



MERN STACK SOCIAL NETWORK



Table of contents

1. Introduction.....	2
2. Express & MongoDB Setup.....	3
0. How to create a new project? (in point of Node Package Manager - npm)	3
1. How to install all the dependencies to project?	3
2. How to install DEV dependencies to project?	3
3. Create a MAIN file (entry point – server.js) with a test route.....	3
4. Set up scripts to start a server.....	3
5. How to start a server?	4
6. Check a test route with a Postman:	4
3. User API Routes & JWT Authentication.....	8
3.1 Creating The User Model (with schema for database)	8
3.1.1 Create a folder MODELS and inside it a file User.js.....	8
3.2 Request & Body Validation.....	8
3.2.1 Make some changes at route users.js	8
3.2.2 Init Middleware function at server.js (to be able work with bodyparser)	8
3.2.4 Validation data from request with express-validator	9

1. Introduction



- VSCode Editor
- ES6+ Syntax
- Async / Await
- React Hooks
- Redux With DevTools
- JWT (JSON Web Tokens)
- Postman HTTP Client
- Mongoose / MongoDB / Atlas
- Bcrypt Password Hashing
- Heroku & Git Deployment

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 command 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

```
JS server.js
JS server.js > ...
1 //Import EXPRESS
2 const express = require('express');
3 //Create a back-end server - APP
4 const app = express();
5 //Set up a port for server
6 const PORT = process.env.PORT || 5000;
7 //Set up a server to listen a port. Log to console string with information about port
8 app.listen(PORT, () => console.log(`Server is started on port ${PORT}`));
9 //Set up 1st test route: GET, to root folder, response will be aq string with a message
10 app.get('/', (req, res) => res.send('API running ...'));
11
```

4. Set up scripts to start a server

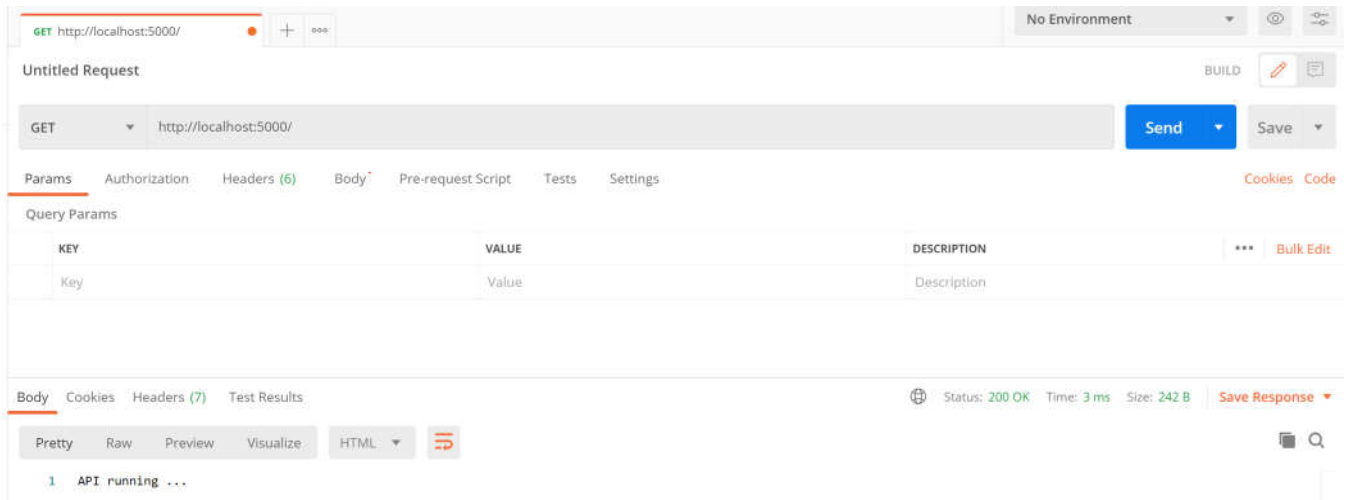
file package.json=>

```
JS server.js {} package.json X
{} package.json > ...
1 {
2   "name": "devconnector",
3   "version": "1.0.0",
4   "description": "Social network",
5   "main": "server.js",
6   "scripts": {
7     "start": "node server",
8     "server": "nodemon server"
9   },
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "bcryptjs": "^2.4.3",
14    "config": "^3.3.3",
15    "express": "^4.17.1",
16    "express-validator": "^6.9.2",
17    "gravatar": "^1.8.1",
18    "jsonwebtoken": "^8.5.1",
19    "mongoose": "^5.11.17",
```

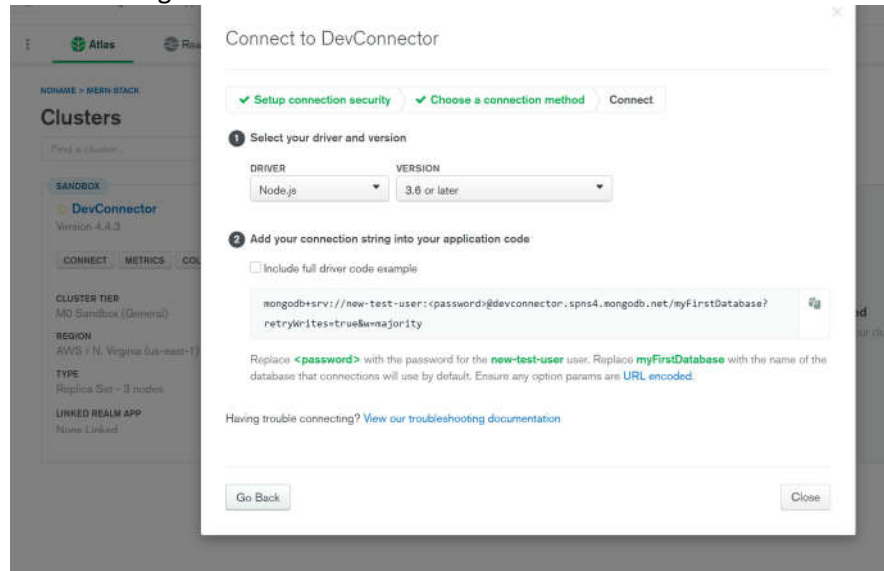
5. How to start a server?

CLI=> npm run server

6. Check a test route with a Postman:



2.3 Connecting to MongoDB with mongoose



1. Create a folder – config

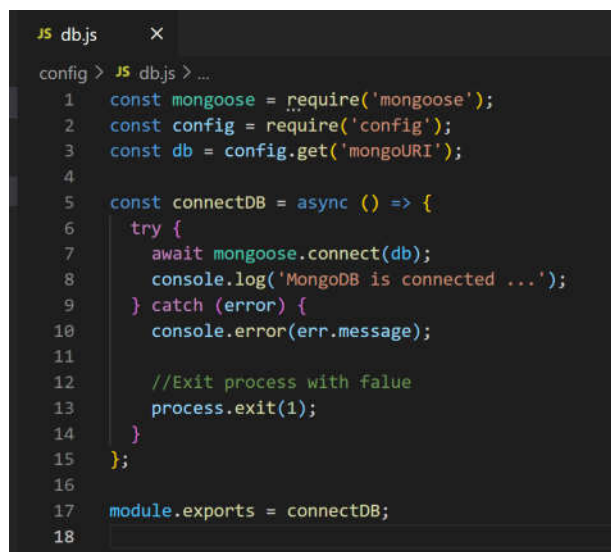
CLI=> mkdir config

CLI=> cd config

CLI=> touch default.json



2. Create a file db.js



```

JS server.js X
JS server.js > ...
1 //Import EXPRESS
2 const express = require('express');
3 //Import a function to connect to DB
4 const connectDB = require('./config/db.js');
5 //Create a back-end server - APP
6 const app = express();
7 //Calling a function to connect to DB
8 connectDB();
9 //Set up a port for server
10 const PORT = process.env.PORT || 5000;
11 //Set up a server to listen a port. Log to console string with information about port
12 app.listen(PORT, () => console.log('Server is started on port ${PORT}'));
13 //Set up 1st test route: GET, to root folder, response will be aq string with a message
14 app.get('/', (req, res) => res.send('API running ...'));
15

```

```

JS db.js X
config > JS db.js > ...
1 const mongoose = require('mongoose');
2 const config = require('config');
3 const db = config.get('mongoURI');
4 const connectDB = async () => {
5   try {
6     await mongoose.connect(db, {
7       useNewUrlParser: true,
8       useUnifiedTopology: true,
9     });
10    console.log('MongoDB is connected ...');
11  } catch (error) {
12    console.error(err.message);
13    //Exit process with false
14    process.exit(1);
15  }
16 };
17 module.exports = connectDB;
18

```

```

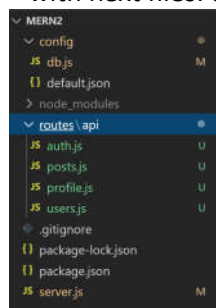
stslon@DESKTOP-06BF17J MINGW64 /f/_React/_files/mern2 (master)
$ npm run server

> devconnecton@1.0.0 server F:\_l_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



JS authjs

routes > api > JS authjs > ...
1 //Import EXPRESS
2 const express = require('express');
3
4 //Create a router from express ROUTER
5 const router = express.Router();
6
7 //@@description: Create a test route with ROUTER
8 //@@route information: GET to api/auth
9 //@@access: PUBLIC
10 router.get('/', (req, res) => res.send('AUTH route'));
11
12 module.exports = router;

JS server.js

JS server.js > ...
1 //Import EXPRESS
2 const express = require('express');
3 //Import a function to connect to DB
4 const connectDB = require('./config/db.js');
5 //Create a back-end server - APP
6 const app = express();
7 //Calling a function to connect to DB
8 connectDB();
9 //Set up a port for server
10 const PORT = process.env.PORT || 5000;
11 //Set up a server to listen a port. Log to console string with information about port
12 app.listen(PORT, () => console.log('Server is started on port \${PORT}'));
13 //Set up 1st test route: GET, to root folder, response will be aq string with a message
14 app.get('/', (req, res) => res.send('API running ...'));
15 //Define ROUTES
16 app.use('/api/users', require('./routes/api/users'));
17 app.use('/api/auth', require('./routes/api/auth'));
18 app.use('/api/profile', require('./routes/api/profile'));
19 app.use('/api/posts', require('./routes/api/posts'));

Postman

File Edit View Help

+ New Import Runner My Workspace Invite

GET http://localhost:5000/api/auth No Environment

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

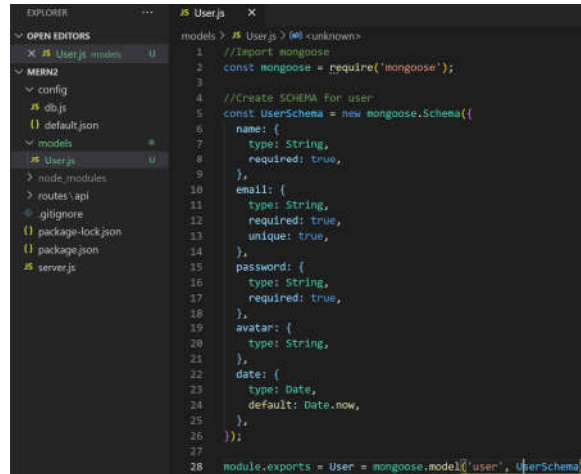
POSTS

PROFILES

3. User API Routes & JWT Authentication

3.1 Creating The User Model (with schema for database)

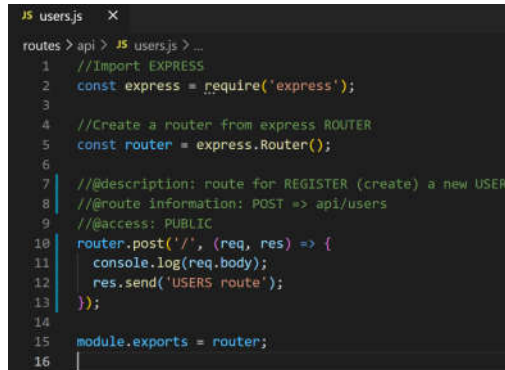
3.1.1 Create a folder MODELS and inside it a file User.js



```
1 //Import mongoose
2 const mongoose = require('mongoose');
3
4 //Create SCHEMA for user
5 const UserSchema = new mongoose.Schema({
6   name: {
7     type: String,
8     required: true,
9   },
10   email: {
11     type: String,
12     required: true,
13     unique: true,
14   },
15   password: {
16     type: String,
17     required: true,
18   },
19   avatar: {
20     type: String,
21   },
22   date: {
23     type: Date,
24     default: Date.now,
25   },
26 });
27
28 module.exports = User = mongoose.model('user', UserSchema);
```

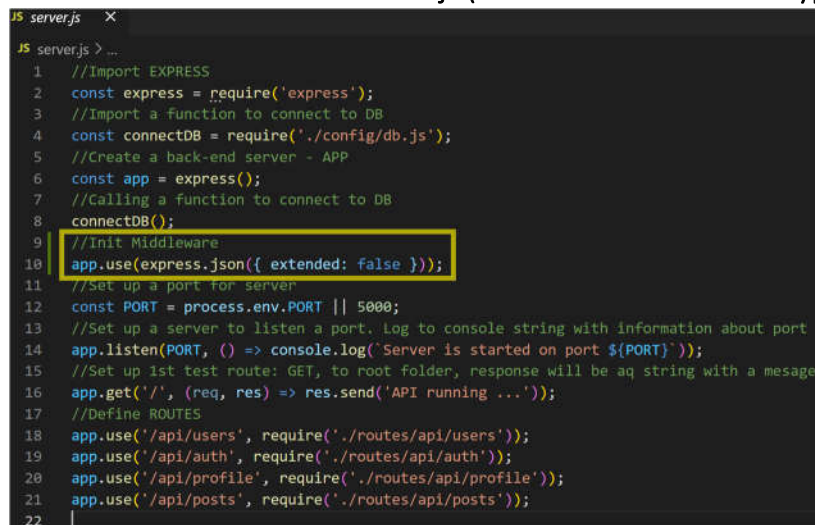
3.2 Request & Body Validation

3.2.1 Make some changes at route users.js



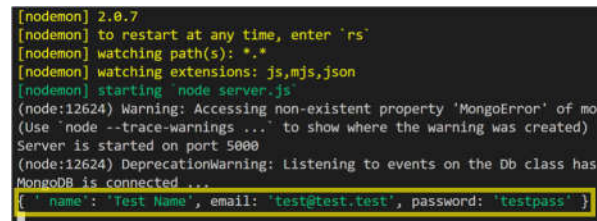
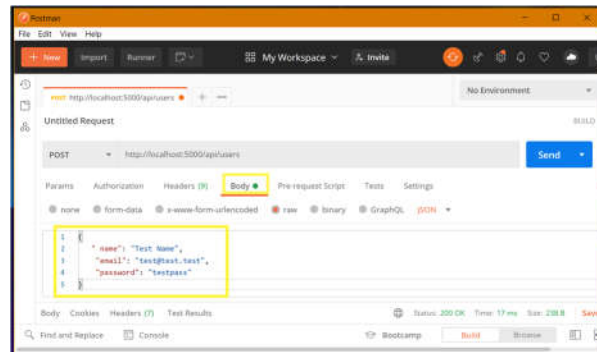
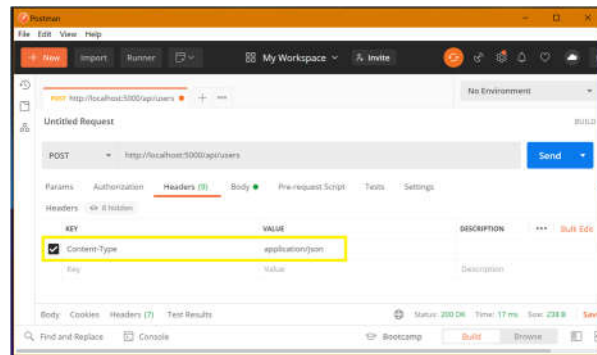
```
1 //Import EXPRESS
2 const express = require('express');
3
4 //Create a router from express ROUTER
5 const router = express.Router();
6
7 //@description: route for REGISTER (create) a new USER
8 //@route information: POST => api/users
9 //@access: PUBLIC
10 router.post('/', (req, res) => {
11   console.log(req.body);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

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



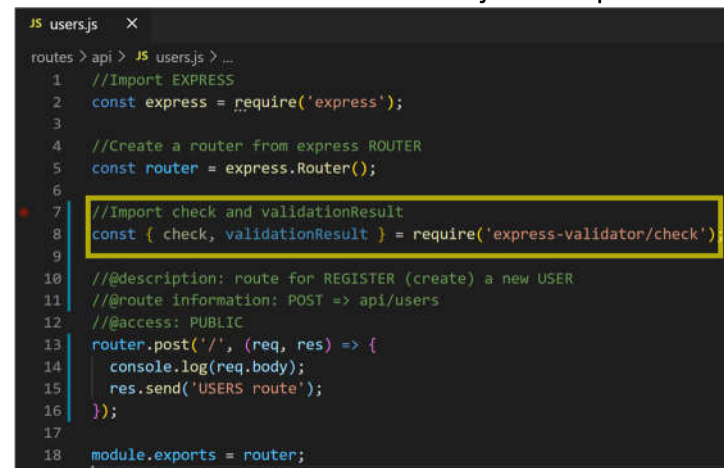
```
1 //Import EXPRESS
2 const express = require('express');
3 //Import a function to connect to DB
4 const connectDB = require('./config/db.js');
5 //Create a back-end server - APP
6 const app = express();
7 //Calling a function to connect to DB
8 connectDB();
9 //Init Middleware
10 app.use(express.json({ extended: false }));
11 //Set up a port for server
12 const PORT = process.env.PORT || 5000;
13 //Set up a server to listen a port. Log to console string with information about port
14 app.listen(PORT, () => console.log('Server is started on port ${PORT}'));
15 //Set up 1st test route: GET, to root folder, response will be aq string with a message
16 app.get('/', (req, res) => res.send('API running ...'));
17 //Define ROUTES
18 app.use('/api/users', require('./routes/api/users'));
19 app.use('/api/auth', require('./routes/api/auth'));
20 app.use('/api/profile', require('./routes/api/profile'));
21 app.use('/api/posts', require('./routes/api/posts'));
22
```

3.2.3 Make some changes at Postman for work with requests



3.2.4 Validation data from request with express-validator

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



Add checking at users.js ()

```

JS users.js X
routes > api > JS 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

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```

2 {
3   "errors": [
4     {
5       "msg": "Please check name and try again",
6       "param": "name",
7       "location": "body"
8     },
9     {
10      "msg": "Please check name and try again",
11      "param": "name",
12      "location": "body"
13    },
14    {
15      "msg": "Please check email and try again",
16      "param": "email",
17      "location": "body"
18    },
19    {
20      "msg": "Please check password and try again",
21      "param": "password",
22      "location": "body"
23    }
24  ]

```

Postman

File Edit View Help

+ New Import Runner My Workspace Invite

POST http://localhost:5000/api/users No Environment

Untitled Request

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

Params Authorization Headers (9) Body Pre-request Script Tests Settings

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

```

1 {
2   "name": "Iurii L",
3   "email": "yalopatenko@gmail.com",
4   "password": "ExtraStrong_pa$$-word"

```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 9 ms Size: 238 B

Pretty Raw Preview Visualize HTML

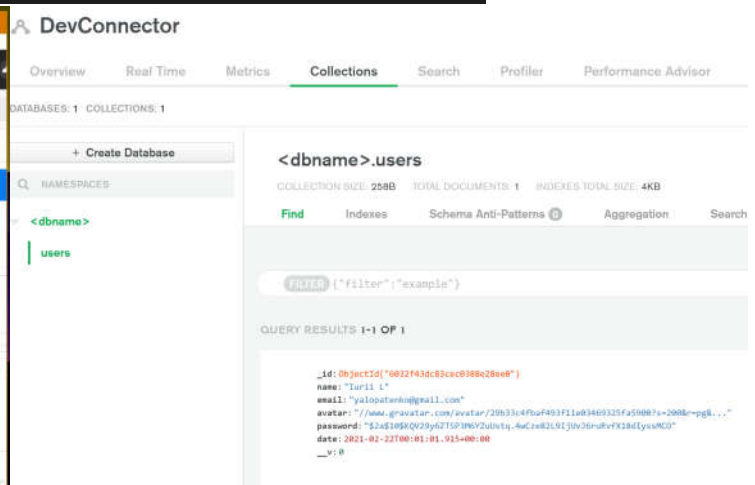
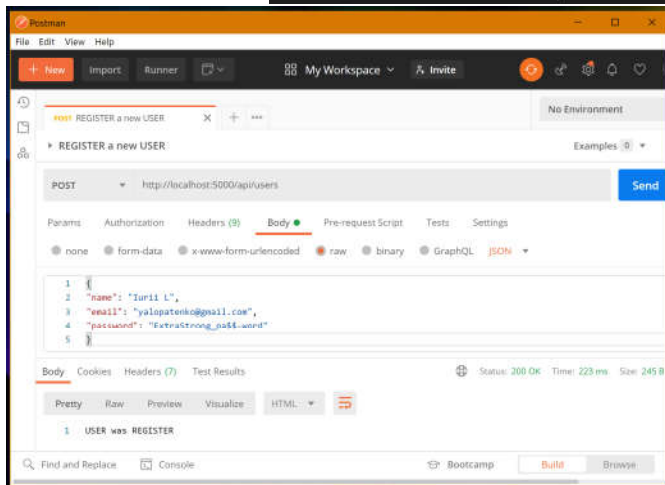
1 USERS route

Find and Replace Console Bootcamp Build Browse

3.3 User REGISTRATION

3.3.1 Add all the logic to route users.js

```
JS users.js
routes > api > .JS users.js > ...
16 //import check and validationResult
17 const { check, validationResult } = require('express-validator');
18
19 //@@description: route for REGISTER (create) a new USER
20 //@@route information: POST => api/users
21 //@@access: PUBLIC
22 router.post('/', [
23   check('name', 'Please check name and try again').isLength({ min: 6 }).isString(),
24   check('email', 'Please check email and try again').isEmail(),
25   check('password', 'Please check password and try again').isLength({ min: 6, }),
26 ], async (req, res) => {
27   const errors = validationResult(req);
28   if (!errors.isEmpty()) {
29     return res.status(400).json({ errors: errors.array() });
30   }
31   const { name, email, password } = req.body;
32   try {
33     //Check if in database exists a user with the same email (email should be unique)
34     let user = await User.findOne({ email });
35     if (user) {
36       return res.status(400).json({ errors: [{ msg: 'User already exists' }] });
37     }
38     //Get users gravatar
39     const avatar = gravatar.url(email, { s: '200', r: 'pg', d: 'mm' });
40     //CREATE a new USER
41     user = new User({ name, email, avatar, password });
42     //Encrypt password with bcrypt
43     //1. Create a salt
44     const salt = await bcrypt.genSalt(10);
45     //2. Change a password to a hash
46     user.password = await bcrypt.hash(password, salt);
47     //SAVE a new USER
48     await user.save();
49     //Return jsonwebtoken - TODO!!!
50     res.send('USER is REGISTERED');
51   } catch (err) {
52     console.error(err.message);
53     res.status(500).send('Server ERROR !!!');
54   }
55 }
56 );
57 module.exports = router;
```



3.4 Implementing JWT

3.4.1 General information about JWT (jwt.io)

The JWT.io interface displays a JWT token and its decoded components. The token is: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4uRG9lIiwiaWF0IjoxNTE2MzE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c`. The header is: `{ "alg": "HS256", "typ": "JWT" }`. The payload is: `{ "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }`. The verify signature section shows the HMACSHA256 algorithm and the secret key.

3.4.2 Implementing JWT

The code snippet shows the implementation of JWT in a Node.js application. It defines a `payload` object with `user` information, signs it using `jwt.sign` with a secret key and expiration time, and returns the token in the response. The Postman screenshot shows a `POST` request to `http://localhost:5000/api/users` with a JSON body containing user details. The response is a `200 OK` status with a JSON body containing the `token`.

The JWT token is: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2V5IjoiIjoiNjAzNDYzMjYxYmZrZjZlIiwiaWF0IjoxNTE2MzE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c`. The header is: `{ "alg": "HS256", "typ": "JWT" }`. The payload is: `{ "user": { "id": "603463161af652234ce294b3", "iat": 1614045975, "exp": 1614405975 } }`. The MongoDB query result shows a document in the `<dbname>.users` collection with fields: `_id`, `name`, `email`, `avatar`, `password`, `date`, and `_v`.