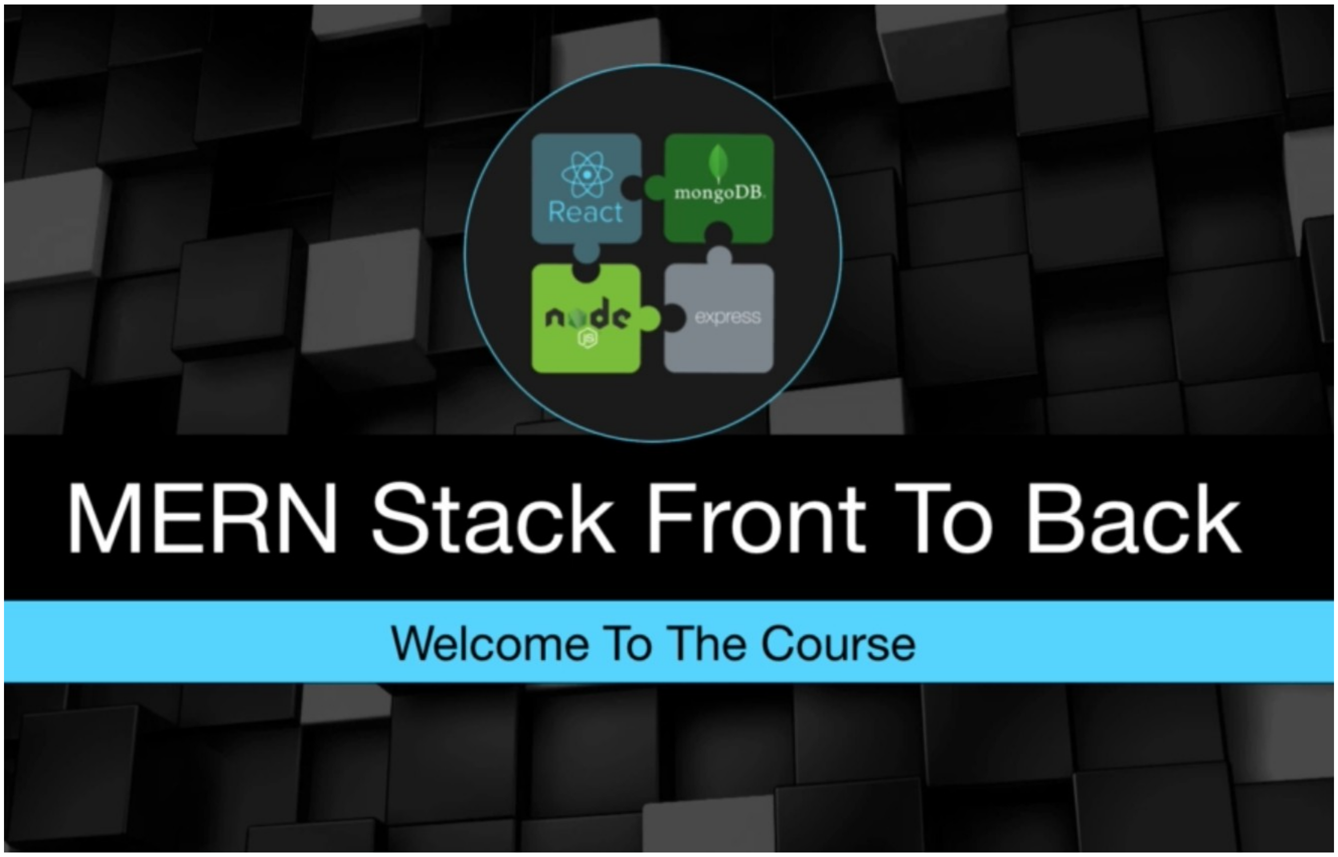
MERN Stack Front To Back: Full Stack React, Redux & Node.js (2020)

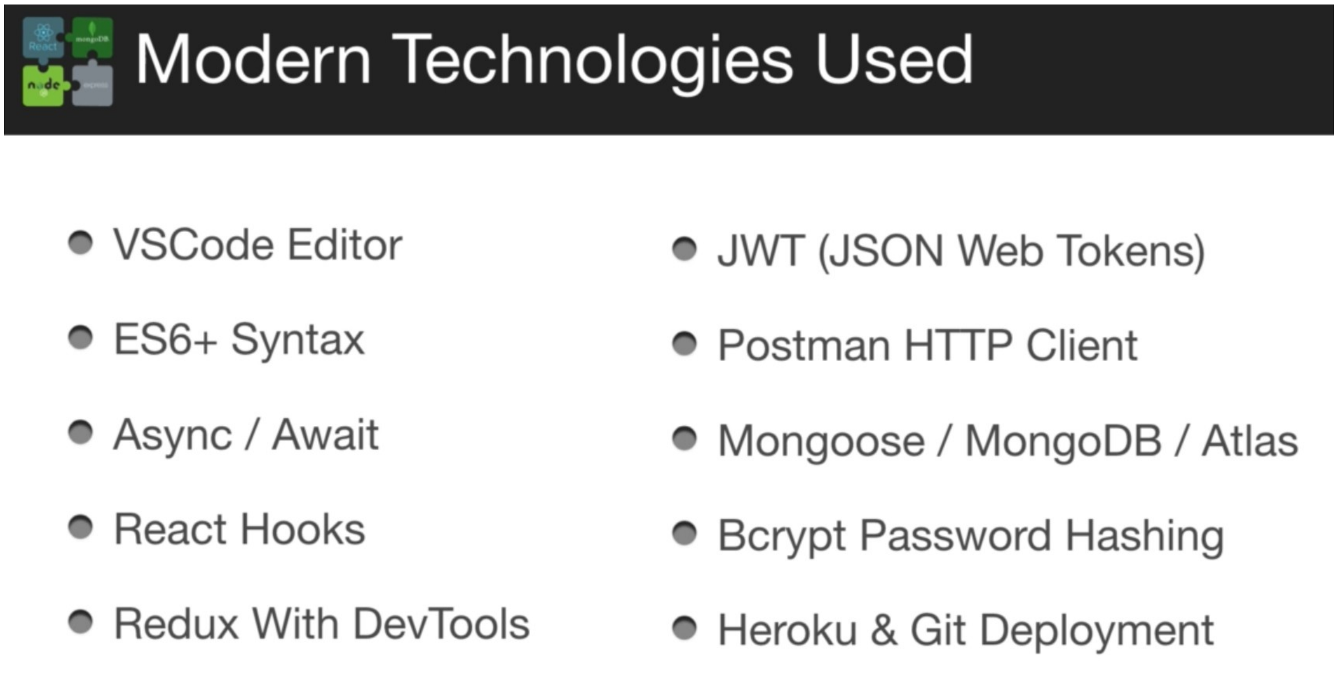


https://www.udemy.com/course/mern-stack-front-to-back/

by Brad Traversy

Table of Contents

0. Introduction



Modern Technologies Used

- 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

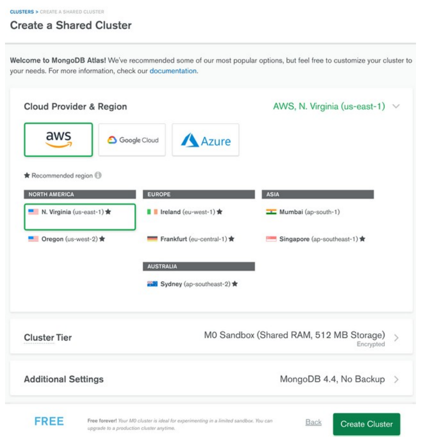
1. Express & MongoDB Setup

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**



CLUSTERS > CREATE A SHARED CLUSTER

Create a Shared Cluster

Welcome to MongoDB Atlas! We've recommended some of our most popular options, but feel free to customize your cluster to your needs. For more information, check our documentation.

Cloud Provider & Region: AWS, N. Virginia (us-east-1)

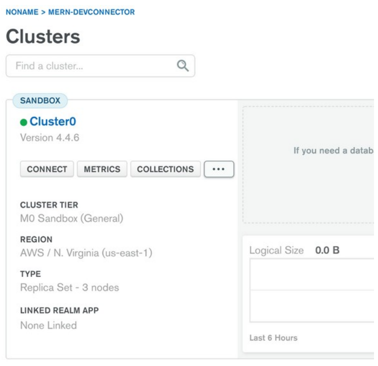
Cluster Tier: M0 Sandbox (Shared RAM, 512 MB Storage) Encrypted

Additional Settings: MongoDB 4.4, No Backup

FREE

Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

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



NONAME > MERN-DEVCONNECTOR

Clusters

Cluster0

Version 4.4.6

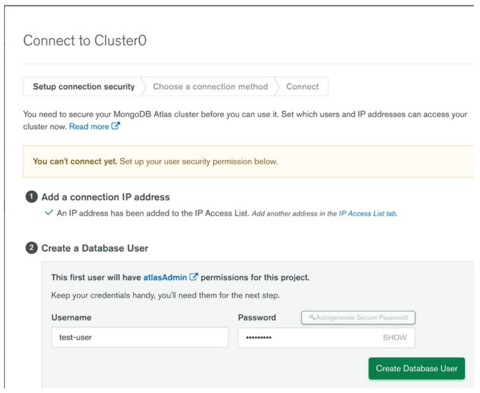
CLUSTER TIER: M0 Sandbox (General)

REGION: AWS / N. Virginia (us-east-1)

TYPE: Replica Set - 3 nodes

LINKED REALM APP: None Linked

Logical Size 0.0 B



Connect to Cluster0

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. Read more

You can't connect yet. Set up your user security permission below.

1. Add a connection IP address

   ✓ An IP address has been added to the IP Access List. *Add another address in the IP Access List tab.*

2. Create a Database User

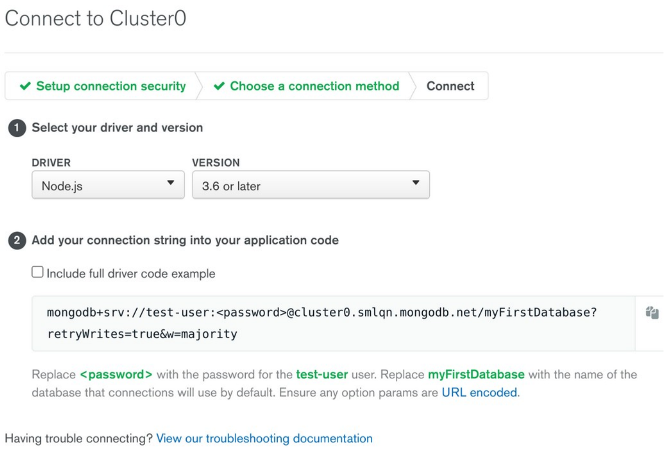
   This first user will have atlasAdmin permissions for this project.

   Keep your credentials handy, you'll need them for the next step.

   Username: test-user

   Password: ••••••••••

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



Connect to Cluster0

1. Select your driver and version

   DRIVER: Node.js

   VERSION: 3.6 or later

2. Add your connection string into your application code

   ☐ Include full driver code example

```
mongodb+srv://test-user:<password>@cluster0.smlqn.mongodb.net/myFirstDatabase?
retryWrites=true&w=majority
```

Replace **<password>** with the password for the **test-user** user. Replace **myFirstDatabase** with the name of the database that connections will use by default. Ensure any option params are URL encoded.

Having trouble connecting? View our troubleshooting documentation

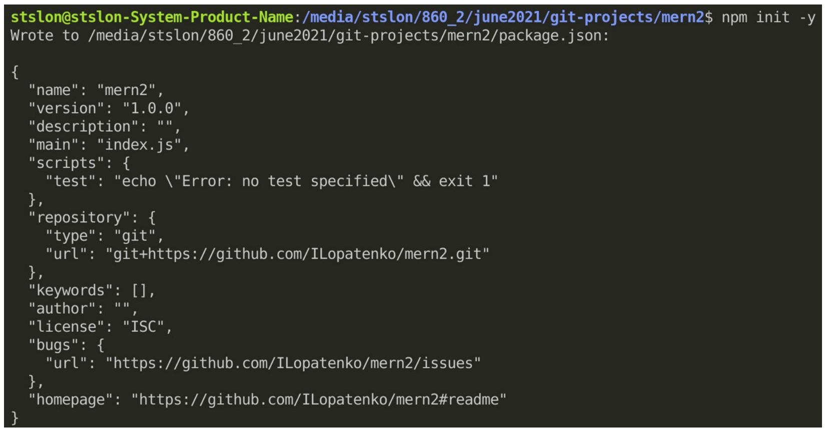
1.1 Express (install a package and setup a server)

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***

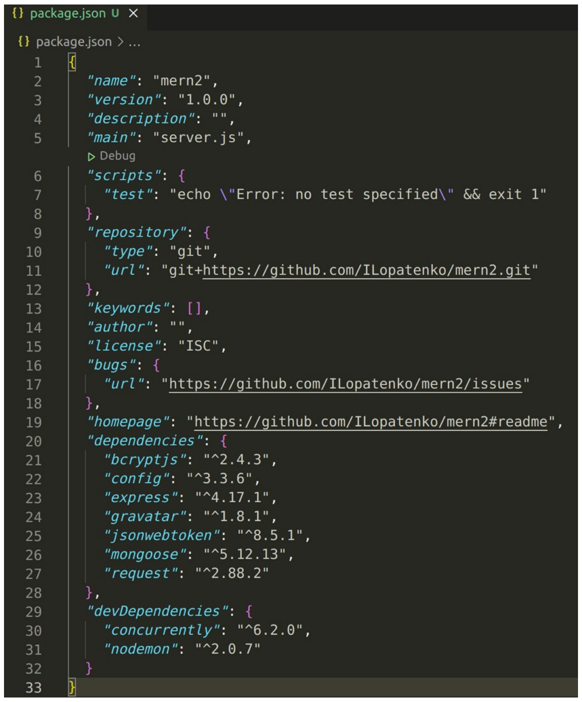
1.1.4 Install all the packages as DEV dependencies

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***

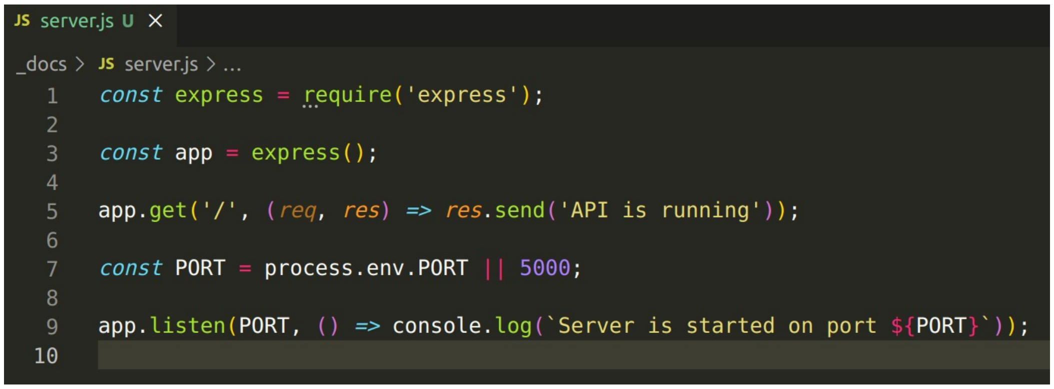


package.json U

package.json > ...

```json
{
  "name": "mern2",
  "version": "1.0.0",
  "description": "",
  "main": "server.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",
  "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

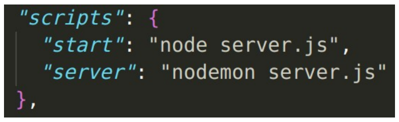


server.js U

_docs > server.js > ...

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

const app = express();

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

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

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

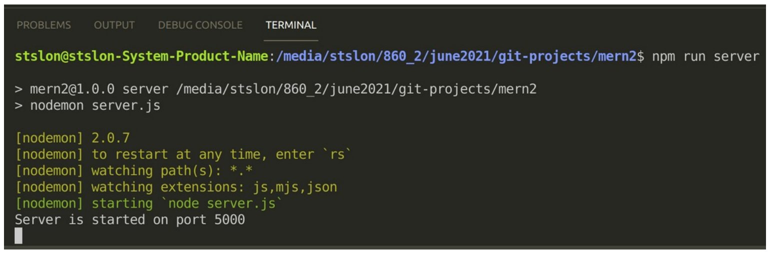
1.1.6 Change run script at package.json



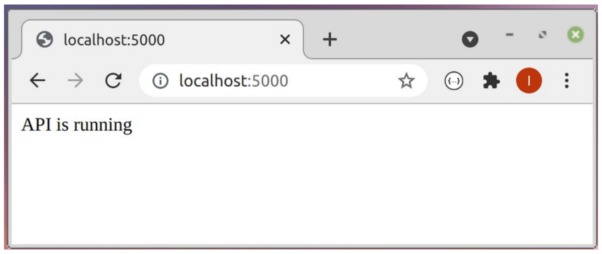
```
"scripts": {
  "start": "node server.js",
  "server": "nodemon server.js"
},
```

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

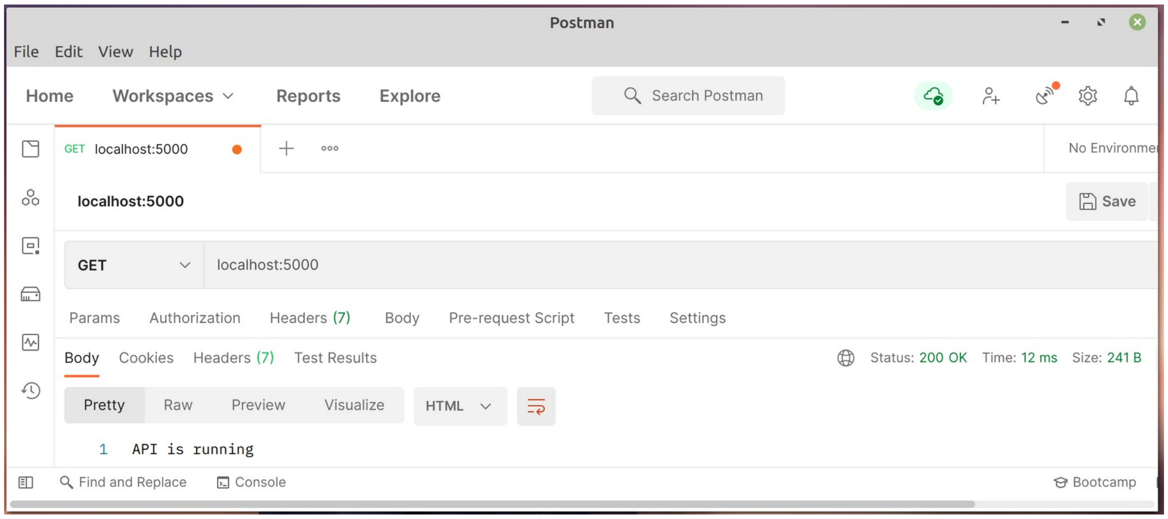
1.1.7 The first run



```
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`
Server is started on port 5000
```



API is running



localhost:5000

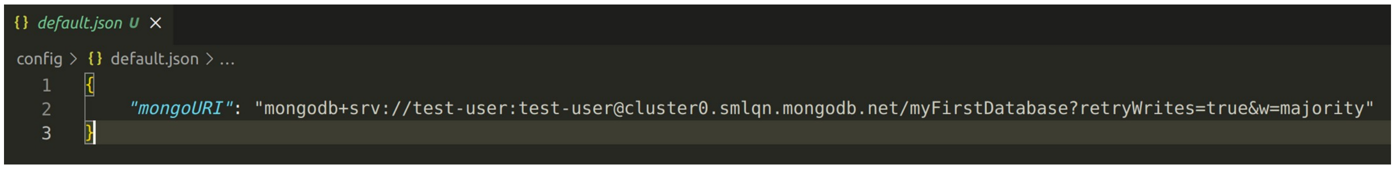
GET localhost:5000

Status: 200 OK Time: 12 ms Size: 241 B

```
API is running
```

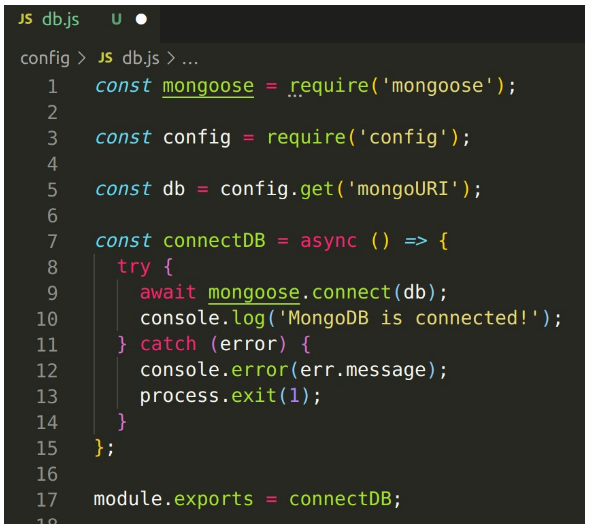
1.2 Create a connection with mongoDB.

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



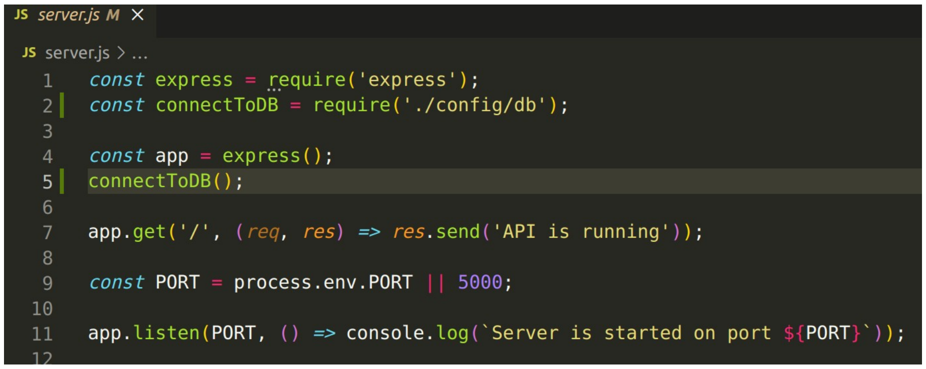
```json
{
    "mongoURI": "mongodb+srv://test-user:test-user@cluster0.smlqn.mongodb.net/myFirstDatabase?retryWrites=true&w=majority"
}
```

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

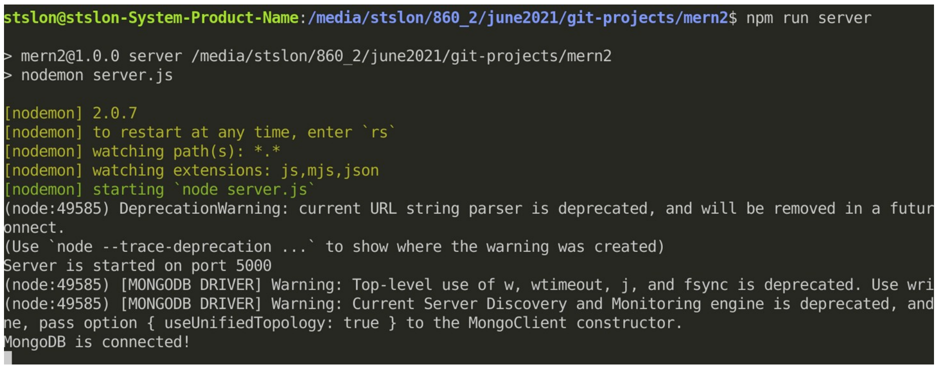


```javascript
const mongoose = require('mongoose');

const config = require('config');

const db = config.get('mongoURI');

const connectDB = async () => {
  try {
    await mongoose.connect(db);
    console.log('MongoDB is connected!');
  } catch (error) {
    console.error(err.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

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

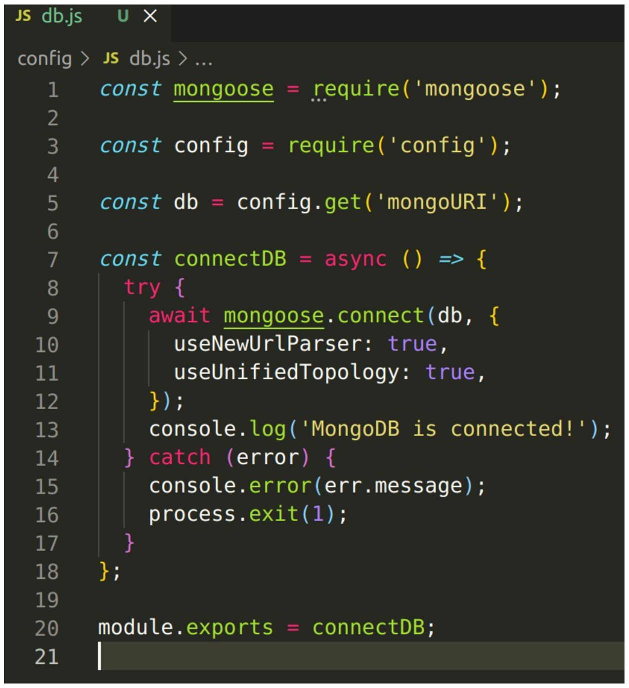


```javascript
const express = require('express');
const connectToDB = require('./config/db');

const app = express();
connectToDB();

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

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

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

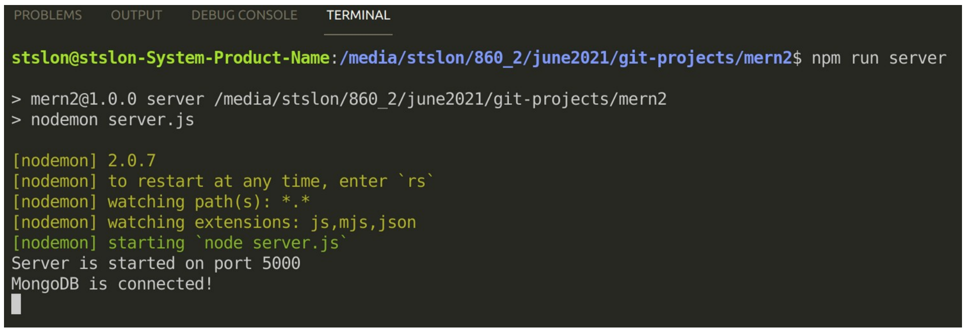


```
stslon@stslon-System-Product-Name:/media/stslon/860_2/june2021/git-projects/mern2$ npm run server

> 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:



```
config > 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);
    process.exit(1);
  }
};

module.exports = connectDB;
```

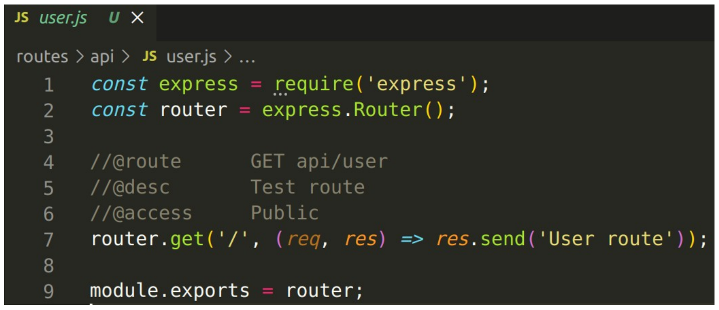


```
stslon@stslon-System-Product-Name:/media/stslon/860_2/june2021/git-projects/mern2$ npm run server

> 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`
Server is started on port 5000
MongoDB is connected!
```

1.3 Create routes.

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

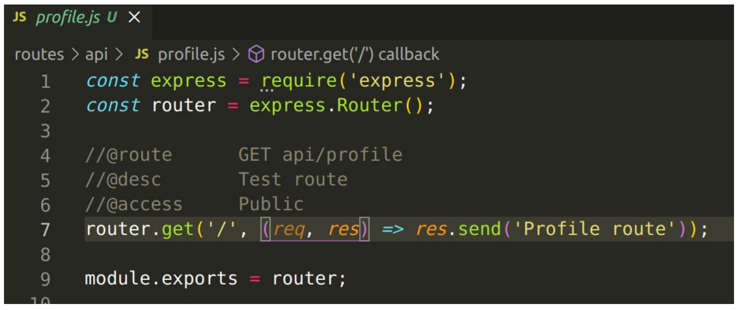
Inside this folder create new files – **user.js, profile.js, post.js and auth.js**



user.js U ×

routes > api > JS user.js > ...

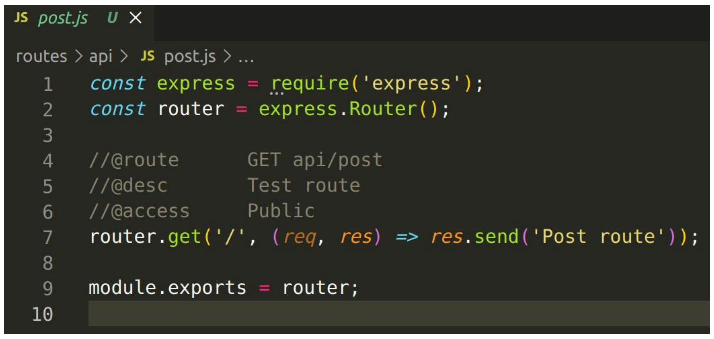
```
const express = require('express');
const router = express.Router();

//@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 > JS profile.js > router.get('/') callback

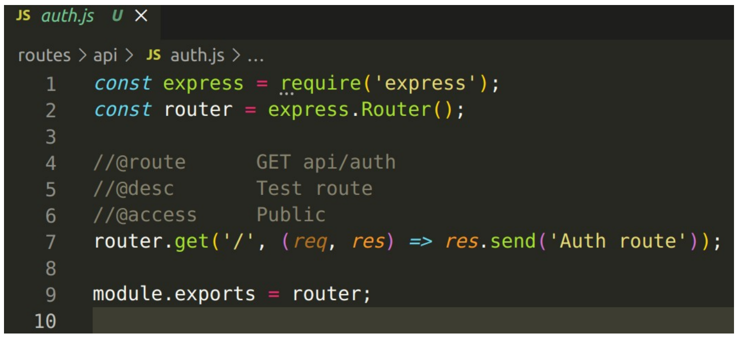
```
const express = require('express');
const router = express.Router();

//@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 > JS 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;

```

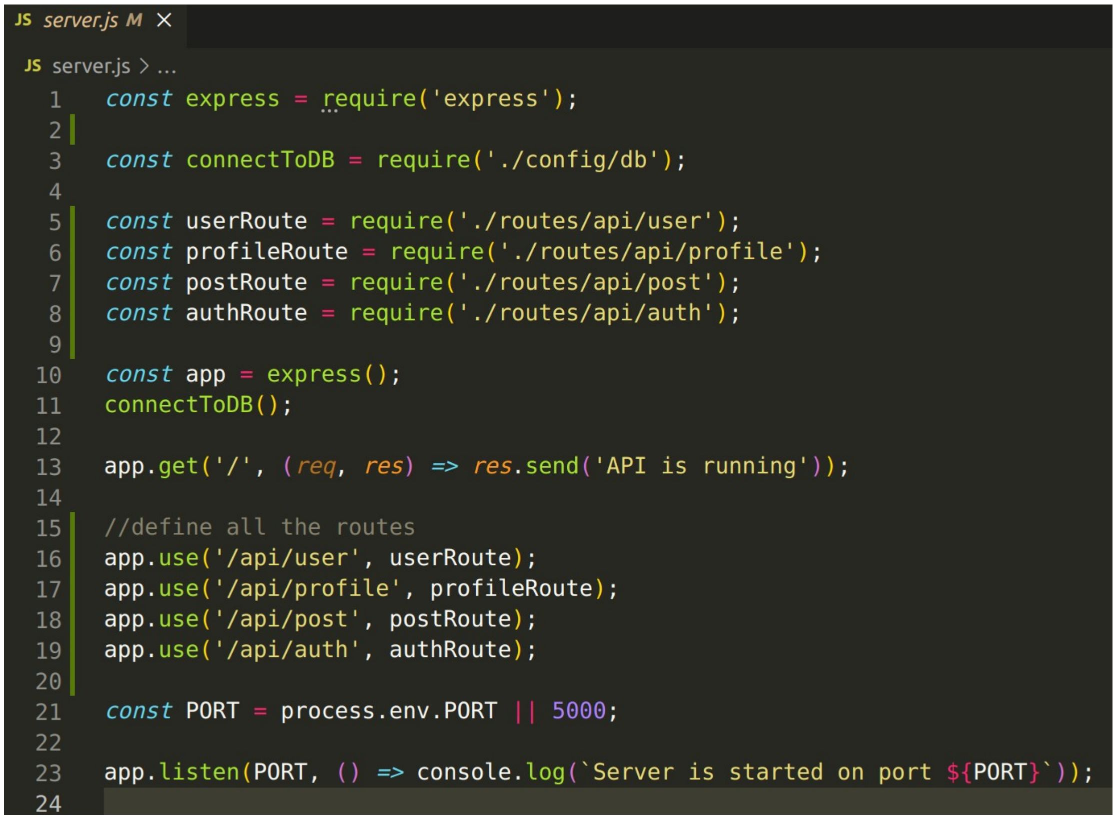


auth.js U ×

routes > api > JS 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;

```

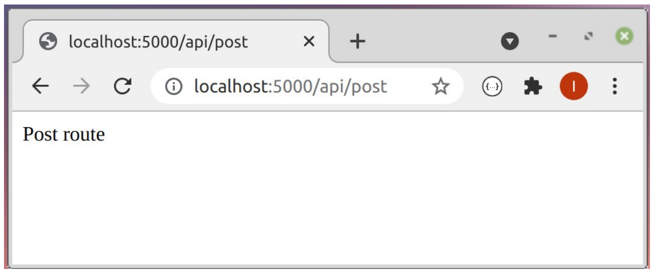
Add all these routes to **server.js**



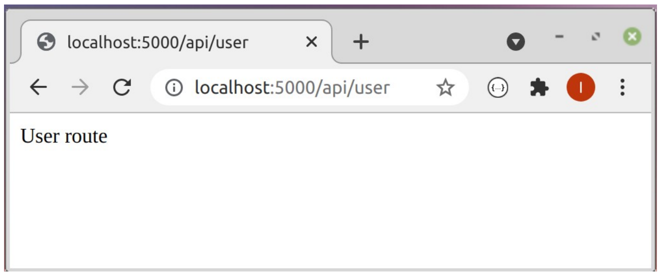
server.js M ×

JS server.js > ...

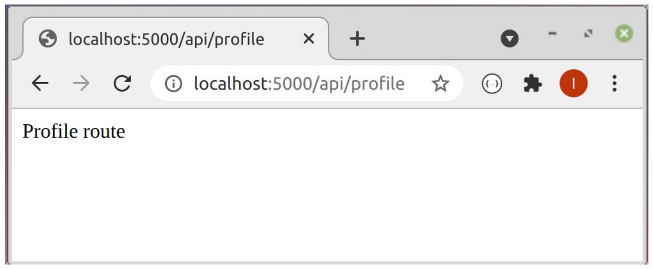
```
const express = require('express');

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}`));

```



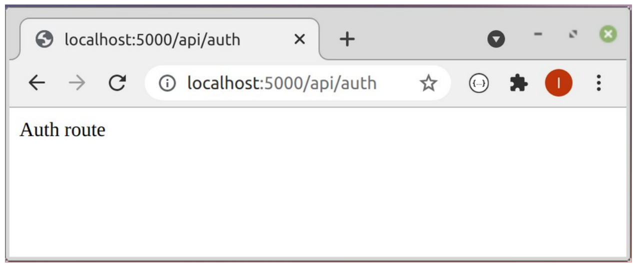
Post route



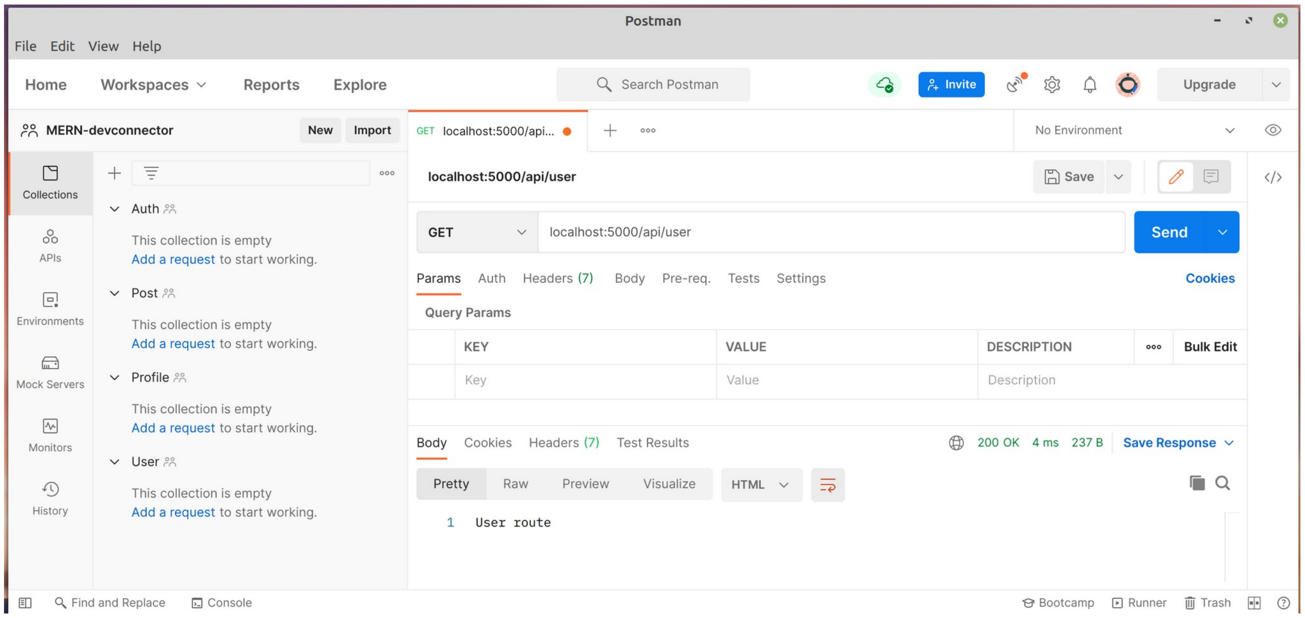
User route



Profile route



Auth route



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.

GET localhost:5000/api/user

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (7) Test Results 200 OK 4 ms 237 B

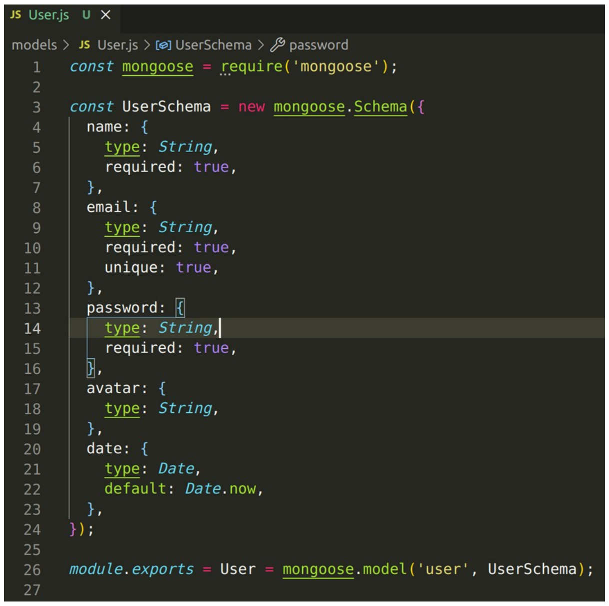
```
User route
```

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.



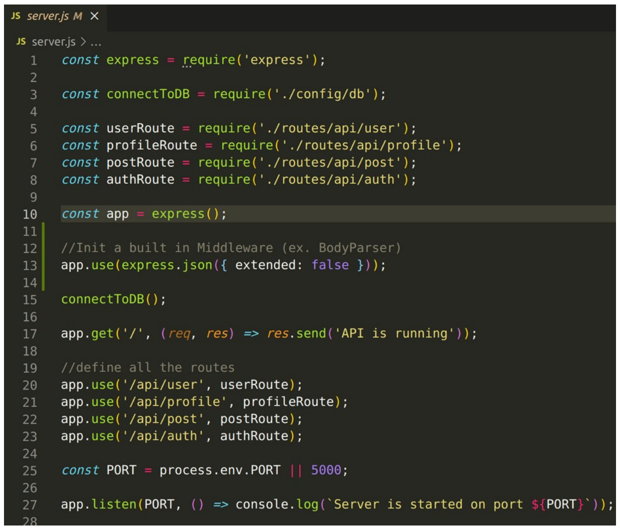
User.js U ×

models > User.js > UserSchema > password

```javascript
const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
  avatar: {
    type: String,
  },
  date: {
    type: Date,
    default: Date.now,
  },
});

module.exports = User = mongoose.model('user', UserSchema);
```

1.4.2 Edit test route to *api/*user endpoint

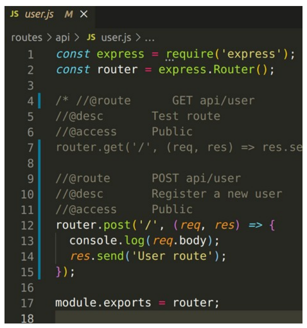
Add Middleware (ex. bodyParser) to work with request object. Change user.js route. Change request in Postman



server.js M ×

server.js > ...

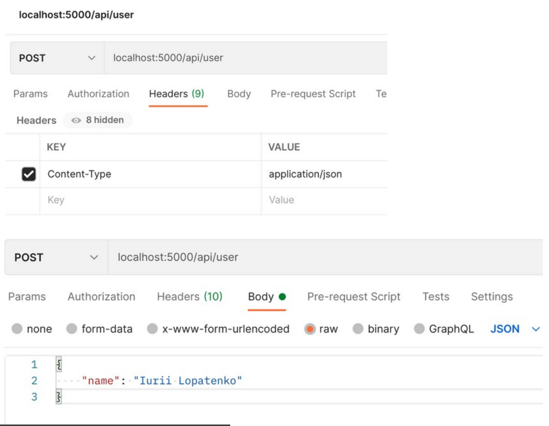
```javascript
const express = require('express');

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();

//Init a built in Middleware (ex. BodyParser)
app.use(express.json({ extended: false }));

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}`));
```



user.js M ×

routes > api > user.js > ...

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

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

//@route      POST api/user
//@desc       Register a new user
//@access     Public
router.post('/', (req, res) => {
  console.log(req.body);
  res.send('User route');
});

module.exports = router;
```



localhost:5000/api/user

POST localhost:5000/api/user

Params | Authorization | Headers (9) | Body | Pre-request Script | Te

Headers 8 hidden

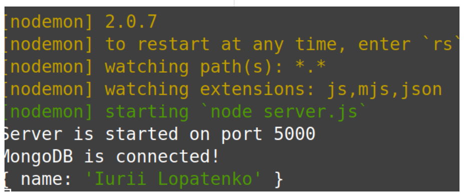
	KEY	VALUE
✓	Content-Type	application/json
	Key	Value

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

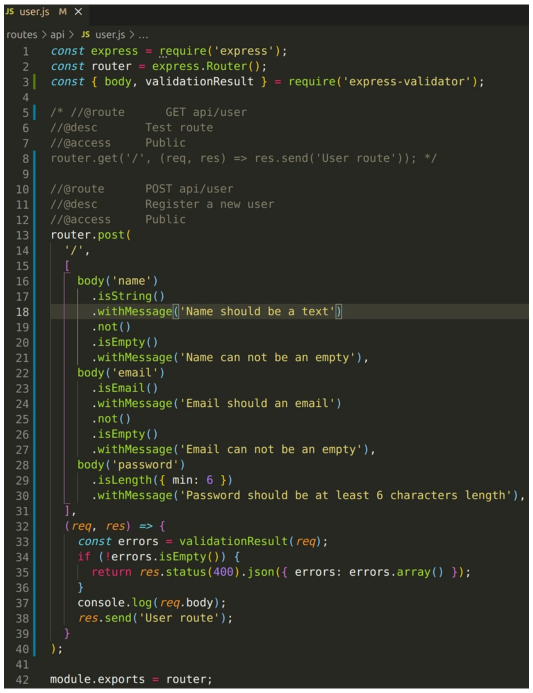
```json
{
    "name": "Iurii Lopatenko"
}
```



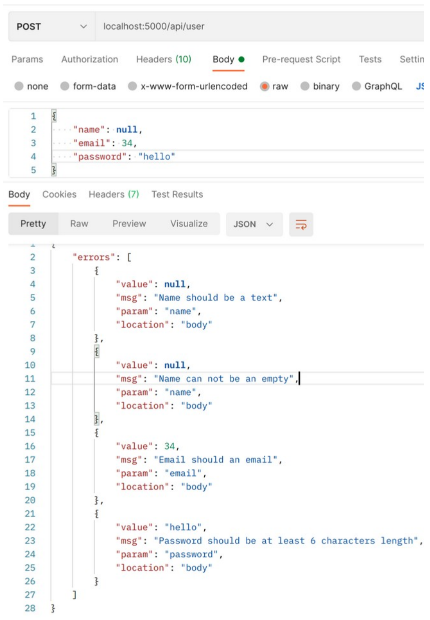
```
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
erver is started on port 5000
ongoDB is connected!
{ name: 'Iurii Lopatenko' }
```

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

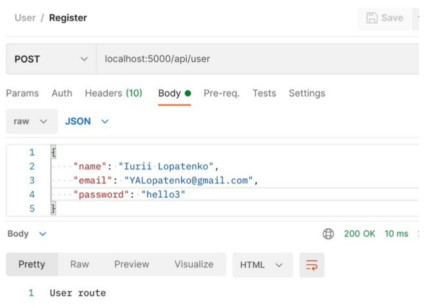


```javascript
const express = require('express');
const router = express.Router();
const { body, validationResult } = require('express-validator');

/* //@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 JS
Body Cookies Headers (7) Test Results
Pretty Raw Preview Visualize JSON

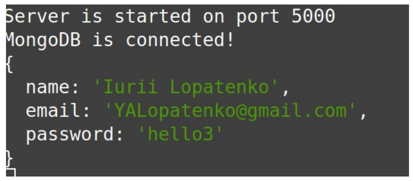
```json
{
    "name": null,
    "email": 34,
    "password": "hello"
}
```

```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
Body
200 OK 10 ms
Pretty Raw Preview Visualize HTML

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

```
User route
```



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

What was used in project?

Operating Systems: Linux Mint 20.1 and Windows 10

Node.js – javascript runtime enviroment.

NPM

GIT -

MongoDB

Mongoose

nodemon

postman