Федеральное государственное бюджетное образовательное учреждение высшего

образования

«Чувашский государственный университет им. И.Н. Ульянова»

Кафедра вычислительной техники

Лабораторная работа № 2

**Разработка Web-приложения, взаимодействующего с СУБД**

**PostgreSQL.**

Выполнил: Иванов В.С.

студент группы ИВТ-41-22

Проверил: Марков А.В.

Чебоксары, 2025

Цель работы.

Разработка web-приложения на node.js и express, взаимодействующего с

СУБД PostgreSQL. Настройка подключения, изучение вопросов безопасности,

тестирования.

Теоретическая часть.

Node.js — это среда выполнения JavaScript на стороне сервера, основанная на движке V8. Она позволяет разрабатывать масштабируемые и высокопроизводительные веб-приложения.

Express.js — это минималистичный фреймворк для Node.js, который упрощает создание серверных приложений, предоставляя инструменты для маршрутизации, middleware и работы с HTTP-запросами.

PostgreSQL — это мощная реляционная СУБД с открытым исходным кодом. Для подключения к PostgreSQL из Node.js используется библиотека pg-promise, которая предоставляет удобный интерфейс для выполнения SQL-запросов.

Для обеспечения безопасности приложения используются:

* Аутентификация: Проверка логина и пароля пользователя.
* Авторизация: Проверка прав доступа пользователя к определённым ресурсам.
* Сессии: Хранение состояния пользователя между запросами (например, с помощью cookies).

SQL-инъекция — это атака, при которой злоумышленник вставляет вредоносный SQL-код в запрос.

Методы защиты:

* Использование параметризованных запросов:
* Экранирование входных данных.
* Ограничение прав пользователей БД.

Git — это распределённая система контроля версий, которая позволяет:

* Отслеживать изменения в коде.
* Работать в команде.
* Возвращаться к предыдущим версиям проекта.

1. **Запуск простейшего приложения**



Рис.1. Запуск простейшего приложения

1. **Подключение к СУБД.**

Сгенерируем каркас приложения

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис.2. Генерация каркаса приложения

Установим pg-promise

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис.3. Установка pg-promise

Добавим в код приложения после var logger = require('morgan');  
  
const pgp = require('pg-promise')(/\* options \*/)  
const db = pgp('postgres://postgres:[63403@localhost](mailto:password@host):5432/laba2')

Добавим перед module.exports = app;

db.one('SELECT $1 AS value', 123)

.then((data) => {

console.log('DATA:', data.value)

})

.catch((error) => {

console.log('ERROR:', error)

})

Добавим перед app.use('/', indexRouter);

app.use(function(req,res,next){

req.db = db;

next();

})

Для маршрута users уже есть сгенерированный файлы в routes/users.js.

Заменим в нём тело функции

router.get('/', function(req, res, next) {

res.send('respond with a resource');

});

на

router.get('/', async function(req, res, next) {

let users = await req.db.any('SELECT \* FROM users')

console.log(users)

res.render('users/list', { title: 'Пользователи', users: users })

});

Создадим в папке views папку users, в папке users файл list.pug с содержимым

extends ../layout

block content

h1= title

table

tr

th id

th ФИО

each item in users

tr

td= item.id

td= item.fio

Отображение <http://localhost:3000/users>

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис.4. Отображение элементов

1. **Модель доступа. Идентификация.**

3.1. **Создадим необходимые таблицы в бд**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис.5. Таблицы в БД.

* 1. **Добавим модуль авторизации для идентификации пользователей**

Добавим в код приложения после app.use('/users', usersRouter);

var api = require('./routes/api');

app.use('/api', api);

var api\_auth = require('./routes/api/auth');

api.use('/auth', api\_auth);

Создадим файл api.js в корневом каталоге с содержимым:

var express = require('express')

var router = express.Router()

module.exports = router

Создадим папку api в папке routes, в папке api создадим файл с содержимым:

var express = require('express');  
var router = express.Router();  
  
router.post('/login', function(req, res) {  
 var cookie = session.login(req, req.body.login, req.body.password)  
 if (cookie) {

res.cookie('app\_user', cookie, { maxAge: 43200\*1000, httpOnly: true, path: '/' });  
 res.json({ msg: ''})  
 return;  
 }  
 res.json({ msg: 'Неверный логин/пароль'})  
});

router.post('/logout', function(req, res) {  
 var user = session.auth(req)  
 if (user) {  
 res.clearCookie('app\_user', { path: '/' });  
 session.logout(user)  
 }  
 res.json({ msg: '' })  
});

module.exports = router;

Помимо модуля авторизации нам необходим модуль работы с сессиями.

Вставим после var app = express();

session = require("./session.js")

Создать файл session.js в корневом каталоге с содержимым

const md5 = require('md5')

const crypto = require('crypto')

exports.sessions = {

}

exports.login = function(req, login, pass) {

var cookies = req.cookies

var secret = cookies['app\_user']

var user = await req.db.one('SELECT \* FROM users WHERE login = $1', login)

if (user && (user.pass\_hash == md5(pass))) {

var secret = 'secret';

var hash = crypto.createHmac('sha256', secret)

.update(login)

.digest('hex');

var cookie = login + '--' + hash;

exports.sessions[login] = {

active: 1,

timestamp: new Date().getTime(),

}

exports.sessions[login].user = user

return cookie;

}

return 0;

}

exports.auth = function(req) {

var cookies = req.cookies

var secret = cookies['app\_user']

if (!secret) {

return {}

}

var res = secret.split('--');

if(!res.length) {

return {}

}

var session = exports.sessions[res[0]]

if (!session) {

return {};

}

var current\_timestamp = new Date().getTime()

if (!session.active || ((current\_timestamp - session.timestamp) > 43200\*1000)) {

return 0;

}

return session;

}

exports.logout = function(login) {

exports.sessions[login] = {

}

}

Установиv библиотеку md5:

npm install --save md5

Модифицируем главную страницу приложения

layout.pug:

doctype html

html

head

title= title

link(rel='stylesheet', href='/stylesheets/style.css')

script(src='/javascripts/jquery.min.js')

script(src='/javascripts/layout.js')

body

block content

div#login\_popup.q-window

div.q-window-block(style="width:400px;height:220px;margin:-150px 0 0 -400px")

div.q-window-header Вход

a#login\_popup\_close(style="position: relative;left: 300px;") X

div.q-window-body.q-form

div.q-window-filters-form

div.q-form-data

label(for="inpLogin") Логин

input#inpLogin(type="text")

div.q-form-data

label(for="inpPassword") Пароль

input#inpPassword(type="password")

div.q-buttons

a#submit\_login Войти

index.pug:

extends layout

block content

br

div

ul

if user

if can\_view\_clients

li

a(href='/clients') Клиенты

if can\_view\_users

li

a(href='/users') Пользователи

li

a(href='', id='Logout') Выйти

else

li

a(href='', id='Login') Войти

Теперь модифицируем маршрут, который отрисовывает главную страницу(routes/index.js):

var express = require('express');

var router = express.Router();

router.get('/', function(req, res, next) {

var user = session.auth(req).user

var can\_view\_users = user && user.id\_role == 1 ? true : false

var can\_view\_clients = true

res.render('index', {

title: "Главная страница",

user: user,

can\_view\_users: can\_view\_users,

can\_view\_clients: can\_view\_clients,

})

});

module.exports = router;

Добавим ссылки на новые страницы на главной странице

В index.js добавим:

var clientsRouter = require('./routes/clients');

var ordersRouter = require('./routes/orders');

var paymentsRouter = require('./routes/payments');

app.use('/clients', clientsRouter);

app.use('/orders', ordersRouter);

app.use('/payments', paymentsRouter);

1. **Модификация данных**

В views создадим папки /clients, /orders, /payments

/clients/list.pug:

extends ../layout

block content

    h1= title

    table

        tr

            th id

            th Имя

        each item in clients

            tr

                td= item.id

                td= item.label

/orders/list.pug:

extends ../layout

block content

    h1= title

    button(id='create\_order') Создать

    table

        tr

            th id

            th Название

            th Статус

            th Клиент

            th Сумма

        each item in orders

            tr

                td= item.id

                td

                    a(href='/orders/' + item.id)= item.label

                td= item.order\_status\_label

                td= item.client\_label

                td= item.amount

    div#create\_order\_popup.q-window

          div.q-window-block(style="width:400px;height:220px;margin:-150px 0 0 -400px")

            div.q-window-header Создать заказ

              a#create\_order\_popup\_close(style="position: relative;left: 300px;") X

            div.q-window-body.q-form

              div.q-window-filters-form

                div.q-form-data

                  label(for="inpLabel") Название

                  input#inpLabel(type="text")

                div.q-form-data

                  label(for="inpClient") Клиент

                  select#inpClient

                    each item in clients

                        option(value= item.id)= item.label

                div.q-form-data

                  label(for="inpAmount") Сумма

                  input#inpAmount(type="text")

            div.q-buttons

              a#submit\_create\_order Создать

              a#cancel\_create\_order Отменить

/orders/view/pug:

extends ../layout

block content

    h1= title

    div

      form

        fieldset

          label Название

          input(type='text', disabled=true, value=order.label)

          br

          label Клиент

          input(type='text', disabled=true, value=order.client\_label)

          br

          label Статус

          input(type='text', disabled=true, value=order.order\_status\_label)

          br

          label Сумма

          input(type='text', disabled=true, value=order.amount)

/payments/list.pug:

extends ../layout

block content

    h1= title

    table

        tr

            th id

            th Заказ

            th Тип платежа

            th Сумма платежа

        each item in payments

            tr

                td= item.id

                td= item.order\_label

                td= item.payment\_type\_label

                td= item.amount

В /routes создадим clients.js, orders.js, payments.js

Clients.js:

var express = require('express');

var router = express.Router();

router.get('/', async function(req, res, next) {

    let clients = await req.db.any(`

        SELECT

            \*

        FROM

            clients

    `)

    console.log(clients)

    res.render('clients/list', { title: 'Клиенты', clients: clients })

});

module.exports = router;

orders.js:

var express = require('express');

var router = express.Router();

router.get('/', async function(req, res, next) {

    let orders = await req.db.any(`

        SELECT

            orders.id AS id,

            orders.label AS label,

            order\_statuses.label AS order\_status\_label,

            clients.label AS client\_label,

            orders.amount AS amount

        FROM

            orders

        INNER JOIN

            clients ON clients.id = orders.id\_client

        INNER JOIN

            order\_statuses ON order\_statuses.id = orders.id\_status

    `)

    console.log(orders)

     let clients = await req.db.any(`

        SELECT

            \*

        FROM

            clients

    `)

    console.log(clients)

    res.render('orders/list', { title: 'Заказы', orders: orders, clients: clients })

});

router.post('/create', async function(req, res, next) {

    let order = req.body

    await req.db.none('INSERT INTO orders(label, id\_client, amount) VALUES(${label}, ${id\_client}, ${amount})', order);

    res.send({msg: ''})

});

router.get('/:id', async function(req, res) {

    let id = req.params.id

    let order = await req.db.one(`

        SELECT

            orders.id AS id,

            orders.label AS label,

            order\_statuses.label AS order\_status\_label,

            clients.label AS client\_label,

            orders.amount AS amount

        FROM

            orders

        INNER JOIN

            clients ON clients.id = orders.id\_client

        INNER JOIN

            order\_statuses ON order\_statuses.id = orders.id\_status

        WHERE

            orders.id = ${id}

    `)

    res.render('orders/view', { title: 'Заказ' + order.label, order: order })

});

module.exports = router;

payments.js:

var express = require('express');

var router = express.Router();

router.get('/', async function(req, res, next) {

    let payments = await req.db.any(`

        SELECT

            payments.id AS id,

            orders.label AS order\_label,

            payment\_types.label AS payment\_type\_label,

            payments.amount AS amount

        FROM

            payments

        INNER JOIN

            payment\_types ON payment\_types.id = payments.id\_payment\_type

        INNER JOIN

            orders ON orders.id = payments.id\_order

    `)

    console.log(payments)

    res.render('payments/list', { title: 'Платежи', payments: payments })

});

module.exports = router;

В /public/javascript создадим users.js, orders.js

Users.js:

$(document).ready(function(){

    $.ajax({

        type: 'GET',

        url: '/api/users',

        dataType: 'JSON'

    }).done(function( response ) {

        response.users.forEach(user => {

            $('#tbl\_users').append(

                `<tr>

                    <td>${user.id}

                    <td>${user.login}

                    <td>${user.fio}

                    <td>${user.role\_label}

                </tr>`

            )

        })

    });

})

Orders.js:

$(document).ready(function(){

    $('#create\_order').click(function(e){

        $('#create\_order\_popup').show()

    })

    $('#create\_order\_popup\_close').click(function(e){

        $('#create\_order\_popup').hide()

    })

    $('#cancel\_create\_order').click(function(e){

        $('#create\_order\_popup').hide()

    })

    $('#submit\_create\_order').click(function(e){

        e.preventDefault()

        let data = {

            label:    $('#inpLabel').val(),

            id\_client: $('#inpClient').val(),

            amount: $('#inpAmount').val(),

        }

        $.ajax({

            type: 'POST',

            data: data,

            url: '/orders/create',

            dataType: 'JSON'

        }).done(function( response ) {

            if (response.msg === '') {

                alert('Заказ создан')

                window.location.reload()

            }

            else {

                alert(response.msg)

            }

        });

    })

});

1. **Тестирование**

Установим

npm install jest --save-dev

npm install supertest --save-dev

npm install cross-env --save-dev

Изменим package.json:

"scripts": {

...

"test": "cross-env NODE\_ENV=test jest --testTimeout=10000"

},

"jest": {

"testEnvironment": "node",

"coveragePathIgnorePatterns": [

"/node\_modules/"

]

},

Для тестирования изменим схему отрисовки экрана users.

App.js:

var api\_users = require('./routes/api/users');

api.use('/users', api\_users);

routes.users.js:

var express = require('express');

var router = express.Router();

router.get('/', async function(req, res, next) {

res.render('users/list', { title: 'Пользователи' })

});

module.exports = router;

views/users/list.pug:

extends ../layout

block content

script(src='/javascripts/users.js')

h1= title

table#tbl\_users

tr

th id

th Имя

th Логин

th Роль

routes/api/users.js:

var express = require('express');

var router = express.Router();

router.get('/', async function(req, res, next) {

let users = await req.db.any(`

SELECT

users.id AS id,

users.login AS login,

users.fio AS fio,

roles.label AS role\_label

FROM

users

INNER JOIN roles ON roles.id = users.id\_role

`)

console.log(users)

res.json({users: users })

});

module.exports = router;

public/javascripts/users/js:

$(document).ready(function(){

$.ajax({

type: 'GET',

url: '/api/users',

dataType: 'JSON'

}).done(function( response ) {

response.users.forEach(user => {

$('#tbl\_users').append(

`<tr>

<td>${user.id}

<td>${user.login}

<td>${user.fio}

<td>${user.role\_label}

</tr>`

)

})

});

})

Напишем тесты в файле test.js:  
const server = require('./app.js');

const supertest = require('supertest');

const requestWithSupertest = supertest(server);

describe('User Endpoints', () => {

// 1. Тест на получение всех пользователей (исходный тест)

it('GET /api/users should show all users', async () => {

const res = await requestWithSupertest.get('/api/users');

expect(res.status).toEqual(200);

expect(res.type).toEqual(expect.stringContaining('json'));

expect(res.body).toHaveProperty('users');

expect(res.body.users.length > 0);

expect(res.body.users[0]).toHaveProperty('id');

expect(res.body.users[0]).toHaveProperty('login');

expect(res.body.users[0]).toHaveProperty('fio');

expect(res.body.users[0]).toHaveProperty('role\_label');

});

// 2. Проверяет, что ответ — JSON

it('GET /api/users should return JSON', async () => {

const res = await requestWithSupertest.get('/api/users');

expect(res.type).toBe('application/json');

});

// 3. Проверяет, что в ответе есть массив users

it('GET /api/users should return array of users', async () => {

const res = await requestWithSupertest.get('/api/users');

expect(Array.isArray(res.body.users)).toBe(true);

});

// 4. Проверяет, что у каждого пользователя есть id

it('GET /api/users users should have "id" field', async () => {

const res = await requestWithSupertest.get('/api/users');

res.body.users.forEach(user => {

expect(user).toHaveProperty('id');

});

});

// 5. Проверяет, что в данных есть тестовый пользователь "admin"

it('GET /api/users should contain test admin', async () => {

const res = await requestWithSupertest.get('/api/users');

const hasAdmin = res.body.users.some(user => user.login === 'admin');

expect(hasAdmin).toBe(true);

});

// 6. Проверяет, что API вообще работает

it('GET /api/users should return status 200', async () => {

const res = await requestWithSupertest.get('/api/users');

expect(res.status).toEqual(200);

});

// 7. Тест на обновление пользователя

it('PUT /api/users/:id should update user', async () => {

const testUserId = 1; // Предполагаем, что пользователь с ID=1 существует

const updates = { fio: 'Обновленное Имя' };

const res = await requestWithSupertest.put(`/api/users/${testUserId}`).send(updates);

expect(res.status).toEqual(200);

expect(res.body.fio).toBe(updates.fio);

});

// 8. Тест на удаление пользователя

it('DELETE /api/users/:id should delete user', async () => {

const testUserId = 2; // Предполагаем, что пользователь с ID=2 существует

const res = await requestWithSupertest.delete(`/api/users/${testUserId}`);

expect(res.status).toEqual(200);

expect(res.body).toHaveProperty('message', 'User deleted');

});

// 9. Тест на фильтрацию пользователей по роли

it('GET /api/users?role=Администратор should filter by role', async () => {

const role = 'Администратор';

const res = await requestWithSupertest.get(`/api/users?role=${role}`);

expect(res.status).toEqual(200);

expect(res.body.users.length).toBeGreaterThan(0);

expect(res.body.users.every(user => user.role\_label === role)).toBe(true);

});

});

Запустим тест командой npm test

Изображение выглядит как текст, Шрифт, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис.6. Результаты тестов

Попробуем сделать sql-инъекцию

Залогинимся через тестового сотрудника

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис.7. Логин через сотрудника

Перейдем во вкладку заказов

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис.8. Страница заказов.

Заменим адресную строку на <http://localhost:3000/orders/88%20UNION%20SELECT%201%20AS%20id,%20users.fio%20AS%20label,%20users.login%20AS%20order_status_label,%20users.pass%20AS%20client_label,%201%20AS%20amount%20FROM%20users%20LIMIT%201>

В результате отобразятся данные из таблицы users

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рис.9. Результат SQL-инъекции.

Чтобы обезопаситься от подобной инъекции, можно сделать различные валидации, чтобы на вход например могло попасть только число, а именно id, а не строка.

Вывод: разработал web-приложение на node.js и express, взаимодействующее с

СУБД PostgreSQL. Настроил подключение, изучил вопросы безопасности,

тестирования.