🎬 Watchlist

Eine benutzergebundene Watchlist für Filme und Serien mit JWT-Authentifizierung, Load Balancing und Containerisierung.

Stack

Schicht	Technologie
Frontend	React 18 + Vite 6 + React Router 7
Backend	Node.js 22 + Express 4 (3 Instanzen)
Datenbank	PostgreSQL 16
Proxy / LB	Nginx (least_conn + failover)
Auth	JWT (HS256, 15 min Access + 7d Refresh, OWASP)
Container	Docker + Docker Compose

Schnellstart

```
git clone <repo>
cd watchlist
docker compose up --build
```

- Frontend: http://localhost:5173
- API / Load Balancer: http://localhost:80
- Demo-Login: **demo** / **Demo1234!**

Kein manueller Setup nötig – DB-Schema und Seed-Daten werden automatisch beim ersten Start eingespielt.

Architektur

```
Browser
  │
  ├── :5173  → frontend (React SPA, nginx)
  │
  └── :80    → nginx (reverse proxy + load balancer)
               ├── least_conn
               ├── failover (max_fails=3, fail_timeout=30s)
               │
               ├── backend1:3000
               ├── backend2:3000
               └── backend3:3000
                        │
                   PostgreSQL:5432
```

JWT / OWASP

- Access Token: 15 min, HS256, `iss` + `aud` validiert, algorithm-pinning (verhindert alg:none)
- Refresh Token: opaque random (64 Bytes), SHA-256-Hash in DB, 7d, Rotation bei jeder Nutzung
- Passwort: bcrypt, cost=12
- Rate Limiting: 20 req/15min auf Auth-Endpoints
- Helmet Security Headers auf allen Responses

Datenmodell

```
users
  id, username, email, password (bcrypt), created_at

categories         genres
  Film               Action, Drama, Horror, SciFi, ...
  Serie              Action, Drama, Horror, SciFi, ...

watchlist_entries
  user_id → users
  category_id → categories
  genre_id → genres
  title, status (planned|watching|completed|dropped)
  rating (1-10, nullable), comment (text, nullable)

refresh_tokens
  user_id, token_hash (SHA-256), expires_at, revoked
```

Load Test

```
# hey installieren: brew install hey
chmod +x load-test.sh
./load-test.sh
```

Oder manuell: ```bash

Token holen

TOKEN=$(curl -s -X POST http://localhost/api/auth/login \ -H "Content-Type: application/json" \ -d '{"username":"demo","password":"Demo1234!"}' \ | grep -o '"accessToken":"[^"]*"' | cut -d"" -f4)

2000 Requests, 50 concurrent gegen JWT-geschützten Endpoint

hey -n 2000 -c 50 \ -H "Authorization: Bearer $TOKEN" \ http://localhost/api/watchlist ```

Erwartete Ergebnisse (auf lokalem Rechner)

Metrik	Typischer Wert
Requests/sec	~800–1500 rps
P50 Latenz	~15–30 ms
P99 Latenz	~80–150 ms
Bottleneck	PostgreSQL (Connection Pool)

Bottleneck-Analyse: Unter hoher Last ist PostgreSQL der Engpass, da jeder Request mindestens eine DB-Abfrage benötigt. Nginx + Node.js selbst skalieren gut horizontal – bei mehr Last einfach weitere Backend-Instanzen in `docker-compose.yml` ergänzen.

Failover demonstrieren

```
# Während des Load-Tests eine Instanz stoppen
docker compose stop backend2
# Nginx erkennt den Ausfall nach max 3 Fehlern und leitet automatisch um
```

Secrets

In Produktion `.env` anlegen: `JWT_SECRET=<mindestens 32 zufällige Zeichen>`

```
docker compose --env-file .env up
```