3 - [Hands-On] gRPC Project Overview & Setup

Go Dependencies Setup

```
$ go get -u google.golang.org/grpc
```

```
$ go get -u github.com/golang/protobuf/protoc-gen-go
```

Code Generation Test

/src/calculator
/src/greet

src/greet/greetpb/greet.proto

```
syntax = "proto3";

package greet;
option go_package = "greetpb";

service GreetService {}
```

/src/generate.sh

```
$ protoc greet/greetpb/greet.proto --go_out=plugins=grpc:.
```

Server Setup Boilerplate Code

gRPC Server Setup

- Let's setup a gRPC Server with no service on it
- We'll see how to properly start & stop the Server on a defined port
- The point of this lecture is just to be done with "boilerplate code"

/src/greet/greet_server/server.go

```
package main

import (
    "fmt"
    "log"
    "net"

```

```
    "github.com/gRPC-go-microservices/src/greet/greetpb"
    "google.golang.org/grpc"
)

type server struct{}

func main() {
    fmt.Println("Hello world!")

    // listener
    // 50051 is the default port number for gRPC
    lis, err := net.Listen("tcp", "0.0.0.0:50051")
    if err != nil {
        log.Fatalf("Failed to listen: %v", err)
    }

    // create grpc server
    s := grpc.NewServer()
    greetpb.RegisterGreetServiceServer(s, &server{})

    // bind the server to the port
    if err := s.Serve(lis); err != nil {
        log.Fatalf("failed to serve: %v", err)
    }
}
```

Client Setup Boilerplate Code

gRPC Client Setup

- Let's setup a gRPC Client that connects to our Server
- We'll see how to properly start & stop the Client
- The point of this lecture is just to be done with "boilerplate code"

/src/greet/greet/greet_client/client.go

```
package main

import (
    "fmt"
    "log"

    "github.com/gRPC-go-microservices/src/greet/greetpb"
    "google.golang.org/grpc"
)

func main() {
    fmt.Println("Hello I'm a client")

    // by default gRPC has SSL, for now, without this
    conn, err := grpc.Dial("localhost:500051", grpc.WithInsecure())
    if err != nil {
        log.Fatalf("could not connect: %v", err)
    }
```

```
19
20      // defer means defer this statement at the very end of this function
21      defer conn.Close()
22
23      // create a new client
24      c := greetpb.NewGreetServiceClient(conn)
25      fmt.Printf("Created client: %f", c)
26  }
```