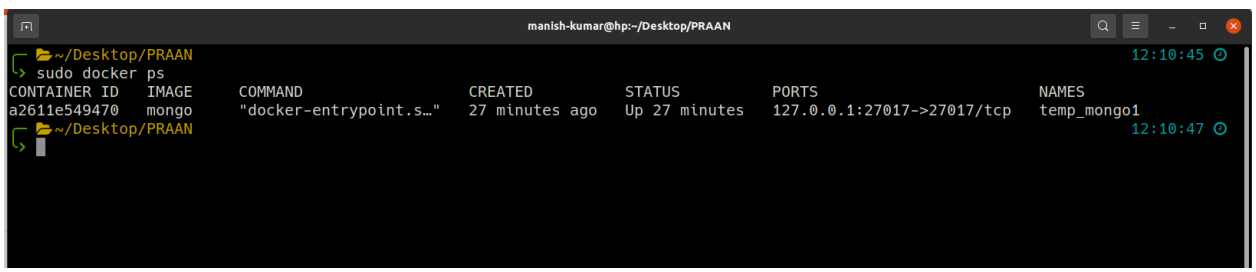
1.) Setting up mongoDB with Docker

```
$ sudo docker pull mongo
$ sudo docker run --name my_mongo -d -p 127.0.0.1:27017:27017 mongo
```

2.) To check if it is running, use given command

```
$ sudo docker ps
```

Following will appear after this command



Now the Database instance is running on your pc.

- **Following dependencies must be installed in order to compile and run the application, which are :**

```
13    "devDependencies": {
14      "@types/express": "^4.17.12",
15      "concurrently": "^6.2.0",
16      "express": "^4.17.1"
17    },
18    "dependencies": {
19      "@types/body-parser": "^1.19.0",
20      "@types/mongoose": "^5.10.5",
21      "body-parser": "^1.19.0",
22      "mongoose": "^5.12.11",
23      "multer": "^1.4.2",
24      "read-excel-file": "^5.1.0",
25      "sequelize": "^6.6.2"
26    }
```

3.) Finally compile the program by this command

```
$ npm run watch-ts
```

After running this command following will be appear :

```
npm run watch-ts
4056         [k: string]: string;
             ~~~~~~~~~~~~~~~~~~~~

node_modules/mongoose/index.d.ts:1:1 - error TS6200: Definitions of the followin
g identifiers conflict with those in another file: DocumentDefinition, FilterQue
ry, UpdateQuery, NativeDate, NativeError, Mongoose, SchemaTypes, STATES, connect
ion, connections, models, mongo, version, CastError, ConnectionOptions, Collecti
on, Connection, disconnected, connected, connecting, disconnecting, uninitialize
d, Error, QueryCursor, VirtualType, Schema, SchemaDefinition, SchemaTypeOpts, Su
bdocument, Array, DocumentArray, Buffer, ObjectId, ObjectIdConstructor, Decimal1
28, Map, Query, DocumentQuery, mquery, Aggregate, SchemaType, Promise, PromisePr
ovider, Model, Document, ModelUpdateOptions

1 declare module 'mongoose' {
  ~~~~~~~~

  node_modules/@types/mongoose/index.d.ts:78:1
    78 declare module "mongoose" {
       ~~~~~~~~
    Conflicts are in this file.

[1:25:00 pm] Found 97 errors. Watching for file changes.
```

4.) Then press “ctrl+c” .

5.) After this, run this command :

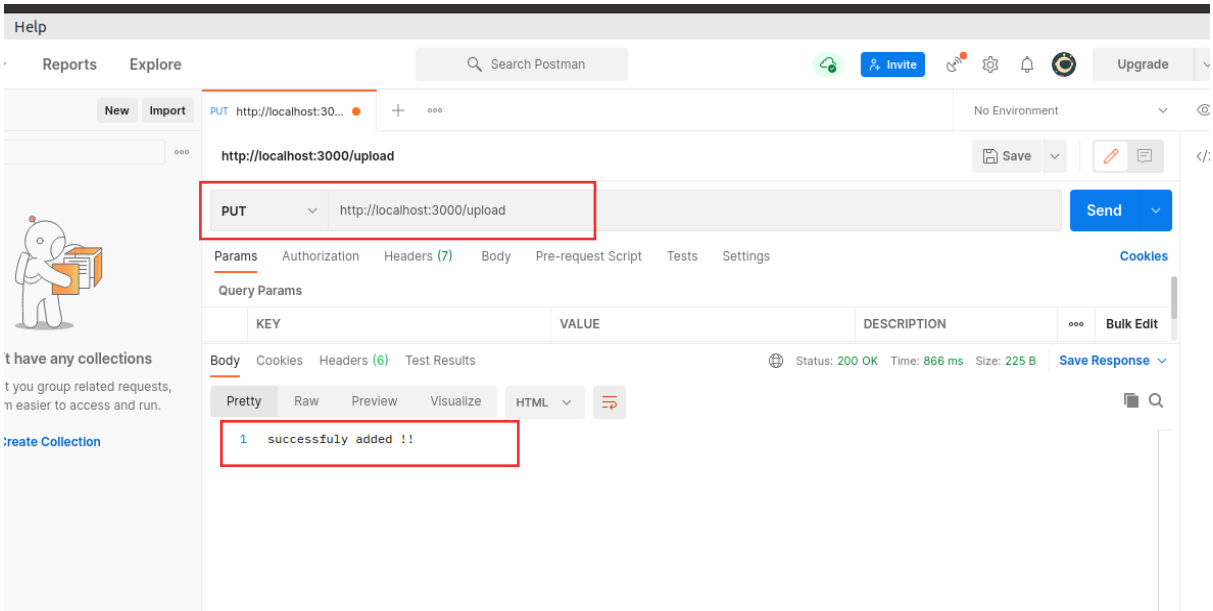
```
$ npm run watch
```

This will appear after that :

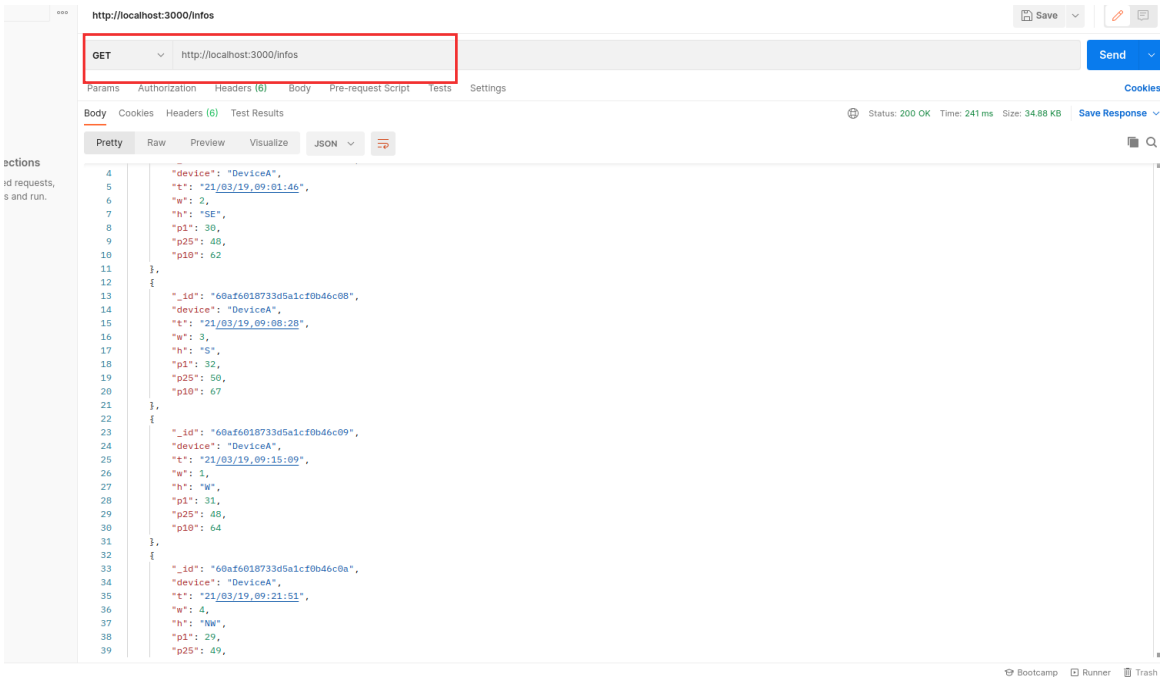
```
npm run watch
[TypeScript] node_modules/@types/mongoose/index.d.ts(4056,9): error TS2374: Dupl
icate string index signature.
[TypeScript] node_modules/mongoose/index.d.ts(1,1): error TS6200: Definitions of
 the following identifiers conflict with those in another file: DocumentDefiniti
on, FilterQuery, UpdateQuery, NativeDate, NativeError, Mongoose, SchemaTypes, ST
ATES, connection, connections, models, mongo, version, CastError, ConnectionOpti
ons, Collection, Connection, disconnected, connected, connecting, disconnecting,
 uninitialized, Error, QueryCursor, VirtualType, Schema, SchemaDefinition, Schem
aTypeOpts, Subdocument, Array, DocumentArray, Buffer, ObjectId, ObjectIdConstruc
tor, Decimal128, Map, Query, DocumentQuery, mquery, Aggregate, SchemaType, Promi
se, PromiseProvider, Model, Document, ModelUpdateOptions
[TypeScript]
[TypeScript] 1:29:46 pm - Found 97 errors. Watching for file changes.
[ Node] [nodemon] starting `node dist/app.js`
[ Node] (node:5210) DeprecationWarning: current URL string parser is deprecated,
 and will be removed in a future version. To use the new parser, pass option { u
seNewUrlParser: true } to MongoClient.connect.
[ Node] (node:5210) [MONGODB DRIVER] Warning: Current Server Discovery and Monit
oring engine is deprecated, and will be removed in a future version. To use the
new Server Discover and Monitoring engine, pass option { useUnifiedTopology: tru
e } to the MongoClient constructor.
[ Node] APP is running on http://localhost:3000
[ Node] successfully connected to MonogDB
```

- You will see it successfully connected, meaning your app is running and the database is connected to it.
- I used a postman application to send delete, post, and put requests. Because browsers do not support put, delete requests.
- These are the following query which my api executes:
    - **PUT:** http://localhost:3000/upload
    - **GET:** http://localhost:3000/infos
    - **PUT:** http://localhost:3000/info
    - **DELETE:** http://localhost:3000/info/:id
    - **POST:** http://localhost:3000/info/:id
    - **GET:** http://localhost:3000/info/pm1/:id
    - **GET:** http://localhost:3000/info/pm2.5/:id
    - **GET:** http://localhost:3000/info/pm10/:id
    - **GET:** http://localhost:3000/info/time-range/:id
    - **GET:** http://localhost:3000/info/:id

1.) PUT: http://localhost:3000/upload



2.) GET: http://localhost:3000/infos



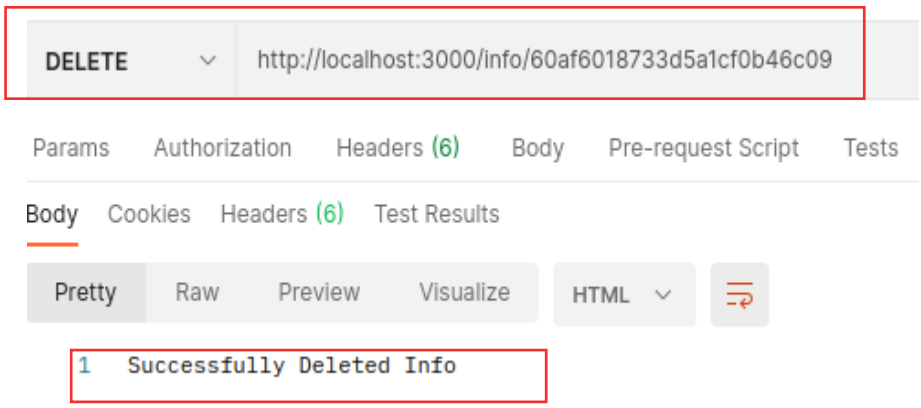
3.) **PUT: http://localhost:3000/info**

It adds a single object to the database.

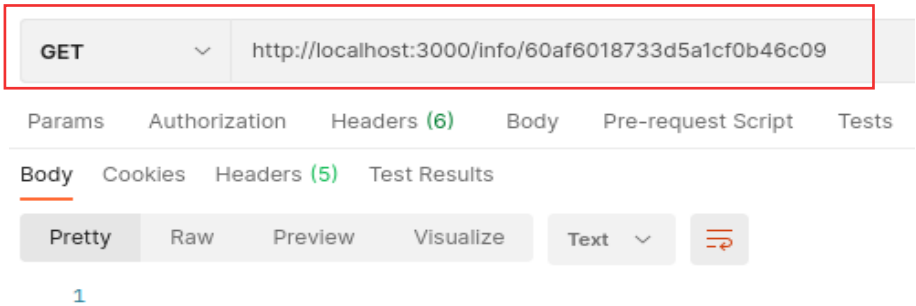
4.) **DELETE: http://localhost:3000/info/:id**

It deletes a single object from the database by the id provided. Where id is the default id provided By monogDB. The id looks like:


```
{
  "_id": "60af6018733d5a1cf0b46c08",
  "device": "DeviceA",
  "t": "21/03/19,09:08:28",
  "w": 3,
  "h": "S",
  "p1": 32,
  "p25": 50,
  "p10": 67
},
{
  "_id": "60af6018733d5a1cf0b46c09",
  "device": "DeviceA",
  "t": "21/03/19,09:15:09",
  "w": 1,
  "h": "W",
  "p1": 31,
  "p25": 48,
  "p10": 64
},
```

After deleting this I checked whether it is still present or not by "get" request :



But it does not show anything and this is true because i have deleted this, hence working correctly.

5.) POST: http://localhost:3000/info/:id

It will update the corresponding object whose id is being provided.

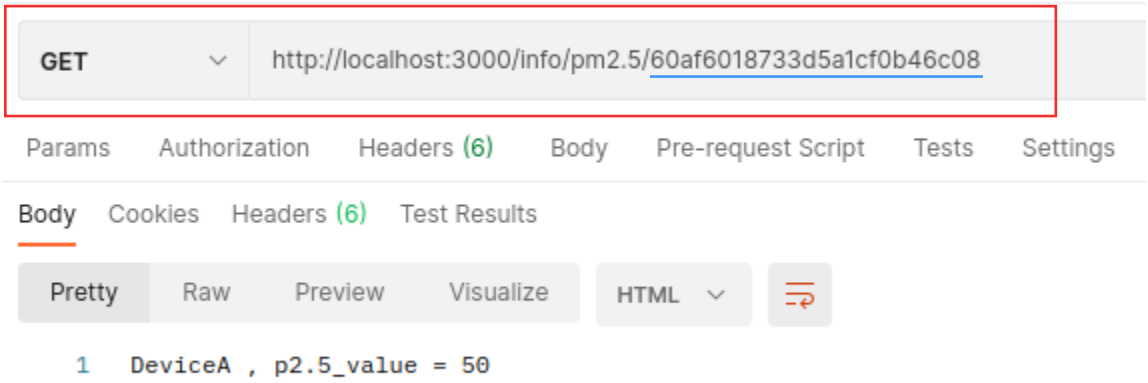
6.) GET: http://localhost:3000/info/pm1/:id

7.) GET: http://localhost:3000/info/pm2.5/:id

8.) GET: http://localhost:3000/info/pm10/:id

These above three work similarly, it will give results about pm1, pm2.5, pm10 specifically for the given device which is encoded with a default unique id.

http://localhost:3000/info/pm2.5/60af6018733d5a1cf0b46c08



http://localhost:3000/info/pm1/60af6018733d5a1cf0b46c08

GET http://localhost:3000/info/pm1/60af6018733d5a1cf0b46c08

Params Authorization Headers (6) Body Pre-request Script Tests

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize HTML

```
1  DeviceA , p1_value = 32
```

http://localhost:3000/info/pm10/60af6018733d5a1cf0b46c08

GET http://localhost:3000/info/pm10/60af6018733d5a1cf0b46c08

Params Authorization Headers (6) Body Pre-request Script Tests Set

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize HTML

```
1  DeviceA , p10_value = 67
```

http://localhost:3000/info/60af6018733d5a1cf0b46c08

GET http://localhost:3000/info/60af6018733d5a1cf0b46c08

Params Authorization Headers (6) Body Pre-request Script Tests

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "_id": "60af6018733d5a1cf0b46c08",
    "device": "DeviceA",
    "t": "21/03/19,09:08:28",
    "w": 3,
    "h": "S",
    "p1": 32,
    "p25": 50,
    "p10": 67
}
```


We can clearly match the value, it is same hence working correctly.

9.) GET: http://localhost:3000/info/time-range/:id

This query take only time-range (not date) as id and According to it, filter out all the devices which have time In between this range.

GET http://localhost:30...

http://localhost:3000/info/time-range/4:00:00-4:10:00

GET http://localhost:3000/info/time-range/4:00:00-4:10:00

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON


```
[
    {
        "_id": "60af6018733d5a1cf0b46caf",
        "device": "DeviceA",
        "t": "21/03/20,04:02:24",
        "w": 0,
        "h": "W",
        "p1": 61,
        "p25": 131,
        "p10": 178
    },
    {
        "_id": "60af6018733d5a1cf0b46cb0",
        "device": "DeviceA",
        "t": "21/03/20,04:09:05",
        "w": 0,
        "h": "W",
        "p1": 55,
        "p25": 118,
        "p10": 153
    },
    {
        "_id": "60af6018733d5a1cf0b46d00",
        "device": "DeviceA",
        "t": "21/04/01,04:04:21",
        "w": 7,
        "h": "NW",
        "p1": 19,
        "p25": 31,
        "p10": 44
    }
]
```


10.) GET: http://localhost:3000/info/:id

This is simple query which takes devices unique id and All info about its attributes.