

DEZSYS_GK72_ELECTION_GRPC

Maged Negm 4CHIT

Questions

Quellen:

[What is RPC? gRPC Introduction. - YouTube](#)

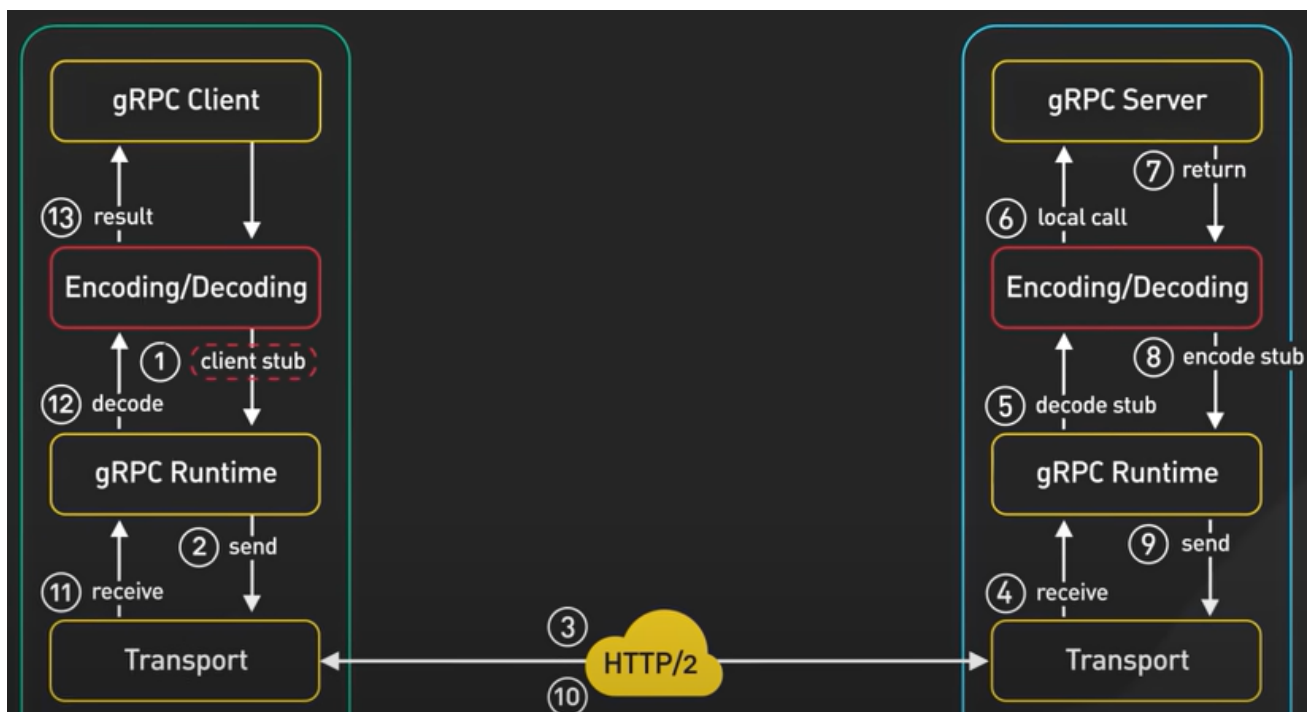
[Introduction to gRPC | gRPC](#)

<https://hackolade.com/help/Protobuf.html#:~:text=Protobuf%20supports%20different%20families%20of,or%20any%20imported%20proto%20files.>

- What is gRPC and why does it work accross languages and platforms?

gRPC is an open-source remote procedure call framework created by Google in 2016. The remote procedure call enables the client to invoke some code on another machine if it is a local function call. So it sends the data and the other machine delivers its Output back to the Client.

- Describe the RPC life cycle starting with the RPC client?



1. Client invokes the client stub (client code generated) by gRPC tooling at build time

2. gRPC encodes the data passed to the client stub into protocol buffers and sends it to the low-level transport layer
3. The data gets sent over the network as a stream of HTTP/2 data frames
4. The server's low-level transport layer receives the data from the network
5. gRPC decodes the data
6. It invokes a local call to the server application
7. The results get returned
8. They get encoded into protocol buffers
9. The buffers get sent to the low-level transport layer again
10. The client's low-level transport layer receives the packet over the network
11. The client receives the data
12. gRPC decodes them
13. The results get sent to the client application

- Describe the workflow of Protocol Buffers?
 - Protocol buffers support strongly-typed schema definitions where the structure of the data is defined in a proto file.
 - The Protobuf compiler (protoc) generates source code from .proto files, creating classes or structs for messages and service interfaces in target programming languages.
 - Applications then use this code to serialize data into a compact binary format and deserialize it back into usable objects.
- What are the benefits of using protocol buffers?
 - It can generate code for different programming languages, so systems written in various languages can easily work together.
 - High-performance because the protocol buffers is a very efficient binary encoding format that saves space and speeds up data handling compared to formats like json or xml AND the use of HTTP/2.

- Protobuf allows for the evolution of data schemas without breaking existing implementations, facilitating smooth updates and versioning.
- When is the use of protocol not recommended?

Text-based formats such as JSON or XML are preferable for configuration files or debugging when human-readable data is required.

- List 3 different data types that can be used with protocol buffers?
 - string
 - bytes
 - int
 - bool

HelloWorld application

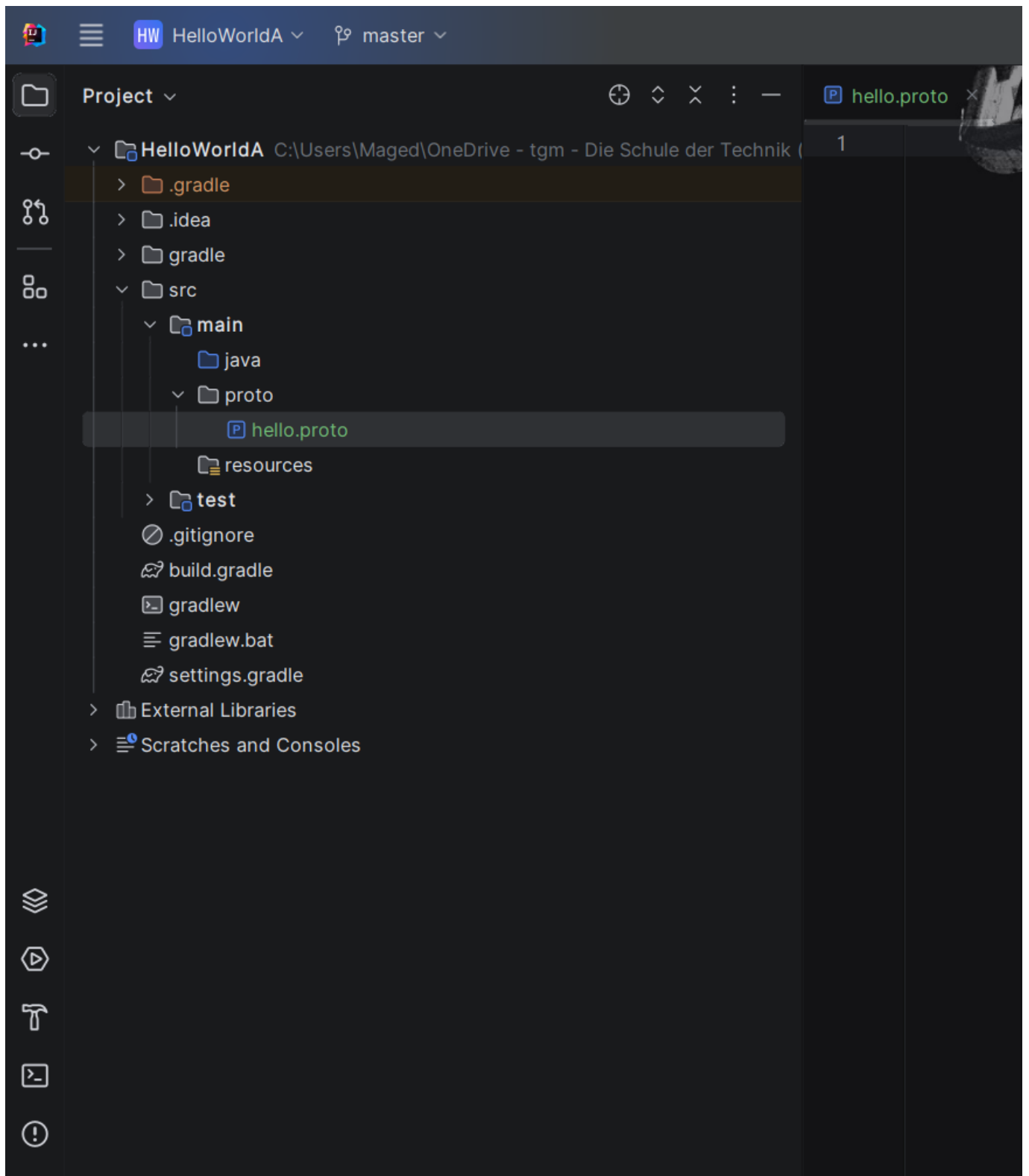
Quellen:

Kurs: [Implementing gRPC Service \(Server\) Using Java + Gradle - YouTube](#)

Kurs: <https://intuting.medium.com/implement-grpc-service-using-java-gradle-7a54258b60b8>

1. Proto-File

First, I created a new project in IntelliJ and added a directory named `proto` inside the `main` package. In the `proto` directory, I created a `.proto` file called `hello.proto`.



Here is the code you can insert into the `hello.proto` file:

```

syntax = "proto3";

service HelloWorldService {
    rpc hello(HelloRequest) returns (HelloResponse) {}
}

message HelloRequest {
    string text = 1;
}

message HelloResponse {
    string text = 1;
}

```

Explanation: This defines the gRPC service and the data structures used for communication.

2. Generate Code

Now you have to put the code given by the website above into the `build.gradle`

Because the tutorial is a bit older you've to do the following changes:

```
classpath 'com.google.protobuf:protobuf-gradle-plugin:0.9.4'
```

in this part write instead of `compile....` `implementation...`

```

compile "io.grpc:grpc-netty:${grpcVersion}"
compile "io.grpc:grpc-protobuf:${grpcVersion}"
compile "io.grpc:grpc-stub:${grpcVersion}"
compile group: 'com.google.protobuf', name: 'protobuf-java-util', version: '3.12.2'

```

then run it with the elephant on the Site and execute `./ gradlew build` on the terminal

Explanation: This setup configures the build system to include gRPC and Protocol Buffers plugins and generates Java code from the `hello.proto` file.

3. HelloWorldService Implementation

I created a HelloWorldServiceImpl-Class in `src/main/java` that extended the auto-generated `HelloWorldServiceGrpc.HelloWorldServiceImplBase`

Then put the code from the tutorial:

```
@Override
    public void hello(Hello.HelloRequest request, StreamObserver<Hello.HelloResponse>
responseObserver) {
        System.out.println(
            "Handling hello endpoint: " + request.toString());

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

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

Explanation: This method handles incoming requests, processes the input, and sends back the response.

4. HelloWorldServer Implementation

I created another java-class called `HelloWorldServer` and put the given code from the tutorial:

```

public class HelloWorldServer {
    private static final int PORT = 50051;
    private Server server;

    public void start() throws IOException {
        server = ServerBuilder.forPort(PORT)
            .addService(new HelloWorldServiceImpl())
            .build()
            .start();
    }

    public void blockUntilShutdown() throws InterruptedException {
        if (server == null) {
            return;
        }

        server.awaitTermination();
    }

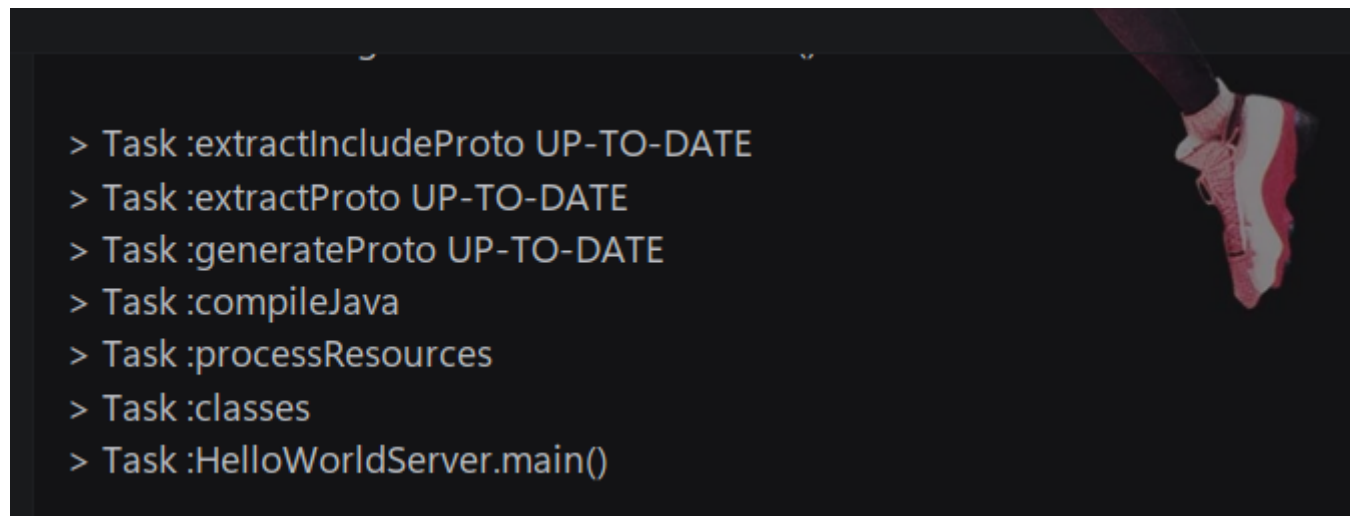
    public static void main(String[] args)
        throws InterruptedException, IOException {
        HelloWorldServer server = new HelloWorldServer();
        server.start();
        server.blockUntilShutdown();
    }
}

```

Explanation: This code starts a gRPC server on port 50051, adds the `HelloWorldServiceImpl`, and keeps the server running until terminated.

The next Step is to import the needed Classes, then execute `./ gradlew build` again

Now run it, when no errors occurs then it works



```

> Task :extractIncludeProto UP-TO-DATE
> Task :extractProto UP-TO-DATE
> Task :generateProto UP-TO-DATE
> Task :compileJava
> Task :processResources
> Task :classes
> Task :HelloWorldServer.main()

```

5. HelloWorldClient Implementation

Implement a gRPC server and gRPC client in a programming language of your choice

Customize the service so that a simple ElectionData record can be transferred

-> extend the proto-file with that code:

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

Explanation: This step updates the data structure for more detailed requests and ensures changes are reflected in generated code.

then execute `./ gradlew build` again

Change the text String in the Class "HelloWorldServiceImpl" to this: `String text = "Hello World, "+request.getFirstname()+" "+ request.getLastname() ;`

Explanation: This step updates the response logic to include first and last names.

Now create the `HelloWorldClient` -Class with this code:


```

import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;

public class HelloWorldClient {
    public static void main(String[] args) {

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

        HelloWorldServiceGrpc.HelloWorldServiceBlockingStub stub =
            HelloWorldServiceGrpc.newBlockingStub(channel);

        Hello.HelloResponse helloResponse = stub.hello(Hello.HelloRequest.newBuilder()
            .setFirstname("Max")
            .setLastname("Mustermann")
            .build());
        System.out.println(helloResponse.getText());
        channel.shutdown();

    }
}

```

Explanation: This client connects to the server, sends a request with first and last names, and prints the server's response.

Now run the Client, this will be the output if everything works:

```

> Task :HelloWorldClient.main()
Hello World, Max Mustermann

```