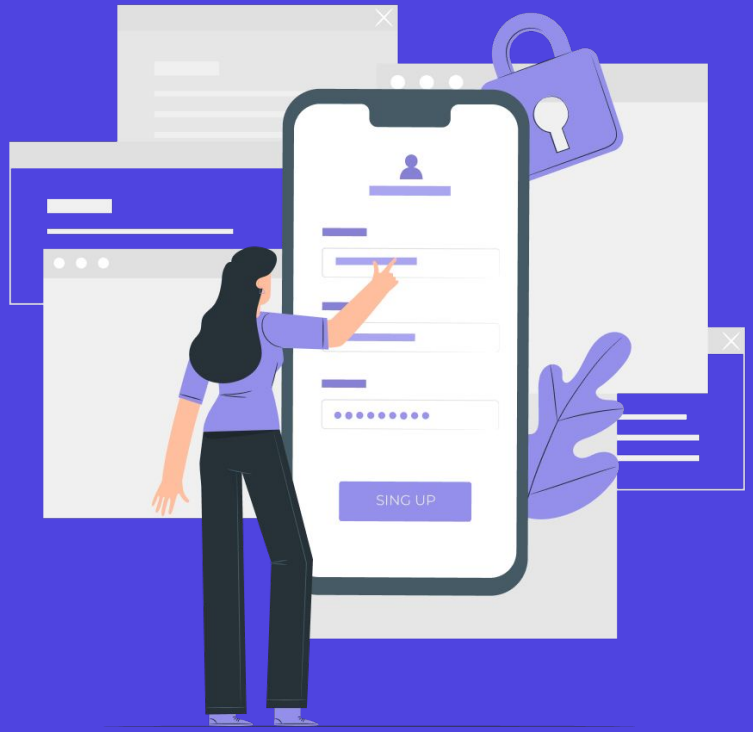
Filters and validations of request using middlewares



Relevel
by Unacademy

Class Agenda

- We will understand what is Authentication and Authorization.
- We will learn what are different types of Authentications: Basic Auth | Bearer Token Auth.
- We will create user SignUp and SignIn APIs using JWT token.

Educator Introduction

Authentication and Authorizations

Authentication - It decides who you are (Identification)

Authorization - It decides what you can do (Access Verification)

For example: To do anything on a website, the user has to login first. This is termed as authentications. However, what he can do on the website depends on his privileges. This is decided by Authorizations

Basic Authentication

- Authentication - It decides who you are (Identification)
- Authorization - It decides what you can do (Access Verification)

What are the limitations of basic authentication :

- Client credentials are overexposed
- Anyone can easily impersonate the client, if they somehow get to know the username password

Token Based Authentication/Bearer Authentication

How Bearer Authentication Works ?

- User registers the same was as before
- User login with the registered credentials. Login response returns Access Token as the response
- Now client instead of passing UserName/Password every time, they need to pass the access token which has a expiry time
- JWT token is a standard token used these days for the token based authentication

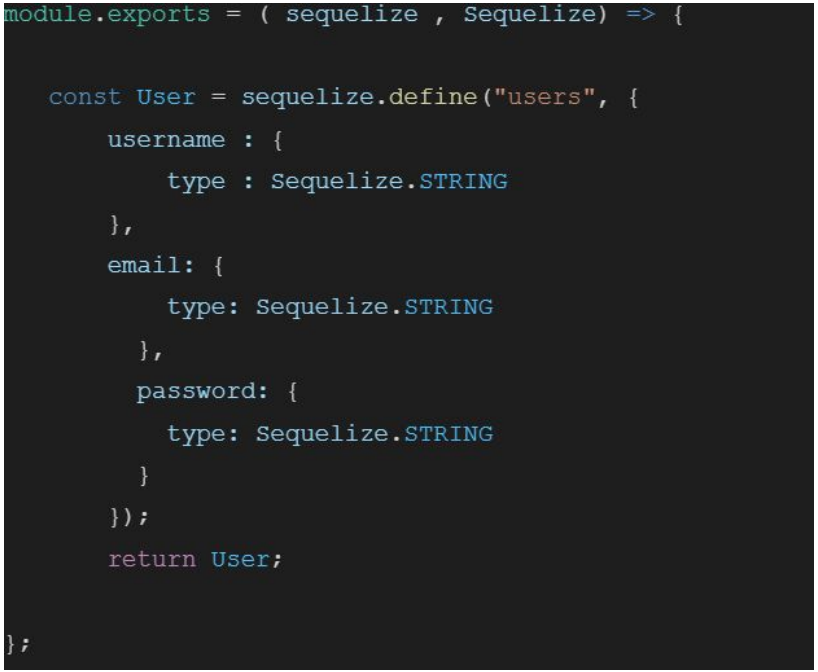
Advantages :

- Secret credential are not overexposed
- Access Tokens are encrypted
- Even if the access token access by someone, due to it’s expiry time, it can’t be used afterwards
- More secure than Basic Auth

Implementation of the token based authentication in our eCommerce application

1. Defining the user schema

https://github.com/Vishwa07dev/eCommerce/blob/session5/models/user.model.js



```
module.exports = ( sequelize , Sequelize) => {

   const User = sequelize.define("users", {
       username : {
           type : Sequelize.STRING
       },
       email: {
           type: Sequelize.STRING
         },
         password: {
           type: Sequelize.STRING
         }
       });
       return User;

};
```

2. Defining the Role Schema

https://github.com/Vishwa07dev/eCommerce/blob/session5/models/role.model.js

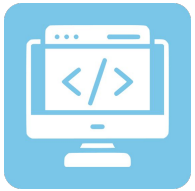
```
module.exports = (sequelize, Sequelize) => {
   const Role = sequelize.define("roles", {
     id: {
       type: Sequelize.INTEGER,
       primaryKey: true
     },
     name: {
       type: Sequelize.STRING
     }
   });
    return Role;
 };
```

3. Establishing the relationship between User and Role

https://github.com/Vishwa07dev/eCommerce/blob/session5/models/index.js

```
  * Establishing the relationship between Role and User
  */
db.role.belongsToMany(db.user, {
   through: "user_roles",
   foreignKey: "roleId",
   otherKey: "userId"
});
db.user.belongsToMany(db.role, {
   through: "user_roles",
   foreignKey: "userId",
   otherKey: "roleId"
});
```

4. Defining the controller for the authentication



https://github.com/Vishwa07dev/eCommerce/blob/session5/controllers/auth.controller.js

```
const config = require("../configs/auth.config");
const User = db.user;
const Role = db.role;

const Op = db.Sequelize.Op; //Operations

var jwt = require("jsonwebtoken");
var bcrypt = require("bcryptjs");

exports.signup = (req, res) => {
   console.log("Inside the sign up call");
   // Save User to Database
   User.create({
     username: req.body.username,
     email: req.body.email,
     password: bcrypt.hashSync(req.body.password, 8)
   })
```

```
    .then(user => {
      console.log("user created");
      if (req.body.roles) {
        Role.findAll({
          where: {
            name: {
              [Op.or]: req.body.roles
            }
          }
        }).then(roles => {
          user.setRoles(roles).then(() => {
            res.send({ message: "User registered successfully!" });
          });
        });
```

```
else {
          // user role = 1
          user.setRoles([1]).then(() => {
            res.send({ message: "User registered successfully!" });
          });
        }
      })
      .catch(err => {
        res.status(500).send({ message: err.message });
      });
 };

 exports.signin = (req, res) => {
   User.findOne({
     where: {
       username: req.body.username
     }
   })
```

```
    .then(user => {
      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 = [];
      user.getRoles().then(roles => {
        for (let i = 0; i < roles.length; i++) {
          authorities.push("ROLE_" + roles[i].name.toUpperCase());
        }
        res.status(200).send({
          id: user.id,
          username: user.username,
          email: user.email,
          roles: authorities,
          accessToken: token
        });
      });
    })
    .catch(err => {
      res.status(500).send({ message: err.message });
    });
};
```

5. Defining the routes for the authentications

https://github.com/Vishwa07dev/eCommerce/blob/session5/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(
       "/ecomm/api/v1/auth/signup",
```

```
      [
        verifySignUp.checkDuplicateUsernameOrEmail,
        verifySignUp.checkRolesExisted
      ],
      controller.signup
    );

  app.post("/ecomm/api/v1/auth/signin", controller.signin);

};
```

6. Write the middle ware to validate the signup request

https://github.com/Vishwa07dev/eCommerce/blob/session5/middlewares/verifySignUp.js

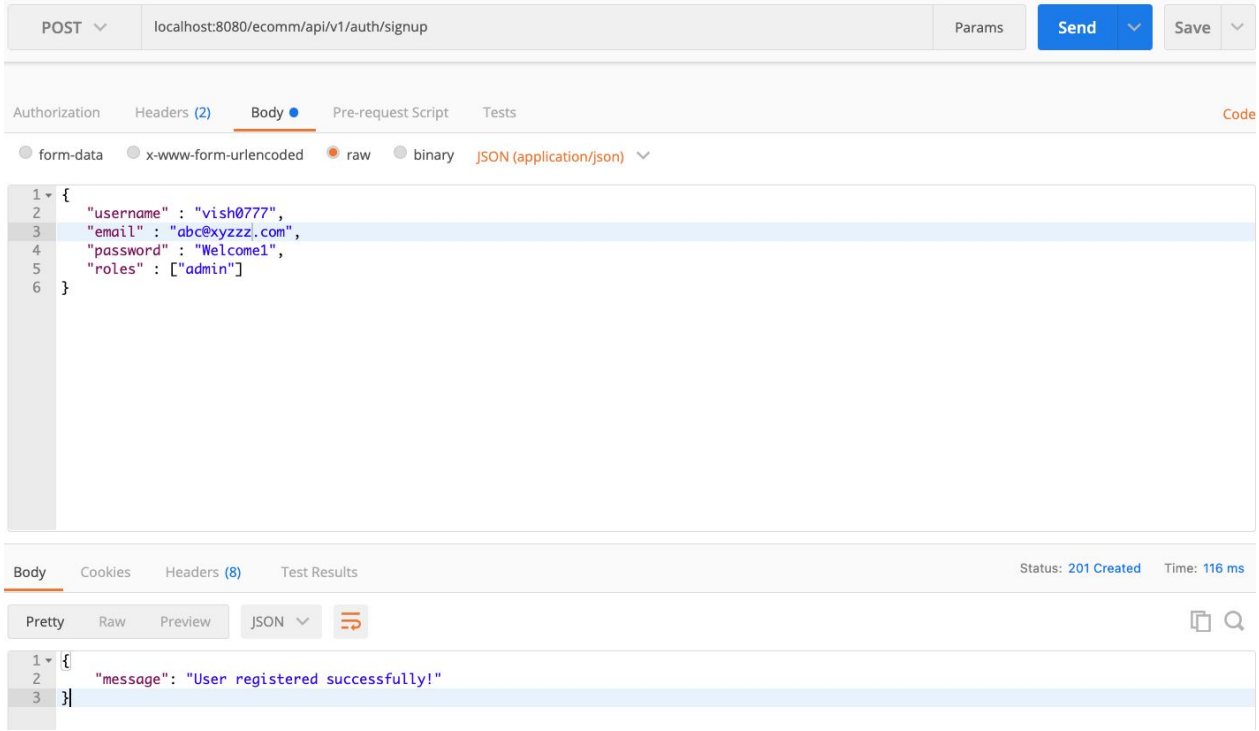
```
const db = require("../models");
const ROLES = db.ROLES;
const User = db.user;

checkDuplicateUsernameOrEmail = (req, res, next) => {
   console.log("Inside the checking if the duplicate username
exists")
   console.log(req.body);
   console.log(req.body.username);
   // Username
   User.findOne({
     where: {
       username: req.body.username
     }
```

```
  }).then(user => {
    if (user) {
      res.status(400).send({
        message: "Failed! Username is already in use!"
      });
      return;
    }
    // Email
    User.findOne({
      where: {
        email: req.body.email
      }
    }).then(user => {
      if (user) {
        res.status(400).send({
          message: "Failed! Email is already in use!"
        });
        return;
      }
      next();
```

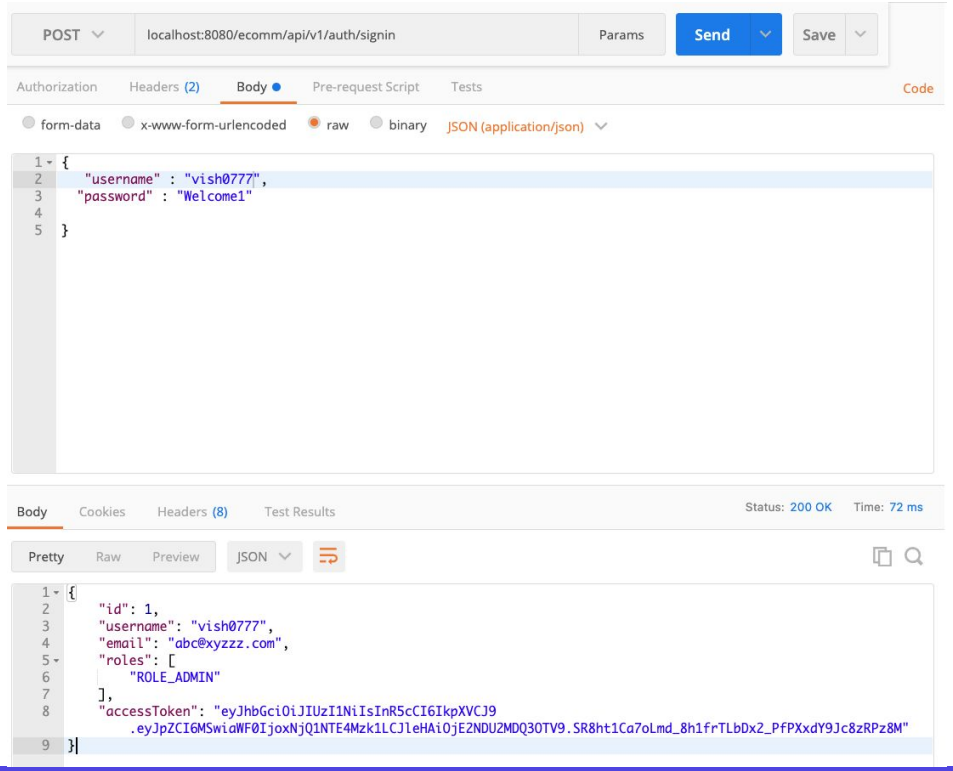
```
    });
  });
};
checkRolesExisted = (req, res, next) => {
  if (req.body.roles) {
    for (let i = 0; i < req.body.roles.length; i++) {
      if (!ROLES.includes(req.body.roles[i])) {
        res.status(400).send({
          message: "Failed! Role does not exist = " +
req.body.roles[i]
        });
        return;
      }
    }
  }
  next();
};
const verifySignUp = {
  checkDuplicateUsernameOrEmail: checkDuplicateUsernameOrEmail,
  checkRolesExisted: checkRolesExisted
};
module.exports = verifySignUp;
```

Testing the signup and signin APIs using POSTMAN

Registration :



Login :


POST
localhost:8080/ecomm/api/v1/auth/signin
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Code
form-data
x-www-form-urlencoded
raw
binary
JSON (application/json)
{
"username" : "vish0777",
"password" : "Welcome1"
}
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 72 ms
Pretty
Raw
Preview
JSON
{
"id": 1,
"username": "vish0777",
"email": "abc@xyzzz.com",
"roles": [
"ROLE_ADMIN"
],
"accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.eyJpZCI6MSwiaWF0IjoxNjQ1NTE4Mzk1LCJleHAiOjE2NDU2MDQ3OTV9.SR8ht1Ca7oLmd_8h1frTLbDx2_PfPXxdY9Jc8zRPz8M"
}

MCQs

1. A JWT consists of 3 parts –

A. Header, Payload & Signature

B. Header, Payload & Footer

C. Header, Token & Signature

D. Token, Payload & Signature

Ans: A

2. What is the process of identifying an individual?

A. authentication

B. authorization

C. accounting

D. auditing

Ans: A

3. How do we pass bearer token in request header?

A. <auth-token> Bearer

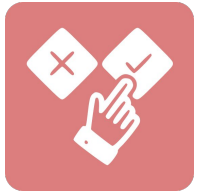
B. Bearer <auth-token>

C. <auth-token>

D. None of the Above

Ans: B

4. How to store local variables that can be accessed within the application?



A. Using app.locals

B. Using app.storage

C. Using database

D. Config file

Ans: A

5. How do we pass more than one middlewares in a route?

A. app.post(“/”, func1, func2, myPostFunc)

B. app.post(“/”, [func1, func2], myPostFunc)

C. app.post(“/”, { func1, func2 }, myPostFunc)

D. None of the above

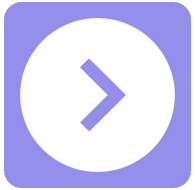
Ans: B

Practice Problems

Let us continues with our mini project and add some authentication which we learnt in today's class.

1. Create User Table, Schema and corresponding route and a controller with login and register functions.
2. Next, create one middleware to validate token given after login on routes such as put, post and delete books.

Next session



- Why authentication is necessary?
- Why Order APIs need to be authenticated and not others
- Create Order
- Update existing Order
- Get order details
- Delete the order

THANK YOU