developerAurel / **Multiplayer_Quiz**

Code Issues Pull requests Agents Actions Projects Security and quality Insights Settings

Watch 0

Challenge Task for Distributed Systems

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

developerAurel updated read me 8f06fd7 · now

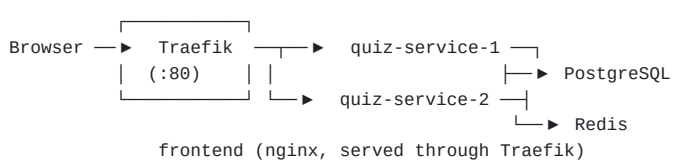
backend	new quiz and version updates	2 minutes ago
docs	updated stability/UI/Gamehandler	last week
frontend	reconnect to other gameserver seamless	10 hours ago
loadtest	bigfix session	last week
.env.example	frontend	2 months ago
.gitignore	cors	3 weeks ago
README.md	updated read me	now
docker-compose.yml	new quiz and version updates	2 minutes ago
start.sh	start skript	2 months ago

README

Multiplayer Quiz

Distributed real-time quiz game. OST FS26 Distributed Systems Challenge Task.

Architecture



- **Frontend:** React 19 + Vite + Tailwind, served via nginx behind Traefik
- **Load balancer:** Traefik v3.3 — healthcheck-driven failover, sticky cookie for Socket.io
- **Backend:** 2 × Node 22 (Express 5 + Socket.io 4)
- **State:**
  - PostgreSQL 17 — users, refresh tokens, quizzes, results (durable)
  - Redis 7 — room state, distributed lock, scheduled-tick queue, Socket.io pub/sub
- **Auth:** 15-min JWT (HS256) + 7-d refresh token (rotated, `httpOnly` `SameSite=Strict` cookie, SHA-256 hashed in DB)

Detailed walk-through: docs/ARCHITECTURE.md. File-by-file map: docs/STRUCTURE.md.

Quick start

```
cp .env.example .env
# set JWT_SECRET to a 64-char hex value, e.g. `openssl rand -hex 32`
docker compose up --build
```

Open http://localhost. Register, host or join a room with the 6-char code, play.

The Postgres database is seeded automatically from `backend/src/db/init.sql` on first boot (4 quizzes, 26 questions — including a hard-mode blockchain quiz).

```
./start.sh    # full reset: down -v && up --build
```

Failover demo

Two backend instances run in parallel. Traefik health-checks `/api/health` every 2 s and removes unhealthy servers. Game state lives in Redis (with a `phaseEndsAt` timestamp + 250 ms sweeper + per-room lock), so the surviving instance picks up where the dead one left off. The frontend re-joins the room on every Socket.io `connect` event — auto-reconnect is transparent to the player, no F5 needed.

```
bash loadtest/failover-demo.sh
# Drops quiz-service-2 mid-load-test; load test continues with 99.4 % success.
```

Manual demo: start a game with two browsers, then in a second terminal:

```
docker stop quiz-service-2     # game continues, surviving instance advances phases
docker start quiz-service-2    # back in the pool ~5 s later
```

Load test

```
docker compose up -d --build
bash loadtest/loadtest.sh             # 4 scaling tests
bash loadtest/saturation-test.sh      # concurrency ramp-up to find the breaking point
```

Results: see loadtest/LOADTEST.md. Peak **3,640 req/s** at 200 concurrent connections on a JWT-protected endpoint, p99 < 500 ms, 100 % success. Bottleneck is the Postgres connection pool — CPU stays around 30 %.

Tests

Backend unit tests (validators, JWT middleware, env-var fail-fast, game state machine):

```
cd backend && npm install && npm test
```

38 tests cover password/email/username validation, room-code generation, the phase-ripeness check, points calculation, and the boot-time env validation.

End-to-end multiplayer smoke test (two headless Socket.io clients play a full round):

```
docker run --rm --network project_quiz-net \
  -v "$(pwd -W)/loadtest:/app" -w /app node:22-alpine \
  sh -c 'npm install --no-save socket.io-client@4 --silent && node multiplayer-smoke.js'
```

Asserts: room creation, second player join, all 7 questions emitted to both clients, points awarded, `game-over` fires with both scores. Completes in ~26 s.

Security notes

OWASP ASVS Level 1 mapping in docs/OWASP_CHECKLIST.md. Highlights:

- Passwords bcrypt-hashed at cost 12; policy ≥8 chars + upper / lower / digit
- IP rate limit (10 / 15 min) on `/api/auth/login` and `/api/auth/register`
- Per-account lockout: 5 failed attempts → 15 min lock
- Constant-time bcrypt compare against a dummy hash for unknown users (anti-enumeration)
- Refresh-token rotation: old token deleted on use (`DELETE ... RETURNING` + new `INSERT`); token-reuse detection without extra tracking
- JWT enforced on the Socket.io handshake
- `helmet` on the API, CSP + X-Frame-Options + Permissions-Policy etc. on the nginx layer
- `JWT_SECRET` must be ≥ 32 chars and not the example value — boot fails otherwise
- Graceful shutdown on `SIGTERM` so Traefik gets a clean drain signal

Documentation

File	Purpose
docs/ARCHITECTURE.md	High-level architecture, data flows, design decisions
docs/STRUCTURE.md	File-by-file map of the repository
docs/REQUIREMENTS_AUDIT.md	Mapping of all 8 challenge-task requirements with file:line citations
docs/OWASP_CHECKLIST.md	OWASP ASVS Level 1 compliance mapping
docs/VERIFICATION.md	5-minute smoke-test runbook
docs/PRESENTATION.md	Slide outline for the 5+5+5-min presentation
docs/COMMANDS.md	All commands grouped by use case (boot, fresh-clone, failover, loadtest, …)
docs/MERMAID_DIAGRAMS.md	Five Mermaid diagrams (topology, auth flow, game state machine, failover sequence, tech stack)
docs/mermaid/	Standalone `.mmd` files paste-ready for mermaid.live
docs/notebooklm/	Six German explainer scripts for NotebookLM audio generation
loadtest/LOADTEST.md	Measured throughput numbers and bottleneck analysis

Project structure

```
backend/
  src/
    server.js             # express app + helmet + pino + graceful shutdown
    config.js             # env validation + service-wide constants
    logger.js             # pino instance
    validators.js         # username/email/password validators (unit-tested)
    routes/auth.js        # register, login (with lockout), refresh, logout
    routes/quizzes.js     # JWT-protected list, history, detail
    middleware/auth.js    # JWT verify
    socket/index.js       # Socket.io with Redis adapter + JWT handshake
    socket/gameManager.js # game flow, Redis state, claim lock, rejoin-room
    socket/sweeper.js     # 250-ms tick that advances overdue rooms
    socket/gameLogic.js   # pure helpers (room code, points, phase check)
    db/index.js           # pg Pool
    db/init.sql           # schema + seed (4 quizzes, 26 questions)
  test/                   # vitest unit tests
frontend/
  src/
    App.jsx
    context/AuthContext.jsx     # access token + auto-refresh + concurrent-refresh guard
    context/SocketContext.jsx   # Socket.io client + connection status + reconnect
    components/ConnectionStatus.jsx
    pages/                      # Login, Register, Lobby, Game, Results
docker-compose.yml              # traefik + postgres + redis + 2× backend + frontend
docs/                           # ARCHITECTURE, STRUCTURE, OWASP, REQUIREMENTS_AUDIT, ...
loadtest/
  loadtest.sh                   # hey-driven scaling tests
  failover-demo.sh              # kill an instance mid-test
  saturation-test.sh            # concurrency ramp-up to the breaking point
  multiplayer-smoke.js          # end-to-end 2-player headless round
  LOADTEST.md                   # measured numbers + bottleneck analysis
```

Releases

No releases p
Create a new

Packages

No packages
Publish your f

Contribut

devel

Language

JavaScrip

Suggested workflows

Based on your tech stack

Publish Node.js Package — Configure

Publishes a Node.js package to npm.

SLSA Generic generator — Configure

Generate SLSA3 provenance for your existing release workflows

Webpack — Configure

Build a NodeJS project with npm and webpack.

More workflows

Dismiss suggestions