Mateusz Śmigała

Opcjonalne pola struktur danych i argumenty wywołania middleware

Wstęp:

Wykorzystane przeze mnie technologie middleware:

- **ICE**
- **gRPC**

W obu przypadkach część serwerowa została napisana w języku Java, a część kliencka w języku Python.

Omówię w jaki sposób w interfacach podobnych technologii definiuje się pola opcjonalne, jak wygląda ich implementacja, oraz serializacja danych. Omówię dane zagadnienie na przykładzie interfacu zawierającego przykładowe dane dotyczące dostawy jedzenia, takie jak adres dostawy (nazwa ulicy, numer domu, oraz opcjonalny numer mieszkania), numer telefonu klienta, opcjonalne napiwki, jeżeli zostanie podana tablica napiwków będę w stanie obliczyć z nich średnią.

Definiowanie pól opcjonalnych w interface'ach:

ICE:

```
3
4   module DeliveryService {
5       sequence<double> DeliveryTips;
6
7       class Delivery {
8           string phoneNumber;
9           string streetName;
10          int houseNumber;
11          optional(1) int apartmentNumber;
12          optional(2) DeliveryTips deliveryTips;
13      };
14
15      interface Promotion {
16          double CountTips(Delivery delivery);
17      };
18  };
19
```

W ICE wystarczy oznaczyć poprzez pole “optional()” a jako argument podać liczbę naturalną która się wcześniej nie powtórzyło.

gRPC:

```
7
8  message Delivery {
9    string phoneNumber = 1;
10   string streetName = 2;
11   int32 houseNumber = 3;
12   optional int32 apartmentNumber = 4;
13   message DeliveryTips {
14     repeated double tip = 1;
15   }
16   optional DeliveryTips deliveryTips = 5;
17 }
18
19 message DeliveryTip {
20   double deliveryAverageTips = 1;
21 }
22
23 service Promotion {
24   rpc CountTips(Delivery) returns (DeliveryTip) {}
25 }
26
```

W gRPC wystarczy oznaczyć poprzez pole “optional”.

Implementacja:

ICE:

```
2 usages
public class PromotionI implements Promotion {
    1 usage
    @Override
    public double CountTips(Delivery delivery, Current current) {
        try {
            double[] tipsArray = delivery.getDeliveryTips();

            if (tipsArray != null && tipsArray.length > 0) {
                double sum = 0;
                for (double tip : tipsArray) {
                    sum += tip;
                }
                return sum / tipsArray.length;
            } else {
                return 0.0;
            }
        } catch (java.util.NoSuchElementException e) {
            System.err.println("Lista deliveryTips nie została podana.");
            e.printStackTrace();
            return 0.0;
        }
    }
}

```

Podczas implementacji metody do obliczania średniej próbę pobrania listy napiwków należy opakować w try/catch, ponieważ zwracany jest nam niżej zaprezentowany exception.

```
    1 usage
    public double[] getDeliveryTips()
    {
        if(!_deliveryTips)
        {
            throw new java.util.NoSuchElementException("deliveryTips is not set");
        }
        return deliveryTips;
    }

```

gRPC:

```java
public class PromotionImpl extends PromotionGrpc.PromotionImplBase {
    1 usage
    @Override
    public void countTips(Delivery request,
                          io.grpc.stub.StreamObserver<sr.grpc.gen.DeliveryTip> responseObserver){
        List<Double> tipsList = request.getDeliveryTips().getTipList();

        double averageTip;
        if (tipsList.isEmpty()) {
            averageTip = Double.NaN;
        } else {
            double sum = 0;
            for (Double tip : tipsList) {
                sum += tip;
            }
            averageTip = sum / tipsList.size();
        }

        DeliveryTip deliveryTipResponse = DeliveryTip.newBuilder()
                .setDeliveryAverageTips(averageTip)
                .build();

        responseObserver.onNext(deliveryTipResponse);
        responseObserver.onCompleted();
    }
}
```

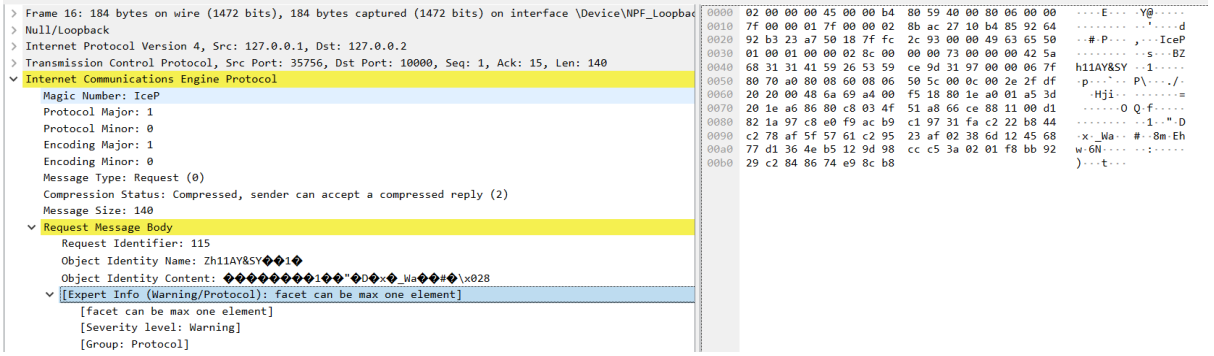
W gRPC w przeciwieństwie do ICE'a nie musimy opakowywać próby pobrania danych oznaczonych poprzez optional w try/catch, ponieważ nie zostanie nam rzucony wyjątek. Jednak wypada samemu zadbać o odpowiednie obwarunkowanie przypadku gdy opcjonalne pola nie zostaną podane.

Serializacja danych:

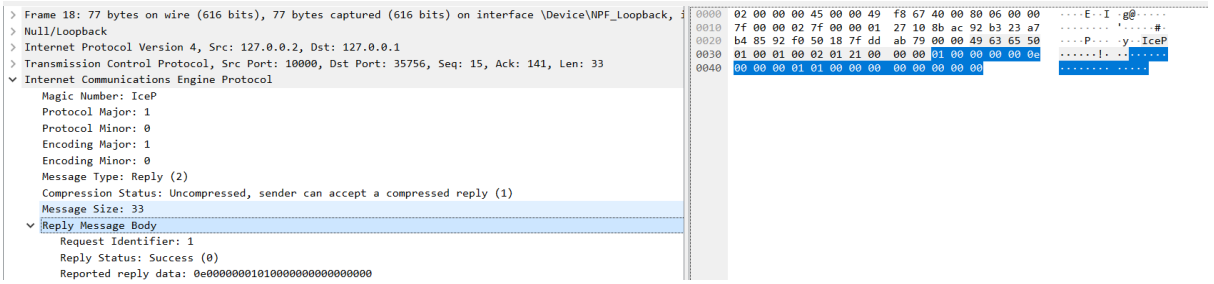
ICE:

W przypadku gdy nie podamy listę napiwków:

Zapytanie na serwer:

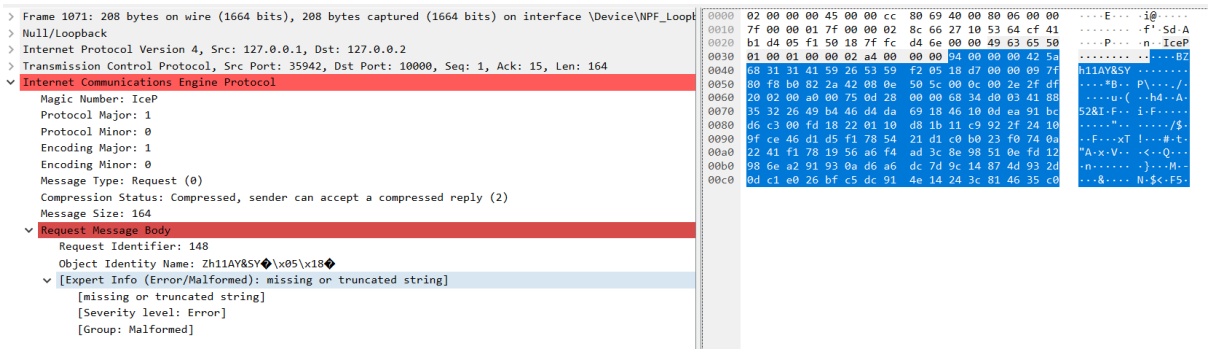


Odpowiedź z serwera:

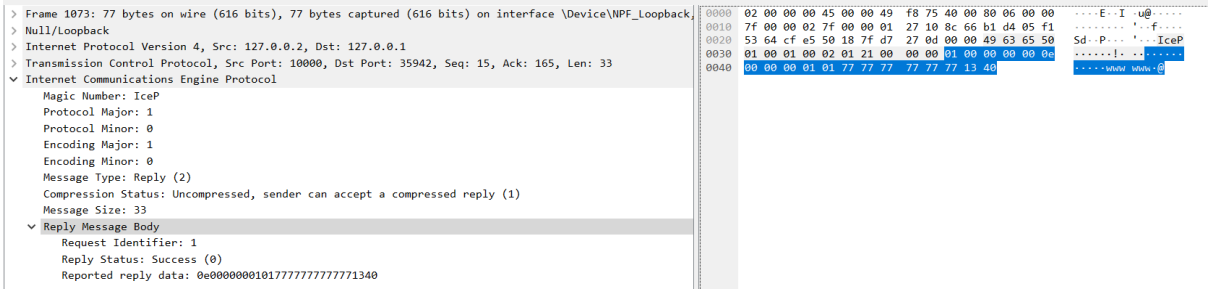


W przypadku gdy podamy listę napiwków:

Zapytanie na serwer:



Odpowiedź z serwera:

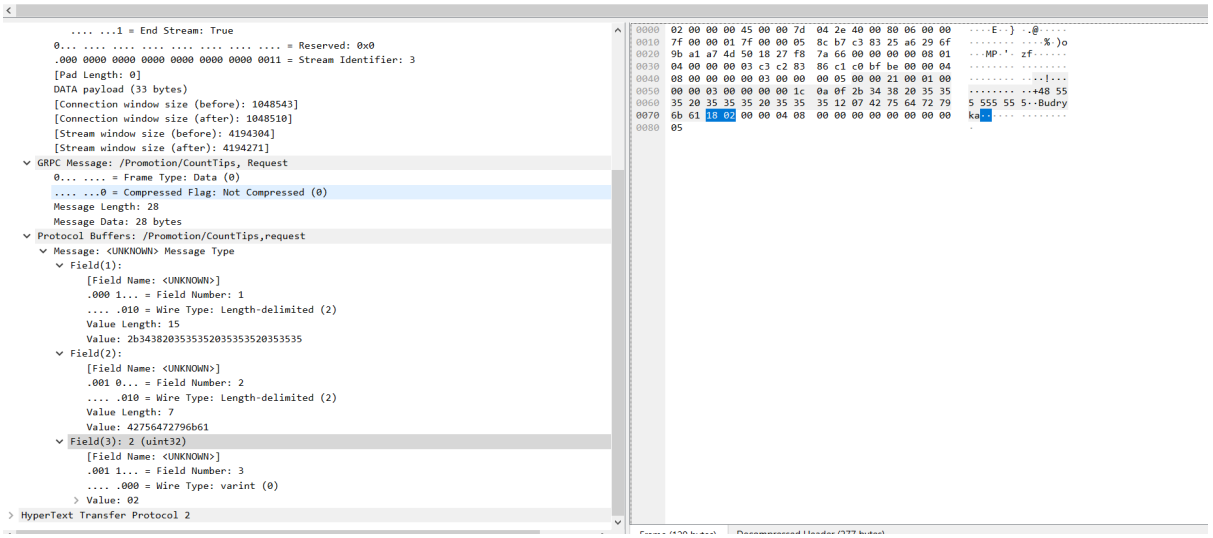


W obu przypadkach wszystkie dane są kompresowane.

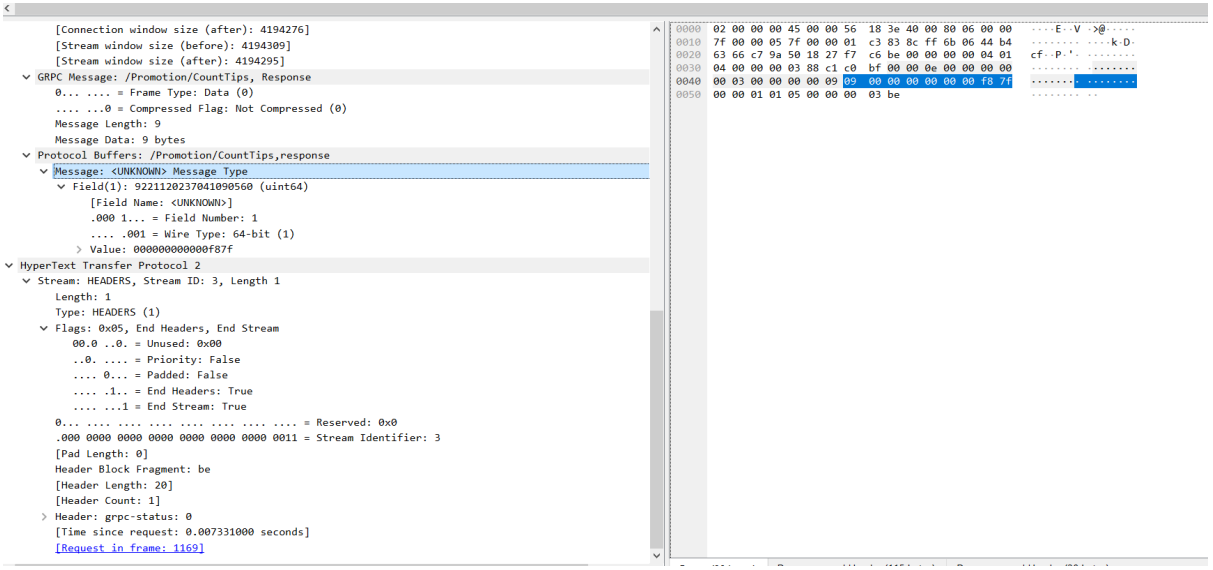
gRPC:

W przypadku gdy nie podamy listę napiwków:

Zapytanie na serwer:

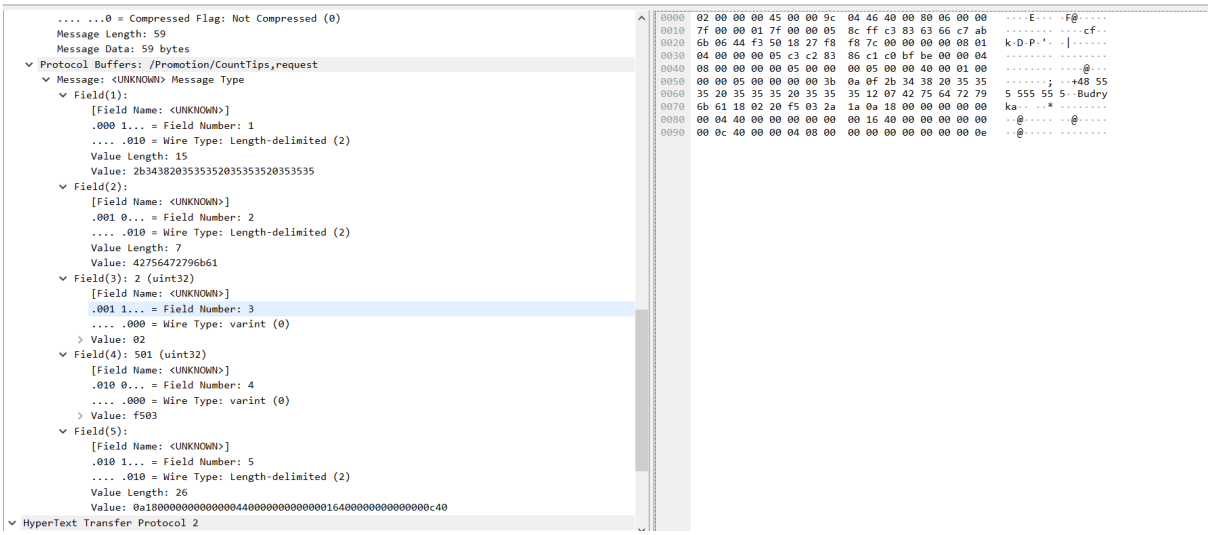


Odpowiedź z serwera:

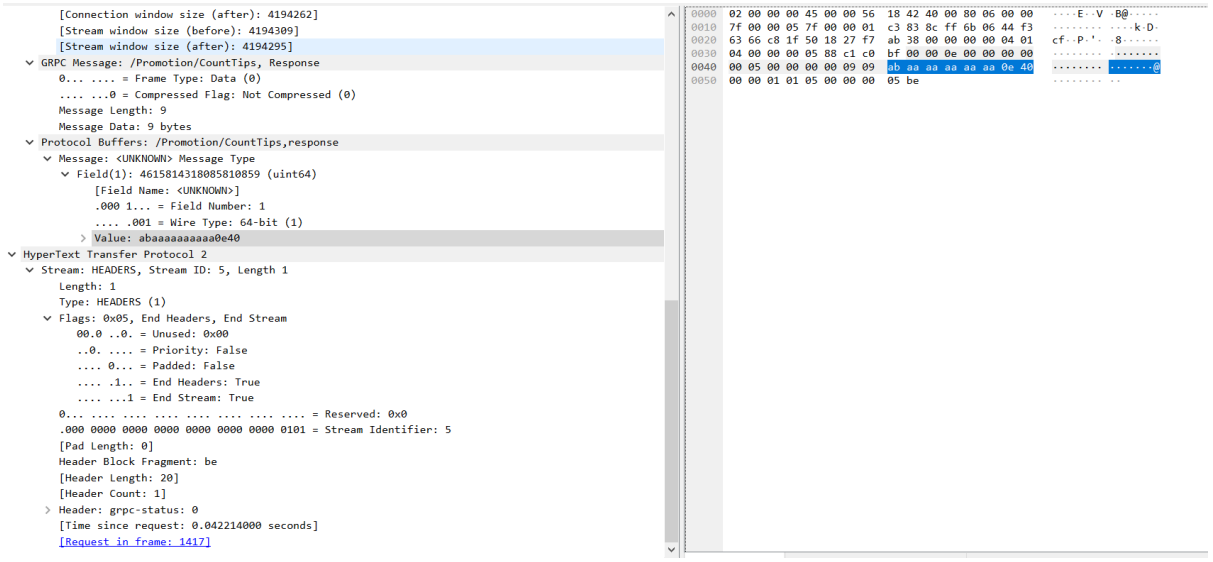


W przypadku gdy podamy listę napiwków:

Zapytanie na serwer:



Odpowiedź z serwera:



Jak możemy zauważyć gRPC używa http2 wartości które są stringami nie są kodowane, natomiast cała reszta już tak.