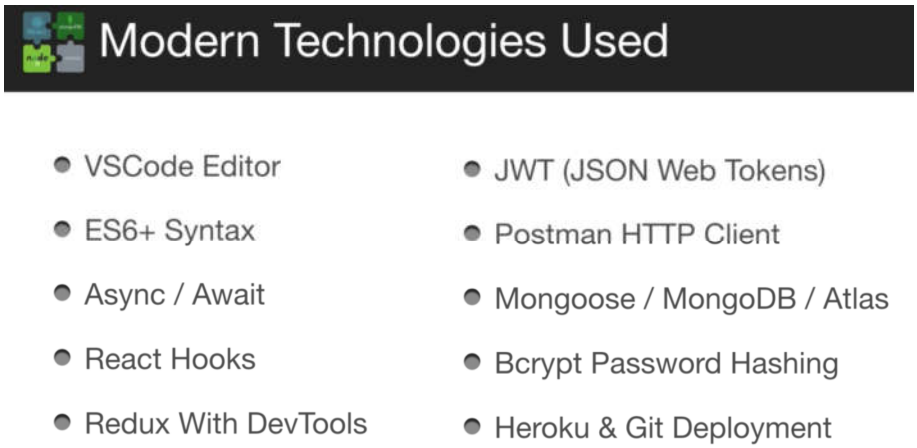
MERN STACK SOCIAL NETWORK

Table of contents

1. Introduction



NodeJS, VSCode (with bracket pair colorizer, prettier, ES7 React/Redux/GQL snippets), git (git-scm.com), Postman, React DevTools and Redux DevTool (Google chrome extensions).

2. Express & MongoDB Setup

VSCode:
CLI=> touch .gitignore
(this commad will create a .gitignore file)
Write to .gitignore node_modules/
(this command will add all the files from node_modules folder to ignore list – git will not upload/download all these files)

0. How to create a new project? (in point of Node Package Manager - npm)

CLI=> npm init

1. How to install all the dependencies to project?

CLI=> npm install express express-validator bcryptjs config gravatar jsonwebtoken mongoose request

2. How to install DEV dependencies to project?

CLI=> npm install -D nodemon concurrently

3. Create a MAIN file (entry point – server.js) with a test route

server.js

```
//Import EXPRESS
const express = require('express');
//Create a back-end server - APP
const app = express();
//Set up a port for server
const PORT = process.env.PORT || 5000;
//Set up a server to listen a port. Log to console string with information about port
app.listen(PORT, () => console.log(`Server is started on port ${PORT}`));
//Set up 1st test route: GET, to root folder, response will be aq string with a mesage
app.get('/', (req, res) => res.send('API running ...'));
```

4. Set up scripts to start a server

file package.json=>

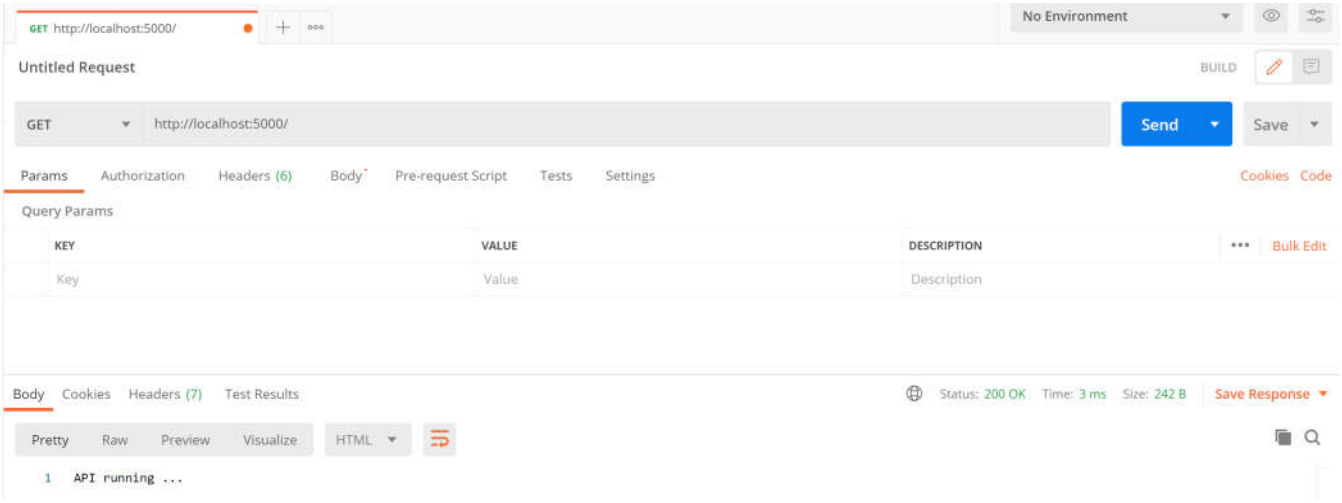
server.js | package.json

```
{
  "name": "devconnector",
  "version": "1.0.0",
  "description": "Social network",
  "main": "server.js",
  Debug
  "scripts": {
    "start": "node server",
    "server": "nodemon server"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "config": "^3.3.3",
    "express": "^4.17.1",
    "express-validator": "^6.9.2",
    "gravatar": "^1.8.1",
    "jsonwebtoken": "^8.5.1",
    "mongoose": "^5.11.17",
```

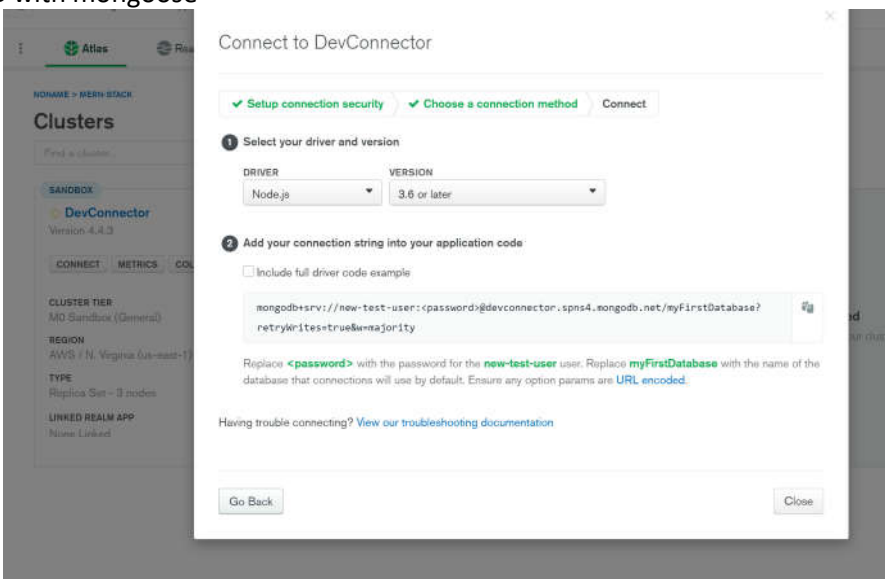
5. How to start a server?

CLI=> npm run server

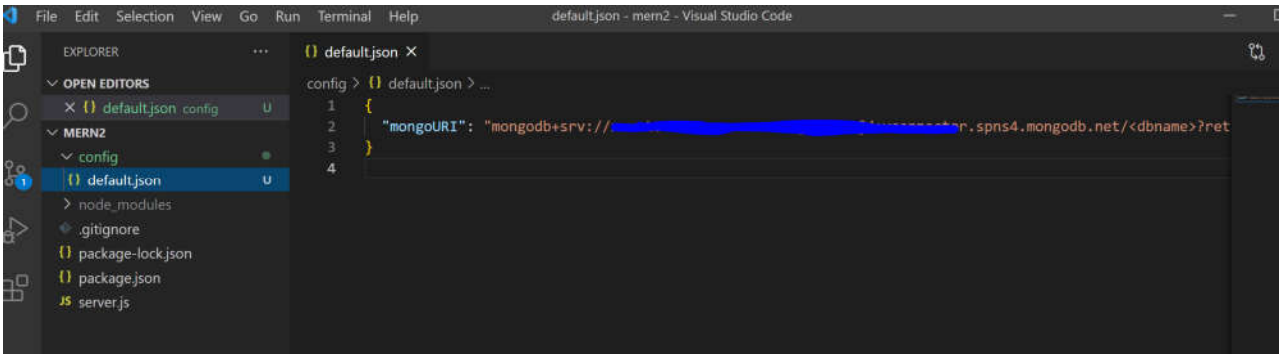
6. Check a test route with a Postman:



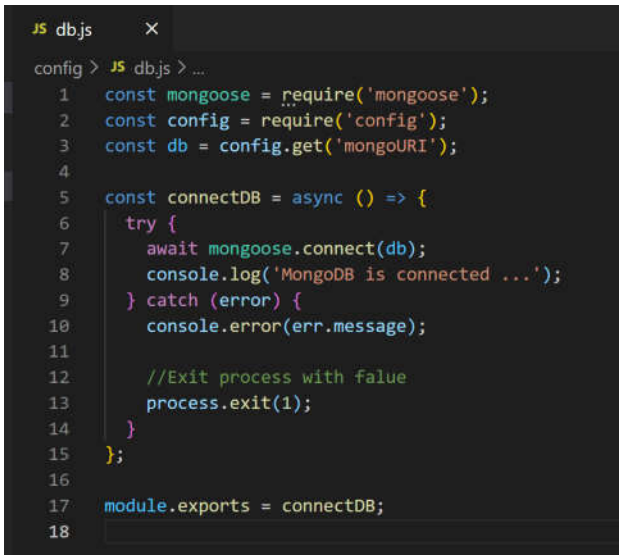
2.3 Connecting to MongoDB with mongoose



1. Create a folder – config
CLI=> mkdir config
CLI=> cd config
CLI=> touch default.json



2. Create a file db.js



server.js

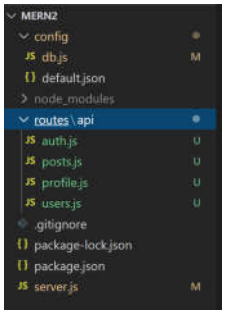
```
//Import EXPRESS
const express = require('express');
//Import a function to connect to DB
const connectDB = require('./config/db.js');
//Create a back-end server - APP
const app = express();
//Calling a function to connect to DB
connectDB();
//Set up a port for server
const PORT = process.env.PORT || 5000;
//Set up a server to listen a port. Log to console string with information about port
app.listen(PORT, () => console.log(`Server is started on port ${PORT}`));
//Set up 1st test route: GET, to root folder, response will be aq string with a mesage
app.get('/', (req, res) => res.send('API running ...'));
```

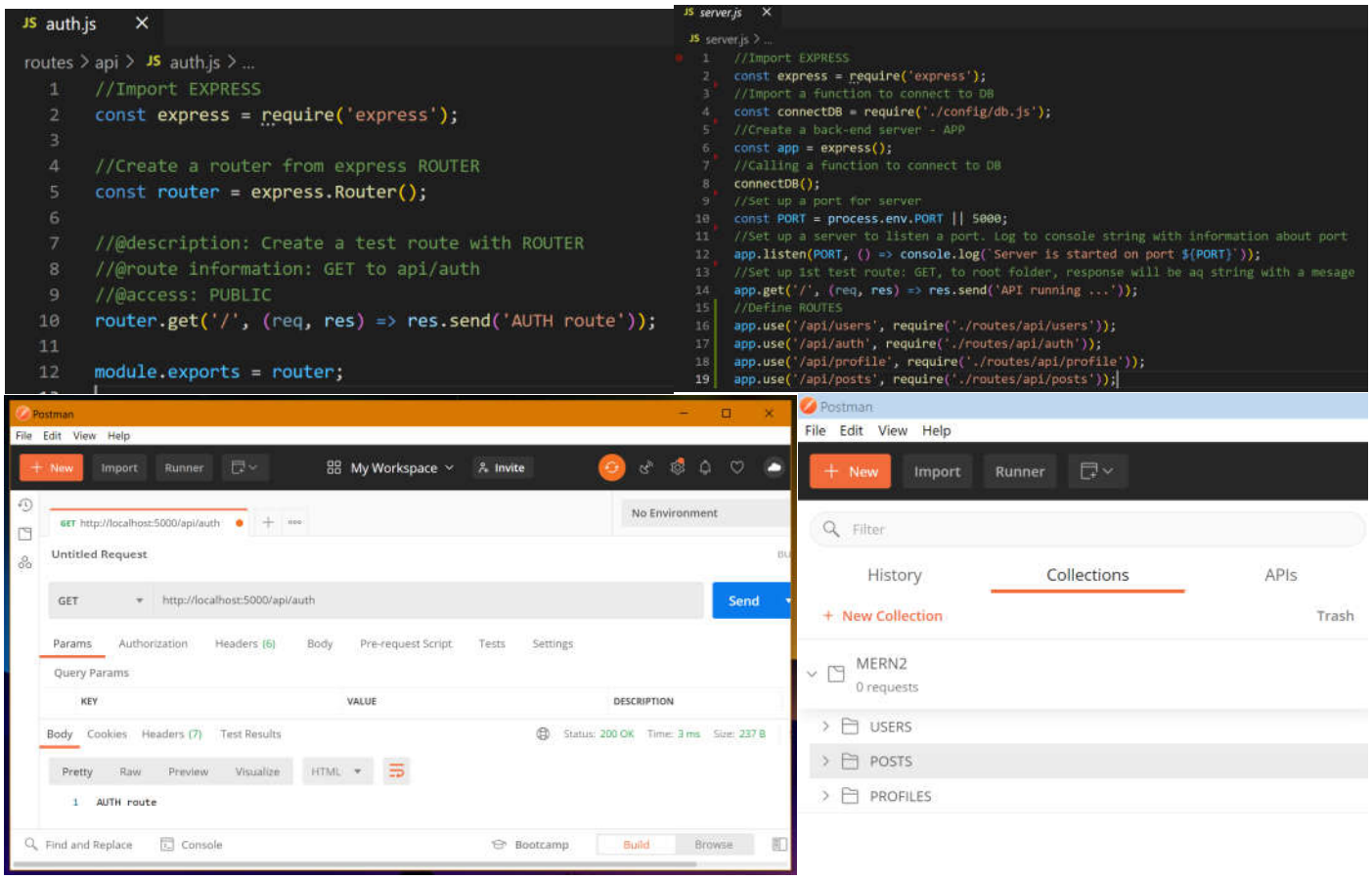
db.js

```
const mongoose = require('mongoose');
const config = require('config');
const db = config.get('mongoURI');
const connectDB = async () => {
  try {
    await mongoose.connect(db, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log('MongoDB is connected ...');
  } catch (error) {
    console.error(err.message);
    //Exit process with falue
    process.exit(1);
  }
};
module.exports = connectDB;
```

```
stslon@DESKTOP-06BF17J MINGW64 /f/_React/_files/mern2 (master)
$ npm run server

> devconnector@1.0.0 server F:\_!_files\mern2
> nodemon server

[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
(node:9068) Warning: Accessing non-existent property 'MongoError' of
 dependency
(Use `node --trace-warnings ...` to show where the warning was creat
Server is started on port 5000
(node:9068) DeprecationWarning: Listening to events on the Db class
 removed in the next major version.
MongoDB is connected ...
```

3. Create a folder for store all the routes – routes/api – with next files: users.js, auth.js, profile.js, posts.js

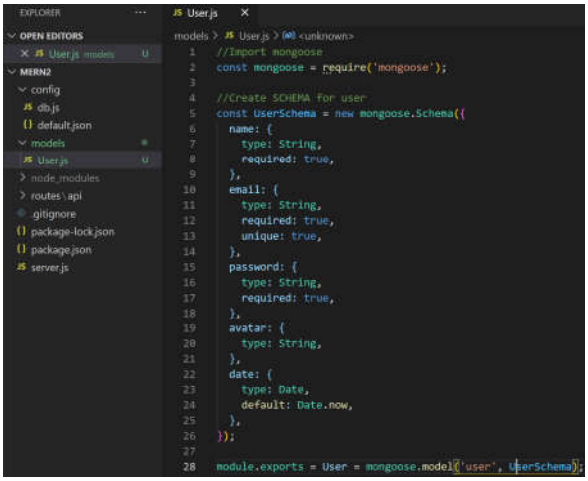



auth.js
routes > api > auth.js > ...
//Import EXPRESS
const express = require('express');
//Create a router from express ROUTER
const router = express.Router();
//@description: Create a test route with ROUTER
//@route information: GET to api/auth
//@access: PUBLIC
router.get('/', (req, res) => res.send('AUTH route'));
module.exports = router;
server.js
//Import EXPRESS
const express = require('express');
//Import a function to connect to DB
const connectDB = require('./config/db.js');
//Create a back-end server - APP
const app = express();
//Calling a function to connect to DB
connectDB();
//Set up a port for server
const PORT = process.env.PORT || 5000;
//Set up a server to listen a port. Log to console string with information about port
app.listen(PORT, () => console.log(`Server is started on port ${PORT}`));
//Set up 1st test route: GET, to root folder, response will be aq string with a mesage
app.get('/', (req, res) => res.send('API running ...'));
//Define ROUTES
app.use('/api/users', require('./routes/api/users'));
app.use('/api/auth', require('./routes/api/auth'));
app.use('/api/profile', require('./routes/api/profile'));
app.use('/api/posts', require('./routes/api/posts'));
Postman
File Edit View Help
New Import Runner
My Workspace
Invite
No Environment
GET http://localhost:5000/api/auth
Untitled Request
GET
http://localhost:5000/api/auth
Send
Params Authorization Headers (6) Body Pre-request Script Tests Settings
Query Params
KEY VALUE DESCRIPTION
Body Cookies Headers (7) Test Results
Status: 200 OK Time: 3 ms Size: 237 B
Pretty Raw Preview Visualize HTML
1 AUTH route
Find and Replace Console Bootcamp Build Browse
Postman
File Edit View Help
New Import Runner
Filter
History Collections APIs
New Collection Trash
MERN2
0 requests
USERS
POSTS
PROFILES

3. User API Routes & JWT Authentication

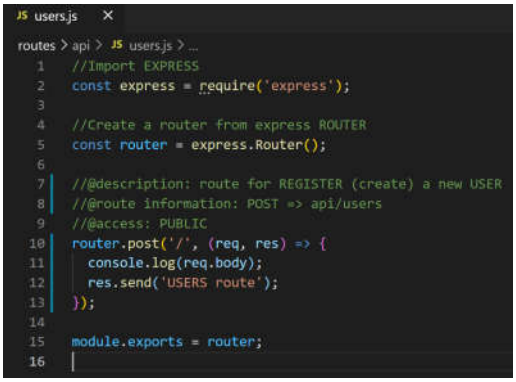
3.1 Creating The User Model (with schema for database)

3.1.1 Create a folder MODELS and inside it a file User.js

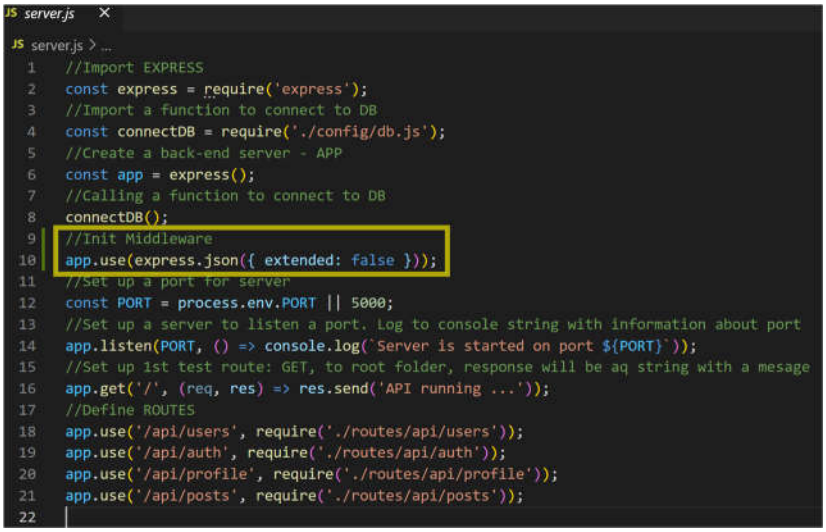


3.2 Request & Body Validation

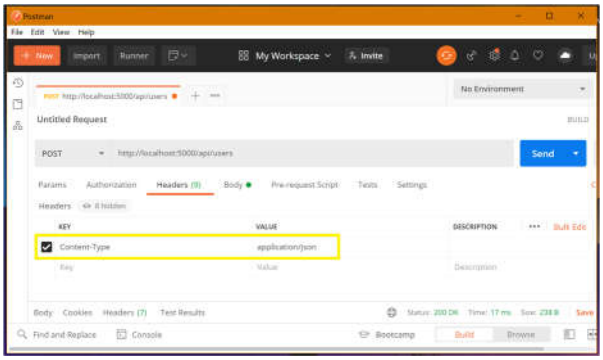
3.2.1 Make some changes at route users.js

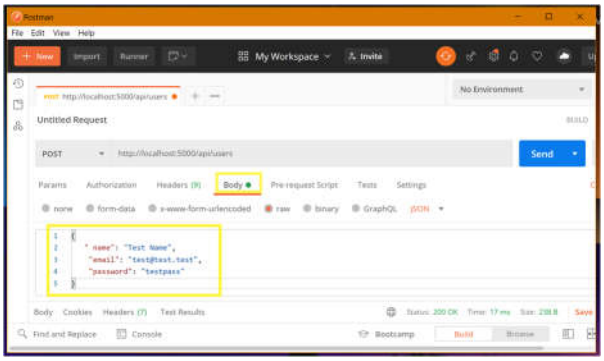


3.2.2 Init Middleware function at server.js (to be able work with bodypharser)



3.2.3 Make some changes at Postman for work with requests





```
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
(node:12624) Warning: Accessing non-existent property 'MongoError' of mod
(Use `node --trace-warnings ...` to show where the warning was created)
Server is started on port 5000
(node:12624) DeprecationWarning: Listening to events on the Db class has
MongoDB is connected ...
{ ' name': 'Test Name', email: 'test@test.test', password: 'testpass' }
```

3.2.4 Validation data from request with express-validator

Import *check* and *validationResult* to route users.js from express-validator/check

users.js

routes > api > users.js > ...

```
1  //Import EXPRESS
2  const express = require('express');
3
4  //Create a router from express ROUTER
5  const router = express.Router();
6
7  //Import check and validationResult
8  const { check, validationResult } = require('express-validator/check');
9
10 //@description: route for REGISTER (create) a new USER
11 //@route information: POST => api/users
12 //@access: PUBLIC
13 router.post('/', (req, res) => {
14   console.log(req.body);
15   res.send('USERS route');
16 });
17
18 module.exports = router;
```

Add checking at users.js ()

```
routes > api > users.js > router.post('/') callback
1  //Import EXPRESS
2  const express = require('express');
3  //Create a router from express ROUTER
4  const router = express.Router();
5  //Import check and validationResult
6  const { check, validationResult } = require('express-validator/check');
7  //@description: route for REGISTER (create) a new USER
8  //@route information: POST => api/users
9  //@access: PUBLIC
10 router.post(
11   '/',
12   [
13     check('name', 'Please check name and try again')
14       .isLength({
15         min: 6,
16       })
17       .isString(),
18     check('email', 'Please check email and try again').isEmail(),
19     check('password', 'Please check password and try again').isLength({
20       min: 6,
21     }),
22   ],
23   (req, res) => {
24     const errors = validationResult(req);
25     if (!errors.isEmpty()) {
26       return res.status(400).json({ errors: errors.array() });
27     }
28     res.send('USERS route');
29   }
30 );
31 module.exports = router;
```

POST http://localhost:5000/api/users

Untitled Request

POST http://localhost:5000/api/users

Params Authorization Headers (9) Body Pre-request Script

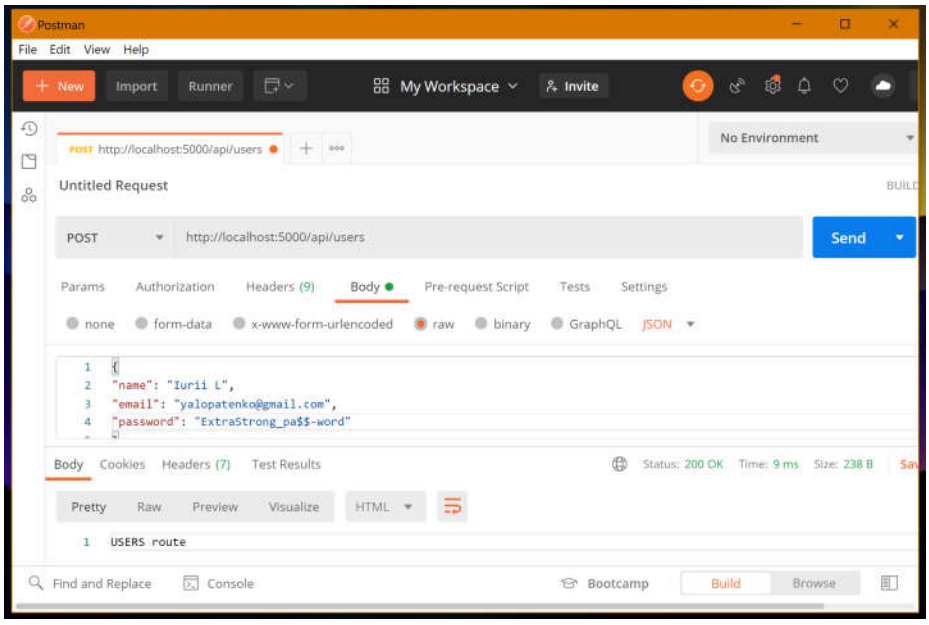
none form-data x-www-form-urlencoded raw binary

```
1 {
2
3 }
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "errors": [
        {
            "msg": "Please check name and try again",
            "param": "name",
            "location": "body"
        },
        {
            "msg": "Please check name and try again",
            "param": "name",
            "location": "body"
        },
        {
            "msg": "Please check email and try again",
            "param": "email",
            "location": "body"
        },
        {
            "msg": "Please check password and try again",
            "param": "password",
            "location": "body"
        }
    ]
}
```



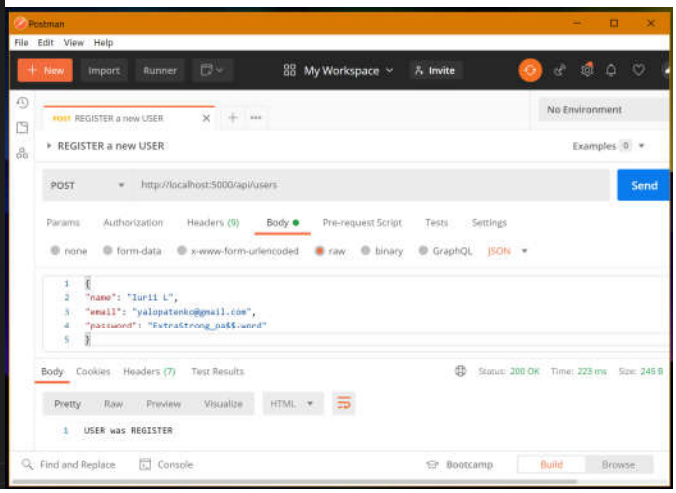
3.3 User REGISTRATION

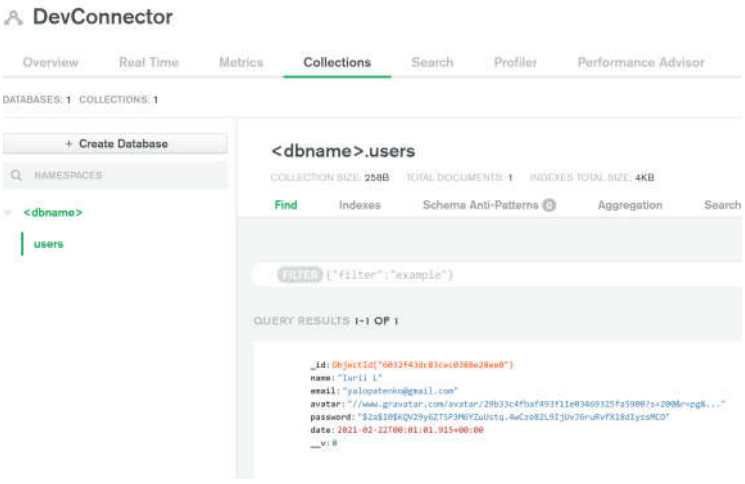
3.3.1 Add all the logic to route users.js

users.js

routes > api > users.js > ...

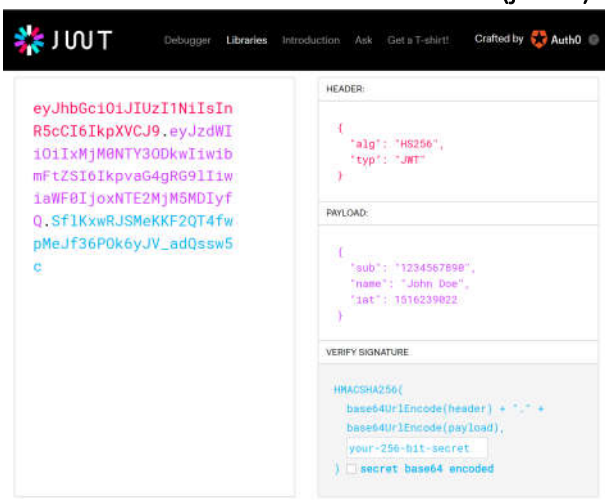
```
//import check and validationResult
const { check, validationResult } = require('express-validator');

//@description: route for REGISTER (create) a new USER
//@route information: POST => api/users
//@access: PUBLIC
router.post('/',[
    check('name', 'Please check name and try again').isLength({ min: 6 }).isString(),
    check('email', 'Please check email and try again').isEmail(),
    check('password', 'Please check password and try again').isLength({ min: 6,}),],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
    const { name, email, password } = req.body;
    try {
      //Check if in database exists a user with the same email (email should be unique)
      let user = await User.findOne({ email });
      if (user) {
        return res.status(400).json({ errors: [{ msg: 'User already exists' }] });
      }
      //Get users gravatar
      const avatar = gravatar.url(email, { s: '200', r: 'pg', d: 'mm' });
      //CREATE a new USER
      user = new User({ name, email, avatar, password });
      //Encrypt password with bcrypt
      //1. Create a salt
      const salt = await bcrypt.genSalt(10);
      //2. Change a password to a hash
      user.password = await bcrypt.hash(password, salt);
      //SAVE a new USER
      await user.save();
      //Return jsonwebtoken - TODO!!!
      res.send('USER is REGISTERED');
    } catch (err) {
      console.error(err.message);
      res.status(500).send('Server ERROR !!!');
    }
  }
);
module.exports = router;
```





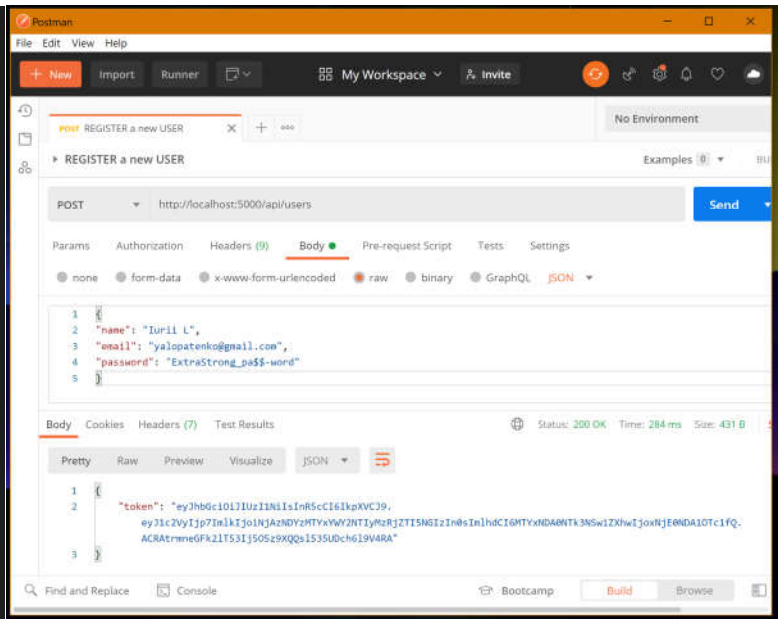
3.4 Implementing JWT

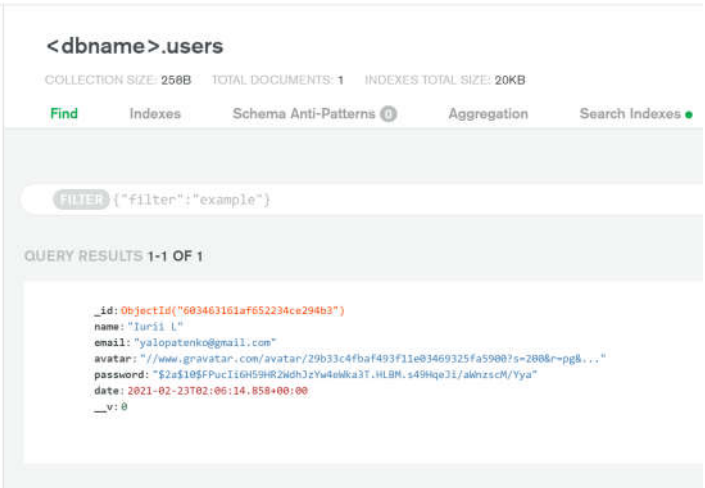
3.4.1 General information about JWT (jwt.io)



3.4.2 Implementing JWT

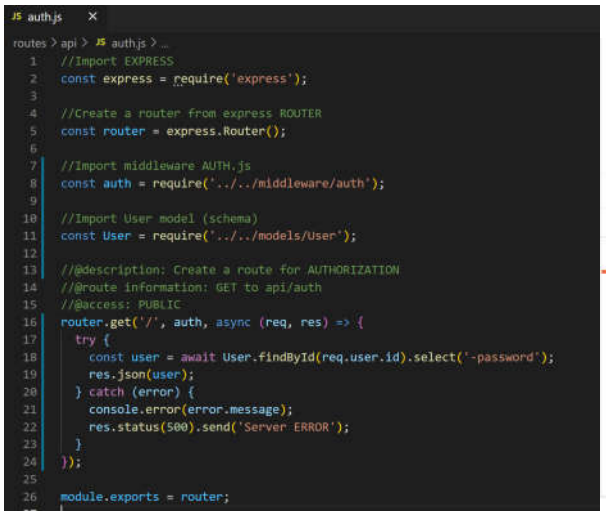
```
//Return jsonwebtoken
const payload = {
  user: {
    id: user.id,
  },
};
jwt.sign(
  payload,
  config.get('jwtSecret'),
  { expiresIn: 360000 },
  (error, token) => {
    if (error) {
      throw error;
    } else {
      res.json({ token });
    }
  }
);
```



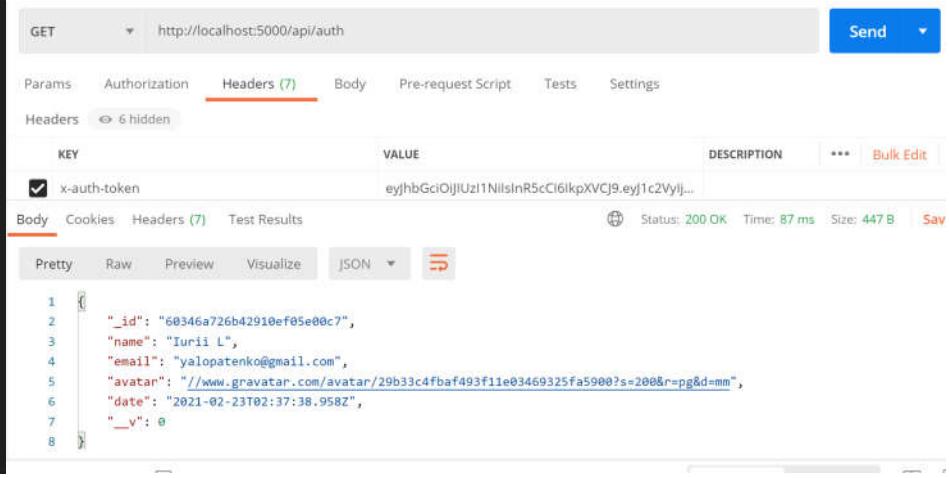


3.5 Custom Auth Middleware & JWT Verify

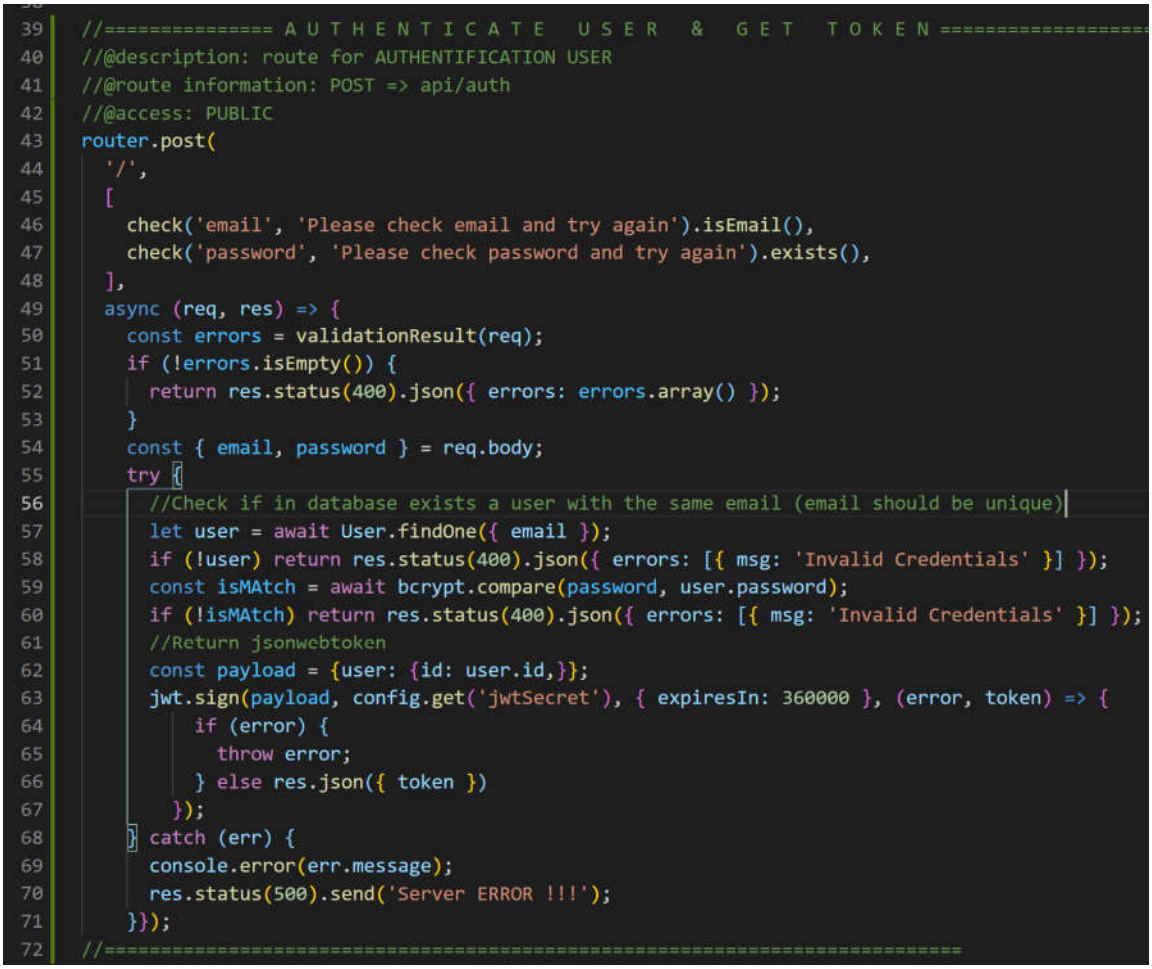
3.5.1 Create a MIDDLEWARE function auth.js



```
routes > api > auth.js > ...
1  //Import EXPRESS
2  const express = require('express');
3
4  //Create a router from express ROUTER
5  const router = express.Router();
6
7  //Import middleware AUTH.js
8  const auth = require('../../middleware/auth');
9
10 //Import User model (schema)
11 const User = require('../../models/User');
12
13 //@description: Create a route for AUTHORIZATION
14 //@route information: GET to api/auth
15 //@access: PUBLIC
16 router.get('/', auth, async (req, res) => {
17   try {
18     const user = await User.findById(req.user.id).select('-password');
19     res.json(user);
20   } catch (error) {
21     console.error(error.message);
22     res.status(500).send('Server ERROR');
23   }
24 });
25
26 module.exports = router;
```



3.6 User Authentication and Login Route



```
39 //============== AUTHENTICATE USER & GET TOKEN ==================
40 //@description: route for AUTHENTIFICATION USER
41 //@route information: POST => api/auth
42 //@access: PUBLIC
43 router.post(
44   '/',
45   [
46     check('email', 'Please check email and try again').isEmail(),
47     check('password', 'Please check password and try again').exists(),
48   ],
49   async (req, res) => {
50     const errors = validationResult(req);
51     if (!errors.isEmpty()) {
52       return res.status(400).json({ errors: errors.array() });
53     }
54     const { email, password } = req.body;
55     try {
56       //Check if in database exists a user with the same email (email should be unique)
57       let user = await User.findOne({ email });
58       if (!user) return res.status(400).json({ errors: [{ msg: 'Invalid Credentials' }] });
59       const isMAtch = await bcrypt.compare(password, user.password);
60       if (!isMAtch) return res.status(400).json({ errors: [{ msg: 'Invalid Credentials' }] });
61       //Return jsonwebtoken
62       const payload = {user: {id: user.id,}};
63       jwt.sign(payload, config.get('jwtSecret'), { expiresIn: 360000 }, (error, token) => {
64           if (error) {
65             throw error;
66           } else res.json({ token })
67         });
68     } catch (err) {
69       console.error(err.message);
70       res.status(500).send('Server ERROR !!!');
71     }});
72 //================================================================================
```