DEZSYS_GK72_DATAWAREHOUSE_GRPC

Mario Milkov 4DHIT

Grundlagen

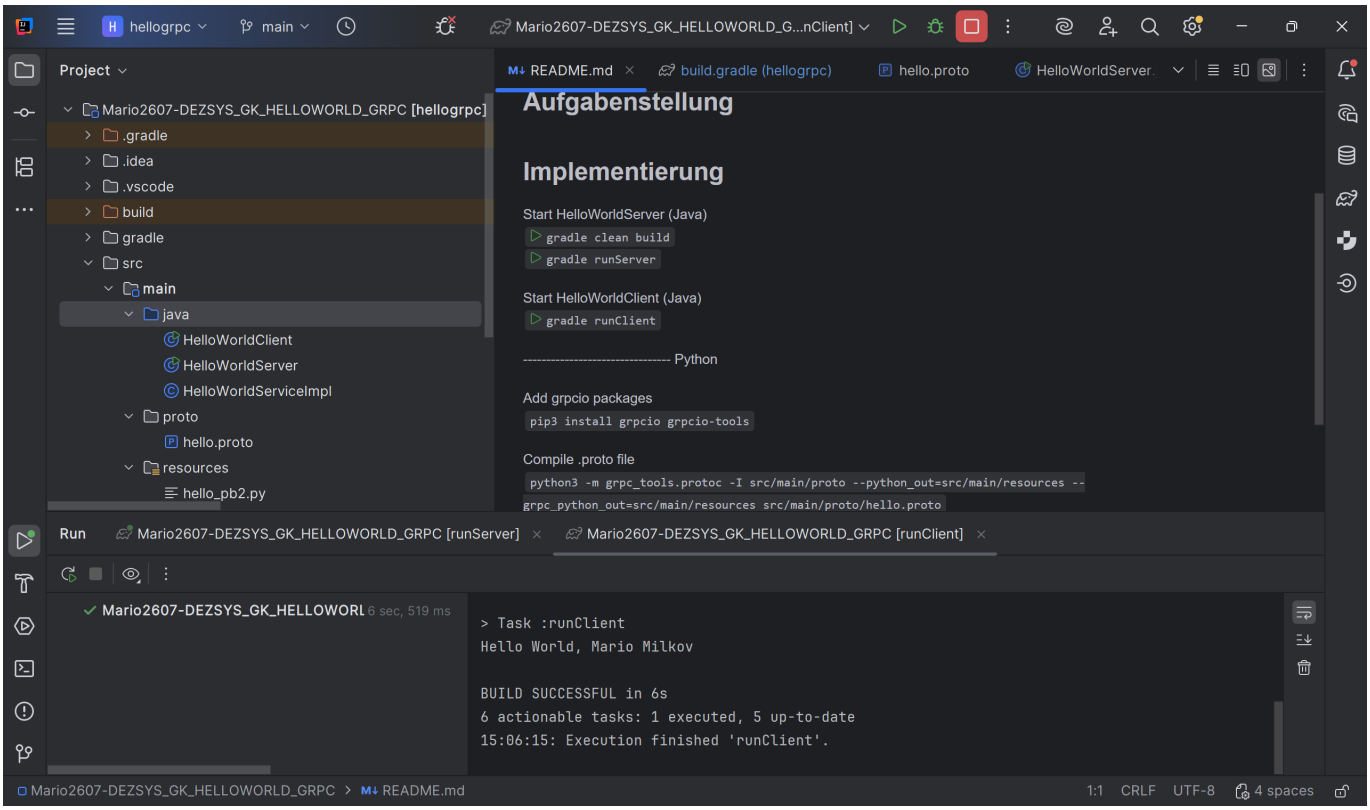
Answer the questions below about the gRPC framework

- What is gRPC and why does it work accross languages and platforms?
  - gRPC is a high-performance open-source RPC framework by Google that lets programs communicate directly across systems. It works across languages and platforms because it uses HTTP/2, Protocol Buffers, and provides official libraries for many languages.
- Describe the RPC life cycle starting with the RPC client?
  - The client calls a local stub method, which sends a request to the server over HTTP/2; the server receives, processes it, and sends back a response; the client stub then returns the result to the caller.
- Describe the workflow of Protocol Buffers?
  - Protocol Buffers work by writing a `.proto` file, generating code from it, and then using the generated classes to serialize (encode) and deserialize (decode) data. Both client and server use the same `.proto` file so they exchange data in a compact, structured format.
- What are the benefits of using protocol buffers?
  - Protocol Buffers are very fast, use little network bandwidth, and create small, efficient messages. They are also strongly typed, easy to version, and work in many programming languages.
- When is the use of protocol not recommended?
  - They are not recommended when you need human-readable data (like JSON or XML). Also not ideal if your system does not need a strict schema or if you work mainly in environments that don't use binary formats.
- List 3 different data types that can be used with protocol buffers?
  - `string`
  - `int32`
  - `bool`

Demo Project

First i forked the project from our teacher and then proceeded with the following steps from the task. For the demo project I had to do all the steps from the `README.md` file and after that I just

had to change my name in the `build.gradle` and in the `HelloWorldClient.java`.



We can see very clear that in the console that the demo project worked!

Erweiterte Grundlagen

Customize the service so that a simple DataWarehouse record (see exercise MidEng 7.1) can be transferred

To complete this task, first i had to change the `hello.proto` file:

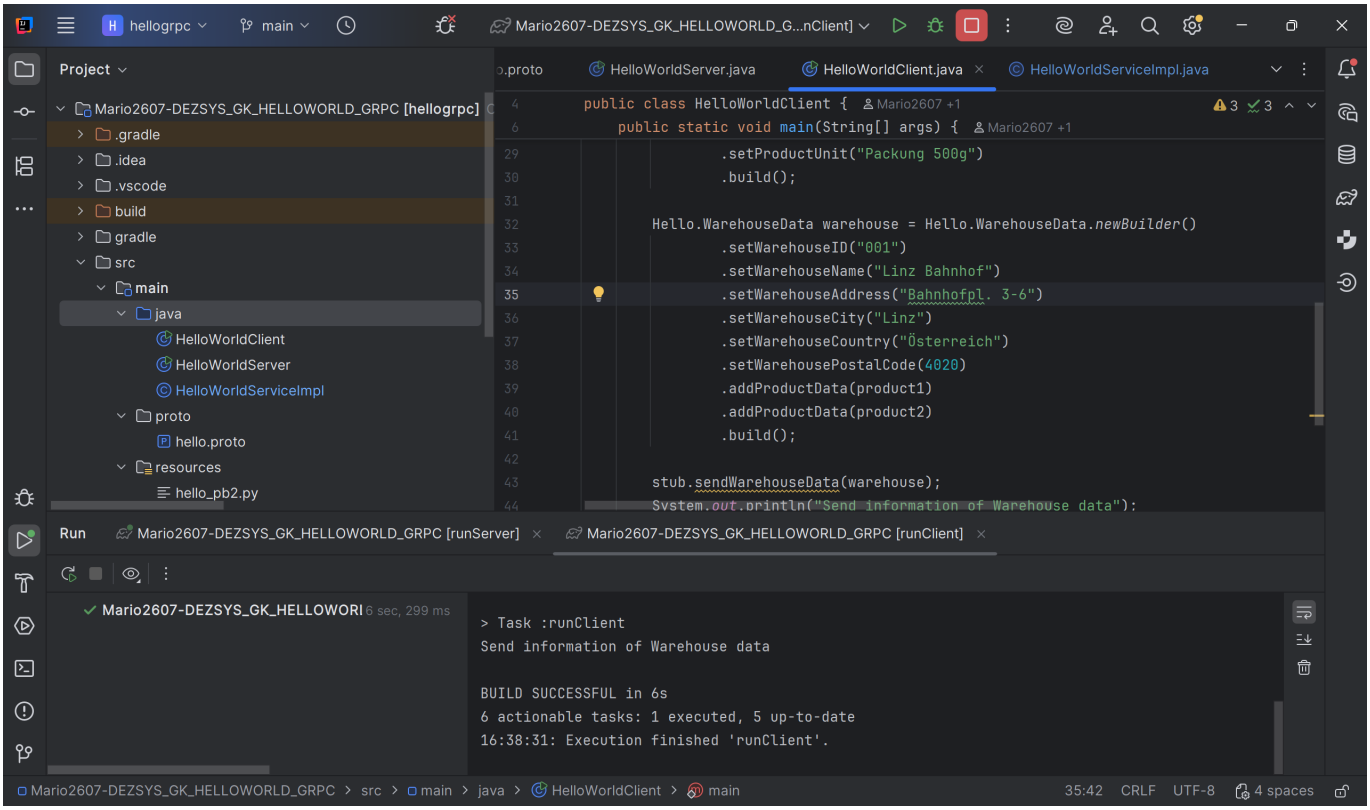
```
syntax = "proto3";

service HelloWorldService {
  rpc hello(HelloRequest) returns (HelloResponse) {}
  rpc sendWarehouseData (WarehouseData) returns (WarehouseResponse);
}

message HelloRequest {
  string firstname = 1;
  string lastname = 2;
}

message HelloResponse {
  string text = 1;
}

message ProductData {
  string productId = 1;
  string productName = 2;
  string productCategory = 3;
  int32 productQuantity = 4;
  string productUnit = 5;
}

message WarehouseData {
  string warehouseID = 1;
  string warehouseName = 2;
  string warehouseCountry = 4;
  string warehouseCity = 5;
  string warehouseAddress = 6;
  repeated ProductData productData = 8;
}

message WarehouseResponse {
  WarehouseData data = 1;
}
```

The proto file defines what data the client and server can send to each other. It includes two functions and simple data structures like WarehouseData and ProductData that describe what information is transferred.

After that I changed the `HelloWorldClient.java` file:

```java
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;

public class HelloWorldClient {

    public static void main(String[] args) {

        String firstname = args.length > 0 ? args[0] : "Max";
        String lastname  = args.length > 1 ? args[1] : "Mustermann";

        ManagedChannel channel = ManagedChannelBuilder.forAddress("localhost", 50051)
                .usePlaintext()
                .build();

        HelloWorldServiceGrpc.HelloWorldServiceBlockingStub stub =
HelloWorldServiceGrpc.newBlockingStub(channel);
        Hello.ProductData product1 = Hello.ProductData.newBuilder()
                .setProductId("00-443175")
                .setProductName("Bio Orangensaft Sonne")
                .setProductCategory("Getraenk")
                .setProductQuantity(1500)
                .setProductUnit("Packung 1L")
                .build();

        Hello.ProductData product2 = Hello.ProductData.newBuilder()
                .setProductId("02-341867")
                .setProductName("Milka Tafel")
                .setProductCategory("Sueßigkeit")
                .setProductQuantity(1000)
                .setProductUnit("Packung 500g")
                .build();

        Hello.WarehouseData warehouse = Hello.WarehouseData.newBuilder()
                .setWarehouseID("001")
                .setWarehouseName("Linz Bahnhof")
                .setWarehouseAddress("Bahnhofpl. 3-6")
                .setWarehouseCity("Linz")
                .setWarehouseCountry("Österreich")
                .setWarehousePostalCode(4020)
                .addProductData(product1)
                .addProductData(product2)
                .build();

        stub.sendWarehouseData(warehouse);
        System.out.println("Send information of Warehouse data");
        channel.shutdown();

    }

}
```

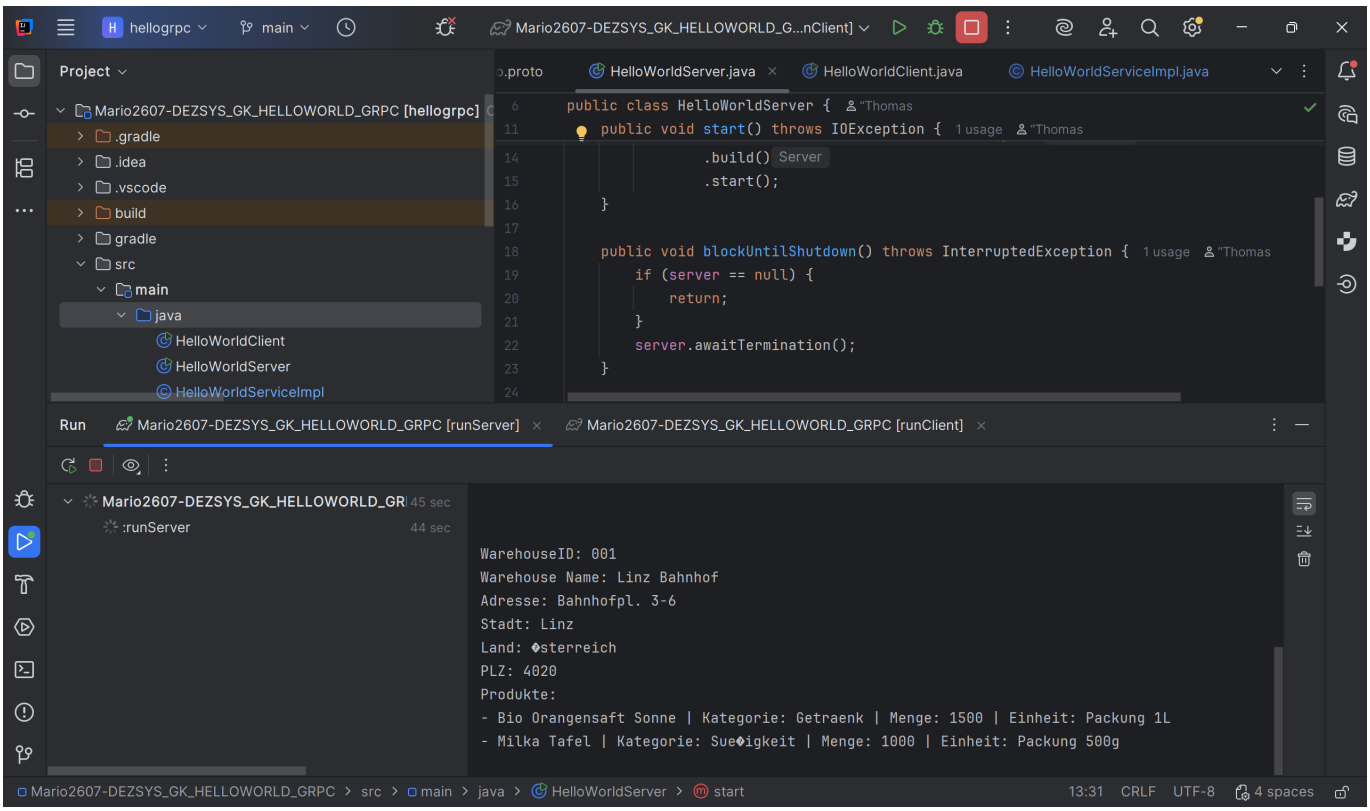
Here the client creates all warehouse data and product data and sends it to the server using the sendWarehouseData function.



And lastly I had to change the `HelloWorldServiceImpl.java` file:

```
import io.grpc.stub.StreamObserver;

public class HelloWorldServiceImpl extends
HelloWorldServiceGrpc.HelloWorldServiceImplBase {

    @Override
    public void hello( Hello.HelloRequest request, StreamObserver<Hello.HelloResponse>
responseObserver) {

        System.out.println("Handling hello endpoint: " + request.toString());

        String text = "Hello World, " + request.getFirstname() + " " +
request.getLastname();
        Hello.HelloResponse response =
Hello.HelloResponse.newBuilder().setText(text).build();

        responseObserver.onNext(response);
        responseObserver.onCompleted();

    }

    @Override
    public void sendWarehouseData(Hello.WarehouseData request,
StreamObserver<Hello.WarehouseResponse> responseObserver) {

        System.out.println("WarehouseID: " + request.getWarehouseID());
        System.out.println("Warehouse Name: " + request.getWarehouseName());
        System.out.println("Adresse: " + request.getWarehouseAddress());
        System.out.println("Stadt: " + request.getWarehouseCity());
        System.out.println("Land: " + request.getWarehouseCountry());
        System.out.println("PLZ: " + request.getWarehousePostalCode());
        System.out.println("Produkte:");

        for (Hello.ProductData p : request.getProductDataList()) {
            System.out.println("- " + p.getProductName() + " | Kategorie: " +
p.getProductCategory() + " | Menge: " + p.getProductQuantity() + " | Einheit: " +
p.getProductUnit());
        }
        Hello.WarehouseResponse response =
Hello.WarehouseResponse.newBuilder().build();

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

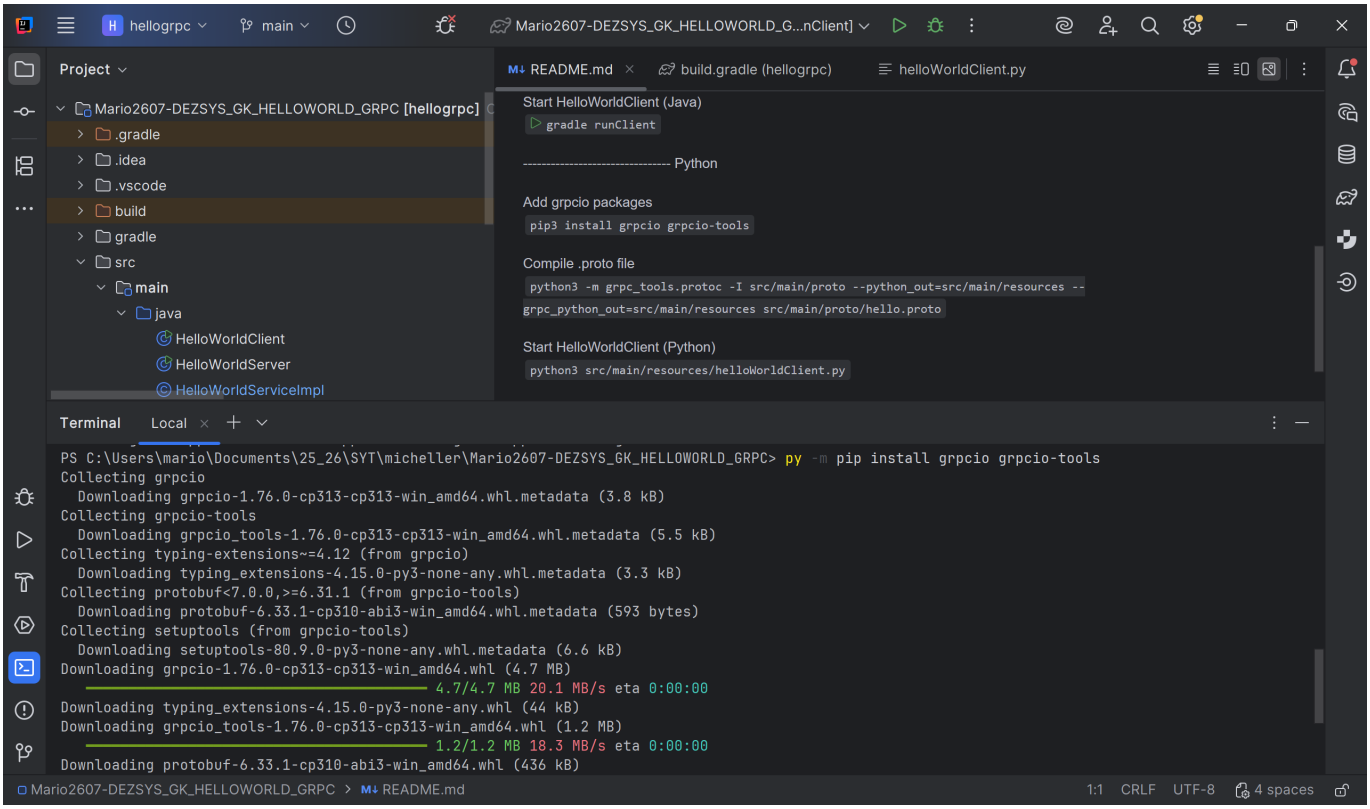
I had to do it so that the server receives the complete warehouse data from the client and prints all information to the console.



And we can see that i works!

Vertiefung

I chose python for my second language and for that i had to run these commands: `py -m pip install grpcio grpcio-tools`, `py -m grpc_tools.protoc -I src/main/proto --python_out=src/main/resources --grpc_python_out=src/main/resources src/main/proto/hello.proto`




```
PS C:\Users\mario\Documents\25_26\SYT\michaller\Mario2607-DEZSYS_GK_HELLOWORLD_GRPC> py -m grpc_tools.protoc -I src/main/proto --python_out=src/main
/resources --grpc_python_out=src/main/resources src/main/proto/hello.proto
PS C:\Users\mario\Documents\25_26\SYT\michaller\Mario2607-DEZSYS_GK_HELLOWORLD_GRPC>
```

Here is the `helloWorldClient.py` file:

```python
import sys
import grpc

import hello_pb2 as hello_pb2
import hello_pb2_grpc as hello_pb2_grpc


def main():
    firstname = sys.argv[1] if len(sys.argv) > 1 else "Max"
    lastname = sys.argv[2] if len(sys.argv) > 2 else "Mustermann"

    # Connect to server
    with grpc.insecure_channel("localhost:50051") as channel:
        stub = hello_pb2_grpc.HelloWorldServiceStub(channel)

        # ---- ORIGINAL HELLO ----
        request = hello_pb2.HelloRequest(firstname=firstname, lastname=lastname)
        response = stub.hello(request)
        print()
        print(response.text)
        print()

        # ---- NEW: PRODUCTS ----
        p1 = hello_pb2.Product(
            productID="00-443175",
            productName="Bio Orangensaft Sonne",
            productCategory="Getraenk",
            productQuantity=1500,
            productUnity="Packung 1L"
        )

        p2 = hello_pb2.Product(
            productID="02-341867",
            productName="Milka Tafel",
            productCategory="Sueßigkeit",
            productQuantity=1000,
            productUnity="Tafel 500g"
        )

        # ---- NEW: WAREHOUSE ----
        warehouse_data = hello_pb2.WarehouseData(
            warehouseID="001",
            warehouseName="Linz Bahnhof",
            warehouseAddress="Bahnhofpl. 3-6",
            warehousePostalCode=4020,
            warehouseCity="Linz",
            warehouseCountry="Österreich",
            products=[p1, p2]
        )
```

```
        # ---- SEND RPC ----
        ack = stub.sendWarehouseData(warehouse_data)
        print("Server says:", ack.status)
        print()



        # ---- FULL ATTRIBUTE OUTPUT ----
        print("=== FULL DATA SENT (Client-side) ===")
        print("WarehouseID:       ", warehouse_data.warehouseID)
        print("WarehouseName:     ", warehouse_data.warehouseName)
        print("Address:           ", warehouse_data.warehouseAddress)
        print("Postal Code:       ", warehouse_data.warehousePostalCode)
        print("City:              ", warehouse_data.warehouseCity)
        print("Country:           ", warehouse_data.warehouseCountry)
        print()

        print("Products:")
        for product in warehouse_data.products:
            print("  ------------------------------")
            print("  ProductID:      ", product.productID)
            print("  Name:           ", product.productName)
            print("  Category:       ", product.productCategory)
            print("  Quantity:       ", product.productQuantity)
            print("  Unity:          ", product.productUnity)
        print("  ------------------------------\n")


if __name__ == "_main_":
    main()t("  ------------------------------\n")


if __name__ == "_main_":
    main()
```

This Python client first sends a hello request, then creates full warehouse data and product data and sends it to the gRPC server. After sending, it prints all data on the client side and shows the server's confirmation message.