Serwer Smart Dartboard

Produkcyjny serwer aplikacji Smart Dartboard z integracją Arduino.

Funkcjonalności

- **Zarządzanie grą w czasie rzeczywistym** via Socket.IO
- **Integracja dartboardu Arduino** via port szeregowy
- **Uwierzytelnianie JWT** dla bezpiecznych sesji użytkownika
- **Wiele trybów gry**: 501, 301, Cricket, Killer
- **Śledzenie statystyk graczy**
- **System lobbyów** z zarządzaniem gospodarzem

Stos techniczny

- **Runtime**: Node.js
- **Framework**: Express 5
- **Czas rzeczywisty**: Socket.IO
- **Uwierzytelnianie**: JWT + bcryptjs
- **Sprzęt**: SerialPort (Arduino)
- **Baza danych**: Magazyn pliku JSON

Wymagania

- Node.js 18+
- Dartboard Arduino (opcjonalnie - serwer działa bez niego)

Instalacja

```
npm install
```

Konfiguracja

Zmienne środowiskowe (opcjonalnie):

Zmienna	Domyślnie	Opis
`PORT`	`3001`	Port serwera
`JWT_SECRET`	`smart-dartboard-secret-key-2024`	Klucz podpisania JWT
`SERIAL_PORT_PATH`	`/dev/ttyACM0`	Port szeregowy Arduino
`SERIAL_BAUD_RATE`	`115200`	Szybkość transmisji szeregowej

Uruchomienie

Produkcja

```
node server.js
```

Serwer będzie dostępny pod adresem:

- **Aplikacja**: `http://localhost:3001`
- **API**: `http://localhost:3001/api`
- **WebSocket**: `ws://localhost:3001`

Z PM2 (rekomendowane)

```
pm2 start server.js --name dartboard
```

Punkty końcowe API

Uwierzytelnianie

Metoda	Endpoint	Opis
POST	`/api/auth/register`	Zarejestruj nowego użytkownika
POST	`/api/auth/login`	Zaloguj użytkownika
GET	`/api/auth/me`	Pobierz bieżącego użytkownika

Profil

Metoda	Endpoint	Opis
PUT	`/api/profile/avatar`	Aktualizuj awatar
GET	`/api/profile/avatars`	Pobierz dostępne awatary
PUT	`/api/profile/username`	Zmień nazwę użytkownika
PUT	`/api/profile/password`	Zmień hasło

Lobby

Metoda	Endpoint	Opis
GET	`/api/lobbies`	Wyświetl wszystkie lobby
POST	`/api/lobbies`	Utwórz lobby
GET	`/api/lobbies/:id`	Pobierz szczegóły lobby
POST	`/api/lobbies/:id/join`	Dołącz do lobby

Metoda	Endpoint	Opis
POST	`/api/lobbies/:id/leave`	Opuść lobby
PUT	`/api/lobbies/:id/mode`	Ustaw tryb gry
POST	`/api/lobbies/:id/start`	Uruchom grę
POST	`/api/lobbies/:id/end`	Zakończ grę
POST	`/api/lobbies/:id/abort`	Przerwij grę
POST	`/api/lobbies/:id/undo-throw`	Cofnij ostatni rzut

Gra

Metoda	Endpoint	Opis
GET	`/api/game/can-start`	Sprawdź dostępność dartboardu
GET	`/api/game/dartboard/status`	Pobierz stan połączenia dartboardu

Zdarzenia Socket.IO

Klient → Serwer

Zdarzenie	Ładunek	Opis
`join_lobby`	`lobbyId: string`	Dołącz do pokoju lobby
`leave_lobby`	`lobbyId: string`	Opuść pokój lobby

Serwer → Klient

Zdarzenie	Ładunek	Opis
`lobby_update`	`Lobby`	Stan lobby zmienił się
`game_update`	`GameState`	Stan gry zmienił się
`game_started`	`GameState`	Gra się zaczęła
`game_ended`	-	Gra zakończyła się normalnie
`game_aborted`	`{ abortedBy: string }`	Gra przerwana
`host_changed`	`{ newHostId, newHostName }`	Host zmienił się
`lobby_deleted`	-	Lobby zostało usunięte
`dartboard_status`	`{ connected: boolean }`	Stan połączenia dartboardu

Protokół Arduino

Serwer oczekuje wiadomości JSON z Arduino poprzez port szeregowy:

```
{
  "event": "hit",
  "sector": 20,
  "multiplier": 3,
  "score": 60
}
```

Pola

Pole	Typ	Opis
`event`	string	Typ zdarzenia (`hit`)
`sector`	number	Sektor dartboardu (1-20, 25 dla bull)
`multiplier`	number	1 = pojedyncze, 2 = podwójne, 3 = potrójne
`score`	number	Obliczony wynik (sektor × mnożnik)

Baza danych

Dane przechowywane są w `database.json`:

```
{
  "users": [],
  "lobbies": [],
  "activeGame": null
}
```

Baza danych jest automatycznie tworzona przy pierwszym uruchomieniu.

Struktura projektu

```
backend-emulator/
├── server.js        # Główny plik serwera
├── database.json    # Magazyn danych
├── package.json     # Zależności
├── public/          # Kompilacja frontendu (z dart-app)
└── README.md        # Ten plik
```

Programowanie

Symuluj rzut (bez Arduino)

```
curl -X POST http://localhost:3001/api/lobbies/{id}/simulate-throw \
  -H "Authorization: Bearer {token}"
```

Licencja

MIT