Project information

Created on
March 01, 2026

Name	Last update
backend	20 hours ago
database	21 hours ago
frontend	19 hours ago
load-test	21 hours ago
nginx	20 hours ago
.gitignore	1 month ago
README.md	18 hours ago
docker-compose.yml	20 hours ago

README.md

Distributed TCG Deck Builder – FS 2026 Challenge Task

Quick Start

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

Open in browser: http://localhost:8080

> No manual setup required. Database schema is applied automatically on first start.

1. Project Overview

A distributed web application for building and managing Pokémon TCG decks.

Users register, log in, search real cards via the official Pokémon TCG API, and save decks to a persistent database. The system runs across two stateless backend instances behind a load balancer — if one instance is shut down, the application continues without interruption.

2. System Architecture

```
Browser
   │
   ▼
Nginx (Load Balancer, port 8080)
   │
   ├──► Backend Instance 1 (Node.js / Express)
   │                  │
   └──► Backend Instance 2 (Node.js / Express)
                      │
```

```
                    ▼
          PostgreSQL (persistent volume)
```

- **Frontend** (React + Vite) is served through Nginx
- **Backend** instances are stateless — no session data stored in memory
- **JWT** is stored in an httpOnly cookie, verified on every protected request
- Any backend instance can serve any request (shared database)
- If one backend stops, Nginx routes all traffic to the other automatically

---

3. Technology Stack

Layer	Technology	Version
Frontend	React + Vite	React 19.1, Vite 7.2
Backend	Node.js + Express	Node 24, Express 5
Auth	JWT + bcrypt	jsonwebtoken 9, bcrypt 5
Database	PostgreSQL	18-alpine
Load Balancer	Nginx	1.29
Containerization	Docker + Docker Compose	latest
Load Testing	wrk	latest (via Docker)
Card Data	Pokémon TCG API	api.pokemontcg.io/v2 (free, no key)

---

4. Features

Authentication (OWASP-compliant)

- Register and login with username + password
- Passwords hashed with bcrypt (10 salt rounds)
- JWT stored in httpOnly, SameSite=Lax cookie (XSS-resistant)
- Refresh tokens: long-lived token (7 days) stored in DB and httpOnly cookie — silently renews the short-lived JWT without re-login
- Token rotation: each refresh invalidates the old refresh token and issues a new one (stolen tokens become useless after next use)
- Logout deletes the refresh token from the DB, immediately invalidating it server-side
- Rate limiting: 20 requests/min on auth endpoints, 100/min on API
- All deck and card endpoints require a valid JWT

Deck Management

- Create and delete decks
- Each deck belongs to the authenticated user (ownership enforced server-side)

Card Search & Deck Building

- Search Pokémon cards by name (proxied through the backend to avoid CORS)
- Paginated results (12 cards per page)
- Add individual cards or bulk-add by pasting a list of card names
- Adjust quantity per card (1–4, enforced by TCG rules)
- Remove cards from deck

---

5. Requirement Mapping

Requirement	Implementation	Status
Simple frontend	React SPA	✅
Load balancer	Nginx round-robin	✅
Two service instances	`backend1` + `backend2` containers	✅

Requirement	Implementation	Status
JWT authentication (OWASP)	httpOnly cookie, bcrypt, rate limiting	✅
Persistent storage	PostgreSQL with named Docker volume	✅
Containerized	Docker Compose, all services	✅
One-command startup	`docker compose up --build`	✅
Failover	Stop one container → other continues	✅
Load test with documentation	wrk, see `load-test/RESULTS.md`	✅

---

6. API Endpoints

Authentication

Method	Path	Auth	Description
POST	`/api/auth/register`	—	Create account
POST	`/api/auth/login`	—	Login, sets JWT + refresh token cookie
POST	`/api/auth/refresh`	—	Issue new JWT using refresh token (token rotation)
POST	`/api/auth/logout`	—	Clear cookies, invalidate refresh token in DB
GET	`/api/auth/me`	✅	Current user info

Decks

Method	Path	Auth	Description
GET	`/api/decks`	✅	List user's decks
POST	`/api/decks`	✅	Create deck
DELETE	`/api/decks/:id`	✅	Delete deck

Deck Cards

Method	Path	Auth	Description
GET	`/api/decks/:id/cards`	✅	List cards in deck
POST	`/api/decks/:id/cards`	✅	Add card (UPSERT, max qty 4)
PATCH	`/api/decks/:deckId/cards/:cardId`	✅	Update quantity
DELETE	`/api/decks/:deckId/cards/:cardId`	✅	Remove card

Utility

Method	Path	Auth	Description
GET	`/api/health`	—	Health check, returns instance name
GET	`/api/cards/search`	✅	Proxy to Pokémon TCG API

---

7. Failover Demonstration

To simulate a backend failure during a demo:

```
docker stop tcg-backend-1
```

The application continues running via `backend-2`. To restore:

```
docker start tcg-backend-1
```

The `instance` field in all API responses shows which backend is currently serving requests.

---

8. Load Testing

Full results and analysis: `load-test/RESULTS.md`

Tested endpoint: `GET /api/decks` — JWT-protected, performs a database read on every request.

Run the load test

```
# 1. Get JWT
curl -c cookies.txt -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"your_user","password":"your_password"}'
JWT=$(grep token cookies.txt | awk '{print $7}')

# 2. Run wrk
docker run --rm --network host williamyeh/wrk \
  -t 4 -c 100 -d 30s \
  -H "Cookie: token=$JWT" \
  http://localhost:8080/api/decks
```

Key Results (2026-05-17, AMD Ryzen 5 4500U, 16 GB RAM)

Metric	Value
Throughput	**1,581 req/s**
Avg latency	105.23ms
Max latency	1.49s
Total requests	47,558 in 30s
Transfer	2.72 MB/s
Error rate	0.00% ✅

Bottleneck: PostgreSQL

The system sustains over 1,500 req/s with zero errors under 100 concurrent connections. At higher concurrency, the bottleneck is **PostgreSQL** — a single shared instance with no read replicas. Nginx and Node.js have significant remaining headroom.

---

9. Secrets Management

The `.env` file is excluded from version control (see `.gitignore`). Copy the example file and fill in your own values:

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

Variable	Description
`POSTGRES_DB`	Database name
`POSTGRES_USER`	Database user
`POSTGRES_PASSWORD`	Database password
`JWT_SECRET`	Secret for signing JWTs (use a long random string)
`FRONTEND_ORIGIN`	Allowed CORS origin (default: `http://localhost:8080`)

---

10. Current Status (Milestone 3 — 2026-05-03)

Phase	Status
Project setup (Docker, Nginx, DB)	✅ Complete
Backend (JWT auth, deck/card CRUD)	✅ Complete
Frontend (React, deck builder, card search)	✅ Complete
Distributed setup (2 instances, failover)	✅ Complete
Load testing + documentation	✅ Complete

Still open for final hand-in (2026-05-18):

- Presentation slides (architecture, demo, Q&A prep)
- Final README polish after any last changes