

TRADER INTERACTIVE - SKILL EXERCISE

02/10/2020

Hariharan Thiyagarajan

hariharan0907@gmail.com

Tech stack for constructing the API

REST API using

- Node.js
- Express.js
- Mongo DB

Why REST?

REST API is always independent of the type of the platform or languages. the REST API always adapts to the type of syntax or platforms being used, which gives considerable freedom when changing or testing new environments within the development. Added to that, reliability and scalability is high.

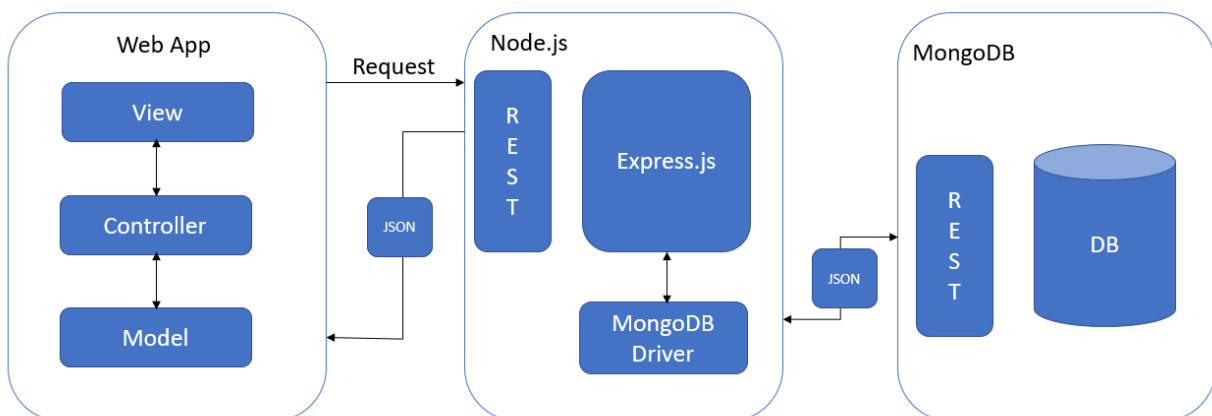
Why Node.js, Express.js, Mongo DB?

- Node.js & Express.js – Using Express framework along with Node.js provide easy way to declare API, handle incoming parameters, errors, transformation to JSON, streaming and sending response.
- MongoDB – In our scenario vehicle data is huge, and the specifications are subjected to change, then schema less & flexible database becomes a natural choice

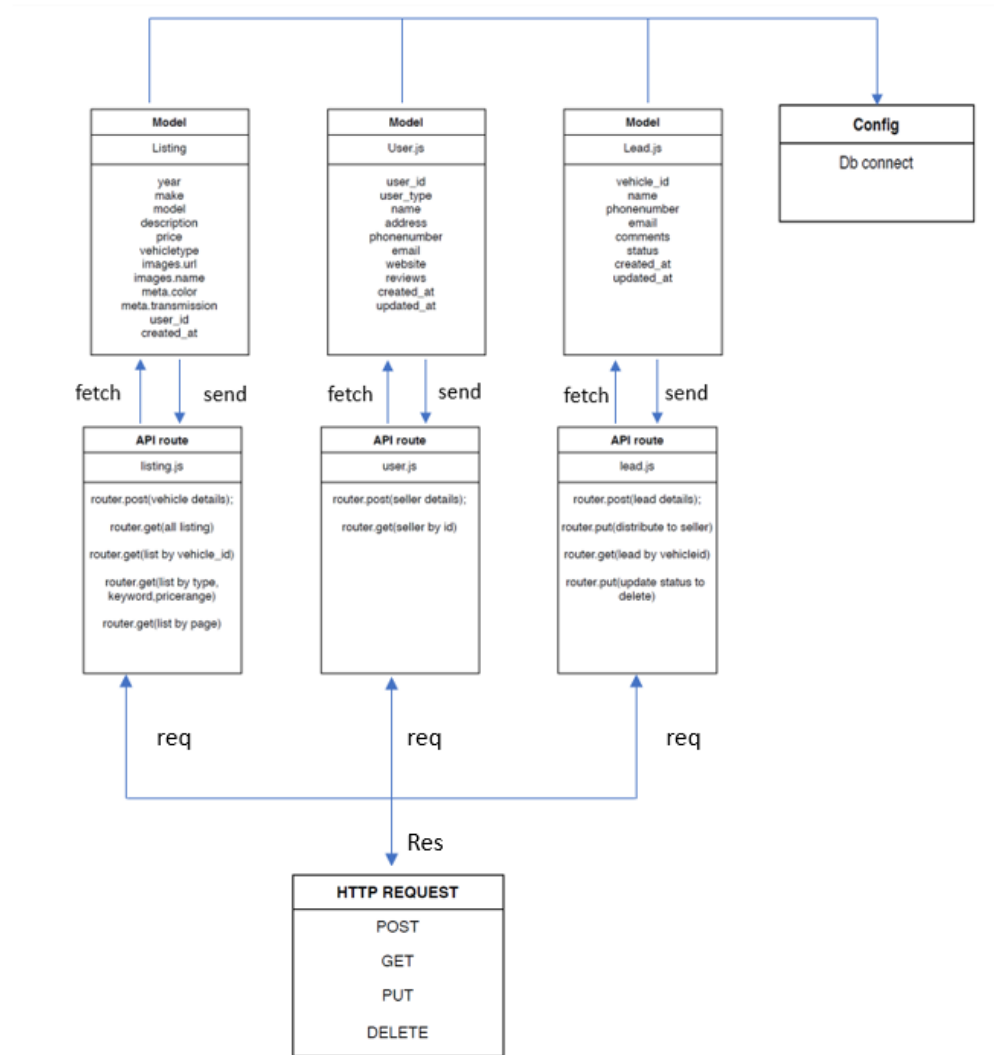
Source code for all the API routes are available in my GitHub repository

Source code: <https://github.com/hthiyaga/REST-API-using-mongodb-express.js-node.js>

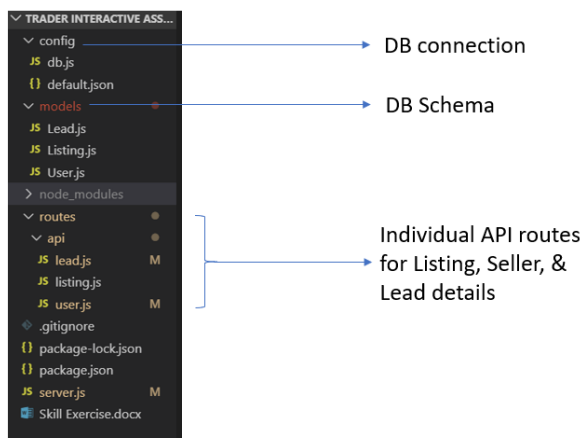
Architecture of REST API using Node.js, Express.js & MongoDB



API Workflow diagram



Folder Structure



Data Model

I'm using Mongo DB to design the database for vehicle listing and selling. It is a NoSQL database and hence the flexibility is more.

There will be three collections, one for listing of vehicles, one for user details and one more for lead details.

Listing Collection

```
{
  "_id": ObjectId("5e3dcc797d620c268c6f5d44"),
  "year": 2015,
  "make": "Travel Lite RV",
  "model": "625SL",
  "description": "Good",
  "price": 32000,
  "vehicleType": "RV",
  "meta": {
    "color": "white",
    "transmission": "Gear"
  },
  "images": [
    {
      "_id": ObjectId("5e3dcc797d620c268c6f5d46"),
      "url": "www.ads.com",
      "name": "logo"
    },
    {
      "_id": ObjectId("5e3dcc797d620c268c6f5d47"),
      "url": "www.asdd.com",
      "name": "picture2"
    }
  ],
  "user_id": 1,
  "created_at": 2020-02-08T20:47:49.851+00:00,
  "__v": 0
}
```

User Collection

```
{
  "_id": ObjectId("5e3cf596f335ae2c7cd30fa6"),
  "reviews": [
    {
      "0": "good",
      "1": "okay"
    }
  ],
  "user_id": 1,
  "user_type": "dealer",
  "name": "Hari",
  "address": "Norfolk",
  "phonenumber": "67987987987",
  "email": "hari@gmail.com",
  "website": "...com",
  "created_at": 2020-02-08T20:50:49.851+00:00,
  "__v": 0
}
```

Lead Collection

```
{
  "_id": ObjectId("5e3ebd5e1c9d4400004dccbe"),
  "vehicle_id": "5e3dcca67d620c268c6f5d48",
  "name": "Santa",
  "phonenumber": 67867668,
  "email": "santa@gmail.com",
  "comments": "I'm interested in your merc",
  "status": "stored",
  "created_at": 2020-02-08T22:51:38.833+00:00,
  "updated_at": 2020-02-08T22:56:16.691+00:00
}
```

Relationship between user and listing

```
[ {
  "_id": "5e3d0e5cd1b083258437f004",
  "year": 2015,
  "make": "Merc",
  "model": "Benz",
  "description": "Good",
  "price": 32000,
  "vehicletype": "Car",
  "meta": {
    "color": "black",
    "transmission": "Gearless"
  },
  "images": [
    {
      "_id": "5e3dcd11e9495041f4a4e474",
      "url": "www.adsasd.com",
      "name": "logo"
    },
    {
      "_id": "5e3dcd11e9495041f4a4e475",
      "url": "www.asddsd.com",
      "name": "picture2"
    }
  ],
  "created_at": "2020-02-08T20:47:49.851Z",
  "user": {
    "user_id": 1,
    "user_type": "dealer",
    "name": "Hari",
    "address": "Norfolk",
    "phonenumber": "6798798797",
    "email": "hari@gmail.com",
    "website": "...com",
    "reviews": [
      "good",
      "okay"
    ],
    "created_at": "2020-02-08T20:50:49.851Z"
  },
}
```

API functional prototype

All vehicle listing

End point: <http://localhost:3000/api/listing>

HTTP request: **GET**

```
router.get('/', async (req, res) => {  
  try {  
    const listing = await Listing.find().populate();  
    res.json(listing);  
  }  
}
```

The screenshot shows a web browser interface with a GET request to `http://localhost:3000/api/listing`. The response is displayed in JSON format, showing a list of two vehicle listings. The first listing is a white Travel Lite RV with a price of 32000, and the second listing is a black Audi Q7 with a price of 50000. Both listings include metadata such as color, transmission, year, make, model, description, price, vehicle type, and images.

```
1 {  
2   {  
3     "meta": {  
4       "color": "white",  
5       "transmission": "Gear"  
6     },  
7     "_id": "5e3dcc797d620c268c6f5d44",  
8     "year": 2015,  
9     "make": "Travel lite RV",  
10    "model": "625SL",  
11    "description": "Good",  
12    "price": 32000,  
13    "vehicletype": "RV",  
14    "images": [  
15      {  
16        "_id": "5e3dcc797d620c268c6f5d46",  
17        "url": "www.ads.com",  
18        "name": "logo"  
19      },  
20      {  
21        "_id": "5e3dcc797d620c268c6f5d47",  
22        "url": "www.asdd.com",  
23        "name": "picture2"  
24      }  
25    ],  
26    "user_id": 1,  
27    "__v": 0,  
28    "created_at": "2020-02-09T20:18:51.282Z"  
29  },  
30  
31  {  
32    "meta": {  
33      "color": "red",  
34      "transmission": "Gearless"  
35    },  
36    "_id": "5e3dcca67d620c268c6f5d48",  
37    "year": 2015,  
38    "make": "Merc",  
39    "model": "Benz",  
40    "description": "Good",  
41    "price": 30000,  
42    "vehicletype": "car",  
43    "images": [  
44      {  
45        "_id": "5e3dcca67d620c268c6f5d4a",  
46        "url": "www.ads.com",  
47        "name": "logo"  
48      },  
49      {  
50        "_id": "5e3dcca67d620c268c6f5d4b",  
51        "url": "www.asdd.com",  
52        "name": "picture2"  
53      }  
54    ],  
55    "user_id": 1,  
56    "__v": 0,  
57    "created_at": "2020-02-09T20:18:51.282Z"  
58  },  
59  
60  {  
61    "meta": {  
62      "color": "black",  
63      "transmission": "Gearless"  
64    },  
65    "_id": "5e3dcd11e9495041f4a4e473",  
66    "year": 2017,  
67    "make": "Audi",  
68    "model": "Q7",  
69    "description": "Good",  
70    "price": 50000,  
71    "vehicletype": "car",  
72    "images": [  
73      {  
74        "_id": "5e3dcd11e9495041f4a4e474",  
75        "url": "www.adsasdd.com",  
76        "name": "logo"  
77      },  
78      {  
79        "_id": "5e3dcd11e9495041f4a4e475",  
80        "url": "www.asddsd.com",  
81        "name": "picture2"  
82      }  
83    ],  
84    "user_id": 1,  
85    "__v": 0,  
86    "created_at": "2020-02-09T04:35:26.564Z"  
87  }  
88  ]  
89 }
```

User search based on Vehicle Type, Keyword & Price range

When the user searches for following details in the form

- Vehicle type
- Keyword
 - It can contain anything from model to metadata of the vehicle
- Price range

On form submission, it will hit the following end point.

End point: <http://localhost:3000/api/listing/searchby>

HTTP request: **GET**

In the backend, it will check the query parameter **?type= &keyword= &fromrange= &torange=** selected in the form and finds the corresponding result from the lists collections in DB.

```
router.get('/searchby/', async (req, res) => {
  try {

    const listing = await Listing.find({
      vehicletype: req.query.type,
      $or: [
        { year: req.query.keyword },
        { make: req.query.keyword },
        { model: req.query.keyword },
        { description: req.query.keyword },
        { 'meta.color': req.query.keyword },
        { 'meta.transmission': req.query.keyword }
      ],
      price: { $gt: req.query.fromrange, $lt: req.query.torange }
    }).populate();
    res.json(listing);
  }
});
```

The below screenshot provides result for
type=car&keyword=red&fromrange=10000&torange=40000

```
GET http://localhost:3000/api/listing/searchby/?type=car&keyword=red&fromrange=20000&torange=40000 Send

Pretty Raw Preview Visualize BETA JSON

1 [
2   {
3     "meta": {
4       "color": "red",
5       "transmission": "Gearless"
6     },
7     "_id": "5e3dcca67d620c268c6f5d48",
8     "year": 2015,
9     "make": "Merc",
10    "model": "Benz",
11    "description": "Good",
12    "price": 30000,
13    "vehicletype": "car",
14    "images": [
15      {
16        "_id": "5e3dcca67d620c268c6f5d4a",
17        "url": "www.ads.com",
18        "name": "logo"
19      },
20      {
21        "_id": "5e3dcca67d620c268c6f5d4b",
22        "url": "www.asdd.com",
23        "name": "picture2"
24      }
25    ],
26    "user_id": 1,
27    "_v": 0,
28    "created_at": "2020-02-09T20:17:57.019Z"
29  }
30 ]
```

```
GET http://localhost:3000/api/listing/searchby/?type=car&keyword=Gearless&fromrange=20000&torange=50000

Pretty Raw Preview Visualize BETA JSON

1 [
2   {
3     "meta": {
4       "color": "red",
5       "transmission": "Gearless"
6     },
7     "_id": "5e3dcca67d620c268c6f5d48",
8     "year": 2015,
9     "make": "Merc",
10    "model": "Benz",
11    "description": "Good",
12    "price": 30000,
13    "vehicletype": "car",
14    "images": [
15      {
16        "_id": "5e3dcca67d620c268c6f5d4a",
17        "url": "www.ads.com",
18        "name": "logo"
19      },
20      {
21        "_id": "5e3dcca67d620c268c6f5d4b",
22        "url": "www.asdd.com",
23        "name": "picture2"
24      }
25    ],
26    "user_id": 1,
27    "_v": 0,
28    "created_at": "2020-02-09T20:13:25.266Z"
29  },
30 ],
31 {
32   "meta": {
33     "color": "black",
34     "transmission": "Gearless"
35   },
36   "_id": "5e3dcd11e9495041f4a4e473",
37   "year": 2017,
38   "make": "Audi",
39   "model": "Q7",
40   "description": "Good",
41   "price": 50000,
42   "vehicletype": "car",
43   "images": [
44     {
45       "_id": "5e3dcd11e9495041f4a4e474",
46       "url": "www.adsasd.com",
47       "name": "logo"
48     },
49     {
50       "_id": "5e3dcd11e9495041f4a4e475",
51       "url": "www.asddsd.com",
52       "name": "picture2"
53     }
54   ],
55   "user_id": 1,
56   "_v": 0,
57   "created_at": "2020-02-09T04:56:40.873Z"
58 }
```


Handling pagination in the API

Since I'm working with dummy data, I'm not able to show the functional prototype of how pagination works in the API. The logical explanation of pagination results as follows:

Endpoint: <http://localhost:3000/listing/>

HTTP request: **GET**

The query parameter will be **?pagenum=**

Based on the number entered we can compute the following in the backend

```
let page_size = 10;
skips = page_size * (req.query.pagenum - 1);
//for page 1
// skips =10 *(1-1);
//skips =0
// It will fetch first 10 records
//for page 2
// skips =10 *(2-1);
//skips =10
// It will skip the first 10 records fetches from 10-20
const listing = await Listing.find()
  .skip(skips)
  .limit(page_size).populate;
```

Each listing result based on the pagination contains data and metadata which can be rendered in the frontend accordingly.

Since multiple results will be populated, in order to access the specific resource, it should hit another end point which carries the unique id of it.

Access specific resource based on URI

Endpoint: <http://localhost:3000/listing/searchby/:id>

HTTP request: **GET**

Since ID is unique it will access only that item from DB

```
router.get('/searchby/:id', async (req, res) => {
  try {
    const listing = await Listing.find({ _id: req.params.id }).populate();
    res.json(list);
  }
});
```

```
GET http://localhost:3000/api/listing/searchby5e3dcca67d620c268c6f5d48 Send Si

Pretty Raw Preview Visualize BETA JSON

1 {
2   {
3     "meta": {
4       "color": "red",
5       "transmission": "Gearless"
6     },
7     "_id": "5e3dcca67d620c268c6f5d48",
8     "year": 2015,
9     "make": "Merc",
10    "model": "Benz",
11    "description": "Good",
12    "price": 30000,
13    "vehicletype": "car",
14    "images": [
15      {
16        "_id": "5e3dcca67d620c268c6f5d4a",
17        "url": "www.ads.com",
18        "name": "logo"
19      },
20      {
21        "_id": "5e3dcca67d620c268c6f5d4b",
22        "url": "www.asdd.com",
23        "name": "picture2"
24      }
25    ],
26    "user_id": 1,
27    "_v": 0,
28    "created_at": "2020-02-09T20:31:07.504Z"
29  }
30 }
```

Accessing reviews of the seller

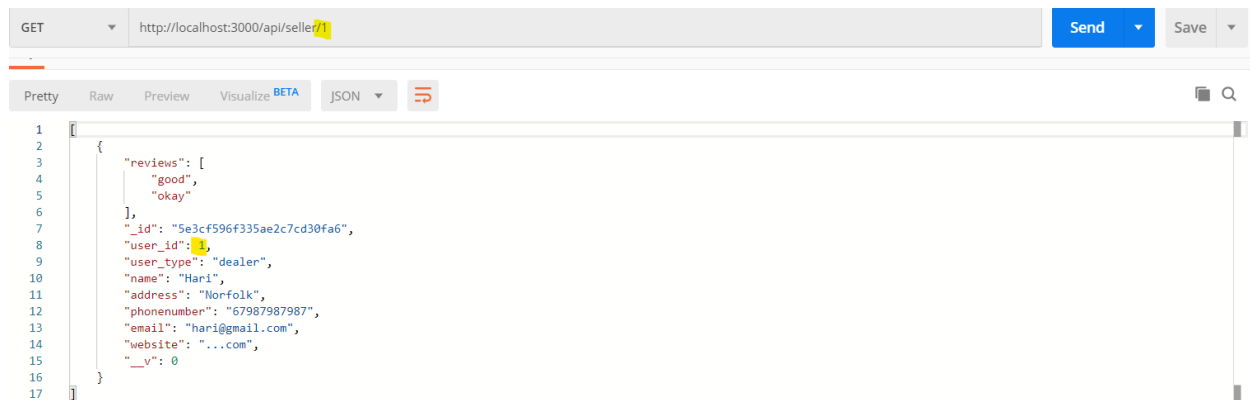
Each listing will have a user id who posted it, when buyer wants to access the reviews of an user, it will be taken as parameter to fetch the necessary details.

End point: : <http://localhost:3000/api/seller/:id>

HTTP request: **GET**

In backend, it will fetch the seller details based on the user_id

```
router.get('/:id', async (req, res) => {
  try {
    const seller = await User.find({ user_id: req.params.id }).populate();
```



Lead submission

Ideally, when the buyer is interested in some vehicle, he/she will submit the following details in the form as a POST request to the API

- Name
- Phone number
- Email
- Comments about the vehicle he/she is interested in

End point: <http://localhost:3000/api/list>

HTTP request: **POST**

Along with this we will be sending vehicle_id and softdelete option to the dB in the backend, which will be used by the seller to update the status of the lead later.

```
router.post('/', async (req, res) => {  
  
  const { vehicle_id, name, phonenumber, email, softdelete } = req.body;  
  try {  
    let lead = new Lead({  
      vehicle_id,  
      name,  
      phonenumber,  
      email,  
      softdelete  
    });  
  
    await lead.save();  
  }  
}
```

The initial status field will be “**stored**” once the buyer submits the request.

```
GET http://localhost:3000/api/lead/5e3dcc797d620c268c6f5d44 Send
Pretty Raw Preview Visualize BETA JSON
{
  "_id": "5e3f48a599f4883d0c94c359",
  "vehicle_id": "5e3dcc797d620c268c6f5d44",
  "name": "asd",
  "phonenumber": 23232323,
  "email": "sddgmail.com",
  "status": "stored",
  "created_at": "2020-02-08T23:47:49.851Z",
  "updated_at": "2020-02-08T23:47:49.851Z",
  "__v": 0
}
```

Later the status will be changed to “**distributed**” after the verification is done.

In order to change in to “**distributed**”, it should be a **PUT** request which will carry the unique id of each leads

End point: <http://localhost:3000/api/lead/:id>

HTTP request: **PUT**

```
// change status to distributed
router.put('/:id', async (req, res) => {
  try {
    let lead = await Lead.findOne({ _id: req.params.id }).populate();
    lead.status = 'Distributed';
    await lead.save();

    res.json(lead);
  }
});
```

```
PUT http://localhost:3000/api/lead/5e3f48a599f4883d0c94c359 Send
Pretty Raw Preview Visualize BETA JSON
{
  "_id": "5e3f48a599f4883d0c94c359",
  "vehicle_id": "5e3dcc797d620c268c6f5d44",
  "name": "asd",
  "phonenumber": 23232323,
  "email": "sddgmail.com",
  "comments": "I'm interested",
  "status": "Distributed",
  "created_at": "2020-02-08T23:47:49.851Z",
  "updated_at": "2020-02-08T23:47:49.851Z",
  "__v": 0
}
```

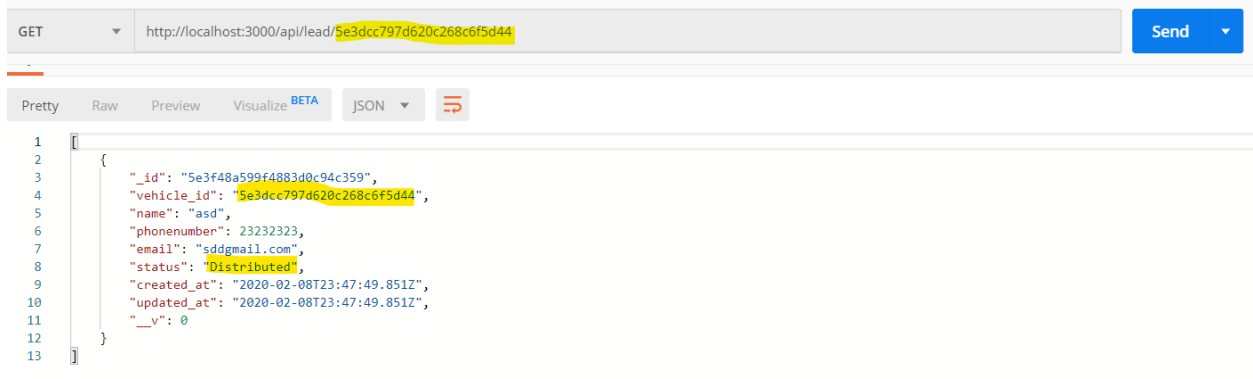
Seller Review of Leads

When the seller wants to access the leads received, the endpoint should carry the vehicle_id so that, it will display the leads for the corresponding vehicles

End point: <http://localhost:3000/api/lead/:id>

HTTP request: **GET**

```
router.get('/:id', async (req, res) => {
  try {
    const lead = await Lead.find({
      vehicle_id: req.params.id,
      status: 'Distributed'
    }).populate();
  }
```



The one major drawback is any seller can view anyone's lead details if they know the endpoint because it is not secured. In order to make it secure, we should use **JWT (JSON web token)** so when someone else tries to access it, it will throw an error as it won't match with the secret key.

Handling Deletion

If the seller wants to update the status of the lead, he should submit a Delete request to the API to do a soft delete

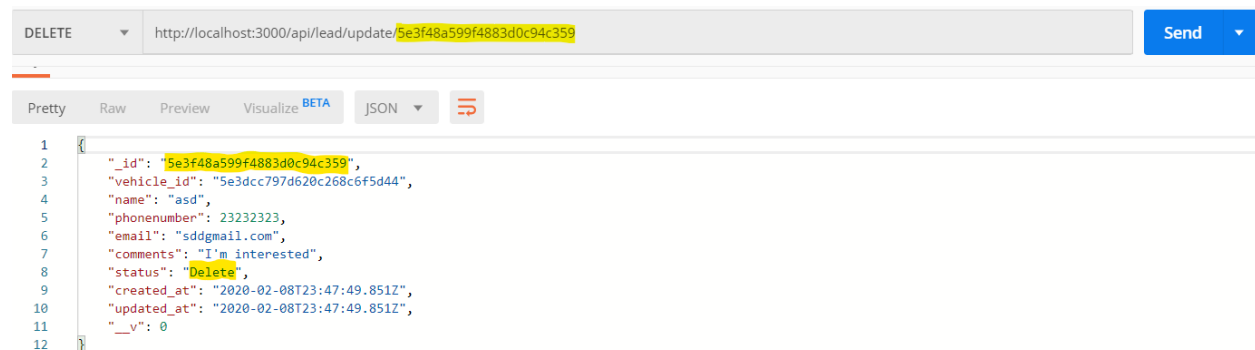
Endpoint: <http://localhost:3000/api/lead/update/:id>

HTTP request: **DELETE**

The unique id created when submitting the lead will be taken as a parameter to update the status of the lead.

This request should also be carried out with JWT.

```
router.delete('/update/:id', async (req, res) => {
  try {
    let lead = await Lead.findOne({ _id: req.params.id }).populate();
    lead.status = 'Delete';
    await lead.save();
  }
})
```



The screenshot shows a REST client interface. At the top, a dropdown menu is set to 'DELETE' and the URL is 'http://localhost:3000/api/lead/update/5e3f48a599f4883d0c94c359'. A 'Send' button is on the right. Below the URL bar, there are tabs for 'Pretty', 'Raw', 'Preview', and 'Visualize BETA', with 'JSON' selected. The response is displayed in a code editor with line numbers 1 through 12. The JSON object contains the following fields: '_id' (the same as the URL), 'vehicle_id', 'name' (value: 'asd'), 'phonenumber' (value: '23232323'), 'email' (value: 'sddgmail.com'), 'comments' (value: 'I'm interested'), 'status' (value: 'Delete'), 'created_at' (value: '2020-02-08T23:47:49.851Z'), 'updated_at' (value: '2020-02-08T23:47:49.851Z'), and '__v' (value: 0).

```
1 {
2   "_id": "5e3f48a599f4883d0c94c359",
3   "vehicle_id": "5e3dcc797d620c268c6f5d44",
4   "name": "asd",
5   "phonenumber": "23232323",
6   "email": "sddgmail.com",
7   "comments": "I'm interested",
8   "status": "Delete",
9   "created_at": "2020-02-08T23:47:49.851Z",
10  "updated_at": "2020-02-08T23:47:49.851Z",
11  "__v": 0
12 }
```