gRPC Resource :

1. What is RPC? gRPC Introduction : https://www.youtube.com/watch?v=gnchfOojMk4
2. gRPC vs REST Performance Comparison : https://www.vinsguru.com/grpc-vs-rest-performance-comparison/
3. Basic tutorial : https://www.youtube.com/watch?v=2CWYorTWyGs&t=488s
4. udemy tutorial_ vinsguru : https://drive.google.com/drive/folders/1GkjRukgV5hn2glM4Bq6QzhrunjZ9YINE?usp=sharing
   (Udemy) vinsguru code example : https://github.com/vinsguru/vinsguru-blog-code-samples/tree/master/grpc
   gRPC Spring Boot Integration Articale : https://www.vinsguru.com/grpc-spring-boot-integration/
5. gRPC file upload : https://www.vinsguru.com/grpc-file-upload-client-streaming/
6. grpc-spring-boot-starter : https://github.com/yidongnan/grpc-spring-boot-starter/tree/master
7. https://github.com/Sayem-Hasnat/gRPC-SpringBoot-Practise/tree/dev
8. bloomrpc (grpc api tester) : https://github.com/bloomrpc/bloomrpc/releases

gRPC with springboot microservice Setup

first have to create a module maven project

proto-module

It will contains the protocol buffers file (.proto) where request , response and fields will be written in **Protobuf** language .

Step 01 : Add Dependencies on **pom.xml** file

```
<name>message-common</name>
 <properties>
    <protobuf.version>3.14.0</protobuf.version>
    <protobuf-plugin.version>0.6.1</protobuf-plugin.version>
    <grpc.version>1.35.0</grpc.version>
    <java.version>11</java.version>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
 </properties>

 <dependencies>
    <dependency>
```

```
        <groupId>io.grpc</groupId>
        <artifactId>grpc-stub</artifactId>
        <version>1.35.0</version>
      </dependency>
      <dependency>
        <groupId>io.grpc</groupId>
        <artifactId>grpc-protobuf</artifactId>
        <version>1.35.0</version>
      </dependency>
      <dependency>
        <!-- Java 9+ compatibility - Do NOT update to 2.0.0 -->
        <groupId>jakarta.annotation</groupId>
        <artifactId>jakarta.annotation-api</artifactId>
        <version>1.3.5</version>
        <optional>true</optional>
      </dependency>
    </dependencies>
<plugins>
        <plugin>
          <groupId>org.xolstice.maven.plugins</groupId>
          <artifactId>protobuf-maven-plugin</artifactId>
          <version>0.6.1</version>
          <configuration>

<protocArtifact>com.google.protobuf:protoc:${protobuf.version}:exe:linux-x86_64</p
rotocArtifact>
              <pluginId>grpc-java</pluginId>

<pluginArtifact>io.grpc:protoc-gen-grpc-java:${grpc.version}:exe:linux-x86_64</plugi
nArtifact>
          </configuration>
          <executions>
            <execution>
              <goals>
                <goal>compile</goal>
                <goal>compile-custom</goal>
              </goals>
            </execution>
          </executions>
        </plugin>
      </plugins>
```

Step 02 : Add **service.proto** in **src/main/proto** packege file

```
syntax = "proto3";

option java_multiple_files = true;
option java_package = "com.hasnat.proto.bankservice";

// client will request by account number on this format to get Balance
message BalanceRequest{
  int32 account_number = 1;
}

//server will response this
message Balance {
  int32  amount = 1;
  int32 account_number = 2;
}
// client will request
message WithdrawRequest{
  int32 account_number = 1;
  int32  requested_amount = 2;
    }

//server will response this
   message Money{
  int32 withdrawal_money = 1;
  int32 availableBalance = 2;
    }

service  BankService{
  //unary service
  rpc getBalance(BalanceRequest) returns (Balance);

  //server side streaming service
  rpc withdraw(WithdrawRequest) returns (stream Money);

}
```

After add the proto file this proto-module need to clean build and add the module on project modules where grpc server and client service .

Step 03 : Add **proto-module** in **the pom.xml** of other microservices

```
<!--greeting message module-->
                <dependency>
                        <groupId>com.hasnat</groupId>
                        <artifactId>message-common</artifactId>
                        <version>0.0.1-SNAPSHOT</version>
                </dependency>
```

Step 04 : Add **grpc client** & **grpc service** dependencies

```
<dependency>
                  <!-- gRPC client dependency -->
                        <groupId>net.devh</groupId>
                        <artifactId>grpc-client-spring-boot-starter</artifactId>
                        <version>2.12.0.RELEASE</version>
                </dependency>
                <!-- gRPC server dependency -->
                <dependency>
                        <groupId>net.devh</groupId>
                        <artifactId>grpc-server-spring-boot-starter</artifactId>
                        <version>2.12.0.RELEASE</version>
</dependency>
```

gRPC server module Setup

Step 01 : Add configuration in **application.properties**

```
spring.application.name=grpc-server-local
grpc.server.port=9898
server.port=8083

grpc.client.grpc-client-local.address=static://127.0.0.1:8085
grpc.client.grpc-client-local.enableKeepAlive=true
grpc.client.grpc-client-local.keepAliveWithoutCalls=true
#grpc.client.grpc-server-local.negotiationType=plaintext
grpc.client.grpc-client-local.negotiationType=PLAINTEXT
```

Step 02 : Add grpc-client service in service class

```
@GrpcService
public class BankService extends BankServiceGrpc.BankServiceImplBase {

    //Unary RPC
    @Override
    public void getBalance(BalanceRequest balanceRequest, StreamObserver<Balance>
responseObserver) {
        int accountNumber = balanceRequest.getAccountNumber();
        Balance balance = Balance.newBuilder()
                .setAmount(BankAccountDB.getBalance(accountNumber))
                .setAccountNumber(accountNumber)
                .build();

        BankAccountDB.printAccountDetails();
        responseObserver.onNext(balance);
        responseObserver.onCompleted();
    }

    //Client-streaming RPC
    @Override
    public void withdraw(WithdrawRequest request, StreamObserver<Money>
responseObserver) {
        int accountNumber = request.getAccountNumber();
        int withdrawalMoney = request.getRequestedAmount();
        int balance = BankAccountDB.getBalance(accountNumber);


        // gRPC error response Handling
        if (balance<withdrawalMoney){
            Status status = Status.FAILED_PRECONDITION
                    .withDescription("Not enough money, account have: "+ balance);
            responseObserver.onError(status.asRuntimeException());
            return;
        }
        for (int i = 0; i < (withdrawalMoney / 10); i++) {
             BankAccountDB.deductBalance(accountNumber, 10);
            Money money  = Money.newBuilder()
                    .setWithdrawalMoney(withdrawalMoney)

.setAvailableBalance(BankAccountDB.getBalance(request.getAccountNumber()))
                    .build();
            responseObserver.onNext(money);

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

gRPC client module Setup

Step 01 : Add configuration in **application.properties**

```
spring.application.name=grpc-client-local
server.port=8081
grpc.server.port=8085


grpc.client.grpc-server-local.address=static://127.0.0.1:9898
grpc.client.grpc-server-local.enableKeepAlive=true
grpc.client.grpc-server-local.keepAliveWithoutCalls=true
#grpc.client.grpc-server-local.negotiationType=plaintext
grpc.client.grpc-server-local.negotiationType=PLAINTEXT
```

Step 02 : Create Controller for grpc-client service

```
@RestController
@RequestMapping("/")
public class BankServiceController {
    @Autowired
    private BankService bankService;

    @GetMapping(value = "client/{accountNumber}")
    public BalanceResponse getBalance(@PathVariable int accountNumber) {
        return bankService.getBalance(accountNumber);
    }

    @PostMapping(value = "client/withdraw")
    public void withdrawRequest(@RequestBody ClientWithdrawRequest
withdrawRequest) {
         bankService.withdrawRequest(withdrawRequest);
    }
}
```

By Sayem Hasnat

Step 03 : Add grpc-client service in service class

```
@Service
public class BankService {
    @GrpcClient("grpc-server-local")
    private BankServiceGrpc.BankServiceBlockingStub bankServiceBlockingStub;
// blockingStub used for Uniray call
    @GrpcClient("grpc-server-local")
    private BankServiceGrpc.BankServiceStub bankServiceStub; // Stub used for async
streaming call

    //check unary RPC
    public BalanceResponse getBalance(int accountNumber) {
        BalanceRequest balanceRequest = BalanceRequest.newBuilder()
                .setAccountNumber(accountNumber)
                .build();
        final Balance balance = this.bankServiceBlockingStub.getBalance(balanceRequest);
        System.out.println("balance" + balance);
        BalanceResponse balanceResponse = new
BalanceResponse(balance.getAccountNumber(),balance.getAmount());
        return balanceResponse;
    }

    public void withdrawRequest(ClientWithdrawRequest clientWithdrawRequest){

        WithdrawRequest withdrawRequestToServer =
                WithdrawRequest.newBuilder()
                        .setAccountNumber(clientWithdrawRequest.getAccountNumber())
                        .setRequestedAmount(clientWithdrawRequest.getAmount())
                        .build();
        this.bankServiceBlockingStub.withdraw(withdrawRequestToServer)
                .forEachRemaining(m -> System.out.println(m + "localTime "+
LocalTime.now()));
        CountDownLatch latch = new CountDownLatch(1);
        this.bankServiceStub.withdraw(withdrawRequestToServer, new
MoneyStreamObserver());
    }
}
```