Files

Projekt-Dokumentation.md

Projekt-Dokumentation.md 3.85 KiB

Kalaha Online - Projektdokumentation

1. Projektübersicht

Kalaha Online ist eine webbasierte Umsetzung des klassischen Brettspiels Kalaha (auch bekannt als Mancala). Das Projekt ist als verteilte Anwendung konzipiert, die hohe Skalierbarkeit und Sicherheit bietet.

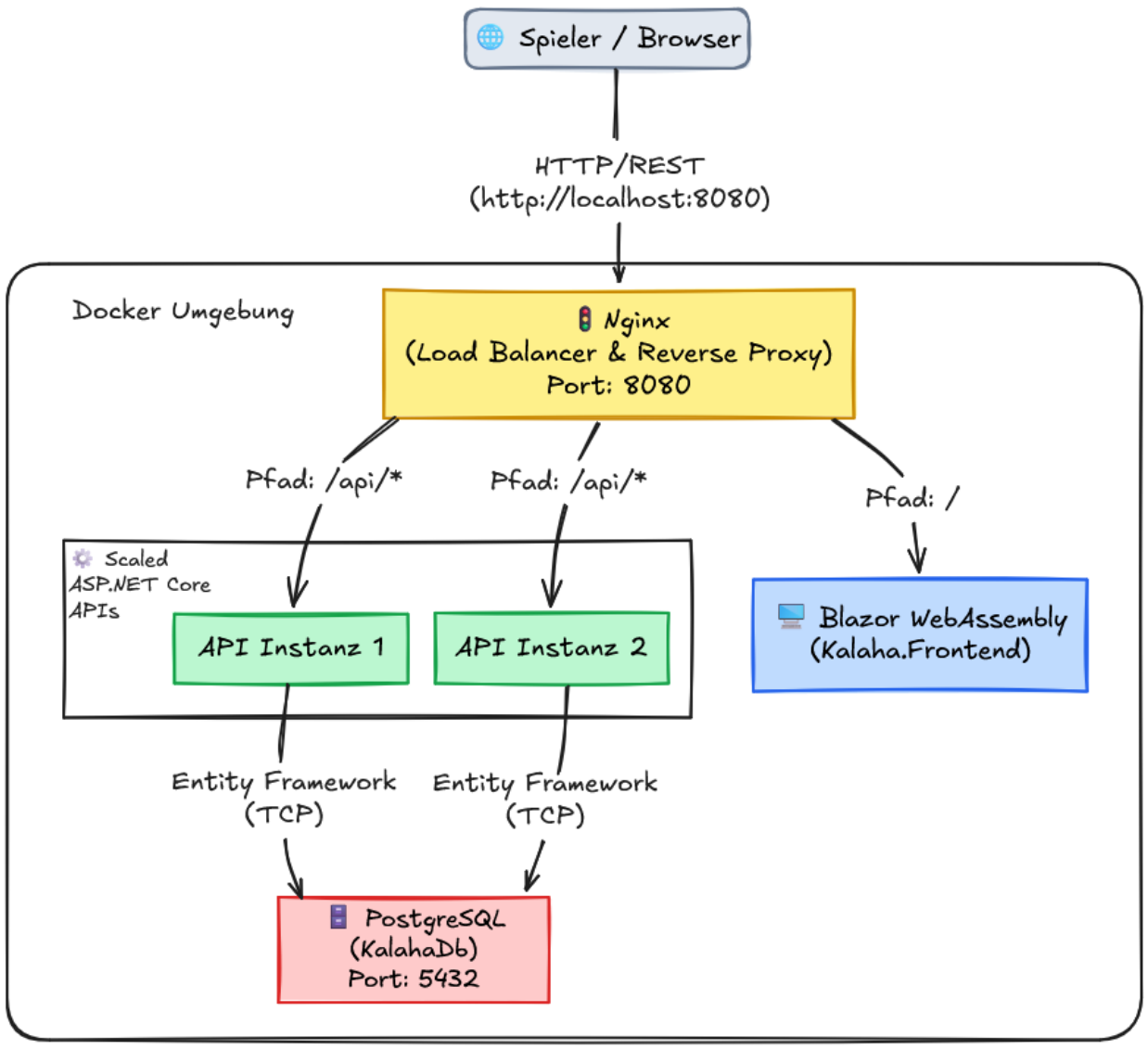
Kerntechnologien

- **Backend:** ASP.NET Core Web API 8.0
- **Frontend:** Blazor WebAssembly (WASM)
- **Datenbank:** PostgreSQL (via Entity Framework Core)
- **Containerisierung:** Docker & Docker Compose
- **Authentifizierung:** JWT (JSON Web Tokens) mit Access- und Refresh-Tokens

---

2. Architektur

Die Anwendung folgt einer klassischen Client-Server-Architektur, ist jedoch für den Betrieb in einer skalierten Umgebung (z.B. Load Balancer mit mehreren API-Instanzen) optimiert.



Komponenten

- **Frontend (Blazor WASM):** Läuft vollständig im Browser des Benutzers. Kommuniziert über REST-Schnittstellen mit dem Backend.
- **Backend (ASP.NET API):** Stateless API-Instanzen. In der Testumgebung skaliert auf 2 Instanzen (`docker compose up --scale api=2`).
- **Datenbank (PostgreSQL):** Zentraler Datenspeicher für Benutzer, Spiele und Spielzustände.

- **Nginx (Optional/Load Balancer):** Verteilt Anfragen auf die verfügbaren API-Instanzen.
- `Database-Schema.puml`: Definition der Datenbanktabellen und Relationen.

---

3. Authentifizierung & Sicherheit

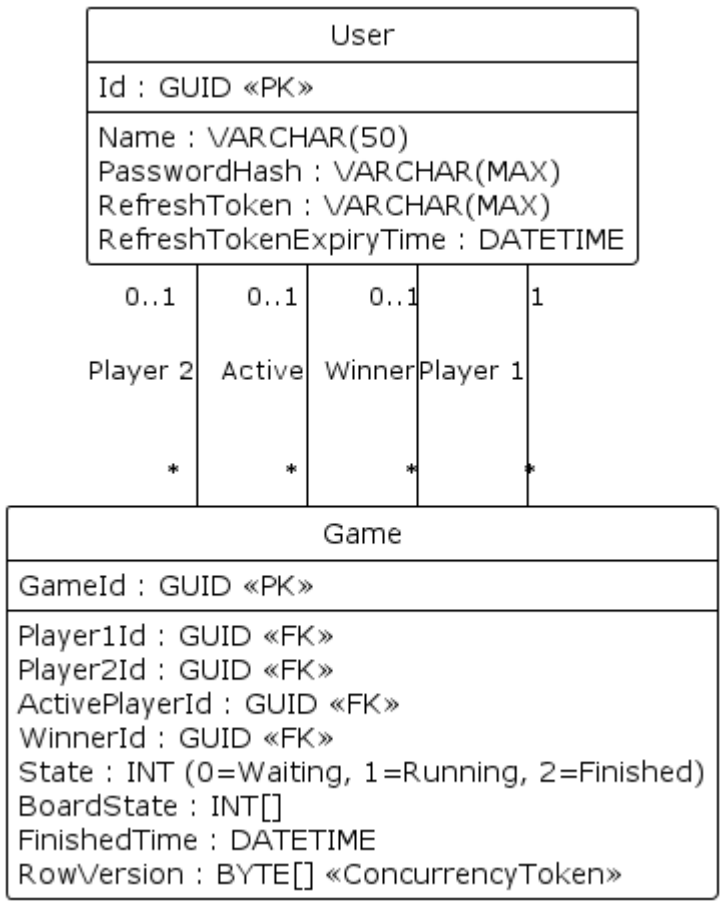
Das System nutzt ein modernes JWT-Verfahren zur Sicherung der Endpunkte.

- **Access Token:** Kurzlebig (15 Min), wird bei jedem Request im Authorization-Header mitgesendet.
- **Refresh Token:** Langlebig (7 Tage), wird genutzt, um automatisch ein neues Access Token zu erhalten, ohne dass der Benutzer sich neu einloggen muss.
- **Token-Rotation:** Bei jedem Refresh wird auch das Refresh Token erneuert, um Missbrauch vorzubeugen.
- **Passwort-Sicherheit:** Passwörter werden mit **BCrypt** gespeichert.

---

4. Datenbankmodell

Das Schema umfasst zwei Hauptentitäten:



User

- `Id`: Eindeutige Kennung (Guid).
- `Name`: Benutzername (max. 50 Zeichen).
- `PasswordHash`: BCrypt-Hash des Passworts.
- `RefreshToken`: Aktuell gültiger Refresh-Token.
- `RefreshTokenExpiryTime`: Ablaufdatum des Refresh-Tokens.

Game

- `GameId`: Eindeutige Kennung des Spiels.
- `Player1Id` / `Player2Id`: Verknüpfung zu den Spielern.
- `ActivePlayerId`: Wer ist aktuell am Zug.
- `WinnerId`: Gewinner des Spiels (nach Abschluss).
- `State`: Zustand des Spiels (`Waiting`, `Running`, `Finished`).
- `BoardState`: Ein Array von 14 Integern, das die Steine in den Mulden repräsentiert.
- `RowVersion`: Concurrency-Token (`Timestamp`), um gleichzeitige Änderungen am Spielzustand sicher zu handhaben.

---

5. Spiel-Logik (Kalaha-Regeln)

Die Spiel-Logik wird serverseitig validiert, um Manipulationen zu verhindern.

1. **Start:** Jede der 6 kleinen Mulden enthält 6 Steine. Die großen Kalahas (Mulden 6 und 13)

sind leer.
2. **Spielzug:** Ein Spieler wählt eine seiner Mulden und verteilt die Steine gegen den Uhrzeigersinn.
3. **Extrazug:** Landet der letzte Stein im eigenen Kalaha, darf der Spieler erneut ziehen.
4. **Schlagen:** Landet der letzte Stein in einer eigenen leeren Mulde und die gegenüberliegende Mulde des Gegners enthält Steine, werden alle diese Steine plus der letzte eigene Stein in das eigene Kalaha gelegt. Kein Extrazug.
5. **Ende:** Das Spiel endet, wenn ein Spieler keine Steine mehr in seinen kleinen Mulden hat. Die verbleibenden Steine des Gegners werden seinem Kalaha gutgeschrieben.

---

6. Deployment & Ausführung

Das gesamte System kann mit Docker Compose gestartet werden:

```
# Startet die Datenbank, 2 API-Instanzen und das Frontend
docker compose up -d --build --scale api=2
```

Die API ist dann (je nach Konfiguration) über den Load Balancer oder direkt erreichbar. Die Datenbank-Migrationen werden beim Start automatisch ausgeführt.