1. Ziel der Aufgabe
   In dieser Aufgabe sollte ich ein einfaches gRPC „HelloWorld“-Projekt erstellen. Dabei ging es darum, zu verstehen, wie ein Client und ein Server über das gRPC-Framework miteinander kommunizieren können. Mithilfe einer Proto-Datei sollten die Datenstrukturen definiert werden, und danach sollte ich einen Server und einen Client implementieren, die diese Schnittstelle nutzen. Anschließend musste ich alle Schritte dokumentieren und Fragen zu gRPC beantworten.

2. Projekt Setup
   Ich habe Java als Programmiersprache verwendet und das Projekt mit Gradle aufgebaut. Als Entwicklungsumgebung habe ich IntelliJ IDEA benutzt. gRPC und Protocol Buffers wurden über das Gradle-Build-System eingebunden. Damit war die Umgebung vollständig vorbereitet, um ein RPC-System zu erstellen.

3. Erstellung der Proto-Datei
   Der erste praktische Schritt war die Erstellung der Datei „hello.proto“. Diese Datei beschreibt die Schnittstelle zwischen Client und Server. Ich habe einen Service mit dem Namen „HelloWorldService“ erstellt und eine Methode namens „hello“ definiert. Außerdem wurden zwei Nachrichten beschrieben: „HelloRequest“ (mit Vor- und Nachname) und „HelloResponse“ (mit dem Rückgabetext). Diese Datei dient als Grundlage für die automatische Codegenerierung.

4. Generierung der gRPC-Klassen
   Mit dem Befehl „gradle clean build“ wurden automatisch alle benötigten Java-Klassen aus der Proto-Datei erzeugt. Diese generierten Klassen enthalten den Stub für den Client und die abstrakte Serviceklasse für den Server. Ohne diesen Schritt könnten Client und Server nicht miteinander kommunizieren.

5. Implementierung des gRPC-Servers
   Ich habe eine Server-Klasse namens „HelloWorldServer“ erstellt. Der Server startet auf Port 50051. Danach habe ich die Klasse „HelloWorldServiceImpl“ geschrieben, die die Methode „hello“ implementiert. Der Server empfängt den Request, setzt die beiden Namen zusammen und schickt eine Antwort zurück. Sobald der Server läuft, wartet er dauerhaft auf eingehende Client-Anfragen.

6. Implementierung des gRPC-Clients
   Der Client wurde in der Datei „HelloWorldClient.java“ implementiert. Der Client stellt eine Verbindung zum Server her und sendet eine HelloRequest-Nachricht. Anschließend ruft er die Methode „hello“ auf und gibt die Antwort vom Server in der Konsole aus. So konnte ich testen, ob der RPC-Aufruf funktioniert.

7. Ausführung & Testergebnis
   Zuerst habe ich den Server gestartet. In der Konsole erschien die Meldung, dass der HelloWorld Service läuft. Danach habe ich den Client gestartet. Der Client hat erfolgreich die Nachricht erhalten:

„Hello World, Mohamed Hasan Alabed“

Damit war klar, dass die Verbindung funktioniert und der RPC richtig ausgeführt wurde.

8. Probleme während der Entwicklung
   Während der Umsetzung gab es ein paar kleine Schwierigkeiten. Am Anfang erhielt der Client keine Antwort, weil der Server noch nicht gestartet war. Nachdem ich die Startreihenfolge beachtet habe, funktionierte alles. Ein weiteres Problem war, dass die gRPC-Klassen nicht erkannt wurden. Dies lag daran, dass die Proto-Datei erst wieder kompiliert werden musste. Durch „gradle clean build“ wurde das Problem schnell behoben.

9. Theoriefragen

10. Was ist gRPC und warum funktioniert es über verschiedene Programmiersprachen?
    gRPC ist ein Remote-Procedure-Call-Framework von Google. Es funktioniert in vielen Programmiersprachen, weil Protocol Buffers automatisch passenden Code für jede Sprache generiert. Dadurch können verschiedene Systeme problemlos miteinander kommunizieren.

11. Beschreibe den Ablauf eines RPC-Aufrufs.
    Der Client ruft eine Methode am Stub auf. Die Daten werden serialisiert und über das Netzwerk an den Server geschickt. Der Server erhält die Anfrage, verarbeitet sie, erstellt eine Antwort und schickt sie zurück. Der Client bekommt die Antwort und zeigt sie an.

12. Wie funktioniert Protocol Buffers?
    Man schreibt eine Proto-Datei, in der die Nachrichten und Services definiert werden. Mit einem Compiler werden daraus Datenklassen erzeugt. Diese Klassen werden von Client und Server genutzt, um Nachrichten zu versenden und zu empfangen.

13. Vorteile von Protocol Buffers
    Es ist sehr schnell, benötigt nur wenig Speicherplatz und ist viel effizienter als JSON oder XML. Außerdem ist es typensicher und kann in vielen Programmiersprachen genutzt werden.

14. Wann sollte man Protocol Buffers nicht verwenden?
    Wenn Menschen die Daten lesen müssen, ist Protobuf ungeeignet, weil es ein Binärformat ist. Auch Webbrowser ohne gRPC-Web-Unterstützung können es nicht direkt verarbeiten.

15. Drei Beispiel-Datentypen
    string, int32, double.

16. Fazit
    Ich habe erfolgreich ein funktionierendes gRPC HelloWorld-Projekt umgesetzt. Der Server und der Client konnten miteinander kommunizieren. Ich habe gelernt, wie man Proto-Dateien erstellt, wie man gRPC-Klassen generiert und wie ein RPC-Aufruf technisch funktioniert. Alle Anforderungen der Grundlagenaufgabe wurden erfüllt.