Jan Sobański 144566
Ryszard Poznański 145407

Sprawozdanie sk2

Klon Kahoot'a

1. Opis projektu

1.1. Opis działania aplikacji

Projekt to klon aplikacji Kahoot, umożliwia tworzenie i realizowanie quizów pośród dużej grupy odbiorców. Działa on jako aplikacja webowa w architekturze klient / serwer. Istnieją dwa rodzaje użytkowników - twórcy autoryzowani przez nazwę użytkownika i hasło oraz uczestnicy autoryzowani przez numer pokoju oraz nazwę użytkownika, unikatową wewnątrz pokoju.

Uprawnienia twórcy quizów:
- Dodawanie nowych quizów
- Tworzenie nowych pokoi, w których odbywać ma się wybrany quiz
- Uruchomienie quizu po dołączeniu uczestników
- Podgląd pytań i odpowiedzi w trakcie działania quizu
- Uruchamianie kolejnych pytań
- Dostęp do podsumowania quizu (rankingu uczestników)
- Zamykanie pokoi

Uprawnienia uczestnika quizów:
- Dołączanie do pokoi
- Odpowiadanie na zadane pytania
- Podgląd rankingu graczy (w tym własnego wyniku)

1.2 Technologie

Technologie użyte do realizacji klienta to:
- React.js
- Typescript
- Socket.io

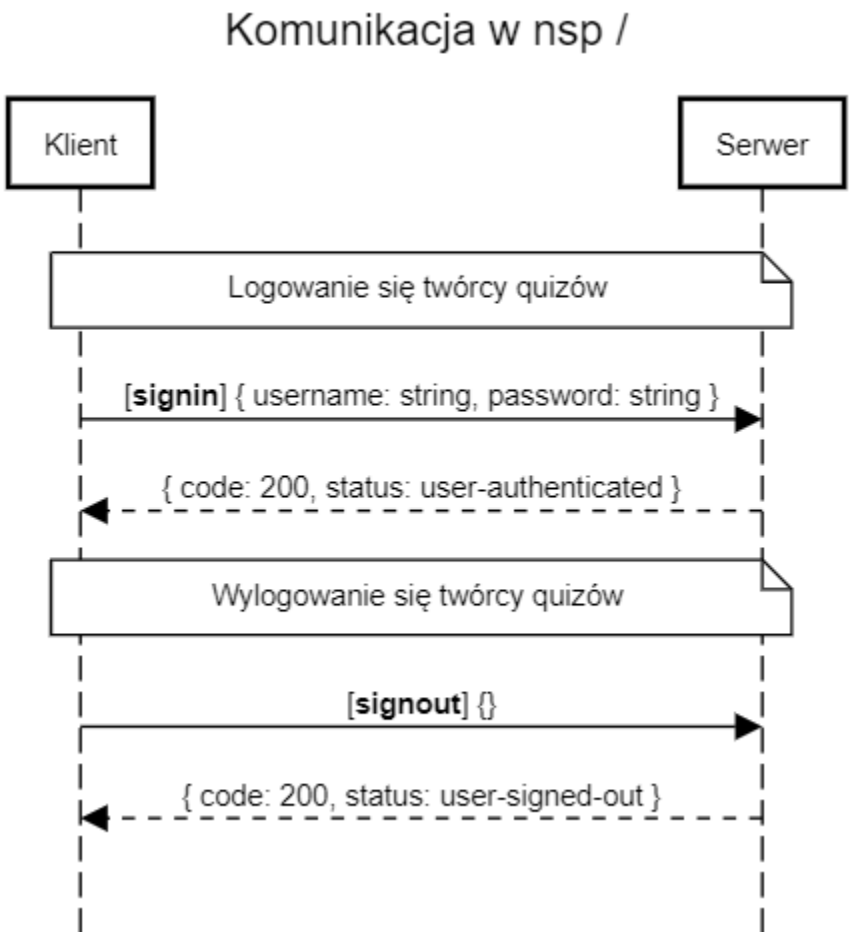
Technologie użyte do realizacji serwera to:
- Node.js
- Express.js
- Typescript
- Socket.io

2. Opis komunikacji pomiędzy serwerem i klientem

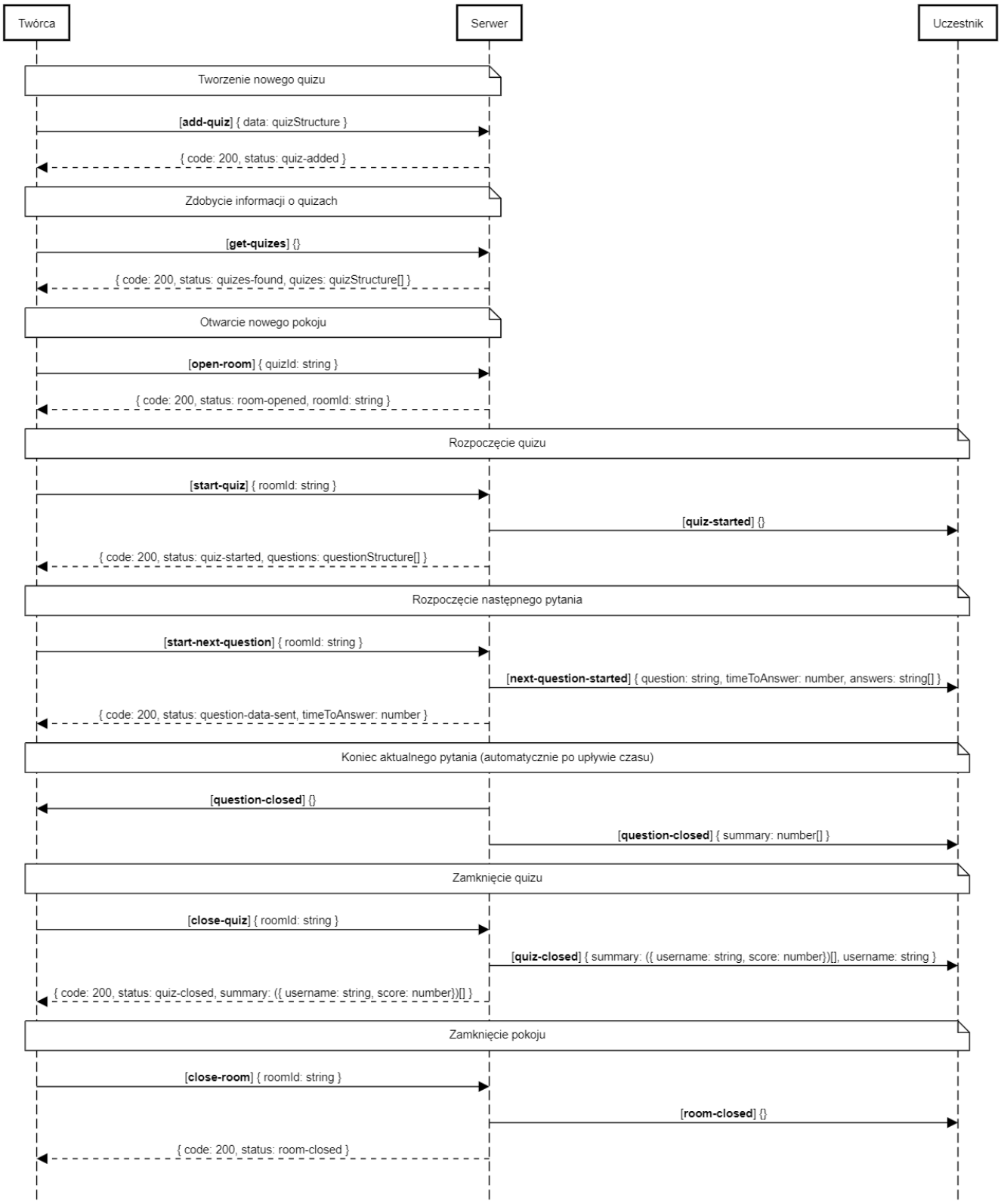
Poniższe schematy komunikacji ilustrują jedynie wyniki udanych zapytań, gdyż każde z nich wiąże się z wieloma potencjalnymi odpowiedziami, przykładowo dla zapytania [**signin**]

Oczekiwana odpowiedź:
- {code: 200, status: user-authenticated}

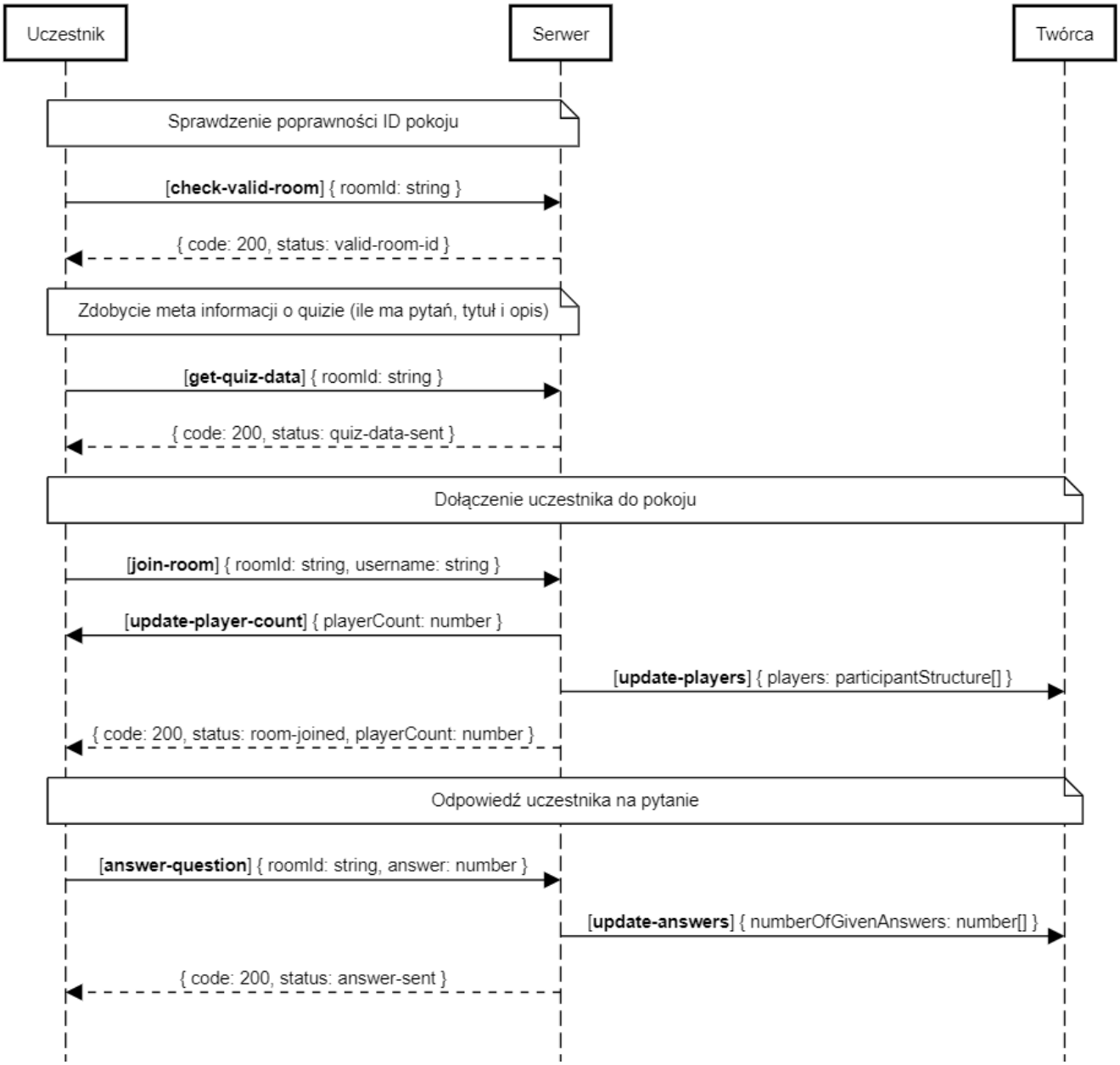
Możliwe błędy:
- {code: 400, status: user-already-authenticated}
- {code: 400, status: invalid-username-or-password}



Komunikacja w nsp /creator



Komunikacja w nsp /participant



3. Podsumowanie

Serwer wykorzystuje architekturę opartą na zdarzeniach (events). Pozwoliło to rozdzielić logikę pomiędzy różnego rodzaju komunikaty wysyłane zarówno przez serwer jak i klienta. Baza danych MongoDB posiada informację o twórcach quizów (creators), którzy mogą dowolnie tworzyć quizy i wykorzystywać je do otwierania pokoi gier. Dane pokoju są przechowywane w pamięci tymczasowej za pomocą bazy danych Redis pracującej w pamięci RAM serwera. Serwer do rozsyłania komunikatów do konkretnych gniazd wykorzystuje abstrakcję przestrzeni nazw (namespace) oraz pokoju (room) biblioteki Socket.IO. Logika aplikacji została rozdzielona na 3 przestrzenie nazw: “/”, “/creator” oraz “/participant”. Każdy klient jest jednocześnie przyłączony do wszystkich przestrzeni nazw, które współdzielą połączenie poprzez protokół websocket. Każda przestrzeń nazw posiada własne funkcje uruchamiane na zdarzenia należące wyłącznie do tej przestrzeni nazw. Dokładna interakcja między klientem a serwerem jest pokazana w sekcji 2. Dodatkowo wykorzystano funkcjonalność Pub/Sub bazy danych Redis, pozwalającą na emitowanie zdarzeń wewnątrz serwera. Ułatwiło to komunikację pomiędzy przestrzeniami nazw, co pozwoliło odseparować logikę na mniejsze części.

Największym problemem było zaprojektowanie stanu aplikacji przechowywanego w Redisie. Wymagało to od nas wymyślenia zbioru stanów, w których pokój może aktywnie się znajdować, oraz zaprojektowanie klienta w taki sposób, aby reagował odpowiednio na zmiany stanu w celu niwelowania możliwości wykonania niedozwolonych akcji. Biblioteka ReactJS umożliwia wygodne zarządzanie stanem aplikacji klienta, jednak integracja tej biblioteki z Socket.IO-client staje się problematyczna wraz z skomplikowaniem architektury serwera.