Authentication: Verifiera identitet.

Vem är du? Http statuskod och meddelandet: 401 unauthorized

Authorization: Verifiera Tillstånd/permission. Vad kan du se, redigera i den här appen? Http statuskod och meddelandet: 403 forbidden

Två vanliga sätt att autentisera user :

Session based : Authentication : referens baserad

JWT: JSON Web token Authentication : Json data/value om user baserad.(med en signatur)

-------------------------------------------

HTTP: Stateless protokoll betyder att den inte kommer ihåg gammal request/tidigare request man har gjort.

Varje gång user skickar request server måste veta vem är den user.

**processen:**

Vad vill du? och vem är du? frågar server.

User säger jag heter till ex. kalle, jag vill se till ex. min profil

server kollar på om personen är legit och sen skickar response.

Men vad händer när användare har loggat in och vill ha mer information. hur kommer ihåg server?

Server kommer ihåg användare med hjälp av session token eller jwt token.

Scenario:

När användare har loggat in och server skickar en session token till user.

När användare skickar request nästa gång , skickar hen session token med sig. Då server behöver inte autentisera mer.

Förresten server har ne log med alla session id av användare.

Hur skickar server session id till?

cookies: gammalt sätt, monolitisk approach

Load balancer:

Redis/cached :

JWT: user/klient har data med app signatur.

**User** ---- ---------------> **server**

<------------------------------

Validation with Joi:

<https://www.npmjs.com/package/joi> /middleware validation , client validation

Jobba med joi:

<https://medium.com/@rossbulat/joi-for-node-exploring-javascript-object-schema-validation-50dd4b8e1b0f>

Lets code :

Skapa en user Model för inloggning:

//skapa ett schema för user

//Skapa en model utifrån schema

// tänk på Email ska vara unique

|  |
| --- |
| const userSchema = new mongoose.Schema({  name: String,  email: {  type:String,  required:true,  unique:true  },  password: {  type:String,  required:true,  minlength:8,  maxlength:1024  }  })  const User = mongoose.model('User',userSchema); |

Lägg till validering med JOI: <https://www.npmjs.com/package/express-joi-validation>

const mongoose = require("mongoose");

const joi = require("joi");

const userSchema = new mongoose.Schema({

username: {type:String, unique:true, minlength:5, maxlength:255, required:true},

email:{type:String, unique:true, minlength:5, maxlength:255, required:true},

password:{type:String, minlength:8, maxlength:1024, required:true}

})

const User = mongoose.model("User", userSchema)

//Input validation with joi

function validateUser(user){

const schema= {

username:joi.string().min(5).max(20).required(),

email:joi.string().min(5).max(255).required().email(),

password:joi.string().min(8).max(255)

}

return joi.validate(user, schema)

}

module.exports.User = User;

module.exports.validateUser = validateUser;

Skapa en Route för userRouter:

const {User, validateUser} = require('../model/user');

const express = require("express");

const router = express.Router();

router.get("/", (req, res)=>{

const msg = "Välkommen till vår hemsida";

res.render("loginPage", {msg})

})

router.post('/', async (req, res) => {

//joi valideringen

const { error } = validateUser(req.body);

if (error) return res.status(400).send(error.details[0].message);

//kolla upp om användare finns redan

let user = await User.findOne({ email: req.body.email });

if (user) return res.status(400).send('User already registered.');

//skapar nya användare

user = new User({

username: req.body.username,

email:req.body.email,

password:req.body.password

})

await user.save()

//skicka nya skapande användare till sin profilsida

res.render("userProfile", {user})

});

module.exports = router;

ejs : singup

<form action="/" method="post">

<input type="text" name="username">

<input type="email" name="email" id="">

<input type="password" name="password" id="">

<button type="submit">Sign up </button>

</form>

//userprofil:

<h1>Välkommen <%= user.username %>

</h1>

main.js/index.js/app.js:

const express = require("express");

const mongoose = require("mongoose");

const userRouter = require("./router/userRouter");

const config = require("./config/config");

const sassMiddleware = require("node-sass-middleware");

const path = require("path");

const app = express();

//middleware

app.use(express.urlencoded({extended:true}))

app.use( sassMiddleware({

src: path.join(\_\_dirname, "scss"),

dest: path.join(\_\_dirname, "public")

})

)

app.use(express.static(path.join(\_\_dirname, "public")));

app.set("view engine", "ejs");

//en till middleware för css

//router

app.use(userRouter);

//listen to port

const port = process.env.PORT || 8000;

const options ={

useUnifiedTopology: true,

useNewUrlParser: true,

useCreateIndex: true,

}

mongoose.connect(config.databaseURL,options ).then(()=> {

console.log("Successful")

//app is listening to port

app.listen(port);

})

Övning:

* Skapa en profilsida för användare

Hashing lösenord :

Docs : <https://github.com/kelektiv/node.bcrypt.js>

<https://www.npmjs.com/package/bcryptjs>

Hashing password :

router.post('/', async (req, res) => {

//joi valideringen

const { error } = validateUser(req.body);

if (error) return res.status(400).send(error.details[0].message);

//kolla upp om användare finns redan

let user = await User.findOne({ email: req.body.email });

if (user) return res.status(400).send('User already registered.');

var salt = await bcrypt.genSalt(10);

var hash = await bcrypt.hash(req.body.password, salt);

//skapar nya användare

user = new User({

username: req.body.username,

email:req.body.email,

password:hash

})

await user.save()

//skicka nya skapande användare till client

res.render("userProfile", {user})

});

Login ejs:

<%- include("partial/header.ejs") %>

<form action="/signin" method="post">

<input type="email" name="email" id="">

<input type="password" name="password" id="">

<button type="submit">Login</button>

</form>

<%- include("partial/footer.ejs") %>

Sign in route :

router.route("/signin")

.get( (req, res)=>{

res.render("loginPage")

})

.post(async (req, res) => {

//kolla upp om användare inte finns

let user = await User.findOne({ email: req.body.email });

if (!user) return res.redirect("/loginpage");

const validUser = await bcrypt.compare(req.body.password, user.password)

validUser?

//skicka användare till sin profil

res.render("userProfile", {user}): res.redirect("/loginpage")

});

Logout user/användare:

router.get("/logout", (req, res)=>{

res.clearCookie("jwtToken").redirect("/signin")

})

Läs artikeln:

<https://medium.com/@mridu.sh92/a-quick-guide-for-authentication-using-bcrypt-on-express-nodejs-1d8791bb418f>

Övning: Gör en authentication för user och admin:

* Användare kan lägga till produkter i cart när de har loggat in
* Användare kan se produkter utan att logga in
* Bara admin kan radera,uppdatera och lägga till produkter.

Session , cookies, JWT:

Lägg till jwt där du skapa token. till ex. userRouter.js

const jwt = require('jsonwebtoken');

OBS:Npm i jsonwebtoken.

router.route("/signin")

.get((req, res) => {

res.render("loginPage")

})

.post(async (req, res, next) => {

//kolla upp om användare inte finns

let user = await User.findOne({ email: req.body.email });

if (!user) return res.redirect("/loginpage");

const validUser = await bcrypt.compare(req.body.password, user.password)

if (!validUser) return res.redirect("/signin")

else

{

jwt.sign({ validUser }, "privatekey", (err, token) => {

if (err) res.redirect("/loginpage")

if (token) {

//Use localstorage istället

const cookie = req.cookies.jwtToken;

if (!cookie) {

//res.header("auth")

res.cookie('jwtToken', token, { maxAge: 900000, httpOnly: true });

}

res.render("userProfile", { user })

}

})

//skicka användare till sin profil

}

});

router.get("/logout", (req, res) => {

res.clearCookie("jwtToken").redirect("/signin")

})

***i*** views : Glöm ej.

<a href="/logout">Logout</a>

Läs mer:

<https://www.youtube.com/watch?v=2PPSXonhIck>

Cookies: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>

Authorisation:

Verifiera token/ kolla om user har en specifik token då kan hen nå till specifik resurser.

skapa en fil i router mappen: till ex. verifyToken.js

const jwt = require("jsonwebtoken");

module.exports = (req, res, next)=> {

const token = req.cookies.jwtToken;

console.log(token);

if(!token) return res.redirect("/signin")

try{

const verified = jwt.verify(token, "privatekey");

console.log(verified)

req.user = verified;

console.log(req.user.\_id)

next();

}

catch(err)

{

res.send("invalid error")

}

}

Lägg till verifyToken route som middleware till de där resurserna du vill skydda: till ex.

router.get("/secret", verifyToken, (req, res) => {

//req.header

res.render("adminSida")

})