
   Erstelle einen Endpunkt, der aktuelle News und Werbeaktionen liefert:

```
app.get('/api/news', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM news ORDER BY created_at DESC
    res.json({ message: 'News erfolgreich abgerufen', data: result.rows });
  } catch (error) {
    console.error('Fehler beim Abrufen der News:', error.message);
    res.status(500).json({ error: 'Serverfehler beim Abrufen der News' });
  }
});
```

2.3 3. Frontend-Integration

1. **Feedback-Formular:**
   In der Detailseite eines Gerätes (z. B. im Dashboard oder einer separaten Geräte-Detailseite) integriere ein Formular, in dem der Nutzer Feedback eingeben kann. Beispiel (FeedbackForm.js):

```
import React, { useState } from 'react';
import { API_URL } from './config';
```

```
function FeedbackForm({ deviceId, userId, onFeedbackSubmitted }) {
  const [feedbackText, setFeedbackText] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await fetch(`${API_URL}/api/feedback`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ userId, deviceId, feedbackText }),
      });
      const result = await response.json();
      if (response.ok) {
        alert(result.message);
        setFeedbackText('');
        if (onFeedbackSubmitted) onFeedbackSubmitted(result.data);
      } else {
        alert(result.error);
      }
    } catch (error) {
      console.error('Fehler beim Senden des Feedbacks:', error);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <textarea
        value={feedbackText}
        onChange={(e) => setFeedbackText(e.target.value)}
        placeholder="Geben Sie hier Ihr Feedback ein..."
        rows="4"
        cols="50"
      />
      <br />
      <button type="submit">Feedback absenden</button>
    </form>
  );
}

export default FeedbackForm;
```

2. **Feedback anzeigen:**
   Erstelle eine Komponente, die das Feedback für ein bestimmtes Gerät anzeigt (z. B. FeedbackList.js):

```
import React, { useState, useEffect } from 'react';
import { API_URL } from './config';
```

```
function FeedbackList({ deviceId }) {
  const [feedbacks, setFeedbacks] = useState([]);

  useEffect(() => {
    async function fetchFeedback() {
      try {
        const response = await fetch(`${API_URL}/api/feedback/${deviceId}`);
        const result = await response.json();
        if (response.ok && result.data) {
          setFeedbacks(result.data);
        } else {
          console.error(result.error);
        }
      } catch (error) {
        console.error('Fehler beim Abrufen des Feedbacks:', error);
      }
    }
    fetchFeedback();
  }, [deviceId]);

  return (
    <div>
      <h3>Feedbacks</h3>
      {feedbacks.length > 0 ? (
        <ul>
          {feedbacks.map(feedback => (
            <li key={feedback.id}>
              <strong>{feedback.created_at}:</strong> {feedback.feedback_text}
            </li>
          ))}
        </ul>
      ) : (
        <p>Kein Feedback vorhanden.</p>
      )}
    </div>
  );
}

export default FeedbackList;
```

3. **News-Bereich:**
   Erstelle eine Komponente, die aktuelle News anzeigt (z. B. NewsFeed.js):

```
import React, { useState, useEffect } from 'react';
import { API_URL } from './config';
```

```
function NewsFeed() {
  const [news, setNews] = useState([]);

  useEffect(() => {
    async function fetchNews() {
      try {
        const response = await fetch(`${API_URL}/api/news`);
        const result = await response.json();
        if (response.ok && result.data) {
          setNews(result.data);
        } else {
          console.error(result.error);
        }
      } catch (error) {
        console.error('Fehler beim Abrufen der News:', error);
      }
    }
    fetchNews();
  }, []);

  return (
    <div>
      <h3>Aktuelle News & Aktionen</h3>
      {news.length > 0 ? (
        <ul>
          {news.map(item => (
            <li key={item.id}>
              <h4>{item.title}</h4>
              <p>{item.content}</p>
              <small>{new Date(item.created_at).toLocaleDateString()}</small>
            </li>
          ))}
        </ul>
      ) : (
        <p>Keine News verfügbar.</p>
      )}
    </div>
  );
}

export default NewsFeed;
```

4. **Integration in eine Gesamtseite:**
   Du könntest z. B. eine Seite für Studiobetreiber erstellen, die das Reporting-Dashboard, Feedback und News kombiniert. Diese Seite könnte `OwnerDashboard.js` heißen und alle relevanten Komponenten (ReportingDashboard, FeedbackList, NewsFeed) anzeigen.

2.4 4. Routing anpassen

1. Öffne deine `App.js` und füge eine neue Route für das Reporting-Dashboard bzw. die Owner-Ansicht hinzu:

```
<Route path="/reporting" element={<ReportingDashboard />} />
```

2. Ergänze in der globalen Navigation einen Link zum Reporting-Dashboard:

```
<li>
  <Link to="/reporting">Reporting-Dashboard</Link>
</li>
```

2.5 5. Testen und Validieren

1. Starte Backend und Frontend neu.
2. Teste den API-Endpunkt `/api/feedback` und `/api/news` (z. B. mit Postman).
3. Überprüfe, ob in der Geräte-Detailansicht (oder im Dashboard) das Feedback-Formular und die Feedback-Liste korrekt angezeigt werden.
4. Teste, ob der Studiobetreiber über das Reporting-Dashboard die aggregierten Nutzungsdaten (und ggf. News) einsehen kann.

Fazit

Dieses Konzept erweitert die App um ein Nutzerfeedback-Feature und einen News-Bereich:

- Nutzer können Feedback zu einzelnen Geräten abgeben, welches in der Datenbank gespeichert wird.
- Der Studiobetreiber kann das Feedback einsehen und bei Bedarf gezielt nachfragen.
- Ein News-Bereich informiert über aktuelle Gym-Neuigkeiten und Aktionen.
- Diese Erweiterungen lassen sich in das bestehende Reporting-Dashboard integrieren, sodass Studiobetreiber einen umfassenden Überblick über die Nutzung und den Zustand der Geräte erhalten.