D2D Vehicle Tracker LLD

# Solution at a glance

* RESTful API end points are created as per the requirement using express library for Node.js environment, that will listen to the location update from the Alligator Shuttles
* Mongo DB is being used to store every location updates received.
* Live visualization of the Alligator shuttles’ locations is created,
  + the static contents for the front-end App are delivered using Node.js app
  + Live visualization App is merged with REST API App for easier deployment and can be decoupled.

# 1. Visualization App

The code for visualization front-end app is merged with the REST API server as of now for the easiness of deployment and infra limitations.

URL : <http://domain/>

Demo: <http://d2d.shakkir.com/>

## UI and UX

In addition to the Map, a few option are added to the UI

1. Textboxes and Buttons to Set city center and coverage in order to control these factors from front-end
2. Freeze/Go Live Button to pause/resume live tracking

Note: Auto update can be set to stop live update after a fixed interval using STOP\_LIVE\_AFTER constant from [globals.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/web/res/globals.js)

## Design and Pseudocode

Express route at [route.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/lib/route.js) delivers index.html also exposes static content (html, JavaScript, CSS, images etc.) via <http://domain/> end-point.

External dependencies

1. JQuery
2. Google maps JavaScript API
3. Google map’s Marker plugin
4. JQuery Easing plugin
5. Marker Animate plugin
6. Sliding Marker plugin

Google map’s API KEY and the callback method initMap() are mentioned at index.html while including google map’s plugin script.

### main.js

Execution begin at [main.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/web/res/main.js) as the map plugin invokes initMap() method.

* The method initializes maps object and tracker object (VehicleTracker constructor from [tracker.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/web/res/tracker.js)) Then invokes goLive() method.
* goLive() invokes requestLocations() method immediately, also starts a timer to continuously invoke requestLocations() method for every 3 seconds (configurable at constants.js)
* requestLocations method makes request GET /locations with city center and boundary info.

End point : “GET /locations”

Query params:

{"center": {"lat": 52.523, "lng": 13.424 }, "range": 3.5 }

Using this Query parameter, the visualization app can control the center and the boundaries of the city to be visualized.

Sample response:

{"f616c6c8-33f2-4c98-9694-4abac3d46199":{"lat":52.53339,"lng":13.43917,"at":"2019-09-22T12:07:19.448Z"},"872e0bdb-4552-40b1-ba3a-9285b37659db":{"lat":52.55506,"lng":13.4148,"at":"2019-09-22T12:07:19.451Z"}}

The active vehicles data object received as response sent to tracker object for tracking via trackAll(data) method

### VehicleTracker Class

This class keeps tracks of the vehicles and their movement and visualizes them in the map by the help of Vehicle Class

#### Constructor: VehicleTracker(google.maps.Map)

Initiates properties and invokes init() method

Properties:

|  |  |
| --- | --- |
| property | details |
| map | Current google map object reference |
| vehiclesTracked | List of Vehicle Objects |
| cluster | Marker cluster object reference |

Methods:

|  |  |
| --- | --- |
| method | details |
| init | Initial setup |
| trackAll(locations) | Start tracking a bunch of vehicles |
| track(id, location) | Start tracking single Vehicle |
| stopTracking(id) | Stop tracking a vehicle |
| VehicleTracker.zoomChange() | Called upon map’s zoom change |

#### Method init()

This is called from the constructor to initialize google map circle (to mark city premises) and Clustering functionality

#### Method trackAll(locations)

This method iterate vehicle data and calls track() method for each vehicle to create/update vehicle object.

Also identifies missing vehicle updates and invokes stopTracking(id) method to remove them from vehiclesTracked list

#### Method track(id,location)

If the vehicle is not present in the vehiclesTracked list then creates a new Vehicle object (from [vehicle.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/web/res/vehicle.js)) and adds its marker to the cluster object to update clustering

Otherwise, fetches the vehicle object from the vehiclesTracked list and invokes moveTo() method to update the vehicle location

#### Method stopTracking(id)

If the id exists in vehiclesTracked then

* Removes its marker from the cluster
* Removes trigger vehicle.detroy() method
* Deletes vehicle from vehicleTracked list

#### Method VehicleTracker.zoomChange()

This method is called on map’s zoom change event. Modifications to the visualization such as marker dimensions are set here.

### Vehicle Class

Manages the lifecycle of a vehicle on the map

Properties:

|  |  |
| --- | --- |
| property | details |
| id | Unique id of the vehicle |
| location | Coordinates in json format such as {lat: 12.9979566,lng: 77.7191163} |
| marker | Google.maps.Marker / SlidingMarker Object created for the vehicle |

Methods:

|  |  |
| --- | --- |
| method | details |
| create(id, location, map) | Create marker |
| destroy() | Remove vehicle from map |
| moveTo(newLocation) | Update location of the vehicle |

#### Constructor: Vehicle(id, location, google.maps.Map)

Initiates properties and triggers create() method

#### Method moveTo(newLocation)

Id the position has changed then

* Calculate the direction of movement
* Get the SVG car icon (from [car-black.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/web/res/car-black.js)) with a rotation indicating the direction of movement by the help of getCarIcon(from, to) and getBearingAngle(from, to)
* Updates the marker position with new location

Since the SlidingMarker is used in place of google.maps.Marker; a change in position will be animated smoothly

### Configurations and constants

Many behaviors of the application are configurable at [globals.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/web/res/globals.js)

CENTER =  { lat: 52.53, lng: 13.403 };//center of the city

RANGE = 3.5; //range to consider

ZOOM\_LEVEL=13; //initial zoom of the map

CAR\_SIZE=1.2; //relative size of the car svg

CLUSTER\_CAR\_IMG = "img/cars-m"; //cluster image

LOCATIONS\_REAST\_URL= "/locations" //locations API URL

CLUSTER\_MIN\_SIZE = 3; //minimum of 3 cars in a confined place will be show as one cluster

CLUSTER\_GRID\_SIZE = 70; //size of the grid where the cars has to be to show as one cluster

UPDATE\_RATE = 3\*1000; //rate at which location update is requested milsec

STOP\_LIVE\_AFTER = 0// 15\*60\*1000; for 15 minutes

# 2. RESTful API

Code repo: <https://github.com/shakkirptb/vehicle-tracker>

## Dependencies

1. Server Env: Node.js v10.16.0
2. MongoDB v4.20

The RESTful API end-points are created using express library in for Node.js server environment.

## Design and Pseudocode

### 1. Start of execution, index.js:

>”npm start” triggers [index.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/index.js)

Express route at [route.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/lib/route.js) handles inbound requests and it is imported to Index.js and the API will listen to the port provided in the properties file ([properties.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/lib/properties.js)). Default port: 3001.

### Properties,

Constants and default properties are stored at [properties.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/lib/properties.js)

### Express route

REST end-points for register, update-location unregister vehicles and get-updates (new) are configured at [route.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/lib/route.js)

Data from the requests are passed to updateVehicleStats (id, newLocation) method for processing.

### Business logic

The object activeVehicles keeps track of all the vehicles registered and running inside the city. And inactiveVehicles tracks all the vehicles outside the city. Together they helps to identify vehicles that has to be accounted for visualization. This is to reduce response JSON size.

updateVehicleStats method determines active vehicles and inactive vehicles and updates the respective constants. Each location update is passed to insert(document) method of [database.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/lib/database.js) for storing to db.

### Database manager [database.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/lib/database.js)

Since 100s of request are received at every moment, sending update for every location update will hamper the performance by increasing process, network traffic and the load on the servers. Hence, the data received by the insert method is added to a buffer object( locationBuffer array). and the data is inserted to the DB in a bunch at regular intervals as configured in [properties.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/lib/properties.js) file (default 5 sec). The timer will starts only on a request and it will wait for a fixed time, then invokes insertBatch method to save the buffer to the DB and then clears the buffer.

### REST End-Point for Visualization

In addition to the end-points register, update, delete, a new end-point has been created that will be consumed by the visualization app in order to fecth all vehicles’ update.

The express route code for this end-point sends activeVehicles objects as response.

End point : “GET /locations”

Query params:

{"center": {"lat": 52.523, "lng": 13.424 }, "range": 3.5 }

Using this Query parameter, the visualization app can modify the center and the boundaries of the city to be visualized.

Sample response:

{"f616c6c8-33f2-4c98-9694-4abac3d46199":{"lat":52.53339,"lng":13.43917,"at":"2019-09-22T12:07:19.448Z"},"872e0bdb-4552-40b1-ba3a-9285b37659db":{"lat":52.55506,"lng":13.4148,"at":"2019-09-22T12:07:19.451Z"}}

### Configurations

Configurations for the app can be set at [properties.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/lib/properties.js)

{

    APP\_PORT: 3001, //server port

    action: { UNREGISTER: 0, REGISTER: 1, UPDATE: 2, AWAY: 3 },

    CITY\_CENTER: { lat: 52.53, lng: 13.403 },

    CITY\_RANGE: 3.5,

    DB\_UPDATE\_INTERVAL:5000,

    DB\_URL: "mongodb://127.0.0.1:27017/", // mongodb connection url

    DB\_DB:"d2d", // mongodb database name

    DB\_COLLECTION:"locations" //mongodb collection name where location updates are has to be stored

}

# 3. Database

MongoDB is being used as DB for its rich query features, high availability and compatibility with JSON data. Given below are the DB details to be configured at the [properties.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/lib/properties.js) file

MongoDB database name: d2d

Collection: locations

Sample document:

{

"vid" : "2e69d162-4f3e-489e-9239-c6090e3f0650",

"location" : {

"lat" : 52.43926,

"lng" : 13.40752,

"at" : "2019-09-22T11:57:08.852Z"

},

"action" : 3

}

action denotes the update given by a vehicle asper below, is configurable at [properties.js](https://github.com/shakkirptb/vehicle-tracker/blob/master/lib/properties.js)

{ UNREGISTER: 0, REGISTER: 1, UPDATE: 2, AWAY: 3 }

# --The End--