Raport gRPC-Web

Najciekawsze aspekty realizacji zadania

1. **Łatwość użycia serwisów opisanych językiem proto z poziomu JavaScriptu**
   Wystarczy skompilować plik proto z odpowiednim pluginem i stworzyć klienta gRPC w skrypcie JavaScript (tu musimy podać jedynie adres IP i port, jest to bardzo proste).
2. **Wykorzystanie proxy Envoy**
   Jako że gRPC korzysta z HTTP/2, a JavaScript nie wspiera wszystkich funkcji tego protokołu, z których korzysta gRPC i nie daje nad nim pełnej kontroli, konieczne jest wykorzystanie proxy, który będzie przesyłał requesty między klientem webowym a serwerem. Do realizacji użyłem proxy Envoy działającego w środowisku kontenera dockerowego (tak sugerowała dokumentacja gRPC-Web). Proxy przyjmuje requesty od klienta webowego wysłane za pomocą HTTP/1.1, konwertuje jest na HTTP/2 i przesyła dalej do serwera, który je obsługuje i odsyła odpowiedź. W ten sposób klient łączy się z proxy (w moim przypadku localhost i port 8080) i „myśli”, że jest połączony bezpośrednio z serwerem. Serwer natomiast dostaje requesty od proxy i również „myśli”, że otrzymuje je bezpośrednio od klienta. Wykorzystanie proxy powoduje, że możemy wywołać zdalnie metody z wykorzystaniem kodu uruchomionego w środowisku przeglądarki i nie musimy przy tym modyfikować serwera, po jego stronie jest nadal zwykłe gRPC.
3. **Możliwość wykorzystania Server-side Streaming**
   gRPC udostępnia 4 typy RPC:
   - Unary RPCs – prosty model request-response,
   - Client-side Streaming RPCs – klient wysyła strumień wiadomości, serwer odsyła pojedynczą odpowiedź,
   - Server-side Streaming RPCs – klient wysyła pojedynczą wiadomość, serwer odsyła strumień w odpowiedzi,
   - Bi-directional Streaming RPCs – połączenie powyższych, klient wysyła strumień i otrzymuje w odpowiedzi strumień.

   gRPC-Web wspiera dwa z powyższych typów – Unary RPCs i Server-side Streaming, nie ma możliwości użycia pozostałych dwóch. W implementacji zostały zaprezentowane oba typy, factorial, uppercase i average to Unary RPCs, a random to Server-side Streaming RPC.

4. **Połączenie dwóch różnych języków**

   Zadanie dobrze pokazało, że interfejs zdefiniowany w języku proto może być z powodzeniem użyty w wielu językach, w tym przypadku Java i JavaScript. Wystarczy tylko skompilować z odpowiednim pluginem.

5. **Analiza wydajności i ograniczeń komunikacji**

   Jako że aplikacja kliencka nie komunikuje się bezpośrednio z serwerem gRPC, tylko z proxy, to komunikacja jest z pewnością wolniejsza niż gdyby była bezpośrednia. Co więcej, działanie połączenia jest uzależnione od działania proxy (klient „myśli”, że jest połączony z serwerem, a serwer „myśli”, że otrzymuje wiadomości od klienta, a w rzeczywistości bezpśrednio komunikują się z proxy). Jest to pewne ograniczenie i skomplikowanie systemu.