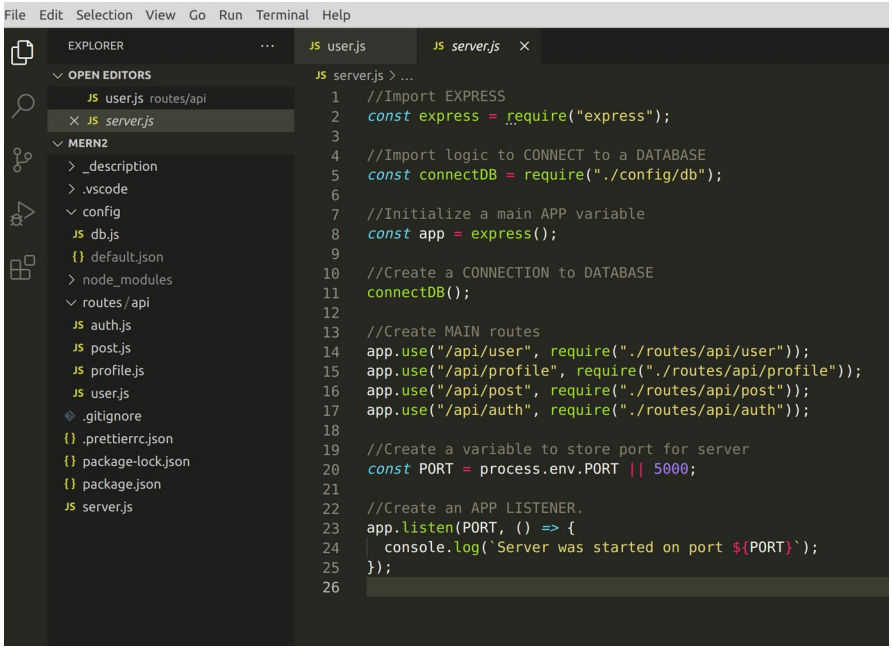


MERN STACK

Social network

DevConnector

Table of Contents

1. Introduction.

There will be a general information about project, MERN stack and all side tecnologies

1.x VSCode

1.x.1 Prettier set up

https://glebbahmutov.com/blog/configure-prettier-in-vscode/#settings

VSCode setup

To use the Prettier we have just installed from VSCode we need to install the Prettier VSCode extension:

1. Launch VS Code Quick Open (Ctrl+P)
2. Run the following command

```
ext install esbenp.prettier-vscode
```

Because you might have global settings related to code formatting, I prefer having in each repository a file with local workspace VSCode settings. I commit this file `.vscode/settings.json` to source control to make sure everyone uses the same extension to format the code.

.vscode/settings.json

```
{
  "editor.defaultFormatter": "esbenp.prettier-vscode",
  "editor.formatOnSave": true
}
```

Now every time we save a JavaScript file, it will be formatted using Prettier automatically. Here is me formatting `projectA/index.js`

2. Express & MongoDB Setup

2.1. MongoDB Atlas Setup

2.1.1 Create an account at MongoDB.com

2.1.2 Log in at MongoDB.com

2.1.3 Create a new project at mongoDB.com

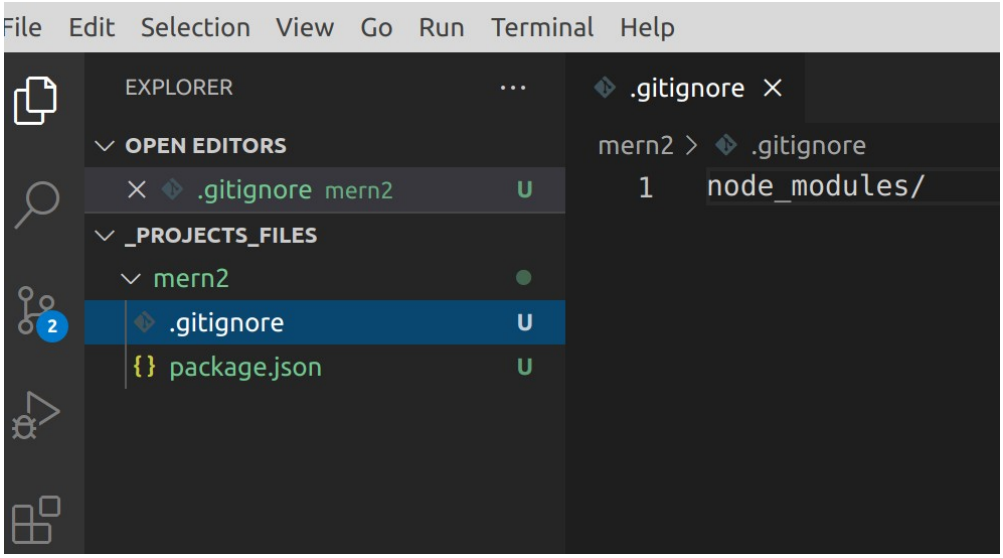
2.1.4 Create a new cluster

2.1.5 Create a new user

2.1.6 Change whitelist of IP addresses

2.2 Install Dependencies & Basic Express Setup

2.2.1 Create a file .gitignore. Add to that file folder node_modules/



2.2.2 Create GIT repository with [CLI=> git init]

2.2.3 Create NPM with [CLI=> npm init]

2.2.4 Install all the regular dependencies

[CLI=> npm i express express-validator bcryptjs config gravatar jsonwebtoken mongoose request]

2.2.5 Install all the developer dependencies

[CLI=> npm i -D nodemon concurrently]

2.2.6 Create a main entry file – server.js

[CLI=> touch server.js]

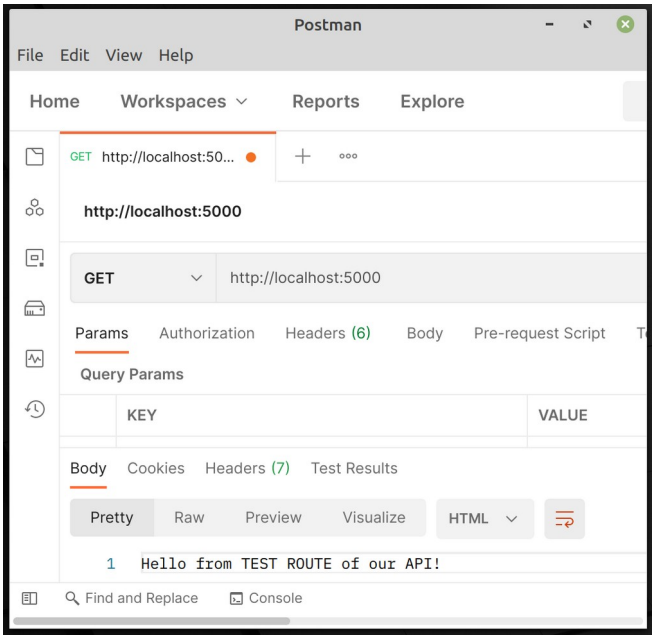
2.2.7 Change start scripts at package.json

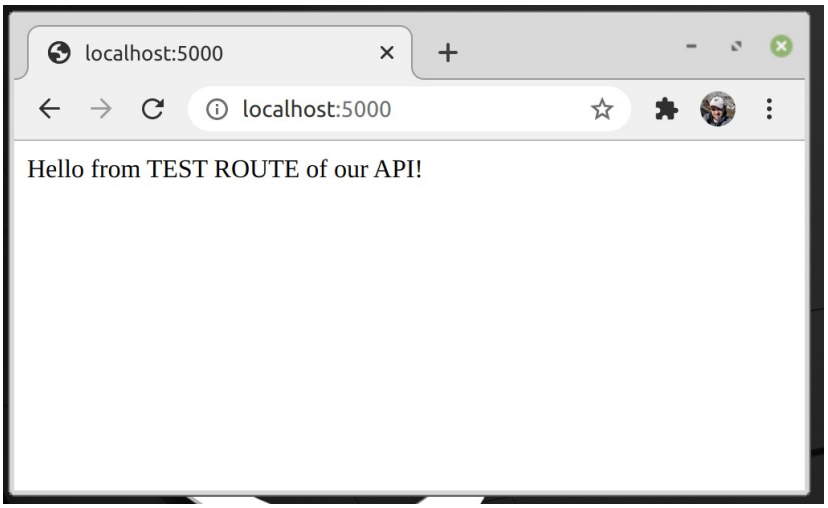
```
{} package.json ×
{} package.json > ...
1  {
2    "name": "mern-devconnector",
3    "version": "1.0.0",
4    "description": "social network MERN stack",
5    "main": "server.js",
       ▷ Debug
6    "scripts": {
7      "start": "node server",
8      "server": "nodemon server.js"
9    },
```

Now to start server: [CLI=> npm run server]. It will start SERVER.JS file with NODEMON

2.2.8 Create a SERVER with simple test route

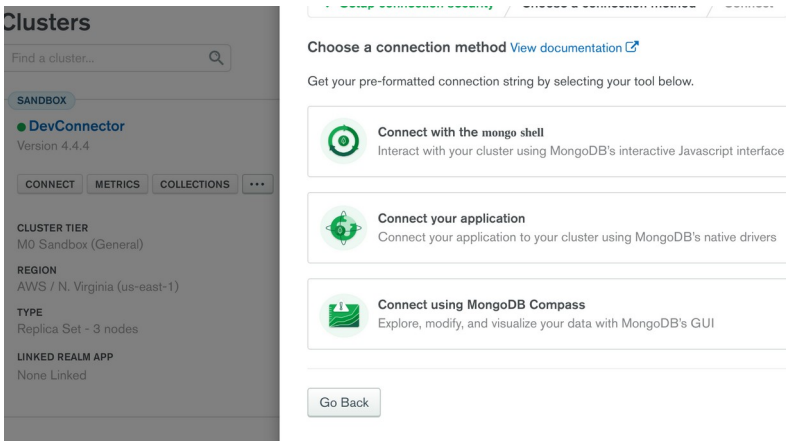
```
JS server.js ×
JS server.js > ...
1   //Import EXPRESS
2   const express = require("express");
3
4   //Initialize a main APP variable
5   const app = express();
6
7   //Create a TEST ROUTE (for GET request to root folder '/')
8   app.get("/", (req, res) => res.send("Hello from TEST ROUTE of our API!"));
9
10  //Create a variable to store port for server
11  const PORT = process.env.PORT || 5000;
12
13  //Create an APP LISTENER.
14  app.listen(PORT, () => {
15    console.log(`Server was started on port ${PORT}`);
16  });
```



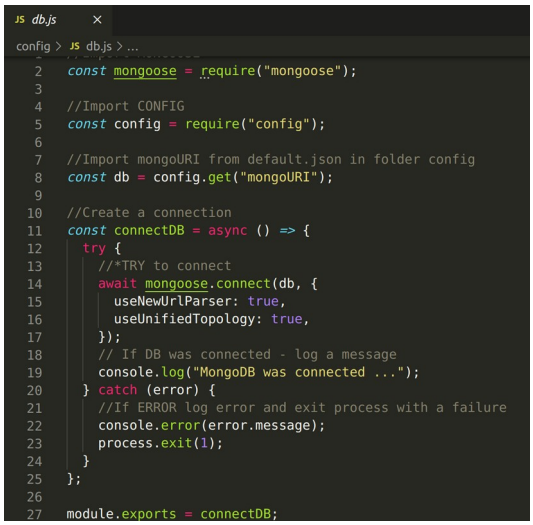


2.3 Connecting To MongoDB With Mongoose

2.3.1 Create a connection to database



2.3.2 Create a connection logic on server



```
const mongoose = require("mongoose");

//Import CONFIG
const config = require("config");

//Import mongoURI from default.json in folder config
const db = config.get("mongoURI");

//Create a connection
const connectDB = async () => {
  try {
    //*TRY to connect
    await mongoose.connect(db, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    // If DB was connected - log a message
    console.log("MongoDB was connected ...");
  } catch (error) {
    //If ERROR log error and exit process with a failure
    console.error(error.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

```
server.js
server.js > ...
1  //Import EXPRESS
2  const express = require("express");
3
4  //Import logic to CONNECT to a DATABASE
5  const connectDB = require("./config/db");
6
7  //Initialize a main APP variable
8  const app = express();
9
10 //Create a CONNECTION to DATABASE
11 connectDB();
12
13 //Create a TEST ROUTE (for GET request to root folder '/')
14 app.get("/", (req, res) => res.send("Hello from TEST ROUTE of our API!"));
15
16 //Create a variable to store port for server
17 const PORT = process.env.PORT || 5000;
18
19 //Create an APP LISTENER.
20 app.listen(PORT, () => {
21   console.log(`Server was started on port ${PORT}`);
22 });
23
```

```
stslon@stslon-System-Product-Name:/media/stslon/8CB04F45B04F354C/

> mern-devconnector@1.0.0 server
> 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:24960) Warning: Accessing non-existent property 'MongoError
(Use `node --trace-warnings ...` to show where the warning was cr
Server was started on port 5000
(node:24960) DeprecationWarning: Listening to events on the Db cl
MongoDB was connected ...
```

2.4 Create route files with Express Router

```
server.js
server.js > ...
1  //Import EXPRESS
2  const express = require("express");
3
4  //Import logic to CONNECT to a DATABASE
5  const connectDB = require("./config/db");
6
7  //Initialize a main APP variable
8  const app = express();
9
10 //Create a CONNECTION to DATABASE
11 connectDB();
12
13 //Create MAIN routes
14 app.use("/api/user", require("./routes/api/user"));
15 app.use("/api/profile", require("./routes/api/profile"));
16 app.use("/api/post", require("./routes/api/post"));
17 app.use("/api/auth", require("./routes/api/auth"));
18
19 //Create a variable to store port for server
20 const PORT = process.env.PORT || 5000;
21
22 //Create an APP LISTENER.
23 app.listen(PORT, () => {
24   console.log(`Server was started on port ${PORT}`);
25 });
26
```

```
EXPLORER
OPEN EDITORS
  user.js routes/api                U
MERN2
  _description
  .vscode
  config
    db.js
    default.json
  node_modules
  routes/api
    auth.js                         U
    post.js                         U
    profile.js                      U
    user.js                         U
  .gitignore
  .prettierrc.json
  package-lock.json
  package.json
  server.js                         M
```

```
user.js
routes > api > user.js > ...
1  //Import EXPRESS
2  const express = require("express");
3
4  //Import EXPRESS ROUTER
5  const router = express.Router();
6
7  router.get("/login", (req, res) => {
8    res.send("hello from api/user/login");
9  });
10
11 router.get("/", (req, res) => {
12   res.send("hello from api/user");
13 });
14
15 module.exports = router;
16
```

3. User API Routes & JWT Authentication

3.1 Create USER MODEL (SCHEMA)

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

3.2 Request & Body Validation

```
JS server.js
JS server.js > ...
1   //Import EXPRESS
2   const express = require("express");
3
4   //Import logic to CONNECT to a DATABASE
5   const connectDB = require("./config/db");
6
7   //Initialize a main APP variable
8   const app = express();
9
10  //Create a CONNECTION to DATABASE
11  connectDB();
12
13  //Import MIDDLEWARE for work with body of REQUEST (bodyparser)
14  app.use(express.json({ extended: false }));
15
16  //Create MAIN routes
17  app.use("/api/user", require("./routes/api/user"));
18  app.use("/api/profile", require("./routes/api/profile"));
19  app.use("/api/post", require("./routes/api/post"));
20  app.use("/api/auth", require("./routes/api/auth"));
21
22  //Create a variable to store port for server
23  const PORT = process.env.PORT || 5000;
24
25  //Create an APP LISTENER.
26  app.listen(PORT, () => {
27    console.log(`Server was started on port ${PORT}`);
28  });
29
```

```
JS registration.js
routes > api > user > JS registration.js > ...
1   //Import EXPRESS
2   const express = require("express");
3   //Import EXPRESS ROUTER
4   const router = express.Router();
5   //Import check and validationResult to work with data from REQUEST
6   const { check, validationResult } = require("express-validator");
7   router.post("/", [
8       check("name", "Check NAME and try again").trim().not().isBoolean().isLength({ min: 6 }),
9       check("email", "Check EMAIL and try again").trim().isEmail(),
10      check("password", "Check PASSWORD and try again").trim().isLength({ min: 6 }),
11    ],(req, res) => {
12      const errors = validationResult(req);
13      if (!errors.isEmpty()) return res.status(400).json({ errors: errors.array() });
14      res.send(req.body);
15    }
16  );
17  module.exports = router;
```

```
JS user.js
routes > api > JS user.js > ...
1   //Import EXPRESS
2   const express = require("express");
3
4   //Import EXPRESS ROUTER
5   const router = express.Router();
6
7   /////////////// START_ROUT ////////////////
8
9   router.use("/registration", require("./user/registration"));
10  router.use("/login", require("./user/login"));
11  ///////////////// END_ROUT ////////////////
12
13  module.exports = router;
```

3.3 User registration logic

registration.js

routes > api > user > registration.js > ...

```
1  //Import EXPRESS
2  const express = require("express");
3  //Import EXPRESS ROUTER
4  const router = express.Router();
5  //Import check and validationResult to work with data from REQUEST
6  const { check, validationResult } = require("express-validator");
7  //Import User model
8  const User = require("../../../models/User");
9  //Import gravatar
10 const gravatar = require("gravatar");
11 //Import BCRYPT
12 const bcrypt = require("bcryptjs");
13 router.post("/",
14 > [ ...
24     ],
25   async (req, res) => {
26     const errors = validationResult(req);
27     if (!errors.isEmpty()) {return res.status(400).json({ errors: errors.array() });}
28     //REGISTRATION ROUTE LOGIC
29     //Check if user (with email from req.body.email) exists in DATABASE
30     //(if exists - send status code and message if does not - create a new USER)
31     //DESTRUCTURING NAME, EMAIL and PASSWORD from request (req.body.name, req.body.ema
32     const { name, email, password } = req.body;
33     try {
34       //Try to find USER with email from request in DATABASE
35       let user = await User.findOne({ email });
36       //If USER was found - return status code 400 with a message
37       if (user) {return res.status(400).json({ errors: [{ msg: "User already exists!" }] });}
38       //If USER was not found in DATABASE - create a new USER
39       //Create an avatar for new USER with GRAVATAR
40       const avatar = gravatar.url(email, {s: "200", r: "pg", d: "mm"});
41       //Create a new instance of USER
42       user = new User({ name, email, password, avatar });
43       //Create SALT for making a HASH of PASSWORD
44       const salt = await bcrypt.genSalt(10);
45       //Change USER.PASSWORD from direct PASSWORD to a HASH of PASSWORD
46       user.password = await bcrypt.hash(password, salt);
47       //SAVE new USER
48       await user.save();
49       //Return a message
50       res.send("User was registered");
51     } catch (error) {
52       console.error(error.message);
53       res.status(500).send("Server error!");
54     }
55   }
56 );
57 module.exports = router;
```

Postman

POST http://localhost:5000/api/user/registration

Body (raw, JSON):

```
{
    "name": "Iurii Lopatenko",
    "email": "yalopatenko@gmail.com",
    "password": "testpass"
}
```

Status: 200 OK Time: 281 ms

```
User was registered
```

Postman

POST http://localhost:5000/api/user/registration

Body (raw, JSON):

```
{
    "name": "Iurii Lopatenko",
    "email": "yalopatenko@gmail.com",
    "password": "testpass"
}
```

Status: 400 Bad Request Time: 83 ms

```
{
    "errors": [
        {
            "msg": "User already exists!"
        }
    ]
```

myFirstDatabase.users

COLLECTION SIZE: 266B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 20KB

Find Indexes Schema Anti-Patterns 0 Aggregation Search Indexes

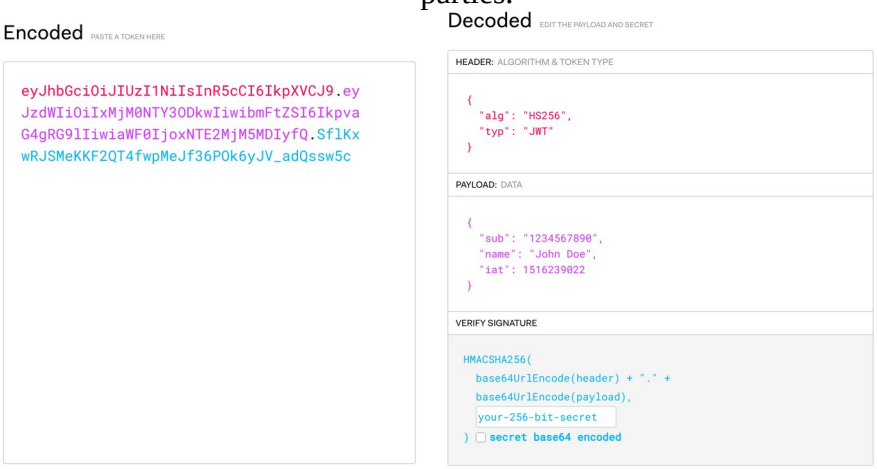
FILTER {"filter":"example"}

QUERY RESULTS 1-1 OF 1

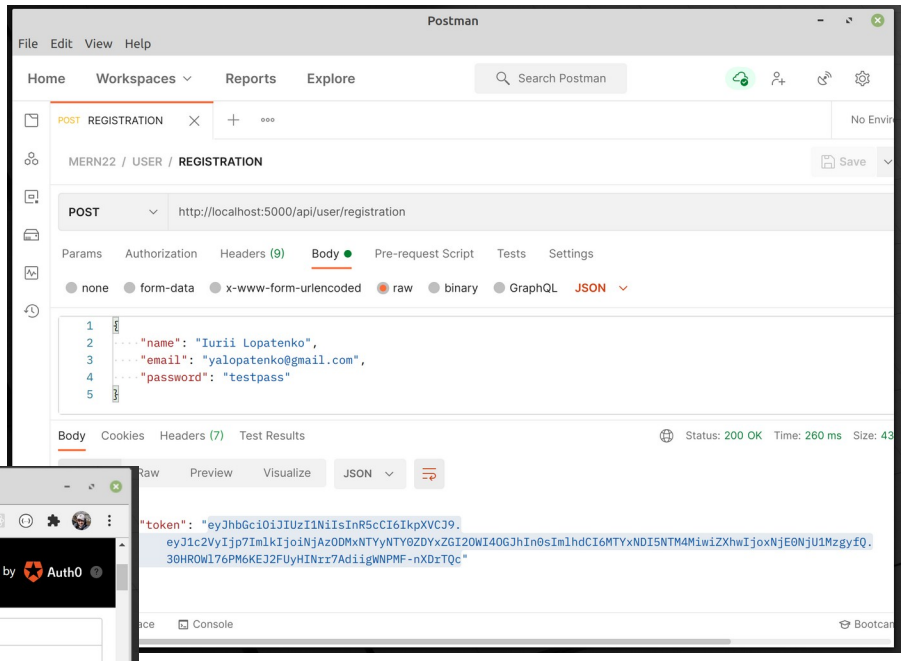
```
_id: ObjectId("603823894e5f00189c1a843e")
name: "Iurii Lopatenko"
email: "yalopatenko@gmail.com"
password: "$2a$10$6RgBg23Ps3NnAjzeJulGUebkJFNm5Vf6jWrwoURe20P2vBW7KpKp."
avatar: "//www.gravatar.com/avatar/29b33c4fbaf493f11e03469325fa5900?s=200&r=pg&..."
date: 2021-02-25T22:24:09.841+00:00
__v: 0
```

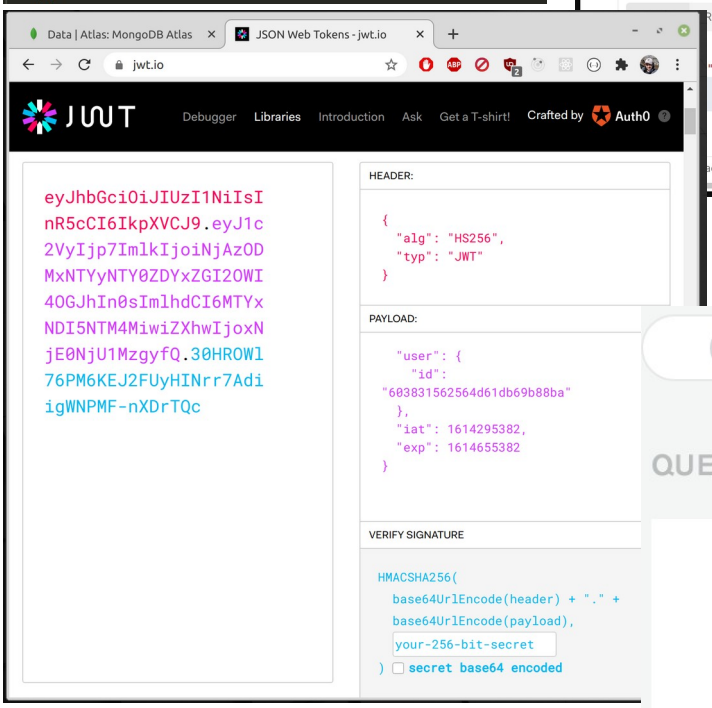
3.4 JWT implementing (jwt.io)

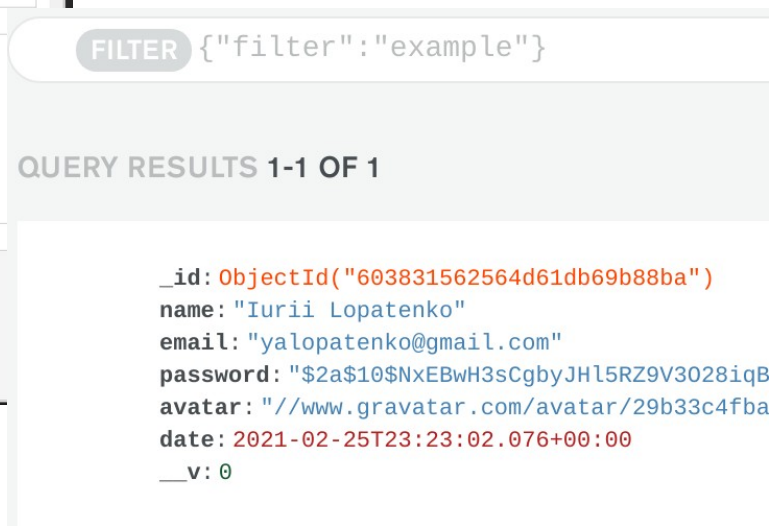
JSON Web Tokens are an open, industry standard RFC 7519 method for representing data securely between two parties.




```
await user.save();
//Create a payload for generate a JWT
const payload = {
  user: { id: user.id },
};
//Generate a TOKEN
jwt.sign(
  payload,
  config.get("secretForJWT"),
  { expiresIn: 360000 },
  (err, token) => {
    if (err) throw err;
    res.json({ token });
  }
);
```







3.5 Custom auth middleware and JWT verify

```
JS login.js
routes > api > user > JS login.js > ...
1  //Import EXPRESS
2  const express = require('express');
3
4  //Import EXPRESS ROUTER
5  const router = express.Router();
6
7  //Import a MIDDLEWARE function
8  const auth = require('../../../middleware/auth');
9
10 //Import User model
11 const User = require('../../../models/User');
12
13 //Import check and validationResult to work with data from REQUEST
14 const { check, validationResult } = require('express-validator');
15
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   }
23 });
24 module.exports = router;
```

```
JS auth.js
middleware > JS auth.js > ...
1  //Import JWT
2  const jwt = require('jsonwebtoken');
3
4  //Import CONFIG
5  const config = require('config');
6
7  module.exports = (req, res, next) => {
8    //get token from request - req.header
9    const token = req.header('x-auth-token');
10
11   //check if token does not exist - return status code 400 and message "No to
12   if (!token) {
13     return res.status(400).json({ msg: 'No token. Authorization denied!' });
14   }
15
16   //if token exists - verify the token
17   try {
18     // decode the TOKEN
19     const decoded = jwt.verify(token, config.get('secretForJWT'));
20     //Change USER at REQUEST to decoded USER (from TOKEN)
21     req.user = decoded.user;
22     //make a signal that a function has finished work
23     next();
24   } catch (error) {
25     res.status(401).json({ msg: 'Token is not valid' });
26   }
27 };
```

Postman

File Edit View Help

Home Workspaces Reports Explore Search Postman

GET LOG IN (with JWT)

MERN22 / USER / LOG IN (with JWT) Save

GET http://localhost:5000/api/user/login

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

Headers 6 hidden

KEY	VALUE	DESCRIPTION
Content-Type	application/json	
x-auth-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp...	

Body Cookies Headers (7) Test Results Status: 200 OK Time: 69 ms Size: 455

Pretty Raw Preview Visualize JSON

```
1 {
2   "_id": "60383bb99bbdbb31a138780c",
3   "name": "Iurii Lopatenko",
4   "email": "yalopatenko@gmail.com",
5   "avatar": "//www.gravatar.com/avatar/29b33c4fbaf493f11e03469325fa5900?s=200&r=pg&d=mm",
```

Find and Replace Console Bootcamp

GET request to /api/user/login

In request.body should be a JWT – server will use a MIDLEWARE function (this function will check if token exist in body of request, then it will decode token and add USER ID from JWT data to REQUEST array as object)

user: { id: ‘xXxXxXx’ }

3.6 User login route

```
JS login.js ●
routes > api > user > JS login.js > ...
33
34  router.post(
35    '/',
36    [
37      check('email', 'Check EMAIL and try again').trim().isEmail(),
38      check('password', 'Password is required').trim().exists(),
39    ],
40    async (req, res) => {
41      const errors = validationResult(req);
42      if (!errors.isEmpty()) {return res.status(400).json({ errors: errors.array() })}
43      //DESTRUCTURING EMAIL and PASSWORD from request (req.body.email, req.body.password)
44      const { email, password } = req.body;
45      try {
46        //Try to find USER with email from request in DATABASE
47        let user = await User.findOne({ email });
48        //If USER was not found - return status code 400 with a message
49        if (!user) {return res.status(400).json({ errors: [{ msg: 'Invalid credentials' }] })}
50        //If USER was found in DATABASE
51        //Check if  password from request is the same as a user.password
52        const isMatch = await bcrypt.compare(password, user.password);
53        if (!isMatch) {return res.status(400).json({ errors: [{ msg: 'Invalid credentials' }] })}
54        //Create a payload for generate a JWT
55        const payload = {user: { id: user.id },};
56        //Generate a TOKEN
57        jwt.sign(
58          payload,
59          config.get('secretForJWT'),
60          { expiresIn: 360000 },
61          (err, token) => {
62            if (err) throw err;
63            //If no any errors - sent a JWT as response
64            res.json({ token });
65          }
66        );
67      } catch (error) {
68        console.error(error.message);
69        res.status(500).send('Server error!');
70      }
71    }
72  );
73  module.exports = router;
74
```

Post request to /api/user/login

In request body should be an object with email and password. Server will check email and password (lines 37, 38), will destructive email and password from body of request to separate variables (line 44), will try to find user with email from body of request in DATABASE (will throw an error if it will not possible), then it will compare password from body of request and password from database (HASH of password) with bcrypt and if they will be the same – server will generate a PAYLOAD for new JWT, will generate JWT itself and will send it as response.

4. Profile API routes

4.1 Create a Profile model

Profile.js

models > Profile.js > ...

```
1   //Import mongoose mongoose
2   //Import MONGOOSE
3   const mongoose = require('mongoose');
4   //Create a PROFILE model (schema)
5   const ProfileSchema = new mongoose.Schema({
6     //Link to another schema
7     user: {type: mongoose.Schema.Types.ObjectId, ref: 'user'},
8     //Main information
9     company: { type: String },
10    website: { type: String },
11    location: { type: String },
12    status: { type: String, required: true },
13    skills: { type: [String], required: true },
14    bio: { type: String },
15    githubusername: { type: String },
16    //Experience
17    experience: [{
18          title: { type: String, required: true },
19          company: { type: String, required: true },
20          location: { type: String },
21          from: { type: Date, required: true },
22          to: { type: Date },
23          current: { type: Boolean, default: false },
24          description: { type: String }}],
25    //Education
26    experience: [{
27          school: { type: String, required: true },
28          degree: { type: String, required: true },
29          fieldofstudy: { type: String, required: true },
30          from: { type: Date, required: true },
31          to: { type: Date },
32          current: { type: Boolean, default: false },
33          description: { type: String }}],
34    social: {
35        youtube: { type: String },
36        twitter: { type: String },
37        facebook: { type: String },
38        linkedin: { type: String },
39        instagram: { type: String }},
40    date: {type: Date, default: Date.now}
41  });
42  module.exports = Profile = mongoose.model('profile', ProfileSchema);
```

14. Basic GIT commands

14.1 Work with commit

14.1.1 Add all the changes from local work directory to a work tree

[CLI=> **git add .**] **-** will add all the changes

[CLI=> **git add** ***[fileName.extension]***] – will add only a single file (fileName.extension) – ***git add index.html***

14.1.2 Create a new commit

[CLI=> **git commit -*[baranch_name]* '*[commit message with meaningful description of changes]*'**] – will create a new commit on branch (branch_name) with message (commit message with meaningful description of changes) – ***git commit -main 'Create a user login functionality'***

14.1.3 Create a new branch and switch to it

[CLI=> **git switch -*C* *[name_for_a_new_branch]***] **-** will create a new branch and then switch to a new branch **-** ***git switch -C User-API-Routes-and-JWT-Authentication***

14.1.4 Merge all the commits to a master branch from another branch

Switch to a MASTER branch with command [CLI=> **git switch master**]

Merge [CLI=> **git merge** [***baranch_name***]] – ***git merge User-API-Routes-and-JWT-Authentication***