Лабораторная работа №5

Cookie-файлы, сессии и авторизация в веб-приложении

Задание:

1) Создать таблицу для хранения аккаунтов пользователей.

2) Создать страницу регистрации пользователя. Пароль зарегистрированного пользователя должен храниться в БД в хешированном виде.

3) Создать страницу авторизации пользователя и реализовать механизм доступа к REST API и странице с формой только для авторизированных пользователей, используя сессии и cookie.

4) Вывести на странице с формой для отправки запросов вывод всех запросов, созданных текущим зарегистрированным пользователем.

Код с примером реализации механизма авторизации пользователей находится в папке express_app_5.

Теоретический материал:

1) HTTP cookie-файлы и сессии.

2) Ознакомиться со следующими методами авторизации: HTTP Basic Authentication, OAuth и JSON Web Token (JWT).

Рекомендуемое программное обеспечение:

1) **DBeaver** – универсальный клиент для удаленной работы с БД:

https://dbeaver.io/files/dbeaver-ce-latest-x86_64-setup.exe

2) **Insomnia** – утилита для тестирования REST API:

https://insomnia.rest/download

Методические рекомендации

Рассмотрим реализацию простой системы авторизации пользователей в веб-приложении с помощью HTTP-сессий и cookie файлов.

1. Установка и подключение необходимых пакетов в проект. Для работы с cookie-файлами Node.js (express) приложение использует пакет **`cookie-parser`**, обычно данный пакет уже подключен в **app.js** по умолчанию при инициализации «каркаса» приложения express:

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

Если данных строчек нет в app.js, устанавливаем в проект:

`npm install --save cookie-parser`

Для работы с сессиями и зашифрованными паролями устанавливаем:

`npm install --save bcrypt express-session`

Подключаем пакет для работы с сессиями в приложении (app.js):

```
var session = require('express-session');
```

ВАЖНО: Подключение пакета express-session должно происходить после пакета cookie-parser, т.к. **`cookie-parser`** является необходимой зависимостью для пакета **`express-session`**.

Далее, используя инструкцию use, инициализируем обработчик сессий пользователей в **app.js**:

```
app.use(session({
  key: 'user_sid',
  secret: 'anypassword',
  resave: false,
  saveUninitialized: true,
  cookie: {
      signed: false,
      maxAge: 600000
  }
}));
```

ВАЖНО: Подключать сессии необходимо ДО подключения обработчиков маршрутов (роутеров)!

Основные параметры сессии:

`key` - имя файла cookie с идентификатором сессии;

`secret` - пароль для подписи cookie;

`cookie: {…}` - параметры cookie сессии.

Основные параметры cookie:

signed - true/false для подписи cookie;

secure - обеспечивает отправку файлов cookie браузером только с использованием протокола HTTPS;

httpOnly - обеспечивает отправку cookie только с использованием протокола HTTP(S), а не клиентского JavaScript, что способствует защите от атак межсайтового скриптинга;

domain - указывает домен cookie; используется для сравнения с доменом сервера, на котором запрашивается данный URL. В случае совпадения выполняется проверка следующего атрибута – пути;

path - указывает путь cookie; используется для сравнения с путем запроса. Если путь и домен совпадают, выполняется отправка cookie в запросе.

maxAge - используется для настройки даты окончания срока хранения для постоянных cookie.

После подключения пакетов для работы с cookie и сессиями в приложении будут доступны объекты **session** и **cookie**, которые можно использовать при обработке запроса (req) и отправки ответа (res). Например, установка параметра сессии user:

```
function(req, res) {
    …
    req.session.user = 'any_value';
    …
}
```

Установка cookie с именем AuthToken и значением 'your_token':

```
function(req, res) {
    …
    res.cookie('AuthToken', 'your_token');
    …
}
```

Чтение значения cookie с именем AuthToken:

```
function(req, res) {
    …
    token = req.cookies['AuthToken'];
    …
}
```

2. Регистрация и авторизация аккаунта пользователя. Для хранения аккаунтов пользователей добавьте соответствующую SQL-инструкцию в скрипт для инициализации вашей БД. Например, для базового аккаунта пользователя можно создать таблицу с тремя полями:

`username` – имя пользователя (логин);

`password` – пароль;

`email` – электронная почта пользователя для подтверждения аккаунта и сброса пароля (в рамках данной лабораторной работы эти функции использоваться не будут).

SQL-инструкция:

```
create table logins (
     id integer PRIMARY KEY autoincrement,
     username varchar(255) NOT NULL UNIQUE,
     email varchar(255) NOT NULL UNIQUE,
     password varchar(255) NOT NULL UNIQUE
);
```

ВАЖНО: если при создании таблицы аккаунтов пользователей объявляется уникальный **primary key**, то в дальнейшем данный ключ можно использовать как внешний в таблицах с данными, например, для определения прав доступа того или иного аккаунта к соответствующим данным. Таким же образом можно создавать отдельные «роли» для аккаунтов: администратор, гость и т.п.

Также стоит учитывать, что поддержка внешних ключей в SQLite по умолчанию отключена, для включения функции необходимо выполнить SQL-инструкцию:

```
PRAGMA foreign_keys=ON
```

2.1 Регистрация аккаунта. В соответствии с параметрами модели создается шаблон страницы с веб формой регистрации (адрес **/register**):

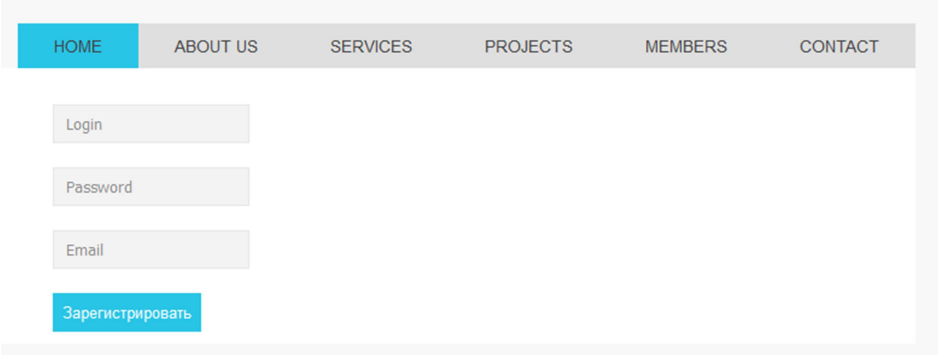

Рис. 1 – Веб-форма регистрации

Как известно (из лабораторной работы №3), рендеринг шаблонов в приложении express осуществляется в специальных обработчиках запросов – роутерах, расположенных в **папке routes** с соответствующим именем.

Создаем роутер **`register.js`** с обработчиком GET-запроса:

```
router.get('/', function(req, res) {
  res.render('register', {
    title: 'Whitesquare',
    pname: 'AUTH',
    navmenu: navmenu });
});
```

Также веб-форма, используя параметры **`method`** (метод отправки) и **`action`** (адрес отправки), будет отправлять данные из полей по соответствующему адресу, например:

```
<form method="POST" action="/register">
```

Таким образом, необходимо реализовать обработчик POST-запроса роутере **register.js**:

```
router.post('/', register_user);
```

Где **`register_user`** – это метод контроллера (**`mainController`**), который получает данные формы, создает по ним новый аккаунт и записывает его в БД.

```
exports.register_user = function(req, res) {
    // Проверяем полученные данные на наличие обязательных полей
    if (!req.body.loginField || !req.body.emailField || !req.body.passField) {
        res.status(400).json({ message: "The data entered are not correct!"
});    // если данные не найдены, возвращаем HTTP-код 400
    return;
    }
    // Создаем хеш пароля с солью
    const salt = bcrypt.genSaltSync();
    var hashed = bcrypt.hashSync(req.body.passField, salt);
    // Создаем пользователя в БД
    dbcontext.query(
        'INSERT INTO logins (`username`, `password`, `email`) VALUES (:username,
:password, :email)',
        {
            replacements: { username: req.body.loginField, password: hashed,
email: req.body.emailField },
            type: dbcontext.QueryTypes.INSERT
        }
    )
    .then(result => {
        console.log(`Registered as ${req.body.loginField}`);
```

```
        // в случае успешной записи переадресуем пользователя на страницу
авторизации
        res.redirect('/login');
    })
    .catch(err => {
        // в случае исключения возвращаем код 500 + json-ответ с ошибкой
        res.status(500).json({ message: err.message });
    });
};
```

Параметры объекта **newLogin** считываются из данных формы с помощью инструкции:

`req.body.<параметр_формы>` ,

где `<параметр_формы>` – это соответствующе значение атрибута name из элемента формы в шаблоне, например, если в форме есть поле:

```
<input name="loginField" type="text" placeholder="Login">
```

То введённые данные получаем инструкцией:

```
req.body.loginField
```

Также по коду видно, что перед записью в БД пароль пользователя хешируется «с солью» (соль хеша – это дополнительная строка-модификатор, которая добавляется к исходному паролю перед вычислением хеш-функции для усложнения декодирования хеша). Хеширование пароля осуществляется с помощью установленной ранее (см. п.1) библиотеки **bcrypt**. Данный пакет должен быть предварительно подключен в контроллере с помощью инструкции:

```
const bcrypt = require('bcrypt');
```

2.2 Авторизация аккаунта. Для авторизации аккаунта пользователя создаем шаблон формы со следующими полями (адрес **/login**):

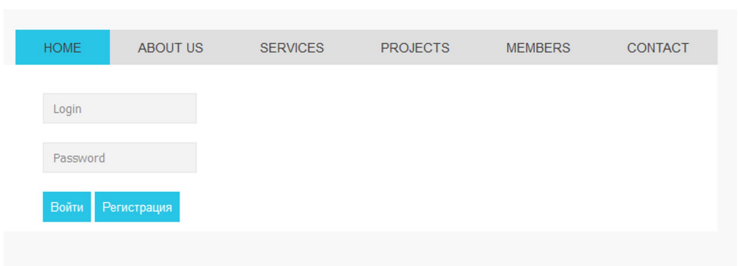

Рис. 2 – Веб-форма авторизации

Также как и для страницы регистрации создаем для данной формы обработчики GET и POST запросов и реализуем метод проверки учётных данных в контроллере (**`login_user`**):

```
exports.login_user = function(req, res) {
    // Получаем логин и пароль из данных формы
    var login = req.body.loginField;
    var password = req.body.passField;
    // Ищем пользователя в БД
    dbcontext.query(
        'SELECT * FROM logins WHERE "username" = :username',
        {
            replacements: { username: login },
            type: dbcontext.QueryTypes.SELECT
        }
    ).then(data => {
        var user = data[0]
        // если пользователь не найден переадресуем на страницу /login
        if (!user) {
            res.redirect('/login');
        // если пользователь найден, проверяем пароль
        } else if (!bcrypt.compareSync(password, user.password)) {
            // если пароль не прошел проверку, переадресуем на страницу /login
            res.redirect('/login');
        } else {
            // иначе регистрируем сессию пользователя (записываем логин
пользователя в параметр user)
            req.session.user = user.username;
            req.session.userId = user.id;
            // высылаем сессионную cookie AuthToken с логином
            res.cookie('AuthToken', user.username);
            res.redirect('/');
        }
    })
    .catch(err => {
        // в случае исключения возвращаем код 500 + json-ответ с ошибкой
        res.status(500).json({ message: err.message });
    });
};
```

В первую очередь данная функция ищет пользователя в БД с совпадающим username, если пользователь найден, то запускается проверка введённого пароля с хешированным паролем из БД. Проверка осуществляется с помощью специальной функции **`compareSync`** из библиотеки bcrypt. Далее, если пароль проверен успешно, создаем в сессии пользователя параметр user, куда записываем текущий username авторизированного пользователя и userId (идентификатор пользователя в БД), а также передаем пользователю сессионный (т.е. актуальный, пока активна сессия пользователя) cookie-файл «AuthToken», который также содержит username.

Для идентификации авторизированного пользователя можно хранить любые параметры в сессии, с любым названием и содержанием, главное, чтобы параметры были уникальны для каждого авторизированного пользователя.

3. Реализация доступа к ресурсам веб-приложения авторизированных пользователей. В Node.js для предоставления доступа к тем или иным ресурсам веб-приложения используется механизм промежуточных callback-функций при обработке пользовательских запросов. Например, ограничим доступ к странице /contact и сделаем её доступной только для авторизированных пользователей. Для этого добавим в обработчик данной страницы (**`contact.js`**) промежуточную функцию **`sessionCheck`**:

```
router.get('/', sessionCheck, function(req, res) {
  res.render('contact', {
    title: 'Whitesquare',
    pname: 'CONTACT',
    user: req.session.user,
    navmenu: navmenu });
});
```

Саму функцию реализуем в контроллере приложения:

```
exports.sessionCheck = (req, res, next) => {
    // Если не установлен параметр сессии user или значение cookie 'AuthToken' не
 равно логину пользователя
    if (!req.session.user || req.cookies['AuthToken'] != req.session.user) {
        // переадресуем на страницу /login
        res.redirect('/login');
    } else {
        // иначе исполняем следующую функцию обработчика
        next();
    }
};
```

Таким образом, при каждом обращении к странице /contact будет предварительно выполняться проверка сессии и cookie-файлов пользователя.

4. Проверка функций авторизации вашего приложения. Помимо стандартной проверки доступа к ресурсам через запрос/ответ к приложению (например, с помощью REST API клиентов Insomnia или Postman) используйте инструментарий для веб-разработки, встроенный в популярные браузеры. Данный инструментарий позволяет отследить и посмотреть содержание ваших cookie-файлов. Например, для браузера Firefox:

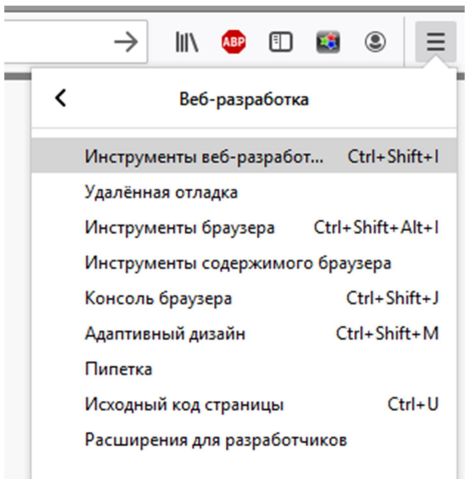

Рис. 3 – Инструменты веб-разработчика в Firefox

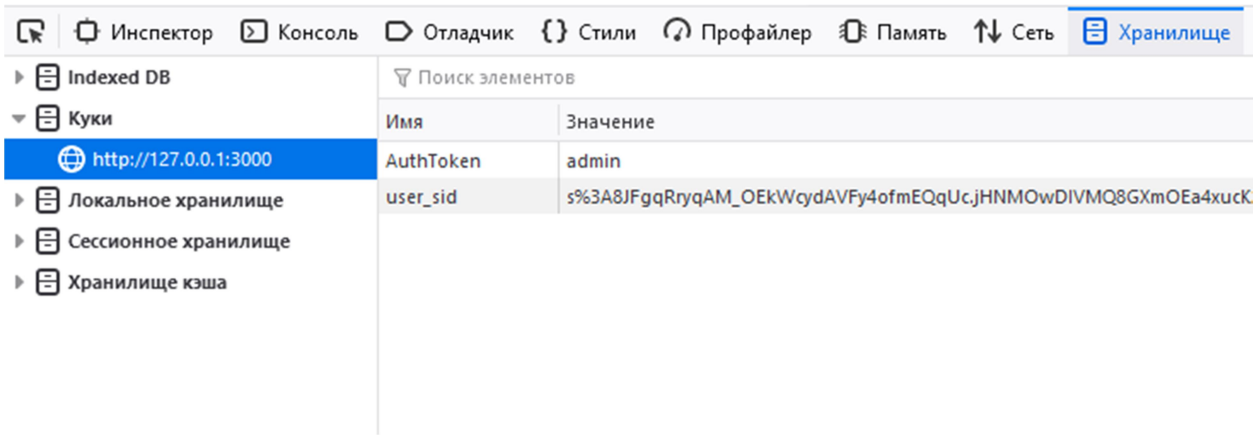

Рис. 4 – Просмотр cookie-файлов в «Хранилище»

Дополнительная литература:

Работа с маршрутами в приложении Node.js:

https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs/routes

Cookie-файлы:

https://developer.mozilla.org/ru/docs/Web/HTTP/Cookies

Работа с cookie-файлами и сессиями в Node.js:

https://nodejsdev.ru/doc/cookie/

https://nodejsdev.ru/doc/sessions/

Работа с bcrypt:

https://www.npmjs.com/package/bcrypt

Дополнительная информация по обеспечению безопасности Node.js приложения:

https://expressjs.com/ru/advanced/best-practice-security.html

Результатом выполнения задания являются файлы с кодом веб-приложения и отчет, содержащий следующую информацию:

1) Краткий конспект теоретического материала по методам авторизации: HTTP Basic Authentication, OAuth и JSON Web Token (JWT).
2) Скриншоты страниц авторизации и регистрации с описанием соответствующих обработчиков.
3) Код функций: добавления аккаунта пользователя в БД, поиска и проверки учетных данных пользователя в БД, промежуточная функция валидации сессии пользователя.
4) Скриншоты, содержащие результаты проверки вашего API в Insomnia или любом другом REST API клиенте, а также скриншоты браузера с вашими cookie-файлами в хранилище.
5) Работающее веб-приложение, реализованное по заданию.

Требования к оформлению отчета:

Способ выполнения текста должен быть единым для всей работы. **Шрифт** –

Times New Roman, кегль 14, **межстрочный интервал** – 1,5, **размеры полей**: левое – 30 мм; правое – 10 мм, верхнее – 20 мм; нижнее – 20 мм. Сокращения слов в тексте допускаются только общепринятые.

Абзацный отступ (1,25) должен быть одинаковым во всей работе. **Нумерация страниц** основного текста должна быть сквозной. Номер страницы на титульном листе не указывается. Сам номер располагается внизу по центру страницы или справа.

Разработано: Юдинцев Б.С.