Node.js + MongoDB: User Authentication & Authorization with JWT

Last modified: December 10, 2021 (https://www.bezkoder.com/node-js-mongodb-auth-jwt/) bezkoder (https://www.bezkoder.com/author/bezkoder/) MongoDB (https://www.bezkoder.com/category/mongodb/), Node.js (https://www.bezkoder.com/category/node-js/), Security (https://www.bezkoder.com/category/security/)

In this tutorial, we're gonna build a Node.js & MongoDB example that supports User Authentication (Registation, Login) & Authorization with JSONWebToken (https://www.npmjs.com/package/jsonwebtoken) (JWT). You'll know:

- Appropriate Flow for User Signup & User Login with JWT Authentication
- Node.js Express Architecture with CORS, Authenticaton & Authorization middlewares, Mongoose ODM
- Way to configure Express routes to work with JWT
- How to define Mongoose Models for Authentication and Authorization
- How to use Mongoose to interact with MongoDB Database

Related Posts:
– Node.js & MongoDB: JWT Refresh Token example (https://www.bezkoder.com/jwt-refresh-token-node-js-mongodb/)
– MERN stack Authentication example (https://www.bezkoder.com/react-node-mongodb-auth/)
– MEAN stack Authentication with Angular 8 example (https://www.bezkoder.com/mean-stack-authentication-angular-8/)
– MEAN stack Authentication with Angular 10 example (https://www.bezkoder.com/mean-stack-authentication-angular-10/)
– MEAN stack Authentication with Angular 11 example (https://www.bezkoder.com/mean-stack-authentication-angular-11/)
– MEAN stack Authentication with Angular 12 example (https://www.bezkoder.com/mean-stack-auth-angular-12/)
– Node.js, Express & MongoDb: Build a CRUD Rest Api example (https://www.bezkoder.com/node-express-mongodb-crud-rest-api/)
– MongoDB One-to-Many Relationship tutorial with Mongoose examples (https://www.bezkoder.com/mongoose-one-to-many-relationship/)
– MongoDB Many-to-Many Relationship with Mongoose examples (https://www.bezkoder.com/mongodb-many-to-many-mongoose/)

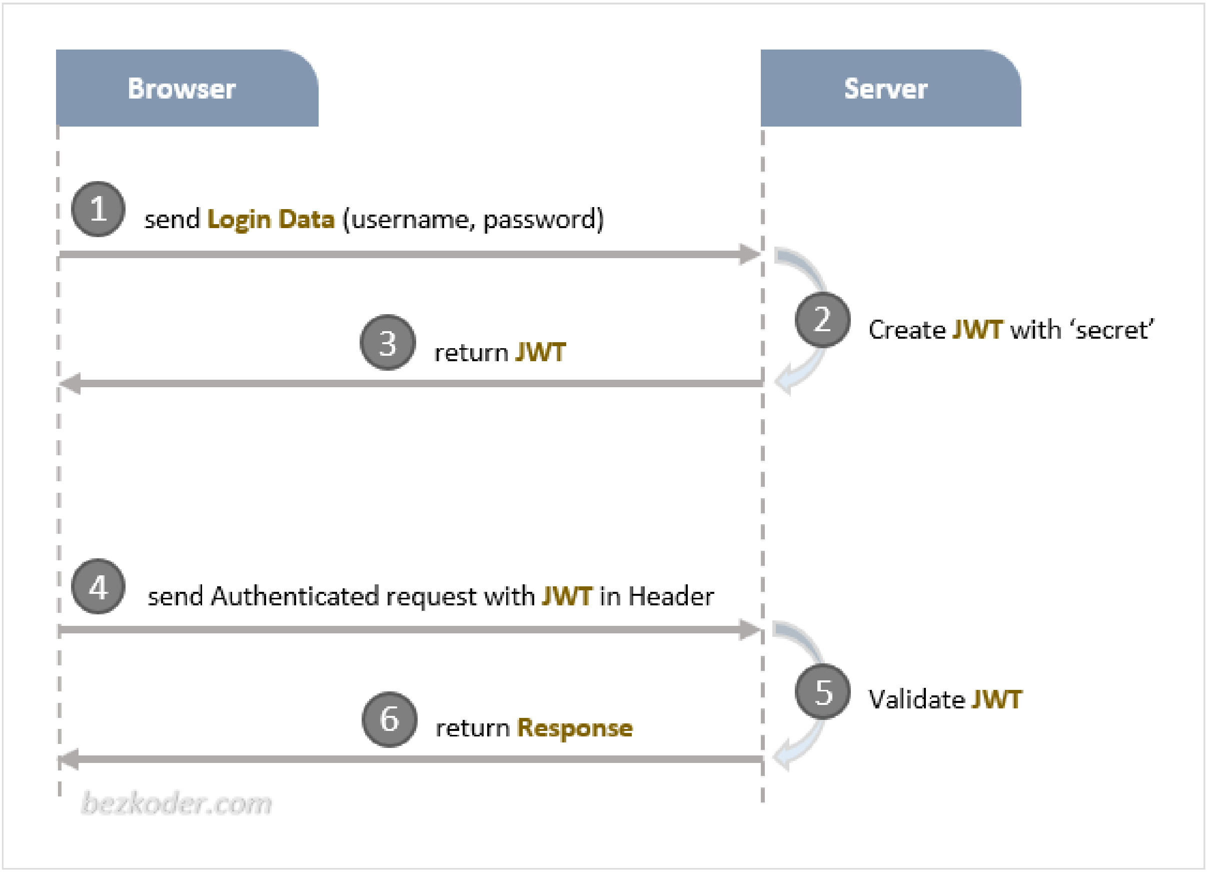
Deployment: Docker Compose: Node.js Express and MongoDB example (https://www.bezkoder.com/docker-compose-nodejs-mongodb/)

Front-end that works well with this:
– Vue.js JWT Authentication with Vuex and Vue Router (https://www.bezkoder.com/jwt-vue-vuex-authentication/)
– Angular 8 JWT Authentication example with Web Api (https://www.bezkoder.com/angular-jwt-authentication/)
– Angular 10 JWT Authentication example with Web Api (https://www.bezkoder.com/angular-10-jwt-auth/)
– Angular 11 JWT Authentication example with Web Api (https://www.bezkoder.com/angular-11-jwt-auth/)
– Angular 12 JWT Authentication example with Web Api (https://www.bezkoder.com/angular-12-jwt-auth/)
– Angular 13 JWT Authentication example with Web Api (https://www.bezkoder.com/angular-13-jwt-auth/)
– React JWT Authentication (without Redux) example (https://www.bezkoder.com/react-jwt-auth/)
– React Hooks: JWT Authentication (without Redux) example (https://www.bezkoder.com/react-hooks-jwt-auth/)
– React Redux: JWT Authentication example (https://www.bezkoder.com/react-redux-jwt-auth/)

Contents [hide]

Token Based Authentication

Comparing with Session-based Authentication that need to store Session on Cookie, the big advantage of Token-based Authentication is that we store the JSON Web Token (JWT) on Client side: Local Storage for Browser, Keychain for IOS and SharedPreferences for Android… So we don't need to build another backend project that supports Native Apps or an additional Authentication module for Native App users.


Browser
Server
1 send Login Data (username, password)
2 Create JWT with 'secret'
3 return JWT
4 send Authenticated request with JWT in Header
5 Validate JWT
6 return Response
bezkoder.com

There are three important parts of a JWT: Header, Payload, Signature. Together they are combined to a standard structure: `header.payload.signature` .

The Client typically attaches JWT in **Authorization** header with Bearer prefix:

```
Authorization: Bearer [header].[payload].[signature]
```

Or only in **x-access-token** header:

```
x-access-token: [header].[payload].[signature]
```

For more details, you can visit:
In-depth Introduction to JWT-JSON Web Token (https://bezkoder.com/jwt-json-web-token/)

Node.js & MongoDB User Authentication example

We will build a Node.js Express application in that:

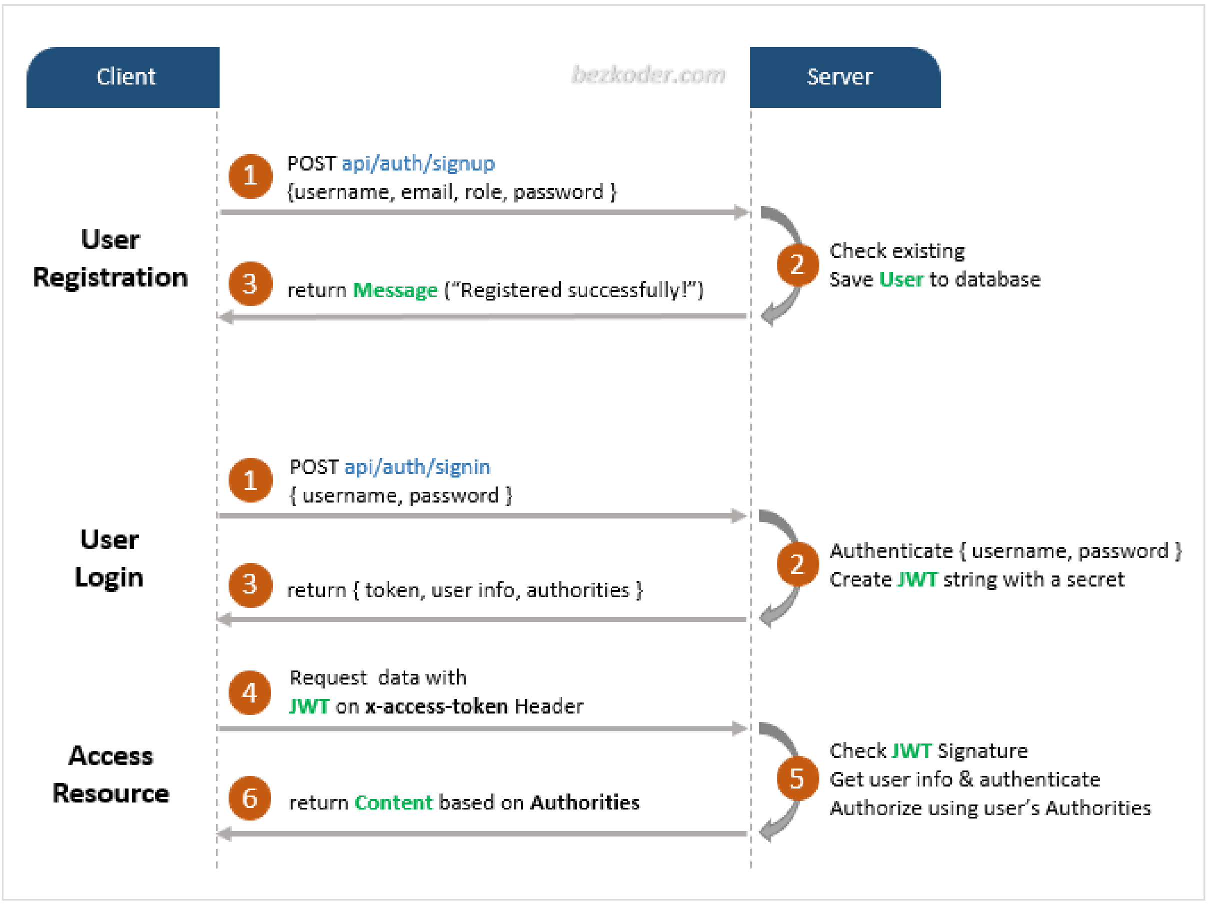
- User can signup new account, or login with username & password.
- By role (admin, moderator, user), the User has access to protected resources or not

These are APIs that we need to provide:

Methods	Urls	Actions
POST	/api/auth/signup	signup new account
POST	/api/auth/signin	login an account
GET	/api/test/all	retrieve public content
GET	/api/test/user	access User's content
GET	/api/test/mod	access Moderator's content
GET	/api/test/admin	access Admin's content

Flow for Signup & Login with JWT Authentication

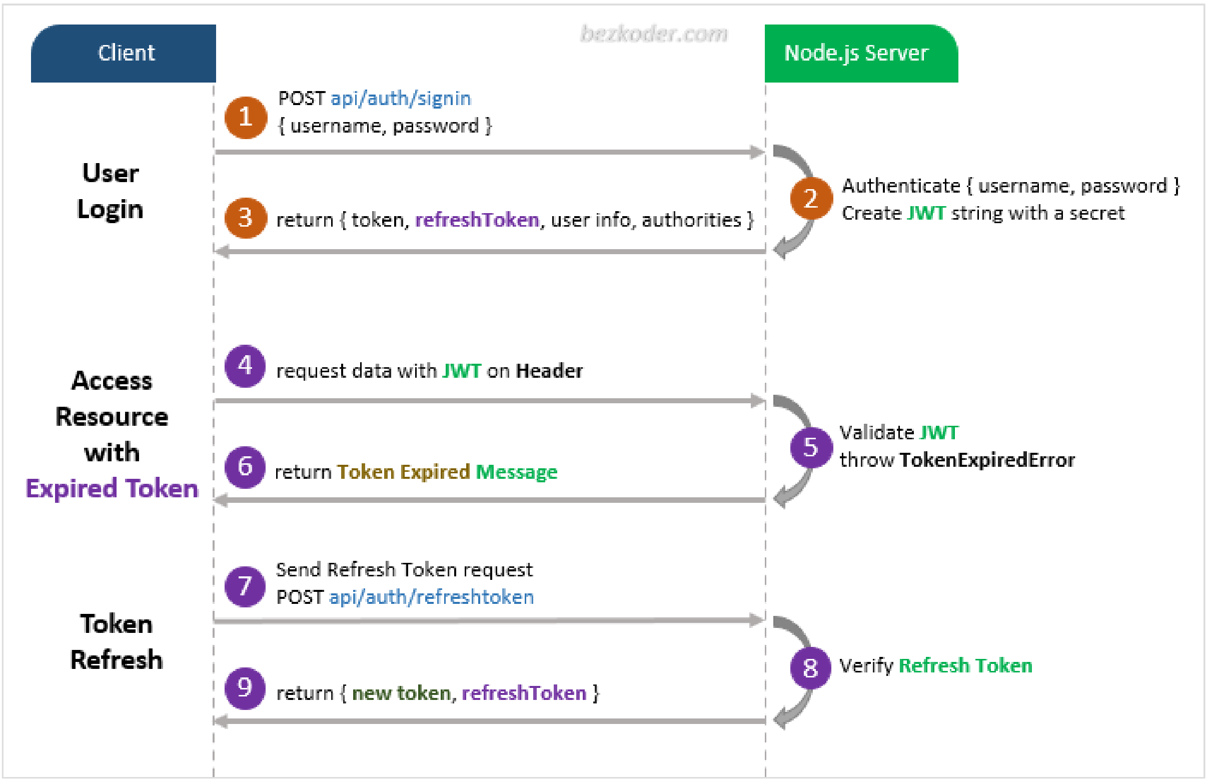
Following diagram shows you the flow that we're gonna implement for User Registration, User Login and Authorization process.


Client
bezkoder.com
Server
User Registration
1 POST api/auth/signup {username, email, role, password }
2 Check existing Save User to database
3 return Message ("Registered successfully!")
User Login
1 POST api/auth/signin { username, password }
2 Authenticate { username, password } Create JWT string with a secret
3 return { token, user info, authorities }
Access Resource
4 Request data with JWT on x-access-token Header
5 Check JWT Signature Get user info & authenticate Authorize using user's Authorities
6 return Content based on Authorities

A legal JWT must be added to HTTP **x-access-token** Header if Client accesses protected resources.

If you want to use Cookies, kindly visit:
Node.js Express and MongoDB: Login and Registration example (https://www.bezkoder.com/node-js-express-login-mongodb/)

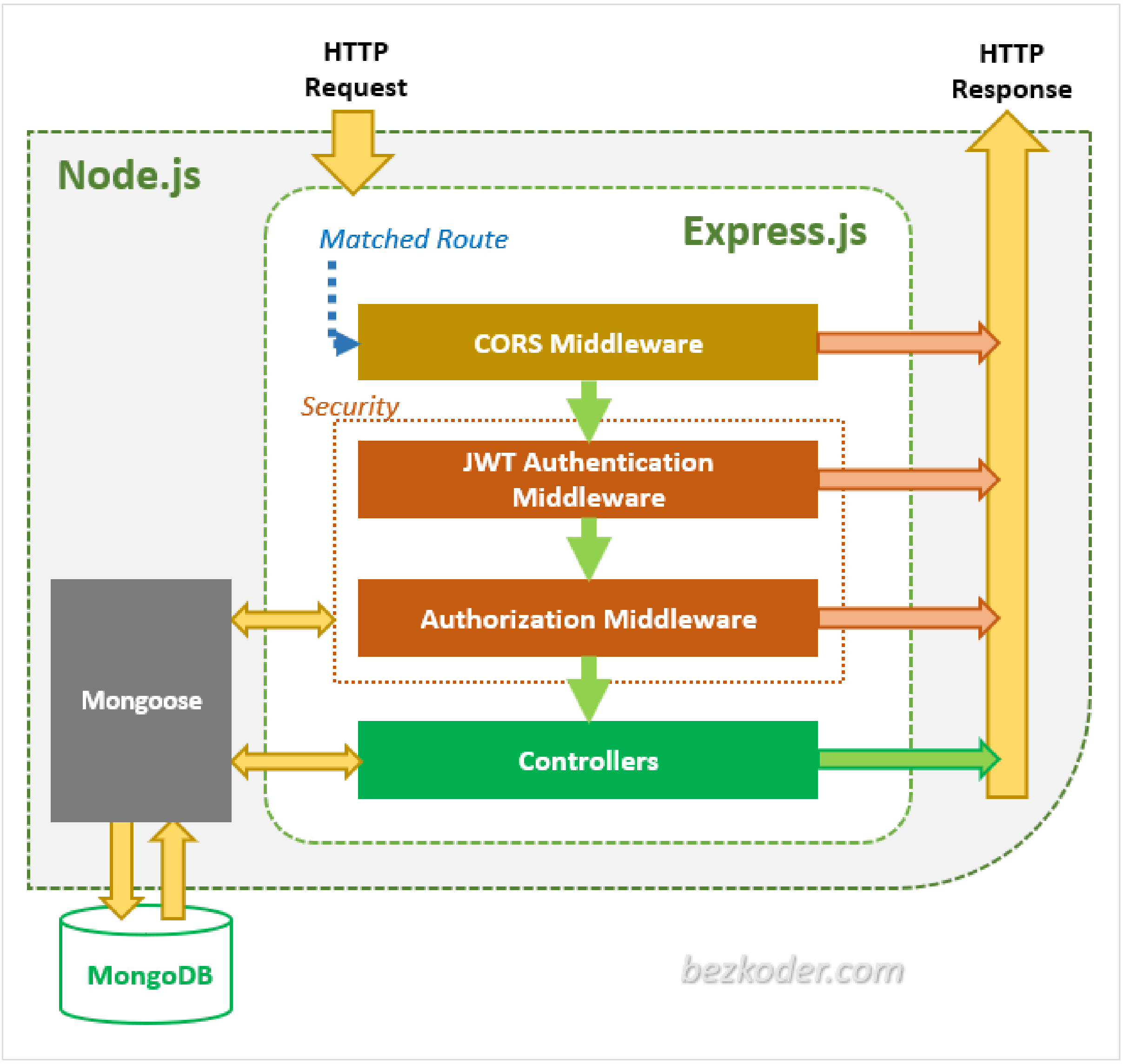
You will need to implement Refresh Token:


bezkoder.com
Client
Node.js Server
User Login
1 POST api/auth/signin { username, password }
2 Authenticate { username, password } Create JWT string with a secret
3 return { token, refreshToken, user info, authorities }
Access Resource with Expired Token
4 request data with JWT on Header
5 Validate JWT throw TokenExpiredError
6 return Token Expired Message
Token Refresh
7 Send Refresh Token request POST api/auth/refreshtoken
8 Verify Refresh Token
9 return { new token, refreshToken }

More details at: Node.js & MongoDB: JWT Refresh Token example (https://bezkoder.com/jwt-refresh-token-node-js-mongodb/)

Node.js Express Architecture with Authentication & Authorization

Here is an overview of our Node.js Express App:



Via *Express* routes, **HTTP request** that matches a route will be checked by **CORS Middleware** before coming to **Security** layer.

Security layer includes:

- JWT Authentication Middleware: verify SignUp, verify token
- Authorization Middleware: check User's roles with record in database

An error message will be sent as HTTP response to Client when the middlewares throw any error, .

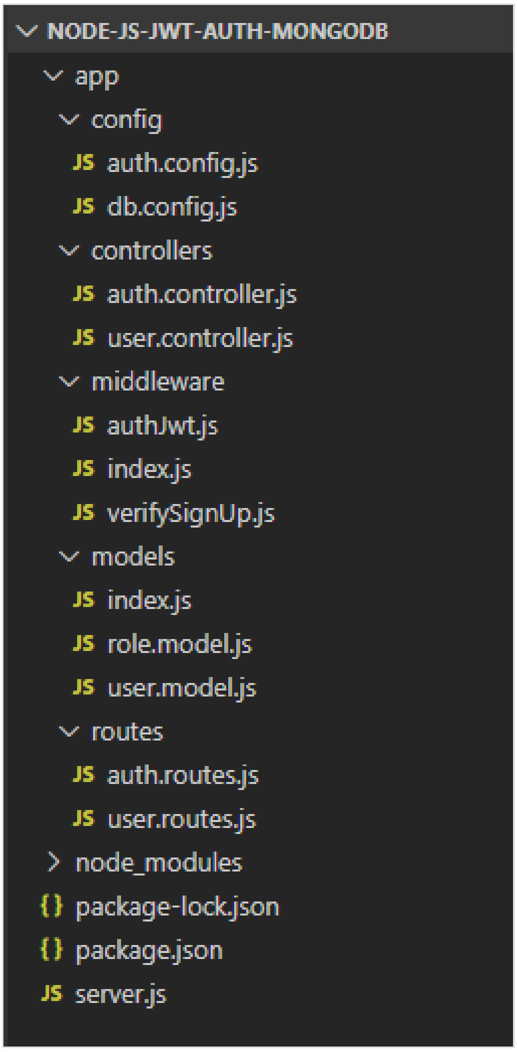
Controllers interact with MongoDB Database via *Mongoose* library and send **HTTP response** (token, user information, data based on roles…) to Client.

Technology

- Express 4.17.1
- bcryptjs 2.4.3
- jsonwebtoken 8.5.1
- mongoose 5.9.1
- MongoDB

Project Structure

This is directory structure for our Node.js Express & MongoDB application:



```
NODE-JS-JWT-AUTH-MONGODB
  app
    config
      auth.config.js
      db.config.js
    controllers
      auth.controller.js
      user.controller.js
    middleware
      authJwt.js
      index.js
      verifySignUp.js
    models
      index.js
      role.model.js
      user.model.js
    routes
      auth.routes.js
      user.routes.js
  node_modules
  package-lock.json
  package.json
  server.js
```

Create Node.js App

Create a folder for our project with command:

```
$ mkdir node-js-jwt-auth-mongodb
$ cd node-js-jwt-auth-mongodb
```

Then we initialize the Node.js App with a *package.json* file:

```
npm init
name: (node-js-jwt-auth-mongodb) 
version: (1.0.0) 
description: Node.js + MongoDB: JWT Authentication & Authorization
entry point: (index.js) server.js
test command: 
git repository: 
keywords: node.js, express, jwt, authentication, mongodb
author: bezkoder
license: (ISC)
Is this ok? (yes) yes
```

Let's install necessary modules such as: `express`, `cors`, `mongoose`, `jsonwebtoken` and `bcryptjs`.
Run the command:

```
npm install express mongoose cors jsonwebtoken bcryptjs --save
```

Check *package.json* file, you can see it looks like this:

```
{
  "name": "node-js-jwt-auth-mongodb",
  "version": "1.0.0",
  "description": "Node.js + MongoDB: JWT Authentication & Authorization",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "node.js",
    "express",
    "jwt",
    "authentication",
    "mongodb"
  ],
  "author": "bezkoder",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "jsonwebtoken": "^8.5.1",
    "mongoose": "^5.9.1"
  }
}
```

Setup Express web server

In the root folder, let's create a new *server.js* file:

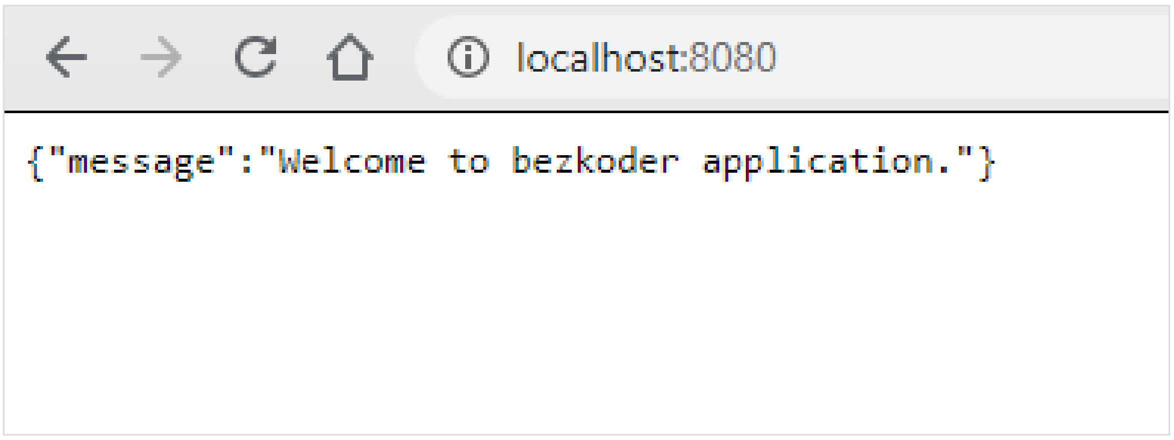
```
const express = require("express");
const cors = require("cors");
const app = express();
var corsOptions = {
  origin: "http://localhost:8081"
};
app.use(cors(corsOptions));
// parse requests of content-type - application/json
app.use(express.json());
// parse requests of content-type - application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));
// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to bezkoder application." });
});
// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

What we've just done in the code above:
– import `express` and `cors` modules:

- Express is for building the Rest apis
- cors (https://www.npmjs.com/package/cors) provides Express middleware to enable CORS

– create an Express app, then add request body parser and `cors` middlewares using `app.use()` method. Notice that we set origin: `http://localhost:8081`.
– define a GET route which is simple for test.
– listen on port 8080 for incoming requests.

Now let's run the app with command: `node server.js`.
Open your browser with url http://localhost:8080/ (http://localhost:8080/), you will see:



```
{"message":"Welcome to bezkoder application."}
```

Configure MongoDB database

In the **app** folder, create **config** folder for configuration.

Then create a new *db.config.js* file that contains parameters for setting up MongoDB later:

```
module.exports = {
  HOST: "localhost",
  PORT: 27017,
  DB: "bezkoder_db"
};
```

Define the Mongoose Model

In *models* folder, create `User` and `Role` data model as following code:

models/*role.model.js*

```
const mongoose = require("mongoose");
const Role = mongoose.model(
  "Role",
  new mongoose.Schema({
    name: String
  })
);
module.exports = Role;
```

models/*user.model.js*

```
const mongoose = require("mongoose");
const User = mongoose.model(
  "User",
  new mongoose.Schema({
    username: String,
    email: String,
    password: String,
    roles: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Role"
      }
    ]
  })
);
module.exports = User;
```

These Mongoose Models represents **users** & **roles** collections in MongoDB database. `User` object will have a `roles` array that contains ids in **roles** collection as reference.

This kind is called *Reference Data Models* or *Normalization*. You can find more details at:
MongoDB One-to-Many Relationship tutorial with Mongoose examples (https://bezkoder.com/mongoose-one-to-many-relationship/)

After initializing Mongoose, we don't need to write CRUD functions because Mongoose supports all of them:

- create a new User: object.save() (https://mongoosejs.com/docs/api/model.html#model_Model-save)
- find a User by id: User.findById(id) (https://mongoosejs.com/docs/api.html#model_Model.findById)
- find User by email: User.findOne (https://mongoosejs.com/docs/api.html#model_Model.findOne)({ email: … })
- find User by username: User.findOne({ username: … })
- find all Roles which name in given `roles` array: Role.find (https://mongoosejs.com/docs/api.html#model_Model.find)({ name: { $in: roles } })

These functions will be used in our Controllers and Middlewares.

Initialize Mongoose

Now create **app**/**models**/*index.js* with content like this:

```
const mongoose = require('mongoose');
mongoose.Promise = global.Promise;
const db = {};
db.mongoose = mongoose;
db.user = require("./user.model");
db.role = require("./role.model");
db.ROLES = ["user", "admin", "moderator"];
module.exports = db;
```

Open *server.js* and add following code to open Mongoose connection to MongoDB database:

```
...
const app = express();
app.use(...);
const db = require("./app/models");
const Role = db.role;
db.mongoose
  .connect(`mongodb://${dbConfig.HOST}:${dbConfig.PORT}/${dbConfig.DB}`, {
    useNewUrlParser: true,
    useUnifiedTopology: true
  })
  .then(() => {
    console.log("Successfully connect to MongoDB.");
    initial();
  })
  .catch(err => {
    console.error("Connection error", err);
    process.exit();
  });
...
function initial() {
  Role.estimatedDocumentCount((err, count) => {
    if (!err && count === 0) {
      new Role({
        name: "user"
      }).save(err => {
        if (err) {
          console.log("error", err);
        }
        console.log("added 'user' to roles collection");
      });
      new Role({
        name: "moderator"
      }).save(err => {
        if (err) {
          console.log("error", err);
        }
        console.log("added 'moderator' to roles collection");
      });
      new Role({
        name: "admin"
      }).save(err => {
        if (err) {
          console.log("error", err);
        }
        console.log("added 'admin' to roles collection");
      });
    }
  });
}
```

`initial()` function helps us to create 3 important rows in `roles` collection.

Configure Auth Key

jsonwebtoken functions such as `verify()` or `sign()` use algorithm that needs a secret key (as String) to encode and decode token.

In the **app**/**config** folder, create *auth.config.js* file with following code:

```
module.exports = {
  secret: "bezkoder-secret-key"
};
```

You can create your own `secret` String.

Create Middleware functions

To verify a Signup action, we need 2 functions:
– check duplications for `username` and `email`
– check if `roles` in the request is legal or not

middlewares/*verifySignUp.js*

```
const db = require("../models");
const ROLES = db.ROLES;
const User = db.user;
checkDuplicateUsernameOrEmail = (req, res, next) => {
  // Username
  User.findOne({
    username: req.body.username
  }).exec((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
      return;
    }
    if (user) {
      res.status(400).send({ message: "Failed! Username is already in use!" 
      return;
    }
    // Email
    User.findOne({
      email: req.body.email
    }).exec((err, user) => {
      if (err) {
        res.status(500).send({ message: err });
        return;
      }
      if (user) {
        res.status(400).send({ message: "Failed! Email is already in use!" }
        return;
      }
      next();
    });
  });
};
checkRolesExisted = (req, res, next) => {
  if (req.body.roles) {
    for (let i = 0; i < req.body.roles.length; i++) {
      if (!ROLES.includes(req.body.roles[i])) {
        res.status(400).send({
          message: `Failed! Role ${req.body.roles[i]} does not exist!`
        });
        return;
      }
    }
  }
  next();
};
const verifySignUp = {
  checkDuplicateUsernameOrEmail,
  checkRolesExisted
};
module.exports = verifySignUp;
```

To process Authentication & Authorization, we create following functions:
- check if `token` is provided, legal or not. We get token from **x-access-token** of HTTP headers, then use **jsonwebtoken**'s `verify()` function
- check if `roles` of the user contains required role or not

middlewares/*authJwt.js*

```
const jwt = require("jsonwebtoken");
const config = require("../config/auth.config.js");
const db = require("../models");
const User = db.user;
const Role = db.role;
verifyToken = (req, res, next) => {
  let token = req.headers["x-access-token"];
  if (!token) {
    return res.status(403).send({ message: "No token provided!" });
  }
  jwt.verify(token, config.secret, (err, decoded) => {
    if (err) {
      return res.status(401).send({ message: "Unauthorized!" });
    }
    req.userId = decoded.id;
    next();
  });
};
isAdmin = (req, res, next) => {
  User.findById(req.userId).exec((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
      return;
    }
    Role.find(
      {
        _id: { $in: user.roles }
      },
      (err, roles) => {
        if (err) {
          res.status(500).send({ message: err });
          return;
        }
        for (let i = 0; i < roles.length; i++) {
          if (roles[i].name === "admin") {
            next();
            return;
          }
        }
        res.status(403).send({ message: "Require Admin Role!" });
        return;
      }
    );
  });
};
isModerator = (req, res, next) => {
  User.findById(req.userId).exec((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
```

```
        return;
      }
      Role.find(
        {
          _id: { $in: user.roles }
        },
        (err, roles) => {
          if (err) {
            res.status(500).send({ message: err });
            return;
          }
          for (let i = 0; i < roles.length; i++) {
            if (roles[i].name === "moderator") {

              next();
              return;
            }
          }
          res.status(403).send({ message: "Require Moderator Role!" });
          return;
        }
      );
    });
};
const authJwt = {
  verifyToken,
  isAdmin,
  isModerator
};
module.exports = authJwt;
```

middlewares/*index.js*

```
const authJwt = require("./authJwt");
const verifySignUp = require("./verifySignUp");
module.exports = {
  authJwt,
  verifySignUp
};
```

Create Controllers

Controller for Authentication

There are 2 main functions for Authentication:
- `signup`: create new User in database (role is **user** if not specifying role)
- `signin`:

- find `username` of the request in database, if it exists
- compare `password` with `password` in database using **bcrypt**, if it is correct
- generate a token using **jsonwebtoken**
- return user information & access Token

controllers/*auth.controller.js*

```
const config = require("../config/auth.config");
const db = require("../models");
const User = db.user;
const Role = db.role;
var jwt = require("jsonwebtoken");
var bcrypt = require("bcryptjs");
exports.signup = (req, res) => {
  const user = new User({
    username: req.body.username,
    email: req.body.email,
    password: bcrypt.hashSync(req.body.password, 8)
  });
  user.save((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
      return;
    }
    if (req.body.roles) {
      Role.find(
        {
          name: { $in: req.body.roles }
        },
        (err, roles) => {
          if (err) {
            res.status(500).send({ message: err });
            return;
          }
          user.roles = roles.map(role => role._id);
          user.save(err => {
            if (err) {
              res.status(500).send({ message: err });
              return;
            }
            res.send({ message: "User was registered successfully!" });
          });
        }
      );
    } else {
      Role.findOne({ name: "user" }, (err, role) => {
        if (err) {
          res.status(500).send({ message: err });
          return;
        }
        user.roles = [role._id];
        user.save(err => {
          if (err) {
            res.status(500).send({ message: err });
            return;
          }
          res.send({ message: "User was registered successfully!" });
        });
      });
    }
  });
};
exports.signin = (req, res) => {
  User.findOne({
    username: req.body.username
  })
    .populate("roles", "-__v")
    .exec((err, user) => {
      if (err) {

        res.status(500).send({ message: err });
        return;
      }
      if (!user) {
        return res.status(404).send({ message: "User Not found." });
      }
      var passwordIsValid = bcrypt.compareSync(
        req.body.password,
        user.password
      );
      if (!passwordIsValid) {
        return res.status(401).send({
          accessToken: null,
          message: "Invalid Password!"
        });
      }
      var token = jwt.sign({ id: user.id }, config.secret, {
        expiresIn: 86400 // 24 hours
      });
      var authorities = [];
      for (let i = 0; i < user.roles.length; i++) {
        authorities.push("ROLE_" + user.roles[i].name.toUpperCase());
      }
      res.status(200).send({
        id: user._id,
        username: user.username,
        email: user.email,
        roles: authorities,
        accessToken: token
      });
    });
};
```

Controller for testing Authorization

There are 4 functions:
– `/api/test/all` for public access
– `/api/test/user` for loggedin users (any role)
– `/api/test/mod` for **moderator** users
– `/api/test/admin` for **admin** users

controllers/*user.controller.js*

```
exports.allAccess = (req, res) => {
  res.status(200).send("Public Content.");
};
exports.userBoard = (req, res) => {
  res.status(200).send("User Content.");
};
exports.adminBoard = (req, res) => {
  res.status(200).send("Admin Content.");
};
exports.moderatorBoard = (req, res) => {
  res.status(200).send("Moderator Content.");
};
```

Let's combine middlewares with controller functions in the next section.

Define Routes

When a client sends request for an endpoint using HTTP request (GET, POST, PUT, DELETE), we need to determine how the server will response by setting up the routes.

We can separate our routes into 2 part: for Authentication and for Authorization (accessing protected resources).

Authentication:

- POST `/api/auth/signup`
- POST `/api/auth/signin`

routes/*auth.routes.js*

```
const { verifySignUp } = require("../middlewares");
const controller = require("../controllers/auth.controller");
module.exports = function(app) {
  app.use(function(req, res, next) {
    res.header(
      "Access-Control-Allow-Headers",
      "x-access-token, Origin, Content-Type, Accept"
    );
    next();
  });
  app.post(
    "/api/auth/signup",
    [
      verifySignUp.checkDuplicateUsernameOrEmail,
      verifySignUp.checkRolesExisted
    ],
    controller.signup
  );
  app.post("/api/auth/signin", controller.signin);
};
```

Authorization:

- GET `/api/test/all`
- GET `/api/test/user` for loggedin users (user/moderator/admin)
- GET `/api/test/mod` for moderator
- GET `/api/test/admin` for admin

routes/*user.routes.js*

```
const { authJwt } = require("../middlewares");
const controller = require("../controllers/user.controller");
module.exports = function(app) {
  app.use(function(req, res, next) {
    res.header(
      "Access-Control-Allow-Headers",
      "x-access-token, Origin, Content-Type, Accept"
    );
    next();
  });
  app.get("/api/test/all", controller.allAccess);
  app.get("/api/test/user", [authJwt.verifyToken], controller.userBoard);
  app.get(
    "/api/test/mod",
    [authJwt.verifyToken, authJwt.isModerator],
    controller.moderatorBoard
  );
  app.get(
    "/api/test/admin",
    [authJwt.verifyToken, authJwt.isAdmin],
    controller.adminBoard
  );
};
```

Don't forget to add these routes in *server.js*:

```
...
// routes
require('./app/routes/auth.routes')(app);
require('./app/routes/user.routes')(app);
// set port, listen for requests
...
```

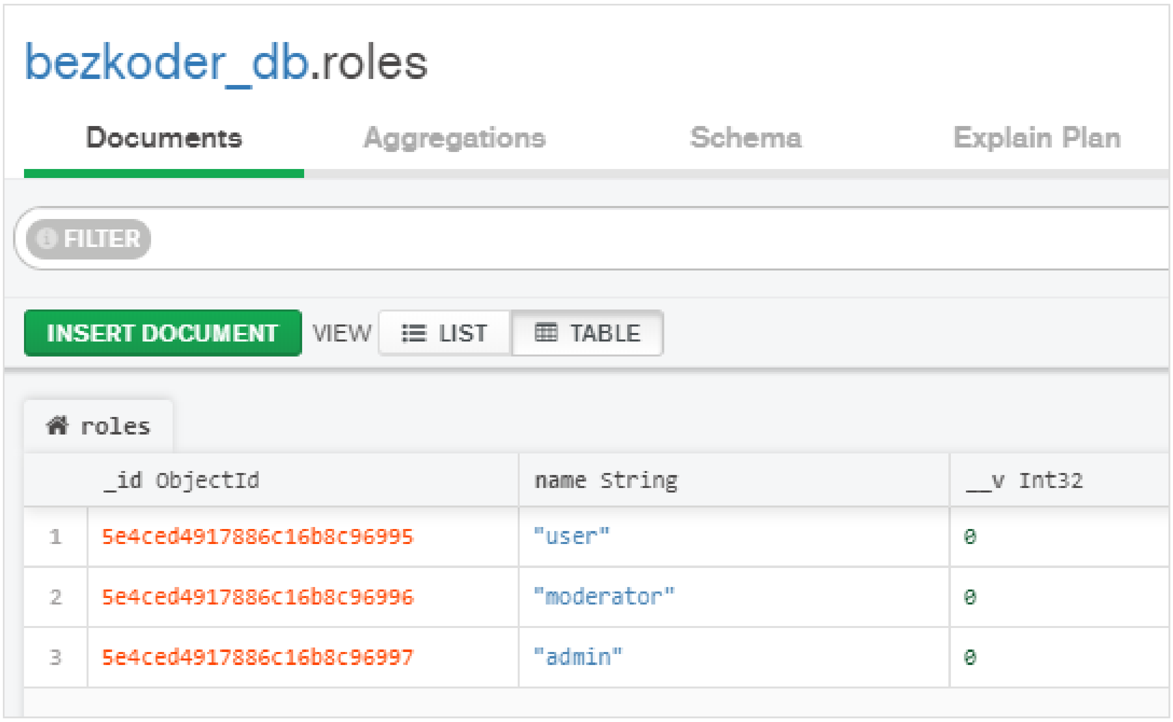
Run & Test with Results

Run Node.js application with command: `node server.js`.

The console shows:

```
Server is running on port 8080.
Successfully connect to MongoDB.
added 'user' to roles collection
added 'admin' to roles collection
added 'moderator' to roles collection
```

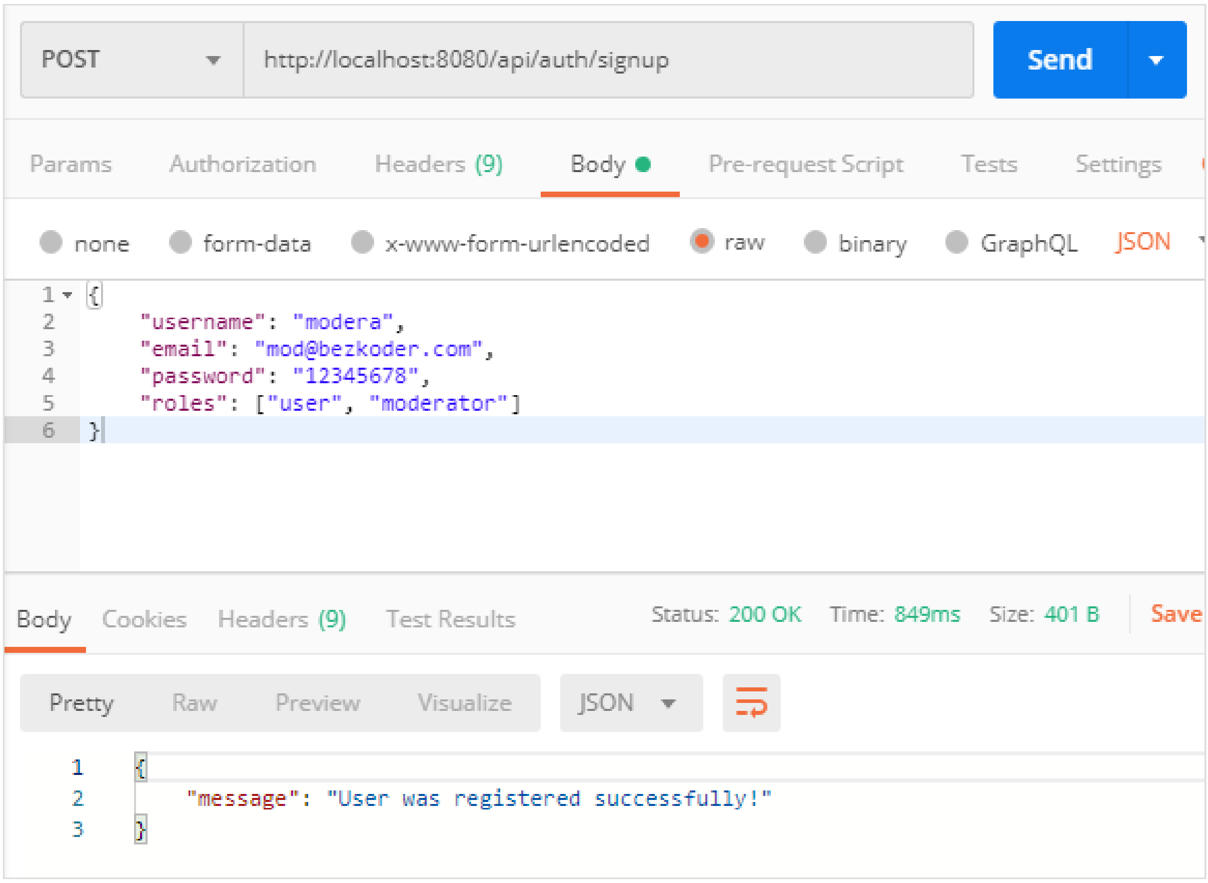
Let's check `roles` collection in MongoDB database:



	_id ObjectId	name String	__v Int32
1	5e4ced4917886c16b8c96995	"user"	0
2	5e4ced4917886c16b8c96996	"moderator"	0
3	5e4ced4917886c16b8c96997	"admin"	0

Register some users with `/signup` API:

- **admin** with `admin` role
- **modera** with `moderator` and `user` roles
- **bezkoder** with `user` role



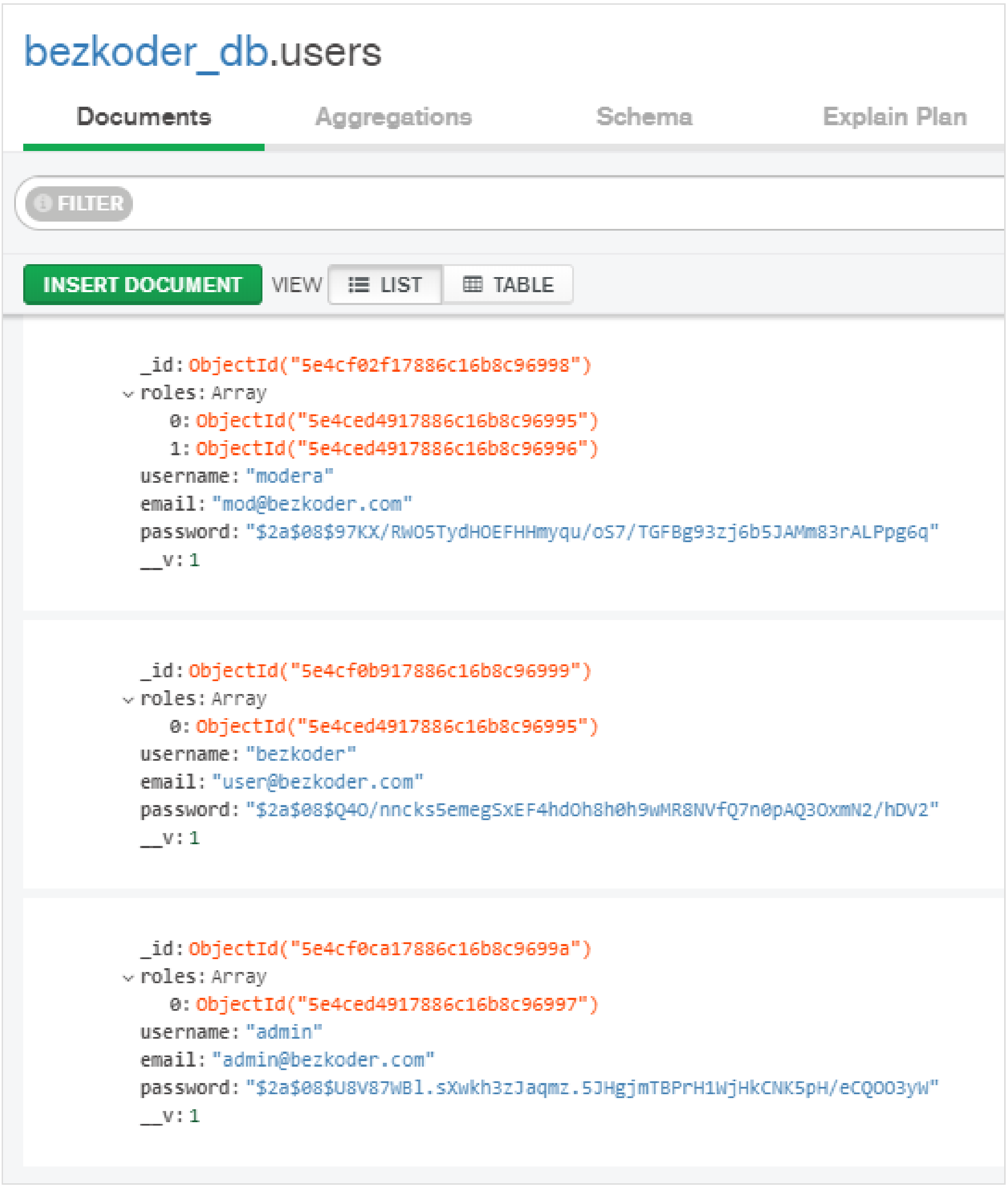
```
http://localhost:8080/api/auth/signup
```

```
{
    "username": "modera",
    "email": "mod@bezkoder.com",
    "password": "12345678",
    "roles": ["user", "moderator"]
}
```

Status: 200 OK Time: 849ms Size: 401 B

```
{
    "message": "User was registered successfully!"
}
```

`users` collection after signup could look like this.

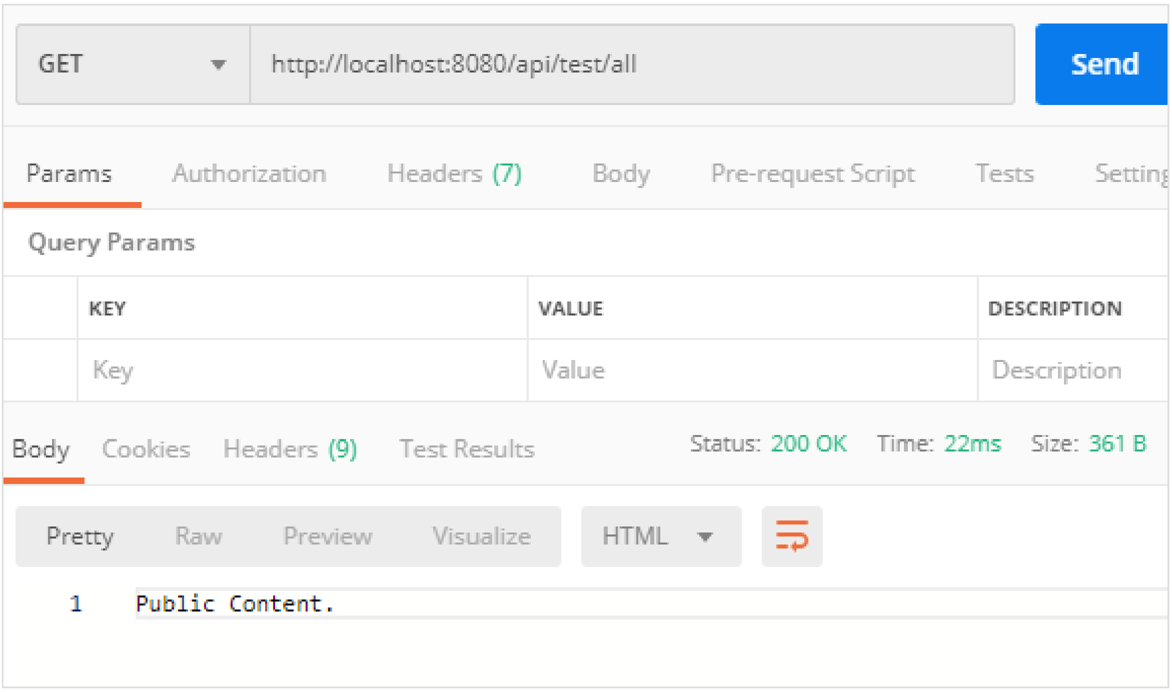


```
  _id: ObjectId("5e4cf02f17886c16b8c96998")
⌄ roles: Array
     0: ObjectId("5e4ced4917886c16b8c96995")
     1: ObjectId("5e4ced4917886c16b8c96996")
  username: "modera"
  email: "mod@bezkoder.com"
  password: "$2a$08$97KX/RWO5TydHOEFHHmyqu/oS7/TGFBg93zj6b5JAMm83rALPpg6q"
  __v: 1
```

```
  _id: ObjectId("5e4cf0b917886c16b8c96999")
⌄ roles: Array
     0: ObjectId("5e4ced4917886c16b8c96995")
  username: "bezkoder"
  email: "user@bezkoder.com"
  password: "$2a$08$Q4O/nncks5emegSxEF4hdOh8h0h9wMR8NVfQ7n0pAQ3OxmN2/hDV2"
  __v: 1
```

```
  _id: ObjectId("5e4cf0ca17886c16b8c9699a")
⌄ roles: Array
     0: ObjectId("5e4ced4917886c16b8c96997")
  username: "admin"
  email: "admin@bezkoder.com"
  password: "$2a$08$U8V87WBl.sXwkh3zJaqmz.5JHgjmTBPrH1WjHkCNK5pH/eCQOO3yW"
  __v: 1
```

Access public resource: GET `/api/test/all`

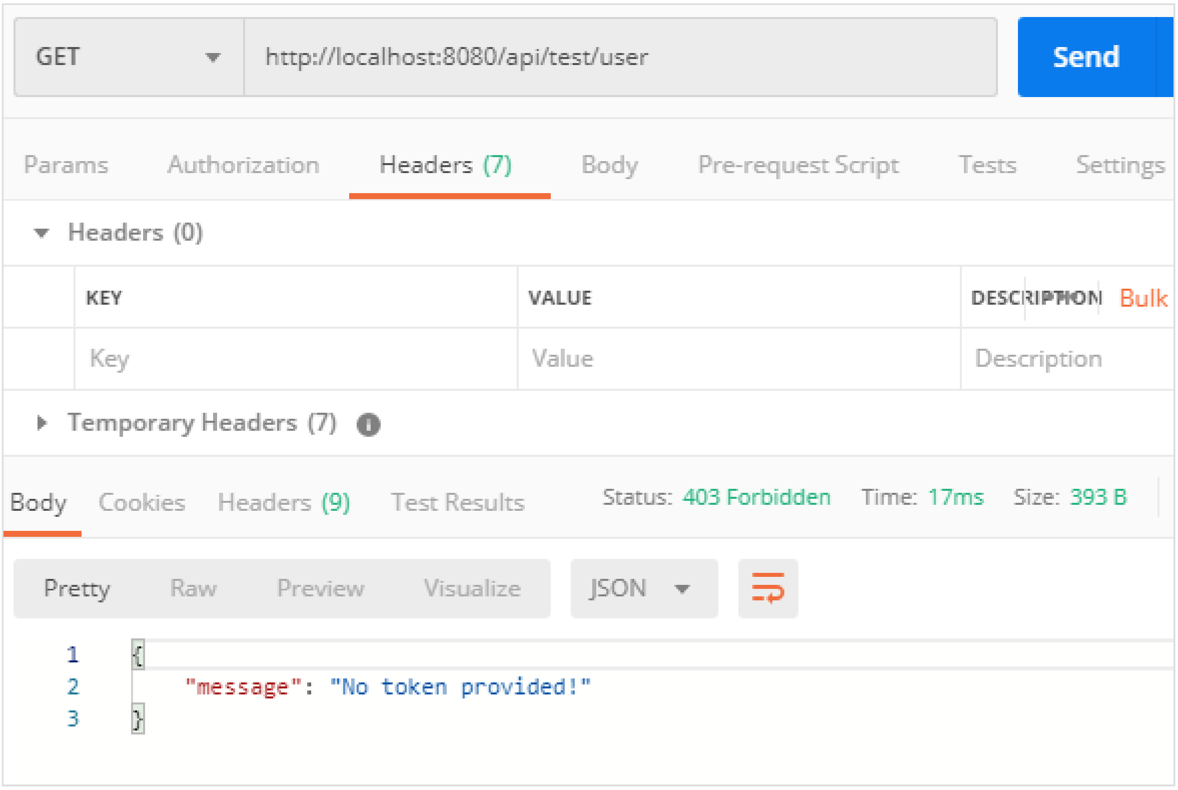


http://localhost:8080/api/test/all

Status: 200 OK Time: 22ms Size: 361 B

```
Public Content.
```

Access protected resource: GET `/api/test/user`

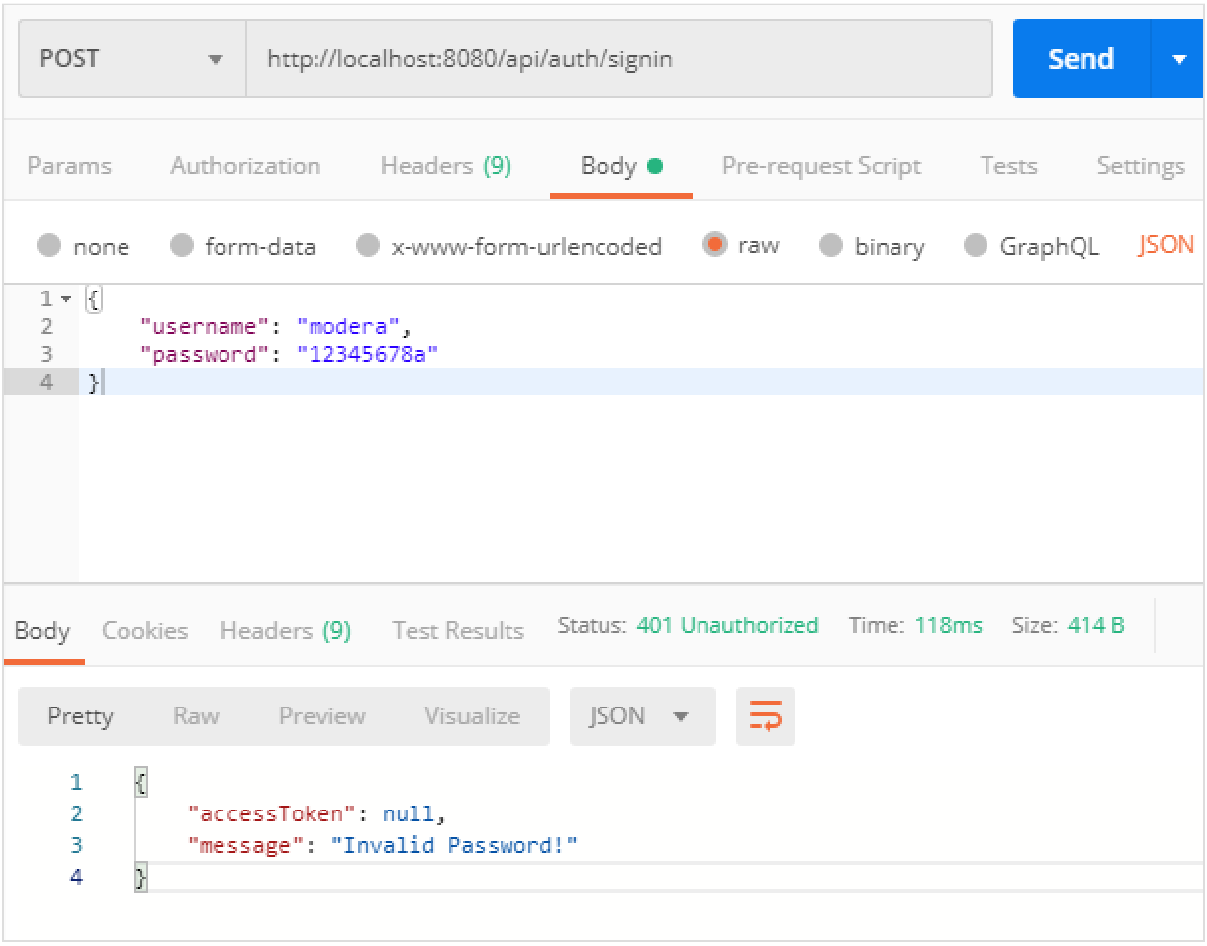


http://localhost:8080/api/test/user

Status: 403 Forbidden Time: 17ms Size: 393 B

```
{
    "message": "No token provided!"
}
```

Login an account (with wrong password): POST `/api/auth/signin`



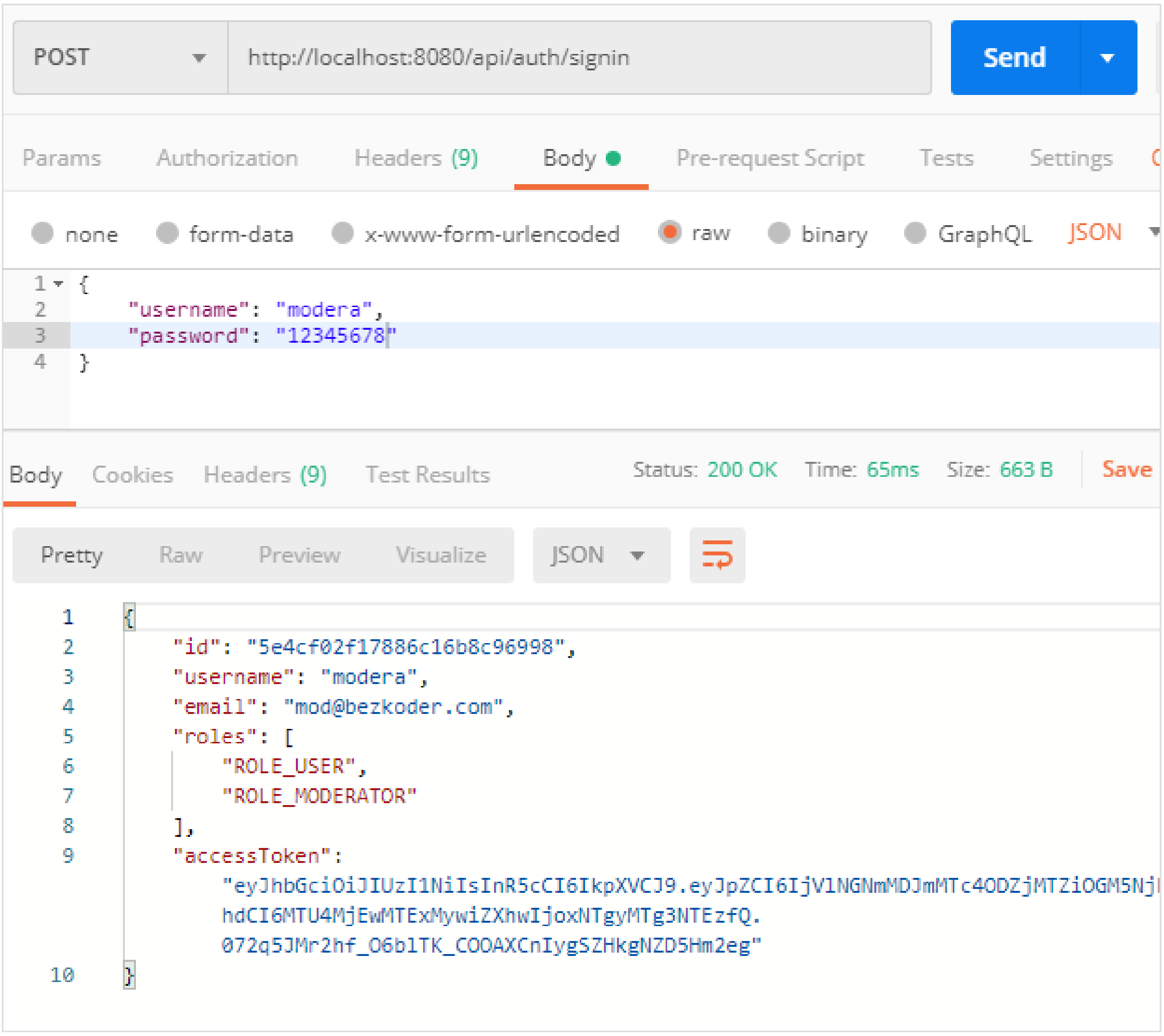
POST http://localhost:8080/api/auth/signin

```
{
    "username": "modera",
    "password": "12345678a"
}
```

Status: 401 Unauthorized Time: 118ms Size: 414 B

```
{
    "accessToken": null,
    "message": "Invalid Password!"
}
```

Login an legal account: POST `/api/auth/signin`



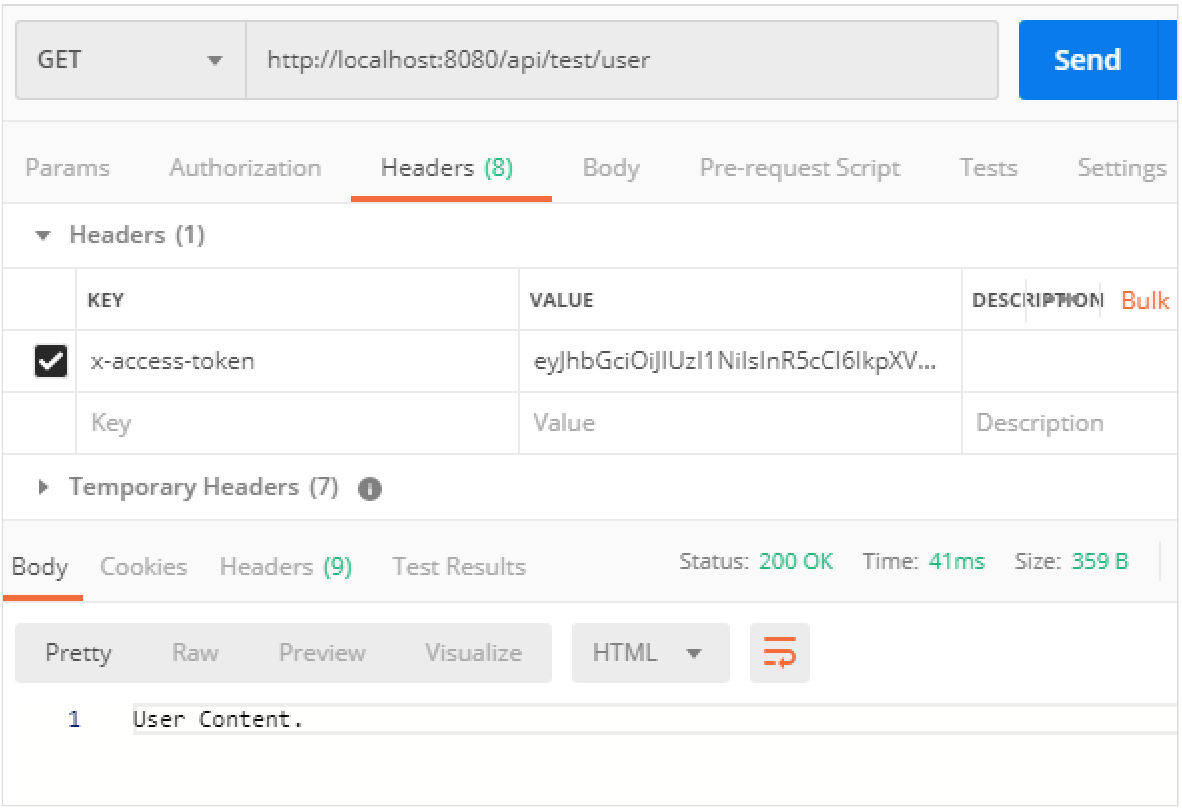
http://localhost:8080/api/auth/signin

```
{
    "username": "modera",
    "password": "12345678"
}
```

Status: 200 OK Time: 65ms Size: 663 B

```
{
    "id": "5e4cf02f17886c16b8c96998",
    "username": "modera",
    "email": "mod@bezkoder.com",
    "roles": [
        "ROLE_USER",
        "ROLE_MODERATOR"
    ],
    "accessToken":
        "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjV1NGNmMDJmMTc4ODZjMTZiOGM5Nj
        hdCI6MTU4MjEwMTExMywiZXhwIjoxNTgyMTg3NTEzfQ.
        072q5JMr2hf_O6blTK_COOAXCnIygSZHkgNZD5Hm2eg"
}
```

Access protected resources: GET `/api/test/user`



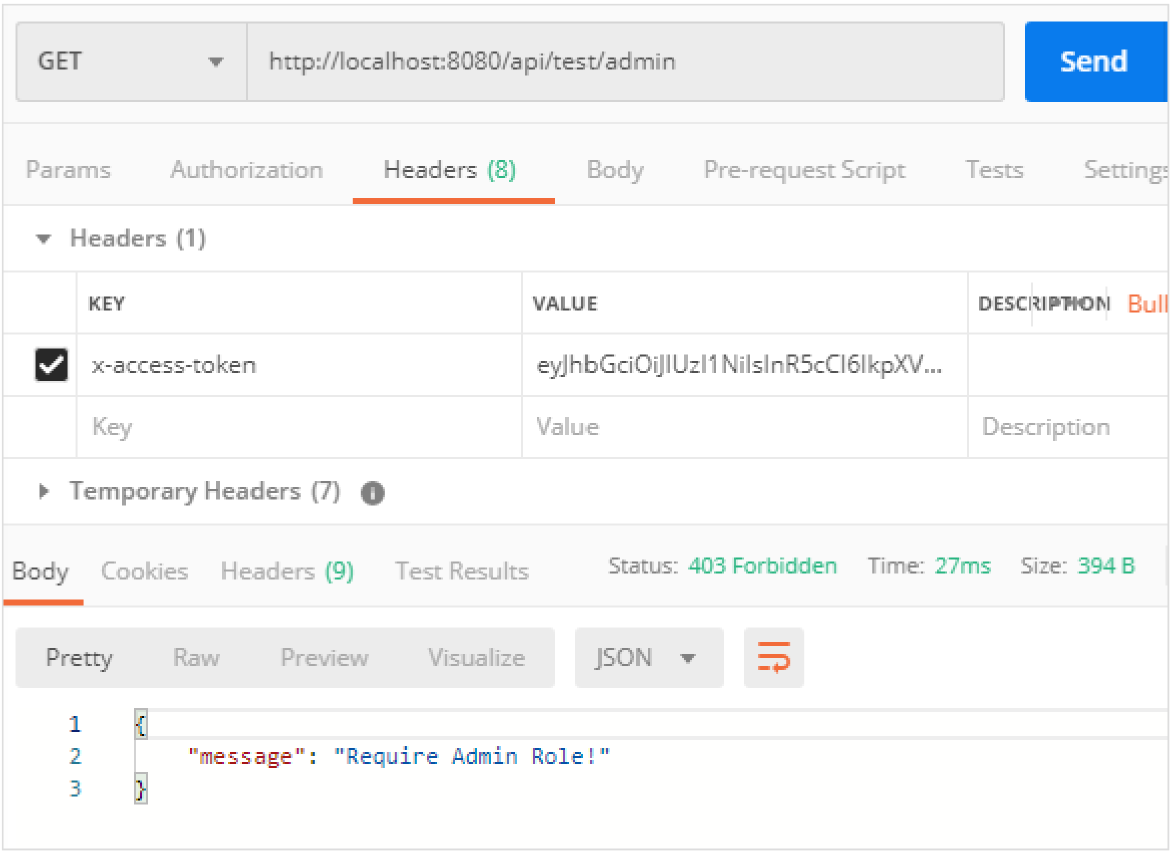
http://localhost:8080/api/test/user

	KEY	VALUE	DESCRIPTION
✓	x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV...	
	Key	Value	Description

Status: 200 OK Time: 41ms Size: 359 B

```
User Content.
```

GET `/api/test/admin`



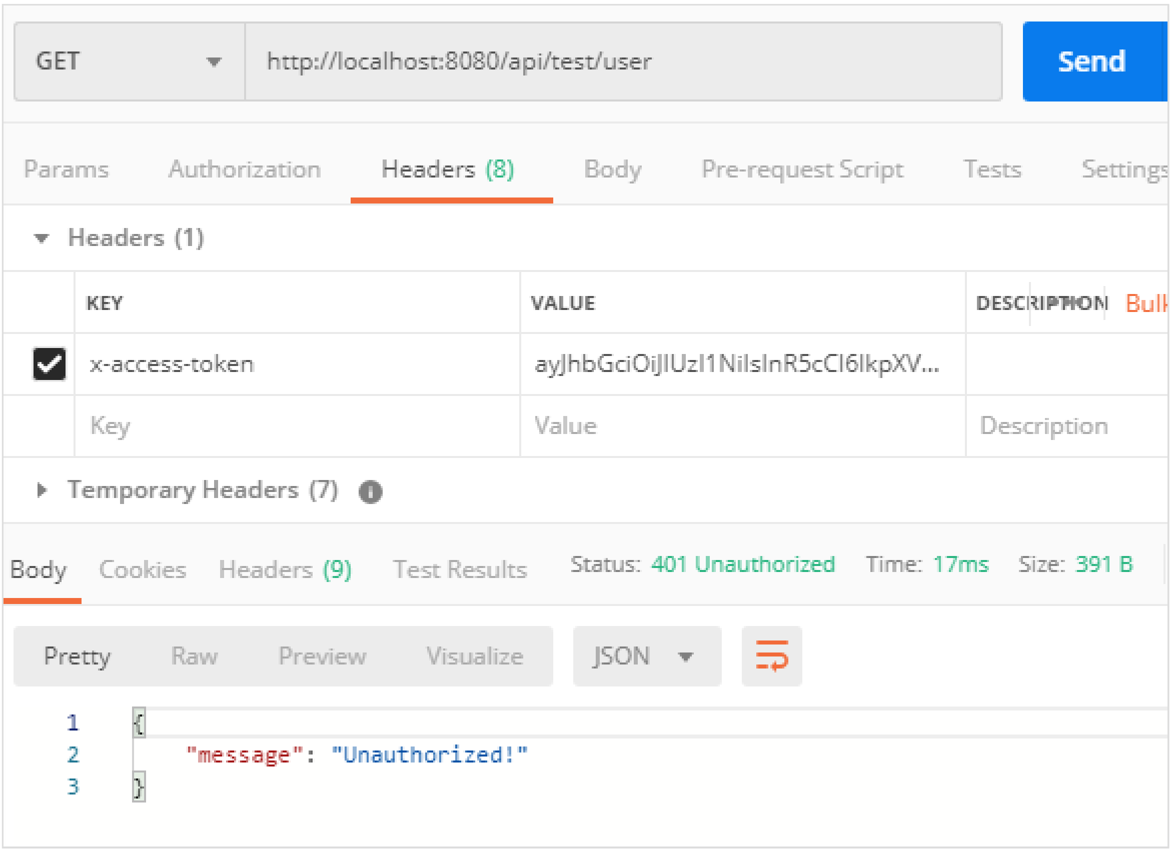
http://localhost:8080/api/test/admin

KEY	VALUE	DESCRIPTION
x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV...	

Status: 403 Forbidden Time: 27ms Size: 394 B

```
{
    "message": "Require Admin Role!"
}
```

If we use a wrong access token:



http://localhost:8080/api/test/user

KEY	VALUE	DESCRIPTION
x-access-token	ayJhbGciOiJIUzI1NiIsInR5cCI6IkpXV...	

Status: 401 Unauthorized Time: 17ms Size: 391 B

```
{
    "message": "Unauthorized!"
}
```

Conclusion

Oh yeah! Today we've learned so many interesting things about Node.js MongoDB User Authentication with JWT (JSONWebToken) in just a Node.js Express Rest Api example. You also know way to implement role-based Authorization to restrict access to protected resources.

You should continue to know how to implement Refresh Token:
Node.js & MongoDB: JWT Refresh Token example (https://bezkoder.com/jwt-refresh-token-node-js-mongodb/)

If you need a working front-end for this back-end, you can find Client App in the post:
- Vue (https://www.bezkoder.com/jwt-vue-vuex-authentication/)
- Angular 8 (https://www.bezkoder.com/angular-jwt-authentication/) / Angular 10 (https://www.bezkoder.com/angular-10-jwt-auth/) / Angular 11 (https://www.bezkoder.com/angular-11-jwt-auth/) / Angular 12 (https://www.bezkoder.com/angular-12-jwt-auth/) / Angular 13 (https://www.bezkoder.com/angular-13-jwt-auth/)
- React (https://www.bezkoder.com/react-jwt-auth/) / React Hooks (https://www.bezkoder.com/react-hooks-jwt-auth/) / React + Redux (https://www.bezkoder.com/react-redux-jwt-auth/)

Fullstack with React.js: MERN stack Authentication example (https://bezkoder.com/react-node-mongodb-auth/)

Happy learning! See you again.

Further Reading

- https://www.npmjs.com/package/express (https://www.npmjs.com/package/express)
- http://expressjs.com/en/guide/routing.html (https://expressjs.com/en/guide/routing.html)
- In-depth Introduction to JWT-JSON Web Token (https://bezkoder.com/jwt-json-web-token/)
- https://mongoosejs.com/docs/queries.html (https://mongoosejs.com/docs/queries.html)
- https://mongoosejs.com/docs/api/model.html (https://mongoosejs.com/docs/api/model.html)

Fullstack:
- MEVN: Vue.js + Node.js + Express + MongoDB example (https://bezkoder.com/vue-node-express-mongodb-mevn-crud/)
- MEAN:

- Angular 8 + Node.js + Express + MongoDB example (https://www.bezkoder.com/angular-mongodb-node-express/)

- Angular 10 + Node.js + Express + MongoDB example (https://www.bezkoder.com/angular-10-mongodb-node-express/)
- Angular 11 + Node.js + Express + MongoDB example (https://www.bezkoder.com/angular-11-mongodb-node-js-express/)
- Angular 12 + Node.js + Express + MongoDB example (https://www.bezkoder.com/angular-12-mongodb-node-js-express/)
- Angular 13 + Node.js + Express + MongoDB example (https://www.bezkoder.com/mean-stack-crud-example-angular-13/)

- MERN: React + Node.js + Express + MongoDB example (https://www.bezkoder.com/react-node-express-mongodb-mern-stack/)

Deployment: Docker Compose: Node.js Express and MongoDB example (https://www.bezkoder.com/docker-compose-nodejs-mongodb/)

Source Code

You can find the complete source code for this tutorial on Github (https://github.com/bezkoder/node-js-jwt-auth-mongodb).

Using Cookies: Node.js Express and MongoDB: Login and Registration example (https://www.bezkoder.com/node-js-express-login-mongodb/)

(https://www.bezkoder.com/tag/authentication/)

(https://www.bezkoder.com/tag/authorization/)

(https://www.bezkoder.com/tag/express/) (https://www.bezkoder.com/tag/jwt/)

(https://www.bezkoder.com/tag/login/) (https://www.bezkoder.com/tag/mongodb/)

(https://www.bezkoder.com/tag/mongoose/)

(https://www.bezkoder.com/tag/node-js/)

(https://www.bezkoder.com/tag/registration/)

(https://www.bezkoder.com/tag/rest-api/) (https://www.bezkoder.com/tag/security/)

(https://www.bezkoder.com/tag/token-based-authentication/)

112 thoughts to “Node.js + MongoDB: User Authentication & Authorization with JWT”