1. Goal of the Assignment

The goal of this assignment was to create a simple gRPC "HelloWorld" project.
The main purpose was to understand how a client and a server can communicate with each other using the gRPC framework.
A proto file had to be created to define the data structures, and then a server and a client had to be implemented using this interface.
Finally, all steps and theory questions about gRPC had to be documented.

2. Project Setup

I used Java as the programming language and built the project using Gradle.
The development environment was IntelliJ IDEA.
gRPC and Protocol Buffers were added using the Gradle build system.
With this setup, the environment was ready for implementing an RPC system.

3. Creating the Proto File

The first step was creating the file **hello.proto**.
This file defines the interface between the client and the server.
I created a service called "HelloWorldService" with one method called "hello".
There are also two messages:

- "HelloRequest" (containing firstname and lastname)
- "HelloResponse" (containing the response text)

This file is the foundation for the automatic code generation.

4. Generating the gRPC Classes

The command **gradle clean build** automatically generated all necessary Java classes from the proto file.
These generated classes include the client stub and the abstract server service class.
Without these classes, the client and server would not be able to communicate.

5. Implementation of the gRPC Server

I created a server class called **HelloWorldServer**.
The server runs on port **50051**.
Then I wrote the class **HelloWorldServiceImpl**, which implements the "hello" method.
The server receives the request, combines the firstname and lastname into a message, and sends the response back.
Once started, the server waits continuously for incoming requests.

6. Implementation of the gRPC Client

The client was implemented in the file **HelloWorldClient.java**.
The client connects to the server and sends a **HelloRequest** message.
Then it calls the “hello” method and prints the server’s response to the console.
This allowed me to test whether the RPC call works correctly.

7. Execution & Test Result

First I started the server.
The console showed that the HelloWorld Service is running.
Then I started the client.
The client successfully received the message:

“Hello World, Mohamed Hasan Alabed”

This confirmed that the connection was successful and the RPC call worked properly.

8. Problems During Development

At the beginning, the client did not receive an answer because the server was not started yet.
After starting the server first, everything worked.
Another issue was that the gRPC classes were not recognized.
This happened because the proto file needed to be compiled again.
Running **gradle clean build** solved the problem.

9. Theory Questions

1. What is gRPC and why does it work across multiple languages?
gRPC is a remote procedure call framework by Google.
It works in many programming languages because Protocol Buffers generate matching code for each language.

2. Describe the flow of an RPC call.
The client calls a method on the stub.
The data is serialized and sent over the network to the server.
The server receives the request, processes it, creates a response, and sends it back to the client.
The client receives the response and displays it.

3. How do Protocol Buffers work?
A proto file is created containing messages and services.
A compiler generates data classes from it.
These classes are used by the client and the server to send and receive messages.

4. Advantages of Protocol Buffers
They are very fast, require little storage space, and are more efficient than JSON or XML.
They are type-safe and work in many programming languages.

5. When should Protocol Buffers not be used?
When humans need to read the data, because Protobuf is binary and not human-readable.
Also, web browsers without gRPC-Web cannot process it directly.

6. Three example data types
string, int32, double.

10. Conclusion

I successfully implemented a working gRPC HelloWorld project.
The server and client were able to communicate with each other.
I learned how to create proto files, generate gRPC classes, and how an RPC call works technically.
All requirements of the basic task were fulfilled.