

# Important points

- Project link  
[https://github.com/bradtraversy/devconnector\\_2.0/tree/originalcoursecode](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

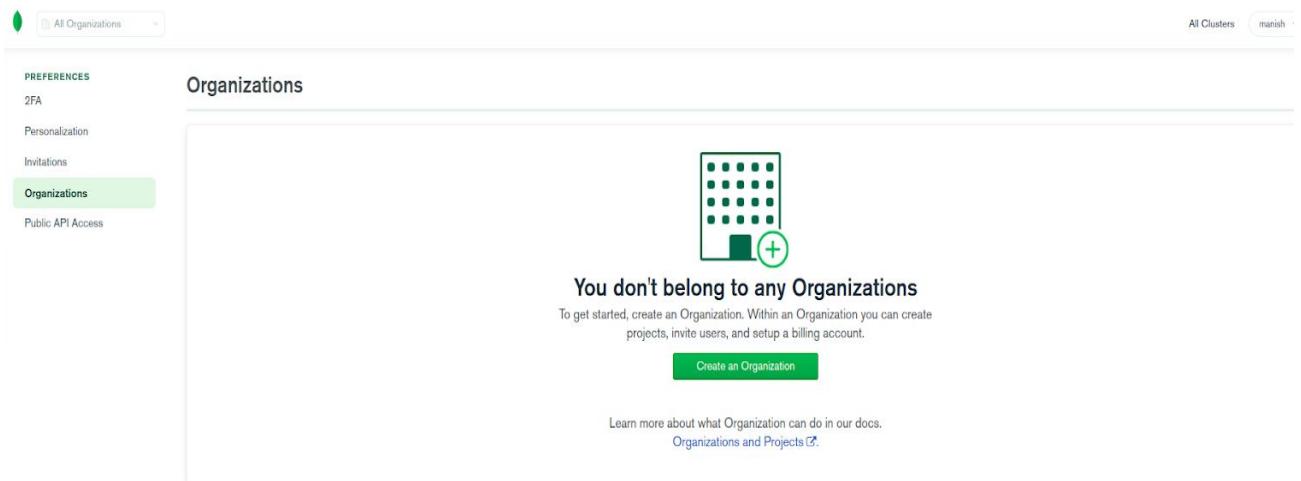
### Steps

#### 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](https://www.mongodb.com)
- b. Login ([manishgoswami495@gmail.com](mailto: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.

The screenshot shows the MongoDB Atlas interface under the 'Database Access' tab. On the left sidebar, 'Database Access' is selected. The main area is titled 'Database Access' and 'Database Users'. It features a green icon of a user with a plus sign. Below it, the text 'Create a Database User' and 'Set up database users, permissions, and authentication credentials in order to connect to your clusters.' A prominent green button labeled 'Add New Database User' is at the bottom. There is also a 'Learn more' link.

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

The screenshot shows the same MongoDB Atlas interface as above, but now with a user listed in the 'Database Users' table. The user 'manish123' is shown with 'SCRAM' as the authentication method and 'readWriteAnyDatabase@admin' as the MongoDB Roles. The 'Actions' column includes 'EDIT' and 'DELETE' buttons. A green button labeled '+ ADD NEW DATABASE USER' is visible at the top right of the table area.

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

The screenshot shows the MongoDB Atlas interface under the 'Network Access' tab, specifically the 'IP Access List' section. A modal dialog is open titled 'Add IP Access List Entry'. It contains fields for 'User Name' (set to 'manish123'), 'Authentication Method' (set to 'SCRAM'), 'MongoDB Roles' (set to 'readWriteAnyDatabase@admin'), and 'Resources' (set to 'All Resources'). An 'Actions' column has 'EDIT' and 'DELETE' buttons. At the bottom of the dialog, there is a note about temporary entries and a 'Confirm' button. The background shows the 'Network Access' interface with tabs for 'IP Access List', 'Peering', and 'Private'.

i. The the next screen will be

Network Access

IP Access List

IP Address	Comment	Status	Actions
0.0.0.0/0 (includes your current IP address)		Active	<a href="#">EDIT</a> <a href="#">DELETE</a>

j. Then come back to cluster and select connect

Clusters

SANDBOX

DevConnector

CONNECT

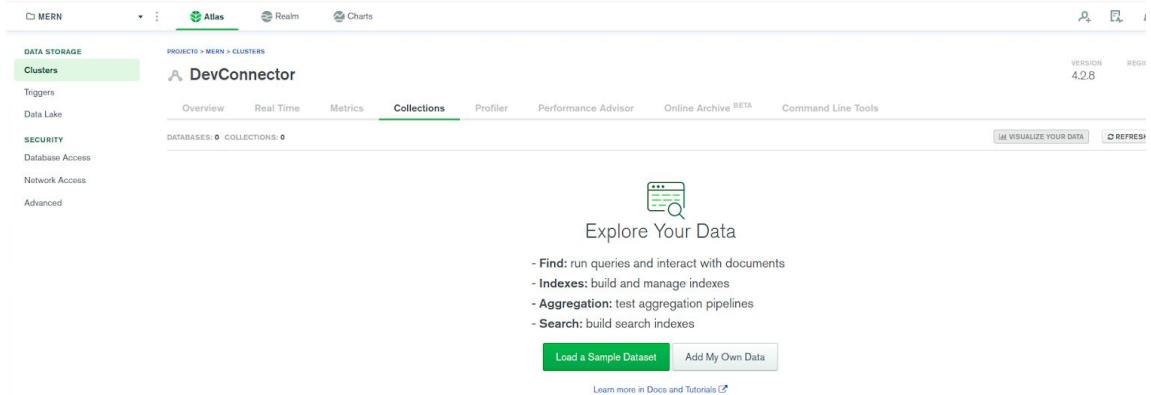
- k. Then select connect your application and copy this so we can use this as plugin
- l. `mongodb+srv://manish123:<password>@devconnector.qe3yb.mongodb.net/<dbname>?retryWrites=true&w=majority`

CONNECT

Click to Copy

Copy

m. Go to collection to see the data



The screenshot shows the MongoDB Atlas interface for a project named 'MERN'. The left sidebar has sections for 'DATA STORAGE' (Clusters, Triggers, Data Lake), 'SECURITY' (Database Access, Network Access, Advanced), and 'Atlas' (Real Time, Metrics, Profiler, Performance Advisor, Online Archive BETA, Command Line Tools). The main area shows a cluster named 'DevConnector' with 0 databases and 0 collections. A central panel titled 'Explore Your Data' provides instructions for using the interface: 'Find: run queries and interact with documents', 'Indexes: build and manage indexes', 'Aggregation: test aggregation pipelines', and 'Search: build search indexes'. It also includes buttons to 'Load a Sample Dataset' and 'Add My Own 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.

<a href="http://www.gravatar.com/avatar/{HASH VALUE} [.jpg] [ ?param=value&amp;param2=value&amp;... ]">http://www.gravatar.com/avatar/{HASH VALUE} [.jpg] [ ?param=value&amp;param2=value&amp;... ]</a>		
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

<a href="http://www.gravatar.com/{HASH VALUE}[File extension]">http://www.gravatar.com/{HASH VALUE}[File extension]</a>	
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

The screenshot shows the jwt.io interface. On the left, under 'Encoded' (PASTE A TOKEN HERE), is a long string of characters: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9IiwiZW5jb2RlIjoxNTE2MjM5MDIyfQ.Sf1KxwRJSMeKKF2QT4fwpMeJf36P0k6yJV\_adQssw5c. On the right, under 'Decoded' (EDIT THE PAYLOAD AND SECRET), the token is split into three sections: HEADER: ALGORITHM & TOKEN TYPE, PAYLOAD: DATA, and VERIFY SIGNATURE. The HEADER section shows "typ": "JWT". The PAYLOAD section shows "name": "John Doe", "iat": 1516239822. The VERIFY SIGNATURE section shows the HMACSHA256 calculation: base64UrlEncode(header) + "." + base64UrlEncode(payload), followed by a placeholder for the secret: your-256-bit-secret. A note indicates that the secret should be base64 encoded.

- 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

The screenshot shows a terminal window with the following content:

```
File Edit Selection View Go Run Terminal Help
□ package.json JS server.js ×
JS server.js ...
1 const express= require('express');
2
3 const app = express();
4
5 app.get('/',(req , res)=>{
6     res.send('API Running');
7 })
8
9 const PORT = process.env.PORT || 5000;
10
11 app.listen(PORT , ()=>{
12     console.log(`server started on port ${PORT}`);
13 });

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
+ concurrently@5.3.0
added 152 packages from 83 contributors and audited 324 packages in 30.237s
16 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
manish@ULTS-LAP106:~/Desktop/Dev-con$ npm run server
> dev-con@1.0.0 server /home/manish/Desktop/Dev-con
> nodemon server
[nodemon] 2.0.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
server started on port 5000
```

#### 4. Go to PostMan and test your api

a. <http://localhost:5000>

The screenshot shows the PostMan application interface with the following details:

- Request URL: `http://localhost:5000/users`
- Method: `GET`
- Headers:
  - Authorization
  - Headers (9)
  - Body (x)
  - Pre-request Script
  - Tests
  - Settings
- Query Params table:

KEY	VALUE	DESCRIPTION	...	Bulk
Key	Value	Description		
- Body tab:
  - Pretty
  - Raw
  - Preview
  - Visualize
  - HTML (selected)
- Response status: `Status: 200 OK Time: 410 ms Size: 215 B`
- Response body:

```
1 API Running
```

## 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 is 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

The screenshot shows the Postman interface with a collection named "users". A POST request is selected with the URL "http://localhost:5000/api/users". The "Headers" tab is active, displaying the following header:

KEY	VALUE	DESCRIPTION
Content-Type	application/json	
Key	Value	Description

In the body

The screenshot shows the Postman interface. At the top, it says "POST" and "http://localhost:5000/api/users". Below that, under "Body", there is a JSON input field containing the following code:

```
1 {
2   "name" : "Manish"
3 }
```

At the bottom right, the status bar shows "Status: 200 OK" and "Time: 239 ms".

- 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

The screenshot shows the Postman interface with a POST request to `http://localhost:5000/api/users`. The Body tab is selected, containing the following JSON:

```
1 {
2   "name" : "manish",
3   "email" : "manishgoswami495@gmail.com",
4   "password" : "1267123"
5 }
```

The response status is 200 OK, with a time of 38 ms and a size of 214 B. The response body is:

```
1 user route
```

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

The screenshot shows the Postman interface with a POST request to `http://localhost:5000/api/users`. The Body tab is selected, containing the following JSON:

```
1 {
2   "name" : "",
3   "email" : "manishgoswami49l.com",
4   "password" : "1267123"
5 }
```

The response status is 400 Bad Request, with a time of 7 ms and a size of 405 B. The response body is:

```
1 {
2   "errors": [
3     {
4       "value": "",
5       "msg": "Name is required",
6       "param": "name",
7       "location": "body"
8     },
9     {
10       "value": "manishgoswami49l.com",
11       "msg": "Please enter valid Email",
12       "param": "email",
13     }
14   ]
15 }
```

### 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 plain 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.eyJlc2VyIjp7ImlkIjoiNWY1N2JkNTNk
NTZhOWMyZDYxYzBjYzA0In0sImlhCI6MTU5OTU4NTYxOSwiZXhwIjoxNTk5NjIxNjE5fQ
.luo-C8YItYKpKtdJWVAI4sMrSawex4H3S6T9EaLWOdQ"
```

}

d. You can recheck your 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's create middleware in root
- Inside this create file auth.js

- We take the header and payload from the upcoming json 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

The screenshot shows a Postman interface with the following details:

- Method:** GET
- URL:** http://localhost:5000/api/auth
- Headers (7):**

| KEY          | VALUE                                                        | DESCRIPTION |
|--------------|--------------------------------------------------------------|-------------|
| x-auth-token | eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vyljp7imkijojN... |             |
| Key          | Value                                                        | Description |
- Body:** (Empty)
- Test Results:** Status: 200 OK, Time: 13 ms, Size: 214 B
- Preview:** Auth route

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')
```

The screenshot shows a Postman interface. At the top, it says "GET" and "http://localhost:5000/api/auth". Below that, under "Headers (7)", there is a table with one row: "x-auth-token" with value "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vyljp7imlkijoiN...". Under "Body", the response tab is selected, showing a JSON object with fields: "\_id": "5f588c0da0cbfe35f8ce8361", "name": "manish Kumar Giri", "email": "manish@gmail.com", "avatar": "//www.gravatar.com/avatar/4def9ae5b32502d59a4c8749974969327s=200&r=pg&d=mm", and "date": "2020-09-09T08:02:21.157Z". The status bar at the bottom indicates "Status: 200 OK Time: 228 ms Size: 429 B".

## 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

The screenshot shows the Postman interface with a collection tree on the left containing various API endpoints. In the main workspace, a POST request to 'User Login' is being tested. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "email" : "manish@gmail.com",
3   "password" : "1267123"
4 }
```

The status bar at the bottom indicates a **400 Bad Request** response.

The screenshot shows the Postman interface with the same collection tree. A POST request to 'User Login' is now successful, indicated by a **200 OK** status bar message. The response body contains a generated JWT token:

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
3   eyJlc2VyIjp7ImlkIjoiNWY1ODhjMGRhMGNiZmUzMWY4Y2U4MzYxIn0sImhdC16MTU5OTY0MTMxNiwizXhwIjoxNTk5Njc3MzE2f0.
4   XUEc8b1w8nn4jYp-BGtuUhY8ThwqlSunq48dPved-dU"
5 }
```

# 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

The screenshot shows two panels of the Postman interface. The left panel displays a list of collections and profiles, with 'Profiles' currently selected. The right panel shows a 'MANAGE HEADER PRESETS' dialog and a 'profile create' request configuration.

**MANAGE HEADER PRESSETS Dialog:**

| KEY          | VALUE            | DESCRIPTION       | Bulk Edit |
|--------------|------------------|-------------------|-----------|
| Content-Type | application/json | Json Content type |           |
| Key          | Value            | Description       |           |

**profile create Request Configuration:**

Method: POST | URL: http://localhost:5000/api/profile

Headers (10):

| KEY          | VALUE            | DESCRIPTION  | Bulk Edit | Presets      |
|--------------|------------------|--------------|-----------|--------------|
| Content-Type | application/json |              |           |              |
| x-auth-token | Content-Type     | manish token |           | manish token |
| Key          | Value            | Description  |           |              |

The image shows a composite view of a REST API testing environment, likely Postman, demonstrating the creation of a profile and handling of authentication tokens.

**Left Panel (Collection View):**

- History:** Shows 2 requests under the "Profiles" collection.
- Collections:**
  - New Collection:** "Profiles" (2 requests)
  - profile check** (GET)
  - profile create** (POST) - Selected
  - Seafood** (16 requests)
  - cloths-supplychain** (40 requests)
  - cloths-supplychain Copy**
- APIs:** Shows 0 requests.
- Trash:** Shows 0 requests.

**Top Right Panel (Request Editor):**

Profile create

POST http://localhost:5000/api/profile

Headers (9)

| KEY                                              | VALUE            | DESCRIPTION |
|--------------------------------------------------|------------------|-------------|
| <input checked="" type="checkbox"/> Content-Type | application/json |             |
| Key                                              | Value            | Description |

**Middle Panel (Header Presets):**

MANAGE HEADER PRESETS

Add Header Preset: manish token

| KEY                                              | VALUE        | DESCRIPTION  |
|--------------------------------------------------|--------------|--------------|
| <input checked="" type="checkbox"/> x-auth-token | Content-Type | manish token |
| Key                                              | Value        | Description  |

**Bottom Panel (Request Details):**

profile

Headers (7)

| KEY                                              | VALUE                                    | DESCRIPTION |
|--------------------------------------------------|------------------------------------------|-------------|
| <input checked="" type="checkbox"/> x-auth-token | eyJhbGciOiJIUzI1NiIsInRS... (long token) |             |
| Key                                              | Value                                    | Description |

Body, Cookies, Headers (6), Test Results

Pretty, Raw, Preview, Visualize, JSON

Status: 400 Bad Request, Time: 133 ms, Size: 264 B, Save Response

```

1 {
2   "msg": "There is no profile for the user"
3 }

```

## So the output for the create profile

The screenshot shows the Postman interface with a successful POST request to `http://localhost:5000/api/profile`. The response body is displayed in JSON format:

```
1  [
2      "company": "ults",
3      "status": "Developer",
4      "website": "www.ults.in",
5      "skills": "python , html,css",
6      "location": "kerla",
7      "bio" : "i am developer",
8      "githubusername" : "manish",
9      "twitter" : "https://twitter.com/manish",
10     "facebook" : "https://facebook.com/manish",
11     "youtube" : "https://youtube.com/manish"
12 ]
```

Below the JSON, the status bar indicates: Status: 200 OK Time: 258 ms Size: 622 B Save Res.

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON

## Update and check while making the change in the bio

The screenshot shows the Postman interface with a successful POST request to `http://localhost:5000/api/profile`. The response body is displayed in JSON format:

```
1  [
2      "company": "ults",
3      "status": "Developer",
4      "website": "www.ults.in",
5      "skills": "python , html,css",
6      "location": "kerla",
7      "bio" : "i am senior developer",
8      "githubusername" : "manish",
9      "twitter" : "https://twitter.com/manish",
10     "facebook" : "https://facebook.com/manish",
11     "youtube" : "https://youtube.com/manish"
12 ]
```

Below the JSON, the status bar indicates: Status: 200 OK Time: 989 ms Size: 629 B Save Res.

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON

POST http://localhost:5000/api/profile

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

=> 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: "5f5a3fb005c6245cb5af238f",
"user": {
    "_id": "5f5a215f3e2ae93180487e7e",
    "name": "manish Kumar Geiri",
    "avatar": "//www.gravatar.com/avatar/c8edb345f5c67c4134955f82e11ea4397s=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) {

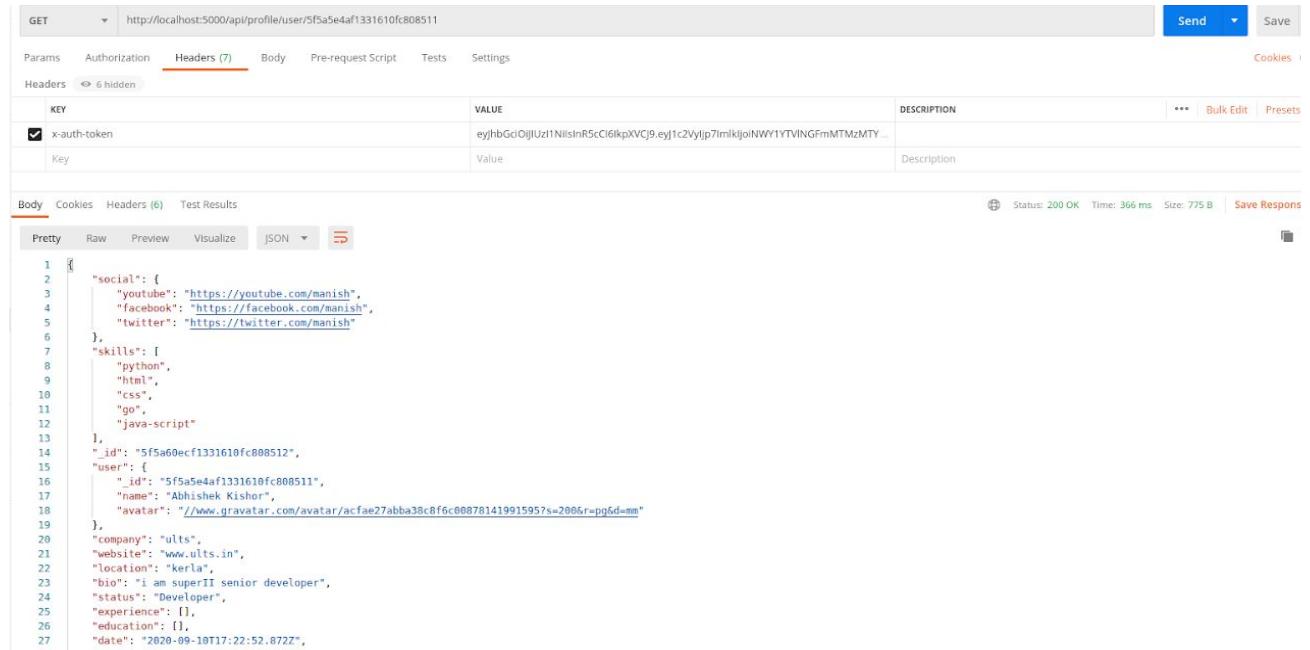
        console.error(err.message);
        // @what if we are giving wrong format of objectid ,then also it should
        // give the msg like "profile not exists instead of server error"
        if(err.kind == 'ObjectId'){
            return res.status(400).json({msg : 'There is no profile for
this user'});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

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

    }
}

```

## Output



The screenshot shows a Postman request to `http://localhost:5000/api/profile/user/5f5a5e4af1331610fc808511`. The Headers tab is selected, showing a single header `x-auth-token` with the value `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJ1c2Vyljp7imlkjoiNWY1YTIVNGFmMTMzMThlZC19`. The Body tab shows the JSON response:

```

1 {
2     "social": {
3         "youtube": "https://youtube.com/manish",
4         "facebook": "https://facebook.com/manish",
5         "twitter": "https://twitter.com/manish"
6     },
7     "skills": [
8         "python",
9         "html",
10        "css",
11        "go",
12        "java-script"
13    ],
14    "_id": "5f5a60ecf1331610fc808512",
15    "user": {
16        "id": "5f5a5e4af1331610fc808511",
17        "name": "Abhishek Kishor",
18        "avatar": "/www.gravatar.com/avatar/acfae27abba38c8f6c00878141991595?size=200&r=pg&d=mm"
19    },
20    "company": "ults",
21    "website": "www.ults.in",
22    "location": "Kerala",
23    "bio": "i am superII senior developer",
24    "status": "Developer",
25    "experience": [],
26    "education": [],
27    "date": "2020-09-10T17:22:52.872Z"
28 }

```

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

The screenshot shows a Postman interface with the following details:

- Method:** GET
- URL:** http://localhost:5000/api/profile/user/5f5a5e4af1331610fc808511122
- Headers:**
  - x-auth-token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9eyJ1c2VyIjp7ImRlcjoiNWY1YTBlNGFmMTMzMjMTY... (with a long JWT string)
  - Key: Value (empty)
- Body:** JSON response:
 

```

1 {
2   "msg": "There is no profile for this user"
3 }
```
- Status:** 400 Bad Request
- Time:** 134 ms
- Size:** 264 B

## 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

```
1 {
2   "title": "Manish & Developer",
3   "company": "DeveloperHub",
4   "location": "www.ults.in",
5   "from": "05-02-1997",
6   "to": "02-05-1997",
7   "current": true,
8   "description" : "job description"
9 }
```

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "skills": [
3     "python"
4   ],
5   "_id": "5f5a790ef22c61368e1eda9b",
6   "user": "5f5a785cf22c61368e1eda9a",
7   "website": "www.ults.in",
8   "status": "Sales",
9   "experience": [
10     {
11       "current": true,
12       "_id": "5f5a92a3547517625ab87bd8",
13       "title": "Manish & Developer",
14       "company": "DeveloperHub",
15       "location": "www.ults.in",
16       "from": "05-02-1997",
17       "to": "1997-02-04T18:30:00.000Z",
18       "description": "job description"
19     }
20   ],
21   "education": [],
22   "date": "2020-09-10T19:05:50.388Z".
}
```

⊕ Status: 200 OK Time: 204 ms

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) {

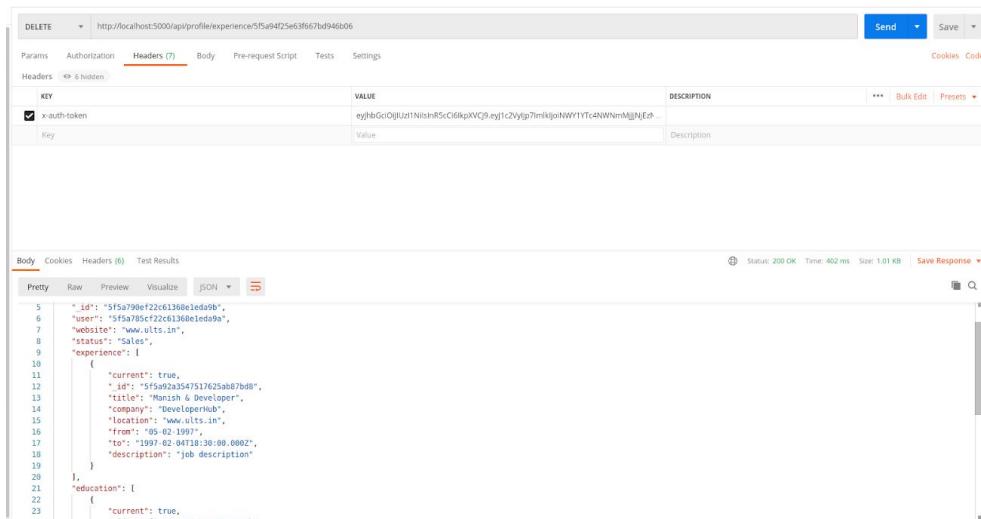
    console.error(err.message);
    res.status(500).send('server error');

  }
}

});
```

### Output

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



The screenshot shows a Postman request to `http://localhost:5000/api/profile/experience/5f5a790ef22c61368e1ed9a9`. The Headers tab shows a `x-auth-token` header with a long token value. The Body tab shows the JSON response:

```
5  "id": "5f5a790ef22c61368e1ed9a9",
6  "user": "5f5a785fc22c61368e1ed9a9",
7  "website": "www.ults.in",
8  "status": "Sales",
9  "experience": [
10    {
11      "current": true,
12      "id": "5f5a7923e0351765a887bd8",
13      "title": "Tech & Developer",
14      "company": "DeveloperHub",
15      "location": "www.ults.in",
16      "from": "05-02-1997",
17      "to": "1997-02-04T18:30:00.000Z",
18      "description": "Job description"
19    }
20  ],
21  "education": [
22    {
23      "current": true,
24      "id": "5f5a7923e0351765a887bd8"
25    }
26  ]
}
```

## 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

The screenshot shows a Postman request for a PUT operation on the URL `http://localhost:5000/api/profile/experience`. The request body is a JSON object representing an education entry:

```

1  {
2     "title": "Manish & Developer",
3     "company": "DeveloperHub",
4     "location": "www.ults.in",
5     "from": "05-02-1997",
6     "to": "02-05-1997",
7     "current": true,
8     "description" : "job description"
9

```

The response status is 200 OK, with a time of 204 ms and a size of 628 B. The response body is identical to the request body, indicating the update was successful.

## 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>
-

The image shows two screenshots of the GitHub developer settings interface. The top screenshot is titled 'Register a new OAuth application' and contains fields for 'Application name', 'Homepage URL', 'Application description', and 'Authorization callback URL'. The bottom screenshot shows the 'OAuth Apps' section with a 'No OAuth applications' message and a 'Register a new application' button.

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

The image shows the 'DeveloperConnect' configuration page for a registered OAuth application. It displays the Client ID (f9c36000e0918d3489e3), Client Secret (b399146dd6d0fa2afbedb86a27762c5bc3555500), and other application details like logo, name, and URL.

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

The screenshot shows a Postman interface with the following details:

- Request URL:** http://localhost:5000/api/profile/github/manish0502
- Method:** GET
- Headers:** **auth-token**: eyJhbGciOiJIUzI1NiIsInR5Ci6IkpXVC9eyJc1c2Vylp7imlkjoiNWY1Yjh2mE2OWM4NjQ...  
Key: Value  
Description: Description
- Body:** Status: 200 OK | Time: 1147 ms | Size: 17.71 KB | Save Response
- Response:** A JSON object representing a GitHub user profile. The response body is as follows:

```
1 [  
2 {  
3   "id": 238599023,  
4   "node_id": "MDExOlJlcG9zaXRvcnkyMzg1OTkzMjM=",  
5   "name": "coffee-supplychain-using-hyperledger-composer",  
6   "full_name": "manish0502/coffee-supplychain-using-hyperledger-composer",  
7   "private": false,  
8   "owner": {  
9     "login": "manish0502",  
10    "id": 56342220,  
11    "node_id": "MDQ6VXNlcjU2MzQyMjIw",  
12    "avatar_url": "https://avatars3.githubusercontent.com/u/56342220?v=4",  
13    "gravatar_id": "",  
14    "url": "https://api.github.com/users/manish0502",  
15    "html_url": "https://github.com/manish0502",  
16    "followers_url": "https://api.github.com/users/manish0502/followers",  
17    "following_url": "https://api.github.com/users/manish0502/following{/other_user}",  
18    "gists_url": "https://api.github.com/users/manish0502/gists{/gist_id}",  
19    "starred_url": "https://api.github.com/users/manish0502/starred{/owner}{/repo}",  
20    "subscriptions_url": "https://api.github.com/users/manish0502/subscriptions",  
21    "organizations_url": "https://api.github.com/users/manish0502/orgs",  
22    "repos_url": "https://api.github.com/users/manish0502/repos",  
23    "events_url": "https://api.github.com/users/manish0502/events{/privacy}",  
24    "received_events_url": "https://api.github.com/users/manish0502/received_events",  
25    "type": "User",  
26    "site_admin": false  
},  
27 ]
```

## 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 monododb

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({
    title: {
        type: String,
        required: true
    },
    description: {
        type: String,
        required: true
    },
    user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'user'
    },
    text: {
        type: String,
        required: true
    },
    name: {
        type: String,
        required: true
    }
}, {
    timestamps: true
});

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

```
        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

The screenshot shows the Postman application interface. At the top, there's a header bar with 'posts' in the left corner, 'Examples 0', 'BUILD', and icons for 'Send', 'Save', and 'Copy'. Below the header, the main interface has several tabs: 'POST' (selected), 'http://localhost:5000/api/posts', 'Send' (blue button), 'Save', and 'Code'. Underneath these are sub-tabs: 'Params', 'Authorization', 'Headers (9)', 'Body' (selected, highlighted in green), 'Pre-request Script', 'Tests', and 'Settings'. Below the tabs, there are radio buttons for 'none', 'form-data', 'x-www-form-urlencoded', 'raw' (selected, highlighted in orange), 'binary', 'GraphQL', and 'JSON' (dropdown menu). To the right of these buttons are 'Cookies' and 'Code' buttons, and a 'Beautify' link. The 'Body' section contains a code editor with the following JSON payload:

```
1 {
2     "text" : "this is insane"
3 }
```

Below the body editor, there are tabs for 'Body', 'Cookies', 'Headers (6)', and 'Test Results'. The 'Body' tab is selected. At the bottom of the interface, there's a status bar showing 'Status: 200 OK', 'Time: 274 ms', 'Size: 475 B', 'Save Response', and a search icon.

The 'Body' tab displays the JSON response from the server. The response is a single object with the following properties:

```
1 {
2     "_id": "5f5c9392ba92f12a435cff37",
3     "text": "this is insane",
4     "name": "Manish Kumar Giri",
5     "avatar": "//www.gravatar.com/avatar/4def9ae5b32582d59a4c8749974909327s=200&r=pg&d=mm",
6     "user": "5fb77df8872384bddb61113",
7     "likes": [],
8     "comments": [],
9     "date": "1599902610353",
10    "_v": 0
11 }
```

### 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

The screenshot shows a Postman request for `http://localhost:5000/api/posts/5f5c97b69ea7b52da04e92d2`. The Headers tab is selected, showing an `x-auth-token` header with value `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJ1c2Vylp7mlkjjoNWY1Yj3ZGY4ODcyMzg0Y...` and a `Key` header with value `Description`. The Body tab displays the response in Pretty format:

```
1  {
2     "_id": "5f5c97b69ea7b52da04e92d2",
3     "text": "this is insane",
4     "name": "Manish kumar Giri",
5     "avatar": "/www.gravatar.com/avatar/4def9ae5b32502d59a4c874997490932?s=200&r=pg&d=mm",
6     "user": "5fb77df8872384bddb61113",
7     "likes": [],
8     "comments": [],
9     "date": "1599903670888",
10    "__v": 0
11 }
```

## 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

The screenshot shows the Postman interface. At the top, there's a header with 'DELETE' and the URL 'http://localhost:5000/api/posts/5f5cb48bbaa7d740bf7a865e'. Below the header are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is selected. Under 'Params', there's a table with one row: 'Key' (Value) and 'Value' (Description). The body tab shows a JSON response with the key 'msg' and value 'post has been Deleted'. The status bar at the bottom indicates a 200 OK status, 186 ms time, and 243 B size.

## 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

The screenshot shows a Postman request to `http://localhost:5000/api/posts/like/5f5cbf41d64def49eb0f9da5`. The response status is 200 OK, time is 261 ms, and size is 282 B. The JSON response body is:

```
1 [ ]  
2 {  
3   "_id": "5f5cbf56d64def49eb0f9da6",  
4   "user": "5f5b77df8872384bddb61113"  
5 }  
6 ]
```

If you click once more

The screenshot shows a Postman request to `http://localhost:5000/api/posts/like/5f5cbf41d64def49eb0f9da5`. The response status is 400 Bad Request. The JSON response body is:

```
1 [ ]  
2 {  
3   "msg": "post already liked"  
4 }  
5 ]
```

And if you get all the post then

```
1  {
2      "_id": "5f5cbf06d64def49eb0f9da3",
3      "text": "this is insane and i am so happy & you too",
4      "name": "Manish kumar Giri",
5      "avatar": "//www.gravatar.com/avatar/4def9ae5b32502d59a4c874997490932?s=200&r=pg&d=mm",
6      "user": "5f5b77df8872384bddb61113",
7      "likes": [
8          {
9              "_id": "5f5cbf2ad64def49eb0f9da4",
10             "user": "5f5b77df8872384bddb61113"
11         }
12     ],
13     "comments": [],
14     "date": "1599913734591",
15     "__v": 1
16 }
```

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

PUT http://localhost:5000/api/posts/unlike/5f5cbf41d64def49eb0f9da5

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Query Params

| KEY | VALUE | DESCRIPTION |
|-----|-------|-------------|
| Key | Value | Description |

Body Cookies Headers (6) Test Results

Status: 400 Bad Request Time: 1s

Pretty Raw Preview Visualize JSON ↻

```
1 {  
2   "msg": "post has not yet liked"  
3 }
```

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON ↻

```
1 {  
2   "_id": "5f5cbf41d64def49eb0f9da5",  
3   "text": "this is insane and i am so happy & you too see you then",  
4   "name": "Manish kumar Giri",  
5   "avatar": "//www.gravatar.com/avatar/4def9ae5b32502d59a4c874997490932?s=200&r=pg&d=mm",  
6   "user": "5f5b77df8872384bddb61113",  
7   "likes": [],  
8   "comments": [],  
9   "date": "1599913793842",  
10  "_v": 10  
11 }
```

## 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

DELETE http://localhost:5000/api/posts/comment/5f5cbf41d64def49eb0f9da5/5f5ccce524ed5754bf0ce2f6

Headers (7)

| KEY          | VALUE                                                                         | DESCRIPTION  |
|--------------|-------------------------------------------------------------------------------|--------------|
| x-auth-token | eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJfc2Vyljp7imikjjoNWY1Yjc3ZGY4ODcyMzg0Ym | manish token |
| Key          | Value                                                                         | Description  |

Body

```

1
2
3
4
5
6
7
8
9
10
{
  "date": "1599917697460",
  "id": "5f5ccce524ed5754bf0ce2f5",
  "text": "this is insane and i am so happyvbvbnnbm",
  "name": "Manish kumar Giri",
  "avatar": "//www.gravatar.com/avatar/4def9ae5b32502d59a4c874997490932?s=200&r=pg&d=mm",
  "user": "5f5b77df8872384b0db61113"
}

```

Status: 200 OK Time: 373 ms Size: 476 B Save Respn

## 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