MidEng 7.2 gRPC Framework (Grundlagen plus Vertiefung)-4hAbschlussbedingungen

1. Theoretische Grundlagen

- **Was ist gRPC?** gRPC ist ein Framework für Remote Procedure Calls (RPC), mit dem Funktionen auf entfernten Rechnern so aufgerufen werden können, als wären sie lokal vorhanden1. Die Plattformunabhängigkeit wird durch **Protocol Buffers (Protobuf)** erreicht: Alle Services werden neutral beschrieben, und der Compiler erzeugt automatisch den passenden Code für Sprachen wie Java, Python oder Go2.
- **Der RPC-Lebenszyklus:**
  1. Der Client ruft eine Methode des lokalen Stubs auf3.
  2. Der Stub verpackt die Daten in eine Protobuf-Nachricht4.
  3. Die Nachricht wird via **HTTP/2** an den Server übertragen5.
  4. Der Server verarbeitet die Anfrage und sendet die Antwort zurück6.
  5. Der Client empfängt und nutzt das Ergebnis7.
- **Protocol Buffers Workflow & Vorteile:** Der Workflow umfasst die Definition in einer `.proto`-Datei, die Codegenerierung mit `protoc` und die Nutzung der Klassen in der Applikation8. Vorteile sind die hohe Geschwindigkeit, geringe Datenmengen und klar definierte Schnittstellen9.
- **Wann sollte man es nicht nutzen?** Wenn Menschenlesbarkeit (wie bei JSON) für das Debugging wichtig ist, bei direktem Datenaustausch im Browser oder bei ständig wechselnden Datenformaten10.
- **Datentypen:** Typische Beispiele sind `string`, `int32` und `bool`11.

---

2. Projektstruktur & Implementierung

Das Projekt ist als polyglottes System aufgebaut (Java Server, Java/Python Clients).

2.1 Service Definition (`hello.proto`)

Die Grundlage bildet die Datei `hello.proto`. Hier wurden neben dem Standard-Service auch die Strukturen für das DataWarehouse definiert121212:

Protocol Buffers

```
service HelloWorldService {
  rpc hello (HelloRequest) returns (HelloResponse);
  rpc sendData (DataRecord) returns (DataResponse); // Erweiterung
}

message DataRecord {
  int32 id = 1;
  string name = 2;
  double value = 3;
}
```

2.2 Server (Java)

Der Server läuft auf Port `50051` und nutzt den `HelloWorldServiceImpl` zur Bearbeitung der Anfragen13131313.

- **Logik:** In `sendData` empfängt der Server einen `DataRecord`, loggt diesen in der Konsole und sendet eine Bestätigung inklusive der extrahierten Daten zurück14.

2.3 Clients (Java & Python)

- **Java Client:** Baut eine Verbindung via `ManagedChannel` auf und nutzt einen `BlockingStub`, um Daten wie "Max Mustermann" oder Sensordaten zu senden151515151515151515.
- **Python Client (Vertiefung):** Zur Demonstration der Sprachübergreifung wurde in `src/main/resources/data_client.py` ein Python-Client implementiert1616. Dieser erstellt einen `DataRecord` (z. B. "Weather Station B") und ruft den RPC-Dienst des Java-Servers auf17171717.

---

3. Ausführung und Ergebnisse

Start des Servers

Der Server wird gestartet und wartet auf Verbindungen:

> `HelloWorld Service is running!` 18

Test mit Java Client

Ausgabe des Servers bei Empfang:

> `Received DataRecord with ID=1, Name=Temperature Sensor A, Value=21.7` 19

Test mit Python Client

Der Python-Client wird über das Terminal aufgerufen:

python3 src/main/resources/data_client.py 20

Ausgabe:

> `Server response: Received DataRecord with ID=42, Name=Weather Station B, Value=18.9` 21
