

## Module 5: REST APIs And GraphQL

---

### Demo Document 2

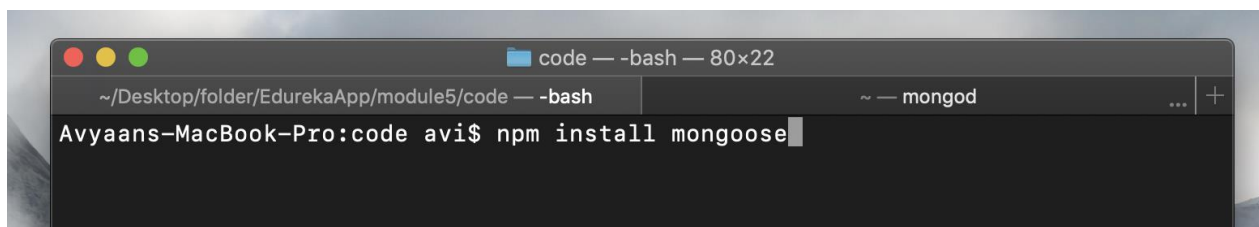
edureka!

**edureka!**

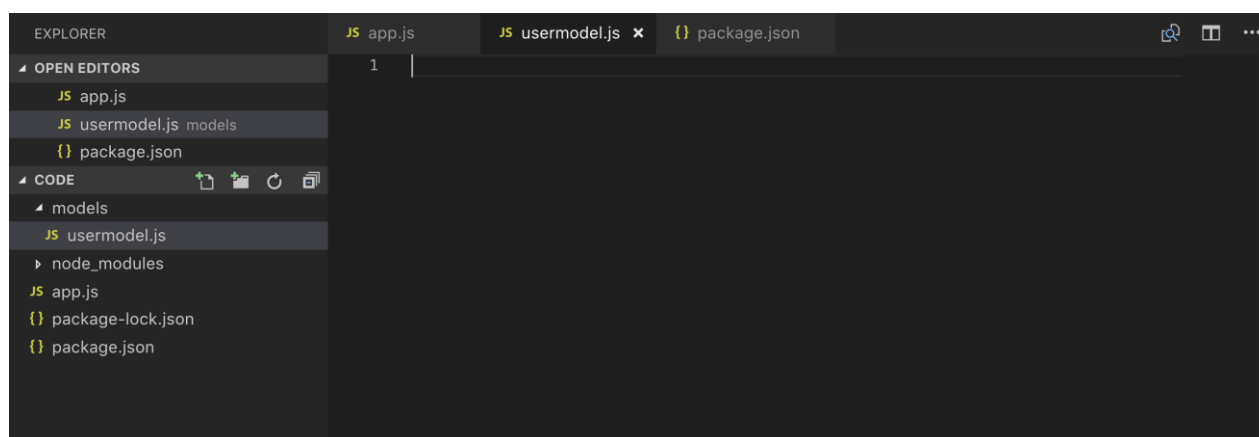
© Brain4ce Education Solutions Pvt. Ltd.

## Working of mongoose API

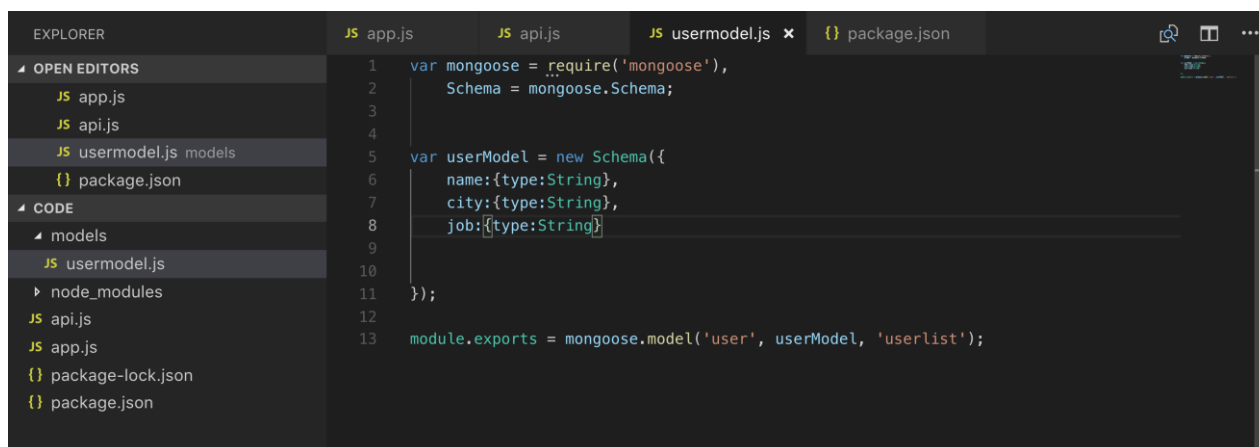
**Step 1:** In continuation with the demo 1. We will integrate mongoose to it, to maintain the schema. For that let us, install mongoose locally in folder using ‘npm install mongoose’



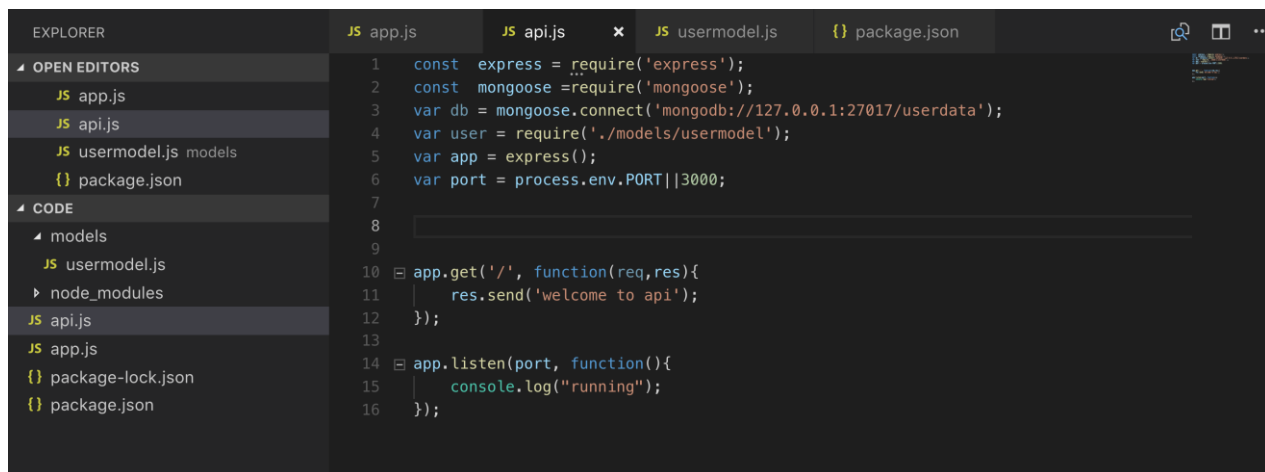
**Step 2:** First we must create one model file to design the schema. We have done it by creating a folder by the name of models, under which we have a file named userModel.js.



**Step 3:** In model file, define the schema with their data type and exports it. So that the APIs can consume it



**Step 4:** Introduce mongoose in the api file and connect it to the database using mongoose.



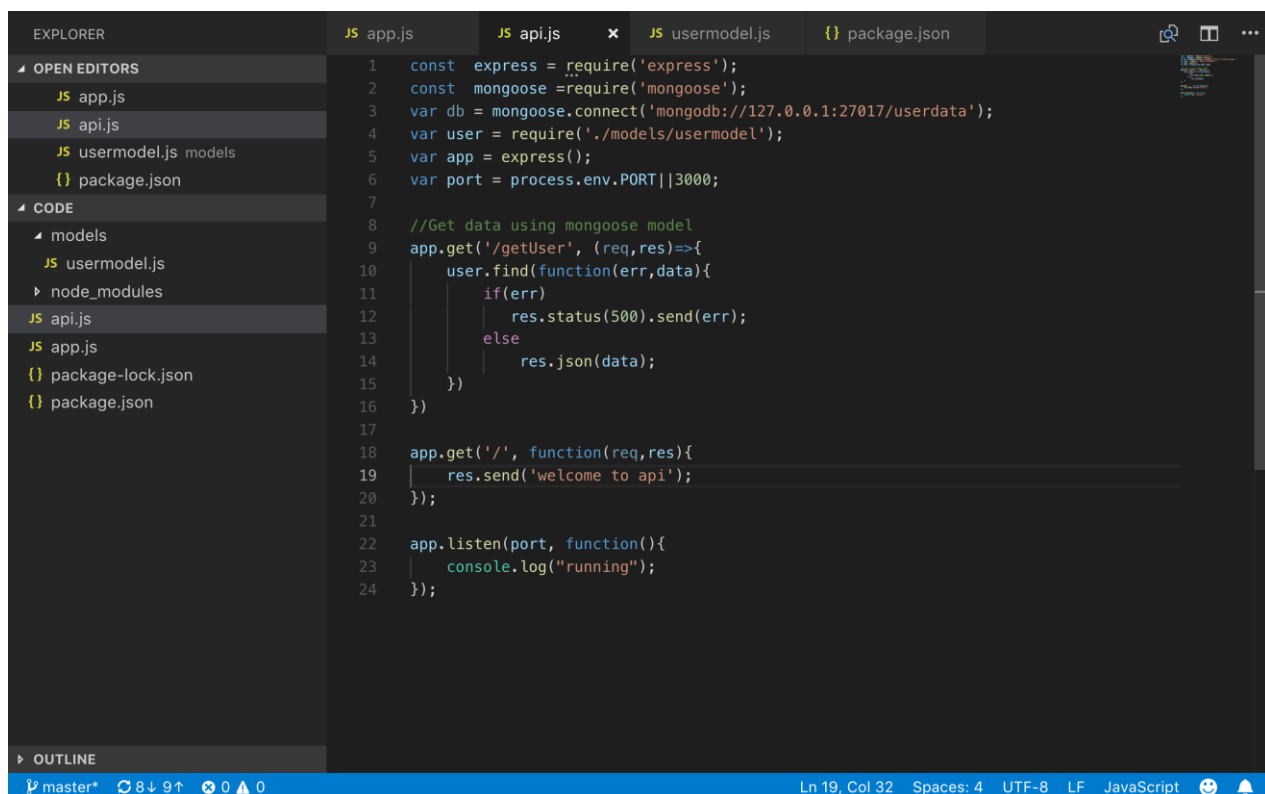
The screenshot shows the VS Code editor with the following files open in the Explorer: app.js, api.js, userModel.js, and package.json. The CODE pane shows the api.js file with the following code:

```

1  const express = require('express');
2  const mongoose = require('mongoose');
3  var db = mongoose.connect('mongodb://127.0.0.1:27017/userdata');
4  var user = require('./models/usermodel');
5  var app = express();
6  var port = process.env.PORT || 3000;
7
8
9
10 app.get('/', function(req,res){
11   res.send('welcome to api');
12 });
13
14 app.listen(port, function(){
15   console.log("running");
16 });

```

**Step 5:** GET request will be same as before but here we don't need to specify the database name or connection string as its maintained by Mongoose



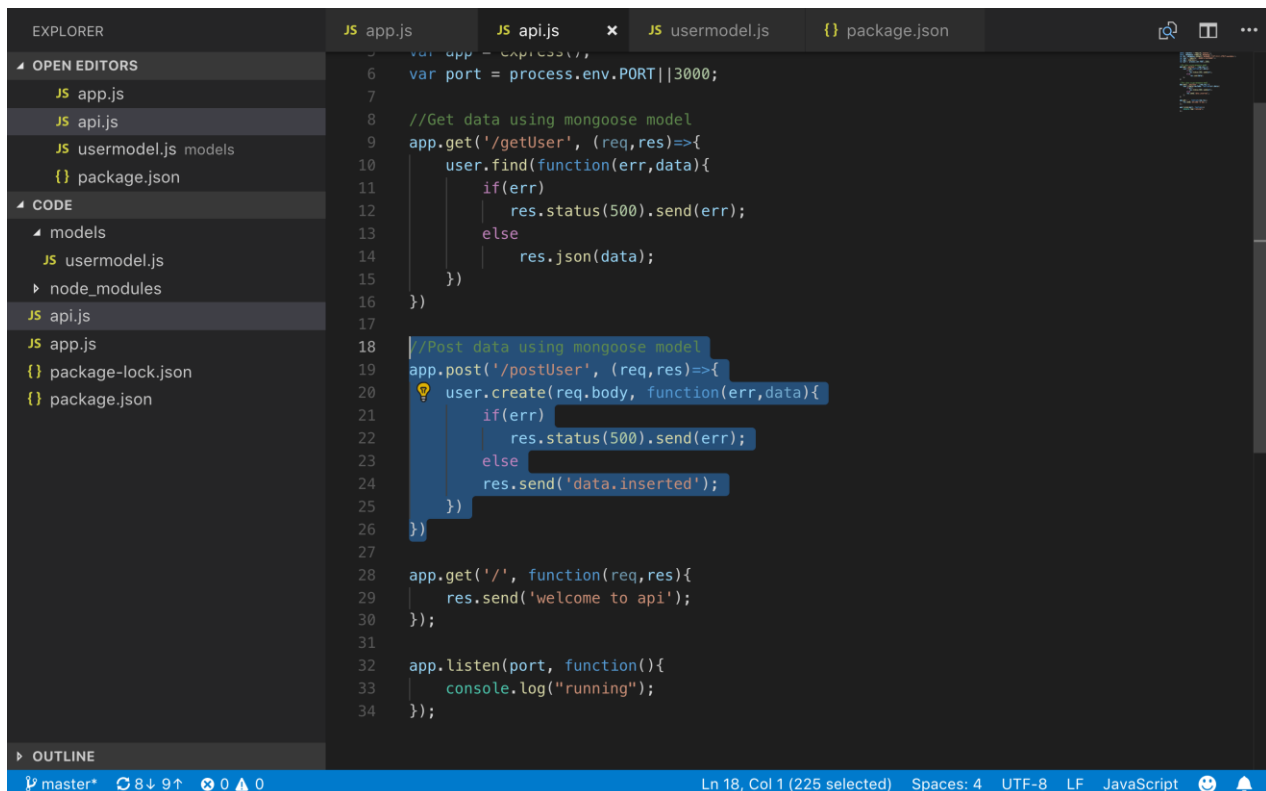
The screenshot shows the VS Code editor with the following files open in the Explorer: app.js, api.js, userModel.js, and package.json. The CODE pane shows the api.js file with the following code:

```

1  const express = require('express');
2  const mongoose = require('mongoose');
3  var db = mongoose.connect('mongodb://127.0.0.1:27017/userdata');
4  var user = require('./models/usermodel');
5  var app = express();
6  var port = process.env.PORT || 3000;
7
8  //Get data using mongoose model
9  app.get('/getUser', (req,res)=>{
10   user.find(function(err,data){
11     if(err)
12       res.status(500).send(err);
13     else
14       res.json(data);
15   })
16 })
17
18 app.get('/', function(req,res){
19   res.send('welcome to api');
20 });
21
22 app.listen(port, function(){
23   console.log("running");
24 });

```

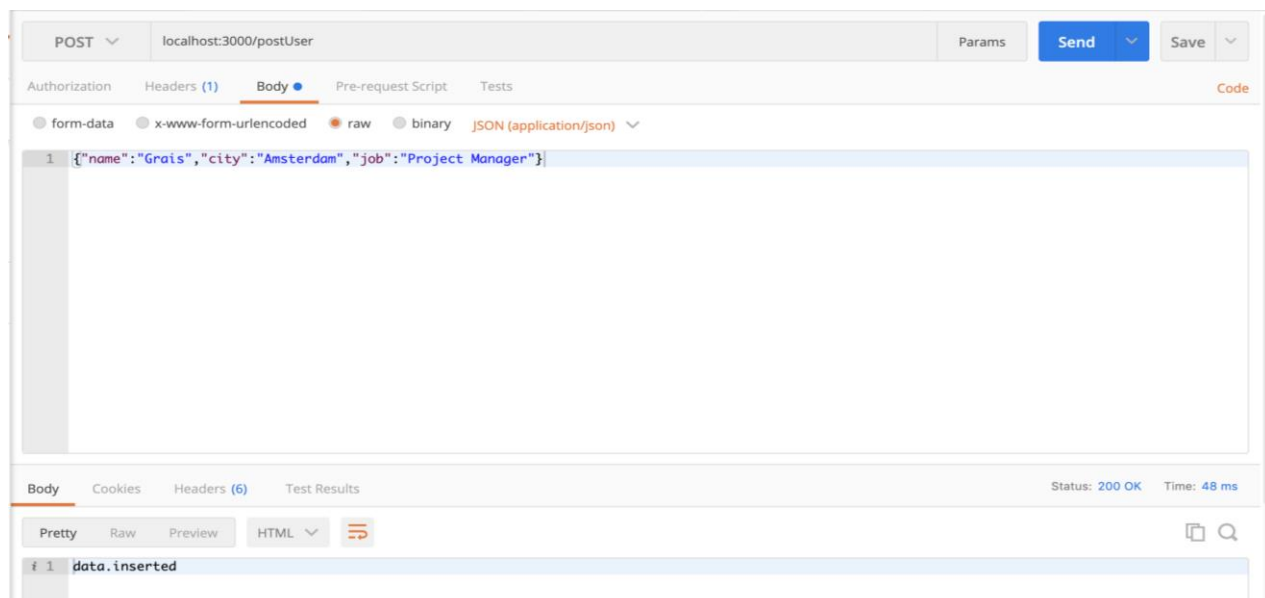
**Step 6:** For POST call, write the codes to insert the data



```

5  var app = express();
6  var port = process.env.PORT || 3000;
7
8  //Get data using mongoose model
9  app.get('/getUser', (req,res)=>{
10     user.find(function(err,data){
11         if(err)
12             res.status(500).send(err);
13         else
14             res.json(data);
15     })
16 })
17
18 //Post data using mongoose model
19 app.post('/postUser', (req,res)=>{
20     user.create(req.body, function(err,data){
21         if(err)
22             res.status(500).send(err);
23         else
24             res.send('data.inserted');
25     })
26 })
27
28 app.get('/', function(req,res){
29     res.send('welcome to api');
30 });
31
32 app.listen(port, function(){
33     console.log("running");
34 });
  
```

**Step 7:** When we test API with postman, we will provide the same result but the main difference is that now we have schema designed and connects with mongoose



**Step 8:** With mongoose PUT query remain same. We are not creating connection again and we don't need to define connection string

The screenshot shows the VS Code editor with the following files open in the Explorer: `app.js`, `api.js`, `package-lock.json`, `usermodel.js`, and `package.json`. The `api.js` file is active, showing the implementation of a PUT endpoint for updating a user. The code uses Mongoose to find a user by name and update their details (name, city, job) using the `findOneAndUpdate` method. The status code 500 is returned for errors, and the updated data is sent as the response.

```

18  })
19  })
20
21  //Post data using mongoose model
22  app.post('/postUser', (req,res)=>{
23    user.create(req.body, function(err,data){
24      if(err)
25        res.status(500).send(err);
26      else
27        res.send('data.inserted');
28    })
29  })
30
31  //update data using mongoose model
32  app.put('/update_user', (req,res)=>{
33    user.findOneAndUpdate({"name": req.body.name},{
34      $set:{
35        name:req.body.name,
36        city:req.body.city,
37        job:req.body.job
38      }
39    },{upsert:true}, function(err,data){
40      if(err)
41        res.status(500).send(err);
42      else
43        res.send(data)
44      })
45    })
46  })
47
48  app.get('/', function(req,res){

```

**Step 9:** With mongoose we can add DELETE query, by using ‘findOneAndDelete’ of mongodb.

The screenshot shows the VS Code editor with the following files open in the Explorer: `app.js`, `api.js`, `package-lock.json`, `usermodel.js`, and `package.json`. The `api.js` file is active, showing the implementation of a DELETE endpoint for deleting a user. The code uses Mongoose to find a user by name and delete it using the `findOneAndDelete` method. The status code 500 is returned for errors, and a success message is sent as the response. The rest of the `api.js` file, including the GET endpoint and the server listening logic, is also visible.

```

41    if(err)
42      res.status(500).send(err);
43    else
44      res.send(data)
45  })
46  })
47
48  //delete data using mongoose model
49  app.delete('/delete_user', (req,res)=>{
50    user.findOneAndDelete({"name": req.body.name},
51      function(err,data){
52        if(err)
53          res.status(500).send(err);
54        else
55          res.send('Deleted Successfully')
56      })
57  })
58
59
60  app.get('/', function(req,res){
61    res.send('welcome to api');
62  });
63
64  app.listen(port, function(){
65    console.log("running");
66  });

```

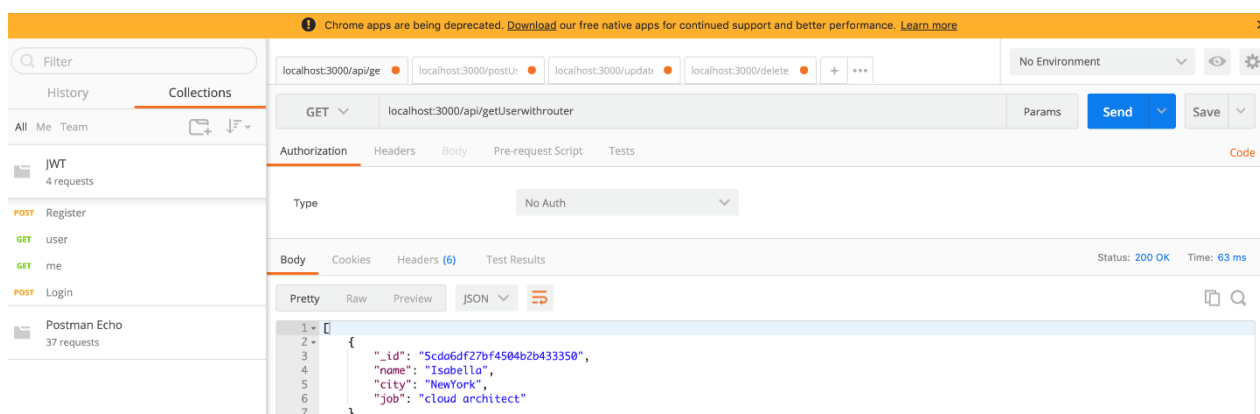
**Step 10:** Express have ‘express router’ to maintain rating for details or submenu in the application. Express Router is inbouded in express. We can use it as middleware with app.use. For that first, we need to define common router for the app and then define sub routes

```

4  var user = require('./models/usermodel');
5  var app = express();
6  var port = process.env.PORT || 3000;
7  const bodyParser = require('body-parser');
8  app.use(bodyParser.urlencoded({extended: true}));
9  app.use(bodyParser.json());
10 var commonRouter = express.Router();
11
12 //Get Api user Express Router
13 commonRouter.route('/getUserwithrouter')
14   .get(function(req, res){
15     user.find(function(err, data){
16       if(err){
17         res.status(500).send(err);
18       } else {
19         res.json(data);
20       }
21     });
22   });
23
24 //Get data using mongoose model
25 app.get('/getUser', (req, res) => {
26   user.find(function(err, data){
27     if(err){
28       res.status(500).send(err);
29     } else {
30       res.json(data);
31     }
32   })
33 });
34
35 //Post data using mongoose model
36 app.post('/postUser', (req, res) => {
37   user.create(req.body, function(err, data){
38     if(err){
39       res.status(500).send(err);
40     } else {
41       res.json(data);
42     }
43   })
44 });
45
46 //Delete data using mongoose model
47 app.delete('/deleteUser', (req, res) => {
48   user.remove(req.body, function(err, data){
49     if(err){
50       res.status(500).send(err);
51     } else {
52       res.json(data);
53     }
54   })
55 });
56
57 //Start the server
58 app.listen(port, () => {
59   console.log(`Server is running on port ${port}`);
60 });

```

**Step 11:** With common router we can define all CRUD operation, but the API linked with it will change like ‘/api/getuserwithrouter’ as /api is common router.



Thus, we have successfully performed CRUD Operation in Moongoose as well