gRPC

Dr. Nuengwong Tuaycharoen

Ref: https://grpc.io/docs/

Objectives

- To be able to create an application with gRPC

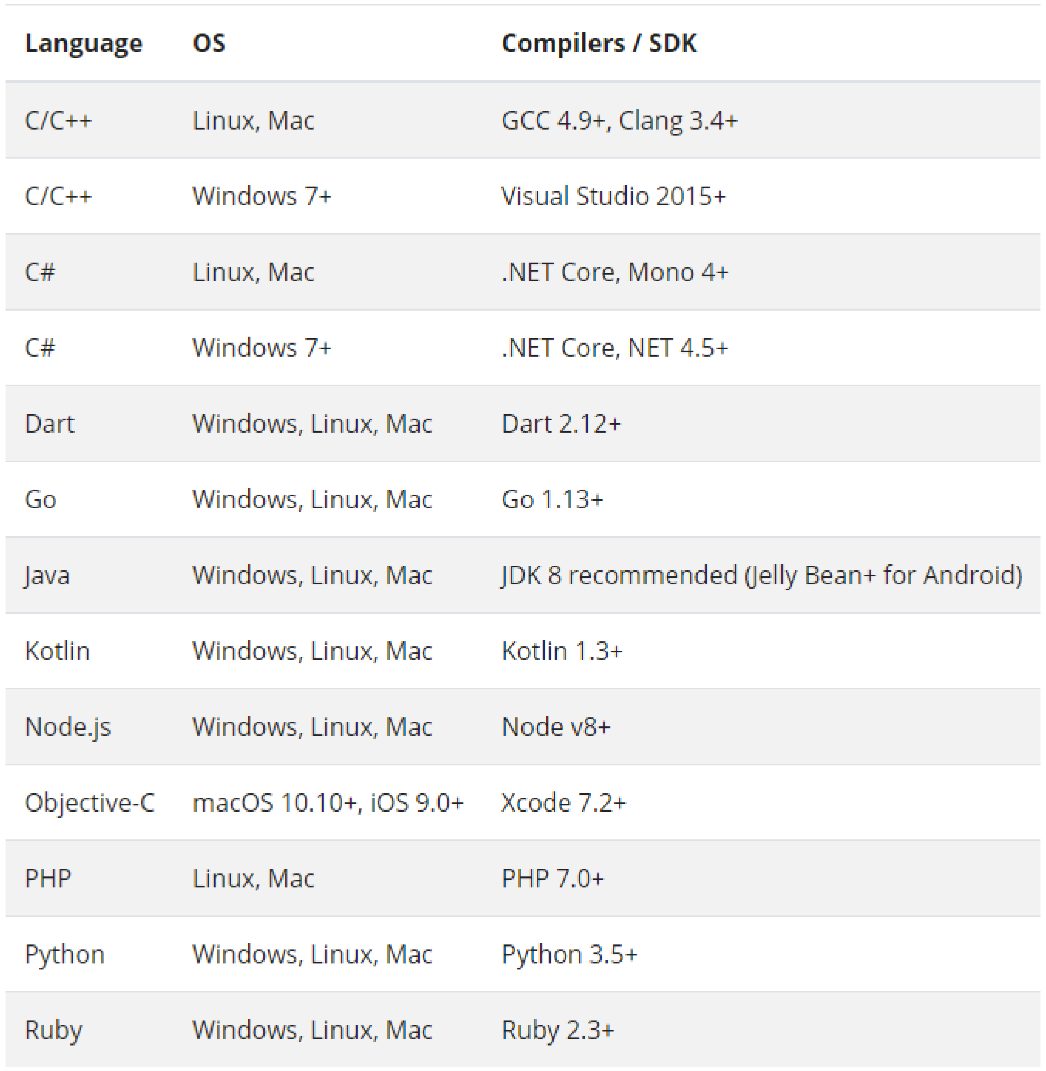
Topics

- gRPC
- Protocol Buffer
- Server code
- Client code
- a full-stack application with gRPC

gRPC

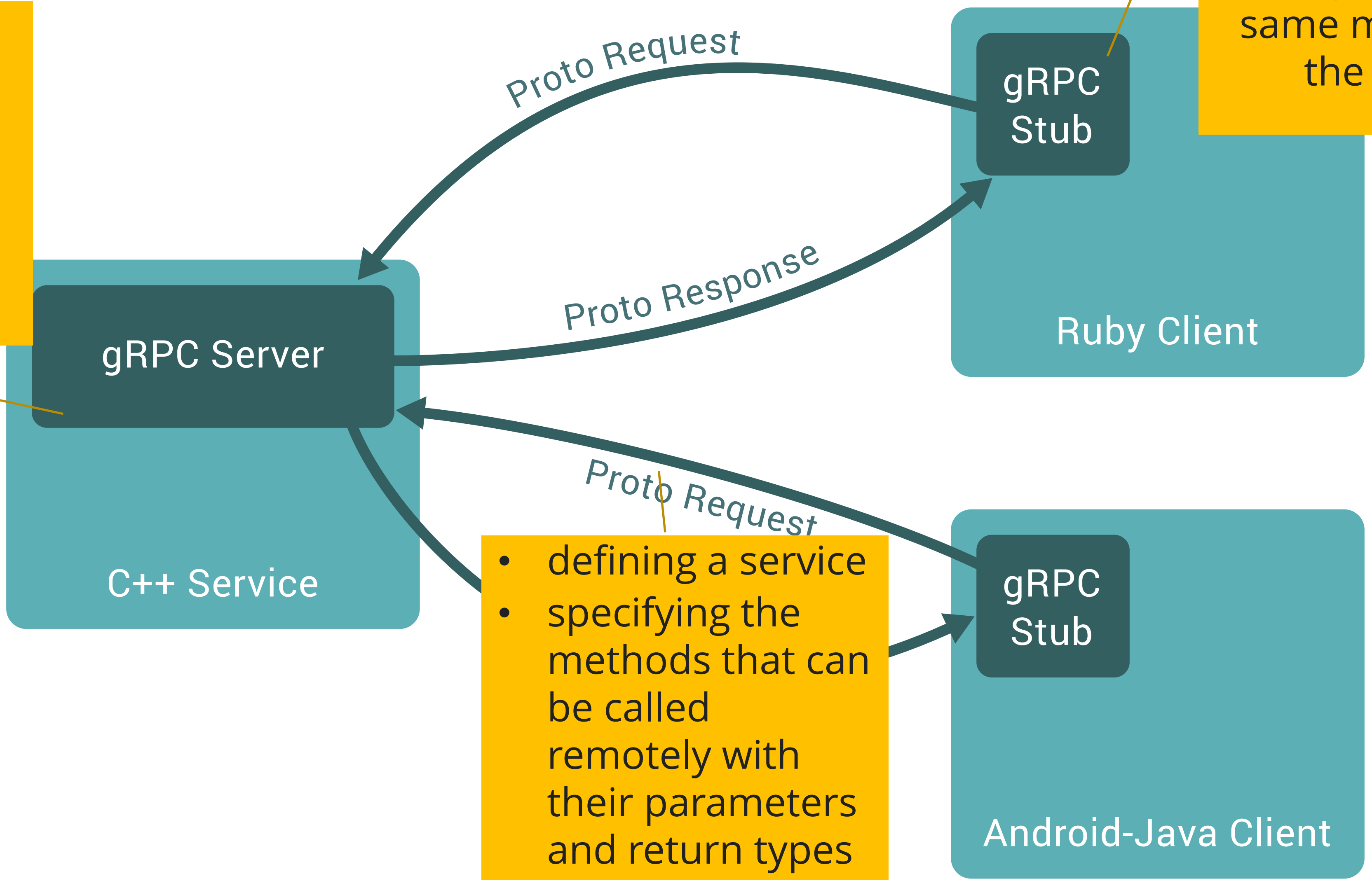
- **g**RPC **R**emote **P**rocedure **C**alls
- gRPC is a modern, open source remote procedure call (RPC) framework that can run anywhere. It enables client and server applications to communicate transparently, and makes it easier to build connected systems.
- It has been used by Google, Square, Netflix, CoreOS, Docker, CockroachDB, Cisco, Juniper Networks and many other organizations

Supported gRPC Languages



Language	OS	Compilers / SDK
C/C++	Linux, Mac	GCC 4.9+, Clang 3.4+
C/C++	Windows 7+	Visual Studio 2015+
C#	Linux, Mac	.NET Core, Mono 4+
C#	Windows 7+	.NET Core, NET 4.5+
Dart	Windows, Linux, Mac	Dart 2.12+
Go	Windows, Linux, Mac	Go 1.13+
Java	Windows, Linux, Mac	JDK 8 recommended (Jelly Bean+ for Android)
Kotlin	Windows, Linux, Mac	Kotlin 1.3+
Node.js	Windows, Linux, Mac	Node v8+
Objective-C	macOS 10.10+, iOS 9.0+	Xcode 7.2+
PHP	Linux, Mac	PHP 7.0+
Python	Windows, Linux, Mac	Python 3.5+
Ruby	Windows, Linux, Mac	Ruby 2.3+

Introduction to gRPC



- implements this interface
- runs a gRPC server to handle client calls.

the client has a stub that provides the same methods as the server.

- defining a service
- specifying the methods that can be called remotely with their parameters and return types

Protocol Buffer (.proto file)

: Interface Definition Language (IDL) for describing both the service interface and the structure of the payload messages

- required [1]
- optional [0..1]
- repeated [0..*]

```
message SearchRequest {
 required string query = 1;
 optional int32 page_number = 2;
 optional int32 result_per_page = 3;
}
```

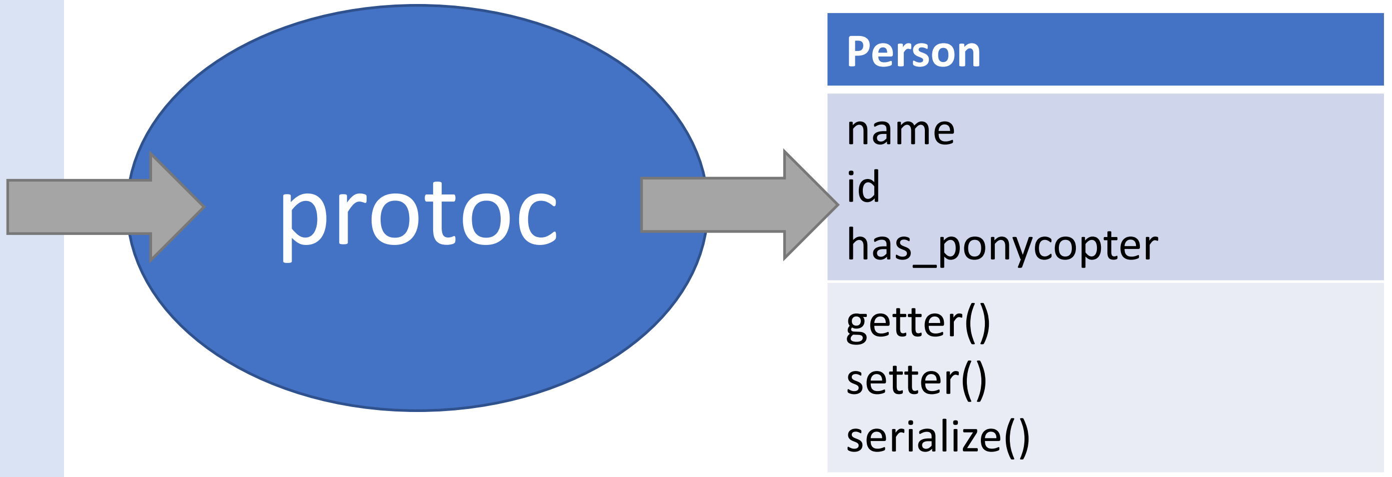
- **unique number**
- From 1 to 2^{29} - 1, or 536,870,911
- Except 19000 through 19999

- double
- float
- Bool
- String
- bytes
- int32/int64
- uint32/uint64
- sint32/sint64

https://developers.google.com/protocol-buffers/docs/overview

Sample Protocol Buffer Message

```
message Person {
  string name = 1;
  int32 id = 2;
  bool has_ponycopter = 3;
}
```



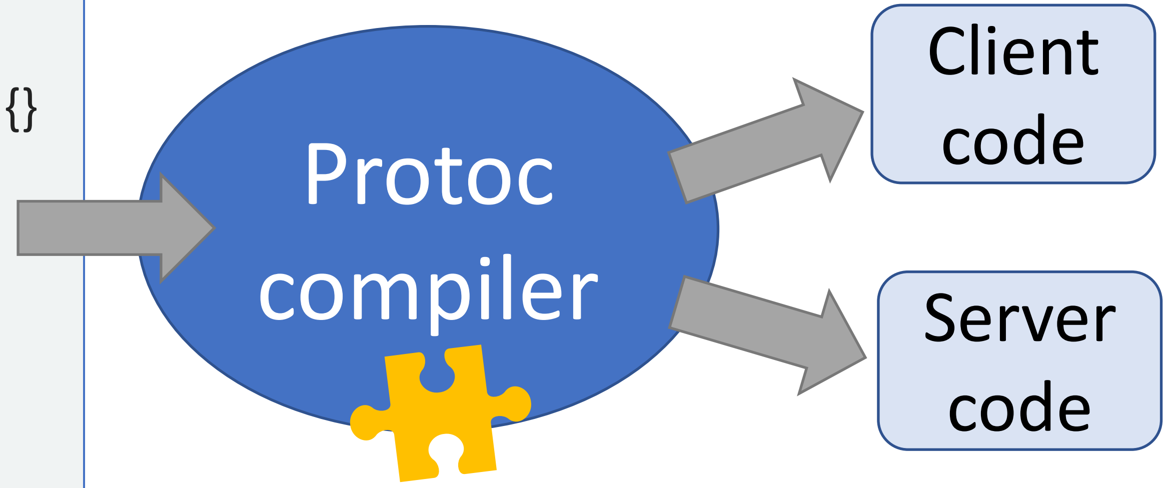
Define gRPC services (.proto)

```
// The greeter service definition.
service Greeter {
// Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloResponse) {}
}

// The request message containing the user's name.
message HelloRequest {
   string greeting = 1;
}

 // The response message containing the greetings
message HelloResponse {
   string reply = 1;

}
```



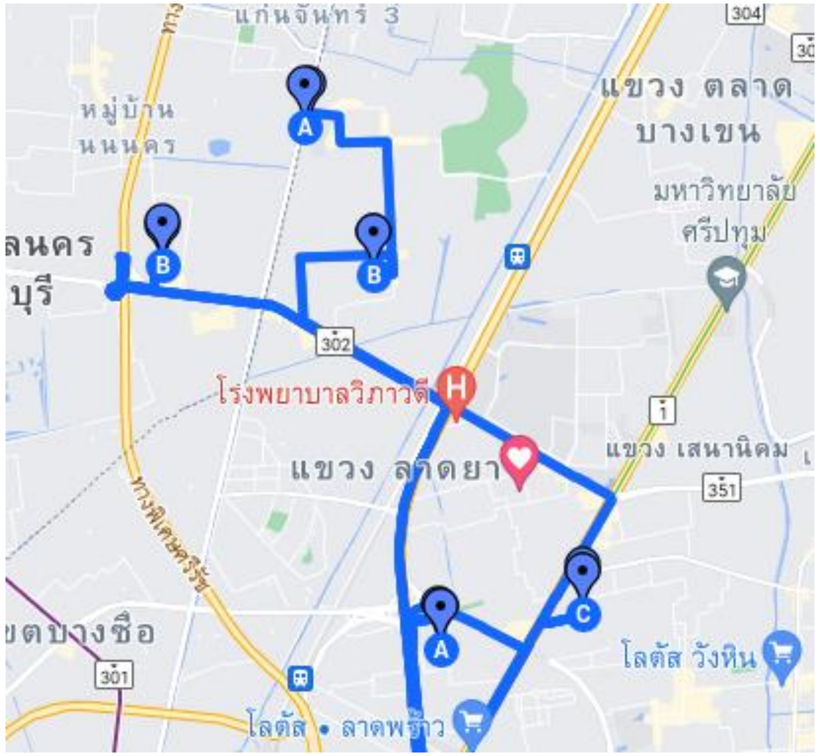
4 kinds of gRPC Service Method

#	Kind	Client's Request	Server's Response
1	Unary RPCs	Single	Single
2	Server streaming RPCs	Single	Stream
3	Client streaming RPCs	Stream	Single
4	Bidirectional streaming RPCs	Stream	Stream

*gRPC guarantee message order

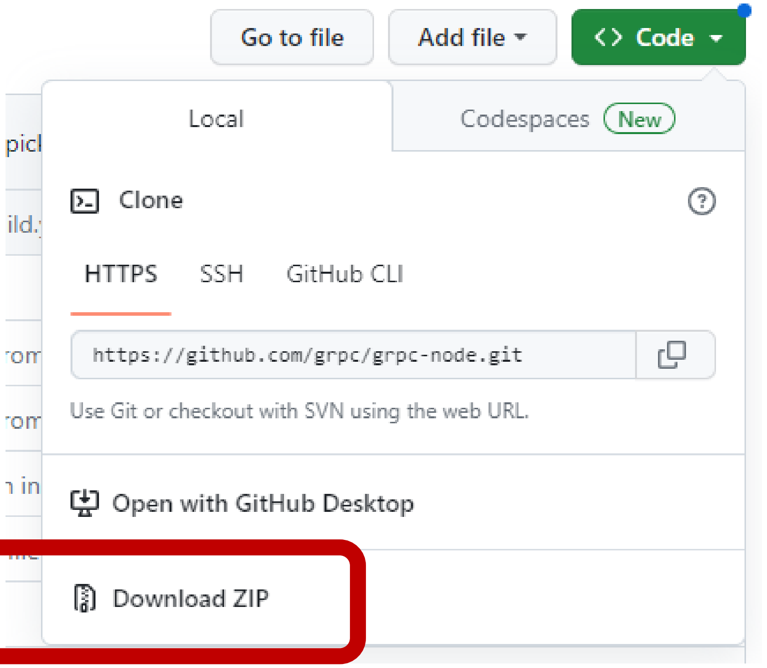
Sample Application: Route Mapping Application

- Clients get information about features on their route
- Clients create a summary of their route
- Clients exchange route information such as traffic updates with the server and other clients.



Clone the sample application directory

1. Goto: https://github.com/grpc/grpc-node/
2. Click on Code Button.
3. Download the zip file



https://github.com/grpc/grpc-node.git

Use Git or checkout with SVN using the web URL.

1. Open your command prompt
2. Type this command

```
$ git clone https://github.com/grpc/grpc-node/
```

3. Change current directory to

```
grpc-node-master\examples\routeguide\dynamic_codegen\
```

version ล่าสุด เปลี่ยน path เป็นตัวแดงนี้แทนนะคะ

The Proto file

examples/proto/route_guide.proto(1) : header

```
syntax = "proto3";

option java_multiple_files = true;
option java_package = "io.grpc.examples.routeguide";
option java_outer_classname = "RouteGuideProto";
option objc_class_prefix = "RTG";

package routeguide;
```

Example/proto/route_guide.proto(2) :Service

Define RPC service

```
service RouteGuide {
  rpc GetFeature(Point) returns (Feature) {}
  rpc ListFeatures(Rectangle) returns (stream Feature) {}
  rpc RecordRoute(stream Point) returns (RouteSummary) {}

  rpc RouteChat(stream RouteNote) returns (stream RouteNote) {}
}
```

Define RPC methods

Request type

Response type

Example/proto/route_guide.proto(3): message

```
message Point {
  int32 latitude = 1;
  int32 longitude = 2;
}
message Rectangle {
  Point lo = 1;
  Point hi = 2;
}
message Feature {
  string name = 1;
  Point location = 2;
}
```

```
message RouteNote {
  Point location = 1;
  string message = 2;
}
message RouteSummary {
  int32 point_count = 1;
  int32 feature_count = 2;
  int32 distance = 3;
  int32 elapsed_time = 4;
}
```

The Node.js file

Loading service descriptors from proto files

```
var PROTO_PATH = __dirname + '/../../../protos/route_guide.proto';
var grpc = require('@grpc/grpc-js');
var protoLoader = require('@grpc/proto-loader');
// Suggested options for similarity to existing grpc.load behavior
var packageDefinition = protoLoader.loadSync(
   PROTO_PATH,
   {keepCase: true,
   longs: String,
   enums: String,
   defaults: true,
   oneofs: true
   });
var protoDescriptor = grpc.loadPackageDefinition(packageDefinition);
// The protoDescriptor object has the full package hierarchy

var routeguide = protoDescriptor.routeguide;
```

Stub:
protoDescriptor.routeguide.RouteGuide

Service:
protoDescriptor.routeguide.RouteGuide.service

The Server file

Create a server

```
function getServer() {
  var server = new grpc.Server();
  server.addService(routeguide.RouteGuide.service, {
    getFeature: getFeature,
    listFeatures: listFeatures,
    recordRoute: recordRoute,
    routeChat: routeChat
  });
  return server;
}
```

start the server

```
var routeServer = getServer();
routeServer.bindAsync('0.0.0.0:50051',
grpc.ServerCredentials.createInsecure(), () => {
routeServer.start(); });
```

Sample code to start the server

```
if (require.main === module) {
  // If this is run as a script, start a server on an unused port
  var routeServer = getServer();
  routeServer.bindAsync('0.0.0.0:50051', grpc.ServerCredentials.createInsecure(), () => {
    var argv = parseArgs(process.argv, {
      string: 'db_path'
    });
    fs.readFile(path.resolve(argv.db_path), function(err, data) {
      if (err) throw err;
      feature_list = JSON.parse(data);
      routeServer.start();
    });
  });
}

exports.getServer = getServer;
```

1. Unary RPC Server Service

```
function checkFeature(point) {
  var feature;
  // Check if there is already a feature object for the given point
  for (var i = 0; i < feature_list.length; i++) {
    feature = feature_list[i];
    if (feature.location.latitude === point.latitude &&
        feature.location.longitude === point.longitude) {
      return feature;
    }
  }
  var name = '';
  feature = {
    name: name,
    location: point
  };
  return feature;
}
```

```
function getFeature(call, callback) {
  callback(null, checkFeature(call.request));
}
```

2. Server Streaming RPC Server Service

```
function listFeatures(call) {
  var lo = call.request.lo;
  var hi = call.request.hi;
  var left = _.min([lo.longitude, hi.longitude]);
  var right = _.max([lo.longitude, hi.longitude])
;
  var top = _.max([lo.latitude, hi.latitude]);
  var bottom = _.min([lo.latitude, hi.latitude]);
```

```
  var bottom = _.min([lo.latitude, hi.latitude]);
  // For each feature, check if it is in the given bounding box
  _.each(feature_list, function(feature) {
    if (feature.name === '') {
      return;
    }
    if (feature.location.longitude >= left &&
        feature.location.longitude <= right &&
        feature.location.latitude >= bottom &&
        feature.location.latitude <= top) {
      call.write(feature);
    }
  });
  call.end();
}
```

Send all messages.

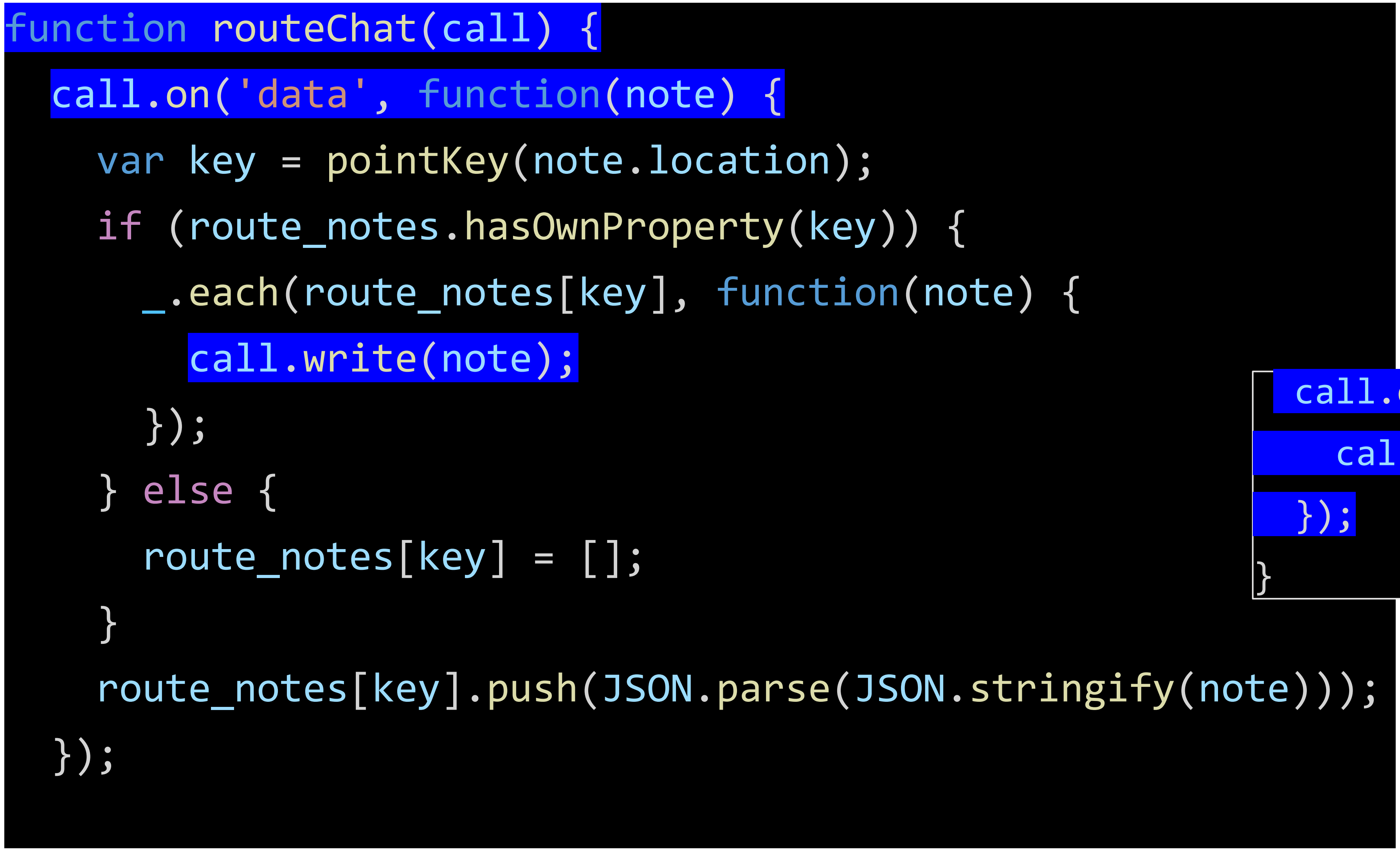
3. Client Streaming RPC Server Service

```
function recordRoute(call, callback) {
  var point_count = 0;
  var feature_count = 0;
  var distance = 0;
  var previous = null;
  var start_time = process.hrtime();
  call.on('data', function(point) {
    point_count += 1;
    if (checkFeature(point).name !== '') {
      feature_count += 1;
    }
    if (previous != null) {
      distance += getDistance(previous, point);
    }
    previous = point;
  });
```

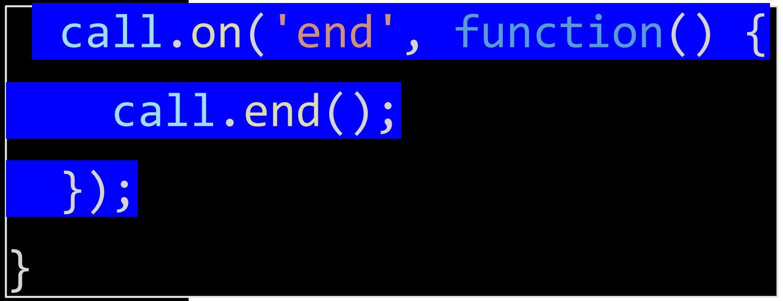
```
  call.on('end', function() {
    callback(null, {
      point_count: point_count,
      feature_count: feature_count,
      // Cast the distance to an integer
      distance: distance|0,
      // End the timer
      elapsed_time: process.hrtime(start_time)[0]
    });
  });
}
```

```
call.on('data', function(point) {
  // Process user data
});
call.on('end', function() {
  callback(null, result);
});
```

4. Bidirectional Streaming RPC Server Service



```
function routeChat(call) {
  call.on('data', function(note) {
    var key = pointKey(note.location);
    if (route_notes.hasOwnProperty(key)) {
      _.each(route_notes[key], function(note) {
        call.write(note);
      });
    } else {
      route_notes[key] = [];
    }
    route_notes[key].push(JSON.parse(JSON.stringify(note)));
  });
```



```
  call.on('end', function() {
    call.end();
  });
}
```

Conclusion:

1. Create a Server constructor from the RouteGuide service descriptor.
2. Implement the service methods.
3. Create an instance of the server by calling the Server constructor with the method implementations.
4. Specify the address and port we want to use to listen for client requests using the instance’s bind() method.
5. Call start() on the instance to start the RPC server.

The Client file

Create a client’s stub

```
var client = new routeguide.RouteGuide('localhost:50051',
    grpc.credentials.createInsecure());
```

Server’s address & port

1. Unary RPC Server Service

```
function runGetFeature(callback) {
  var next = _.after(2, callback);
  function featureCallback(error, feature) {
    if (error) {
      callback(error);
      return;
    }
    if (feature.name === '') {
      console.log('Found no feature at ' +
          feature.location.latitude/COORD_FACTOR + ', ' +
          feature.location.longitude/COORD_FACTOR);
    } else {
      console.log('Found feature called "' + feature.name + '" at ' +
          feature.location.latitude/COORD_FACTOR + ', ' +
          feature.location.longitude/COORD_FACTOR);
    }
    next();
  }
```

```
  var point1 = {
    latitude: 409146138,
    longitude: -746188906
  };
  var point2 = {
    latitude: 0,
    longitude: 0
  };
  client.getFeature(point1, featureCallback);
  client.getFeature(point2, featureCallback);
}
```

```
var point = {latitude: 409146138, longitude: -746188906};
client.getFeature(point, function(err, feature) {
 if (err) {
// process error
} else {
// process feature
}
});
```

2. Server Streaming RPC Server Service (1 of 2)

```
var call = client.listFeatures(rectangle);
call.on('data', function(feature) {
      console.log('Found feature called "' + feature.name + '" at ‘ +
         feature.location.latitude/COORD_FACTOR + ', ‘ +
         feature.location.longitude/COORD_FACTOR);
});
call.on('end', function() {
   // The server has finished sending
});
call.on('error', function(e) {
   // An error has occurred and the stream has been closed.
});
call.on('status', function(status) {
   // process status

});
```

2. Server Streaming RPC Server Service (2 of 2)

```
function runListFeatures(callback) {
  var rectangle = {
    lo: {
      latitude: 400000000,
      longitude: -750000000
    },
    hi: {
      latitude: 420000000,
      longitude: -730000000
    }
  };
```

```
 console.log('Looking for features between 40, -
75 and 42, -73');
  var call = client.listFeatures(rectangle);
  call.on('data', function(feature) {
      console.log('Found feature called "' + feature.
name + '" at ’ +feature.location.latitude/COORD_FACTO
R + ', ' + feature.location.longitude/COORD_FACTOR);
  });
  call.on('end', callback);
}
```

3. Client Streaming RPC Server Service (1 of 2)

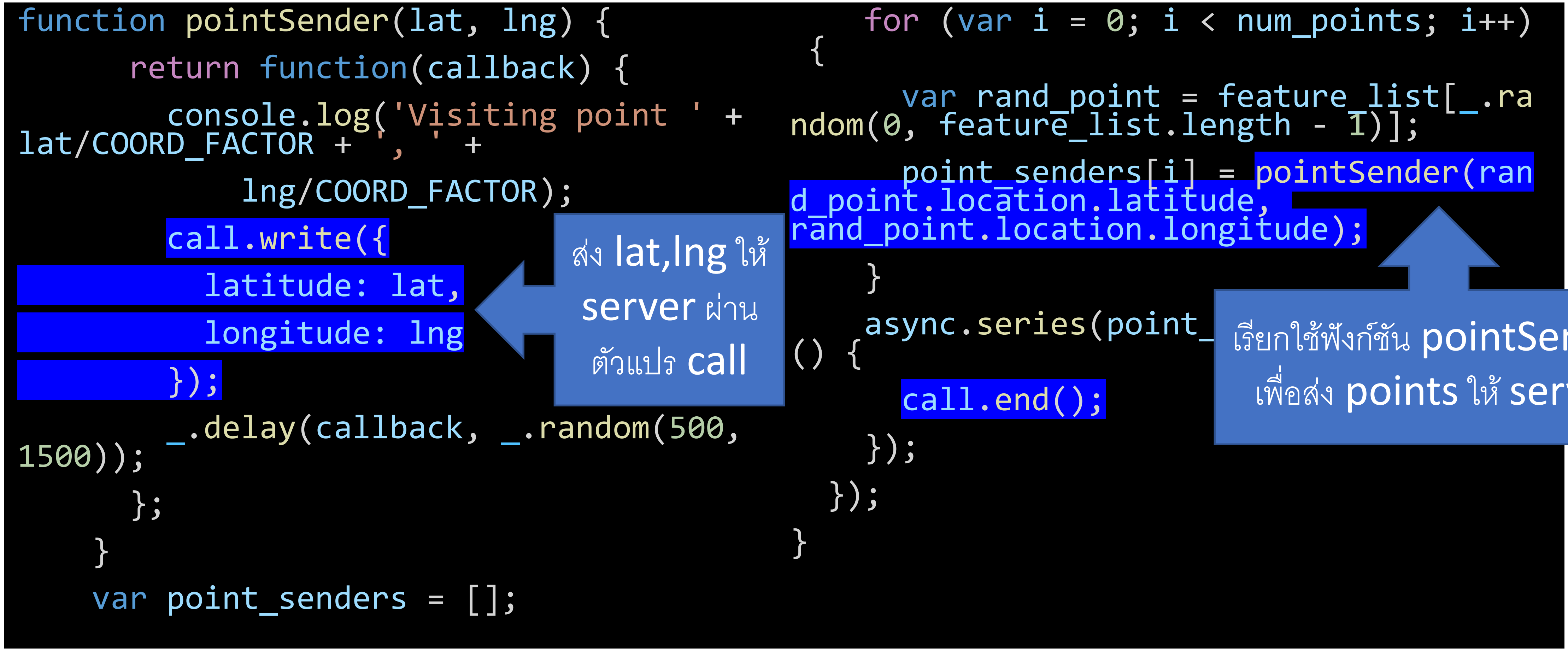
```
function runRecordRoute(callback) {
  var argv = parseArgs(process.argv, {
    string: 'db_path'
  });
  fs.readFile(path.resolve(argv.db_path),
    function(err, data) {
    if (err) {
      callback(err);
      return;
    }
    var feature_list = JSON.parse(data);

    var num_points = 10;
```

```
   var call = client.recordRoute(function(error,
stats) {
      if (error) {
        callback(error);
        return;
      }
      console.log(‘#pts: ', stats.point_count);
      console.log(‘#ftr: ', stats.feature_count);
      console.log(‘distance: ', stats.distance);
      console.log(‘time: ', stats.elapsed_time);
      callback();
   });
```

เรียก **recordRoute** และส่ง callback function เพื่อสรุปข้อมูลที่ส่งให้ server

3. Client Streaming RPC Server Service(2 of 2)



```
function pointSender(lat, lng) {
      return function(callback) {
         console.log('Visiting point ' +
lat/COORD_FACTOR + ', ' +
            lng/COORD_FACTOR);
         call.write({
           latitude: lat,
           longitude: lng
         });
         _.delay(callback, _.random(500,
1500));
      };
    }
    var point_senders = [];
```

```
      for (var i = 0; i < num_points; i++)
 {
         var rand_point = feature_list[_.ra
ndom(0, feature_list.length - 1)];
         point_senders[i] = pointSender(ran
d_point.location.latitude,
rand_point.location.longitude);
      }
      async.series(point_
() {
         call.end();
      });
   });
}
```

ส่ง lat,lng ให้ server ผ่านตัวแปร call

เรียกใช้ฟังก์ชัน pointSender เพื่อส่ง points ให้ server

4. Bidirectional Streaming RPC Server Service

```
var notes = [{
  location: {
    latitude: 0,
    longitude: 0
  },
  message: 'First message'
}, {
  location: {
    latitude: 0,
    longitude: 1
  },
  message: 'Second message'
}, {
```

```
function runRouteChat(callback) {
  var call = client.routeChat();
  call.on('data', function(note) {
    console.log('Got message "' + note.message + '" at ' +
        note.location.latitude + ', ' + note.location.longitude);
  });

  call.on('end', callback);
```

เรียก routeChat () บนserver

ถ้า server จบการส่งข้อมูล ให้ทำงานฟังก์ชัน callback

```
    for (var i = 0; i < notes.length; i++) {
    var note = notes[i];
    console.log('Sending message "' + note.message + '" at ' +
        note.location.latitude + ', ' + note.location.longitude);
    call.write(note);
  }
  call.end();
}
```

ส่งข้อมูลให้ server ผ่านตัวแปร call

จบการส่งข้อมูลให้ server

Main Client Program

```
function main() {
  async.series([
    runGetFeature,
    runListFeatures,
    runRecordRoute,
    runRouteChat
  ]);
}
```

```
if (require.main === module) {
  main();
}

exports.runGetFeature = runGetFeature;

exports.runListFeatures = runListFeatures;

exports.runRecordRoute = runRecordRoute;

exports.runRouteChat = runRouteChat;
```

Try it out!

1. Install the dependencies at examples/routeguide directory

```
$ cd ..
$ npm install
```

2. Run the server:

```
$ node . /route_guide_server.js --db_path=route_guide_db.json
```

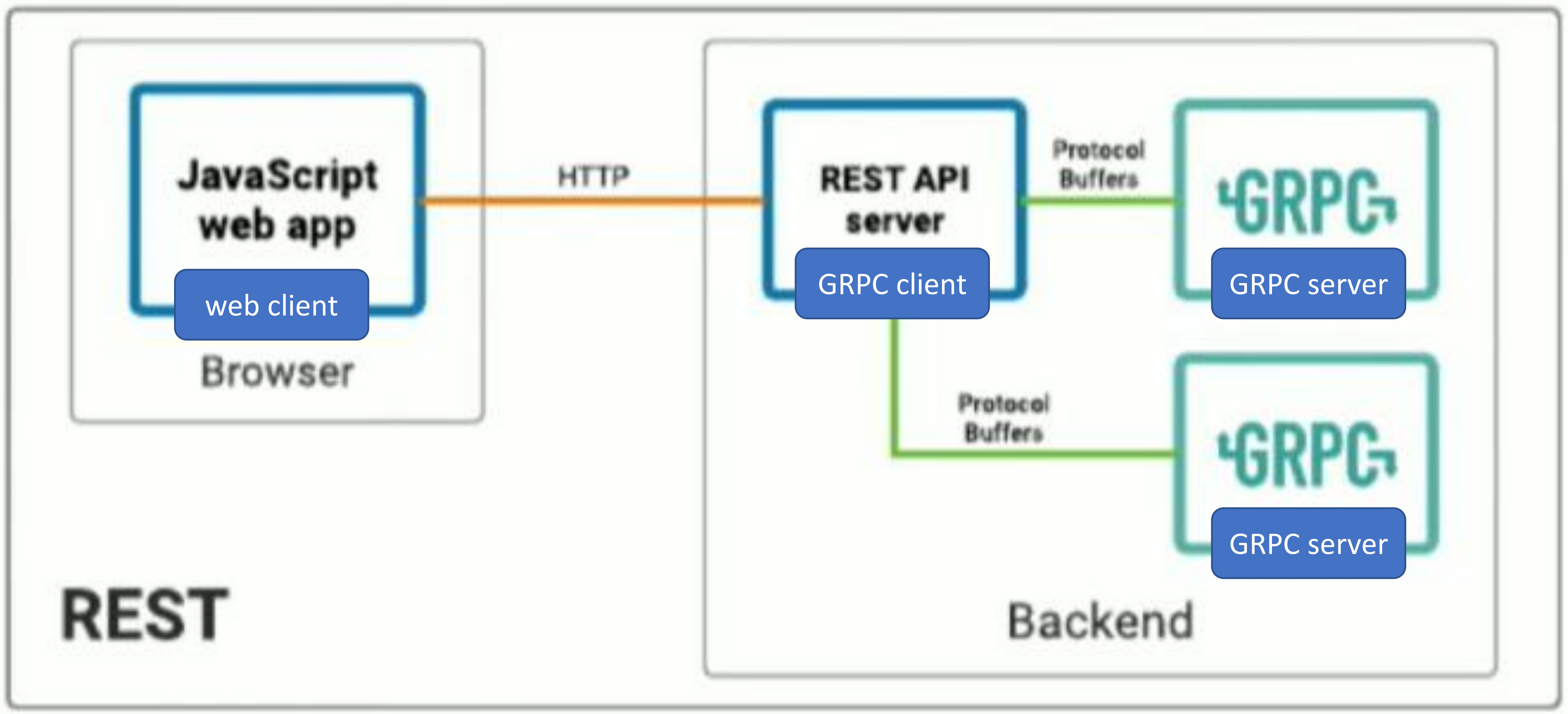
3. From a different terminal, run the client:

```
$ node ./route_guide_client.js --db_path=route_guide_db.json
```

Creating a CRUD API with node-express-grpc

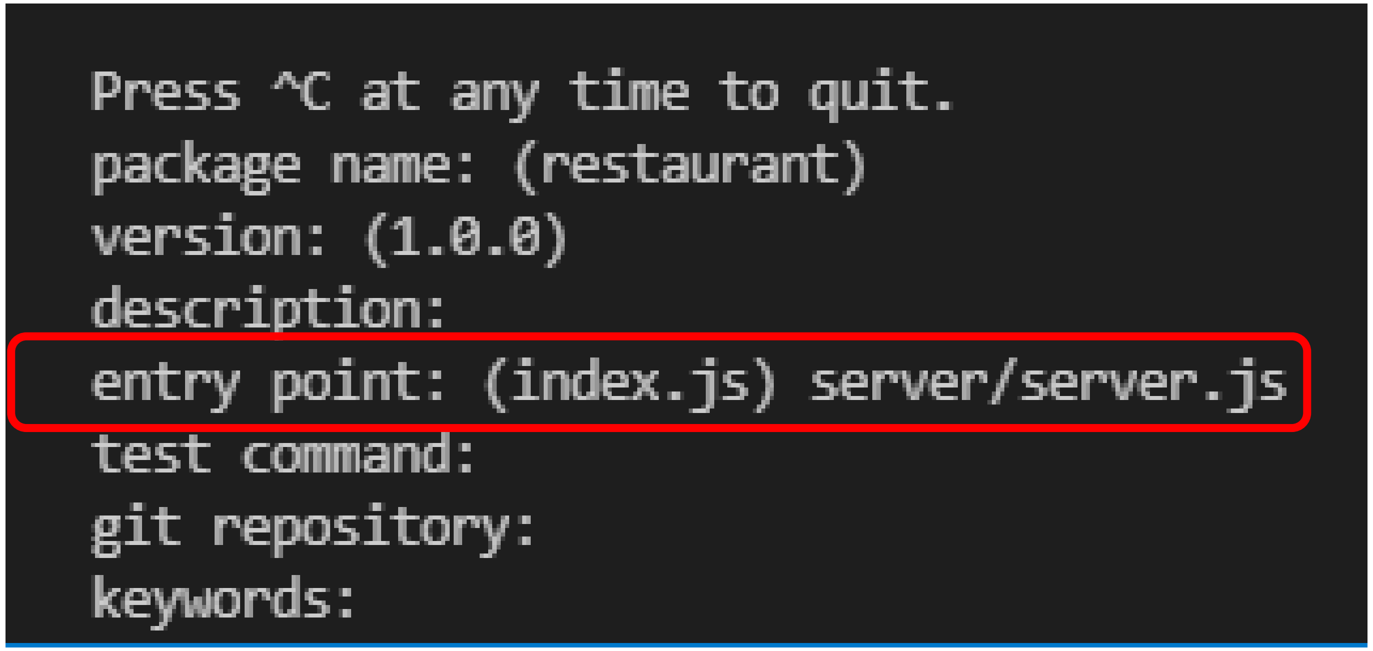
https://blog.logrocket.com/creating-a-crud-api-with-node-express-and-grpc/

The Architecture



Let’s start!

- Download files: https://drive.google.com/file/d/1B0e5kn2xWbN1vkIy10HKnCctNDl7RBdn/view?usp=sharing
- Open VS Code & open your project folder
- npm install –save @grpc/grpc-js @grpc/proto-loader uuid express hbs body-parser
- npm init



```
Press ^C at any time to quit.
package name: (restaurant)
version: (1.0.0)
description:
entry point: (index.js) server/server.js
test command:
git repository:
keywords:
```

package.json

```
{
  "name": "restaurant",
  "version": "1.0.0",
  "description": "",
  "main": "server/server.js",
  "dependencies": {
    "@grpc/proto-loader": "^0.6.4",
    "body-parser": "^1.19.0",
    "express": "^4.17.1",
    “@grpc/grpc-js": "^1.1.0",
    "hbs": "^4.1.2",
    "uuid": "^8.3.2"
  },
  "devDependencies": {},
  "scripts": {
    "start": "node server/server.js"
  },
  "author": "",
  "license": "ISC"
}
```

restaurant.proto

```
syntax ="proto3";

service RestaurantService {
    rpc GetAllMenu(Empty) returns (MenuList) {}
    rpc Get (MenuId) returns (MenuItem){}
    rpc Insert (MenuItem) returns (MenuItem) {}
    rpc Update (MenuItem) returns (MenuItem) {}
    rpc Remove (MenuId) returns (Empty) {}
}

message Empty{}
```

```
message MenuItem {
    string id =1;
    string name=2;
    int32 price=3;
}

message MenuList{
    repeated MenuItem menu=1;
}

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

server.js (1/7)

```
const PROTO_PATH="./restaurant.proto";

var grpc = require("grpc");
var protoLoader = require("@grpc/proto-loader");

var packageDefinition = protoLoader.loadSync(PROTO_PATH,{
    keepCase: true,
    longs: String,
    enums: String,
    arrays: true
});

var restaurantProto =grpc.loadPackageDefinition(packageDefinition);

const {v4: uuidv4}=require("uuid");

const server = new grpc.Server();
```

server.js (2/7)

```
const menu=[
    {
        id: "a68b823c-7ca6-44bc-b721-fb4d5312cafc",
        name: "Tomyam Gung",
        price: 500
    },
    {
        id: "34415c7c-f82d-4e44-88ca-ae2a1aaa92b7",
        name: "Somtam",
        price: 60
    },
    {
        id: "8551887c-f82d-4e44-88ca-ae2a1ccc92b7",
        name: "Pad-Thai",
        price: 120
    }
];
```

server.js (3/7)

```
server.addService(restaurantProto.RestaurantService.service,{
    getAllMenu: (_,callback)=>{
        callback(null, {menu});
    },
    get: (call,callback)=>{
        let menuItem = menu.find(n=>n.id==call.request.id);

        if(menuItem) {
            callback(null, menuItem);
        }else {
            callback({
                code: grpc.status.NOT_FOUND,
                details: "Not found"
            });
        }
    },
```

server.js (4/7)

```
    insert: (call, callback)=>{
        let menuItem=call.request;

        menuItem.id=uuidv4();
        menu.push(menuItem);
        callback(null,menuItem);
    },
```

server.js (5/7)

```
    update: (call,callback)=>{
        let existingMenuItem = menu.find(n=>n.id==call.request.id);

        if(existingMenuItem){
            existingMenuItem.name=call.request.name;
            existingMenuItem.price=call.request.price;
            callback(null,existingMenuItem);
        } else {
            callback({
                code: grpc.status.NOT_FOUND,
                details: "Not Found"
            });
        }
    },
```

server.js (6/7)

```
    remove: (call, callback) => {
        let existingMenuItemIndex = menu.findIndex(n=>n.id==call.request.id);

        if(existingMenuItemIndex != -1){
            menu.splice(existingMenuItemIndex,1);
            callback(null,{});
        } else {
            callback({
                code: grpc.status.NOT_FOUND,
                details: "NOT Found"
            });
        }
    }
});
```

server.js (7/7)

```
server.bind("127.0.0.1:30043",grpc.ServerCredentials.createInsecure());
console.log("Server running at http://127.0.0.1:30043");
server.start();
```

client.js (1/2)

```
const PROTO_PATH="../restaurant.proto";

const grpc = require("grpc");
const protoLoader = require("@grpc/proto-loader");

var packageDefinition = protoLoader.loadSync(PROTO_PATH,{
    keepCase: true,
    longs: String,
    enums: String,
    arrays: true
});
```

client.js (1/2)

```
var restaurantService =grpc.loadPackageDefinition(pa
ckageDefinition).RestaurantService;

const client = new restaurantService("localhost:3004
3", grpc.credentials.createInsecure());

module.exports = client;
```

index.js (1/5)

```
const client = require("./client");

const path = require("path");
const express = require("express");
const bodyParser = require("body-parser");

const app = express();

app.set("views",path.join(__dirname,"views"));
app.set("view engine","hbs");

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended:false}));
```

index.js (2/5)

```
app.get("/",(req,res)=>{
    client.getAllMenu(null,(err,data)=>{
        if(!err){
            res.render("menu",{
                results: data.menu
            });
        }
    });
});
```

index.js (3/5)

```
app.post("/save",(req,res)=>{
    let newMenuItem={
        name:req.body.name,
        price: req.body.price
    };

    client.insert(newMenuItem,(err,data)=>{
        if(err) throw err;

        console.log("New Menu created successfully", data);
        res.redirect("/");
    });
});
```

index.js (4/5)

```
app.post("/update", (req, res) => {
    const updateMenuItem = {
        id: req.body.id,
        name: req.body.name,
        price: req.body.price,
    };
    console.log("update Item %s %s %d",updateMenuItem.id, req.body.name, req.body.price);

    client.update(updateMenuItem, (err, data) => {
        if (err) throw err;

        console.log("Menu Item updated successfully", data);
        res.redirect("/");
    });
});
```

index.js (5/5)

```
app.post("/remove",(req,res)=>{
    client.remove({id: req.body.menuItem_id},(err,_)=>{
        if(err) throw err;
        console.log("Menu Item removed successfully");
        res.redirect("/");
    });
});

const PORT = process.env.PORT || 3000;
app.listen(PORT,()=>{
    console.log("Server running at port %d",PORT);
});
```

menu.hbs (1)

```
<html lang="en">

<head>
    <meta charset="utf-8">
    <title>Restaurant CRUD with gRPC and NodeJS</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.
min.css"
        integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
    <style>
        .restaurant {
            background-color: #764abc;
            color: white;
        }
    </style>
</head>
```

menu.hbs (2)

```
<body>
    <div class="container">
        <div class="py-5 text-center">
            <img class="d-block mx-auto mb-4"
                src="https://www.cp.eng.chula.ac.th/wp-
content/uploads/2014/08/ChulaEngineeringComputer-
formal.png" alt="Logo"
                height="72">
            <h2>Menu's List</h2>
            <p class="lead">Example of CRUD made with Node.js
, Express, Handlebars and gRPC</p>
        </div>
```

menu.hbs (3)

```
        <table class="table" id="Menus_table">
            <thead>
                <tr>
                    <th>Menu ID</th>
                    <th>Name</th>
                    <th>Price</th>
                    <th>Action</th>
                </tr>
            </thead>
```

menu.hbs (4)

```
            <tbody>
                {{#each results}}
                <tr>
                    <td>{{ id }}</td>
                    <td>{{ name }}</td>
                    <td>{{ price }} THB</td>
                    <td>
                        <a href="javascript:void(0);" class="btn btn-
sm edit restaurant" data-id="{{ id }}"
                            data-name="{{ name }}" data-age="{{ age }}">Edit</a>
                        <a href="javascript:void(0);" class="btn btn-sm btn-
danger remove" data-id="{{ id }}">Remove</a>
                    </td>
                </tr>
```

menu.hbs (5)

```
                {{else}}
                <tr>
                    <td colspan="5" class="text-center">No data to display.</td>
                </tr>
                {{/each}}
            </tbody>
        </table>
        <button class="btn btn-success float-right" data-toggle="modal" data-
target="#newMenuModal">Add New</button>
    </div>
```

menu.hbs (6)

```
    <!-- New Menu Modal -->
    <form action="/save" method="post">
        <div class="modal fade" id="newMenuModal" role="dialog">
            <div class="modal-dialog" role="document">
                <div class="modal-content">
                    <div class="modal-header">
                        <h4 class="modal-title">New Menu</h4>
                        <button type="button" class="close" data-
dismiss="modal">
                            <span>&times;</span>
                        </button>
                    </div>
```

menu.hbs (7)

```
          <div class="modal-body">
            <div class="form-group">
              <input type="text" name="name" class="f
orm-control" placeholder="Menu Name"
                required="required">
            </div>

            <div class="form-group">
              <input type="number" name="price" class
="form-control" placeholder="Price" required="required">
            </div>
          </div>
```

menu.hbs (8)

```
                    <div class="modal-footer">
                        <button type="button" class="btn btn-
secondary" data-dismiss="modal">Close</button>
                        <button type="submit" class="btn rest
aurant">Create</button>
                    </div>
                </div>
            </div>
        </div>
    </form>
```

menu.hbs (9)

```
    <!-- Edit Menu Modal -->
    <form action="/update" method="post">
        <div class="modal fade" id="editMenuModal" role="dialog">
            <div class="modal-dialog" role="document">
                <div class="modal-content">
                    <div class="modal-header">
                        <h4 class="modal-title">Edit Menu</h4>
                        <button type="button" class="close" data-
dismiss="modal">
                            <span>&times;</span>
                        </button>
                    </div>
```

menu.hbs (10)

```
                    <div class="modal-body">
                        <div class="form-group">
                            <input type="text" name="name" class="f
orm-control name" placeholder="Menu Name"
                                required="required">
                        </div>

                        <div class="form-group">
                            <input type="number" name="price" class
="form-control price" placeholder="Price"
                                required="required">
                        </div>
                    </div>
```

menu.hbs (11)

```
                    </div>
                    <div class="modal-footer">
                        <input type="hidden" name="id" class="menu_
id">
                        <button type="button" class="btn btn-
secondary" data-dismiss="modal">Close</button>
                        <button type="submit" class="btn logrocket"
>Update</button>
                    </div>
                </div>
            </div>
        </div>
    </form>
```

menu.hbs (12)

```
    <!-- Remove Menu Modal -->
    <form id="add-row-form" action="/remove" method="post">
        <div class="modal fade" id="removeMenuModal" role="di
alog" aria-labelledby="myModalLabel">
            <div class="modal-dialog">
                <div class="modal-content">
                    <div class="modal-header">
                        <h4 class="modal-
title"></h4>Remove Menu</h4>
                        <button type="button" class="close" d
ata-dismiss="modal"><span>&times;</span></button>
                    </div>
```

menu.hbs (13)

```
                    <div class="modal-body">
                        Are you sure?
                    </div>
                    <div class="modal-footer">
                        <input type="hidden" name="menuItem_id" class="form-control Menu_id_removal"
                            required="required">
                        <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
                        <button type="submit" class="btn restaurant">Remove</button>
                    </div>
                </div>
            </div>
        </div>
    </form>
```

menu.hbs (14)

```
    <script src="https://code.jquery.com/jquery-
3.3.1.slim.min.js"
        integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1
Pi6jizo"
        crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/boots
trap/4.3.1/js/bootstrap.min.js"
        integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM
+B07jRM"
        crossorigin="anonymous"></script>
```

menu.hbs (15)

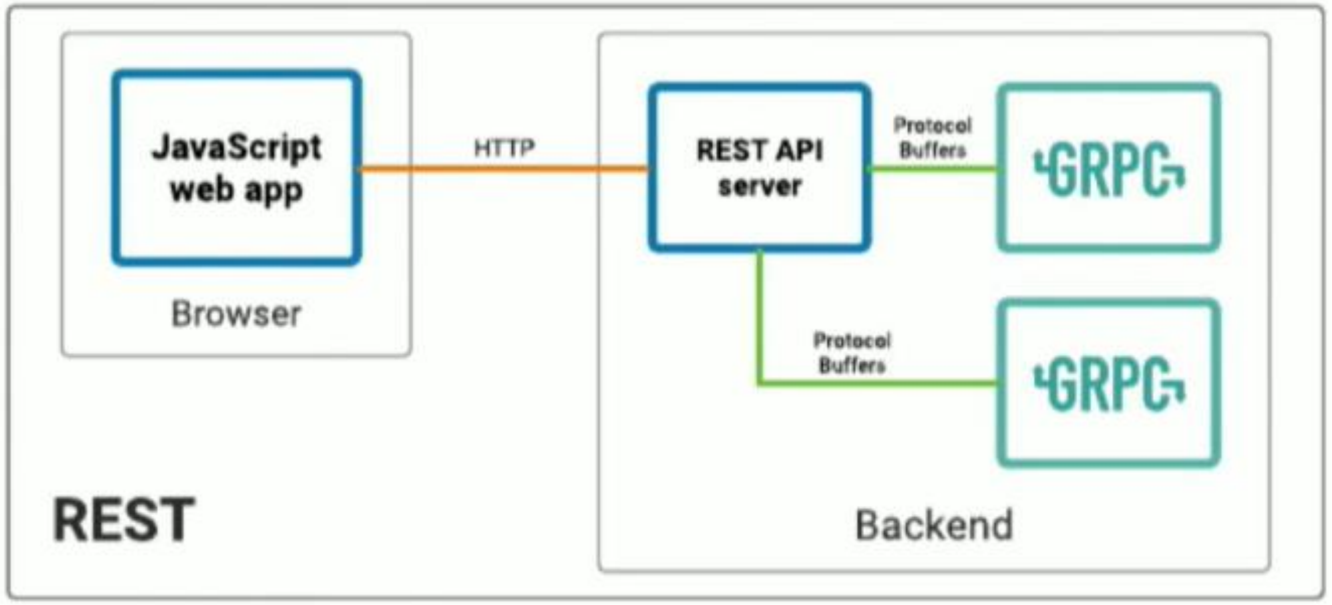
```
        <script>
        $(document).ready(function () {
            $('#Menus_table').on('click', '.edit', function () {
                $('#editMenuModal').modal('show');

                $('.menu_id').val($(this).data('id'));
                $('.name').val($(this).data('name'));
                $('.age').val($(this).data('age'));
                $('.address').val($(this).data('address'));
            }).on('click', '.remove', function () {
                $('#removeMenuModal').modal('show');

                $('.Menu_id_removal').val($(this).data('id'));
            });
        });
    </script>
</body>
</html>
```

Let’s RUN it!

- Run the gRPC server:
  - npm start
- Run the gRPC client(REST API server)
  - cd client
  - node index.js
- Run the web client (Browser)
  - Open a web browser
  - Go to: localhost:3000



Assignment: connect gRPC Server with MongoDB

Create a model file: models/Menu.js

MenuSchema includes name and price

Edit server.js to connect with database

- Require mongoose
- Connect to your database via mongodb.com
- Edit the Service function to connect to your mongodb database
- Don't forget async-await when call your database

Run your server and client/index

Record your CRUD demonstration and your mongodb database [<5min]