CLAUDE.md

This file provides guidance to Claude Code (claude.ai/code) when working with code in this repository.

Project Overview

DSYS is a distributed real-time competitive math game. Players solve math exercises to claim cells on a grid, earning points based on difficulty. The system demonstrates load balancing and fault tolerance with two backend instances behind a Traefik reverse proxy.

Architecture

```
Browser → Traefik (:80) → backend1/backend2 (:3000) → PostgreSQL (:5432)
                        → frontend (Nginx :80)
```

- **Traefik** routes `/api` and `/ws` to backend, `/` to frontend. Sticky sessions via cookie. Health checks every 5s with automatic failover.
- **Backend** (x2 instances): Express 5 + TypeScript on Node 22. Stateless JWT auth. WebSocket broadcasts cell claims to all clients.
- **Frontend**: React 19 + Vite + React Router 7. Served by Nginx in production. Inline styles only (no CSS framework).
- **Database**: PostgreSQL 17. Schema in `db/init.sql`. Only one active game at a time (enforced by partial unique index).

Commands

Full stack (Docker Compose)

```
docker compose up             # Start everything at http://localhost
docker compose up --build     # Rebuild and start
docker compose down -v        # Tear down including database volume
```

Backend (backend/)

```
npm install
npm run dev         # tsx watch with hot reload
npm run build       # TypeScript compile to dist/
npm start           # Run compiled JS (production)
```

Frontend (frontend/)

```
npm install
npm run dev         # Vite dev server (proxies /api and /ws to localhost:80)
npm run build       # TypeScript check + Vite build to dist/
```

```
npm run preview      # Preview production build
```

Database

Schema changes require recreating the volume: `docker compose down -v && docker compose up`

Key Design Decisions

- **Exercise storage is in-memory** per backend instance (Map keyed by UUID, 5-min TTL). This means exercises are not shared across instances — sticky sessions are required.
- **Cell claiming uses optimistic concurrency**: the UPDATE includes `WHERE claimed_by IS NULL` to handle race conditions at the DB level.
- **Game creation is transactional** (BEGIN/COMMIT/ROLLBACK) — deactivates current game and creates all cells atomically.
- **No test framework is configured.** Neither backend nor frontend has test scripts or testing dependencies.
- **No linter is configured.**
- **Admin routes are protected** — only users with `is_admin = TRUE` can create games. Seeded admin user: `admin`/`admin`.
- **Cross-instance WebSocket** uses PostgreSQL LISTEN/NOTIFY so cell claims broadcast to all clients across all backend instances.

API Routes

All backend routes are prefixed with `/api`:

- `POST /api/auth/register` and `/api/auth/login` — return JWT (access + refresh token)
- `POST /api/auth/refresh` — exchange refresh token for new access + refresh token pair
- `GET /api/game` — active game state with cells (protected)
- `GET /api/game/cell/:cellId/exercise` — generate exercise (protected)
- `POST /api/game/cell/:cellId/answer` — submit answer, claims cell if correct (protected)
- `GET /api/highscores` — public leaderboard
- `POST /api/admin/games` — create new game (admin only)
- `GET /api/health` — health check (used by Traefik)
- `WS /ws` — real-time cell claim broadcasts

Environment Variables

Secrets are managed via `.env` file (gitignored). See `.env.example` for required variables.

Variable	Used By	Description
POSTGRES_USER	db, backend	PostgreSQL username
POSTGRES_PASSWORD	db, backend	PostgreSQL password
POSTGRES_DB	db, backend	PostgreSQL database name
JWT_SECRET	backend	Secret for signing JWTs
INSTANCE_ID	backend	Set per-instance in docker-compose.yml

Database Schema

Four tables: `users`, `games`, `cells`, `scores`. Key constraints:

- Partial unique index on `games(active) WHERE active = TRUE` — enforces single active game
- `cells(game_id, row_index, col_index)` is unique
- `scores(user_id, game_id)` is unique (for upserts)
- Cells cascade delete with their game