gRPC

Creating .proto file

```
syntax = "proto3";
option java_package = "warehouseProto";
```

Sets the proto version to 3 (Default is 2)

Generates the Files for java in the warehouseProto package

```
service Warehouse {
  rpc getData (WarehouseRequest) returns (WarehouseData) {}
}
```

Define the name of the Service and the methods that should be implemented

```
message WarehouseRequest {
  string id = 1;
}

message Product {
  string id =  1;
  string name =  2;
  string category = 3;
  string amount = 4;
  string unit = 5;
}

message WarehouseData {
  string id =  1;
  string name =  2;
  string timestamp =  3;
  string street =  4;
  string city =  5;
```

```
  string country =  6;
  string plz =  7;
  repeated Product product_data =  8;
}
```

Define the Request with the id of the Warehouse and the WarehouseData and Product Objects

Arrays or Lists are stated as repeaded <Type>

Creating Java Server

Creating the Project

```
mvn -B archetype:generate -DgroupId=at.czlabinger -DartifactId=
```

This creates a new maven project

Setting up maven

Adding dependency for some Annotations that gRPC needs

```
<dependency>
  <groupId>javax.annotation</groupId>
  <artifactId>javax.annotation-api</artifactId>
  <version>1.2</version>
</dependency>
```

Adding dependencies for gRPC

```
<!-- gRPC dependencies -->
        <dependency>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-netty-shaded</artifactId>
            <version>${grpc.version}</version>
        </dependency>
```

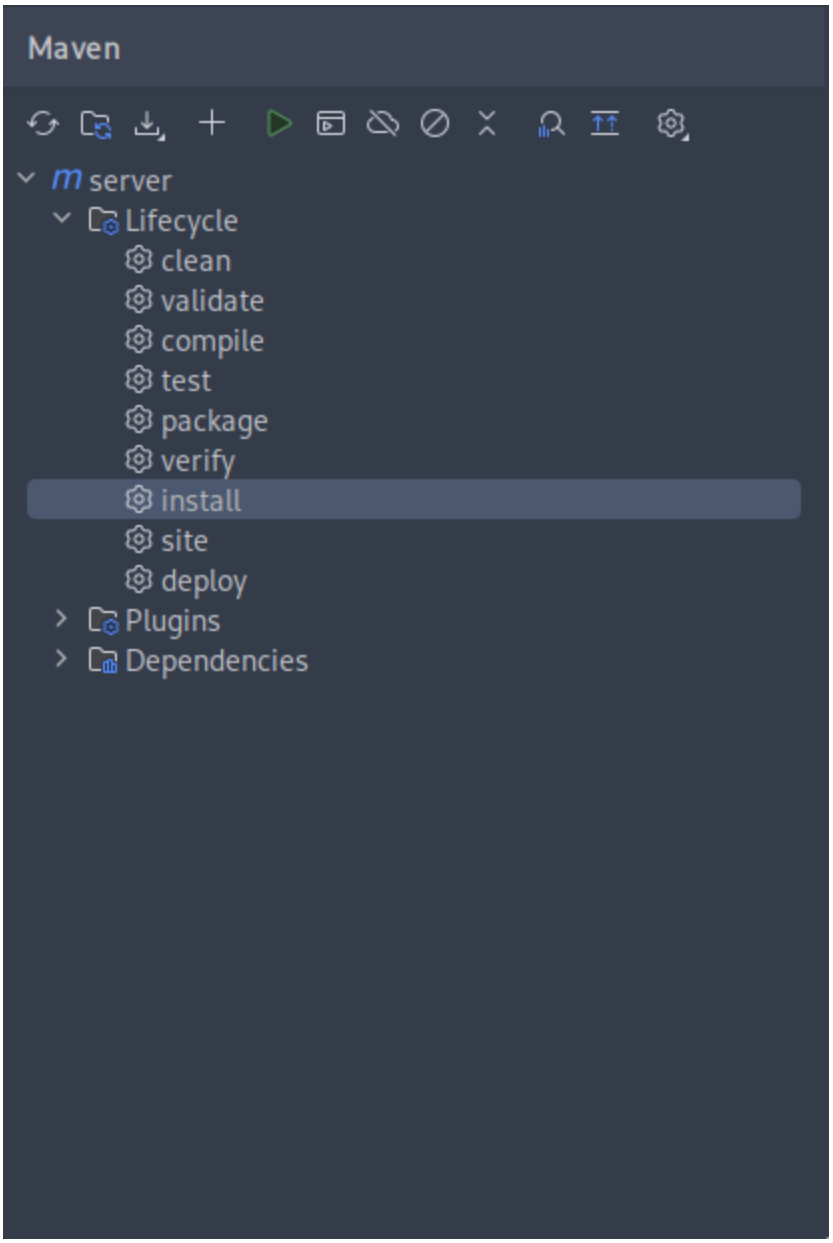
```
        <dependency>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-protobuf</artifactId>
            <version>${grpc.version}</version>
        </dependency>

        <dependency>
            <groupId>io.grpc</groupId>
            <artifactId>grpc-stub</artifactId>
            <version>${grpc.version}</version>
        </dependency>
```

Adding Lifecycle for generation of sources

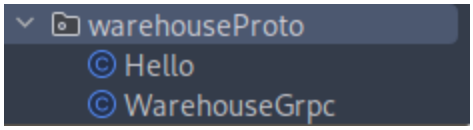
```
<!-- compile proto file into java files. -->
<plugin>
  <groupId>com.github.os72</groupId>
  <artifactId>protoc-jar-maven-plugin</artifactId>
  <version>3.6.0.1</version>
  <executions>
    <execution>
      <phase>generate-sources</phase>
      <goals>
        <goal>run</goal>
      </goals>
      <configuration>
        <includeMavenTypes>direct</includeMavenTypes>

        <inputDirectories>
          <include>/home/stoffi05/Documents/School/4xHIT/SYT/DZS
        </inputDirectories>

        <outputTargets>
          <outputTarget>
            <type>java</type>
            <outputDirectory>src/main/java</outputDirectory>
```

```
            </outputTarget>
            <outputTarget>
              <type>grpc-java</type>
              <pluginArtifact>io.grpc:protoc-gen-grpc-java:1.15.0
              <outputDirectory>src/main/java</outputDirectory>
            </outputTarget>
          </outputTargets>
        </configuration>
      </execution>
    </executions>
  </plugin>
```

Running maven install to generate the sources



Sources are now generated



Implementing the Server

```
@Override
public void getData(Hello.WarehouseRequest request, StreamObse
    WarehouseSimulation simulation = new WarehouseSimulation(
    WarehouseData warehouseData = simulation.getData(request.g

    //Add WarehouseData
          //...

    Hello.WarehouseData serializedWarehouseData = warehouseDat

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

A new Warehouse gets generated and the data gets added to the response that gets send back to the client

Creating Python client

Setting up the sources

In the client directory run the command:

```
python -m grpc_tools.protoc -I/home/stoffi05/Documents/School/4x
```

Implementing the Client

```
    channel = grpc.insecure_channel('localhost:8999')
    stub = hello_pb2_grpc.WarehouseStub(channel)

    # Create a HelloRequest instance and set the id field
    request = hello_pb2.WarehouseRequest(id="001")

    # Pass the request instance to the sayHello method
```

```
    response = stub.getData(request)
    print(response)
```

A connection to the server via localhost:8999 is created and a WarehouseRequest is created with id=001. Then the request gets send to the getData method of the server.

Questions

- What is gRPC and why does it work accross languages and platforms?
  - high-performance
  - open-source framework
  - developed by Google
  - enables communication between services using HTTP for transport and Protocol Buffers (Protobuf) as the interface definition language
  - It is language-agnostic, meaning it allows developers to define services and message types in .proto files, which can then be compiled into source code for any of the supported languages
- Describe the RPC life cycle starting with the RPC client?
  - Unary RPC: The client sends a single request and waits for a single response.
  - Server streaming RPC: The client sends a request and receives a stream of responses from the server.
  - Client streaming RPC: The client sends a stream of requests and waits for a single response from the server.
  - Bidirectional streaming RPC: Both the client and server send streams of messages to each other concurrently
- Describe the workflow of Protocol Buffers?
  - It allows you to define simple data structures in a special format, which can then be compiled into source code for multiple languages

- What are the benefits of using protocol buffers?
  - Efficiency: Protobuf serializes data into a binary format, which is more compact and faster to process than text-based formats like JSON.
  - Language-agnosticism: Protobuf allows for code generation in multiple languages, facilitating interoperability across different technology stacks.
  - Strong typing: Protobuf enforces strong typing, which helps prevent errors and mismatches when data is exchanged between services.
  - Extensibility: You can add new fields or methods to your services and messages without breaking existing client code, which is particularly useful in distributed systems.
  - Code generation: The Protobuf compiler generates client and server code from .proto files, reducing manual coding effort and ensuring consistency
- When is the use of protocol not recommended?
  - Human readability of the data format is a priority, as Protobuf is a binary format.
  - The data structures are not well-defined or frequently changing, as Protobuf requires a schema definition.
  - The system does not require the efficiency gains provided by binary serialization, such as in simple applications or when the data size is small.
- List 3 different data types that can be used with protocol buffers?
  - `int32`
  - `string`
  - `message`

Sources

Maven Setup: https://medium.com/@lucian.ritan/setup-and-run-a-grpc-project-eda408c8cef0