ГУАП

КАФЕДРА № 33

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доцент		К.А. Жиданов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

по дисциплине: ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 3337		А.А. Беляев
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Разработать веб-приложение на платформе Node.js с использованием базы данных MySQL для управления списком задач. Обеспечить возможность авторизации пользователей, добавления и отображения списка задач через веб-интерфейс.

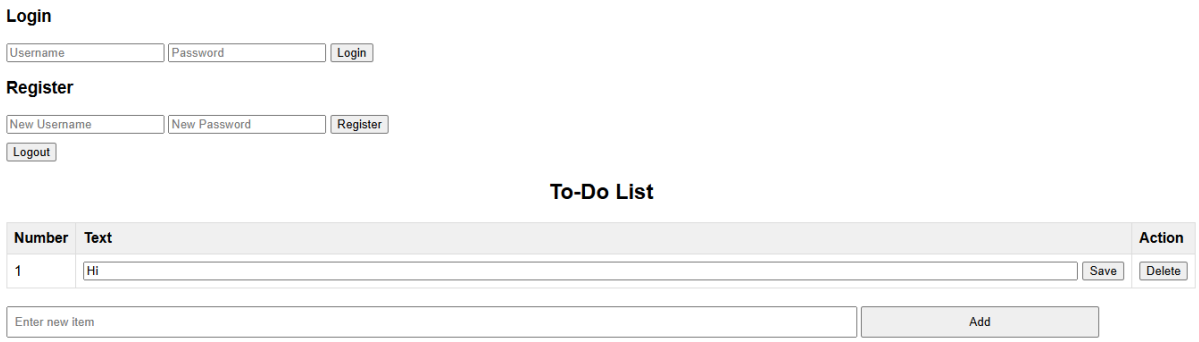
ЗАДАЧИ

1. Реализация серверной логики с использованием Node.js.
2. Настройка базы данных MySQL и создание необходимых таблиц.
3. Реализация функций регистрации и авторизации пользователей.
4. Организация отображения, добавления, удаления и редактирования задач в веб-интерфейсе.
5. Простейшая интеграция Telegram-бота.
6. Настройка базы данных MySQL и создание необходимых таблиц.
7. Реализация функций регистрации и авторизации пользователей.
8. Организация отображения и добавления задач в веб-интерфейсе.

ОПИСАНИЕ ИНТЕРФЕЙСА

Интерфейс включает в себя следующие блоки:

- Форма авторизации (вход по логину и паролю);
- Форма регистрации нового пользователя;
- Кнопка выхода (logout);
- Таблица с задачами: номер, текст задачи, действие;
- Форма добавления новой задачи.
- Кнопки удаления и редактирования задачи.



(Рисунок 1. Интерфейс веб-приложения)

Все элементы размещены по центру и оформлены с использованием базового CSS для повышения читаемости и удобства.

ОПИСАНИЕ ЛОГИКИ РАБОТЫ

Регистрация:

- Пользователь вводит имя и пароль в форму регистрации;
- Данные отправляются методом POST на сервер по маршруту `/register`;
- Сервер записывает данные в таблицу `users`.

Авторизация:

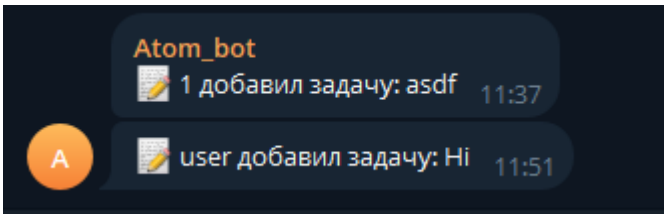
- Через POST-запрос на `/login` происходит проверка введённых данных по таблице `users`;
- При совпадении данных пользователь допускается к интерфейсу задач.

Работа с задачами:

- Задачи хранятся в таблице `items` с полями `id`, `user_id`, `text`;
- Добавление задачи происходит через форму с POST-запросом на `/add`;
- Удаление задачи реализуется через кнопку удаления и запрос на сервер по соответствующему маршруту;
- Редактирование текста задачи возможно с помощью формы и запроса на сервер;
- Таблица задач обновляется при каждом заходе пользователя в систему или обновления страницы веб-приложения.
- Удаление и редактирование задачи происходит через кнопки с POST маршрутом /delete и /edit соответственно

Telegram-бот:

- Бот отправляет сообщения в Telegram-группу при добавлении, удалении или редактировании задач (например: **`user`** `добавил задачу:` **`Hi`**);



(Рисунок 2. Пример работы бота)

- Реакции на команды не реализованы: бот не принимает команды от пользователей, а служит исключительно для нотификации;

СТРУКТУРА БАЗЫ ДАННЫХ

```
CREATE DATABASE IF NOT EXISTS todolist;
USE todolist;

CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
```

```
    password VARCHAR(255) NOT NULL
);

CREATE TABLE IF NOT EXISTS items (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    text VARCHAR(255) NOT NULL
);
```

ВЫВОД

Разработанное веб-приложение успешно выполняет основные функции: регистрация и авторизация пользователя, добавление и отображение задач, хранение информации в базе данных. Проект выполнен в соответствии с требованиями лабораторной работы и может быть дополнен расширенным функционалом (например, редактированием задач, интеграцией с Telegram-ботом и другими возможностями).

Приложения:

А. Код HTML-файла `index.html`

Б. Код серверной логики (`index.js`) с реализацией Telegram-бота

Приложение А

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>To-Do List</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }
        #todoList {
            border-collapse: collapse;
            width: 70%;
            margin: 0 auto;
        }
        #todoList th, #todoList td {
            border: 1px solid #ddd;
            padding: 8px;
            text-align: left;
        }
        #todoList th {
            background-color: #f0f0f0;
        }
        #todoList th:first-child, #todoList th:last-child {
            width: 5%;
        }
        #todoList th:nth-child(2) {
            width: 90%;
        }
        .add-form {
            margin-top: 20px;
            width: 70%;
            margin: 20px auto;
        }
        .add-form input[type="text"] {
            padding: 8px;
            width: 70%;
        }
        .add-form button {
            padding: 8px;
            width: 20%;
        }
        .auth-forms {
            width: 70%;
            margin: 20px auto;
        }
    </style>
</head>
<body>
```

```
<div class="auth-forms">
    <h3>Login</h3>
    <form method="POST" action="/login">
        <input type="text" name="username" placeholder="Username" required>
        <input type="password" name="password" placeholder="Password"
required>
        <button type="submit">Login</button>
    </form>

    <h3>Register</h3>
    <form method="POST" action="/register">
        <input type="text" name="username" placeholder="New Username"
required>
        <input type="password" name="password" placeholder="New Password"
required>
        <button type="submit">Register</button>
    </form>

    <form method="GET" action="/logout" style="margin-top: 10px;">
        <button type="submit">Logout</button>
    </form>
</div>

<h2 style="text-align: center;">To-Do List</h2>

<table id="todoList">
    <thead>
        <tr>
            <th>Number</th>
            <th>Text</th>
            <th>Action</th>
        </tr>
    </thead>
    <tbody id="listBody">
        {{rows}}
    </tbody>
</table>

<form class="add-form" method="POST" action="/add">
    <input type="text" name="text" placeholder="Enter new item" required>
    <button type="submit">Add</button>
</form>

</body>
</html>
```

Приложение Б

```
const http = require('http');
const fs = require('fs');
const path = require('path');
const mysql = require('mysql2/promise');
const crypto = require('crypto');
const https = require('https');
const { parse } = require('querystring');
const PORT = 3000;

const dbConfig = {
    host: 'localhost',
    user: 'atom',
    password: 'qweqwe123',
    database: 'todolist',
};

const TELEGRAM_BOT_TOKEN = 'YOUR_BOT_TOKEN';
const TELEGRAM_CHAT_ID = 'YOUR_CHAT_ID';

function hashPassword(password) {
    return crypto.createHash('sha256').update(password).digest('hex');
}

function parseCookies(req) {
    const raw = req.headers.cookie || '';
    return Object.fromEntries(raw.split('; ').filter(Boolean).map(c =>
c.split('=')));
}

async function notifyTelegram(taskText, username = 'Unknown') {
    const message = `📝 ${username} добавил задачу: ${taskText}`;
    const url =
`https://api.telegram.org/bot${TELEGRAM_BOT_TOKEN}/sendMessage?chat_id=${TELE
GRAM_CHAT_ID}&text=${encodeURIComponent(message)}`;
    https.get(url, res => res.on('data', () => {})).on('error', err =>
console.error('Telegram error:', err));
}

async function retrieveListItems(userId) {
    const connection = await mysql.createConnection(dbConfig);
    const [rows] = await connection.execute('SELECT id, text FROM items WHERE
user_id = ?', [userId]);
    await connection.end();
    return rows;
}

async function addItemToDatabase(text, userId, username) {
    const connection = await mysql.createConnection(dbConfig);
    await connection.execute('INSERT INTO items (text, user_id) VALUES (?,
?)', [text, userId]);
```

```
    await connection.end();
    await notifyTelegram(text, username);
}

async function updateItemInDatabase(id, text) {
    const connection = await mysql.createConnection(dbConfig);
    await connection.execute('UPDATE items SET text = ? WHERE id = ?', [text,
id]);
    await connection.end();
}

async function deleteItemFromDatabase(id) {
    const connection = await mysql.createConnection(dbConfig);
    await connection.execute('DELETE FROM items WHERE id = ?', [id]);
    await connection.end();
}

async function getHtmlRows(userId) {
    const todoItems = await retrieveListItems(userId);
    return todoItems.map((item, index) => `
        <tr>
            <td>${index + 1}</td>
            <td>
                <form method="POST" action="/edit" style="display: flex; gap:
5px;">
                    <input type="hidden" name="id" value="${item.id}">
                    <input type="text" name="text" value="${item.text}"
style="flex: 1;">
                    <button type="submit">Save</button>
                </form>
            </td>
            <td>
                <form method="POST" action="/delete" onsubmit="return
confirm('Delete this item?');">
                    <input type="hidden" name="id" value="${item.id}">
                    <button type="submit">Delete</button>
                </form>
            </td>
        </tr>
    `).join('');
}

async function handleRequest(req, res) {
    const cookies = parseCookies(req);

    if (req.method === 'GET' && req.url === '/') {
        try {
            const html = await fs.promises.readFile(path.join(__dirname,
'index.html'), 'utf8');
            let processedHtml = '';

            if (!cookies.user || !cookies.userId) {
```

```
                processedHtml = html.replace('{{rows}}', `
                    <tr><td colspan="3" style="text-align: center; color:
grey;">Please log in to view your to-do list.</td></tr>
                `);
            } else {
                processedHtml = html.replace('{{rows}}', await
getHtmlRows(cookies.userId));
            }

            res.writeHead(200, { 'Content-Type': 'text/html' });
            res.end(processedHtml);
        } catch (err) {
            console.error(err);
            res.writeHead(500, { 'Content-Type': 'text/plain' });
            res.end('Error loading index.html');
        }

    } else if (req.method === 'POST' && req.url === '/add') {
        let body = '';
        req.on('data', chunk => body += chunk);
        req.on('end', async () => {
            const parsed = parse(body);
            const text = parsed.text?.trim();
            if (text && cookies.userId) {
                try {
                    await addItemToDatabase(text, cookies.userId,
cookies.user);
                    res.writeHead(302, { Location: '/' });
                    res.end();
                } catch (err) {
                    res.writeHead(500, { 'Content-Type': 'text/plain' });
                    res.end('Error adding item');
                }
            } else {
                res.writeHead(400, { 'Content-Type': 'text/plain' });
                res.end('Invalid item text or not logged in');
            }
        });

    } else if (req.method === 'POST' && req.url === '/delete') {
        let body = '';
        req.on('data', chunk => body += chunk);
        req.on('end', async () => {
            const parsed = new URLSearchParams(body);
            const id = parseInt(parsed.get('id'), 10);
            if (!isNaN(id)) {
                try {
                    await deleteItemFromDatabase(id);
                    res.writeHead(302, { Location: '/' });
                    res.end();
                } catch (err) {
                    res.writeHead(500, { 'Content-Type': 'text/plain' });
```

```
                    res.end('Error deleting item');
                }
            } else {
                res.writeHead(400, { 'Content-Type': 'text/plain' });
                res.end('Invalid ID');
            }
        });

    } else if (req.method === 'POST' && req.url === '/edit') {
        let body = '';
        req.on('data', chunk => body += chunk);
        req.on('end', async () => {
            const parsed = new URLSearchParams(body);
            const id = parseInt(parsed.get('id'), 10);
            const text = parsed.get('text')?.trim();
            if (!isNaN(id) && text) {
                try {
                    await updateItemInDatabase(id, text);
                    res.writeHead(302, { Location: '/' });
                    res.end();
                } catch (err) {
                    res.writeHead(500, { 'Content-Type': 'text/plain' });
                    res.end('Error updating item');
                }
            } else {
                res.writeHead(400, { 'Content-Type': 'text/plain' });
                res.end('Invalid data');
            }
        });

    } else if (req.method === 'POST' && req.url === '/register') {
        let body = '';
        req.on('data', chunk => body += chunk);
        req.on('end', async () => {
            const data = new URLSearchParams(body);
            const username = data.get('username');
            const password = hashPassword(data.get('password'));
            const conn = await mysql.createConnection(dbConfig);
            try {
                await conn.execute('INSERT INTO users (username, password)
VALUES (?, ?)', [username, password]);
                const [rows] = await conn.execute('SELECT id FROM users WHERE
username = ?', [username]);
                const userId = rows[0].id;
                res.writeHead(302, {
                    'Set-Cookie': [
                        `user=${username}; HttpOnly`,
                        `userId=${userId}; HttpOnly`
                    ],
                    'Location': '/'
                });
                res.end();
```

```
            } catch (err) {
                res.writeHead(500, { 'Content-Type': 'text/plain' });
                res.end('User already exists or error');
            } finally {
                await conn.end();
            }
        });

    } else if (req.method === 'POST' && req.url === '/login') {
        let body = '';
        req.on('data', chunk => body += chunk);
        req.on('end', async () => {
            const data = new URLSearchParams(body);
            const username = data.get('username');
            const password = hashPassword(data.get('password'));
            const conn = await mysql.createConnection(dbConfig);
            const [rows] = await conn.execute('SELECT id FROM users WHERE
username = ? AND password = ?', [username, password]);
            await conn.end();
            if (rows.length > 0) {
                const userId = rows[0].id;
                res.writeHead(302, {
                    'Set-Cookie': [
                        `user=${username}; HttpOnly`,
                        `userId=${userId}; HttpOnly`
                    ],
                    'Location': '/'
                });
                res.end();
            } else {
                res.writeHead(401, { 'Content-Type': 'text/plain' });
                res.end('Invalid credentials');
            }
        });

    } else if (req.method === 'GET' && req.url.startsWith('/logout')) {
        res.writeHead(302, {
            'Set-Cookie': [
                'user=; Max-Age=0',
                'userId=; Max-Age=0'
            ],
            'Location': '/'
        });
        res.end();

    } else {
        res.writeHead(404, { 'Content-Type': 'text/plain' });
        res.end('Route not found');
    }
}

const server = http.createServer(handleRequest);
```

```
server.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```