Important points

- Project link
  https://github.com/bradtraversy/devconnector_2.0/tree/originalcoursecode

- Reference - MERN Stack Front To Back Full Stack React, Redux & Node.js

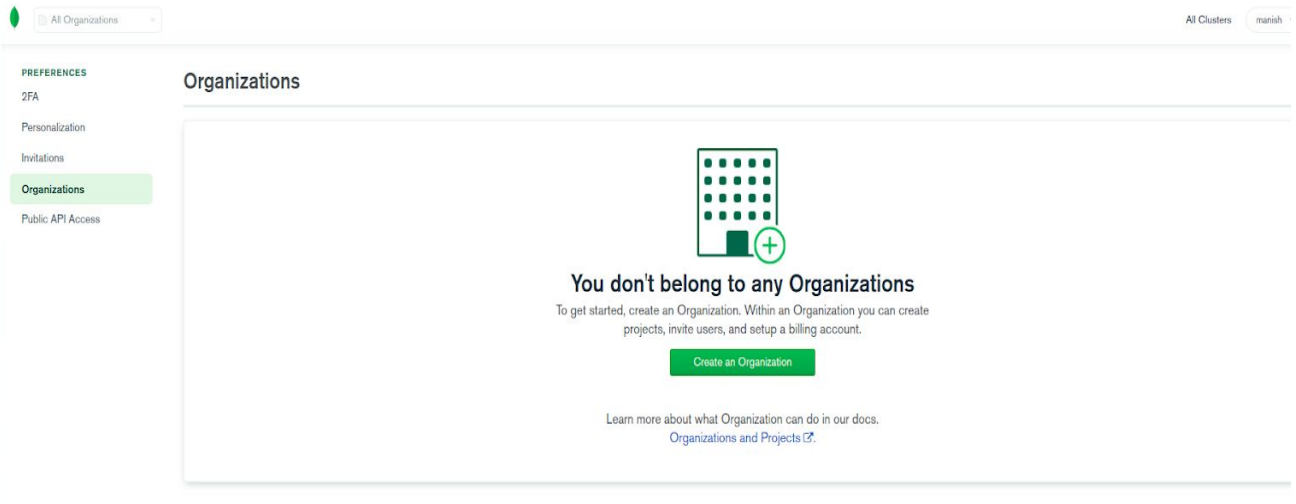
Building a developer-connect

Steps

1. **Mongo Atlas setup - we are using cloud database for following reason**
   a. It's easier to manage , you don't have to go for installation
   b. It is supported in ubuntu ,window etc
   c. We are going to deploy in heroku(PAS).

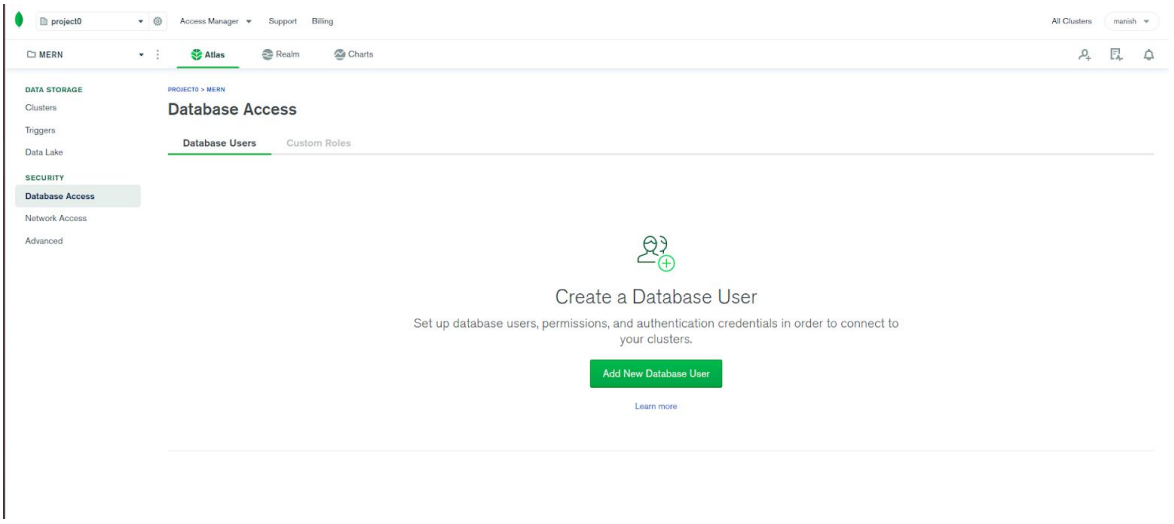
Process for mongo Atlas

a. Go to the link mongodb.com
b. Login (manishgoswami495@gmail.com pass-)
c. You will get the screen as below.

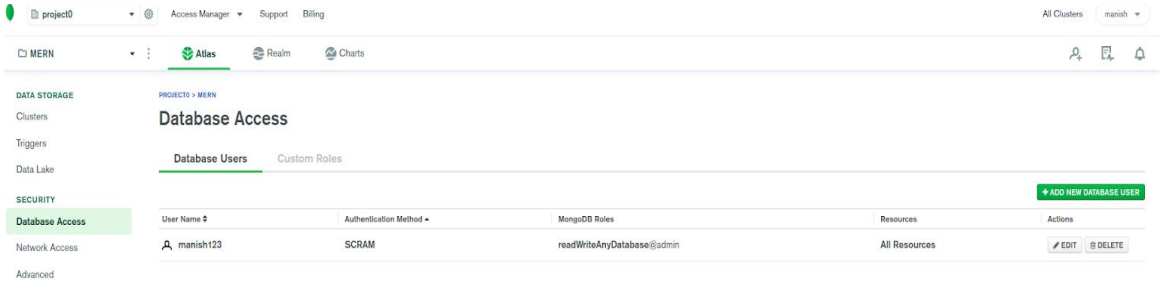


d. Create new organization named project0and then as MERN and create the cluster named DevConnector
e. https://cloud.mongodb.com/v2/5f565635c238e3340cc885b2#clusters

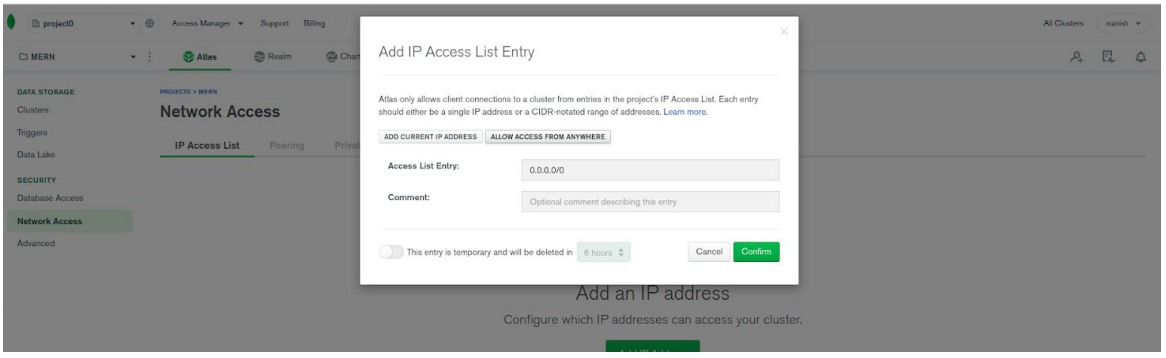
f. After the cluster is done we need to do our security goals and you will get the screen as below.



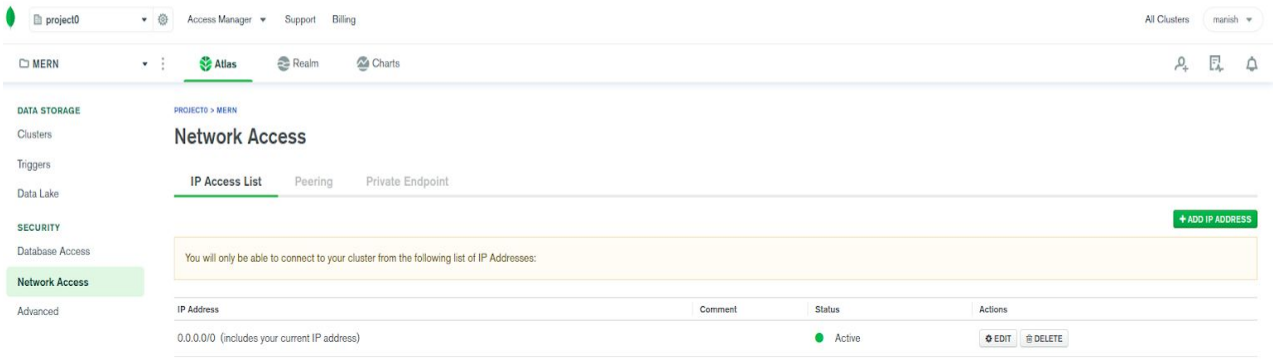
g. Create a user ex- manish123 and password - manish123



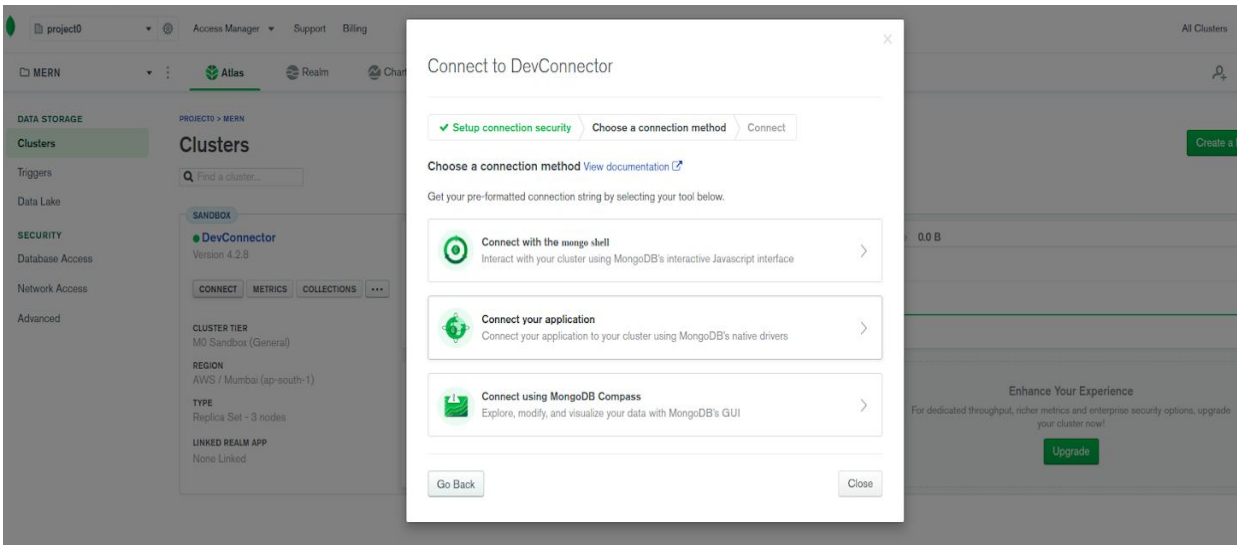
h. Then go for ip whitelist and select from anywhere and confirm



i. The the next screen will be

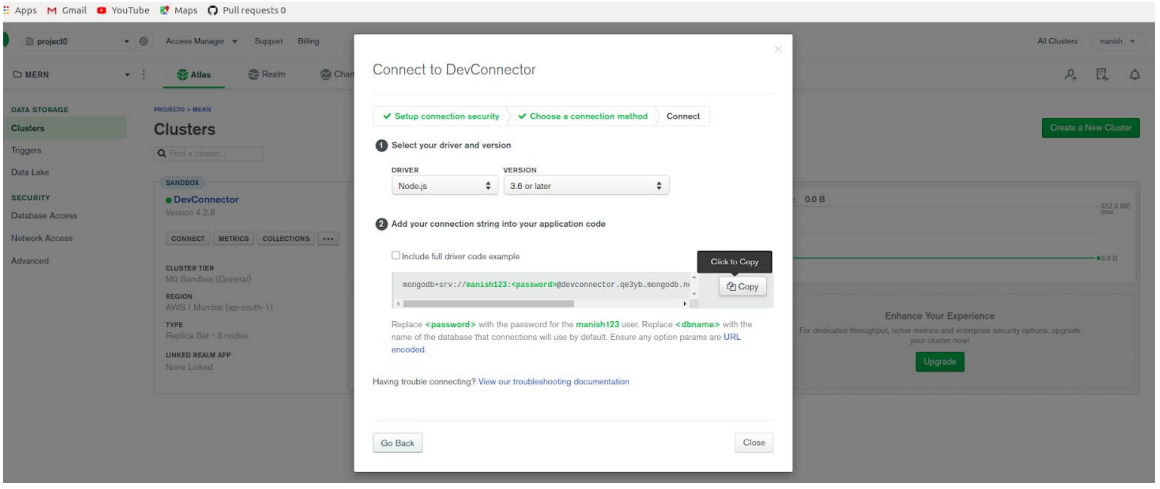


j. Then come back to cluster and select connect

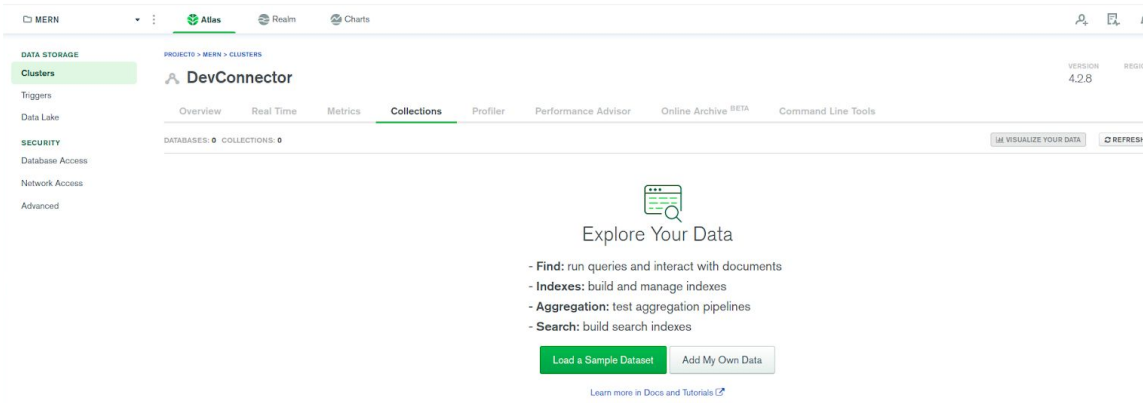


k. Then select connect your application and copy this so we can use this as plugin

l. mongodb+srv://manish123:<password>@devconnector.qe3yb.mongodb.net/<dbname>?retryWrites=true&w=majority



m. Go to collection to see the data



i. Link to access db - https://cloud.mongodb.com/v2/5f565635c238e3340cc885b2#metrics/replicaSet/5f565798a79fea3362312619/explorer

ii. We are going to use something called mongoose https://mongoosejs.com/

2. Installation of basic dependencies and express setup

a. Create a folder called DEV-CON
b. Create file .gitignore file and write node_modules/ and save
c. And in terminal do “git init” to initialize the repo
d. Go and do “npm init”
e. This will be our package.json

```
{
 "name": "dev-con",
 "version": "1.0.0",
 "description": "Social network for developer",
 "main": "server.js",
 "scripts": {
   "test": "echo \"Error: no test specified\" && exit 1"
 },
 "author": "Manish",
 "license": "MIT"
}
```

Make some change in package .json
Add under script ...start and server.

```
{
 "name": "dev-con",
 "version": "1.0.0",
 "description": "Social network for developer",
 "main": "server.js",
 "scripts": {
   "start": "node server",
   "server": "nodemon server"
 },
 "author": "Manish",
 "license": "MIT",
 "dependencies": {
   "bcryptjs": "^2.4.3",
   "config": "^3.3.1",
   "express": "^4.17.1",
   "express-validator": "^6.6.1",
   "gravatar": "^1.8.1",
   "jsonwebtoken": "^8.5.1",
   "mongoose": "^5.10.3",
   "request": "^2.88.2"
 },
 "devDependencies": {
   "concurrently": "^5.3.0",
   "nodemon": "^2.0.4"
 }
}
```

- Installing dependencies
- npm i express express-validator bcryptjs config gravatar jsonwebtoken mongoose request
- It will create package-lock.json

Install devDependencies using below command and you can see this in the package.json

- npm i -D nodemon concurrently

Lets understands every dependencies one by one

1. Bcryptjs -
    a. https://github.com/dcodeIO/bcrypt.js
    b. https://medium.com/@vuongtran/using-node-js-bcrypt-module-to-hash-password-5343a2aa2342
    c. https://medium.com/@manishsundriyal/a-quick-way-for-hashing-passwords-using-bcrypt-with-nodejs-8464f9785b67
    d. Using bcrypt.js module to hash password as my first solution when develop app in Node.js

Username	Password	Salt value	String to be hashed	Hashed value = Hashing Algo (Salt value + Password)
user1	password1	JVOLESN8T1NS	password1JVOLESN8T1NS	CB19EN3T15WQD32Z0R34
user2	password2	HCX3E9JZ2N47	password2HCX3E9JZ2N47	PTBCKO2LG8EWONHA0YR8
user3	password3	CO3MMH5NSM6	password3CO3MMH5NSM6	9LJYGGA6RF30F96HA36V

Salt Rounds: This is the cost factor that indicates the amount of time needed to calculate a single bcrypt hash. Higher the salt rounds, the more hashing rounds are done, hence the time and difficulty is increased while brute-forcing. For example, a cost factor of n means that the calculation will be done 2^n times.

Implementation

```
const bcrypt = require('bcrypt');

const saltRounds = 10;

const yourPassword = "some Random Password Here";
```

2. Config -
    a. https://www.npmjs.com/package/config
    b. Node-config organizes hierarchical configurations for your app deployments.
    c. It lets you define a set of default parameters, and extend them for different deployment environments (development, qa, staging, production, etc.).
    d. Configurations are stored in configuration files within your application, and can be overridden and extended by environment variables, command line parameters, or external sources.
    e. This gives your application a consistent configuration interface shared among a growing list of npm modules also using node-config.

3. Gravatar -

    a. https://medium.com/@robinlphood/gravatar-in-your-node-js-application-6aded410e883#:~:text=Introduction,get%20ready%20to%20use%20it.
    b. Gravatar (Global Recognized Avatar) is a world wide platform for linking a user profile to all kinds of websites using an email address.

http://www.gravatar.com/avatar/{HASH VALUE}[.jpg][?param=value&param2=value&...]		
Parameter	**Explanation**	**Values**
s or size	Number of pixels to resize the square avatar	0 - N
d or default	The default image path if no gravatar account was found	Any URL
f or forcedefault	Forces the default gravatar image	y or n
r or rating	Loads only avatars that meet the given rating	g, pg, r, x

http://www.gravatar.com/{HASH VALUE}[File extension]	
File extension	**Full name**
.xml	eXtended Markup Language
.json	JavaScript Object Notation, using JSONP is also possible
.qr	QR-Code which is really cool
.vcf	vCard
.php	PHP format

4. Request -

   https://stackabuse.com/the-node-js-request-module/#:~:text=The%20request%20module%20is%20by,it%20a%20whole%20lot%20easier.

   a. The request module is by far the most popular (non-standard) Node package for making HTTP requests. Actually, it is really just a wrapper around Node's built in http module, so you can achieve all of the same functionality on your own with `http`, but `request` just makes it a whole lot easier.

Ex-

```
const request = require('request');
 request('http://stackabuse.com', function(err, res, body)
{
 console.log(body);
  }
);
```

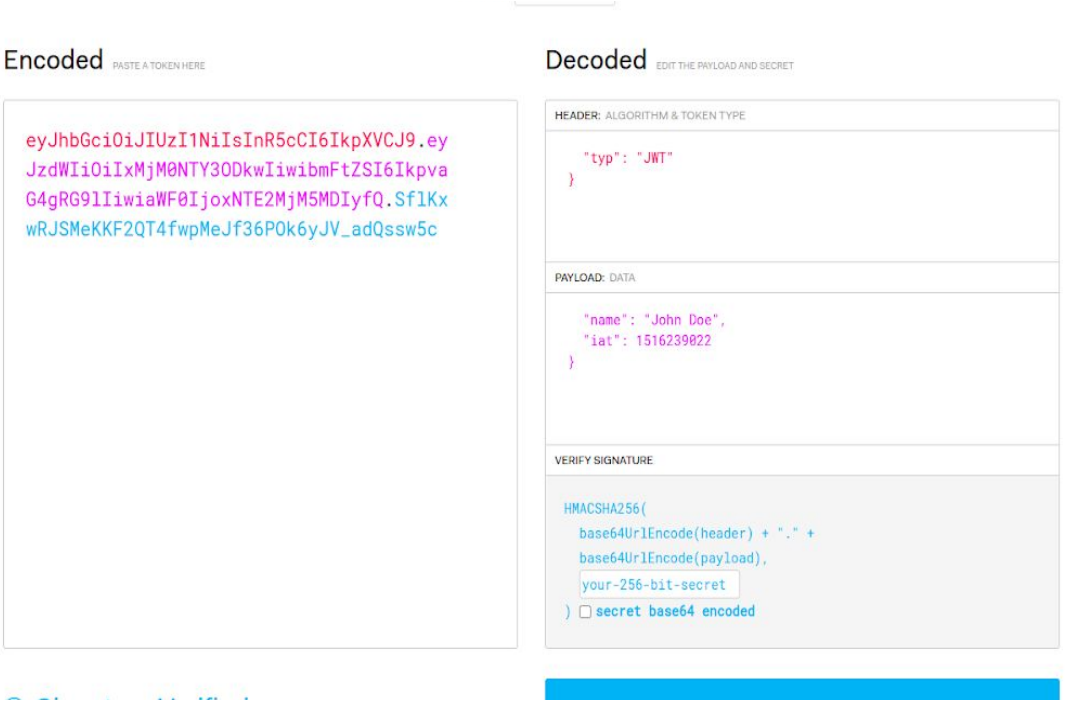
- `request.get(options, callback)`
- `request.post(options, callback)`
- `request.head(options, callback)`
- `request.delete(options, callback)`

5. Jsonwebtoken -

   https://github.com/auth0/node-jsonwebtoken

For information - https://jwt.io/

- Here the code divided in to the different colors, see the colour and their decode



- Payload is nothing but data you want to send.
- First we sign then verify , we send the jwt through the token.
-

  - An implementation of JSON Web Tokens.
  - jwt.sign(payload, secretOrPrivateKey, [options, callback])

Ex-

Synchronous Sign with default (HMAC SHA256)

```
var jwt = require('jsonwebtoken');


var token = jwt.sign({ foo: 'bar' }, 'shhhhh');
```

Synchronous Sign with RSA SHA256

```
// sign with RSA SHA256
```

```
var privateKey = fs.readFileSync('private.key');


var token = jwt.sign({ foo: 'bar' }, privateKey, { algorithm:
'RS256'});
```

Sign asynchronously

```
jwt.sign({ foo: 'bar' }, privateKey, { algorithm: 'RS256' },
function(err, token) {


  console.log(token);


});
```

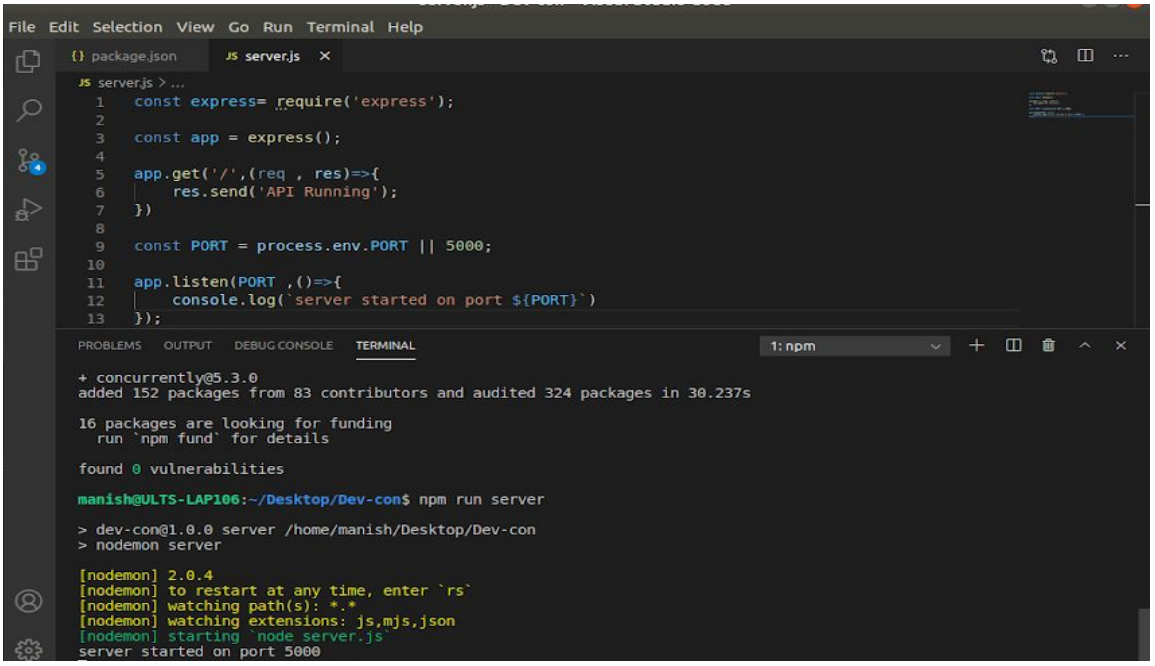
jwt.verify(token, secretOrPublicKey, [options, callback])

3. Now lets create the our entry file server.js

```
const express= require('express');

const app = express();

app.get('/',(req , res)=>{
   res.send('API Running');
})

const PORT = process.env.PORT || 5000;

app.listen(PORT ,()=>{
   console.log(`server started on port ${PORT}`)
});
```

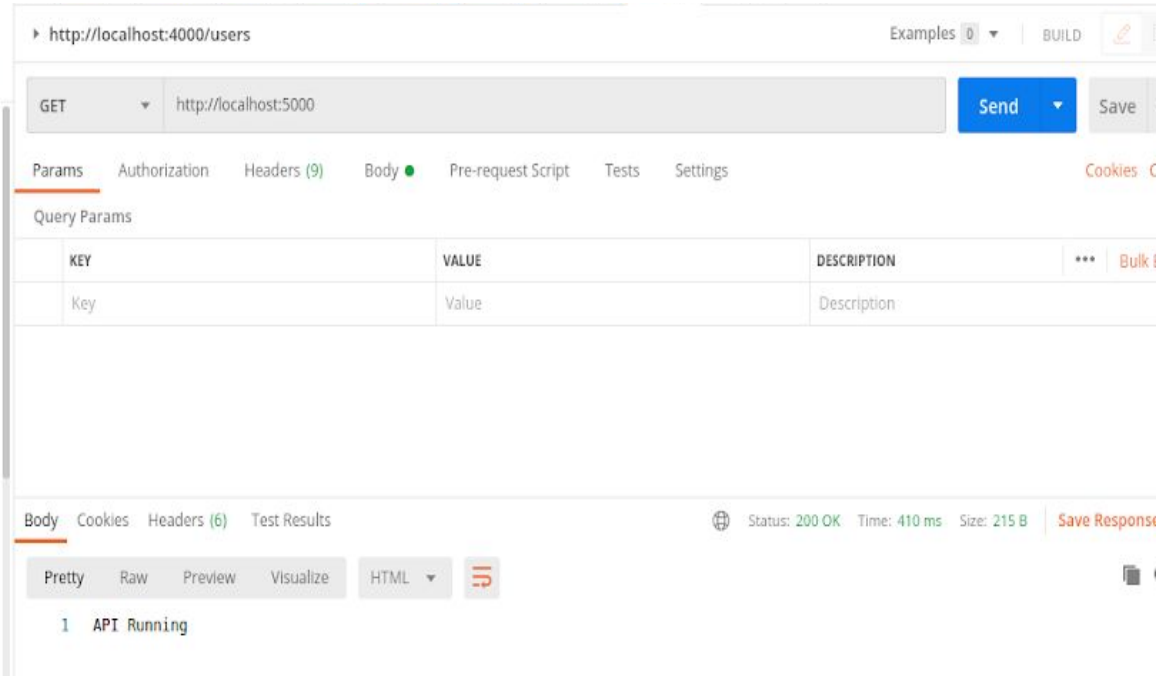
To run this “node run server”

- And you will get the screen as below



4. Go to PostMan and test your api

a. http://localhost:5000



5. Connecting mongoDb with mongoose

- Now create a folder config ,we are writing configuration and will make it global and use it.
- Inside config , create one file default.json
- Also create db.js for writing the connection

6. Route file with Express

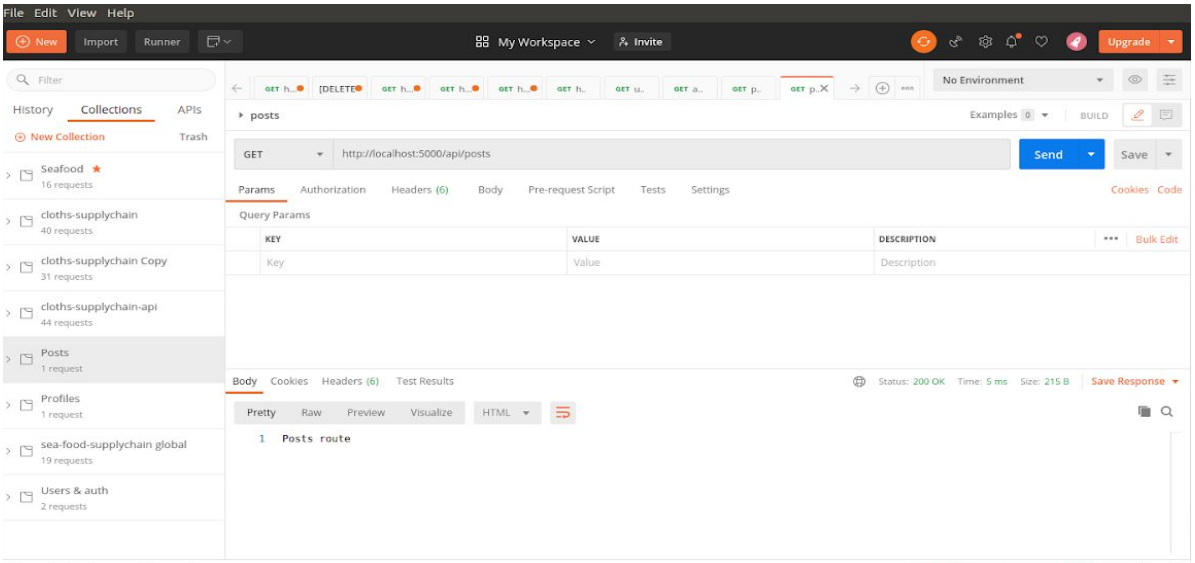
- Create folder routes/api and inside this create users.js & auth.js , profile.js , posts.js
- In some cases we need jwt token authorization in such cases called as private route
- We can write the api other places and we can access those in server by using app.use()

Ex - `app.use('/api/posts', require('./routes/api/posts')) in server.js file`

- Standard practice of writing api will be

```
const express = require('express');
const router = express.Router();

// @route   GET api/users
// @desc    Test route
// @access  Public

router.get('/',(req,res)=>{
   res.json({ msg: 'users Works' })
})

module.exports= router;
```

- Also create three collection in postman named as : Users & Auth , Profiles , Posts
- And test it



You can also make use as `res.json({ msg: 'users Works' })`
And it will the response in json as below

```
{
    "msg": "users Works"
}
```

Section-3 : User API Routes & JWT Authentication

1. Creating The User Model

- To interact with the database we need to create our model for each.
- Create a folder called models and inside this create the file User.js for modeling the user {always use the first character as uppercase while writing the model file}.
- To model this before we need to have schema
- So we can write model file as below

```
const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({
 name:{
     type : String,
     required : true
 } ,
 email :{
     type :String,
     required : true,
     unique: true
 },
 password: {
     type : String,
     required : true
 },
 //it will help you to attach your profile with the email without even
realising
 avatar :{
     type : String
 },
 date : {
     type : Date,
     default: Date.now
 }
});

//model it
// const User = mongoose.model('user',UserSchema);
// module.exports= User

//Alternative for Model

module.exports= User = mongoose.model('user',UserSchema);;
```

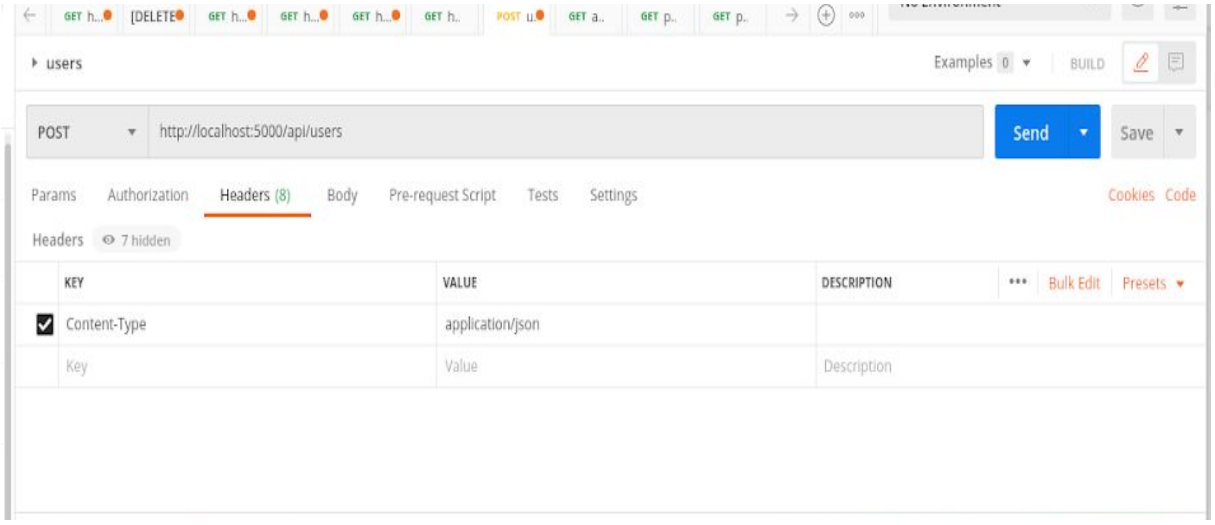
-

2. Request & Body Validation

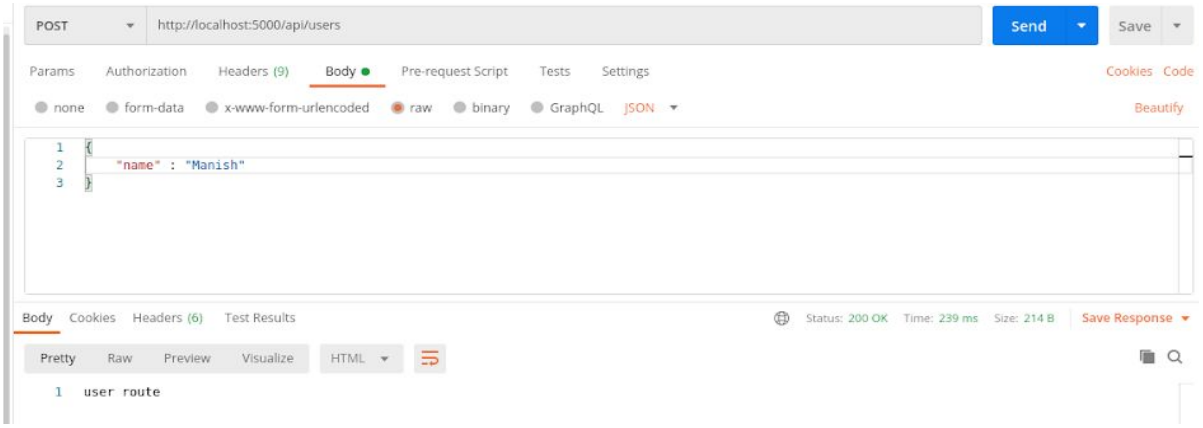
- This the api

```
const express = require('express');
const router = express.Router();

// @route    POST api/users
// @desc     Register user
// @access   Public

router.post('/',(req,res)=>{
   console.log(req.body);
   res.send('user route');
})

module.exports= router;
```

Then do some setup in postman



In the body



- Let's make use of express-validator so for details go to https://express-validator.github.io/docs/

We can make use as

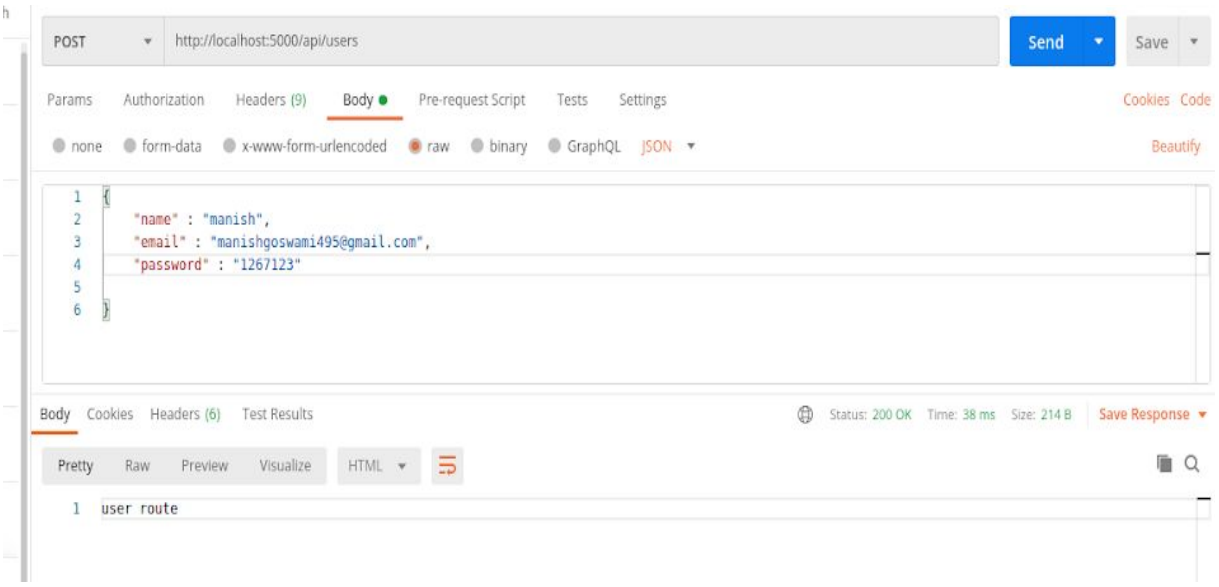
```
const { check, validationResult } = require('express-validator/check');
```

Note : And always put a validator in between route and callback function in the api and also [ ] if there is more field then you can make arrays.

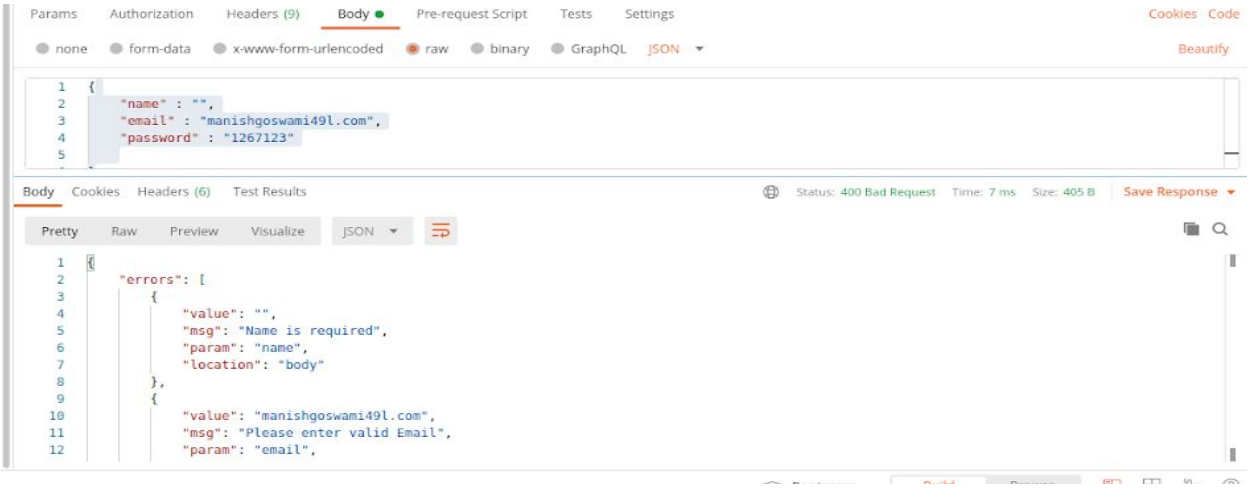
Example api with validator -

```
router.post('/',[
   check('name' ,'Name is required').not().isEmpty(),
   check('email','Please enter valid Email').isEmail(),
   check('password' ,'please Enter password of minimum 6 letters of
characters').isLength({min: 6})
],
   (req,res)=>{
       const errors = validationResult(req);
       if (!errors.isEmpty())
       {
         return res.status(400).json({ errors: errors.array() });
       }
       res.send('user route');

})
```

And output will be



Output when we give wrong value then validator will throw the error as below



3. User Registration

- In this section we will be doing the following things.

```
// See if user exists

// Get users gravatar

// Encrypt password

// Return JsonWebtoken
```

- So call your model from models/User in the file api/user.js
- For using gravatar make sure you will import gravatar package

- For using bcrypt make sure use will import the bcryptjs

```
        const salt = await bcrypt.genSalt(10);

       // In this , its taking as a plan password and making it hash
        // Anything returning promise make sure you use await

        user.password = await bcrypt.hash(password ,salt);
        await user.save();
```

4. Implementing JWT

- General format -

```
jwt.sign({ foo: 'bar' }, privateKey, { algorithm: 'RS256' },
function(err, token) {
  console.log(token);
});
```

- We need to write `"jwtSecret" : "mysecrettoken"`
- In config/default.json and import in users.js

    a. 1st create payload as below

```
const payload = {
                user: {
                    id : user.id
                }
            }

            jwt.sign(
                payload,
                config.get('jwtSecret'),
                { expiresIn : 36000},
                (err , token) =>{
                    if(err) throw err;
```

```
                    res.json({ token })
                }
            );
```

b. Alternative

```
const payload = {
                user: {
                    id: user.id
                }
            }
            const privatekey = config.get('jwtSecret');
            jwt.sign(
                payload,
                privatekey,
                { expiresIn: 36000 },
                (err, token) => {
                    if (err) throw err;
                    res.json({ token })
                }
            );
```

c. `{ "token":`

`"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7ImlkIjoiNWY1N2JkNTNk`
`NTZhOWMyZDYxYzBjYzA0In0sImlhdCI6MTU5OTU4NTYxOSwiZXhwIjoxNTk5NjIxNjE5fQ`
`.1uo-C8YItYKpKtdJWVAI4sMrSawex4H3S6T9EaLWOdQ"`

}

d. You can recheck you token https://jwt.io/

5. Custom Auth Middleware & JWT Verify

- https://www.sohamkamani.com/blog/javascript/2019-03-29-node-jwt-authentication/
- To verify jwt token , we need to send back to authenticate it
- So let create middleware in root
- Inside this create file auth.js

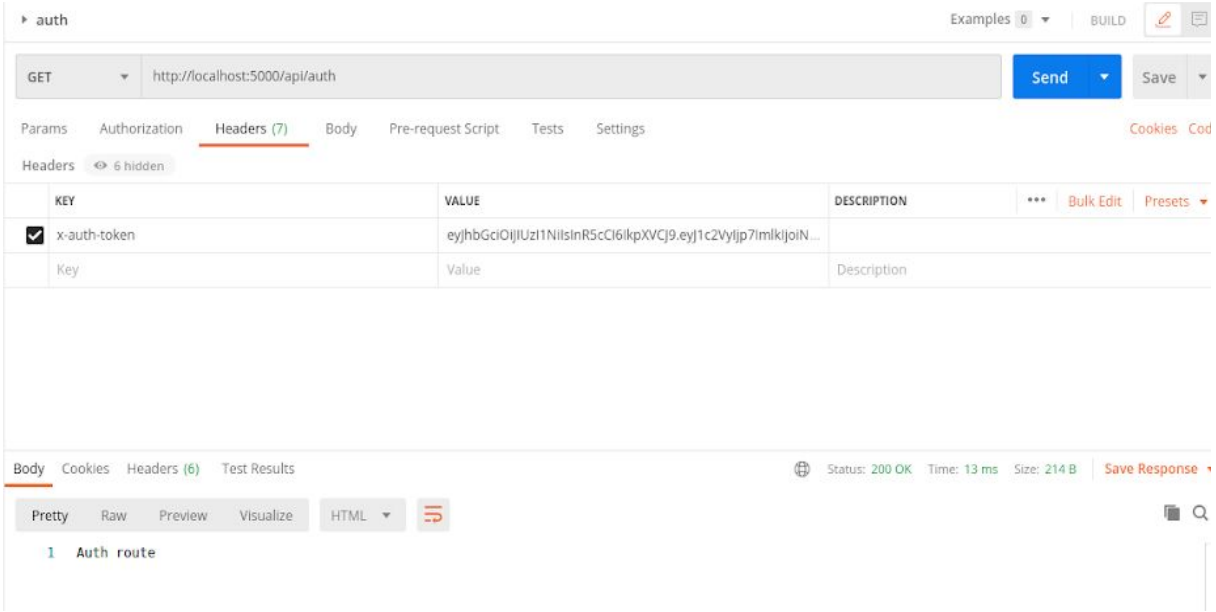
- We take the header and payload from the upcoming jsow token and also using jwtSecret(only application has) will generate a one more token and matches the secret . if it's found same then consider as valid
- Code to verify jwt is as below

```
// @verify Token
try {
    const secret = config.get('jwtSecret');
    const decoded = jwt.verify(token, secret);

    // @matching
    req.user = decoded.user;
    next();
} catch (err) {
    res.status(401).json({ msg: 'Token is not Valid' })
}
```

- Now let's go to the routes/api/auth for verification
- Whenever you want to add some middleware you add as a second parameters in the api

```
router.get('/',auth,(req,res))
```

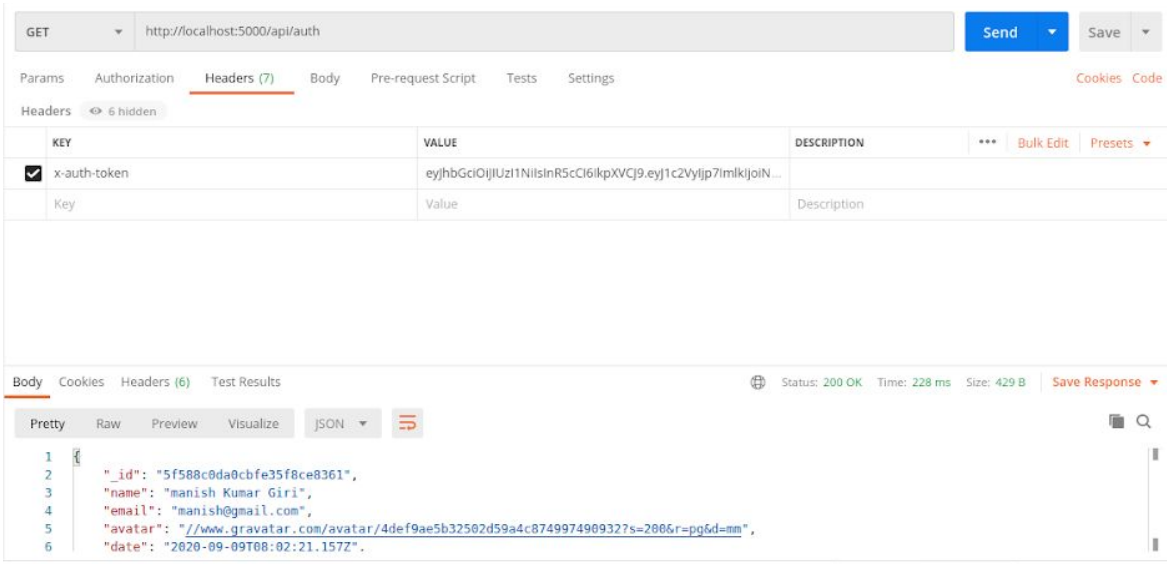
- Ac
- To verify the token you can give as below in the postman



If you don't want something should be displayed from the database to the ui you can ignore those by using select

Ex- i do not want to display this name and password so i used below method

```
const user = await
User.findById(req.user.id).select('-password').select('-name')
```

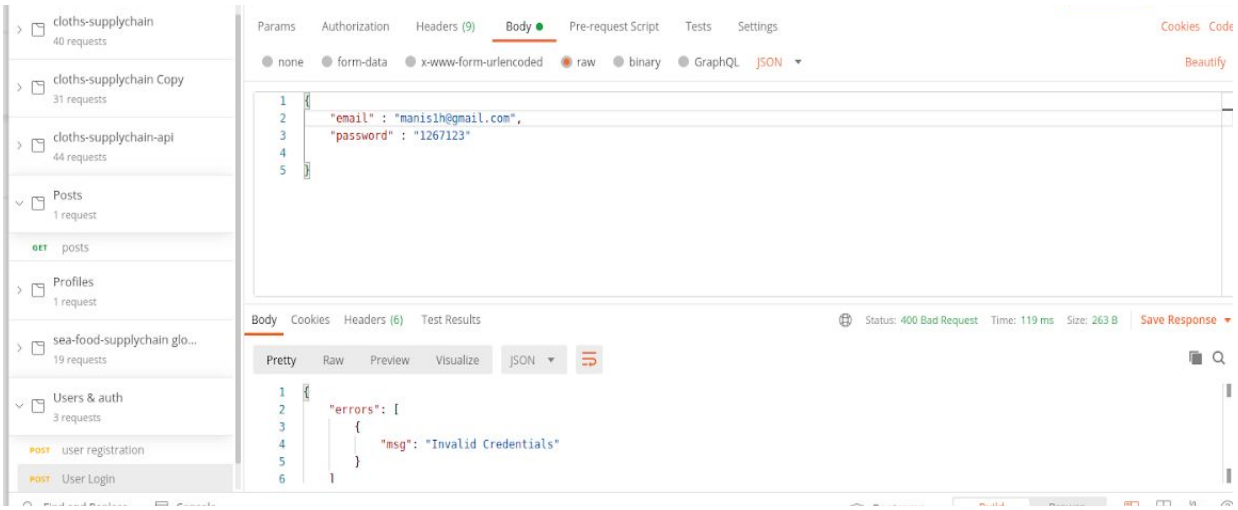


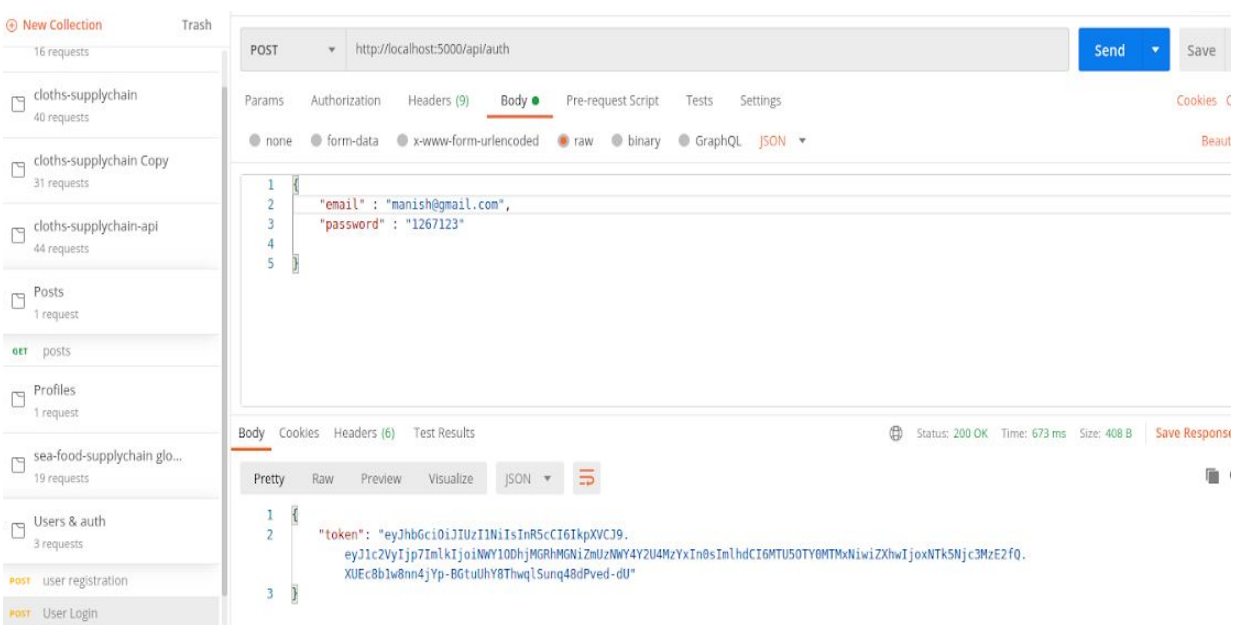
6. User Authentication Login Route

- **To check a password:**

```
- // Load hash from your password DB.
- bcrypt.compare(myPlaintextPassword, hash, function(err,
  result) {
-     // result == true
- });
- bcrypt.compare(someOtherPlaintextPassword, hash,
  function(err, result) {
-     // result == false
- });
-
```

When you give the correct api


cloths-supplychain
40 requests
cloths-supplychain Copy
31 requests
cloths-supplychain-api
44 requests
Posts
1 request
GET posts
Profiles
1 request
sea-food-supplychain glo...
19 requests
Users & auth
3 requests
POST user registration
POST User Login
Params Authorization Headers (9) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
{
"email" : "manis1h@gmail.com",
"password" : "1267123"
}
Body Cookies Headers (6) Test Results
Status: 400 Bad Request Time: 119 ms Size: 263 B
Save Response
Pretty Raw Preview Visualize JSON
{
"errors": [
{
"msg": "Invalid Credentials"
}
]


New Collection
Trash
16 requests
cloths-supplychain
40 requests
cloths-supplychain Copy
31 requests
cloths-supplychain-api
44 requests
Posts
1 request
GET posts
Profiles
1 request
sea-food-supplychain glo...
19 requests
Users & auth
3 requests
POST user registration
POST User Login
POST http://localhost:5000/api/auth
Send
Save
Params Authorization Headers (9) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
{
"email" : "manish@gmail.com",
"password" : "1267123"
}
Body Cookies Headers (6) Test Results
Status: 200 OK Time: 673 ms Size: 408 B
Pretty Raw Preview Visualize JSON
{
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ1c2VyIjp7ImlkIjoiNWY1ODhjMGRhMGNiZmUzNWY4Y2U4MzYxIn0sImlhdCI6MTU5OTY0MTMxNiwiZXhwIjoxNTk5Njc3MzE2fQ.
XUEc8b1w8nn4jYp-BGtuUhY8ThwqlSunq48dPved-dU"
}

Section 4 : Profile API Routes

1. Creating The Profile Model

- Create model for profile so lets make a file under modes/Profile.js

```
const mongoose = require('mongoose');

const ProfileSchema = new mongoose.Schema({

   // @we are linking each profile with user
   // @here ObjectId means _id from mongodb

   user: {
       type: mongoose.Schema.Types.ObjectId,
       ref: 'user'
   },
   company: {
       type: String
   },
   website: {
       type: String
   },
   location: {
       type: String
   },
   status: {
       type: String,
       required: true,
   },
   skills: {
       type: [String],
       required: true
   },
   bio: {
       type: String,
   },
   githubusername: {
       type: String
```

```
  },

  // @Adding experience details as Array of Objects

  experience: [{

      title: {
          type: String,
          required: true
      },
      company: {
          type: String,
          required: true
      },
      location: {
          type: String,
      },
      from: {
          type: String,
          required: true
      },
      to: {
          type: Date

      },
      current: {
          type: Boolean,
          default: false

      },
      description: {
          type: String,

      }

  }
  ],

  // @Adding educational details as Array of Objects
```

```
education:
    [
        {

            school: {
                type: String,
                required: true
            },
            degree: {
                type: String,
                required: true
            },
            fieldofstudy: {
                type: String,
            },
            from: {
                type: String,
                required: true
            },
            to: {
                type: Date
            },
            current: {
                type: Boolean,
                default: false
            },
            description: {
                type: String,

            }

        }
    ],

// @adding social media detils its object of objects
social: {

    youtube: {
        type: String
```

```
        },
        twitter: {
            type: String

        },
        facebook: {
            type: String,
        },
        linkedin: {
            type: String

        },
        instagram: {
            type: String,

        }

    },

    date: {
        type: Date,
        default: Date.now
    }
});


//Alternative for Model

module.exports = Profile = mongoose.model('profile', ProfileSchema);;
```

2. Get Current User Profile

- Go to routes/api/profile.js
- If we want to display something from another model to our model then you can use the method called “populate(‘databasename’ , ‘[things you want to display]’).

Ex-

```
       const profile = await Profile.findOne({ user :
req.user.id}).populate('user',['name','avatar']);
```

Api -

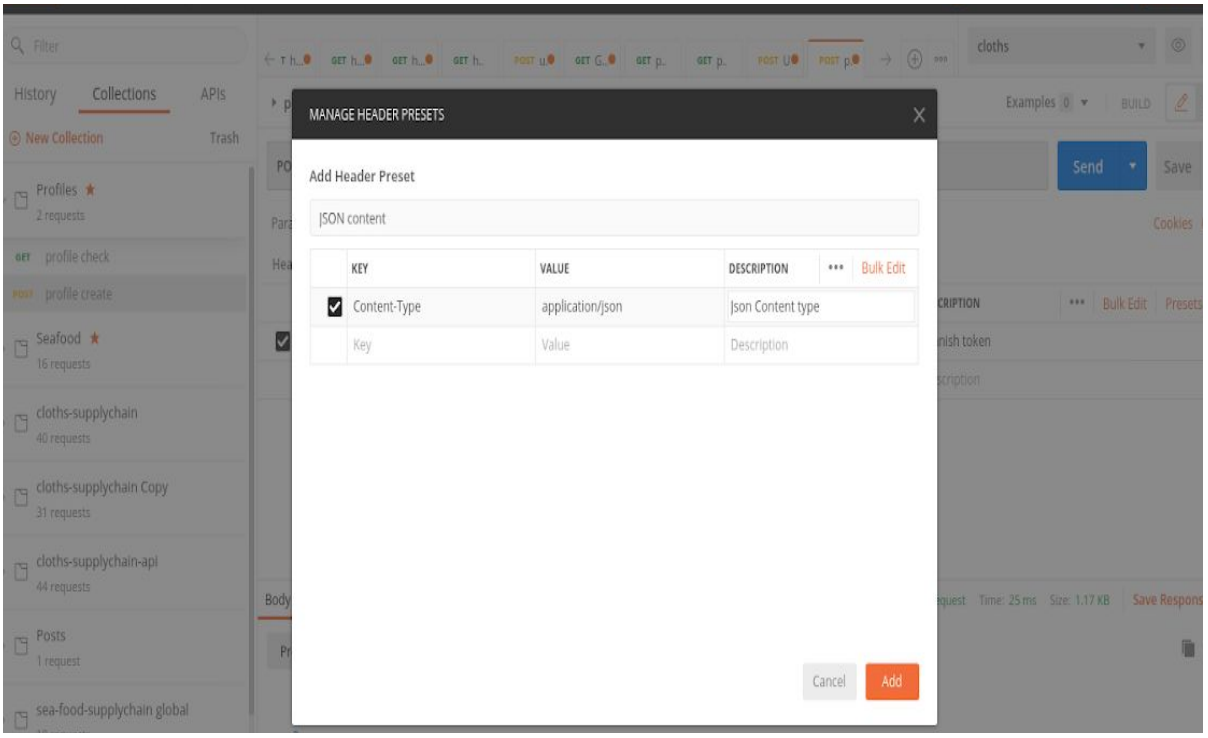
```
router.get('/me', auth, async (req, res) => {

   try {

       // @indirectly we are referencing the user model and populating
fields from there

       const profile = await Profile.findOne({ user :
req.user.id}).populate('user',['name','avatar']);
        // @check whether user exists or not
       if(!profile) {
           return res.status(400).json({msg : 'There is no profile for
the  user'});
       }

       res.json(profile);

   } catch (err) {
       console.error(err.message);
       res.status(500).send('server error');
   }

});
```

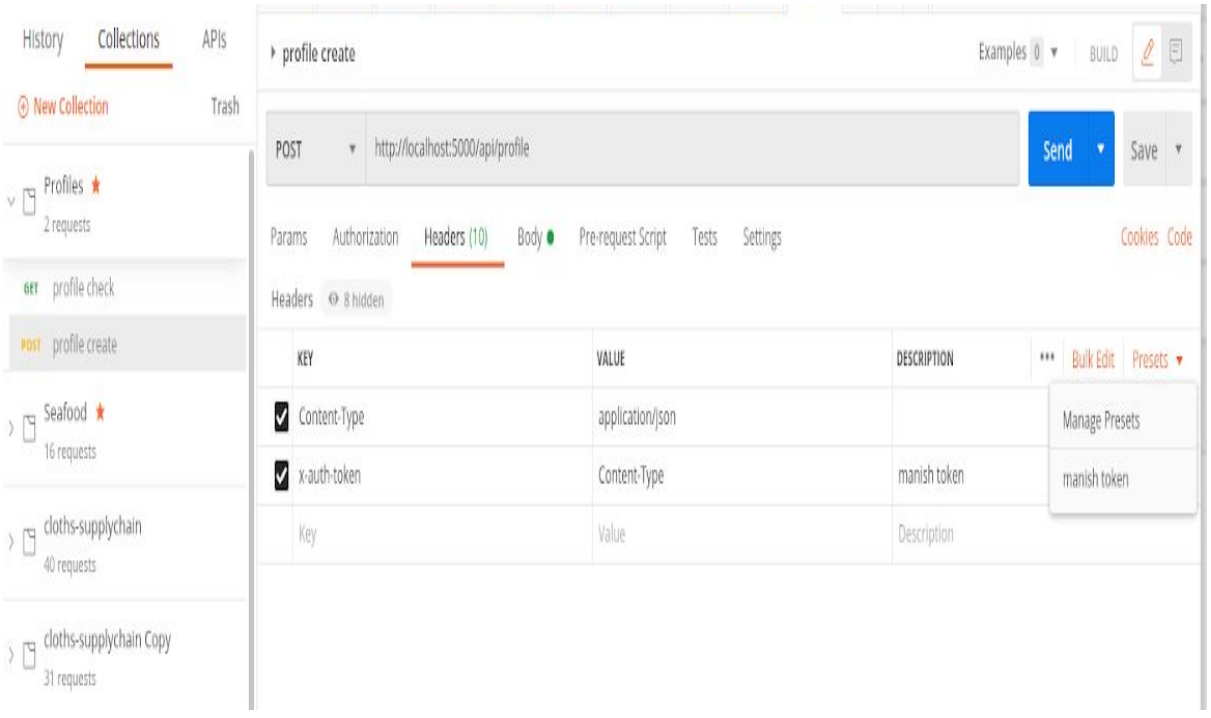
3. Create & Update Profile Routes

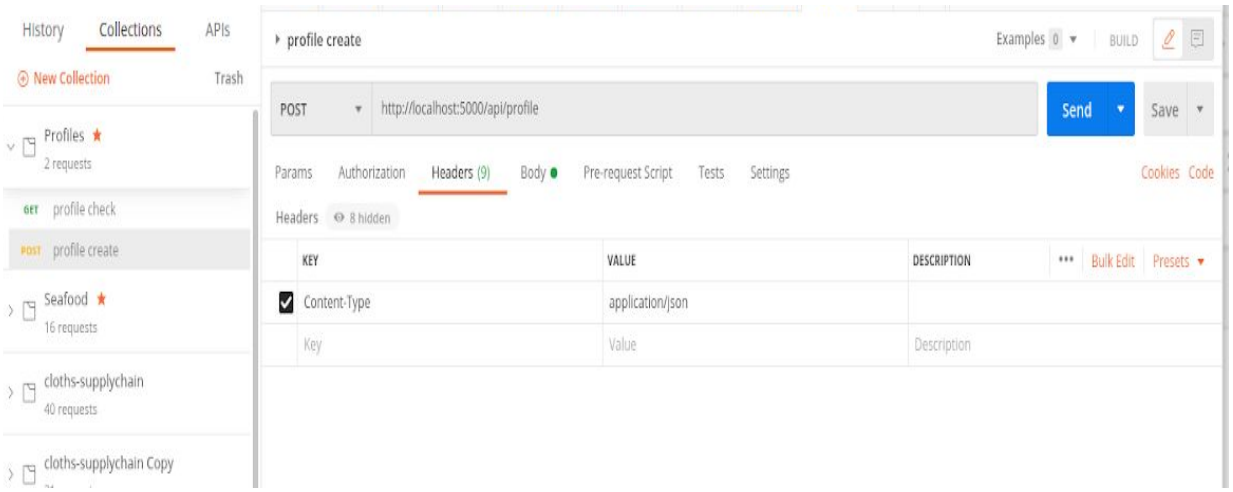
- In Profile there are only 2 fields required i.e- skills and status so those only should be written with the validator.

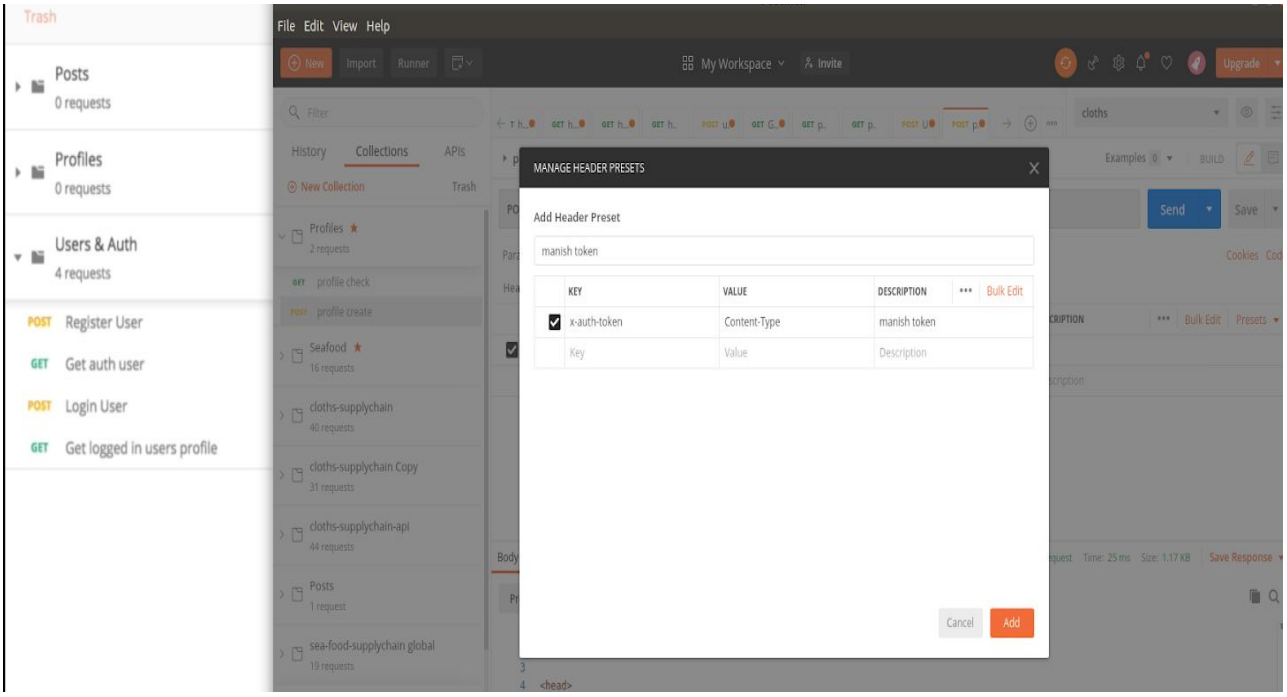
We can do shortcut in the postman

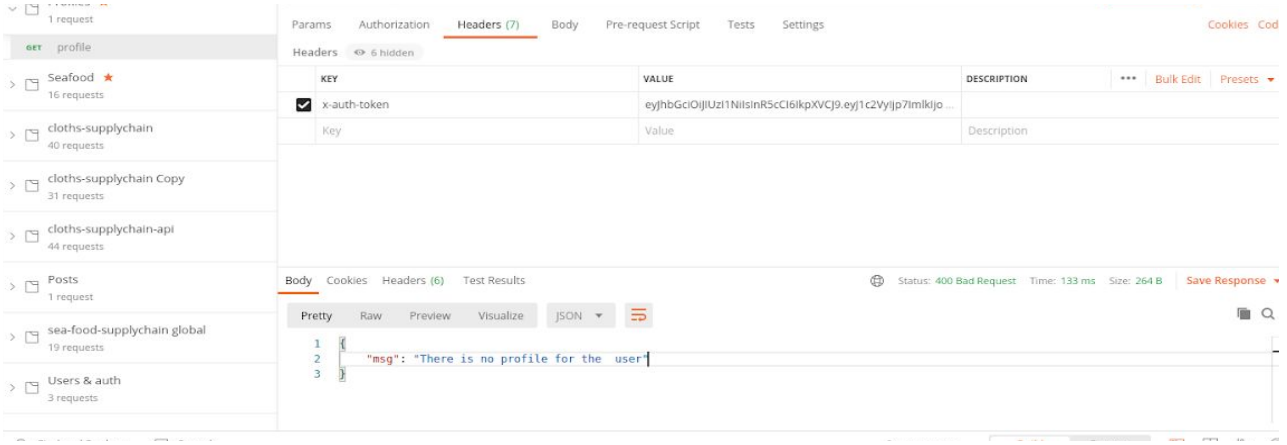
- In the fields so add
  X-auth-token - token



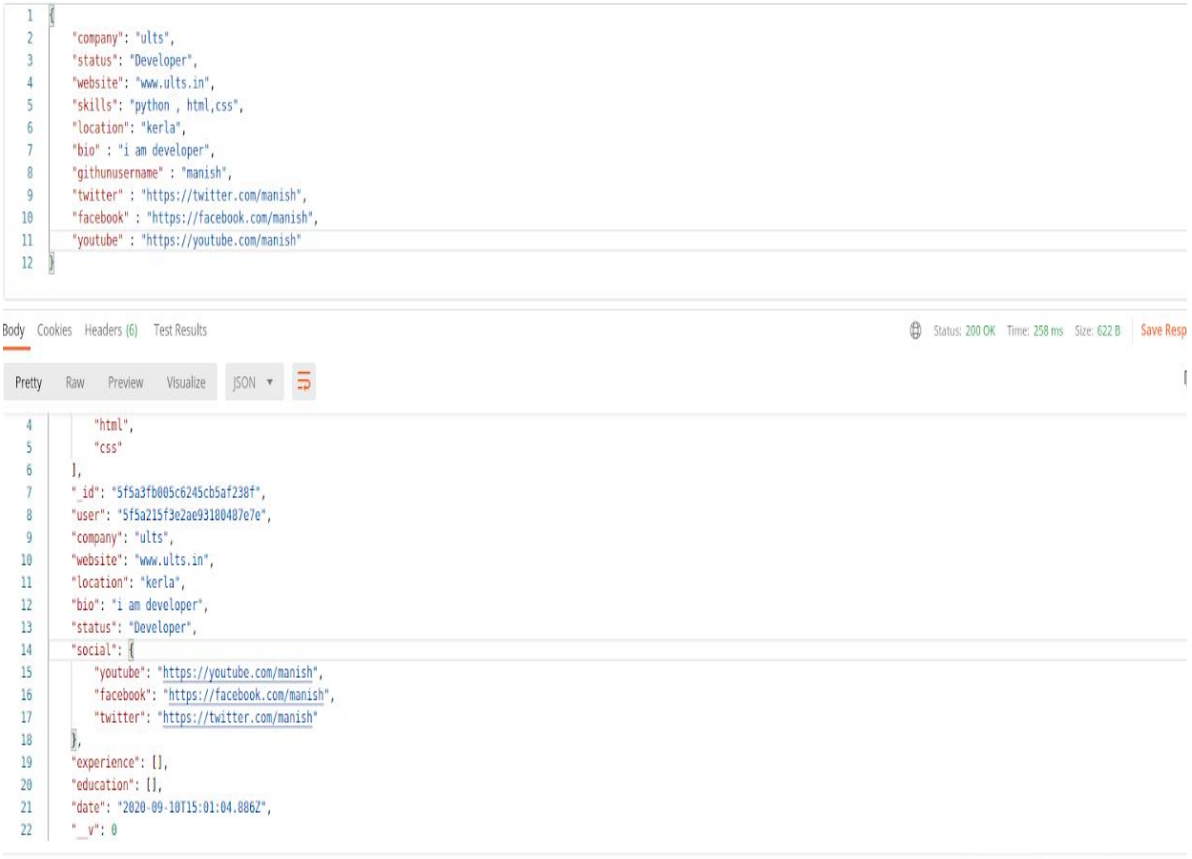



History
Collections
APIs
New Collection
Trash
Profiles
2 requests
GET profile check
POST profile create
Seafood
16 requests
cloths-supplychain
40 requests
cloths-supplychain Copy
profile create
Examples 0
BUILD
POST
http://localhost:5000/api/profile
Send
Save
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Headers
8 hidden
KEY
VALUE
DESCRIPTION
Bulk Edit
Presets
Content-Type
application/json
Key
Value
Description

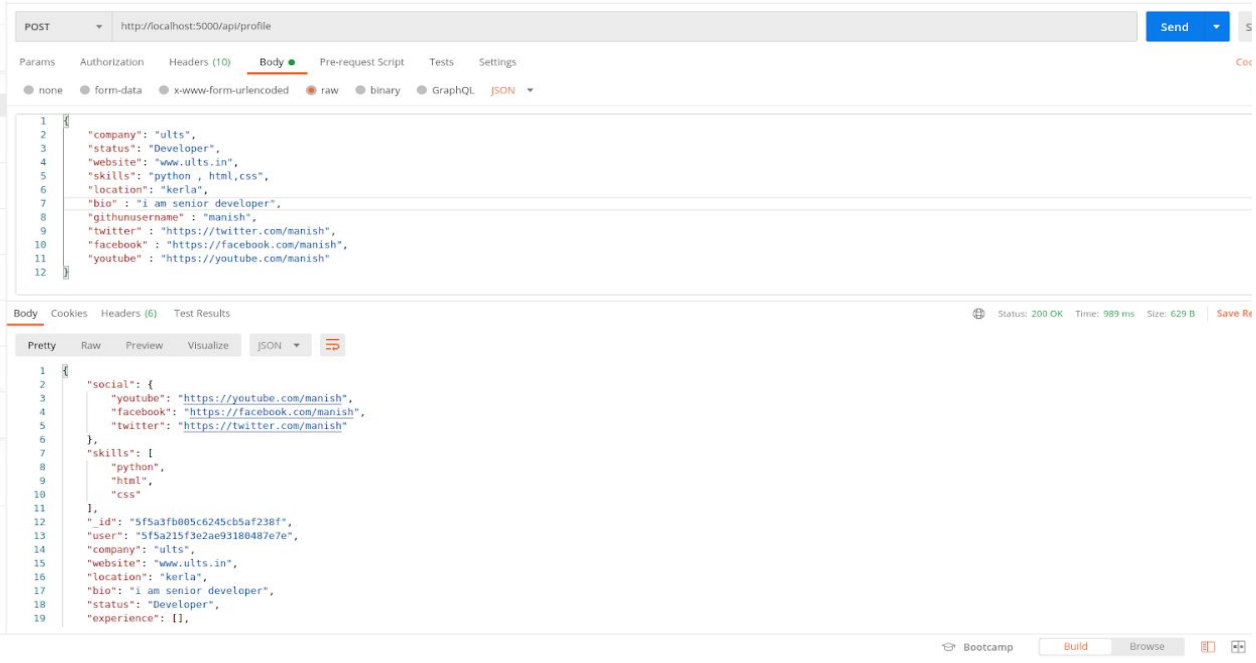

Trash
Posts
0 requests
Profiles
0 requests
Users & Auth
4 requests
POST Register User
GET Get auth user
POST Login User
GET Get logged in users profile
File Edit View Help
New
Import
Runner
My Workspace
Invite
Upgrade
MANAGE HEADER PRESETS
Add Header Preset
manish token
KEY
VALUE
DESCRIPTION
Bulk Edit
x-auth-token
Content-Type
manish token
Key
Value
Description
Cancel
Add


1 request
GET profile
Seafood
16 requests
cloths-supplychain
40 requests
cloths-supplychain Copy
31 requests
cloths-supplychain-api
44 requests
Posts
1 request
sea-food-supplychain global
19 requests
Users & auth
3 requests
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Headers
6 hidden
KEY
VALUE
DESCRIPTION
Bulk Edit
Presets
x-auth-token
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7Imlkjo ...
Key
Value
Description
Body
Cookies
Headers (6)
Test Results
Status: 400 Bad Request
Time: 133 ms
Size: 264 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"msg": "There is no profile for the user"
}

So the output for the create profile



Update and check while making the change in the bio



=> trim( ) : trim is used to remove the whitespace

Ex -
```
var str = "    Hello World!";
var trimmedStr = str.trim();
 console.log(trimmedStr);
```

Output : "Hello World!"

=> split ( ) : split is used whenever we need to split the string on the specific parameter passed in the function

Ex- method - `skills = skills.split(',')`

input - `"skills": "python , html,css",`

Output -
```
"skills":
        [
       "python",
        "html",
        "css"
              ],
```

=> map() :

- The map() method creates a new array with the results of calling a function for every array element.
- The map() method calls the provided function once for each element in an array, in order.
- Note: map() does not execute the function for array elements without values.
- Note: this method does not change the original array.

  Ex- skills = skills.split(',').map(skill => skill.trim( ));

  Output -
```
"skills":
        [
       "python",
        "html",
        "css"
              ],
```

ERROR- if you are getting his kind of warning then

Ex-

DeprecationWarning: Mongoose: `findOneAndUpdate()` and `findOneAndDelete()` without the `useFindAndModify` option set to false are deprecated.

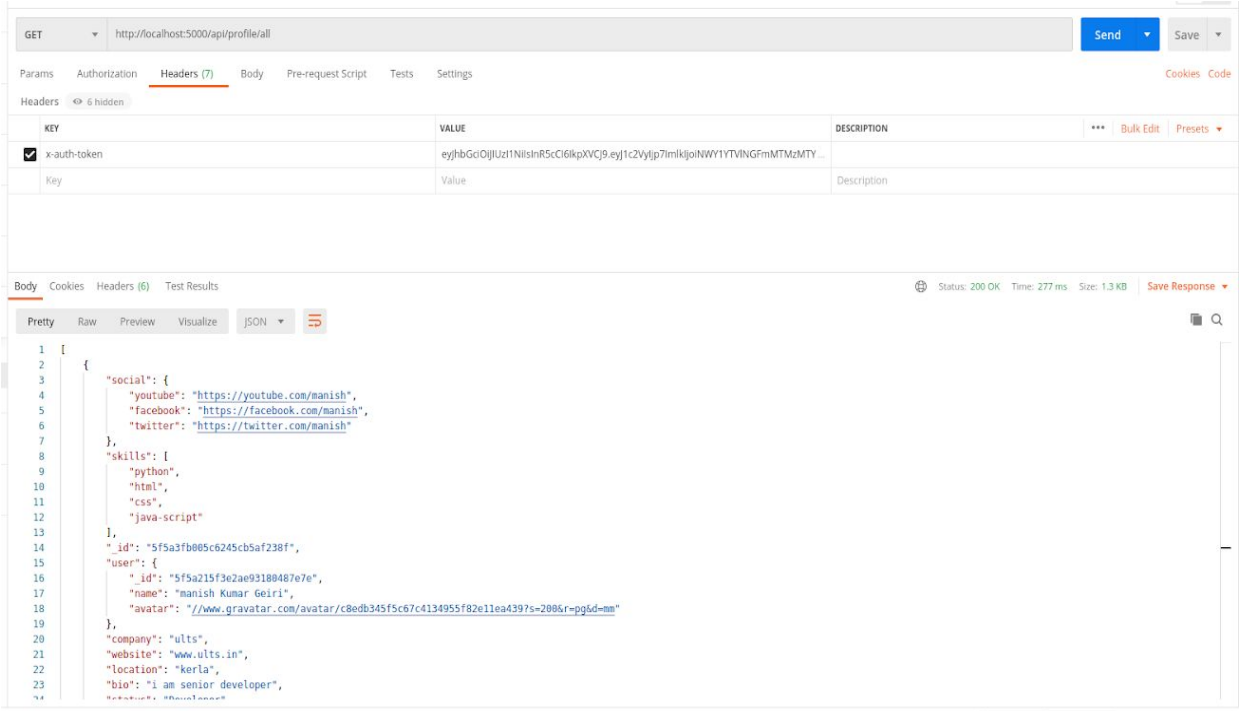
Sol- See: https://mongoosejs.com/docs/deprecations.html#findandmodify

And go to config/db.js

- Add `useFindAndModify:false`
- In mongoose.connect

4. Get All Profiles & Profile By User ID

- Trycatch+enter -its shortcut for the writing try and catch block
- Create one more api
- To get all the profiles



Here is two kind of id one is profile id and other is user id

Api for getting detail by using profile id

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

    try {

        // @here i am using profiles as variable to get all the profiles
        const profiles = await Profile.find().populate('user', ['name',
'avatar']);
        res.json(profiles);

    } catch (err) {

        console.error(err.message);
        res.status(500).send('server error');

    }
})
```

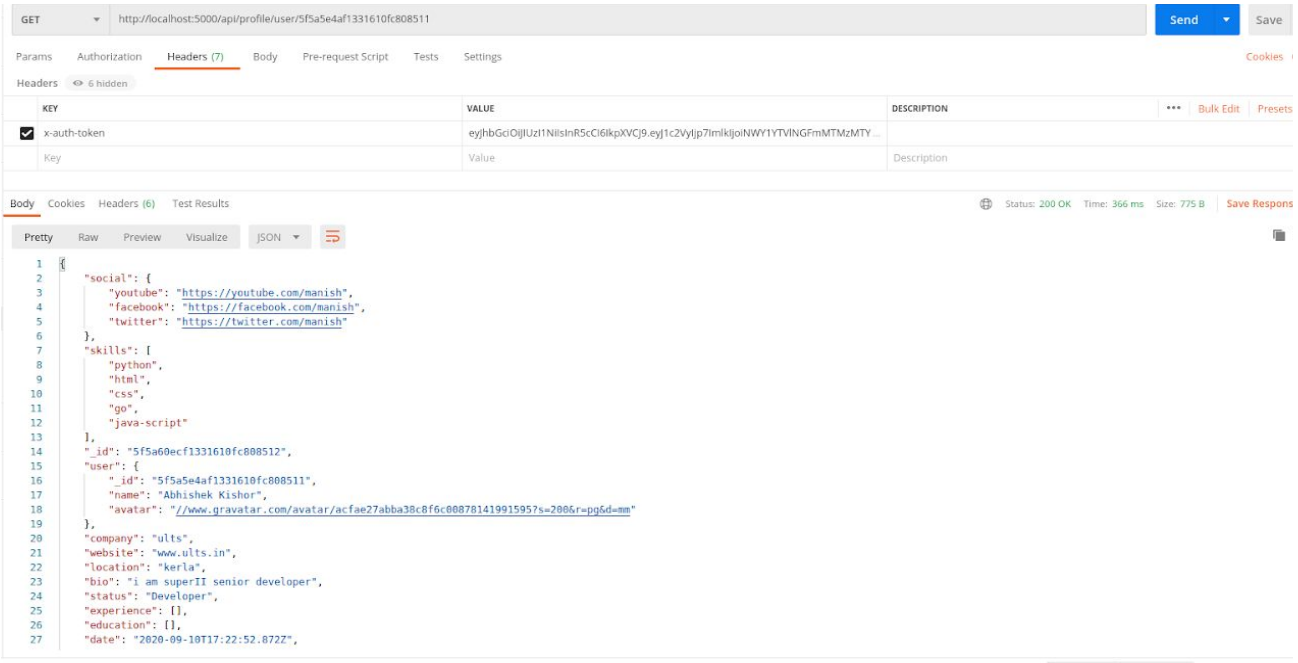

```
    "_id": "5f5a3fb805c6245cb5af238f",
    "user": {
        "_id": "5f5a215f3e2ae93188487e7e",
        "name": "manish Kumar Geiri",
        "avatar": "//www.gravatar.com/avatar/c8edb345f5c67c4134955f82e11ea439?s=200&r=pg&d=mm"
    },
    "company": "ults",
    "website": "www.ults.in",
    "location": "kerla",
    "bio": "i am senior developer",
```


Now let's write the api to get the data by user id not by profile id

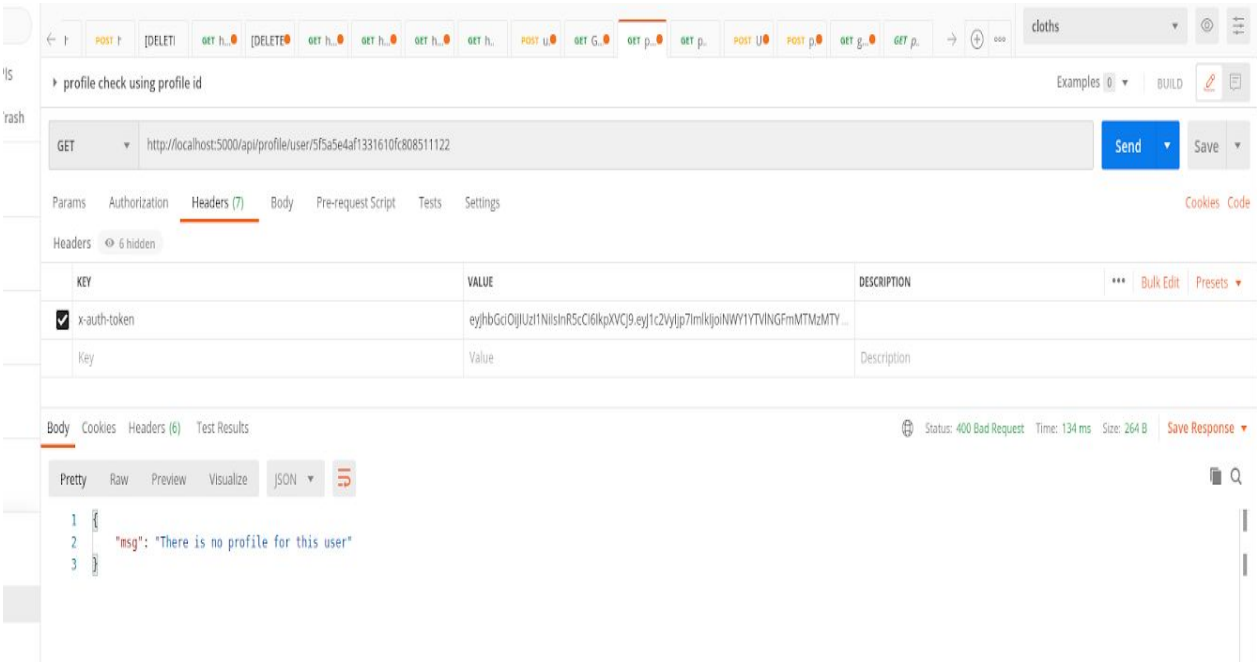
```
router.get('/user/:user_id', async (req, res) => {

    try {

        // @here i am using profile as variable to get specific user
using user_id
        const profile = await Profile.findOne({user:
req.params.user_id}).populate('user', ['name', 'avatar']);



        // @Check whether profile exists or not
```

```
        if (!profile) {
            return res.status(400).json({msg : 'There is no profile for this user'});
        }
        res.json(profile);

    } catch (err) {

        console.error(err.message);
// @what if we are giving wrong format of objectid ,then also it should give the msg like "profile not exists instead of server error"
        if(err.kind == 'ObjectId'){
            return res.status(400).json({msg : 'There is no profile for this user'});
        }

        res.status(500).send('server error');

    }
})
```

Output



What if you are giving wrong format of objectid then output will be as below



5. Delete Profile & User

- To make it private api better use auth

```
// @route    DELETE api/profile
// @desc     Delete profile , user & posts
// @access   Private so use Auth

router.delete('/',auth, async (req, res) => {

   try {
       // @todo - remove users & posts


       // @Remove profile

       await Profile.findOneAndRemove({ user: req.user.id});


       // @Remove user so use actual _id from db
       await User.findOneAndRemove({ _id: req.user.id});
```

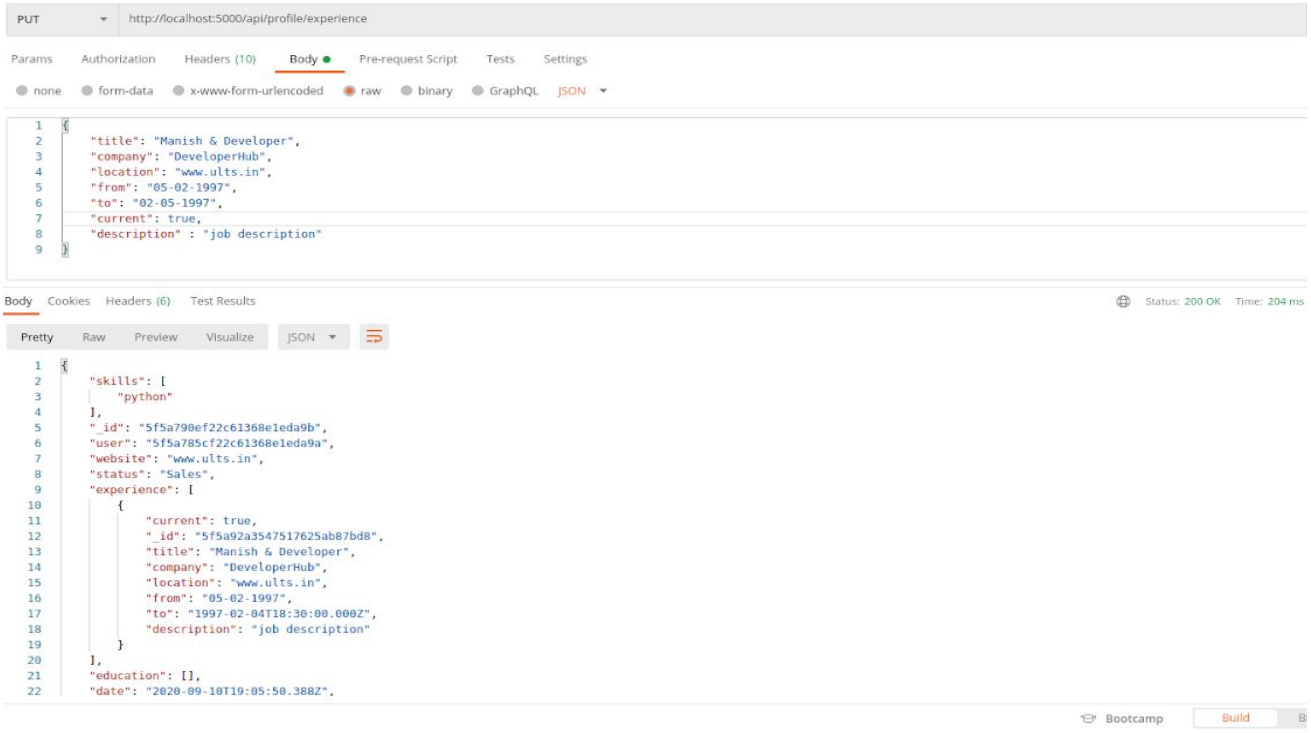
```
        res.json({msg : 'User Deleted'});

    } catch (err) {

        console.error(err.message);
        res.status(500).send('server error');

    }
})
```

6. Add Profile Experience

```
// @route    PUT api/profile/experience
// @desc     Add profile experience
// @access   Private so use Auth
// @this is Array of object
router.put('/experience', [auth,

    check('title', 'Title  is required').not().isEmpty(),
    check('company', 'company is required').not().isEmpty(),
    check('from', 'From date is required').not().isEmpty()

],
    async(req, res) => {
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
            return res.status(400).json({ errors: errors.array() });
        }
        const {
            title ,
            company,
            location,
            from,
```

```
            to,
            current,
            description
        } = req.body;

        const newExp ={
            title ,
            company,
            location,
            from,
            to,
            current,
            description
        }


        try {
            const profile = await Profile.findOne({ user: req.user.id
});
            // @this is arrray of object so only using unshift and it
will add the element at the start of the index
            profile.experience.unshift(newExp)
           await profile.save();
           res.json(profile);
        }
         catch (err) {
            console.error(err.message);
            res.status(500).send({ msg: "server Error" })
        }
    })
```

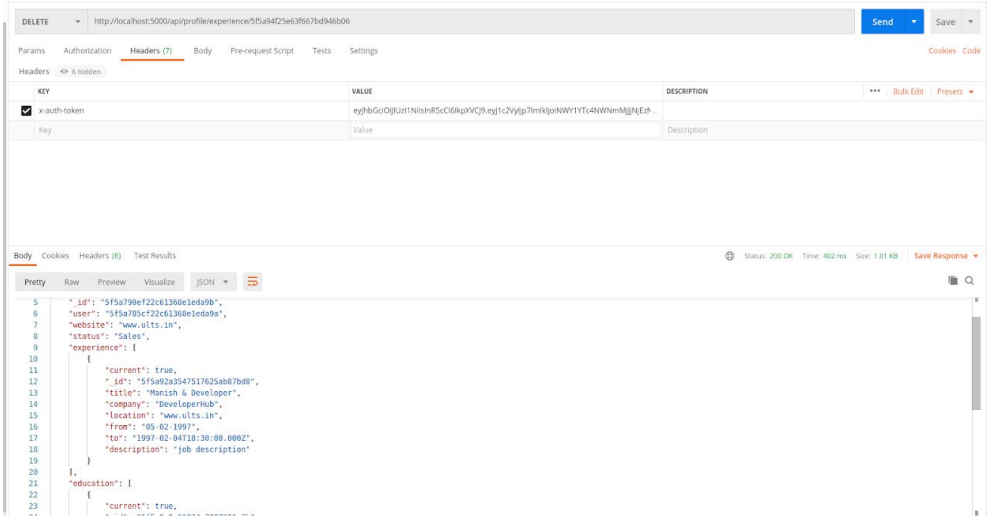
Output


PUT
http://localhost:5000/api/profile/experience
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"title": "Manish & Developer",
"company": "DeveloperHub",
"location": "www.ults.in",
"from": "05-02-1997",
"to": "02-05-1997",
"current": true,
"description" : "job description"
}
Body
Cookies
Headers (6)
Test Results
Status: 200 OK
Time: 204 ms
Pretty
Raw
Preview
Visualize
JSON
{
"skills": [
"python"
],
"_id": "5f5a790ef22c61368e1eda9b",
"user": "5f5a785cf22c61368e1eda9a",
"website": "www.ults.in",
"status": "Sales",
"experience": [
{
"current": true,
"_id": "5f5a92a3547517625ab87bd8",
"title": "Manish & Developer",
"company": "DeveloperHub",
"location": "www.ults.in",
"from": "05-02-1997",
"to": "1997-02-04T18:30:00.000Z",
"description": "job description"
}
],
"education": [],
"date": "2020-09-10T19:05:50.388Z",
Bootcamp
Build

7. Delete profile Experience

```
// @route    DELETE api/profile/experience/:exp_id
// @desc     Delete experience from profile
// @access   Private so use Auth
router.delete('/experience/:exp_id', auth, async (req, res) => {

    try {
        const profile = await Profile.findOne({ user: req.user.id });

        // @GEt remove index
       const removeIndex = profile.experience.map(item =>
item.id).indexOf(req.params.exp_id);
       profile.experience.splice(removeIndex, 1);
       await profile.save();
       res.json(profile);
    } catch (err) {

        console.error(err.message);
        res.status(500).send('server error');

    }
});
```

Output
You can see there was 2 experience , now there is only one

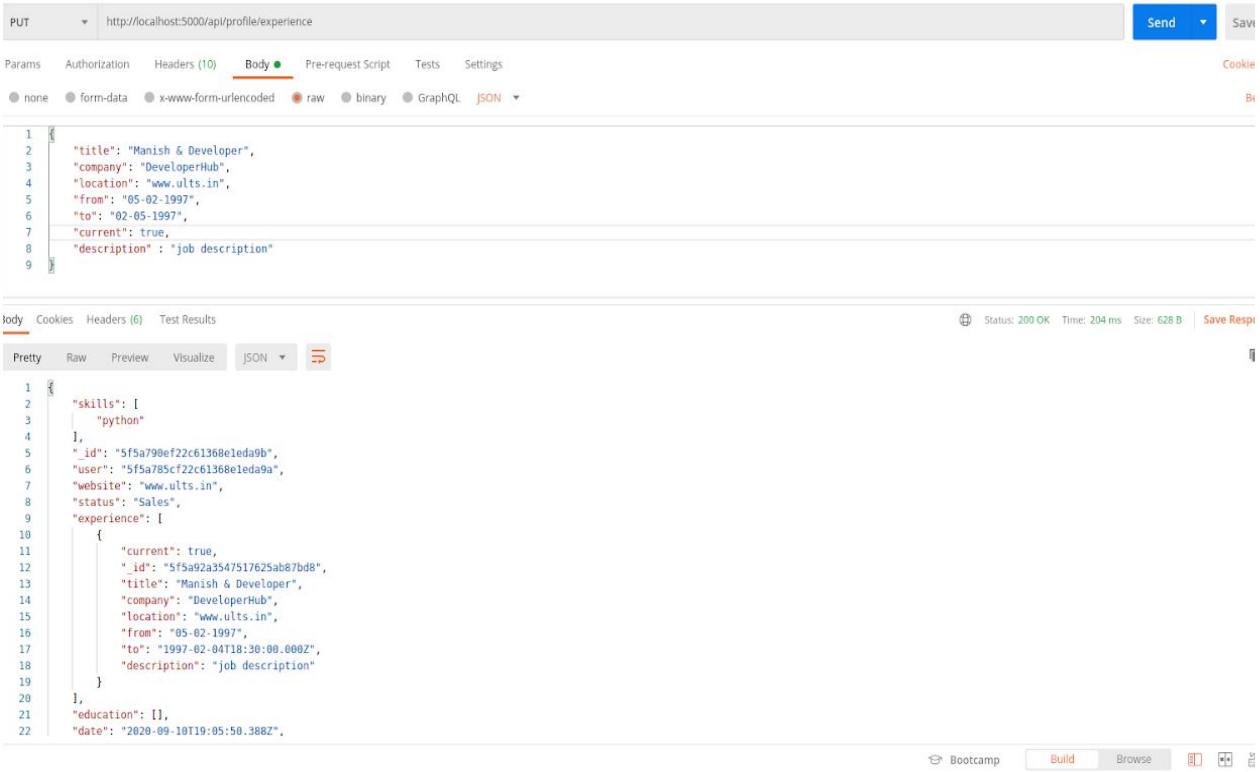


8. Add Profile Education

```
// @route   PUT api/profile/education
// @desc    Add profile education
// @access  Private so use Auth
// @this is Array of object

router.put('/education', [auth,

   check('school', 'school  is required').not().isEmpty(),
   check('degree', 'degree is required').not().isEmpty()
],
   async (req, res) => {
       const errors = validationResult(req);
       if (!errors.isEmpty()) {
           return res.status(400).json({ errors: errors.array() });
       }
       const {
           school,
           degree,
           fieldofstudy,
           from,
           to,
           current,
           description
       } = req.body;

       const Edu = {
           school,
           degree,
           fieldofstudy,
           from,
           to,
           current,
           description
       }


       try {
```

```
            const profile = await Profile.findOne({ user: req.user.id
});
            // @this is array of object so only using unshift and it
will add the element at the start of the index
            profile.education.unshift(Edu)
            await profile.save();
            res.json(profile);
        }
        catch (err) {
            console.error(err.message);
            res.status(500).send({ msg: "server Error" })
        }
    })
```

Output



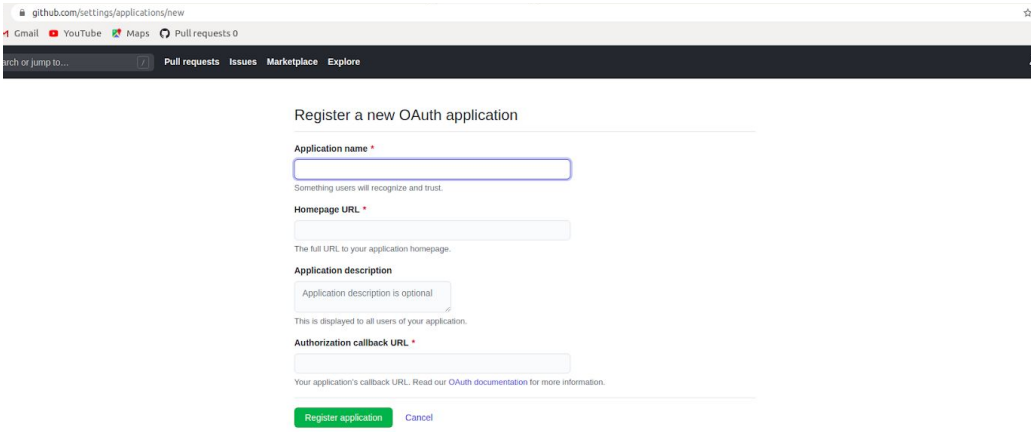
Delete the education from the profile

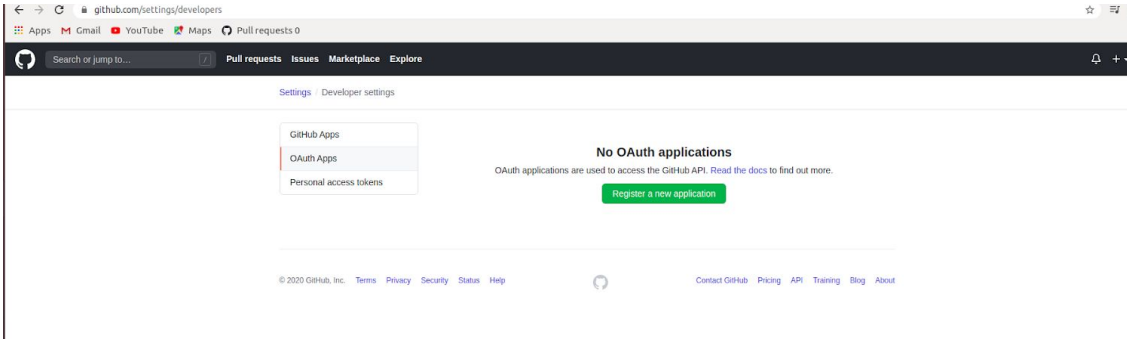
```
router.delete('/education/:edu_id', auth, async (req, res) => {

    try {
        const profile = await Profile.findOne({ user: req.user.id });
```

```
        // @GEt remove index
        const removeIndex = profile.education.map(item =>
item.id).indexOf(req.params.edu_id);
        profile.education.splice(removeIndex, 1);
        await profile.save();
        res.json(profile);
        return res.status(400).json({ msg: 'No education details
available for this profile' });

    } catch (err) {

        console.error(err.message);
        res.status(500).send('server error');

    }
});
```

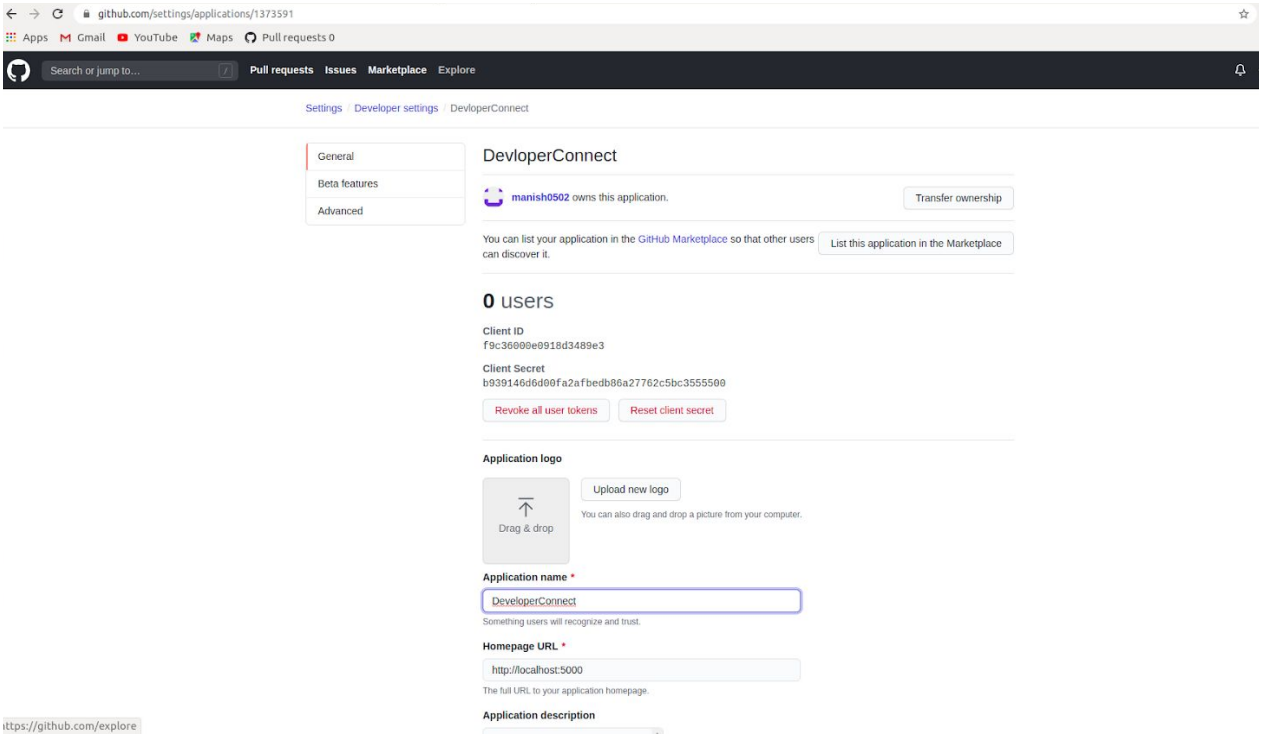
9. Get Github Repos For Profile

- Now we will try to get the repo of specific developer
- Follow the link https://github.com/settings/developers
- You can use to get client Oauth id
- https://developer.github.com/v3/guides/basics-of-authentication/#providing-a-callback
-





After registering a new Oauth Application you will get the page as below



Take clientid and github secret and copy this under config/default.json

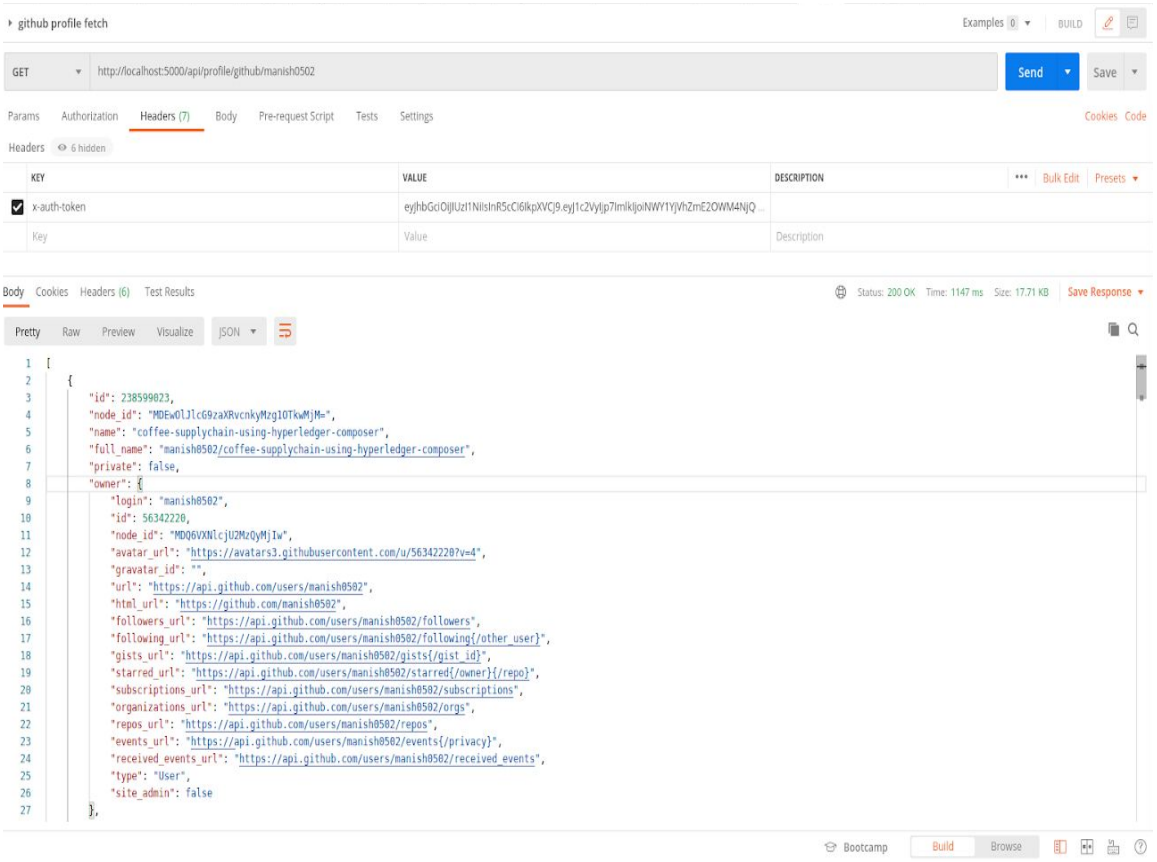
```
-  "githubClientId" : "f9c36000e0918d3489e3",
-   "githubSecret" : "b939146d6d00fa2afbedb86a27762c5bc3555500"
```

- Now go back to the profile.js and write a get api

```
// @10
// @route   DELETE api/profile/github/:username
// @desc    Get user repos from Github
// @access  Public

router.get('/github/:username' ,(req ,res)=>{

   try{
       const options = {
           uri:
`https://api.github.com/users/${req.params.username}/repos?per_page
=5&sort=created:

asc&client_id=${config.get('githubClientId')}&client_secret=${config.ge
t('githubSecret')}`,
           method : 'GET',
           headers: {'user-agent':'node.js'}
       };
       request(options ,(error , response ,body)=>{
           if(error) console.log(error);
           if(response.statusCode !== 200){
               res.status(404).json({ msg :'No github profile found'})
           }
           res.json(JSON.parse(body));
       })


   }catch(err){
       console.error(err.message);
       res.status(500).send('server error');
   }
});
```

Kindly export request and config in profile.js

Output -
This will list all the project publicly available repo



Extra-Things

PHASE-1 API

- https://www.getpostman.com/collections/d9adf0b8e571503f113c
- https://www.getpostman.com/collections/8709b16d9b16381ddd77

Make sure the database started

- sudo systemctl start mongodb
- mongo (connecting to: mongodb://127.0.0.1:27017)

For compass

- mongodb-compass

Link for installing mondodb

1. https://www.digitalocean.com/community/tutorials/how-to-install-mongodb-on-ubuntu-18-04
2. https://stackoverflow.com/questions/48092353/failed-to-start-mongod-service-unit-mongod-service-not-found

---

Note : To connect with local mongodb and compass use the above command to start

Make this change config/default.json

```
"mongoURI": "mongodb://localhost:27017/DevConnector",
```

And start mongo to start the connection

Section 5:Post API Routes

1. Creating The Post Model

- We will create model for post ,create a file models/Post.js

```
const mongoose = require('mongoose');

const PostSchema = new mongoose.Schema({


   // @we are linking each profile with user
   // @here ObjectId means _id from mongodb

   user: {
       type: mongoose.Schema.Types.ObjectId,
       ref: 'user'
   },
   text: {
       type: String,
       required: true
   },
   name: {
```

```
        type: String
    },

    // @even if the user deleted their accounts the post will not be
deleted

    avatar: {
        type: string
    },


    likes: [
        {
            user: {
                type: mongoose.Schema.Types.ObjectId,
                ref: 'user'
            }
        }
    ],
    comments: [
        {
            user: {
                type: mongoose.Schema.Types.ObjectId,
                ref: 'user'
            },

            text: {
                type: String,
                required: true
            },
            name: {
                type: String
            },
            avatar: {
                type: String
            },

            date: {
                type: String,
                default: Date.now
```

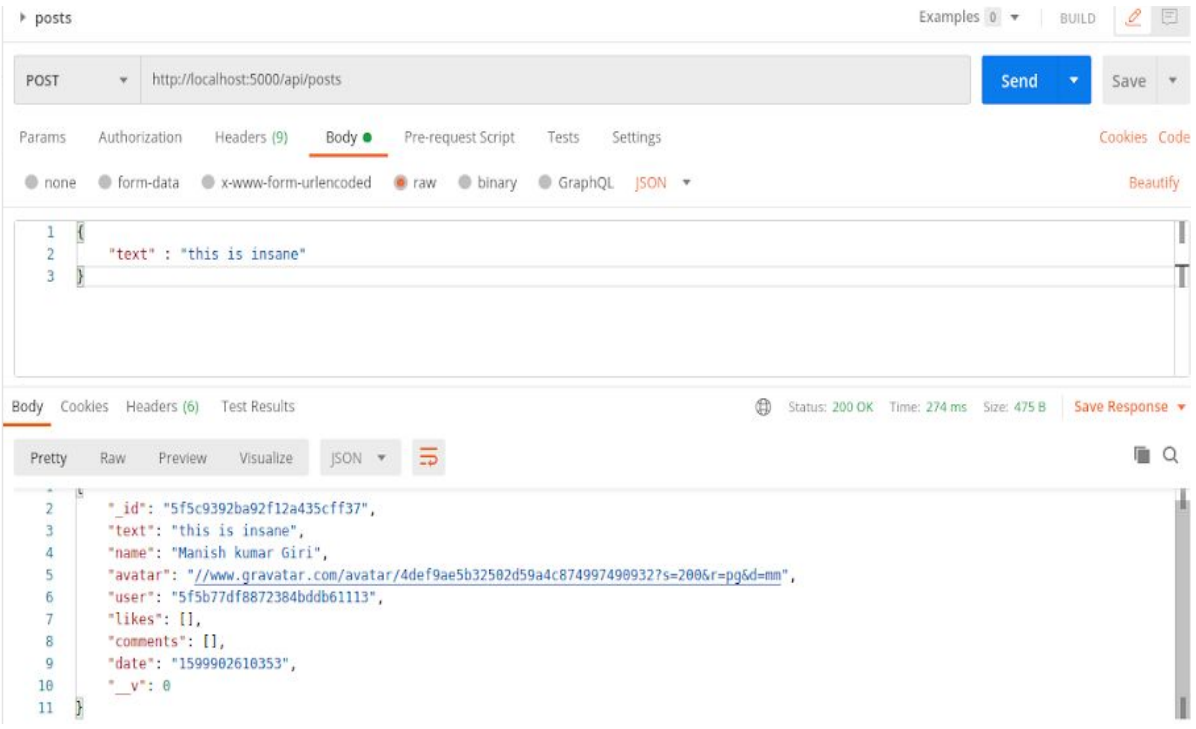
```
            },
        }
    ],

    date: {
        type: String,
        default: Date.now
    }

});

module.exports = PostSchema = mongoose.model('post', PostSchema);
```

2. Add Post Route

```
router.post('/', [auth,

    check('text', 'text is required').not().isEmpty()

],
    async (req, res) => {

        const errors = validationResult(req);
        if (!errors.isEmpty()) {
            return res.status(400).json({ errors: errors.array()
});
        }
        try {
            const user = await
User.findById(req.user.id).select('-password');

            const newPost = new Post({
                text: req.body.text,
                name: user.name,
                avatar: user.avatar,
                user: req.user.id
            });

            const post = await newPost.save();
```

```
            res.json(post);

        } catch (err) {
            console.log(err.message);
            res.status(500).send("Server Error");
        }



});
```

Output



3. Get and delete the post

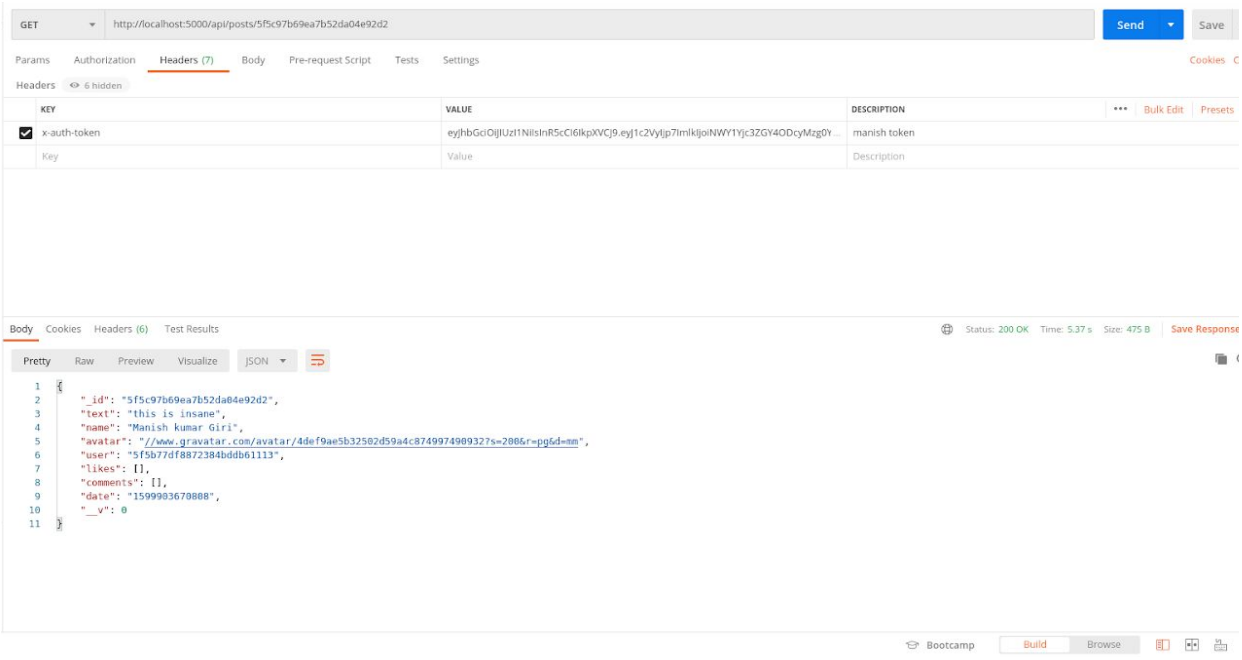
Tips : never leave any white space vehicle giving route

```
router.get('/:id ' , auth, async (req, res) => {
```

The above routing will not work because it contains one extra space

```
router.get('/:id', auth, async (req, res) => {

    try {

        const post = await Post.findById(req.params.id);
        if (!post) {
            return res.status(404).json({ msg: 'post not found' });
        }

        res.json(post);


    } catch (err) {

        if (err.kind == 'ObjectId') {
            return res.status(404).json({ msg: 'post not found' });
        }

        res.status(500).send("Server Error");
    }


});
```

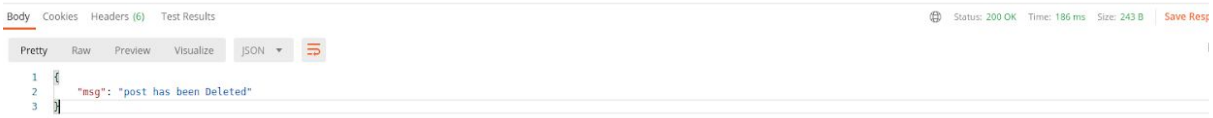
Output



Deleting the post by id

```
router.delete('/:id', auth, async (req, res) => {

   try {

       const post = await Post.findByIdAndRemove(req.params.id);
       if (!post) {
           return res.status(404).json({ msg: 'post not found' });
       }

       res.json({ msg: 'post has been Deleted' });


   } catch (err) {

       if (err.kind == 'ObjectId') {
           return res.status(404).json({ msg: 'post not found' });
       }

       res.status(500).send("Server Error");
   }


});
```

Output

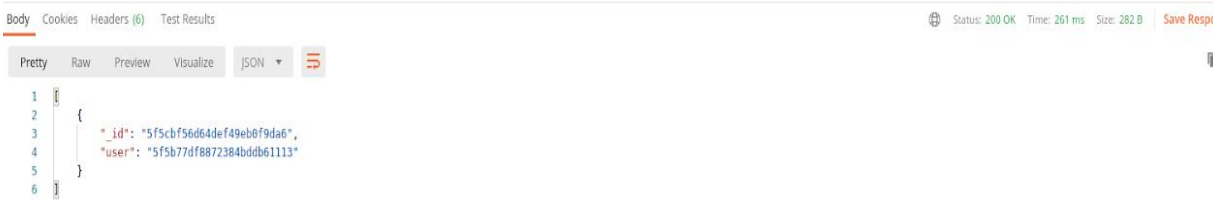


4. Post Like & Unlike Routes

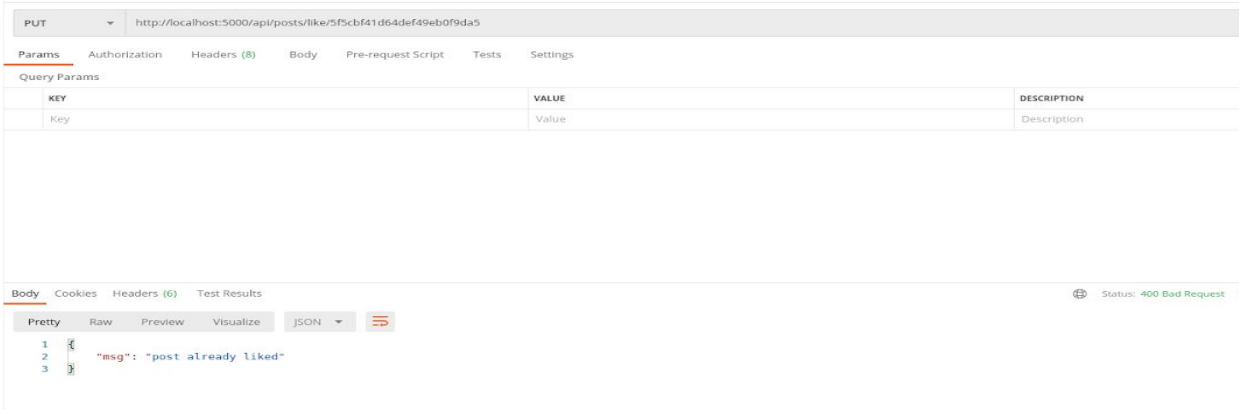
```
router.put('/like/:id' ,auth , async(req ,res)=>{
    try {

        const post = await Post.findById(req.params.id);

        // @Check if the post has already been liked
        // @Here likes is a array defined in model and filter is a array
method
        if(post.likes.filter(like =>like.user.toString() ===
req.user.id).length > 0){
            return res.status(400).json({msg : 'post already liked'})
        }

        post.likes.unshift({user :req.user.id})

        await post.save();
        res.json(post.likes)

    } catch (err) {

        if (err.kind == 'ObjectId') {
            return res.status(404).json({ msg: 'post not found' });
        }
        res.status(500).send("server error");
    }
```

```
} )
```

Output



If you click once more



And if you get all the post then

```
{
    "_id": "5f5cbf06d64def49eb0f9da3",
    "text": "this is insane and i am so happy & you too",
    "name": "Manish kumar Giri",
    "avatar": "//www.gravatar.com/avatar/4def9ae5b32502d59a4c8749974909327s=200&r=pg&d=mm",
    "user": "5f5b77df8872384bddb61113",
    "likes": [
        {
            "_id": "5f5cbf2ad64def49eb0f9da4",
            "user": "5f5b77df8872384bddb61113"
        }
    ],
    "comments": [],
    "date": "1599913734591",
    "__v": 1
}
```

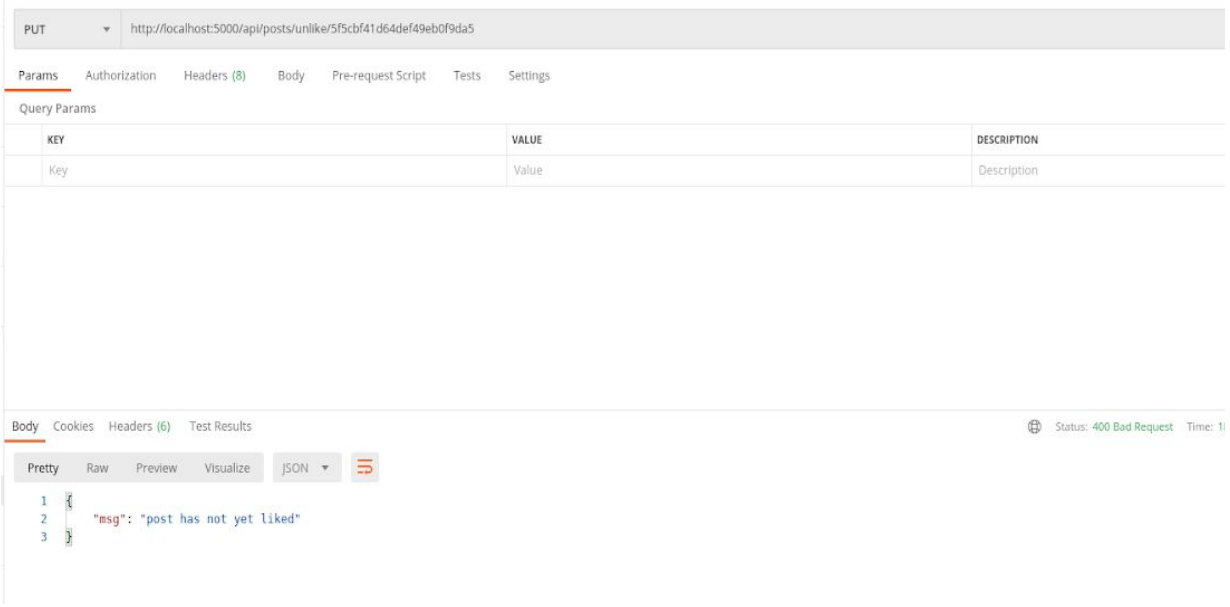

Now let's write for unlike

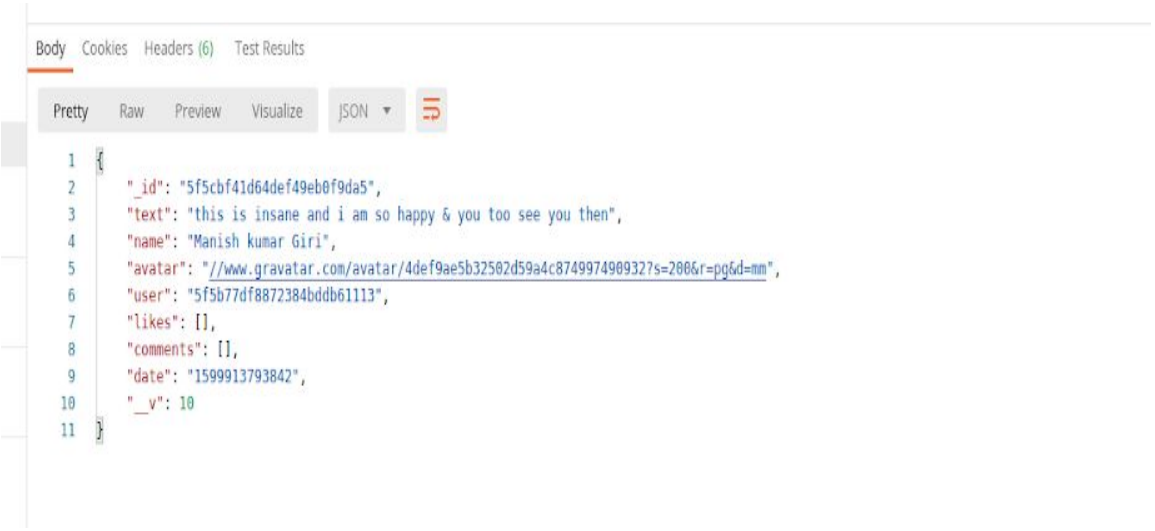
```
// @6
// @route   PUT api/posts/unlike/:id
// @desc    like a post
// @access  Private

router.put('/unlike/:id' ,auth , async(req ,res)=>{
   try {

       const post = await Post.findById(req.params.id);

       // @Check if the post is liked or not

       if(post.likes.filter(like =>like.user.toString() === req.user.id).length
== 0){
           return res.status(400).json({msg : 'post has not yet liked'})
       }

       // Get remove index
       const removeIndex = post.likes.map(like =>
like.user.toString()).indexOf(req.user.id);
       post.likes.splice(removeIndex, 1);
       await post.save();
       res.json(post)

   } catch (err) {

       if (err.kind == 'ObjectId') {
           return res.status(404).json({ msg: 'post not found' });
       }
```

```
        res.status(500).send("server error");
    }
})
```

Output

- When you press it will unlike or if you do once more it will be shown that post is not liked





5. Add & Remove Comment Routes

```
// @7
// @route   PUT api/posts/comment/:id
// @desc    Comment on a post
// @access  Private
router.post('/comment/:id', [auth,

   check('text', 'text is required').not().isEmpty()

],
   async (req, res) => {

       const errors = validationResult(req);
       if (!errors.isEmpty()) {
           return res.status(400).json({ errors: errors.array() });
       }
       try {
           const user = await
User.findById(req.user.id).select('-password');
           const post = await Post.findById(req.params.id)


           //we are not storing this to db
           const newComment = {
               text: req.body.text,
               name: user.name,
               avatar: user.avatar,
               user: req.user.id
           };
           post.comments.unshift(newComment);

           await post.save();
           res.json(post.comments);

       } catch (err) {
           console.log(err.message);
           res.status(500).send("Server Error");
       }
```

```
});
```

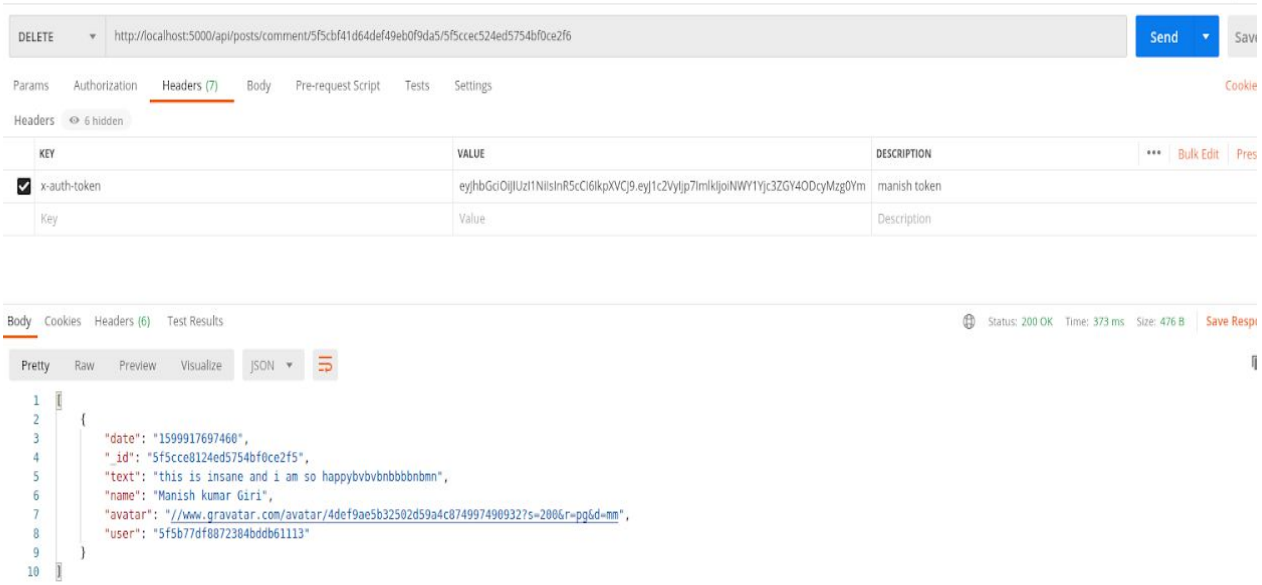
Output

Removing the comments

```
// @8
// @route    DELETE api/posts/comment/:id/:comment_id
// @desc    delete the Comment on a post
// @access   Private


router.delete('/comment/:id/:comment_id' , auth , async(req , res)=>{

    try {
```

```
        const post = await Post.findById(req.params.id);

        // @Pull out the comment
        const comment = post.comments.find(comment =>comment.id ===
req.params.comment_id);

        // @Make sure comment exists
        if(!comment){
            return res.status(404).json({ msg: 'No comment not found'
});
        }

        // @Check user
        if(comment.user.toString() !== req.user.id){
            return res.status(404).json({ msg: 'User not found' });

        }

        // @Get remove comments
        const removeIndex = post.comments.map(comment =>
comment.user.toString()).indexOf(req.user.id);
        post.comments.splice(removeIndex, 1);
        await post.save();
        res.json(post.comments)

    } catch (error) {
        console.log(err.message);
        res.status(500).send("Server Error");
    }
})
```

Output



API - 21
Posts - https://www.getpostman.com/collections/58ae417911bfaec3038e
Profiles- https://www.getpostman.com/collections/d9adf0b8e571503f113c
User & Auth - https://www.getpostman.com/collections/8709b16d9b16381ddd77

---------- Backend Done ----------

6. Asfa
7. Csa
8.

- We will create model for post ,create a file models/Post.js

1. Front-End : Based Of Reactjs

Link- https://www.youtube.com/watch?v=LXJOvkVYQqA&list=WL&index=79&t=910s

- Title - In-Depth React Tutorial: Build a Hotel Reservation Site (with Contentful and Netlify)

About :

-

Application-programming interface

Materials links :

1. https://www.youtube.com/watch?v=2jqok-Wgell&list=WL&index=75&t=3560s
2. https://www.youtube.com/watch?v=vjf774RKrLc&list=WL&index=72&t=2483s
3.

Build A Restful Api With Node.js Express & MongoDB