

Audyt produkcyjny Ping-Pong 1v1 (cel: 200 tys. graczy)

Data: 2026-05-13

Zakres: public_html/disciplines/ping-pong/1v1, public_html/api/matches/ping-pong/1v1, public_html/cron

Metoda: statyczny przegląd kodu + przegląd architektury i wdrożenia

Werdykt wykonawczy

Obecna implementacja **NIE JEST** gotowa na obsługę 200 tys. graczy na produkcji.

Szacowany bezpieczny zakres (bez większego redesignu):

- kilkanaście tysięcy zarejestrowanych użytkowników,
- kilka tysięcy jednocześnie zalogowanych użytkowników (zależnie od wydajności Redis/MySQL i infrastruktury).

Głównymi blokadami są skalowalność matchmakingu oraz obciążenie zapisami bazy MySQL, a także ryzyko niespójności nagród między różnymi ścieżkami wykonania.

PO (krytyczne przed wdrożeniem)

PO-1: Niespójność nagród między ścieżką Node a fallbackiem PHP

Wpływ:

- zwycięzcy mogą otrzymać różne wypłaty w zależności od ścieżki procesowania,
- niespójna ekonomia/księgowość, ryzyko sporów, obciążenie supportu.

Dowody:

- Bezpośrednia ścieżka Node nagradza zwycięzcę kwotą **0.80**:
.../node-server/src/mysqlWriter.js:130
- Fallback PHP nagradza zwycięzcę kwotą **1.00**:
.../api/matches/ping-pong/1v1/index.php:157

Dlaczego to krytyczne:

- ten sam typ meczu nigdy nie powinien generować różnych wypłat w zależności od gałęzi wykonawczej.

Wymagana poprawka:

- scentralizowanie stałych dotyczących nagród i wymuszenie jednego "źródła prawdy" (single source of truth).

P0-2: Algorytm matchmakingu to $O(n)$ na pełnej kolejce w każdym cyklu + blokada globalna

Wpływ:

- skoki opóźnień (latency spikes) kolejki pod wysokim obciążeniem,
- ryzyko całkowitego załamania przepustowości w szczytach.

Dowody:

- skanowanie całej kolejki przez `zRange(0, -1)`: `.../node-server/src/matchmaking.js:27`
- globalna blokada (lock) dla kolejki: `.../node-server/src/matchmaking.js:24`
- pętla matchmakingu co 150ms: `.../node-server/src/index.js:729`

Dlaczego to krytyczne:

- przy dużych rozmiarach kolejki tworzy to niepotrzebne "gorące punkty" (hot spots) i konflikty blokad.

Wymagana poprawka:

- przejście na kolejki segmentowane (bucketed/sharded) lub atomowy skrypt `dequeue` po stronie serwera,
- unikanie pełnych skanów `zRange` przy każdej iteracji parowania.

P0-3: Tryb klastra polega na Redis; fallback do pamięci RAM degradowuje semantykę

Wpływ:

- dane o routingu między workerami i własności meczów mogą przestać działać poprawnie,
- bezpieczeństwo ponownych połączeń i spójność kolejki w skali klastra zostaje naruszone.

Dowody:

- Maksymalna liczba workerów klastra PM2: `.../node-server/ecosystem.config.cjs:14`
- Operacje IPC typu "no-op" w trybie pamięci lokalnej: `.../node-server/src/ipc.js:37`
- Fallback Redisa do klienta in-memory: `.../node-server/src/redisClient.js:90`

Dlaczego to krytyczne:

- produkcja z wieloma workerami bez twardej zależności od Redis tworzy zachowanie typu

"split-brain".

Wymagana poprawka:

- wymuszenie Redis jako twardego wymagania w trybie klastra,
- blokada startu aplikacji, jeśli Redis jest niedostępny w klastrze.

P0-4: Brak automatycznych testów dla ścieżki serwera gry

Wpływ:

- regresje w fizyce, ponownych połączeniach czy idempotencji nagród mogą trafić na produkcję niezauważone.

Dowody:

- brak plików testowych w node-server,
- skrypty package.json nie zawierają komendy testowej: .../node-server/package.json:7

Dlaczego to krytyczne:

- gra czasu rzeczywistego o wysokiej współbieżności bez testów to ogromne ryzyko przy rolloutcie.

Wymagana poprawka:

- dodanie testów jednostkowych, integracyjnych oraz bramek wydajnościowych (load gates) w CI przed startem.

P1 (wysoki priorytet)

P1-1: Wysokie obciążenie zapisami MySQL przez okresowe aktualizacje meczów

Wpływ:

- nasycenie bazy danych przy dużej liczbie graczy,
- nieaktualne stany / opóźnione zapisy / wzrost kolejki połączeń (pool queue).

Dowody:

- ścieżka aktualizacji co sekundę: .../node-server/src/index.js:538
- opcjonalne inserty match_ticks w tej samej ścieżce: .../node-server/src/index.js:545
- domyślny limit połączeń ustawiony na 20: .../node-server/src/config.js:33

Wymagana poprawka:

- zmniejszenie częstotliwości zapisu lub grupowanie (batching) zapisów,
- ustawienie match_ticks jako opcjonalne (opt-in) na produkcji,
- zwiększenie puli i wydajności bazy danych zgodnie z przetestowanymi SLO.

P1-2: Operacje DDL w krytycznej ścieżce przyznawania nagród

Wpływ:

- niepotrzebny narzut i ryzyko blokad podczas ruchu produkcyjnego,
- logika zmiany schematu bazy wykonywana na żywych żądaniach użytkowników.

Dowody:

- CREATE TABLE w handlerze endpointu: .../api/matches/ping-pong/1v1/index.php:59
- próba ALTER TABLE w handlerze endpointu: .../api/matches/ping-pong/1v1/index.php:104

Wymagana poprawka:

- przeniesienie migracji schematu wyłącznie do procesu deploymentu,
- usunięcie DDL ze ścieżki obsługi żądań (request path).

P1-3: Ograniczona obserwowalność (metryki/tracing/alerty)

Wpływ:

- wolna reakcja na incydenty,
- słabe planowanie wydajności i trudność w przypisaniu wąskich gardeł.

Dowody:

- endpoint /health istnieje, ale brak eksportu metryk/tracingu w ścieżce serwera,
- widoczność operacyjna oparta głównie na logach.

Wymagana poprawka:

- dodanie metryk (rozmiar kolejki, lag ticku meczu, sesje WS, latencja nagród, błędy DB),
- dodanie progów alarmowych i dashboardów.

P1-4: Ścieżka fallback HTTP dla nagród nie posiada polityki timeoutów/retry

Wpływ:

- w trybie awaryjnym długie zawieszenia mogą opóźnić czyszczenie cyklu życia meczu i propagację statusu.

Dowody:

- zwykły fetch bez kontrolera timeoutu ani strategii ponowień:

.../node-server/src/rewardsClient.js:12

Wymagana poprawka:

- dodanie timeoutów + ograniczonych ponowień + mechanizmu Circuit Breaker.

P2 (średni priorytet)

P2-1: Strażnik uczestników w pollingu statusu zależy od kompletności payloadu

Wpływ:

- wyciek informacji w skrajnych przypadkach, jeśli payload jest uszkodzony lub brakuje ID zwycięzcy/przegranego.

Wymagana poprawka:

- wymuszenie ścisłej relacji właściciel/uczestnik poprzez znormalizowane kolumny tabeli.

P2-2: Próg klasyfikacji jakości pingu jest zbyt uproszczony

Wpływ:

- błędne etykiety "słabego połączenia" przy chwilowym jitterze mogą psuć UX i zwiększać liczbę celowych rozłączeń.

Dowody:

- klasyfikacja "weak" przy >150ms bez wygładzania danych: .../js/online.js:123

Wymagana poprawka:

- użycie średniej kroczącej + histerezy (N kolejnych próbek).

Dobre wieści / Co już działa

- Statusy rozłączenia przeciwnika i słabego połączenia są zaimplementowane w logice UI.
- Istnieją animowane style statusów (czerwony/pomarańczowy).
- Po stronie serwera istnieje logika obsługi rozłączenia obu graczy oraz czas na ponowne połączenie (reconnect grace logic).

Rekomendacja wydania

Nie uruchamiać pełnego wdrożenia na 200 tys. użytkowników w tym stanie.

Zalecany plan etapowy:

1. Naprawa wszystkich punktów **P0**.
2. Przeprowadzenie produkcyjnych testów obciążeniowych i spełnienie bramek SLO.
3. Naprawa przynajmniej **P1-1** i **P1-2** przed szerokim rolloutem.
4. Uruchomienie canary rollout (1-5%), a następnie stopniowe zwiększanie skali z monitorowaniem na żywo.

Minimalne bramki wejścia (Go-live gates)

- **Spójność nagród:** 100% determinizmu we wszystkich ścieżkach.
- **Matchmaking:** p95 czasu oczekiwania poniżej celu przy założonym szczycie.
- **Match tick lag:** p99 poniżej zdefiniowanego celu.
- **Baza danych:** opóźnienia zapisu i wskaźnik błędów poniżej progu.
- **Reconnect:** skuteczność ponownych połączeń powyżej ustalonego progu.
- **Monitoring:** aktywne dashboardy i alerty o incydentach.