Министерство цифрового развития, связи и массовых коммуникаций РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
Уральский технический институт связи и информатики (филиал) ФГБОУ ВО
"Сибирский государственный университет телекоммуникаций и информатики" в г. Екатеринбурге (УрТИСИ СибГУТИ)

КАФЕДРА
Информационных систем и технологий
(ИСТ)

ОТЧЕТ
По дисциплине
«Сетевое программирование»
Практическое занятие №3
«Работа с авторизацией и аутентификацией»

Выполнил: студент гр. ПЕ-11б
Хлебникова Е.А.
Проверил: преподаватель
Бурумбаев Д.И.

Екатеринбург, 2024

Работа с авторизацией и аутентификацией

1 Цель работы:

1.1 Научиться работать с регистрацией, авторизацией и аутентификацией;

1.2 Закрепить знания по теме «Аутентификация и авторизация пользователей в клиент-серверных приложениях».

2 Перечень оборудования:

2.1 Персональный компьютер;

2.2 Postman;

2.3 node.js;

2.4 Visual Studio Code.

3.Ход работы:

3.1 Для реализации системы регистрации, авторизации и аутентификации необходимо выполнить ряд действий. Первым делом необходимо настроить базу данных. Для примера, можно использовать базу данных MySQL при помощи XAMPP, WAMP или другого ПО. Внутри необходимо создать базу данных familia_login. Затем внутри этой базы данных нужно создать таблицу с именем users при помощи SQL-запроса, который представлен в листинге 1.

Листинг 1 – SQL-запрос для создания таблицы

```
CREATE TABLE users (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  name varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
  email varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
  password varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY email (email)
 )    ENGINE=InnoDB    AUTO_INCREMENT=2    DEFAULT    CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

Далее в качестве примера представлена реализация при помощи node.js и JavaScript. Для начало необходимо установить NPM при помощи команды:

Листинг 2 – Инициализация NPM

```
npm i express ejs express-session express-validator bcryptjs mysql2
```

А также изменить созданный файл package.json

Листинг 3 – Package.json

```
{
  "name": "nodejs-login-registration",
  "version": "1.0.0",
  "description": "Node JS Login And Registration System",
  "main": "index.js",
```

```
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "w3jar.com (chandan)",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "ejs": "^3.1.6",
    "express": "^4.17.1",
    "express-session": "^1.17.2",
    "express-validator": "^6.11.1",
    "mysql2": "^2.2.5"
  }
}
```

Далее необходимо создать структуру для реализации системы как показано на рисунке 1.

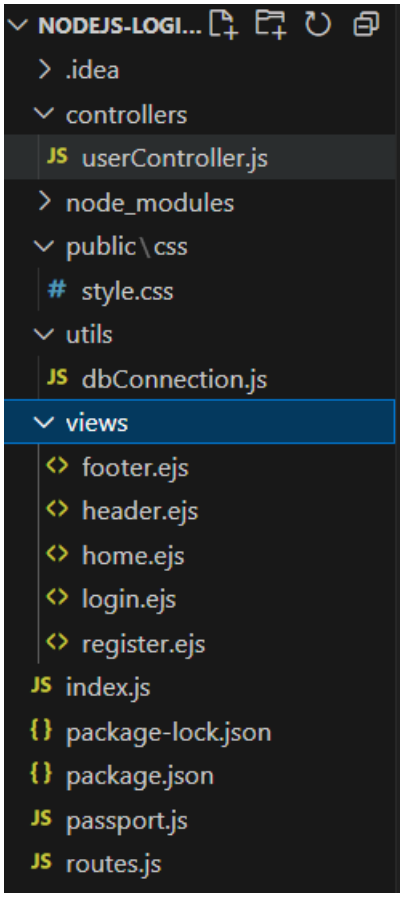

Рисунок 1 – Структура проекта

Далее для подключения базы данных к проекту необходимо в файле dbConnection.js необходимо прописать код, представленный в листинге 4.

Листинг 4 – Код в файле dbConnection

```
const mysql = require('mysql2');

const dbConnection = mysql.createPool({
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'nodejs_login'
});

module.exports = dbConnection.promise();
```

Далее необходимо наполнить каталог views. Описанные далее файлы будут находиться в этой папке. Содержимое файлов представлено в листингах 5-10.

Листинг 5 – Код в файле header.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta   name="viewport"   content="width=device-width,   initial-
scale=1.0">
    <title><%= (typeof(title) !== 'undefined') ? title : 'Node JS Login
and Registration System - W3jar' %></title>
    <link rel="preconnect" href="https://fonts.gstatic.com">
    <link
href="https://fonts.googleapis.com/css2?family=Ubuntu:wght@400;700&di
splay=swap" rel="stylesheet">
    <link rel="stylesheet" href="./style.css">
</head>
<body>
```

Листинг 6 – Код в файле footer.ejs

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

Листинг 7 – Код в файле login.ejs

```
<%- include('header',{
    title:'Login'
}); -%>

<div class="container">
    <h1>Login</h1>
    <form action="" method="POST">
        <label for="user_email">Email</label>
        <input     type="email"     class="input"     name="_email"
id="user_email" placeholder="Enter your email">
        <label for="user_pass">Password</label>
        <input   type="password"   class="input"   name="_password"
id="user_pass" placeholder="Enter new password">
        <% if(typeof error !== 'undefined'){ %>
        <div class="err-msg"><%= error %></div><% } %>
```

```
        <input type="submit" value="Login">
        <div class="link"><a href="./signup">Sign Up</a></div>
    </form>
</div>

<%- include('footer'); -%>
```

Листинг 8 – Код в файле register.ejs

```
<%- include('header',{
    title:'Signup'
}); -%>

<div class="container">
    <h1>Sign Up</h1>
    <form action="" method="POST">
        <label for="user_name">Name</label>
        <input type="text" class="input" name="_name" id="user_name"
placeholder="Enter your name">
        <label for="user_email">Email</label>
        <input      type="email"      class="input"      name="_email"
id="user_email" placeholder="Enter your email">
        <label for="user_pass">Password</label>
        <input    type="password"    class="input"    name="_password"
id="user_pass" placeholder="Enter new password">
        <% if(typeof msg != 'undefined'){ %>
        <div class="success-msg"><%= msg %></div><% }
        if(typeof error !== 'undefined'){ %>
        <div class="err-msg"><%= error %></div><% } %>
        <input type="submit" value="Sign Up">
        <div class="link"><a href="./login">Login</a></div>
    </form>
</div>

<%- include('footer'); -%>
```

Листинг 9 – Код в файле home.ejs

```
<%- include('header'); -%>
<div class="container">
    <div class="profile">
        <div class="img">&#128100;</div>
        <h2><%= user.name %></h2>
        <span><%= user.email %></span>
        <a href="/logout">Log Out</a>
    </div>
</div>
<%- include('footer'); -%>
```

Листинг 10 – Код в файле routes.js

```
const router = require("express").Router();
const { body } = require("express-validator");

const {
    homePage,
    register,
    registerPage,
    login,
```

```
    loginPage,
} = require("./controllers/userController");

const ifNotLoggedin = (req, res, next) => {
    if(!req.session.userID){
        return res.redirect('/login');
    }
    next();
}

const ifLoggedin = (req,res,next) => {
    if(req.session.userID){
        return res.redirect('/');
    }
    next();
}

router.get('/', ifNotLoggedin, homePage);

router.get("/login", ifLoggedin, loginPage);
router.post("/login",
ifLoggedin,
    [
        body("_email", "Invalid email address")
            .notEmpty()
            .escape()
            .trim()
            .isEmail(),
        body("_password", "The Password must be of minimum 4
characters length")
            .notEmpty()
            .trim()
            .isLength({ min: 4 }),
    ],
    login
);

router.get("/signup", ifLoggedin, registerPage);
router.post(
    "/signup",
    ifLoggedin,
    [
        body("_name", "The name must be of minimum 3 characters
length")
            .notEmpty()
            .escape()
            .trim()
            .isLength({ min: 3 }),
        body("_email", "Invalid email address")
            .notEmpty()
            .escape()
            .trim()
            .isEmail(),
        body("_password", "The Password must be of minimum 4
characters length")
            .notEmpty()
            .trim()
            .isLength({ min: 4 }),
```

```
    ],
    register
);

router.get('/logout', (req, res, next) => {
    req.session.destroy((err) => {
        next(err);
    });
    res.redirect('/login');
});

module.exports = router;
```

Далее необходимо в корневом каталоге приложения создать папку controllers и внутри этой папки создать файл userController.js, содержание которого представлено в листинге 11.

Листинг 11 – Код в файле userController.ejs

```
const { validationResult } = require("express-validator");
const bcrypt = require('bcryptjs');
const dbConnection = require("../utils/dbConnection");

// Home Page
exports.homePage = async (req, res, next) => {
    const [row] = await dbConnection.execute("SELECT * FROM `users` 
WHERE `id`=?", [req.session.userID]);

    if (row.length !== 1) {
        return res.redirect('/logout');
    }

    res.render('home', {
        user: row[0]
    });
}

// Register Page
exports.registerPage = (req, res, next) => {
    res.render("register");
};

// User Registration
exports.register = async (req, res, next) => {
    const errors = validationResult(req);
    const { body } = req;

    if (!errors.isEmpty()) {
        return res.render('register', {
            error: errors.array()[0].msg
        });
    }

    try {

        const [row] = await dbConnection.execute(
            "SELECT * FROM `users` WHERE `email`=?",
```

```
            [body._email]
        );

        if (row.length >= 1) {
            return res.render('register', {
                error: 'This email already in use.'
            });
        }

        const hashPass = await bcrypt.hash(body._password, 12);

        const [rows] = await dbConnection.execute(
            "INSERT      INTO      `users`(`name`,`email`,`password`)
VALUES(?,?,?)",
            [body._name, body._email, hashPass]
        );

        if (rows.affectedRows !== 1) {
            return res.render('register', {
                error: 'Your registration has failed.'
            });
        }

        res.render("register", {
            msg: 'You have successfully registered.'
        });

    } catch (e) {
        next(e);
    }
};

// Login Page
exports.loginPage = (req, res, next) => {
    res.render("login");
};

// Login User
exports.login = async (req, res, next) => {

    const errors = validationResult(req);
    const { body } = req;

    if (!errors.isEmpty()) {
        return res.render('login', {
            error: errors.array()[0].msg
        });
    }

    try {

        const  [row]  =  await  dbConnection.execute('SELECT  *  FROM
`users` WHERE `email`=?', [body._email]);

        if (row.length != 1) {
            return  res.render('login',  {
                error: 'Invalid email address.'
            });
```

```
        }

        const checkPass = await bcrypt.compare(body._password,
row[0].password);

        if (checkPass === true) {
            req.session.userID = row[0].id;
            return res.redirect('/');
        }

        res.render('login', {
            error: 'Invalid Password.'
        });


    }
    catch (e) {
        next(e);
    }

}
```

Для добавления стилистики нашего приложения необходимо добавить файл style.css внутри общей папки приложения. Код представлен в листинге 12.

Листинг 12 – Код в файле style.css

```
*,
*::before,
*::after {
    box-sizing: border-box;
}

html {
    -webkit-text-size-adjust: 100%;
    -webkit-tap-highlight-color: rgba(0, 0, 0, 0);
    font-size: 16px;
}

body {
    background-color: #f7f7f7;
    font-family: "Ubuntu", sans-serif;
    margin: 0;
    padding: 0;
    color: #222222;
    overflow-x: hidden;
    overflow-wrap: break-word;
    -moz-osx-font-smoothing: grayscale;
    -webkit-font-smoothing: antialiased;
    padding: 50px;
}

.container {
    background-color: white;
    max-width: 450px;
    margin: 0 auto;
    padding: 40px;
```

```
    box-shadow: 0 1rem 3rem rgba(0, 0, 0, 0.175);
    border-radius: 3px;
}

.container  h1  {
    margin: 0 0 40px 0;
    text-align: center;
}

input,
button {
    font-family: "Ubuntu", sans-serif;
    outline: none;
    font-size: 1rem;
}

.input {
    padding: 10px;
    width: 100%;
    margin-bottom: 10px;
    border: 1px solid #bbbbbb;
    border-radius: 3px;
}

.input:hover {
    border-color: #999999;
}

.input:focus {
    border-color: #0d6efd;
}

[type="submit"] {
    background: #0d6efd;
    color: white;
    border: 1px solid rgba(0, 0, 0, 0.175);
    border-radius: 3px;
    padding: 12px 0;
    cursor: pointer;
    box-shadow: 0 0.125rem 0.25rem rgba(0, 0, 0, 0.075);
    margin-top: 5px;
    font-weight: bold;
    width: 100%;
}

[type="submit"]:hover {
    box-shadow: 0 0.5rem 1rem rgba(0, 0, 0, 0.15);
}

label {
    font-weight: bold;
}

.link {
    margin-top: 10px;
    text-align: center;
}
```

```
.link a {
    color: #0d6efd;
}

.success-msg,
.err-msg {
    color: #dc3545;
    border: 1px solid #dc3545;
    padding: 10px;
    border-radius: 3px;
}

.success-msg {
    color: #ffffff;
    background-color: #20c997;
    border-color: rgba(0, 0, 0, 0.1);
}

.profile {
    text-align: center;
}

.profile .img {
    font-size: 50px;
}

.profile h2 {
    margin-bottom: 3px;
    text-transform: capitalize;
}

.profile span {
    display: block;
    margin-bottom: 20px;
    color: #999999;
}

.profile a {
    display: inline-block;
    padding: 10px 20px;
    text-decoration: none;
    border: 1px solid #dc3545;
    color: #dc3545;
    border-radius: 3px;
}

.profile a:hover {
    border-color: rgba(0, 0, 0, 0.1);
    background-color: #dc3545;
    color: #ffffff;
}
```

Последним действием является оформление главного файла index.js, представленного в листинге 13.

Листинг 13 – Код в файле index.js

```
const express = require('express');
const session = require('express-session');
const path = require('path');
const routes = require('./routes');
const app = express();

app.set('views', path.join(_dirname, 'views'));
app.set('view engine', 'ejs');

app.use(express.urlencoded({ extended: false }));
app.use(session({
    name:  'session',
    secret: 'my_secret',
    resave: false,
    saveUninitialized: true,
    cookie: {
        maxAge: 3600 * 1000, // 1hr
    }
}));

app.use(express.static(path.join(_dirname, 'public')));
app.use(routes);

app.use((err, req, res, next) => {
    // console.log(err);
    return res.send('Internal Server Error');
});

app.listen(3000, () => console.log('Server is runngin on port 3000'));
```

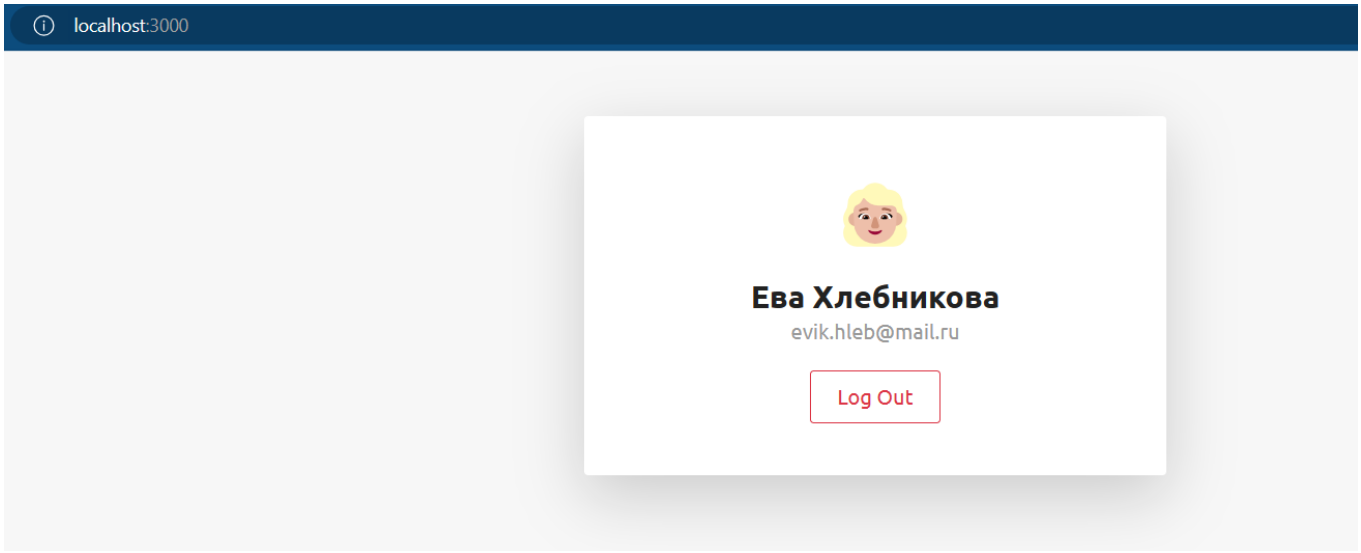

Рисунок 2 – Регистрация в практике

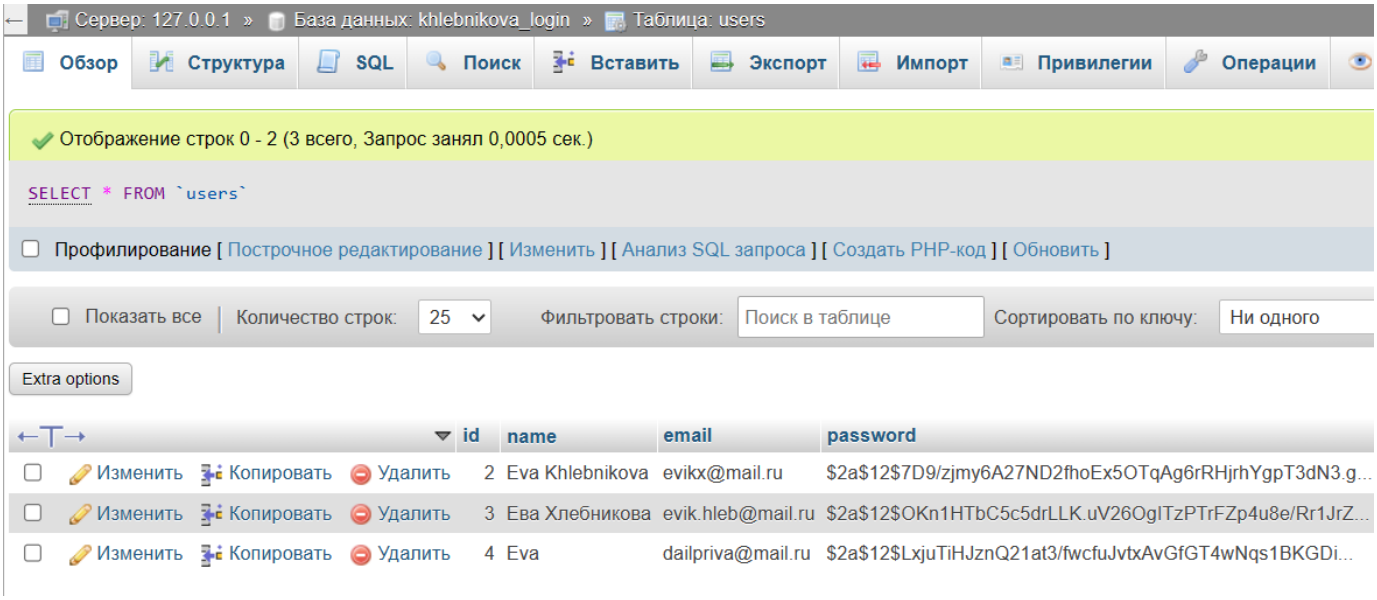

Рисунок 3 – Добавление учётной записи в базу данных

3.2 Далее в качестве индивидуального задания 1 необходимо проверить работоспособность полученного приложения. Результаты занести скриншотами из базы данных в отчет

3.3 В качестве индивидуального задания 2 необходимо интегрировать дополнительный способ авторизации через Google и Github.

Для авторизации через Google необходимо инициализировать дополнительные модули через команду, представленную в листинге 14.

Листинг 14 – Установка дополнительных модулей

```
npm install express passport passport-google-oauth2 cookie-session
```

Далее необходимо добавить в корневую структуру добавить файл passport.js, содержащий код, представленный в листинге 15.

Листинг 15 – Код файла passport.js

```
const passport = require('passport');
const GoogleStrategy = require('passport-google-oauth2').Strategy;

passport.serializeUser((user , done) => {
    done(null , user);
})
passport.deserializeUser(function(user, done) {
    done(null, user);
```

```
});

passport.use(new GoogleStrategy({
    clientID:"YOUR ID", // Данные из вашего аккаунта.
    clientSecret:"YOUR SECRET", // Данные из вашего аккаунта.
    callbackURL:"http://localhost:4000/auth/callback",
    passReqToCallback:true
  },
  function(request, accessToken, refreshToken, profile, done) {
    return done(null, profile);
  }
));
```

Для получения данных с платформы Google необходимо:

1) Перейти на сайт https://console.cloud.google.com/ и зайти в раздел API's & Services

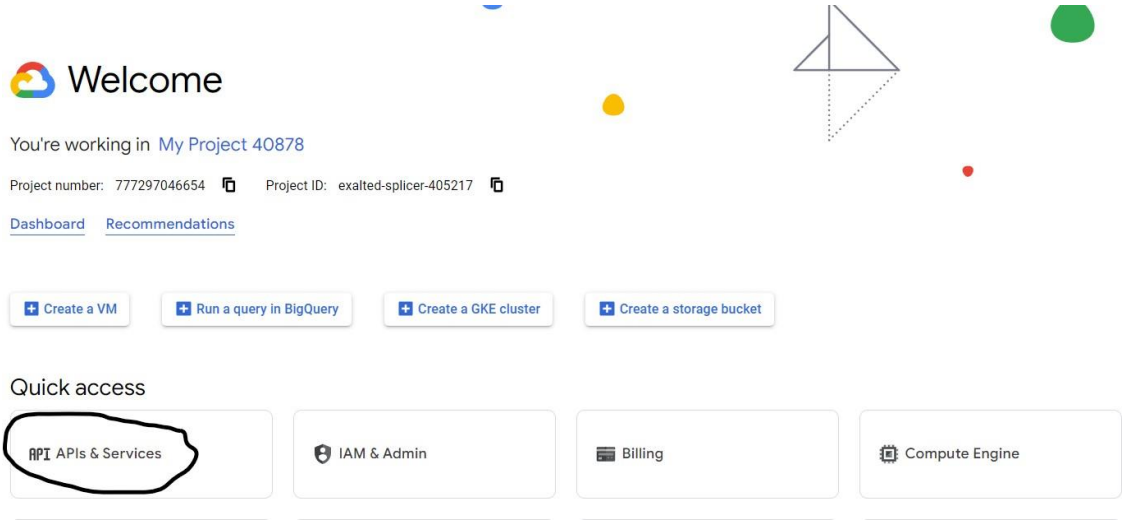

Рисунок 5 – Главная страница Google Cloud

Далее необходимо зайти во вкладку Credentials, а затем Create Credentials и выбрать Create OAuth client ID. Далее необходимо выбрать тип приложения "Web application" (при условии разработки веб-приложения). В ячейке «Authorized redirect URIs» нажать вкладку «Add URL» и добавить две ссылки, как показано на рисунке 3 и нажать кнопку «Create».

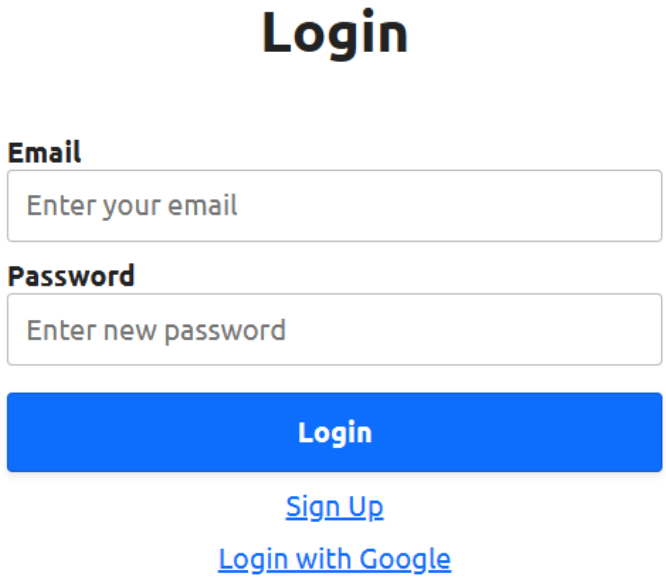

Рисунок 6 – Добавление варианта авторизации через Google

После этого появится ID и Secret, как показано на рисунке 4, которые необходимо добавить в код.

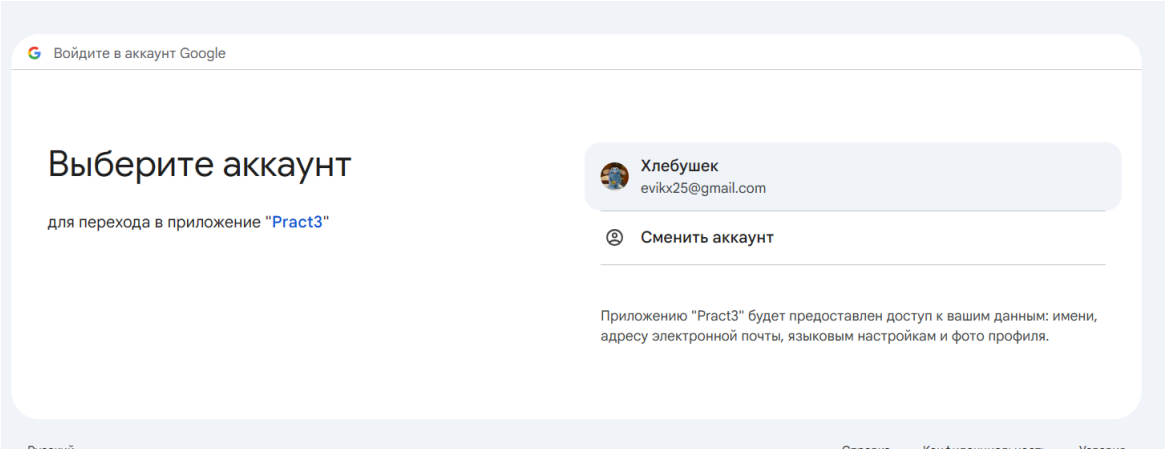

Рисунок 7 – Процесс авторизации через Google

Авторизацию через github необходимо изучить самостоятельно и интегрировать в ваше решение.

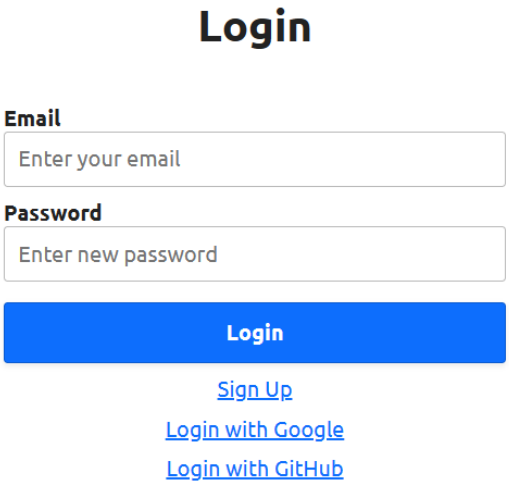

Рисунок 9 – Добавление варианта авторизации через GitHub

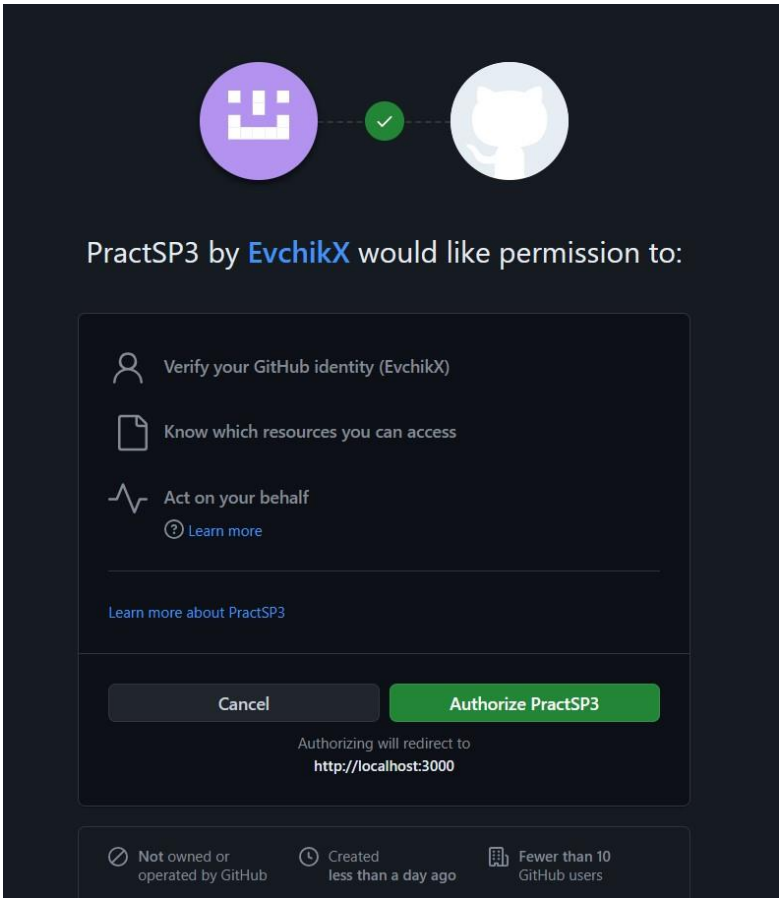

Рисунок 10 – Процесс авторизации через GitHub

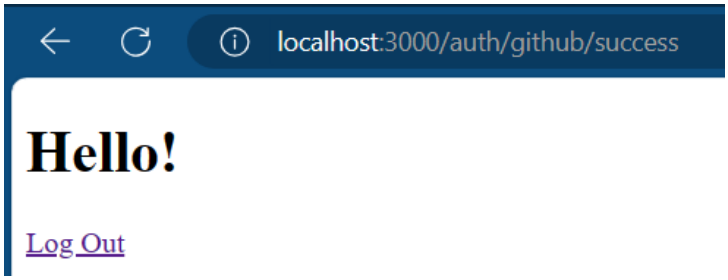

Рисунок 11 – Результат успешной авторизации через GitHub

4. Контрольные вопросы:

4.1 Роль аутентификации в клиент-серверных приложениях состоит в проверке подлинности пользователей, то есть подтверждении их идентификационных данных, таких как логин и пароль. Авторизация же определяет, какие действия и ресурсы пользователь имеет право использовать после успешной аутентификации.

4.2 В клиент-серверных приложениях можно использовать различные методы аутентификации, такие как:

- HTTP Basic Authentication
- Формы для ввода логина и пароля
- OAuth
- JWT (JSON Web Tokens)

4.3 Для обеспечения безопасности процесса аутентификации в клиент-серверных приложениях можно использовать шифрование паролей, защищенные соединения (например, HTTPS), а также использование надежных алгоритмов аутентификации.

4.4 Для обеспечения безопасности при аутентификации в клиент-серверных приложениях могут использоваться различные технологии и протоколы, такие как:

- TLS/SSL
- OAuth
- OpenID Connect
- SAML (Security Assertion Markup Language)

4.5 Многофакторная аутентификация - это процесс аутентификации, который требует от пользователя предоставить несколько форм идентификации, таких как пароль, SMS-код, отпечаток пальца и т.д. Преимущества многофакторной аутентификации в клиент-серверных приложениях включают усиление безопасности за счет добавления дополнительных уровней проверки подлинности.

4.6 Основные принципы при осуществлении процесса авторизации в клиент-серверных приложениях включают в себя:

- Принцип наименьших привилегий: пользователю должны предоставляться только те разрешения, которые необходимы для выполнения его задач.

- Прозрачность: пользователю должна быть предоставлена информация о его текущих правах доступа и возможностях.

- Аудит: следует вести журнал всех попыток доступа и изменений прав доступа.

4.7 Для защиты от несанкционированного доступа к ресурсам в клиент-серверных приложениях можно использовать методы аутентификации и авторизации, а также механизмы контроля доступа, такие как ролевая модель и ACL (Access Control Lists).

4.8 Сессионная аутентификация в клиент-серверных приложениях реализуется путем создания уникальной сессии для каждого пользователя после успешной аутентификации. Это может быть достигнуто путем использования куки или токена сессии, который передается между клиентом и сервером при каждом запросе.

4.9 Для обеспечения безопасности при передаче учетных данных между клиентом и сервером в клиент-серверных приложениях можно использовать шифрование данных с использованием SSL/TLS, а также хэширование паролей перед их передачей.

4.10 Для обработки ошибок аутентификации и авторизации в клиент-серверных приложениях можно использовать механизмы обработки исключений, а также предоставление информативных сообщений об ошибках пользователю. Важно также вести аудит действий пользователей и предоставлять возможность блокировки учетных записей при обнаружении подозрительной активности.