1. Technical Specification
    1) Operating System : Ubuntu 16.04 LTS AMD64 (Compulsory)
    2) Server Framework : .NET Core 1.0 (1.0.0-rc3-004530)
    3) Client Framework : AngularJS
    4) Database : In-Memory Cache not using Redis Cache
    5) Source Control : git
    6) Development IDE : Visual Studio Code
    7) Load Testing Tool : Open-Source wrk
    8) Production Server : Azure Web App

2. Overview of Web API
    1) Web API

    I assume the sample location data must be generated by calling POST /api/drivers/init. You are able to execute APIs through HTML5 Test Driver Page and you can open it after building/running/deploying on your local vm machine or my cloud server which is Azure.

API	Description	Request Body	Response Body
POST /api/drivers/init	This should be requested at the top of priority to generate sample data.	None	Message for processing
GET /api/drivers/getalllocation	This is just to check for sample data on the server.	None	Array of overall sample location items
GET /api/drivers	This is to find driver's locations within my condition.	Search condition ex) api/drivers?Id=0&Latitude=-6.5 &Longitude=120.65454&Radiu 0000&limit=10	Array of overall found location items

2) Customized "HTTP HEADERS" Against Every Requests

Header	Description
GOJEK-API-PROCESSING-ELAPSEDTIME:1	I have implemented this custom header to provide each API processing elapsed and measured time. You can easily analyze the performance value only with this HTTP header value. (unit is millisecond)
X-Content-Type-Options:nosniff	This allows to opt-out of MIME type sniffing.
X-Frame-Options:DENY	To provide clickjacking protection
X-XSS-Protection:1; mode=block	The HTTP X-XSS-Protection response header is a feature of Internet Explorer, Chrome and Safari that stops pages from loading when they detect reflected cross-site scripting (XSS) attacks.

3. Prerequisites

1) Installing Ubuntu 16.04 AMD64
http://kartolo.sby.datautama.net.id/ubuntu-cd/16.04/
2) Installing .NET Core & SDK
sudo sh -c 'echo "deb [arch=amd64]
https://apt-mo.trafficmanager.net/repos/dotnet-release/ xenial main" >
/etc/apt/sources.list.d/dotnetdev.list'
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 417A0893
sudo apt-get update
sudo apt-get install dotnet-dev-1.0.0-rc3-004530
3) Installing git Depending on your machine
sudo apt-get install git
4) Installing Library For Load Testing Tools
sudo apt-get install build-essential libssl-dev git -y
5) Building Load Testing Tool wrk
If you check for my Solution folder, there is a Tools directory. Under Tools directory
there is wrk folder. You can just run 'Make' command to build it.
6) Installing Visual Studio Code Depending on your prefer
https://code.visualstudio.com/download

4. Manual Build

1) Go to the main project folder which is 'FindGojeks'.
2) Type restore command with .NET Core compiler.
dotnet restore
3) Type build command with .NET Core compiler.
dotnet build --configuration Release
4) Go to the unit test project folder which is 'UnitTest'.
5) Type restore command with .NET Core compiler.
dotnet restore
6) Type build command with .NET Core compiler.
dotnet build --configuration Release
7) To run server, go to the main project root and type run command
dotnet run --configuration Release
8) To run unit test, go to the unit test root and type run command
dotnet run --configuration Release

5. Testing

1) Unit Testing

I implemented a separate unit test project with ‘Dependency Injection’ method using xUnit. It includes test methods in order to test the Controller’s methods corresponding to web apis. You can refer to my source code and run it according to above.

2) Load Testing

I used one of open source load test tools named wrk. You are able to build and install according to above steps. To launch the test, go to LoadTest folder and run bash shell script depending on server. I have implemented scripts only for /api/drivers API for test purpose. You can refer below for result of test example.

```
bash RunLoadTestOnLocal.sh
Running 5s test @ http://localhost:5000/
  2 threads and 2 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    95.97ms  293.60ms   1.50s    90.57%
    Req/Sec     1.68k   636.12     3.11k    81.40%
  14528 requests in 5.00s, 5.03MB read
  Non-2xx or 3xx responses: 14528
Requests/sec:   2902.92
Transfer/sec:      1.00MB
```

3) GUI Testing

According to below last paragraph, if you build all successfully, you will be able to test on the test driver web page which is based on AngulaJS. However you have to run the first web api to initialize sample location data and also you can refer http header values to understand the server side performance with ‘GOJEK-API-PROCESSING-ELAPSEDTIME’ header value.

6. Automated Build, Unit Test and Deployment to Azure

Inside of the main project folder there is a bash shell script file named ‘autodeploy.sh’.
It includes building, unit testing and deployment to azure cloud server. For deployment, it uses ‘git’ deployment method using git command. So you need to prepare git configuration such as ‘git configure --global user.name’ and ‘git configure --global user.email’ as prerequisite. If done, you can simply run the script with one parameter it is my azure git deployment remote address,
which is https://yh100002@gojektest22.scm.azurewebsites.net:443/gojektest22.git

- Run Script : bash autodeply.sh https://yh100002@gojektest22.scm.azurewebsites.net:443/gojektest22.git
- While running script, you can monitor the overall process and you have to authenticate password which is yh100002 and you can type it on prompt.

After integrated deployment process, you can test the web site on my Azure.
As you can see below, you will be able to test on it according to above calling sequence for sample location data.

http://gojektest22.azurewebsites.net/