Systemy rozproszone
i przetwarzanie równoległe

Mini-Twitter

Prowadzący:
prof. dr hab. inż. Andrzej Feliks Duda

Wykonali:

Grzegorz Febert

Sebastian Karlewicz

Warszawa 2025

1. Opis projektu

MiniTwitter to rozproszona aplikacja umożliwiająca przesyłanie krótkich wiadomości pomiędzy użytkownikami. Projekt implementuje prosty system podobny do Twittera, gdzie użytkownicy mogą rejestrować się, logować oraz wysyłać wiadomości o maksymalnej długości 80 znaków.

Technologie wykorzystane w projekcie:

- **gRPC** – do komunikacji między klientem a serwerem.
- **Protocol Buffers** – do definiowania struktury wiadomości przesyłanych między klientem a serwerem.
- **Python** – jako główny język implementacji.
- **SQL** – baza danych do przechowywania użytkowników i wiadomości.

2. Architektura systemu

Aplikacja składa się z trzech głównych komponentów:

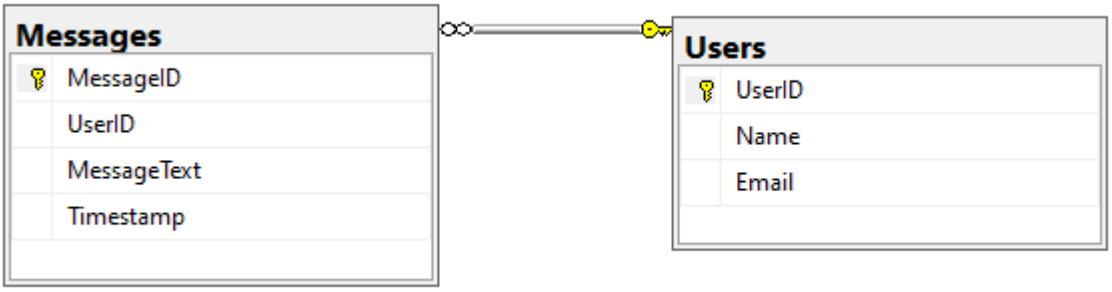
- **Klient** – interfejs użytkownika działający w trybie tekstowym, umożliwiający rejestrację, logowanie, wysyłanie wiadomości oraz pobieranie ostatnich wiadomości.

```
--- MiniTwitter ---
1. Register
2. Login
3. Send a message
4. Get messages
5. Exit
Choose an option (1/2/3/4/5):
```

- **Serwer (gRPC)** – obsługuje żądania klientów, przechowuje wiadomości w bazie danych i zwraca odpowiedzi poprzez gRPC.

```
Server running on port 50052
```

- **Baza danych (SQL Server)** – przechowuje użytkowników i wiadomości, wraz z informacją o czasie ich wysłania.



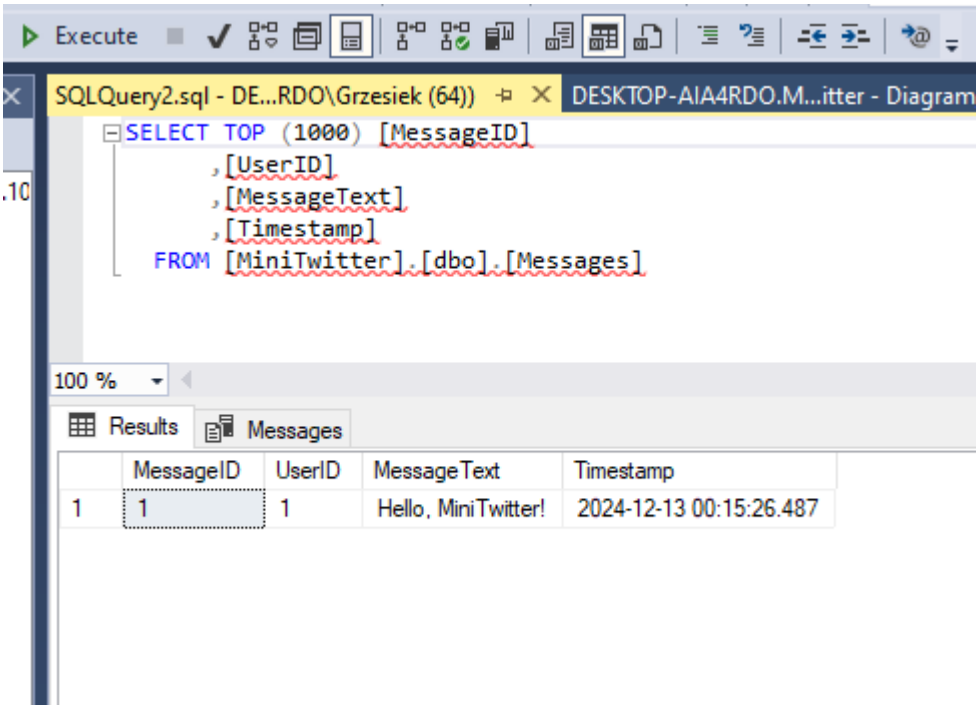
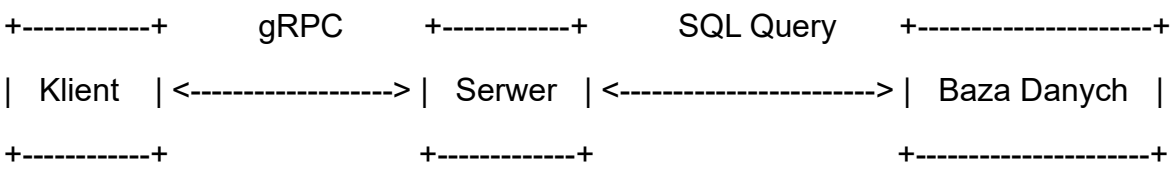


Diagram architektury



3. Opis implementacji

Rejestracja i logowanie

- **Rejestracja:** Użytkownik podaje nazwę i email. Serwer sprawdza, czy nie ma duplikatów i zapisuje dane.

```
def register_user(user_name, email):
    with grpc.insecure_channel('localhost:50052') as
channel:
        stub =
minitwitter_pb2_grpc.MiniTwitterStub(channel)
        response =
stub.registerUser(minitwitter_pb2.RegisterUserRequest(nam
e=user_name, email=email))
        print("Server response:", response.status)
```

```
--- MiniTwitter ---
1. Register
2. Login
5. Exit
Choose an option (1/2/3/4/5): 1
Enter your username: Kamil
Enter your email: kamil@example.com
Server response: User registered successfully
```

- **Logowanie:** Sprawdzane jest istnienie użytkownika w bazie.

```
def login_user(user_name, email):
    with grpc.insecure_channel('localhost:50052') as
channel:
        stub =
minitwitter_pb2_grpc.MiniTwitterStub(channel)
        response =
stub.loginUser(minitwitter_pb2.LoginUserRequest(name=user
_name, email=email))
        print("Server response:", response.status)
        return response.status == "Login successful"  #
Zwracamy True, jeśli login był udany
```

```
--- MiniTwitter ---
1. Register
2. Login
5. Exit
Choose an option (1/2/3/4/5): 2
Enter your username: Kamil
Enter your email: kamil@example.com
Server response: Login successful
```

Wysyłanie wiadomości

- Serwer zapisuje wiadomość wraz z czasem wysłania i przypisuje ją do użytkownika.

```
def send_message(user_name, content):
    with grpc.insecure_channel('localhost:50052') as
channel:
        stub =
minitwitter_pb2_grpc.MiniTwitterStub(channel)
        response =
stub.sendMessage(minitwitter_pb2.MessageRequest(name=user
_name, content=content))
        print("Server response:", response.status)
```

```
--- MiniTwitter ---
1. Register
2. Login
3. Send a message
4. Get messages
5. Exit
Choose an option (1/2/3/4/5): 3
Enter your message (max 80 characters): Witam wszystkich!!!
Server response: Message received
```

Pobieranie wiadomości

- Serwer zwraca n najnowszych wiadomości na podstawie żądania klienta.

```
def get_messages(count):
    with grpc.insecure_channel('localhost:50052') as
channel:
        stub =
minitwitter_pb2_grpc.MiniTwitterStub(channel)
        response =
stub.getMessages(minitwitter_pb2.GetMessagesRequest(count
=count))

        print("\n--- Najnowsze wiadomości ---")
        for message in response.messages:
            print(f"- {message}")
```

```
--- MiniTwitter ---
1. Register
2. Login
3. Send a message
4. Get messages
5. Exit
Choose an option (1/2/3/4/5): 4
How many messages to retrieve? 7

--- Najnowsze wiadomości ---
- Kamil (2025-01-30 22:06:20): Witam wszystkich!!!
- Damian (2025-01-28 21:51:34): cokolwiek 12 222 4
- Adam (2025-01-25 23:49:35): Hello
- Grzesiek (2025-01-25 22:32:22): fsjiufiusef nfesnefus ujesdue
- Adam (2025-01-25 22:31:08): adamjestem
- Grzesiek (2025-01-25 22:30:29): grzesiek
- Grzesiek (2025-01-25 21:46:41): aaa
```

4. Testy i wyniki

Testy funkcjonalne

- **Rejestracja nowego użytkownika** – poprawne i błędne przypadki.

```
Enter your username: Kamil
Enter your email: kamil@example.com
Server response: User registered successfully
```

- **Logowanie** – sprawdzenie poprawnych i niepoprawnych danych.

```
Enter your username: adad
Enter your email: wdwdw
Server response: User not found
```

- **Wysyłanie wiadomości** – sprawdzenie limitu znaków.

```
Enter your message (max 80 characters): qqqqqqqqqqqqq
Message too long! It must be 80 characters or less.
```

- **Pobieranie wiadomości** – sprawdzenie zwracania odpowiedniej liczby wiadomości.

```
How many messages to retrieve? 4

--- Najnowsze wiadomości ---
- Kamil (2025-01-30 22:06:20): Witam wszystkich!!!
- Damian (2025-01-28 21:51:34): cokolwiek 12 222 4
- Adam (2025-01-25 23:49:35): Hello
- Grzesiek (2025-01-25 22:32:22): fsjiufiusef nfesnefus ujesdue
```

Wyniki testów

Przeprowadziliśmy szczegółowe testy manualne, weryfikując poprawność działania wszystkich funkcjonalności aplikacji. Testy obejmowały zarówno scenariusze pozytywne, jak i negatywne, sprawdzając poprawne rejestrowanie i logowanie użytkowników, ograniczenia dotyczące długości wiadomości oraz poprawność zwracanych danych przy pobieraniu wiadomości. Dodatkowo przeprowadziliśmy testy obciążeniowe, symulując wielu użytkowników korzystających jednocześnie z aplikacji, aby ocenić stabilność działania serwera.

5. Podział pracy w zespole

Projekt został zrealizowany przez dwie osoby:

- **Grzegorz Febert** – zajmował się implementacją klienta, opracował interfejs użytkownika, stworzył bazę danych, pomagał przy implementacji serwera gRPC, przeprowadził testy manualne obejmujące różne scenariusze oraz przygotował dokumentacje projektu.
- **Sebastian Karlewicz** – odpowiadał za implementację serwera gRPC, pomagał przy implementacji klienta poprzez obsługę zapytań klientów, a także przy tworzeniu bazy danych, zajmował się integracją z bazą danych oraz zapewnieniem poprawnego przetwarzania wiadomości.

6. Podsumowanie

Projekt MiniTwitter pokazuje możliwości zastosowania gRPC w aplikacjach rozproszonych. System działa zgodnie z założeniami.