Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное  
учреждение высшего образования  
 «ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Инженерно-технологическая академия

Институт компьютерных технологий и информационной безопасности

Кафедра систем автоматизированного проектирования

Отчёт по практической работе №3

по дисциплине «Разработка серверной части веб-приложений»

**Авторизация в Web-приложениях Node.js**

Выполнила

студентка группы КТбо3-4 Е. О. Локота

Принял

доцент кафедры МОП ЭВМ А. Н. Шкурко

Таганрог 2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc134049455)

[1 Подключение библиотек 4](#_Toc134049456)

[2 Решение задачи 4](#_Toc134049457)

[2.1 Алгоритм решения 4](#_Toc134049458)

[2.2 Программная реализация 4](#_Toc134049459)

[2.2.1 Файл index.js 5](#_Toc134049464)

[2.2.2 Файл users.json 8](#_Toc134049465)

[2.2.3 Файл profile.liquid 8](#_Toc134049466)

[2.2.4 Файл login.liquid, access\_denied.liquid и users.liquid 8](#_Toc134049467)

[2.3 Результат работы программы 9](#_Toc134049468)

[ЗАКЛЮЧЕНИЕ 11](#_Toc134049469)

[ПРИЛОЖЕНИЕ 12](#_Toc134049470)

Введение

**Цель работы**

Данная практическая работа направлена на изучение и практическое использование методов авторизации в web-приложениях Node.js.

**Задание**

В рамках выполнения практической работы требуется выполнить следующие задачи и продемонстрировать полученные навыки:

– Создать новое node.js приложение;

– Подключить к приложению библиотеки express.js и liquid;

– Разработать HTML-шаблоны для страниц приложения;

– Реализовать функциональность авторизации и аутентификации пользователя;

– Реализовать логику обработки на сервере согласно варианту задания;

**Вариант задания:**

**Задача 1**

Необходимо реализовать простое приложение, содержащее форму для ввода логина и пароля, а также следующие страницы:

- страницу информации о текущем пользователе;

- страницу со списком пользователей;

- страницу, указывающую на отсутствие доступа к ресурсу.

Страница информации о пользователе должна отображать данные текущего авторизованного пользователя и должна быть доступна только после ввода пароля. Также при авторизации пользователя должна устанавливаться его роль (user или admin). Страница со списком пользователей должна быть доступна только пользователям с ролью admin. В случае отсутствия доступа к странице должна отображаться соответствующая страница. Страница информации о пользователе должна содержать кнопку logout, которая возвращает пользователя на страницу ввода пароля. Пароли пользователей и их роли должны храниться в файле.

Необходимо использовать механизм сессий express.js.

1. Подключение библиотек

Используемы библиотеки изображены на рисунке 1. Они находятся в файле package.json.

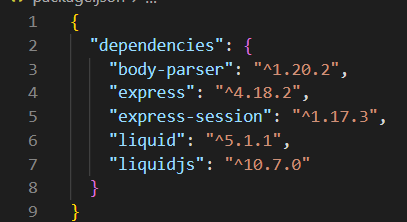


Рисунок 1 – Данные об установленных библиотеках

1. Решение задачи
   1. Алгоритм решения

* При запуске программы отображается страница авторизации
* Если пользователь ввел неправильный логин/пароль он остается на той же странице, в случае если пароль верный – он попадает на страницу с персональными данными, на которой отображается его имя, пароль и его роль
* Так же, на данной странице есть 2 кнопки. Одна – выход из аккаунта, вторая – переход на страницу с данными обо всех пользователях.
* Если пользователь администратор – он опадает на эту страницу, в противном случае он попадает на страницу с отказом в доступе.
  1. Программная реализация

Дерево проекта изображено на рисунке 2.

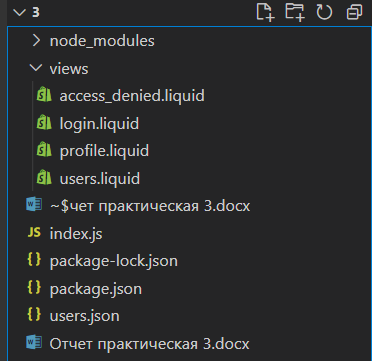


Рисунок 2 – Дерево проекта

2. 2. 1. Файл index.js

Подключаем модуль body-parser для обработки тела запросов.

«body-parser» — это посредник (middleware) для обработки данных тела запросов, отправленных клиентом на сервер. Он позволяет парсить данные из тела запроса и представлять их в удобном формате, например, в виде JavaScript-объекта.

Далее, добавляем middleware для приложения Express, который парсит тело запросов с типом данных JSON. Это позволяет обрабатывать и использовать данные, отправленные клиентом в формате JSON.

Данные полученные благодаря данному модулю будут храниться в свойстве «req.body» (рисунок 3).

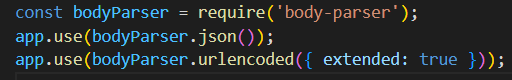


Рисунок 3 – Подключение необходимого модуля

Далее идет настройка сессий (рисунок 4).

* secret: это строка, используемая для подписи cookie-файлов, связанных с сессией, чтобы их нельзя было подделать.
* resave: это логическое значение, указывающее, должен ли сервер сохранять сессию при каждом запросе, даже если она не изменилась. Если он установлен в false, сессия будет сохраняться только при изменении.
* saveUninitialized: это логическое значение, указывающее, должен ли сервер сохранять новые и незаполненные сессии. Если он установлен в true, новая сессия будет сохраняться при первом запросе.

Таким образом, данный код создает middleware-функцию, которая позволяет создавать и использовать сессии в приложении Express.

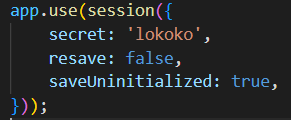


Рисунок 4 – Код для создания и использования сессий

Следующим шагом является чтение и запись данных из файла в массив (рисунок 5).

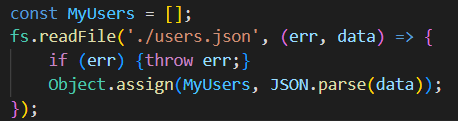


Рисунок 5 – Запись данных из файла в массив

Следующим шагом были написаны функции для проверки авторизации прав админа (рисунок 6–7).

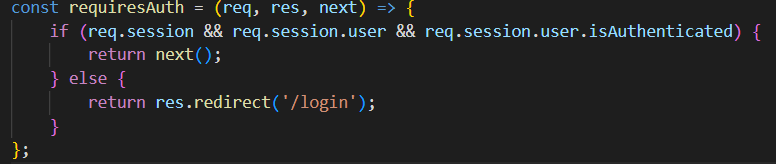


Рисунок 6 – Проверка авторизации

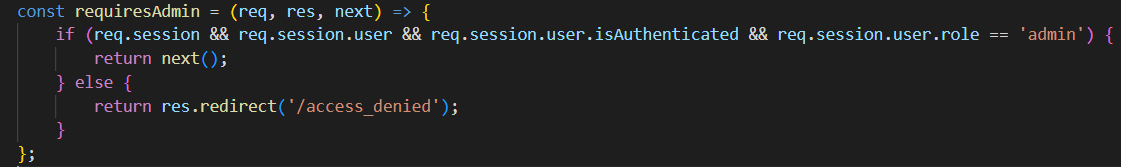


Рисунок 7 – Проверка прав администратора

Для проверки авторизации пользователя происходит чтение данных из полей для ввода, далее – происходит поиск пользователя из массива с таким же логином и паролем. Если пользователь найден – то в сессию записывается то, что он прошел аутентификацию и все его данные и далее перенаправляется на необходимую страницу (рисунок 8).

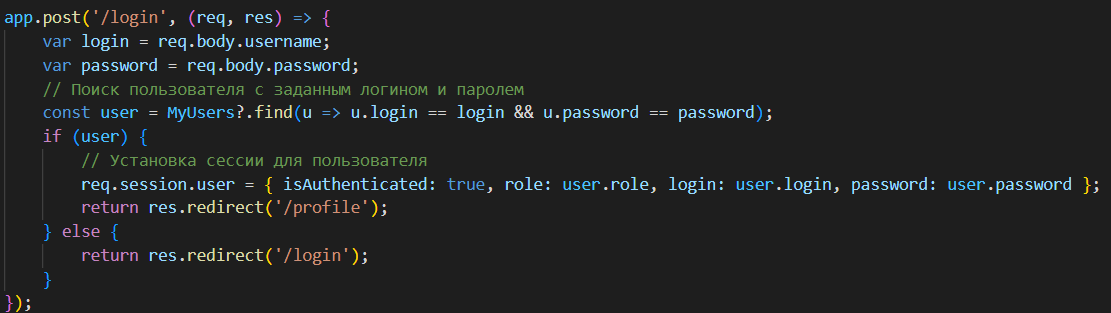


Рисунок 8 – Аутентификация

На страницу с информацией о пользователе выводятся его данные, которые хранятся в записи сессии (рисунок 9).

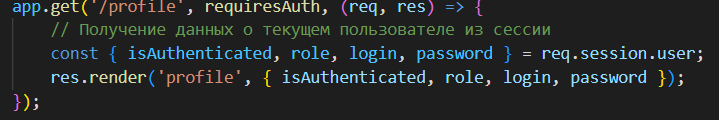


Рисунок 9 – Передача данных на страницу

* + 1. Файл users.json

В данном файле хранятся данные пользователей (рисунок 10).



Рисунок 10 – Данные о пользователях

* + 1. Файл profile.liquid

В данном файле написан код, осуществляющий вывод данных и содержащий в себе кнопки, благодаря которым, в дальнейшем, можно перейти на определенные страницы (рисунок 11).

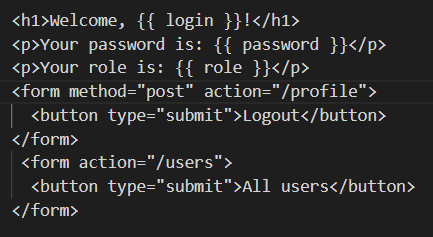


Рисунок 11 – Форма, отвечающая за одну кнопку

* + 1. Файл login.liquid, access\_denied.liquid и users.liquid

Данные файлы предназначены для отображения в случаях:

- если пользователь попал на сайт (login.liquid)

- если пользователю отказано в доступе (access\_denied.liquid)

- если пользователь является администратором (users.liquid).

* 1. Результат работы программы

Основная страница выглядит так, как изображено на рисунке 12.

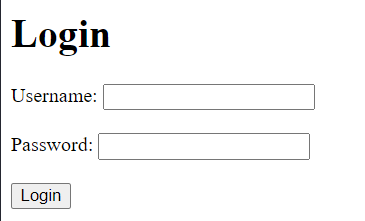


Рисунок 12 – Основная страница

После того, как пользователь ввел данные (рисунок 13), он нажимает на кнопку «Login» и попадает на страницу с информацией о себе (рисунок 14).

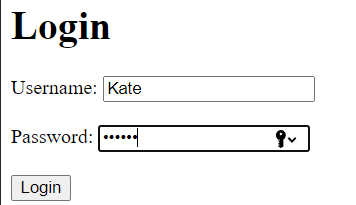


Рисунок 13 – Ввод данных

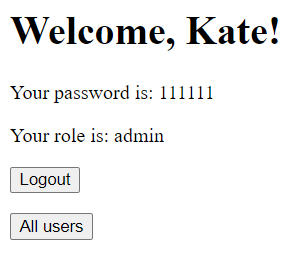


Рисунок 14 – Страница с информацией о пользователе

C этой страницы пользователь может выйти, нажав на кнопку «Logout» или же перейти на страницу со всеми пользователями (рисунок 15).

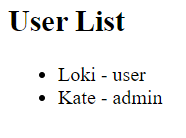


Рисунок 15 – Страница с информацией обо всех пользователях

В случае, если пользователь не является админом – он попадет на страницу с отказом в доступе (рисунок 16).



Рисунок 16 – Страница с отказом в доступе

Заключение

В ходе выполнения практического задания были изучены сессии в express.js, так же рассмотрен новый модуль body-parser.

Так же, были развиты знания Java Script.

ПРИЛОЖЕНИЕ

Файл index.js

const express = require('express');

const app = express();

const session = require('express-session');

var { Liquid } = require('liquidjs');

var engine = new Liquid();

const fs = require('fs');

const bodyParser = require('body-parser');

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true })); //парсинг массивов и пр и запись в сво-во req.body

//настройка для отображения html с помощью liquid

app.engine('liquid', engine.express());

app.set('views', './views');           //путь к шаблонам

app.set('view engine', 'liquid');      //установка механизма отображения на liquid

//app.set('trust proxy', 1) // trust first proxy

// Настройки сессий

app.use(session({

    secret: 'lokoko',             // секретный ключ для подписи cookie

    resave: false,

    saveUninitialized: true,

}));

const MyUsers = [];

fs.readFile('./users.json', (err, data) => {

    if (err) {throw err;}

    Object.assign(MyUsers, JSON.parse(data));        //слияние

});

//проверка авторизации

const requiresAuth = (req, res, next) => {

    if (req.session && req.session.user && req.session.user.isAuthenticated) {

        return next();

    } else {

        return res.redirect('/login');

    }

};

// проверка прав администратора

const requiresAdmin = (req, res, next) => {

    if (req.session && req.session.user && req.session.user.isAuthenticated && req.session.user.role == 'admin') {

        return next();

    } else {

        return res.redirect('/access\_denied');

    }

};

// Страница ввода логина и пароля

app.get('/login', (req, res) => {

    res.render('login');

});

app.post('/login', (req, res) => {

    var login = req.body.username;

    var password = req.body.password;

    // Поиск пользователя с заданным логином и паролем

    const user = MyUsers?.find(u => u.login == login && u.password == password);

    if (user) {

        // Установка сессии для пользователя

        req.session.user = { isAuthenticated: true, role: user.role, login: user.login, password: user.password };

        return res.redirect('/profile');

    } else {

        return res.redirect('/login');

    }

});

// Страница информации о пользователе

app.get('/profile', requiresAuth, (req, res) => {

    // Получение данных о текущем пользователе из сессии

    const { isAuthenticated, role, login, password } = req.session.user;

    res.render('profile', { isAuthenticated, role, login, password });

});

app.post('/profile', requiresAuth, (req, res) => {

    req.session.destroy();

   res.redirect('/login');

});

// Страница со списком пользователей (доступна только администраторам)

app.get('/users', requiresAdmin, (req, res) => {

    res.render('users', { users: MyUsers });

});

// Страница отсутствия доступа

app.get('/access\_denied', (req, res) => {

    res.render('access\_denied');

});

app.listen(3000, () => {

    console.log('Server started on port 3000');

});

Файл users.json

[

    {

        "login": "Loki",

        "password": 321123,

        "role": "user"

    },

    {

        "login": "Kate",

        "password": 111111,

        "role": "admin"

    }

]

Файл login.liquid

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Login Page</title>

</head>

<body>

<h1>Login</h1>

<form action="/login" method="POST">

Username: <input type="text" name="username"><br><br>

Password: <input type="password" name="password"><br><br>

<input type="submit" value="Login">

</form>

</body>

</html>

Файл profile.liquid

<h1>Welcome, {{ login }}!</h1>

<p>Your password is: {{ password }}</p>

<p>Your role is: {{ role }}</p>

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

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

</form>

<form action="/users">

<button type="submit">All users</button>

</form>

Файл users.liquid

<h2>User List</h2>

<ul>

{% for user in users %}

<li>{{ user.login }} - {{ user.role }}</li>

{% endfor %}

</ul>

Файл access\_denied.liquid

<h1>Ошибка доступа</h1>

<p>У вас нет прав доступа к этой странице.</p>