Dokumentacja Techniczna Systemu Quizyx

Zespół Deweloperski

22 stycznia 2026

Spis treści

1 Wstęp

Niniejsza dokumentacja techniczna przeznaczona jest dla deweloperów oraz administratorów systemu. Zawiera ona szczegóły implementacyjne warstwy backendowej i frontendowej, schemat bazy danych, opis kluczowych modeli, specyfikację endpointów REST API oraz mechanizmów komunikacji w czasie rzeczywistym (Socket.io).

2 Struktura katalogów

2.1 Backend (`backend/`)

- `server.ts` – Główny plik serwera (Express + Socket.io). Zawiera definicje endpointów, middleware autoryzacyjne i logikę realtime.
- `models.ts` – Definicje schematów Mongoose (`User`, `Quiz`, `Result`, `Advertisement`).
- `package.json` – Skrypty uruchomieniowe i zależności backendu.
- `tsconfig.json` – Konfiguracja kompilatora TypeScript.

2.2 Frontend (`frontend/`)

- `src/` – Źródła aplikacji React + TypeScript.
- `src/pages/` – Główne widoki aplikacji:
  - `AdminDashboard.tsx` – Panel administratora.
  - `Leaderboard.tsx` – Tablica wyników.
  - `Profile.tsx` – Profil użytkownika.
  - `QuizRoom.tsx` – Ekran rozgrywki.
  - `Social.tsx` – Widok społecznościowy.
- `src/components/` – Komponenty UI (np. `Navbar.tsx`, `AdComponents.tsx`).
- `public/` – Zasoby statyczne (awatary, grafiki reklamowe).
- `vite.config.ts` – Konfiguracja bundlera Vite.

3 Technologie i zależności

System został zbudowany w oparciu o nowoczesny stos technologiczny JavaScript/TypeScript:

- **Backend:** Node.js, Express, TypeScript.
  - Narzędzia dev: `ts-node`, `nodemon`.
  - Kluczowe biblioteki: `mongoose`, `socket.io`, `jsonwebtoken`, `bcryptjs`.
- **Frontend:** React, Vite, TypeScript.
  - Stylowanie: TailwindCSS (opcjonalnie).
  - Komunikacja: `axios`, `socket.io-client`.
- **Baza danych:** MongoDB (instancja lokalna lub MongoDB Atlas).

4 Modele bazy danych

Poniżej przedstawiono szczegółową strukturę kolekcji w bazie MongoDB.

4.1 Kolekcja `User`

Przechowuje dane użytkowników i ich statystyki.

- `username`: String (unikalny, wymagany).
- `email`: String (unikalny, wymagany).
- `password`: String (hash, wymagany).
- `avatarUrl`: String (domyślnie URL do DiceBear).
- `role`: String (enum: `'user'`, `'admin'`).
- `lastUsernameChange`: Date.
- `history`: Tablica `ObjectId` (ref: `Result`).
- `winstreak`: Number.
- `maxWinstreak`: Number.
- `friends`: Tablica `ObjectId` (ref: `User`).
- `friendRequests`: Tablica `ObjectId` (ref: `User`).

4.2 Kolekcja `Quiz`

Główny model quizu zawierający zagnieżdżone pytania.

- `title`, `description`: String.
- `difficulty`: Enum (`easy`, `medium`, `hard`).
- `type`: String (`standard`, `exam`, `duel`).
- `author`: `ObjectId` (ref: `User`).
- `timeLimit`: Number (czas w sekundach).
- **questions** (Tablica obiektów):
  - `content`: Treść pytania.
  - `answers`: Tablica odpowiedzi (String).
  - `correctAnswers`: Tablica indeksów poprawnych odpowiedzi (Number).
  - `type`: `'single'` lub `'multi'`.

4.3 Kolekcja `Result`

Zapisuje wyniki poszczególnych podejść do quizów.

- `userId`: `ObjectId` (ref: `User`).
- `quizId`: `ObjectId` (ref: `Quiz`).
- `quizType`, `score`, `maxScore`.
- `timeSpent`, `date`.
- `userWinstreak`: Stan passy w momencie zakończenia gry.

4.4 Kolekcja `Advertisement`

Model zarządzania reklamami w systemie.

- `title`, `content`: Treść reklamy.
- `location`: Enum (`home_top`, `quiz_sidebar`, `popup`, `fullscreen`).
- `triggerType`, `triggerValue`, `priority`, `active`.

5 Autoryzacja i zabezpieczenia

System wykorzystuje standard JSON Web Token (JWT) do autoryzacji bezstanowej.

1. **Tokeny:** Podpisywane kluczem `JWT_SECRET`. Przekazywane w nagłówku HTTP: `Authorization: Bearer <token>`.
2. **Middleware `authenticateJWT`:** Weryfikuje podpis tokenu i dołącza obiekt użytkownika do `req.user`.
3. **Middleware `isAdmin`:** Weryfikuje, czy `req.user.role === 'admin'`.
4. **Hasła:** Haszowane przy użyciu biblioteki `bcryptjs` przed zapisem do bazy.

Rekomendacje rozwojowe: Wdrożenie rotacji kluczy JWT, mechanizmu *Refresh Tokens* oraz walidacji danych wejściowych (np. `express-validator`).

6 Specyfikacja REST API

Większość endpointów zwraca obiekty Mongoose, często z wykorzystaniem metody `populate` do pobierania relacji.

6.1 Autoryzacja

POST /api/register Rejestracja nowego użytkownika.

```
Body: { "username": "jan", "email": "a@b.pl", "password": "pass" }
Response (201): {
  "token": "...",
  "user": { "_id": "...", "username": "...", "role": "user" }
}
```

POST /api/login Logowanie użytkownika.

GET /api/auth/me Pobranie danych zalogowanego użytkownika (wymaga tokenu). Zwraca pełny obiekt `User` z historią gier.

6.2 Quizy i Wyniki

GET /api/quizzes Lista dostępnych quizów.

POST /api/quizzes (Admin) Dodawanie nowego quizu.

POST /api/quizzes/import (Admin) Masowy import quizów (tablica JSON).

GET /api/leaderboard Pobiera ranking najlepszych wyników.

POST /api/results Zapis wyniku gry.

```
Body: {
  "quizId": "...", "quizType": "standard",
  "score": 10, "maxScore": 15, "won": true
}
```

7 Socket.io – Komunikacja Realtime

Moduł WebSocket wykorzystywany jest do trybu pojedynków (Duel) oraz wyzwań.

7.1 Główne zdarzenia (Events)

- `register_user`: Klient wysyła `userId` zaraz po połączeniu. Serwer mapuje `userId` na `socketId`.
- `challenge_friend`: Inicjacja wyzwania.

```
Payload: {
  "friendId": "<target_id>",
  "quizId": "<quiz_id>",
  "challengerName": "Jan",
  "challengerAvatar": "https://..."
}
```

- `incoming_challenge`: Zdarzenie odbierane przez wyzwanego gracza.
- `accept_challenge`: Akceptacja wyzwania. Serwer tworzy pokój gry i emituje `match_found`.
- `join_duel`: Dołączenie do kolejki losowych pojedynków. Serwer paruje oczekujących graczy.

- `send_progress`: Wysyłanie aktualnego wyniku w trakcie gry. Serwer przekazuje to do przeciwnika jako `opponent_update`.
- `finish_duel`: Zakończenie gry przez jednego z graczy.

8 Testy i CI

Zalecane środowisko testowe:

- **Unit Testy:** Jest.
- **Integration Testy:** Supertest (dla endpointów Express).
- **Mockowanie:** msw lub nock dla zewnętrznych API.

Zaleca się konfigurację potoku CI (GitHub Actions/GitLab CI) z etapami: `install` → `build` → `test` → `deploy`.

9 Architektura i Rozszerzalność

Sugestie dotyczące rozwoju kodu (Refactoring):

1. Separacja odpowiedzialności: Wydzielenie logiki z `server.ts` do dedykowanych kontrolerów (katalog `controllers/`) i tras (`routes/`).
2. Warstwa serwisów: Implementacja `UserService`, `QuizService` do obsługi logiki biznesowej niezależnej od HTTP.
3. Walidacja DTO: Użycie bibliotek takich jak `Zod` lub `Joi` do ścisłej walidacji danych wejściowych.

10 Uruchamianie i Debugowanie

- **Backend:** `npm run dev` uruchamia serwer z `nodemon`. Debugowanie możliwe poprzez *VS Code Attach to Node Process*.
- **Frontend:** `npm run dev` (Vite) zapewnia Hot Module Replacement (HMR).
- **Dokumentacja API:** Możliwość wygenerowania `openapi.yaml` i hostowania Swagger UI pod adresem `/api/docs`.