Лабораторна робота №3

Тема: Автентифікація користувача. Passport.js. Чат на socket.io.

Теоретичні відомості

Хід роботи:

1) Створюєм каталог **Auth** з файлами **chat.html** та **server.js** з наступним вмістимим:

chat.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>MyChat</title>
</head>
<body>
    <div>Chat</div>
</body>
</html>
```

server.js:

```
//підключаєм express і створюємо app
var express=require('express');
var app=express();
//підключаєм модуль body-parser і інтегруєм в express
var bodyParser=require('body-parser');
app.use(bodyParser.urlencoded({extended:true}));
app.use(bodyParser.json());
//задаєм папку для статичного контенту
app.use(express.static(__dirname));
//опрацювання кореневого шляху
app.get('/',function(req,res){
    res.sendFile(__dirname+'/chat.html');
})
//порт прослуховування
app.listen(8080);
console.log('Run server!');
```

2) Створюєм файл **package.json**

```
{
  "name": "auth",
  "version": "1.0.0",
  "description": "auth passport",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
```

```
  "keywords": [
    "auth",
    "passport"
  ],
  "author": "name",
  "license": "ISC"
}
```

3) Інсталюєм локально модулі **express** та **body-parser** і додаєм їх в залежності в **package.json**

npm install express --save-dev

npm install body-parser --save-dev

3) Запускаєм сервер

node server.js

По адресу **localhost:8080** отримуєм файл **chat.html**.

4) Створюєм файл **login.html**, який міститеме форму для автентифікації користувача по логіну та паролю

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous">
    <title>Login</title>
</head>
<body>
    <div class="row mt-5">
        <form action="/login" method="post" class="col-sm-4 offset-sm-4 bg-
secondary text-white">
            <div class="form-group">
                <label>login:</label>
                <input type="text" name="username" class="form-control"
                placeholder="input login">
            </div>
            <div class="form-group">
                <label>password:</label>
                <input type="text" name="password" class="form-control"
                placeholder="input password">
            </div>
            <div class="form-group text-center">
                <input type="submit" value="Send" class="btn btn-primary">
            </div>

        </form>
```

```
    </div>
</body>
</html>
```

Поля для вводу даних містять атрибут **name** зі значеннями **username** та **password** відповідно. Дані з форми відправлятимуться на сервер методом post (шлях -'**/login**')

5) Інсталюєм локально модулі **cookie-parser**, **cookie-session, passport, passport-local** і додаєм їх в залежності в **package.json**

npm install cookie-parser --save-dev

npm install cookie-session --save-dev

npm install passport --save-dev

npm install passport-local --save-dev

6) Підключаєм в файлі server.js модуль cookie-parser і інтегруєм його в express

```
var cookieParser=require('cookie-parser')();
app.use(cookieParser);
```

7) Підключаєм в файлі server.js модуль cookie-session, час життя сесії - 2 год.

```
var session=require('cookie-session')({keys:['secret'],
maxAge:2*60*60*1000});
app.use(session);
```

8) Створюєм в файлі chatuser.js модель ChatUser для роботи з базою даних

```
var mongoose=require('./mongoose');
var schemaChatUser=new mongoose.Schema({
        username:{
        type:String,
        require:true,
        unique:true
    },
        password:{
        type:String,
        require:true,
    }
    })
var ChatUser=mongoose.model("ChatUser",schemaChatUser);
module.exports=ChatUser;
```

9) Інсталюєм локально модуль **mongoose** і створюєм файл підключення до бази даних **mongoose.js**

npm install mongoose --save-dev

```
var mongoose=require('mongoose');
mongoose.Promise = global.Promise;
mongoose.connect('mongodb://gnat:userdb@passdb.mlab.com:33260/mybase');
console.log("mongodb connect...")
module.exports=mongoose;
```

10) Підключаємо модель на сервері

```
var ChatUser=require('./chatuser');
```

11) Підключаємо passport-local для автентифікації, створюємо екземпляр passport-local, реалізуєм логіку автентифікації

```
var LocalStrategy=require('passport-local').Strategy;
passport.use(new LocalStrategy(
    function(username,password,done){
      ChatUser.find({username:username,password:password},
      function(err,data){
        console.log("data:");
        console.log(data);
        if(data.length)
        return done(null,{id:data[0]._id,username:data[0].username});
        return done(null,false);
      })

  }));
```

12) Записуємо дані об'єкта, які повертає local-стратегія після автентифікації в сесію, користувач авторизується

```
passport.serializeUser(function(user,done){
  console.log("serialize user:");
  console.log(user);
  done(null,user);
});
```

13) При всіх наступних зверненнях авторизованого користувача до сервера відбувається десеріалізація (використання даних сесії)

```
passport.deserializeUser(function(id,done){
  console.log("deserialize user:");
  console.log(id);
  ChatUser.find({_id:id.id},
      function(err,data){
        console.log(data);
      if(data.length==1)
      done(null,{username:data[0].username});
    });
});
```

14) Реалізуєм запуск автентифікації на основі local-стратегії з відповідним редіректом, а також створюєм middleware-функцію MyAuth, яка перевіряє, чи користувач є авторизованим

```
var auth=passport.authenticate(
  'local',{
    successRedirect:'/',
    failureRedirect:'/login'
  });
var myAuth=function(req,res,next){
  if(req.isAuthenticated())
    next();
  else {
    res.redirect('/login');
  }
}
```

15) Реалізуєм обробники клієнтських запитів на сервері

```
//перевірка чи user автентифікований
app.get('/',myAuth);
//опрацювання кореневого шляху
app.get('/',function(req,res){
  //console.log("req.user:");
  console.log("req.user:");
  console.log(req.user);
  console.log("req.session:");
  console.log(req.session);
  res.sendFile(__dirname+'/chat.html');
})
app.post('/login',auth);
app.get('/login',function(req,res){
  res.sendFile(__dirname+'/login.html');
})
```

15) Для миттєвої взаємодії між клієнтом та сервером (написання чату) інсталюєм модуль socket.io

npm install mongoose --save-dev

16) Підключаєм модуль socket.io та налаштовуємо сервер

```
var server=require('http').createServer(app);
var io=require('socket.io')(server);
```

...

```
server.listen(8080);
```

17) Прив'язуєм соккет до сесії

```
io.use(function(socket, next) {
    var req = socket.handshake;
    var res = {};
    cookieParser(req, res, function(err) {
        if (err) return next(err);
        session(req, res, next);
    });
});
```

18) Підключаємо socket.io на клієнті в файлі chat.html

```
<script src="https://cdn.socket.io/socket.io-1.4.5.js"></script>
```

19) Створюєм файл chat.js і встановлюєм зв'язок з сервером

```
$(document).ready(function(){
  // під'єднуємось до сервера - створюєм новий сокет
  var socket=io.connect('http://localhost:8080');
  // відправляємо повідомлення про під'єднання нового користувача
  socket.emit('joinclient', "is connected!");
```

20) Слухаєм подію 'joinclient' на сервері

```
var users=[];
io.on('connection', function (socket) {
    var user=socket.handshake.session.passport.user.username;
    var pos=users.indexOf(user);
    if(pos==-1) users.push(user);

    socket.on('joinclient',function(data){
      //console.log("push");
      console.log(data);
      console.log("socket-clients:");
      console.log(Object.keys(io.sockets.sockets));
      socket.emit('joinserver',{msg:"Привіт "+user+"!", users:users});
      socket.broadcast.emit('joinserver',{msg:"В чат увійшов "+user, users:users});
    })
```

21) Реалізувати чат, враховуючи, що

- **socket.on(event_name,function(data){..})** - прослуховування події, data - дані, які передаються;

- **socket.emit(event_name,data_obj)** - відправити дані data_obj лише тому користувачу, який звернувся до сервера;

socket.broadcast.emit(event_name, data_obj) - відправити дані data_obj всім користувачам, за виключенням того, який звернувся до сервера

io.sockets.emit(event_name, data_obj) - відправити дані data_obj всім користувачам.