Ausführliche Schritt-für-Schritt-Anleitung für die Deployment-Pipeline und GitHub-Synchronisation der GymApp

Dein Name

30. März 2025

1 Übersicht

Diese Anleitung beschreibt alle notwendigen Schritte, um sicherzustellen, dass:

1. Dein Node/Express-Backend über Render korrekt konfiguriert ist und mit der Render-Datenbank kommuniziert.
2. Deine Datenbank (mit SSL) aus der `.env`-Datei ausgelesen wird.
3. Du deine lokale Datenbank in Render migrierst.
4. Deine Flutter-App die öffentliche URL deines Render-Backends nutzt, sodass sie von überall (z.B. im Fitnessstudio) auf die Daten zugreifen kann.
5. Du deine Änderungen im GitHub-Repository synchronisierst, sodass Render automatisch aktualisierte Versionen deployed.
6. Backup-Strategien für die Datenbank erläutert werden.

2 Backend-Konfiguration und Deployment

2.1 1. .env-Datei erstellen und konfigurieren

Erstelle im Root-Verzeichnis deines Backend-Projekts (dort, wo sich `package.json` befindet) eine Datei namens `.env`. Beispielinhalt:

```
# Lokale Konfiguration (zum Testen auf deinem PC):
%
% PORT=3000
% JWT_SECRET=schluessel
% DATABASE_URL=postgres://postgres:Test123!@127.0.0.1:5432/gymapp
%

# Render-Konfiguration (f r die Cloud-Umgebung):
PORT=3000
JWT_SECRET=schluessel
```

```
DATABASE_URL=postgres://gymapp_db_isz2_user:
    IBX6ieKGWnZ1XTRPjKo1cUCUMGAml6Ci@dpg-cvjkpmq4d50c73djnrc0-a.
    frankfurt-postgres.render.com/gymapp_db_isz2
```

Hinweis: Kommentiere die Zeilen aus, die du gerade nicht nutzen möchtest. Achte darauf, dass diese Datei in deiner `.gitignore` aufgeführt wird, um sensible Daten vor dem Push in dein öffentliches Repository zu schützen.

2.2 2. Anpassung der `server.js`

Öffne deine `server.js` und füge ganz oben die Zeile zum Laden der Umgebungsvariablen ein:

```
require('dotenv').config();
```

Ersetze alle Vorkommen des hardcodierten JWT-Schlüssels `'schluessel'` durch `process.env.JWT`$_S EC$

2.3 3. Anpassung der `db.js`

Öffne db.js und passe es wie folgt an, um SSL zu aktivieren:

```
require('dotenv').config(); // Umgebungsvariablen laden
const { Pool } = require('pg');

const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
  max: 200,                % Maximale Verbindungen
  idleTimeoutMillis: 30000, % Schlie t inaktive Verbindungen
     nach 30 Sekunden
  ssl: { rejectUnauthorized: false } % Aktiviert SSL f r Render
});

module.exports = pool;
```

2.4 4. GitHub Repository aktualisieren und Deploy triggern

Führe im Root-Verzeichnis deines Projekts (dort, wo sich auch .env und package.json befinden) folgende Befehle in deinem Terminal aus:

```
git status
git add .
git commit -m "Update: Backend-Konfiguration (.env, server.js, db
    .js) angepasst f r Render"
git push origin main
```

Render ist mit deinem GitHub-Repository verknüpft, sodass bei jedem Push ein neuer Build und Deploy ausgelöst wird.

3 Datenbankmigration von Lokal zu Render

3.1 1. Lokalen Dump erstellen

Erstelle einen Dump deiner lokalen Datenbank. Öffne eine Windows-Eingabeaufforderung (CMD) und führe folgende Befehle aus:

```
cd C:\Users\danie\Desktop
set PGPASSWORD=Test123!
"C:\Program Files\PostgreSQL\17\bin\pg_dump.exe" -U postgres -h
   127.0.0.1 -d gymapp -p 5432 --verbose -f gymapp_dump.sql
```

Überprüfe anschließend den Inhalt der Datei gymapp_dump.sql in einem Texteditor, um sicherzustellen, dass sie SQL-Befehle (CREATE TABLE, INSERT INTO etc.) enthält.

3.2 2. Dump in die Render-Datenbank importieren

Verwende den externen Connection String, um den Dump in deine Render-Datenbank zu importieren:

```
set PGPASSWORD=IBX6ieKGWnZ1XTRPjKo1cUCUMGAml6Ci
"C:\Program Files\PostgreSQL\17\bin\psql.exe" -h dpg-
   cvjkpmq4d50c73djnrc0-a.frankfurt-postgres.render.com -U
   gymapp_db_isz2_user -d gymapp_db_isz2 -p 5432 -f "C:\Users\
   danie\Desktop\gymapp_dump.sql"
```

Überprüfe anschließend in einem Tool wie pgAdmin oder über den psql-Befehl \dt, ob alle Tabellen und Daten in der Render-Datenbank vorhanden sind.

4 Flutter-App Konfiguration und Deployment

4.1 1. API-Endpunkt in der Flutter-App konfigurieren

Öffne die Datei lib/config.dart in deinem Flutter-Projekt und stelle sicher, dass der API-URL auf deine öffentliche Backend-URL zeigt. Beispiel:

```
/* F r den produktiven Betrieb (Render): */
const String API_URL = "https://gymapp-test1.onrender.com";

/* F r lokale Tests:
const String API_URL = "http://192.168.212.149:3000";
*/
```

4.2 2. App auf einem Android-Gerät starten

Schließe dein Android-Gerät an oder starte einen Emulator und führe in deinem Flutter-Projek

```
flutter run
```

Dadurch wird die App gebaut und auf dem Gerät installiert. Die App kommuniziert über die in config.dart eingestellte API-URL mit deinem Render-Backend, welches auf die Render-Datenbank zugreift.

4.3 3. Debuggen und Testen

- Überprüfe in der Flutter-Konsole, ob API-Aufrufe (z.B. /api/devices) erfolgreich Daten liefern.
- Nutze bei Bedarf Tools wie Postman, um einzelne Endpunkte direkt zu testen.
- Schau in die Render-Logs, falls 500er-Fehler auftreten, um den genauen Fehler zu identifizieren.

5 GitHub-Synchronisation und kontinuierliches Deployment

5.1 1. Änderungen lokal committen und pushen

Wenn du in deinem Flutter- oder Backend-Projekt Änderungen vornimmst, führe folgende Schritte in deinem Terminal (im Root-Verzeichnis des Projekts) aus:

```
git status
git add .
git commit -m "Beschreibung der vorgenommenen  nderungen  "
git push origin main
```

Da Render mit deinem GitHub-Repository verknüpft ist, wird bei jedem Push automatisch ein neuer Build ausgelöst und deine Änderungen werden live geschaltet.

5.2 2. Regelmäßige Backups der Datenbank erstellen

1. Lokaler Backup: Erstelle regelmäßig einen Dump deiner Render-Datenbank (z.B. einmal im Monat) mit dem externen Connection String:

```
set PGPASSWORD=IBX6ieKGWnZ1XTRPjKo1cUCUMGAml6Ci
"C:\Program Files\PostgreSQL\17\bin\pg_dump.exe" -U
   gymapp_db_isz2_user -h dpg-cvjkpmq4d50c73djnrc0-a.
   frankfurt-postgres.render.com -d gymapp_db_isz2 -p 5432 --
   verbose -f "C:\Users\danie\Desktop\render_dump.sql"
```

2. Render-Datenbank-Backup: Render bietet keine integrierte Backup-Funktion in der kostenlosen Variante. Daher musst du deine Backups lokal erstellen und speichern. Alternativ kannst du ein Skript einrichten, das den Dump regelmäßig (z.B. per Cron-Job) erstellt und in einem sicheren Cloud-Speicher ablegt.

6 Zusammenfassung

1. Backend-Konfiguration:
   - Erstelle und konfiguriere die .env-Datei mit lokalen und Render-Varianten.
   - Passe server.js und db.js an, sodass Umgebungsvariablen und SSL für Render genutzt werden.
2. GitHub-Synchronisation:

- Committe und pushe alle Änderungen in dein GitHub-Repository, sodass Render automatisch neu deployed.

3. Datenbankmigration:

   - Erstelle einen Dump deiner lokalen Datenbank mit pg_dump und importiere diesen in deine Render-Datenbank.

4. Flutter-App Konfiguration:

   - Stelle in lib/config.dart den API-Endpunkt auf die öffentliche URL deines Render-Backends.
   - Starte die App mit flutter run auf deinem Android-Gerät.

5. Kontinuierliche Updates und Backups:

   - Synchronisiere regelmäßig deine Änderungen via Git (commit & push).
   - Erstelle regelmäßige Backups der Render-Datenbank (lokal via pg_dump oder automatisiert per Skript).