# GRPC - Protobuf

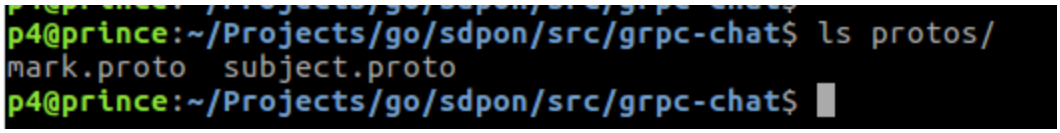
## Protobuf

Ref: https://developers.google.com/protocol-buffers/docs/proto3
https://dev.to/techschoolguru/protocol-buffer-deep-dive-52d9

- Compiling protos for Grpc connection
  - protoc --go_out=plugins=grpc:pkg/ protos/*.proto

- Compile with import files
  - protoc --proto_path=protos --go_out=pkg/proto-pb/ protos/subject.proto protos/mark.proto
- Compile protos with import protos for grpc connection
  - protoc --proto_path=protos --go_out=plugins=grpc:pkg/proto-pb/ protos/subject.proto protos/mark.proto

- Protos created:



- File **protos/subject.proto** :

```
/*
 This is a block comment.
*/

// This is one line comment.

/*
  'syntax' should be the first uncommented line of a proto file.
  If you did not specify 'syntax', by default it will take 'proto2'.
*/
syntax = "proto3";

/*
  'package' keyword is used to avoid name collision.
  You cannot have different package names in multiple files in same folder unless you
  are compiling a single proto file.
*/
package student;

// This will tag the '.go' generated files with mentioned package name which is 'pb' here.
option go_package="proto-pb";

// 'enum' can be used to predefine values.
enum Subject{
  PHYSICS = 0;          // enum values should always start with '0'
  CHEMISTRY = 1;
  MATHS = 2;
}

/*
 'message' keyword is used to define a message which will be in turn used as 'request' or 'response'.
 'Names' for 'message' will be in the standard of 'PascalCase' letters.
 The field numbers '1,2,3...' represents the index of values appearing in the encoded streams.
 It can also be represented in '1,3,2..', '1, 3,4' but not '1,2,2...'.

 Field numbers '1-15' take 1 byte to encode, where as field numbers '16-2047' takes 2 bytes.
 So, you should reserve the numbers 1 through 15 for very frequently occurring message
 elements.
*/

message SubjectInfo{
  int32 sl_no = 1;       // Inside element names follow 'snake_case' standard.
  string name = 2;
  Subject subject = 3; // You can directly use outer defined enum values or you can define the
                       // enum values internal to the message.
}
```

- File **protos/mark.proto** :

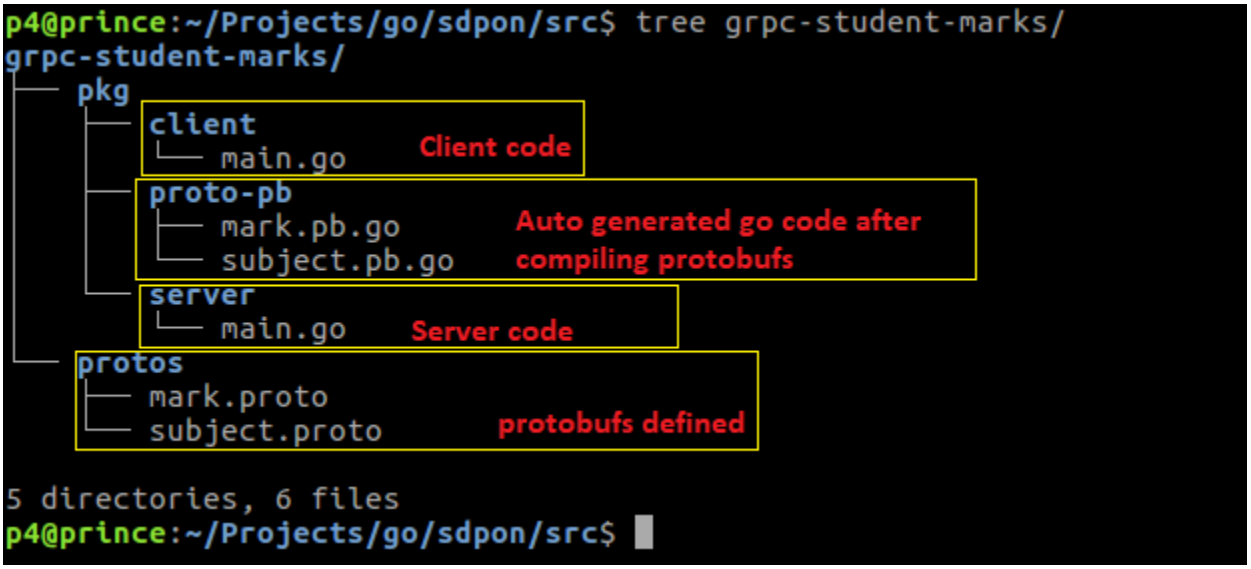
```
syntax = "proto3";

package student;

option go_package = "proto-pb";

/*
  You can import messages from other protos using 'import' and with or without 'alias'
*/
import "subject.proto";

message MarkReq{
  SubjectInfo subject_info = 1; // You can reuse the predefined common message structure anytime.
}

message MarkResp{
  SubjectInfo subject_info = 1;
  int32 mark = 2;                    // There were 'required' and 'optional' parameters which can be set for fields
                                     // on 'proto2' but it is removed in 'proto3'.  'required' is removed due to
                                     // backward compatibility issues and 'optional' is removed because without
                                     // 'required', 'optional' doesn't make sense.
}

message AllMarksResp{
  int32 sl_no = 1;
  string name = 2;
  message MarkInfo{              // You can define a message inside another message.
    Subject subject_name = 1;
    int32 mark = 2;
  }
  repeated MarkInfo marks = 3; // Protobuf supports 'repeated' keyword to create slice of values.
}

/*
  'service' will help to establish a grpc channel and pass values between
  the microservices (client and server) with the rpc communication.
*/
service StudentService{

  rpc GetMark (MarkReq) returns (MarkResp);          // Each message call invoked by a client will be part of rpc
                                                     // methods. It contains a request message and return
                                                     // message (response).

  rpc GetAllMarks (MarkReq) returns (AllMarksResp); // We can define multiple rpc calls on single service.

}
```

**GRPC Tutorial**

**Ref**:

- https://www.grpc.io/docs/languages/go/
- https://github.com/grpc/grpc-go/blob/master/examples/route_guide/client/client.go
- https://github.com/grpc/grpc-go/blob/master/examples/route_guide/server/server.go

- **Directory structure**:




- **Run the code** :
  - Run the server code:

    $ go run pkg/server/main.go

  - Run the client code:

    $ go run pkg/client/main.go

- File **pkg/client/main.go** :

```
package main

import (
	"context"
	"flag"
	"fmt"
	pbproto "grpc-student-marks/pkg/proto-pb"
	"log"
	"time"

	"github.com/golang/protobuf/proto"
	"google.golang.org/grpc"
)

// Ref: https://github.com/grpc/grpc-go/blob/master/examples/route_guide/client/client.go

var (
	serverAddr = flag.String("server_addr", "localhost:10000", "The server address in the format of
host:port")
)

// Invoking rpc call towards the server
func sendGetAllMarksReq(ctx context.Context, client pbproto.StudentServiceClient) {
	fmt.Println("\n>>>>>>>>>>>>>>>.sendGetAllMarksReq")
	req := new(pbproto.MarkReq)
	subjectInfo := new(pbproto.SubjectInfo)
	subjectInfo.SlNo = 123
	subjectInfo.Name = "Prince Pereira"
	req.SubjectInfo = subjectInfo
	btes, _ := proto.Marshal(req)
	fmt.Printf("\nMarshalled Data : %v\n", btes)
	fmt.Printf("Get-all-marks-for-Prince : %v\n", req)
	resp, err := client.GetAllMarks(ctx, req)

	if err != nil {
		log.Fatalf("could not get all marks: %v", err)
	}

	log.Printf("\nAll-Marks-for-prince: %s\n", resp)
}

// Invoking rpc call towards the server
func sendGetMarkReq(ctx context.Context, client pbproto.StudentServiceClient) {
	fmt.Println("\n>>>>>>>>>>>>>>>.sendGetMarkReq")
	req := new(pbproto.MarkReq)
	subjectInfo := new(pbproto.SubjectInfo)
	subjectInfo.SlNo = 123
	subjectInfo.Name = "Prince Pereira"
	subjectInfo.Subject = pbproto.Subject_PHYSICS
	req.SubjectInfo = subjectInfo
```

```
	btes, _ := proto.Marshal(req)
	fmt.Printf("Marshalled Data : %v\n", btes)
	fmt.Printf("Get-mark-for-Prince : %v\n", req)
	resp, err := client.GetMark(ctx, req)

	if err != nil {
		log.Fatalf("could not get marks: %v\n", err)
	}

	log.Printf("Mark-for-prince: %s\n", resp)
}

func main() {
	var opts []grpc.DialOption
	opts = append(opts, grpc.WithInsecure())
	opts = append(opts, grpc.WithBlock())

	// Establishing the server connection
	conn, err := grpc.Dial(*serverAddr, opts...)
	if err != nil {
		log.Fatalf("did not connect: %v", err)
	}

	defer conn.Close()

	// Constructing a client object
	client := pbproto.NewStudentServiceClient(conn)
	ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
	defer cancel()
	if ctx == nil || client == nil {
		fmt.Println("Conext or client is nil")
	}

	sendGetMarkReq(ctx, client)
	sendGetAllMarksReq(ctx, client)
}
```

- File pkg/server/main.go :

```
package main

import (
	"context"
	"flag"
	"fmt"
	pbproto "grpc-student-marks/pkg/proto-pb"
	"log"
	"net"

	"google.golang.org/grpc"
)

// Ref: https://github.com/grpc/grpc-go/blob/master/examples/route_guide/server/server.go

var (
	port = flag.Int("port", 10000, "The server port")
)

// Server struct for invoking server calls
type Server struct {
}

// GetMark refers to the method getting called for RPC call invoked for Get Single Subject Mark.
func (server *Server) GetMark(ctx context.Context, req *pbproto.MarkReq) (*pbproto.MarkResp, error) {
	fmt.Printf("Request received for GetMark() request. Req : %v \n", req)
	resp := new(pbproto.MarkResp)

	subjectInfo := new(pbproto.SubjectInfo)
	subjectInfo.SlNo = req.SubjectInfo.SlNo
	subjectInfo.Name = req.SubjectInfo.Name
	subjectInfo.Subject = req.SubjectInfo.Subject
	resp.SubjectInfo = subjectInfo

	if req.SubjectInfo.Subject == pbproto.Subject_PHYSICS {
		resp.Mark = 98
	} else {
		resp.Mark = 97
	}
	fmt.Printf("Response send for GetMark() request. Req : %v , Resp : %v\n", req, resp)
	return resp, nil
}

// GetAllMarks refers to the method getting called for RPC call invoked for Get all Subject Marks.
func (server *Server) GetAllMarks(ctx context.Context, req *pbproto.MarkReq) (*pbproto.AllMarksResp, error) {
	fmt.Printf("Request received for GetAllMarks() request. Req : %v \n", req)
	resp := new(pbproto.AllMarksResp)
	resp.SlNo = req.SubjectInfo.SlNo
	resp.Name = req.SubjectInfo.Name
```

```
	var markInfo []*pbproto.AllMarksResp_MarkInfo

	phyMark := new(pbproto.AllMarksResp_MarkInfo)
	phyMark.SubjectName = 0
	phyMark.Mark = 98
	markInfo = append(markInfo, phyMark)

	mathsMark := new(pbproto.AllMarksResp_MarkInfo)
	mathsMark.SubjectName = 2
	mathsMark.Mark = 97
	markInfo = append(markInfo, mathsMark)

	resp.Marks = markInfo

	fmt.Printf("Response send for GetAllMarks() request. Req : %v , Resp : %v\n", req, resp)
	return resp, nil
}

func main() {
	flag.Parse()
	// Listening to the port
	lis, err := net.Listen("tcp", fmt.Sprintf("localhost:%d", *port))
	if err != nil {
		log.Fatalf("failed to listen: %v", err)
	}
	// Constructing a grpc server object.
	grpcServer := grpc.NewServer()

	pbproto.RegisterStudentServiceServer(grpcServer, &Server{})
	// Starting the server
	grpcServer.Serve(lis)
}
```

- ss