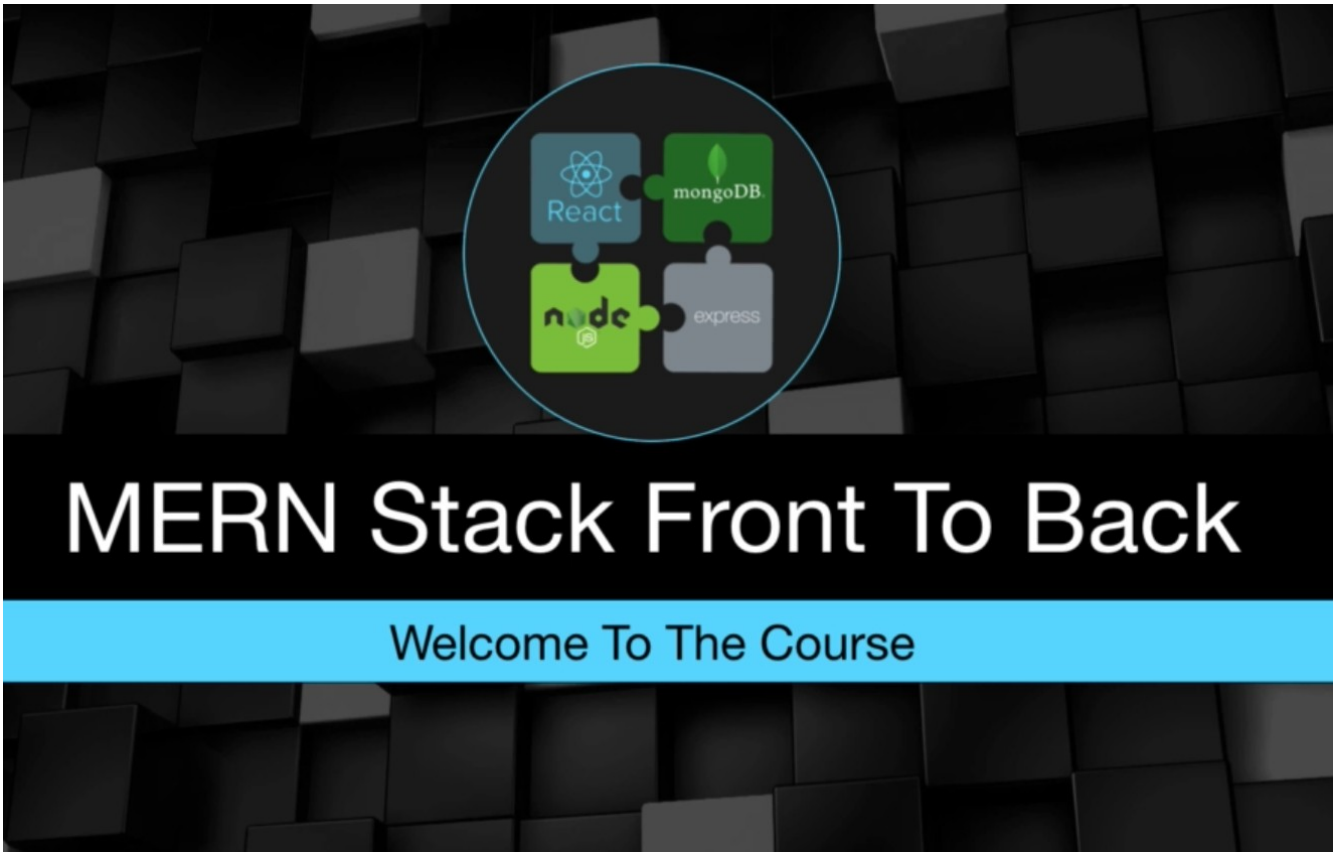
MERN Stack Front To Back: Full Stack React, Redux & Node.js (2020)



https://www.udemy.com/course/mern-stack-front-to-back/

by Brad Traversy

Table of Contents

0. 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

0.1 Environment & Setup

0.1.1 Node.js

Windows: https://nodejs.org/en/ - just download and install with GUI.

Linux:
CLI=> *sudo apt-get update*
CLI=> *sudo apt install curl build-essential*
CLI=> *curl -sL https://deb.nodesource.com/setup_14.x | sudo -E bash -*
CLI=> *sudo apt install -y nodejs*
CLI=> *node -v [=> v14.16.1]*

0.1.2 Visual Studio Code (VSC)

Windows: https://code.visualstudio.com/ - just download and install with GUI.

Linux:
CLI=> *sudo apt-get update*
CLI=> *sudo apt install code*

0.1.3 GIT

Windows: https://git-scm.com/ just download and install with GUI.

Linux: **GIT is a basic component**

0.1.4 Postman

Windows: https://www.postman.com/ - just download and install with GUI.

Linux: *https://www.postman.com/downloads/ - download archive tar.gz*
https://dl.pstmn.io/download/latest/linux64

go to download folder and unpack that file with command *tar xvzf [**PACKAGENAME**].tar.gz*

CLI=> *tar xvzf Postman-linux-x64-8.6.1.tar.gz*

go to folder and use shortcut

0.1.5 React Developer Tools chrome extension

0.1.6 Redux DevTools chrome extension

0.1.7 VSC extensions

- Bracket Pair Colorizer - https://marketplace.visualstudio.com/items?itemName=CoenraadS.bracket-pair-colorizer

- ES7 React/Redux/GraphQL/React-Native snippets - https://marketplace.visualstudio.com/items?itemName=dsznajder.es7-react-js-snippets

- Prettier - Code formatter - https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode

VSC=>Manage=>Settings=> “format on save” should be enable

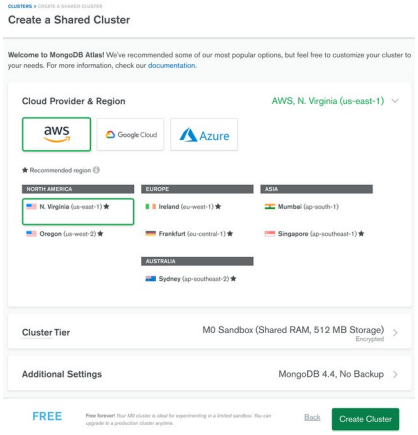
1. Express & MongoDB Setup

1.1 MongoDB (create a new cluster for project)

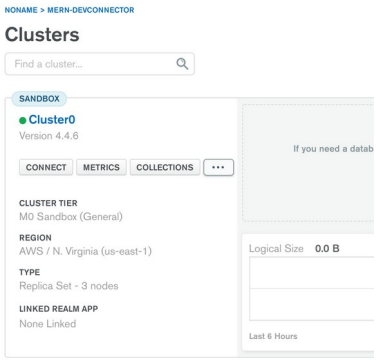
Create an account at https://www.mongodb.com/. Sign in to an account.

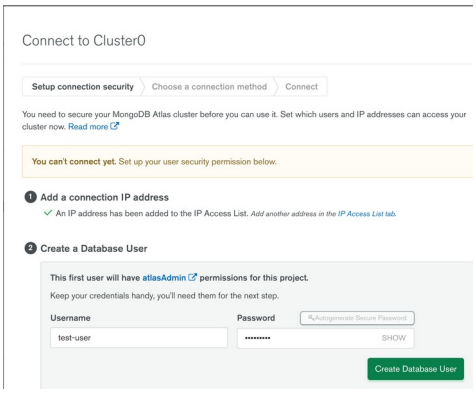
Create a new project – **MERN-devconnector**

Create a new cluster - **Cluster0**

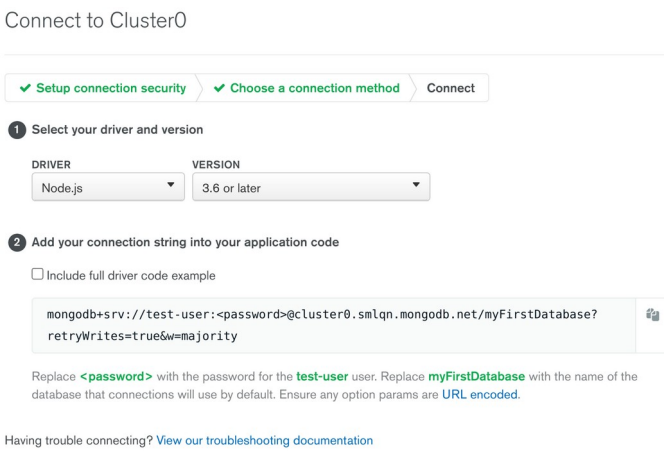


Push button “CONNECT” and add your IP adress and create a new database user to connect to a DB.





Push button “Choose a connection methood” and “connect your application” and save that link



1.1 Express (install a package and setup a server)

1.1.1 Create a .gitignore file

It is a file with list of files/folders witch will be ignored by GIT. Add **node_modules** folder

1.1.2 Initialize an NMP project and setup entry point

CLI=> *npm init -y*

```
stslon@stslon-System-Product-Name:/media/stslon/860_2/june2021/git-projects/mern2$ npm init -y
Wrote to /media/stslon/860_2/june2021/git-projects/mern2/package.json:

{
  "name": "mern2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/ILopatenko/mern2.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/ILopatenko/mern2/issues"
  },
  "homepage": "https://github.com/ILopatenko/mern2#readme"
}
```

This command creates a new file package.json with basic information about project and all the dependencies. I'm going to change entry point ("main":"server.js") for this project to **server.js**

1.1.3 Install all the packages as regular dependencies

I'am going to use in this project next packages:

- **express** – backend server;

- **express-validator** – for validation data;

- **bcryptjs** – for making hash for passwords;

- **config** – for making global variables;

- **gravatar** – for working with user's avatars;

- **jsonwebtoken** – for working with JWT;

- **mongoose** – for working with mongoDB database;

- **request** – for working with another API based services;

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

1.1.4 Install all the packages as DEV dependencies

DEV dependencies will be installed without direct reference to a project (npm will add them to package.json file as a devDependencies)

- **nodemon** – will track all the changes and make a restart server automatically;

- **concurrently** – allows me to run few scripts at the same time with a single command;

CLI=> *npm install* ***-D nodemon concurrently***

```
{} package.json U ×
{} package.json > ...
{
  "name": "mern2",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/ILopatenko/mern2.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/ILopatenko/mern2/issues"
  },
  "homepage": "https://github.com/ILopatenko/mern2#readme",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "config": "^3.3.6",
    "express": "^4.17.1",
    "gravatar": "^1.8.1",
    "jsonwebtoken": "^8.5.1",
    "mongoose": "^5.12.13",
    "request": "^2.88.2"
  },
  "devDependencies": {
    "concurrently": "^6.2.0",
    "nodemon": "^2.0.7"
  }
}
```

1.1.5 Create an express server

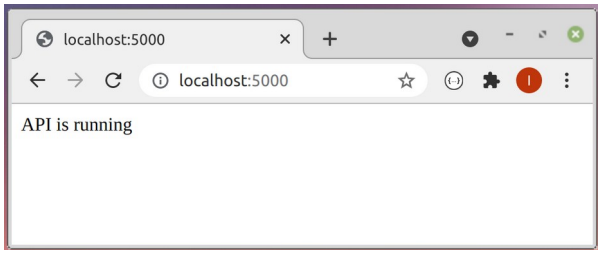
```
JS server.js U ×
_docs > JS server.js > ...
1  const express = require('express');
2
3  const app = express();
4
5  app.get('/', (req, res) => res.send('API is running'));
6
7  const PORT = process.env.PORT || 5000;
8
9  app.listen(PORT, () => console.log(`Server is started on port ${PORT}`));
10
```

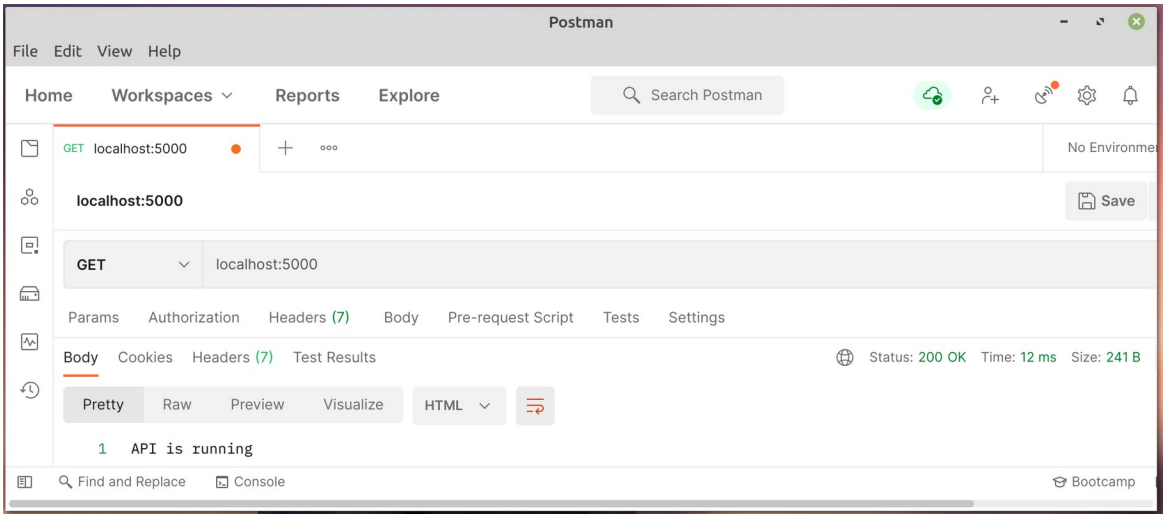
1.1.6 Change run script at package.json

```
"scripts": {
  "start": "node server.js",
  "server": "nodemon server.js"
},
```

Now command “npm run start” will run server.js with node.js and command “npm run server” will run server.js with nodemon

1.1.7 The first run





1.2 Create a connection with mongoDB.

Create a new folder **config**. Inside this folder create a new file – **default.json –** with information to connect with a database

```
{} default.json U ×
config > {} default.json > ...
1  {
2      "mongoURI": "mongodb+srv://test-user:test-user@cluster0.smlqn.mongodb.net/myFirstDatabase?retryWrites=true&w=majority"
3  }
```

Create a new file – **db.js –** with connection logic

```
JS db.js U ●
config > JS db.js > ...
1   const mongoose = require('mongoose');
2
3   const config = require('config');
4
5   const db = config.get('mongoURI');
6
7   const connectDB = async () => {
8     try {
9       await mongoose.connect(db);
10      console.log('MongoDB is connected!');
11    } catch (error) {
12      console.error(err.message);
13      process.exit(1);
14    }
15  };
16
17  module.exports = connectDB;
```

Add **db.js** to a **server.js**

```
JS server.js M ×
JS server.js > ...
1   const express = require('express');
2   const connectToDB = require('./config/db');
3
4   const app = express();
5   connectToDB();
6
7   app.get('/', (req, res) => res.send('API is running'));
8
9   const PORT = process.env.PORT || 5000;
10
11  app.listen(PORT, () => console.log(`Server is started on port ${PORT}`));
12
```

```
stslon@stslon-System-Product-Name:/media/stslon/860_2/june2021/git-projects/mern2$ npm run server

> mern2@1.0.0 server /media/stslon/860_2/june2021/git-projects/mern2
> nodemon server.js

[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:49585) DeprecationWarning: current URL string parser is deprecated, and will be removed in a futur
onnect.
(Use `node --trace-deprecation ...` to show where the warning was created)
Server is started on port 5000
(node:49585) [MONGODB DRIVER] Warning: Top-level use of w, wtimeout, j, and fsync is deprecated. Use wri
(node:49585) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated, and
ne, pass option { useUnifiedTopology: true } to the MongoClient constructor.
MongoDB is connected!
```

Fix all the warnings and run a server again:

1.3 Create routes.

Create a new folder – **routes/api** – to store and work with all the routes.

Inside this folder create new files – **user.js, profile.js, post.js and auth.js**

user.js U

routes > api > user.js > ...

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

//@route      GET api/user
//@desc       Test route
//@access     Public
router.get('/', (req, res) => res.send('User route'));

module.exports = router;
```

profile.js U

routes > api > profile.js > router.get('/') callback

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

//@route      GET api/profile
//@desc       Test route
//@access     Public
router.get('/', (req, res) => res.send('Profile route'));

module.exports = router;
```

post.js U

routes > api > post.js > ...

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

//@route      GET api/post
//@desc       Test route
//@access     Public
router.get('/', (req, res) => res.send('Post route'));

module.exports = router;
```

auth.js U

routes > api > auth.js > ...

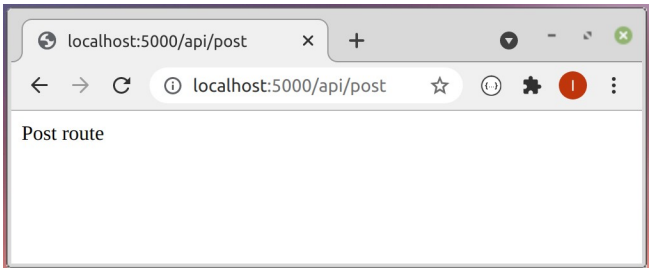
```
const express = require('express');
const router = express.Router();

//@route      GET api/auth
//@desc       Test route
//@access     Public
router.get('/', (req, res) => res.send('Auth route'));

module.exports = router;
```

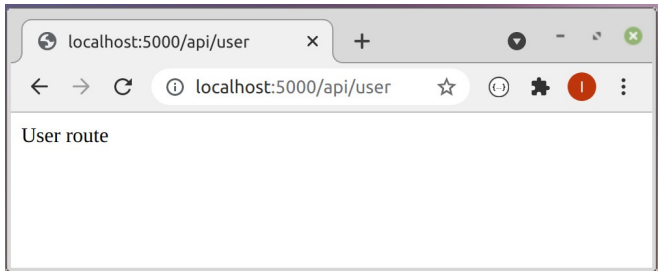
Add all these routes to **server.js**

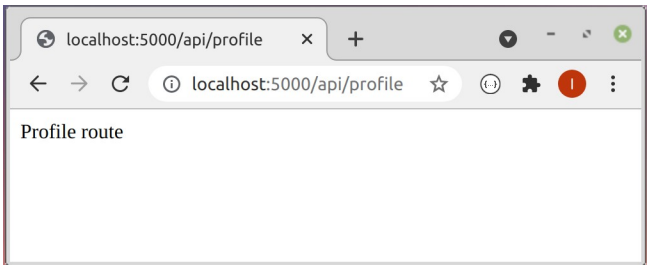
server.js M

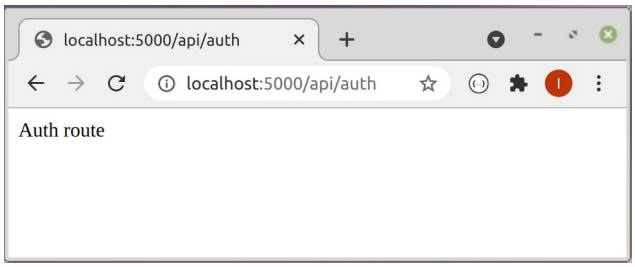
server.js > ...

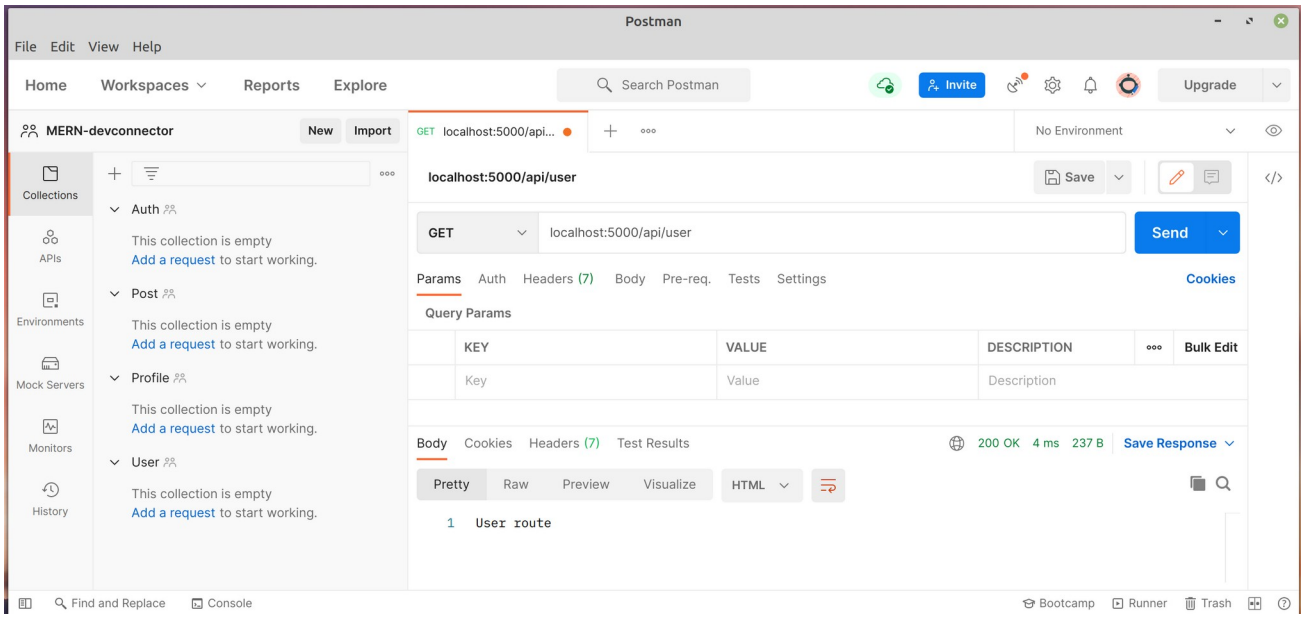
```
const express = require('express');

const connectToDB = require('./config/db');

const userRoute = require('./routes/api/user');
const profileRoute = require('./routes/api/profile');
const postRoute = require('./routes/api/post');
const authRoute = require('./routes/api/auth');

const app = express();
connectToDB();

app.get('/', (req, res) => res.send('API is running'));

//define all the routes
app.use('/api/user', userRoute);
app.use('/api/profile', profileRoute);
app.use('/api/post', postRoute);
app.use('/api/auth', authRoute);

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

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


localhost:5000/api/post
localhost:5000/api/post
Post route


localhost:5000/api/user
localhost:5000/api/user
User route


localhost:5000/api/profile
localhost:5000/api/profile
Profile route


localhost:5000/api/auth
localhost:5000/api/auth
Auth route


Postman
File Edit View Help
Home
Workspaces
Reports
Explore
Search Postman
Invite
Upgrade
MERN-devconnector
New
Import
GET localhost:5000/api...
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
Auth
This collection is empty
Add a request to start working.
Post
This collection is empty
Add a request to start working.
Profile
This collection is empty
Add a request to start working.
User
This collection is empty
Add a request to start working.
localhost:5000/api/user
Save
GET
localhost:5000/api/user
Send
Params
Auth
Headers (7)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (7)
Test Results
200 OK
4 ms
237 B
Save Response
Pretty
Raw
Preview
Visualize
HTML
1 User route
Find and Replace
Console
Bootcamp
Runner
Trash

1.4 User API Routes & JWT Authentication

1.4.1 Create an User schema for MongoDB

Create a new folder – **models** – to store all the schemas.

Create a new file – **User.js** – inside this folder.

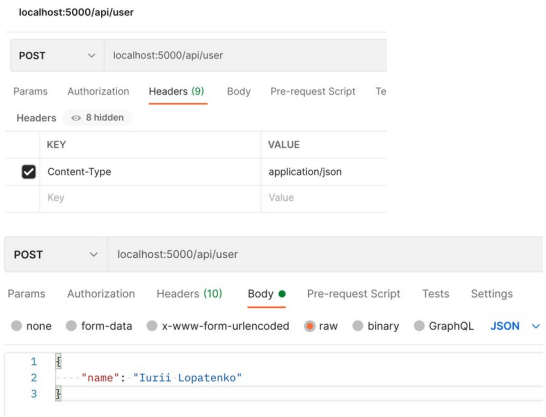
```
JS User.js U ×
models > JS User.js > UserSchema > password
1  const mongoose = require('mongoose');
2
3  const UserSchema = new mongoose.Schema({
4    name: {
5      type: String,
6      required: true,
7    },
8    email: {
9      type: String,
10     required: true,
11     unique: true,
12   },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

1.4.2 Edit test route to *api/*user endpoint

Add Middleware (ex. bodyParser) to work with request object. Change user.js route. Change request in Postman

```
JS server.js M ×
JS server.js > ...
1  const express = require('express');
2
3  const connectToDB = require('./config/db');
4
5  const userRoute = require('./routes/api/user');
6  const profileRoute = require('./routes/api/profile');
7  const postRoute = require('./routes/api/post');
8  const authRoute = require('./routes/api/auth');
9
10 const app = express();
11
12 //Init a built in Middleware (ex. BodyParser)
13 app.use(express.json({ extended: false }));
14
15 connectToDB();
16
17 app.get('/', (req, res) => res.send('API is running'));
18
19 //define all the routes
20 app.use('/api/user', userRoute);
21 app.use('/api/profile', profileRoute);
22 app.use('/api/post', postRoute);
23 app.use('/api/auth', authRoute);
24
25 const PORT = process.env.PORT || 5000;
26
27 app.listen(PORT, () => console.log(`Server is started on port ${PORT}`));
28
```

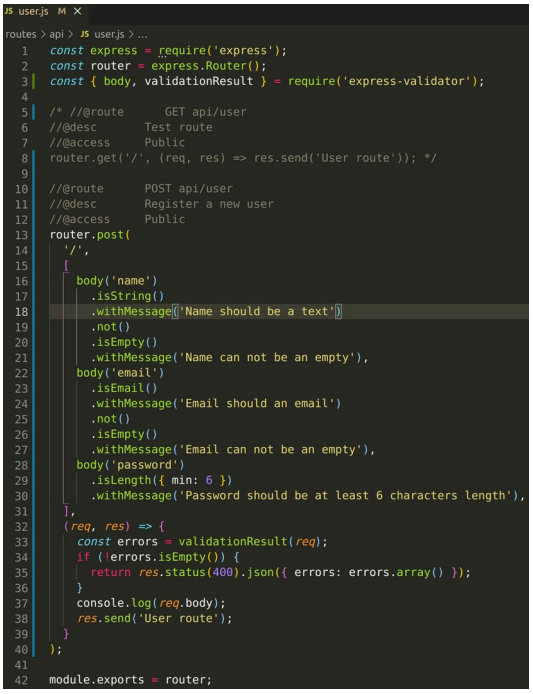
```
JS user.js M ×
routes > api > JS user.js > ...
1  const express = require('express');
2  const router = express.Router();
3
4  /* //@route      GET api/user
5  //@desc       Test route
6  //@access     Public
7  router.get('/', (req, res) => res.se
8
9  //@route      POST api/user
10 //@desc       Register a new user
11 //@access     Public
12 router.post('/', (req, res) => {
13   console.log(req.body);
14   res.send('User route');
15 });
16
17 module.exports = router;
18
```



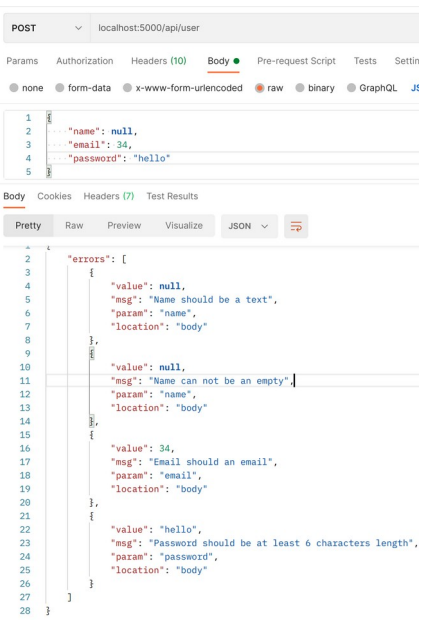

```
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Server is started on port 5000
MongoDB is connected!
{ name: 'Iurii Lopatenko' }
```

Now Postman can send a request (with some data in body) and server can receive this request and work with data (in this example server just sent an object from req.body to console.log)

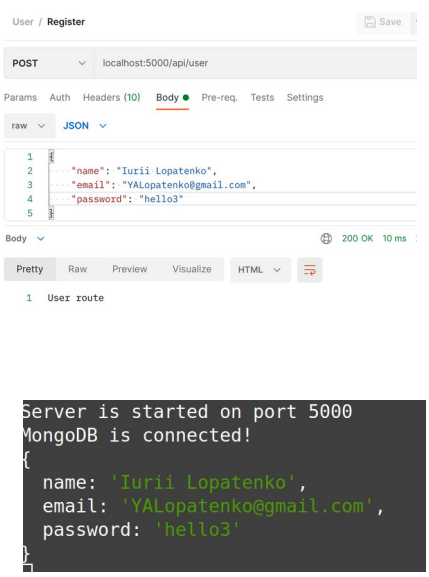
1.4.3 Add data validation with express-validate npm module



```
routes > api > user.js > ...
const express = require('express');
const router = express.Router();
const { body, validationResult } = require('express-validator');

/* //@route      GET api/user
//@desc       Test route
//@access     Public
router.get('/', (req, res) => res.send('User route')); */

//@route      POST api/user
//@desc       Register a new user
//@access     Public
router.post(
  '/',
  [
    body('name')
      .isString()
      .withMessage('Name should be a text')
      .not()
      .isEmpty()
      .withMessage('Name can not be an empty'),
    body('email')
      .isEmail()
      .withMessage('Email should an email')
      .not()
      .isEmpty()
      .withMessage('Email can not be an empty'),
    body('password')
      .isLength({ min: 6 })
      .withMessage('Password should be at least 6 characters length'),
  ],
  (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
    console.log(req.body);
    res.send('User route');
  }
);

module.exports = router;
```



```
POST  localhost:5000/api/user

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

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

{
    "name": null,
    "email": 34,
    "password": "hello"
}

Body  Cookies  Headers (7)  Test Results

Pretty  Raw  Preview  Visualize  JSON

{
    "errors": [
        {
            "value": null,
            "msg": "Name should be a text",
            "param": "name",
            "location": "body"
        },
        {
            "value": null,
            "msg": "Name can not be an empty",
            "param": "name",
            "location": "body"
        },
        {
            "value": 34,
            "msg": "Email should an email",
            "param": "email",
            "location": "body"
        },
        {
            "value": "hello",
            "msg": "Password should be at least 6 characters length",
            "param": "password",
            "location": "body"
        }
    ]
}
```

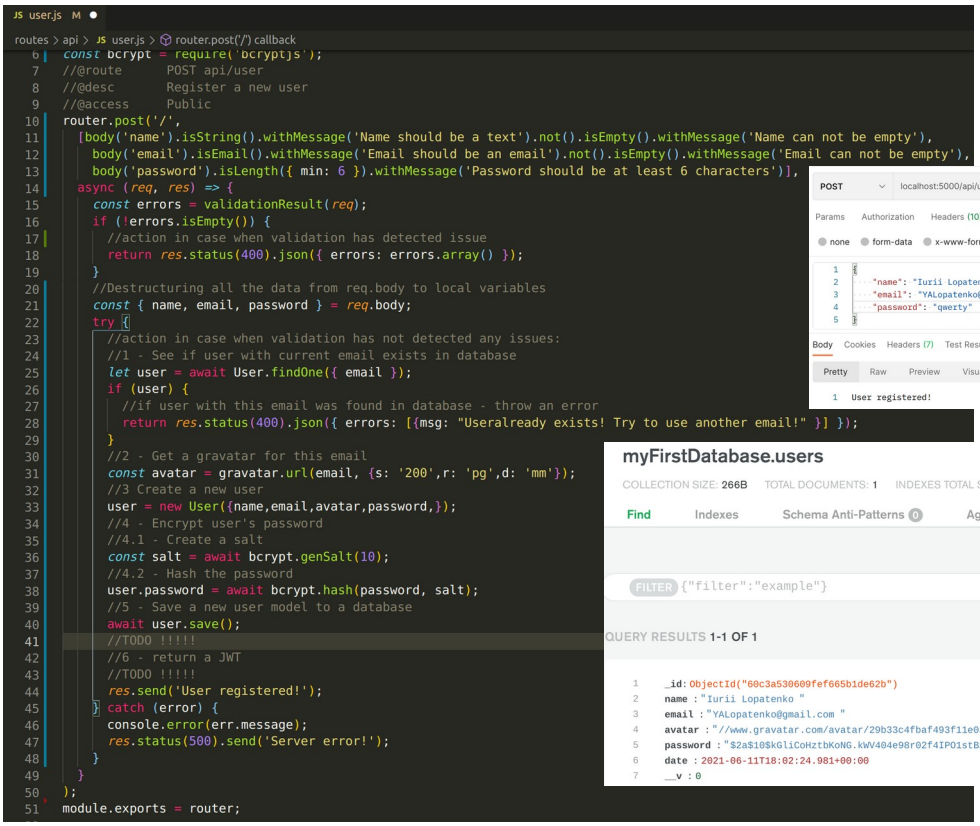


```
User / Register  Save

POST  localhost:5000/api/user

Params  Auth  Headers (10)  Body  Pre-req.  Tests  Settings

raw  JSON

{
    "name": "Iurii Lopatenko",
    "email": "YALopatenko@gmail.com",
    "password": "hello3"
}

Body  200 OK  10 ms

Pretty  Raw  Preview  Visualize  HTML

User route
```

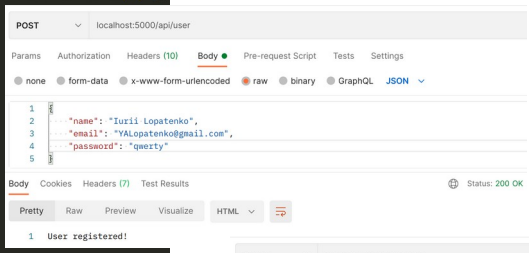


```
Server is started on port 5000
MongoDB is connected!
{
  name: 'Iurii Lopatenko',
  email: 'YALopatenko@gmail.com',
  password: 'hello3'
}
```

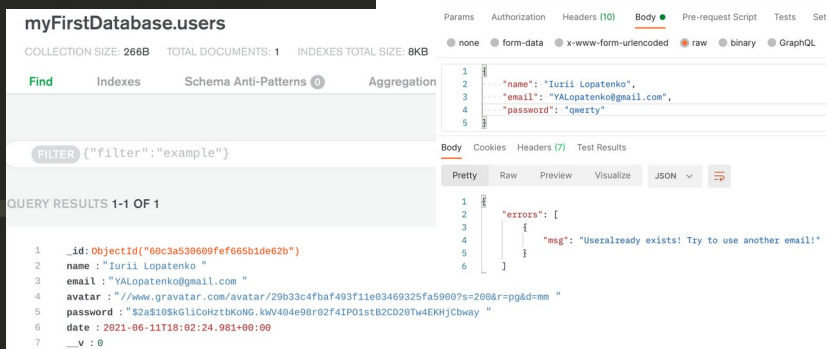
1.4.4 Register a new user.



```
routes > api > user.js > router.post('/') callback
const bcrypt = require('bcryptjs');
//@route      POST api/user
//@desc       Register a new user
//@access     Public
router.post('/',
  [body('name').isString().withMessage('Name should be a text').not().isEmpty().withMessage('Name can not be empty'),
    body('email').isEmail().withMessage('Email should be an email').not().isEmpty().withMessage('Email can not be empty'),
    body('password').isLength({ min: 6 }).withMessage('Password should be at least 6 characters')],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      //action in case when validation has detected issue
      return res.status(400).json({ errors: errors.array() });
    }
    //Destructuring all the data from req.body to local variables
    const { name, email, password } = req.body;
    try {
      //action in case when validation has not detected any issues:
      //1 - See if user with current email exists in database
      let user = await User.findOne({ email });
      if (user) {
        //if user with this email was found in database - throw an error
        return res.status(400).json({ errors: [{msg: "Useralready exists! Try to use another email!" }] });
      }
      //2 - Get a gravatar for this email
      const avatar = gravatar.url(email, {s: '200',r: 'pg',d: 'mm'});
      //3 Create a new user
      user = new User({name,email,avatar,password,});
      //4 - Encrypt user's password
      //4.1 - Create a salt
      const salt = await bcrypt.genSalt(10);
      //4.2 - Hash the password
      user.password = await bcrypt.hash(password, salt);
      //5 - Save a new user model to a database
      await user.save();
      //TODO !!!!!
      //6 - return a JWT
      //TODO !!!!!
      res.send('User registered!');
    } catch (error) {
      console.error(err.message);
      res.status(500).send('Server error!');
    }
  }
);
module.exports = router;
```



```
POST  localhost:5000/api/user

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

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

{
    "name": "Iurii Lopatenko",
    "email": "YALopatenko@gmail.com",
    "password": "qwerty"
}

Body  Cookies  Headers (7)  Test Results  Status: 200 OK

Pretty  Raw  Preview  Visualize  HTML

User registered!
```



```
myFirstDatabase.users

COLLECTION SIZE: 266B  TOTAL DOCUMENTS: 1  INDEXES TOTAL SIZE: 8KB

Find  Indexes  Schema Anti-Patterns  Aggregation

FILTER {"filter":"example"}

QUERY RESULTS 1-1 OF 1

_id: ObjectId("60c3a530609fef665b1de62b")
name : "Iurii Lopatenko "
email : "YALopatenko@gmail.com "
avatar : "//www.gravatar.com/avatar/29b33c4fbaf493f11e03469325fa5900?s=200&r=pg&d=mm "
password : "$2a$10$kGliCoHztbKoNG.kWV404e98r02f4IPO1stB2CD20Tw4EKHjCbway "
date : 2021-06-11T18:02:24.981+00:00
__v : 0
```



```
POST  localhost:5000/api/user

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

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

{
    "name": "Iurii Lopatenko",
    "email": "YALopatenko@gmail.com",
    "password": "qwerty"
}

Body  Cookies  Headers (7)  Test Results

Pretty  Raw  Preview  Visualize  JSON

{
    "errors": [
        {
            "msg": "Useralready exists! Try to use another email!"
        }
    ]
}
```

1.4.5 JWT

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

```
//6 - return a JWT
//6.1 - create a payload
const payload = {
  user: {
    id: user.id,
  },
};
//6.2 - create a JWT with payload and secret and a callback function
jwt.sign(
  payload,
  config.get('jwtSecret'),
  { expiresIn: 360000 },
  (err, token) => {
    if (err) throw err;
    res.json({ token });
  }
);
```

default.json M

config > {} default.json > ...

```
{
    "mongoURI": "mongodb+srv://test-user:test-user@cluster0.smlqn.mongodb.net/myFirstDatabase?retryWrites=true&w=majority",
    "jwtSecret": "superSecret"
}
```

User / Register

Save

POST localhost:5000/api/user

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

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

```
{
    "name": "Test User",
    "email": "test@test.test",
    "password": "qwerty"
}
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 245 ms Size: 431 B

Pretty Raw Preview Visualize JSON

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJ1c2VyIjp7ImlkIjoiNjBjM2FmZTc3ZWE3YjE3ZTEwYmMyOWE0In0sImlhdCI6MTYyMzQzNzI4NywiZXhwIjoxNjIzNzk3Mjg3fQ.
        7FYDsNtfzW57Tz2y0wZfI27291CmXvzfTF5BUuwTv-Y"
}
```

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7ImlkIjoiNjBjM2FmZTc3ZWE3YjE3ZTEwYmMyOWE0In0sImlhdCI6MTYyMzQzNzI4NywiZXhwIjoxNjIzNzk3Mjg3fQ.7FYDsNtfzW57Tz2y0wZfI27291CmXvzfTF5BUuwTv-Y
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "user": {
    "id": "60c3afe77ea7b17e10bc29a4"
  },
  "iat": 1623437287,
  "exp": 1623797287
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

```
_id: ObjectId("60c3afe77ea7b17e10bc29a4")
name: "Test User"
email: "test@test.test"
avatar: "//www.gravatar.com/avatar/dd46a756faad4727fb679320751f6dea?s=200&r=pg&..."
password: "$2a$10$iBBLldnM4f2gUkVctSp85OPktCTBRWrb3918WdPyrlxSINqiikxhu"
date: 2021-06-11T18:48:07.420+00:00
__v: 0
```

1.4.6 Build a custom middleware to work with protected routes

middleware > auth.js > ...

```js
const jwt = require('jsonwebtoken');
const config = require('config');

module.exports = function (req, res, next) {
  //Get a JWT
  const token = req.header('x-auth-token');
  //IF JWT is not exists - throw an error
  if (!token) {
    return res.status(401).json({ msg: 'No token, auth denied!' });
  }
  //Check a JWT
  try {
    //Create a decoded variable with payload (userId, and information abot JWT life time) from decoded JWT
    const decoded = jwt.verify(token, config.get('jwtSecret'));
    //Create a new parameter in request with userID - { id: '60c3bdba5debe09dde3855a7' }
    req.user = decoded.user;
    next();
  } catch (error) {
    res.status(401).json({ msg: 'Token is not valid' });
  }
};
```

routes > api > auth.js > ...

```js
const express = require('express');
const router = express.Router();
const auth = require('../../middleware/auth');
const User = require('../../models/User');

//@route      GET api/auth
//@desc       Test route
//@access     Public
router.get('/', auth, async (req, res) => {
  try {
    const user = await User.findById(req.user.id).select('-password');
    res.json(user);
  } catch (error) {
    console.error(err.message);
    res.status(500).send('Server error');
  }
});

module.exports = router;
```

GET localhost:5000/api/auth

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

Headers (7 hidden)

KEY	VALUE	DESCRIPTION
x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp...	

Body | Cookies | Headers (7) | Test Results — Status: 200 OK Time: 93 ms

Pretty | Raw | Preview | Visualize | JSON

```json
{
    "_id": "60c3b7a68c9b3891ceccbe55",
    "name": "Test2 User",
    "email": "test2@test.test",
    "avatar": "//www.gravatar.com/avatar/8a50f54d8fcf69f53c98c4fd6e647995?s=200&r=pg&d=mm",
    "date": "2021-06-11T19:21:10.466Z",
    "__v": 0
}
```

GET localhost:5000/api/auth

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

Headers (7 hidden)

KEY	VALUE	DESCRIPTION
x-auth-token	wrongToken	

Body | Cookies | Headers (7) | Test Results — Status: 401 Unauthorized Time: 12 ms

Pretty | Raw | Preview | Visualize | JSON

```json
{
    "msg": "Token is not valid"
}
```

1.4.7 User authentication

routes > api > auth.js > router.post('/') callback

```js
//@route      POST api/auth
//@desc       Authenticate user & get a JWT
//@access     Public
router.post('/',[
    body('email').isEmail().withMessage('Email should be an email'),
    body('password').exists().withMessage('Password is required')],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      //action in case when validation has detected issue
      return res.status(400).json({ errors: errors.array() });
    }
    //Destructuring all the data from req.body to local variables
    const { email, password } = req.body;
    try {
      //action in case when validation has not found a user:
      //1 - See if user with current email exists in database
      let user = await User.findOne({ email });
      if (!user) {
        //if user with this email was not found in database - throw an error
        return res.status(400).json({
          errors: [{ msg: 'Invalid credentials' }],
        });
      }
      //2 - checking the password
      const isMatch = await bcrypt.compare(password, user.password);
      if (!isMatch) {
        //if password is wrong - throw an error
        return res.status(400).json({
          errors: [{ msg: 'Invalid credentials' }],
        });
      }
      //3 - return a JWT
      //3.1 - create a payload
      const payload = {
        user: {
          id: user.id,
        },
      };
      //3.2 - create a JWT with payload and secret and a callback function
      jwt.sign(
        payload,
        config.get('jwtSecret'),
        { expiresIn: 360000 },
        (err, token) => {
          if (err) throw err;
          res.json({ token });
        }
      );
    } catch (error) {
      console.error(err.message);
      res.status(500).send('Server error!');
    }
  }
);

module.exports = router;
```

POST localhost:5000/api/auth

Params | Authorization | Headers (11) | Body | Pre-request Script | Tests | Settings

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

```json
{
    "email": "test@test.test",
    "password": "qwerty"
}
```

Body | Cookies | Headers (7) | Test Results — Status: 200 OK Time: 152 ms Size: 431 B

Pretty | Raw | Preview | Visualize | JSON

```json
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJ1c2VyIjp7ImlkIjoiNjBjM2U3ZDIwZGZhNTIxZjZiOTcxMzg5In0sImlhdCI6MTYyMzQ1MjI4NywiZXhwIjoxNjIzODEyMjg3fQ.
        _Kqo9Bd7SKICC15NR_SvnV3LePVrkn6DtpwQLeOsVj8"
}
```

1.5 Profile API Routes

1.5.1 Create a Profile Schema

```
const mongoose = require('mongoose');
const ProfileSchema = new mongoose.Schema({
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},

experience: [{
title: {type: String, required: true},
company: {type: String, required: true},
location: {type: String},
from: {type: Date, required: true},
to: {type: Date},
current: {type: Boolean, required: false},
description: {type: String},
}],

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

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

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

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

What was used in project?

Operating Systems: Linux Mint 20.1 and Windows 10

Node.js – javascript runtime enviroment.

NPM

GIT -

express

express-validator

MongoDB

Mongoose

jwt

nodemon

postman

VSCode

Chrome