luga3 / **FitnessTracker**

Code Issues Pull requests Agents Actions Projects Security and quality Insight

Watch 0

0 stars 0 forks 0 watching Branches Activity
Tags

Private repository

1 Branch 0 Tags Go to file T Go to file Add file Code

luga3 fix 22265cb · 9 hours ago

backend	fix: persist login state across page r...	2 days ago
frontend	fix	9 hours ago
.env.example	readmee	last week
FitnessTracker.sln	wip	3 months ago
README.md	readmee	last week
docker-compose.yml	versionen	last week
hey.exe	load test with hey + fix	2 weeks ago
heycommand.txt	Update heycommand.txt	2 days ago
nginx.conf	fix	9 hours ago

README

FitnessTracker – Distributed System

1. Project Idea

FitnessTracker is a distributed web application to track workouts, exercises and sets.

Users can:

- Register and login securely
- Create, view and delete workouts
- Track exercises and sets (weight, reps) per workout
- View workout statistics

The goal is to build a scalable distributed system using Docker, load balancing, JWT authentication and a persistent database.

2. Architecture

System Overview

```
Browser
↓
Frontend (React, Port 3000)
↓
Nginx (Reverse Proxy / Load Balancer, Port 80)
↓
Backend1 (.NET 8 API, Port 5001)    Backend2 (.NET 8 API, Port 5002)
↓
PostgreSQL Database (Port 5432)
```

3. Components

Frontend

- React 19 application
- Served via Nginx container
- Communicates with backend via REST API
- JWT Access Token stored in memory (OWASP compliant)
- Automatic token refresh via httpOnly Cookie

Reverse Proxy / Load Balancer

- Nginx
- Distributes requests between backend1 and backend2 (Round-Robin)
- Enables horizontal scaling and failover

Backend

- .NET 8 Web API (2 instances)
- Entity Framework Core 8 (Code First)
- Automatic migrations at startup
- JWT authentication with Refresh Token rotation
- REST endpoints for workouts, exercises and sets
- Swagger documentation

Database

- PostgreSQL 16
- Running inside Docker

- Persistent volume (data survives restarts)
- Managed via EF Core migrations
- Seeded with demo user and 10 default exercises

4. Design Decisions

Why Docker?

- Reproducible environment
- Easy deployment with single command
- Clean separation of services

Why two backends?

- Demonstrates horizontal scaling
- Enables load balancing
- Failover when one instance goes down

Why Nginx?

- Reverse proxy and load balancer
- Production-like setup
- Dynamic DNS resolution for containers

Why JWT with Refresh Tokens?

- OWASP compliant authentication
- Access Token (15min) in memory — safe from XSS
- Refresh Token (7 days) in httpOnly Cookie — safe from JavaScript
- Token rotation on every refresh

5. How to Run

Setup

1. Clone the repository
2. Copy the environment file:

```
cp .env.example .env
```

3. Edit `.env` and set your secrets

Start

```
docker compose up --build
```

The system starts automatically including:

- Database migration
- Demo data seeding

Demo Account

Field	Value
Email	demo@demo.com
Password	Demo1234!

Endpoints

Service	URL
Frontend	http://localhost
Backend 1 Swagger	http://localhost:5001/swagger
Backend 2 Swagger	http://localhost:5002/swagger
Health Check	http://localhost/health
Instance Info	http://localhost/api/instance

6. Load Test

Load test against JWT-protected endpoint using hey:

```
.\hey.exe -n 1000 -c 50 -H "Authorization: Bearer <token>" http://localhost/api/workouts
```

Results

Metric	Value
Total Requests	1000
Requests/sec	680
Average Latency	66ms
Fastest	5ms
Slowest	542ms
Success Rate	100% (200 OK)

Bottleneck Analysis

The bottleneck is the PostgreSQL database (single instance). Under higher load the DB connection pool would be exhausted first.

Possible solutions:

- Read Replicas for read-heavy workloads
- PgBouncer (Connection Pooler)
- Redis Cache for frequently read data

7. Failover Demo

```
# Stop one backend instance
docker compose stop backend1

# System still works via backend2
curl http://localhost/api/instance

# Restart
docker compose start backend1
```

8. Data Model

```
User
 └── Workouts (1:n)
      └── WorkoutExercises (1:n)
           ├── Exercise (n:1)
           └── Sets (1:n)

User
 └── Exercises (1:n)

User
 └── RefreshTokens (1:n)
```

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 1

luga3

Languages

C# 65.8% **JavaScript** 30.5% **HTML** 2.5% **Other** 1.2%

Suggested workflows

Based on your tech stack

Deno

Test your Deno project

Configure

.NET

Build and test a .NET or ASP.NET Core project.

Configure

.NET Desktop

Build, test, sign and publish a desktop application built on .NET.

Configure

More workflows

Dismiss suggestions