CLOUDBEE ASSIGNMENT

GIVEN QUESTION:

All API referenced are gRPC APIs, not REST ones.

I want to board a train from London to France. The train ticket will cost $20.

Create API where you can submit a purchase for a ticket. Details included in the receipt are: From, To, User , price paid.

User should include first and last name, email address

The user is allocated a seat in the train. Assume the train has only 2 sections, section A and section B.

An API that shows the details of the receipt for the user

An API that lets you view the users and seat they are allocated by the requested section An API to remove a user from the train

An API to modify a user's seat , give detailed information about this also the errorless code?

SOLUTION:

DIRECTORY STRUCTURE:

```
train-ticketing/
|-- client
|   `-- main.go
|-- proto
|   `-- train.proto
|-- server
|   `-- main.go
|-- go.mod
`-- go.sum
```

PROTOCOL BUFFER:

```
syntax = "proto3";

package train;
```

```
message User {
  string first_name = 1;
  string last_name = 2;
  string email = 3;
}

message Ticket {
  string from = 1;
  string to = 2;
  User user = 3;
  float64 price = 4;
}

message Seat {
  string section = 1;
  int32 seat_number = 2;
}

service TrainService {
  rpc PurchaseTicket(Ticket) returns (Seat);
  rpc GetReceipt(User) returns (Ticket);
  rpc GetSeatDetails(string) returns (Seat);
  rpc RemoveUser(User) returns (Seat);
  rpc ModifySeat(User, Seat) returns (Seat);
}
```

IMPLEMENT IN SERVER SIDE:

```
package main

import (
        "context"
        "fmt"
```

```
	"log"
	"net"

	"google.golang.org/grpc"
	"train-ticketing/proto"
)

type trainServer struct {
	userSeats map[string]proto.Seat
}

func (s *trainServer) PurchaseTicket(ctx context.Context, ticket *proto.Ticket) (*proto.Seat,
error) {
	// Generate a seat and assign it to the user
	seat := &proto.Seat{
		Section:    fmt.Sprintf("Section %s", string('A'+len(s.userSeats)%2)),
		SeatNumber: int32(len(s.userSeats) + 1),
	}
	s.userSeats[ticket.User.Email] = *seat

	// Print the receipt on the server side
	log.Printf("Ticket Purchased:\nFrom: %s\nTo: %s\nUser: %s %s\nPrice: $%.2f\n",
		ticket.From, ticket.To, ticket.User.FirstName, ticket.User.LastName,
ticket.Price)

	return seat, nil
}

// Implement other RPC methods similarly...

func main() {
	server := grpc.NewServer()
	proto.RegisterTrainServiceServer(server, &trainServer{userSeats:
make(map[string]proto.Seat)})
```

```
	listener, err := net.Listen("tcp", ":50051")
	if err != nil {
		log.Fatalf("Failed to listen: %v", err)
	}

	log.Println("Server is listening on :50051")
	if err := server.Serve(listener); err != nil {
		log.Fatalf("Failed to serve: %v", err)
	}
}
```

IMPLEMENT IN CLIENT SIDE:

```
package main

import (
	"context"
	"fmt"
	"log"

	"google.golang.org/grpc"
	"train-ticketing/proto"
)

func main() {
	connection, err := grpc.Dial(":50051", grpc.WithInsecure())
	if err != nil {
		log.Fatalf("Failed to connect: %v", err)
	}
	defer connection.Close()

	client := proto.NewTrainServiceClient(connection)
```

```
	// Example usage: Purchase a ticket
	purchasedSeat, err := client.PurchaseTicket(context.Background(), &proto.Ticket{
		From:  "London",
		To:    "France",
		User:  &proto.User{FirstName: "John", LastName: "Doe", Email: "john.doe@example.com"},
		Price: 20.0,
	})

	if err != nil {
		log.Fatalf("Failed to purchase ticket: %v", err)
	}

	fmt.Printf("Purchased Seat: %s %d\n", purchasedSeat.Section, purchasedSeat.SeatNumber)

	// Implement other RPC calls similarly...
}
```