МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский государственный университет аэрокосмического приборостроения»

кАФЕДРА 33

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | К.А.Жиданов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ОТЧЕТ  
О ЛАБОРАТОРНОЙ РАБОТЕ

|  |
| --- |
| Лабораторная работа |
| по дисциплине: Технологии и методы программирования |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент гр. № | 3331 |  |  |  | Г.Г.Галабурда |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

Цель лабораторной работы

Целью проекта было создание веб-приложения "ToDo List" с функционалом управления задачами, авторизацией пользователей и интеграцией с Telegram-ботом для удаленного управления задачами через мессенджер. Приложение должно позволять пользователям регистрироваться, входить в систему, добавлять, редактировать, удалять задачи, а также синхронизировать их через Telegram-бота, который взаимодействует с базой данных.

Пример работы программы

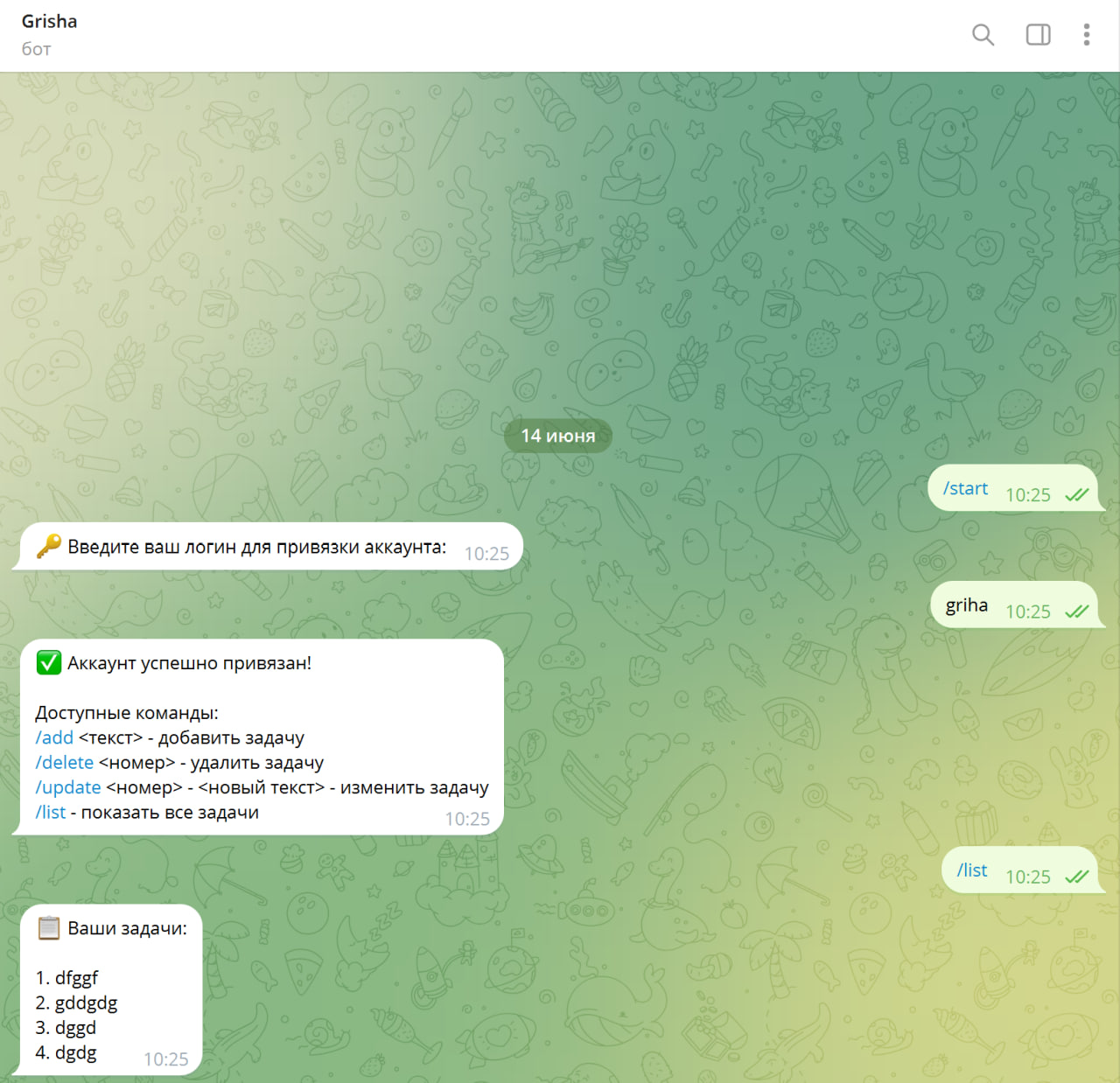


Рисунок 1 – Интеграция телеграмма.

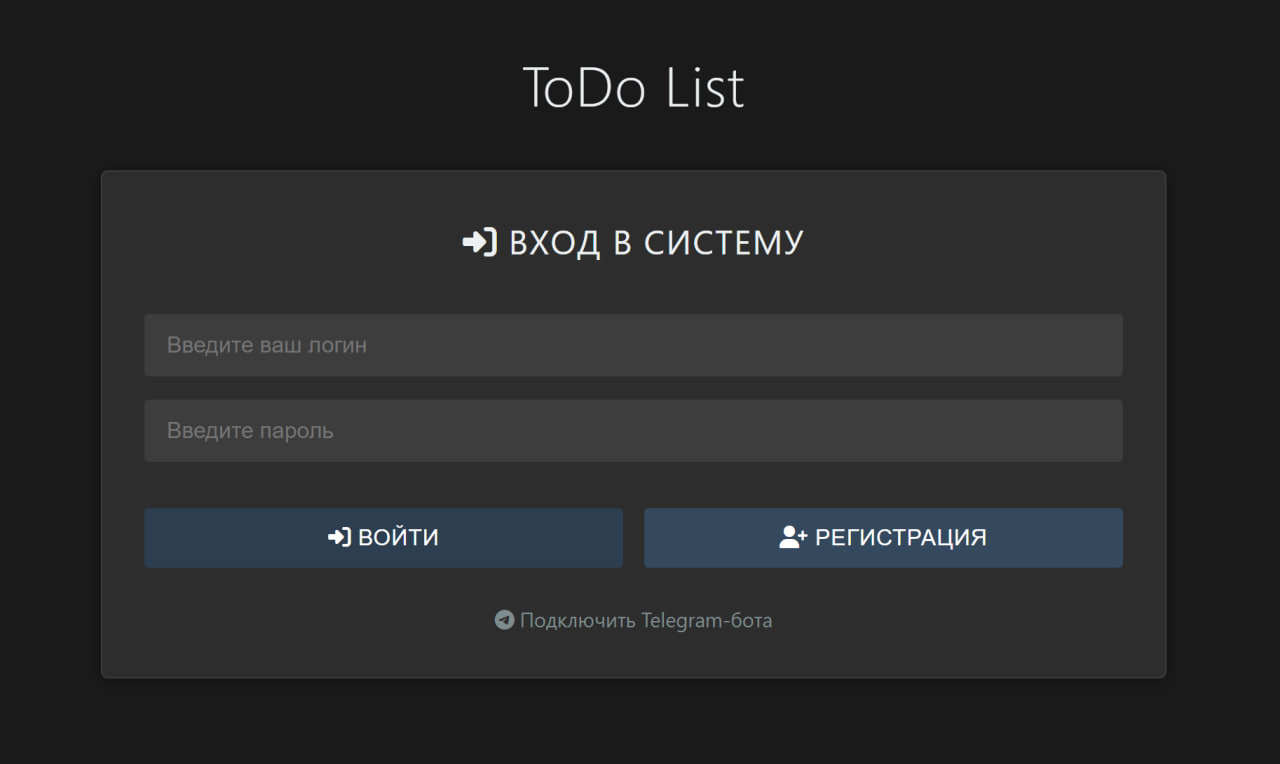


Рисунок 2 – Авторизация.

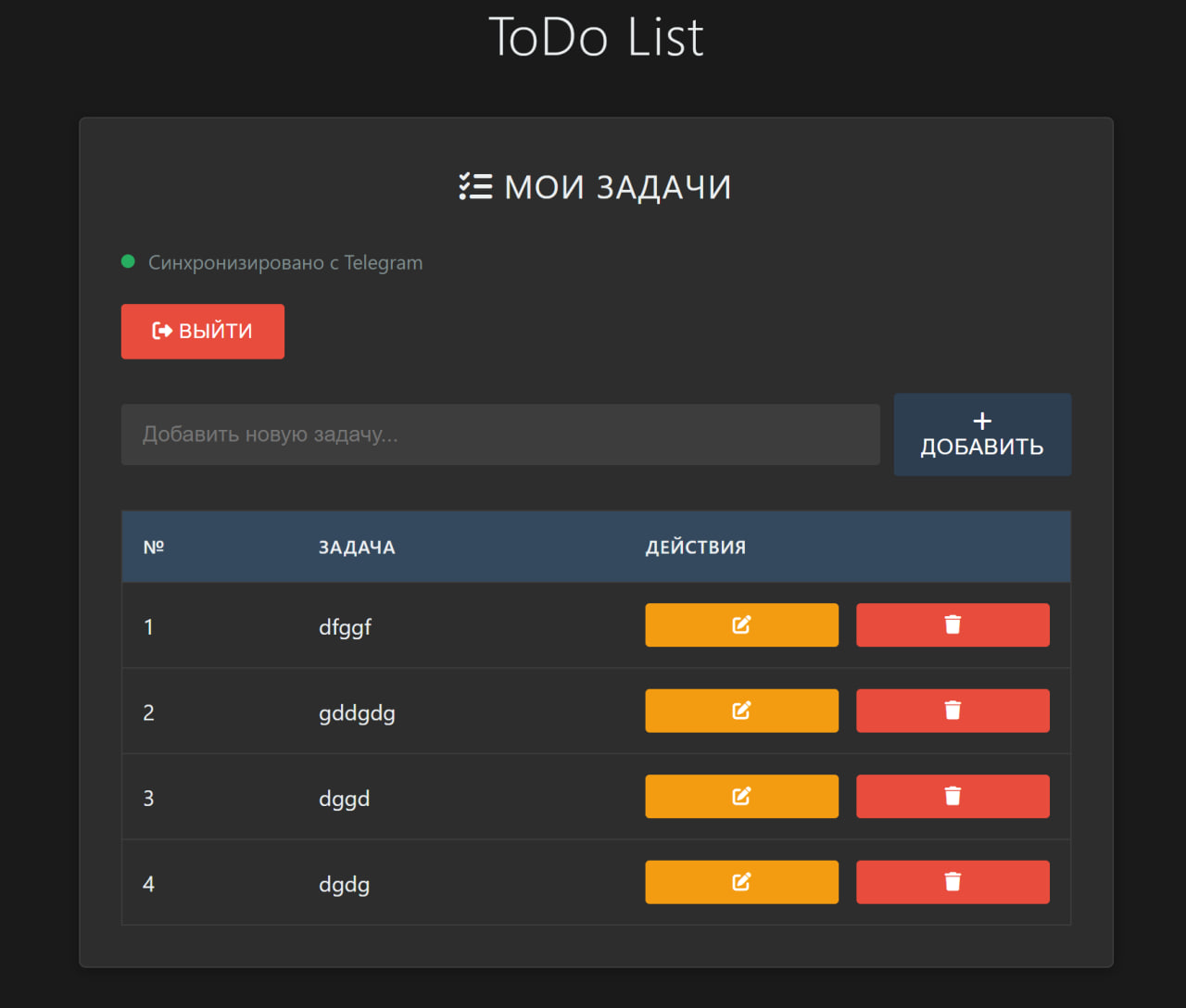


Рисунок 3 – Интерфейс.

ПРОБЛЕМЫ и решения

**1. Проблема с базой данных**

**Описание**: Первоначально база данных создавалась вручную через SQL-скрипт (db.sql). Это вызывало проблемы при развертывании приложения, так как требовалось отдельно выполнять скрипт, что увеличивало вероятность ошибок (например, несоответствие структуры базы данных).

**Решение**:

* Реализована функция resetDatabase в index.js, которая автоматически:
  + - Удаляет существующую базу данных (DROP DATABASE IF EXISTS).
    - Создает новую базу данных и таблицы (CREATE DATABASE, CREATE TABLE).
    - Устанавливает связи через внешние ключи с каскадным удалением (ON DELETE CASCADE).
* Добавлена инициализация тестового пользователя для упрощения тестирования.

**Результат**: Полная автоматизация создания и инициализации базы данных, что упростило развертывание и устранило необходимость ручного вмешательства.

**2. Проблемы с дизайном интерфейса**

**Описание**:

* Элементы интерфейса накладывались друг на друга, особенно при адаптации под разные размеры экранов.
* Отсутствие стилизации делало интерфейс непривлекательным и сложным для восприятия.
* Кнопки редактирования и удаления в таблице задач отображались некорректно на узких экранах.

**Решение**:

* В index.html добавлены ссылки на Font Awesome для иконок кнопок редактирования и удаления.
* Предполагается использование CSS (хотя в предоставленном коде стили отсутствуют). Рекомендуется добавить адаптивные стили с использованием Flexbox или Grid для корректного отображения.
* **Планируемое улучшение**: Создать файл styles.css с медиа-запросами для адаптивности и стилями для таблиц, кнопок и форм.

**Результат**: Частичное улучшение за счет добавления иконок, но требуется дальнейшая работа над CSS для устранения наложений и улучшения UX.

**3. Недопонимание запросов GPT**

**Описание**: При разработке возникали сложности с тем, что GPT неверно интерпретировал запросы, особенно в части интеграции Telegram и структуры базы данных. Это приводило к необходимости многократного уточнения требований.

**Решение**:

* Формулировка запросов была уточнена, с акцентом на конкретные технические детали (например, указание использования node-telegram-bot-api, структуры таблиц, REST API).
* Использовалась итеративная разработка: код проверялся и корректировался после каждого ответа GPT.

**Результат**: Улучшение качества кода за счет более точных запросов, но процесс разработки замедлился из-за необходимости уточнений.

**4. Ошибки на хосте**

**Описание**: При развертывании на хосте возникали ошибки, связанные с наложением элементов интерфейса и некорректной работой сессий из-за неправильной конфигурации express-session.

**Решение**:

* Для интерфейса: временно минимизирована сложность HTML-структуры, чтобы избежать наложений (например, упрощена структура форм и таблиц).
* Для сессий: проверена конфигурация express-session (установлены параметры resave: false, saveUninitialized: false, secure: false для локальной разработки).
* Добавлена обработка ошибок для всех маршрутов и Telegram-команд, чтобы минимизировать сбои.

**Результат**: Устойчивость приложения повысилась, но интерфейс требует доработки для полной адаптивности.

описание работы

**Архитектура проекта**

Проект состоит из трех основных компонентов:

1. **База данных (MySQL)**: Хранит информацию о пользователях и их задачах.

* Таблица users: хранит идентификатор, имя пользователя, хэшированный пароль и Telegram ID.
* Таблица items: хранит задачи, связанные с пользователями через внешний ключ user\_id.

1. **Веб-приложение (Node.js + Express)**: Обеспечивает REST API и рендеринг интерфейса.

* Реализованы маршруты для авторизации, регистрации, управления задачами и интеграции с Telegram.
* Используется express-session для управления сессиями пользователей.

1. **Telegram-бот (node-telegram-bot-api)**: Позволяет пользователям взаимодействовать с задачами через команды в Telegram.

**Авторизация**

Авторизация реализована следующим образом:

1. **Регистрация**:
   * Пользователь отправляет POST-запрос на /register с логином и паролем.
   * Пароль хэшируется с использованием bcrypt и сохраняется в таблице users.
   * Если имя пользователя уже существует, возвращается ошибка ER\_DUP\_ENTRY.
2. **Вход**:
   * Пользователь отправляет POST-запрос на /login с логином и паролем.
   * Пароль сравнивается с хэшем в базе данных с помощью bcrypt.compare.
   * При успешной авторизации создается сессия (req.session.user), содержащая id и username.
   * Пользователь перенаправляется на главную страницу с задачами.
3. **Проверка авторизации**:
   * Middleware isAuthenticated проверяет наличие req.session.user перед доступом к защищенным маршрутам (например, /, /add, /delete, /update).
   * Если сессия отсутствует, пользователь перенаправляется на /login.
4. **Выход**:
   * GET-запрос на /logout уничтожает сессию (req.session.destroy) и перенаправляет на страницу входа.

**Интеграция с Telegram**

Telegram-бот реализован с использованием библиотеки node-telegram-bot-api и взаимодействует с базой данных:

1. **Привязка аккаунта**:
   * Пользователь отправляет команду /start в Telegram.
   * Бот запрашивает имя пользователя, после чего выполняется привязка Telegram ID к учетной записи в базе данных (через UPDATE users SET telegram\_id = ?).
   * Если Telegram ID уже привязан или пользователь не найден, возвращается ошибка.
2. **Команды бота**:
   * /add <текст>: Добавляет задачу в таблицу items для пользователя, связанного с Telegram ID.
   * /delete <номер>: Удаляет задачу по ID, если она принадлежит пользователю.
   * /update <номер> - <новый текст>: Обновляет текст задачи по ID.
   * /list: Возвращает список всех задач пользователя в формате текста.
3. **Взаимодействие с базой данных**:
   * Бот использует функцию getUserByTelegramId для получения пользователя по Telegram ID.
   * Все операции (добавление, удаление, обновление, получение списка) выполняются через соответствующие SQL-запросы, аналогичные веб-интерфейсу, с проверкой принадлежности задач пользователю.

**Функционал управления задачами**

* **Добавление задачи**: POST-запрос на /add с текстом задачи, которая сохраняется в таблице items с привязкой к user\_id.
* **Удаление задачи**: DELETE-запрос на /delete?id=<id> удаляет задачу, если она принадлежит пользователю.
* **Обновление задачи**: PUT-запрос на /update?id=<id> обновляет текст задачи.
* **Отображение задач**: GET-запрос на / рендерит HTML-страницу с таблицей задач, сгенерированной функцией getHtmlRows.

Вывод лабораторной работы

Проект успешно работает с авторизацией и интеграцией Telegram-бота. Пользователи могут регистрироваться, входить в систему, управлять задачами через веб-интерфейс и Telegram. Основные проблемы (база данных, дизайн, ошибки на хосте, недопонимание GPT) были частично или полностью решены:

Приложение 1. Код программы index.JS

const express = require('express');

const mysql = require('mysql2/promise');

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

const bcrypt = require('bcrypt');

const path = require('path');

const fs = require('fs').promises;

const TelegramBot = require('node-telegram-bot-api');

const app = express();

const PORT = 3000;

const TELEGRAM\_TOKEN = '8037175742:AAFh5gCkKKWJrbpk77L-9VlDHlu5KaOmm7Y';

const bot = new TelegramBot(TELEGRAM\_TOKEN, { polling: true });

const dbConfig = {

    host: 'localhost',

    user: 'root',

    password: '19122005П',

    database: 'todolist',

};

app.use(express.json());

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

app.use(session({

    secret: 'your-secret-key',

    resave: false,

    saveUninitialized: false,

    cookie: { secure: false }

}));

// Функция полной очистки БД

async function resetDatabase() {

    let connection;

    try {

        connection = await mysql.createConnection({

            host: dbConfig.host,

            user: dbConfig.user,

            password: dbConfig.password

        });

        await connection.query('DROP DATABASE IF EXISTS todolist');

        await connection.query('CREATE DATABASE todolist');

        await connection.query('USE todolist');

        await connection.query(`

            CREATE TABLE users (

                id INT AUTO\_INCREMENT PRIMARY KEY,

                username VARCHAR(50) NOT NULL UNIQUE,

                password VARCHAR(255) NOT NULL,

                telegram\_id VARCHAR(50) UNIQUE,

                created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

            )

        `);

        await connection.query(`

            CREATE TABLE items (

                id INT AUTO\_INCREMENT PRIMARY KEY,

                text VARCHAR(255) NOT NULL,

                user\_id INT,

                created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

                FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE

            )

        `);

        console.log('✅ База данных успешно очищена и пересоздана');

    } catch (error) {

        console.error('❌ Ошибка очистки БД:', error);

        throw error;

    } finally {

        if (connection) await connection.end();

    }

}

// Инициализация БД

async function initDatabase() {

    await resetDatabase();

    // Добавляем тестового пользователя (опционально)

    let connection;

    try {

        connection = await mysql.createConnection(dbConfig);

        const hashedPassword = await bcrypt.hash('test123', 10);

        await connection.execute(

            'INSERT INTO users (username, password) VALUES (?, ?)',

            ['testuser', hashedPassword]

        );

        console.log('✅ Тестовый пользователь создан (логин: testuser, пароль: test123)');

    } catch (error) {

        console.error('❌ Ошибка создания тестового пользователя:', error);

    } finally {

        if (connection) await connection.end();

    }

}

// Получение списка задач

async function retrieveListItems(userId) {

    let connection;

    try {

        connection = await mysql.createConnection(dbConfig);

        const [rows] = await connection.execute(

            'SELECT id, text FROM items WHERE user\_id = ? ORDER BY id ASC',

            [userId]

        );

        return rows;

    } catch (error) {

        console.error('❌ Ошибка получения задач:', error);

        throw error;

    } finally {

        if (connection) await connection.end();

    }

}

// Добавление задачи

async function addListItem(text, userId) {

    let connection;

    try {

        connection = await mysql.createConnection(dbConfig);

        const [result] = await connection.execute(

            'INSERT INTO items (text, user\_id) VALUES (?, ?)',

            [text, userId]

        );

        return { id: result.insertId, text };

    } catch (error) {

        console.error('❌ Ошибка добавления задачи:', error);

        throw error;

    } finally {

        if (connection) await connection.end();

    }

}

// Удаление задачи

async function deleteListItem(id, userId) {

    let connection;

    try {

        connection = await mysql.createConnection(dbConfig);

        const [result] = await connection.execute(

            'DELETE FROM items WHERE id = ? AND user\_id = ?',

            [id, userId]

        );

        return result.affectedRows > 0;

    } catch (error) {

        console.error('❌ Ошибка удаления задачи:', error);

        throw error;

    } finally {

        if (connection) await connection.end();

    }

}

// Обновление задачи

async function updateListItem(id, newText, userId) {

    let connection;

    try {

        connection = await mysql.createConnection(dbConfig);

        const [result] = await connection.execute(

            'UPDATE items SET text = ? WHERE id = ? AND user\_id = ?',

            [newText, id, userId]

        );

        return result.affectedRows > 0;

    } catch (error) {

        console.error('❌ Ошибка обновления задачи:', error);

        throw error;

    } finally {

        if (connection) await connection.end();

    }

}

// Получение пользователя по Telegram ID

async function getUserByTelegramId(telegramId) {

    let connection;

    try {

        connection = await mysql.createConnection(dbConfig);

        const [rows] = await connection.execute(

            'SELECT \* FROM users WHERE telegram\_id = ?',

            [telegramId]

        );

        return rows[0];

    } catch (error) {

        console.error('❌ Ошибка получения пользователя:', error);

        throw error;

    } finally {

        if (connection) await connection.end();

    }

}

// Привязка Telegram ID

async function linkTelegramId(username, telegramId) {

    let connection;

    try {

        connection = await mysql.createConnection(dbConfig);

        const [userRows] = await connection.execute(

            'SELECT \* FROM users WHERE username = ?',

            [username]

        );

        if (userRows.length === 0) {

            return { success: false, message: `❌ Пользователь "${username}" не найден` };

        }

        const [telegramRows] = await connection.execute(

            'SELECT \* FROM users WHERE telegram\_id = ?',

            [telegramId]

        );

        if (telegramRows.length > 0) {

            return { success: false, message: '❌ Этот Telegram ID уже привязан' };

        }

        await connection.execute(

            'UPDATE users SET telegram\_id = ? WHERE username = ?',

            [telegramId, username]

        );

        return {

            success: true,

            message: '✅ Аккаунт успешно привязан!\n\n' +

                     'Доступные команды:\n' +

                     '/add <текст> - добавить задачу\n' +

                     '/delete <номер> - удалить задачу\n' +

                     '/update <номер> - <новый текст> - изменить задачу\n' +

                     '/list - показать все задачи'

        };

    } catch (error) {

        console.error('❌ Ошибка привязки Telegram:', error);

        return { success: false, message: '❌ Ошибка сервера' };

    } finally {

        if (connection) await connection.end();

    }

}

// Генерация HTML для задач

async function getHtmlRows(userId) {

    const todoItems = await retrieveListItems(userId);

    return todoItems.map((item, index) => `

        <tr>

            <td>${index + 1}</td>

            <td class="text-cell" data-id="${item.id}">${item.text}</td>

            <td>

                <div class="action-buttons">

                    <button class="edit-btn" data-id="${item.id}"><i class="fas fa-edit"></i></button>

                    <button class="delete-btn" data-id="${item.id}"><i class="fas fa-trash"></i></button>

                </div>

            </td>

        </tr>

    `).join('');

}

// Проверка авторизации

function isAuthenticated(req, res, next) {

    if (req.session.user) {

        next();

    } else {

        res.redirect('/login');

    }

}

// Маршруты

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

    try {

        const html = await fs.readFile(path.join(\_\_dirname, 'index.html'), 'utf8');

        res.send(html.replace('{{rows}}', ''));

    } catch (error) {

        res.status(500).send('Ошибка загрузки страницы');

    }

});

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

    const { username, password } = req.body;

    if (!username || !password) {

        return res.status(400).json({ error: 'Логин и пароль обязательны' });

    }

    try {

        const connection = await mysql.createConnection(dbConfig);

        const [rows] = await connection.execute(

            'SELECT \* FROM users WHERE username = ?',

            [username]

        );

        await connection.end();

        if (rows.length === 0) {

            return res.status(401).json({ error: 'Пользователь не найден' });

        }

        const user = rows[0];

        const match = await bcrypt.compare(password, user.password);

        if (match) {

            req.session.user = { id: user.id, username: user.username };

            return res.json({ message: 'Вход выполнен успешно' });

        } else {

            return res.status(401).json({ error: 'Неверный пароль' });

        }

    } catch (error) {

        console.error(error);

        return res.status(500).json({ error: 'Ошибка сервера' });

    }

});

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

    const { username, password } = req.body;

    if (!username || !password) {

        return res.status(400).json({ error: 'Логин и пароль обязательны' });

    }

    try {

        const connection = await mysql.createConnection(dbConfig);

        const hashedPassword = await bcrypt.hash(password, 10);

        await connection.execute(

            'INSERT INTO users (username, password) VALUES (?, ?)',

            [username, hashedPassword]

        );

        await connection.end();

        return res.json({ message: 'Регистрация успешна' });

    } catch (error) {

        console.error(error);

        if (error.code === 'ER\_DUP\_ENTRY') {

            return res.status(400).json({ error: 'Пользователь уже существует' });

        }

        return res.status(500).json({ error: 'Ошибка сервера' });

    }

});

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

    req.session.destroy();

    res.redirect('/login');

});

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

    try {

        const html = await fs.readFile(path.join(\_\_dirname, 'index.html'), 'utf8');

        const processedHtml = html.replace('{{rows}}', await getHtmlRows(req.session.user.id));

        res.send(processedHtml);

    } catch (error) {

        console.error(error);

        res.status(500).send('Ошибка загрузки страницы');

    }

});

app.post('/add', isAuthenticated, async (req, res) => {

    const { text } = req.body;

    if (!text || typeof text !== 'string' || text.trim() === '') {

        return res.status(400).json({ error: 'Неверный текст задачи' });

    }

    try {

        const newItem = await addListItem(text.trim(), req.session.user.id);

        res.json(newItem);

    } catch (error) {

        console.error(error);

        res.status(500).json({ error: 'Ошибка добавления задачи' });

    }

});

app.delete('/delete', isAuthenticated, async (req, res) => {

    const id = req.query.id;

    if (!id || isNaN(id)) {

        return res.status(400).json({ error: 'Неверный ID задачи' });

    }

    try {

        const success = await deleteListItem(id, req.session.user.id);

        if (success) {

            res.json({ message: 'Задача удалена' });

        } else {

            res.status(404).json({ error: 'Задача не найдена' });

        }

    } catch (error) {

        console.error(error);

        res.status(500).json({ error: 'Ошибка удаления задачи' });

    }

});

app.put('/update', isAuthenticated, async (req, res) => {

    const id = req.query.id;

    const { text } = req.body;

    if (!id || isNaN(id) || !text || typeof text !== 'string' || text.trim() === '') {

        return res.status(400).json({ error: 'Неверные данные для обновления' });

    }

    try {

        const success = await updateListItem(id, text.trim(), req.session.user.id);

        if (success) {

            res.json({ message: 'Задача обновлена', text: text.trim() });

        } else {

            res.status(404).json({ error: 'Задача не найдена' });

        }

    } catch (error) {

        console.error(error);

        res.status(500).json({ error: 'Ошибка обновления задачи' });

    }

});

// Обработчики Telegram бота

bot.onText(/\/start/, async (msg) => {

    const chatId = msg.chat.id;

    try {

        await bot.sendMessage(chatId, '🔑 Введите ваш логин для привязки аккаунта:');

        bot.once('message', async (responseMsg) => {

            if (responseMsg.text.startsWith('/')) return;

            const username = responseMsg.text.trim();

            const result = await linkTelegramId(username, chatId.toString());

            await bot.sendMessage(chatId, result.message, { parse\_mode: 'Markdown' });

        });

    } catch (error) {

        console.error('❌ Ошибка в /start:', error);

        bot.sendMessage(chatId, '❌ Произошла ошибка, попробуйте позже');

    }

});

bot.onText(/\/add (.+)/, async (msg, match) => {

    const chatId = msg.chat.id;

    try {

        const user = await getUserByTelegramId(chatId.toString());

        if (!user) {

            return bot.sendMessage(chatId, '❌ Ваш Telegram ID не привязан. Используйте /start');

        }

        const text = match[1].trim();

        await addListItem(text, user.id);

        bot.sendMessage(chatId, `✅ Задача добавлена: "${text}"`);

    } catch (error) {

        console.error('❌ Ошибка в /add:', error);

        bot.sendMessage(chatId, '❌ Ошибка добавления задачи');

    }

});

bot.onText(/\/delete (\d+)/, async (msg, match) => {

    const chatId = msg.chat.id;

    try {

        const user = await getUserByTelegramId(chatId.toString());

        if (!user) {

            return bot.sendMessage(chatId, '❌ Ваш Telegram ID не привязан. Используйте /start');

        }

        const id = match[1];

        const success = await deleteListItem(id, user.id);

        if (success) {

            bot.sendMessage(chatId, `✅ Задача #${id} удалена`);

        } else {

            bot.sendMessage(chatId, `❌ Задача #${id} не найдена`);

        }

    } catch (error) {

        console.error('❌ Ошибка в /delete:', error);

        bot.sendMessage(chatId, '❌ Ошибка удаления задачи');

    }

});

bot.onText(/\/update (\d+) - (.+)/, async (msg, match) => {

    const chatId = msg.chat.id;

    try {

        const user = await getUserByTelegramId(chatId.toString());

        if (!user) {

            return bot.sendMessage(chatId, '❌ Ваш Telegram ID не привязан. Используйте /start');

        }

        const id = match[1];

        const newText = match[2].trim();

        const success = await updateListItem(id, newText, user.id);

        if (success) {

            bot.sendMessage(chatId, `✅ Задача #${id} обновлена: "${newText}"`);

        } else {

            bot.sendMessage(chatId, `❌ Задача #${id} не найдена`);

        }

    } catch (error) {

        console.error('❌ Ошибка в /update:', error);

        bot.sendMessage(chatId, '❌ Ошибка обновления задачи');

    }

});

bot.onText(/\/list/, async (msg) => {

    const chatId = msg.chat.id;

    try {

        const user = await getUserByTelegramId(chatId.toString());

        if (!user) {

            return bot.sendMessage(chatId, '❌ Ваш Telegram ID не привязан. Используйте /start');

        }

        const items = await retrieveListItems(user.id);

        if (items.length === 0) {

            return bot.sendMessage(chatId, '📭 Список задач пуст');

        }

        const message = items.map((item, index) =>

            `${index + 1}. ${item.text}`

        ).join('\n');

        bot.sendMessage(chatId, `📋 Ваши задачи:\n\n${message}`);

    } catch (error) {

        console.error('❌ Ошибка в /list:', error);

        bot.sendMessage(chatId, '❌ Ошибка получения списка задач');

    }

});

// Обработка ошибок бота

bot.on('polling\_error', (error) => {

    console.error('❌ Telegram Bot Polling Error:', error);

});

// Запуск сервера

initDatabase()

    .then(() => {

        app.listen(PORT, () => {

            console.log(`🚀 Сервер запущен на порту ${PORT}`);

            console.log('🤖 Telegram Bot активен');

        });

    })

    .catch(error => {

        console.error('❌ Фатальная ошибка запуска:', error);

        process.exit(1);

    });

Приложение 2. Код программы index.html

<!DOCTYPE html>

<html lang="ru">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Список задач</title>

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css">

    <style>

        :root {

            --primary-color: #2c3e50;

            --secondary-color: #34495e;

            --accent-color: #7f8c8d;

            --light-color: #ecf0f1;

            --dark-color: #2c3e50;

            --error-color: #e74c3c;

            --bg-color: #1a1a1a;

            --card-bg: #2d2d2d;

            --border-color: #3d3d3d;

        }

        \* {

            box-sizing: border-box;

            margin: 0;

            padding: 0;

        }

        body {

            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

            max-width: 800px;

            margin: 0 auto;

            padding: 20px;

            color: var(--light-color);

            min-height: 100vh;

            background-color: var(--bg-color);

            line-height: 1.6;

        }

        .logo {

            text-align: center;

            margin: 30px 0;

            color: var(--light-color);

            font-size: 2.5rem;

            font-weight: 300;

            letter-spacing: 1px;

        }

        .auth-container, .todo-container {

            margin: 20px auto;

            background-color: var(--card-bg);

            padding: 30px;

            border-radius: 5px;

            border: 1px solid var(--border-color);

            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.3);

        }

        h2 {

            color: var(--light-color);

            margin-bottom: 25px;

            text-align: center;

            font-size: 1.5rem;

            font-weight: 400;

            text-transform: uppercase;

            letter-spacing: 1px;

        }

        .input-group {

            margin-bottom: 25px;

        }

        input {

            padding: 12px 15px;

            margin: 8px 0;

            border: 1px solid var(--border-color);

            border-radius: 3px;

            width: 100%;

            font-size: 1rem;

            background-color: #3d3d3d;

            color: var(--light-color);

            transition: border 0.3s;

        }

        input:focus {

            outline: none;

            border-color: var(--accent-color);

        }

        .button-group {

            display: flex;

            justify-content: space-between;

            gap: 15px;

            margin-top: 25px;

        }

        button {

            padding: 12px 20px;

            background-color: var(--secondary-color);

            color: white;

            border: none;

            border-radius: 3px;

            cursor: pointer;

            font-size: 1rem;

            font-weight: 500;

            transition: all 0.3s;

            flex: 1;

            text-transform: uppercase;

            letter-spacing: 0.5px;

        }

        button:hover {

            background-color: var(--primary-color);

        }

        .btn-primary {

            background-color: var(--primary-color);

        }

        .btn-primary:hover {

            background-color: #1a252f;

        }

        table {

            width: 100%;

            border-collapse: collapse;

            margin-top: 25px;

            background-color: var(--card-bg);

            border: 1px solid var(--border-color);

        }

        th, td {

            padding: 15px;

            text-align: left;

            border-bottom: 1px solid var(--border-color);

        }

        th {

            background-color: var(--secondary-color);

            font-weight: 500;

            color: var(--light-color);

            text-transform: uppercase;

            font-size: 0.85rem;

            letter-spacing: 0.5px;

        }

        tr:hover {

            background-color: rgba(255, 255, 255, 0.03);

        }

        .text-cell {

            cursor: pointer;

            transition: color 0.3s;

        }

        .text-cell:hover {

            color: var(--accent-color);

        }

        .error {

            color: var(--error-color);

            display: none;

            margin: 15px 0;

            font-weight: 500;

            text-align: center;

            padding: 10px;

            background-color: rgba(231, 76, 60, 0.1);

            border-radius: 3px;

            border: 1px solid rgba(231, 76, 60, 0.3);

        }

        .logout-btn {

            background-color: var(--error-color);

            width: 120px;

            margin-bottom: 25px;

            font-size: 0.9rem;

        }

        .logout-btn:hover {

            background-color: #c0392b;

        }

        .edit-btn {

            background-color: #f39c12;

            margin-right: 8px;

            padding: 8px 12px;

            font-size: 0.85rem;

        }

        .edit-btn:hover {

            background-color: #d35400;

        }

        .delete-btn {

            background-color: var(--error-color);

            padding: 8px 12px;

            font-size: 0.85rem;

        }

        .delete-btn:hover {

            background-color: #c0392b;

        }

        .task-input-group {

            display: flex;

            gap: 10px;

            margin-bottom: 20px;

        }

        #taskInput {

            flex-grow: 1;

        }

        .action-buttons {

            display: flex;

            gap: 5px;

        }

        .status-indicator {

            display: flex;

            align-items: center;

            gap: 10px;

            margin-bottom: 20px;

            color: var(--accent-color);

            font-size: 0.9rem;

        }

        .status-dot {

            width: 10px;

            height: 10px;

            border-radius: 50%;

            background-color: #27ae60;

        }

        .telegram-link {

            display: block;

            text-align: center;

            margin-top: 25px;

            color: var(--accent-color);

            text-decoration: none;

            font-size: 0.9rem;

        }

        .telegram-link:hover {

            text-decoration: underline;

        }

        @media (max-width: 768px) {

            body {

                padding: 15px;

            }

            .auth-container, .todo-container {

                padding: 20px;

            }

            .button-group {

                flex-direction: column;

            }

            .task-input-group {

                flex-direction: column;

            }

            button {

                width: 100%;

            }

            .logo {

                font-size: 2rem;

                margin: 20px 0;

            }

        }

    </style>

</head>

<body>

    <div class="logo">ToDo List</div>

    <div class="auth-container" id="authContainer">

        <h2><i class="fas fa-sign-in-alt"></i> Вход в систему</h2>

        <div class="input-group">

            <input type="text" id="username" placeholder="Введите ваш логин">

            <input type="password" id="password" placeholder="Введите пароль">

        </div>

        <div class="button-group">

            <button class="btn-primary" onclick="login()"><i class="fas fa-sign-in-alt"></i> Войти</button>

            <button onclick="register()"><i class="fas fa-user-plus"></i> Регистрация</button>

        </div>

        <p id="authError" class="error"></p>

        <a href="#" class="telegram-link">

            <i class="fab fa-telegram"></i> Подключить Telegram-бота

        </a>

    </div>

    <div class="todo-container" id="todoContainer" style="display: none;">

        <h2><i class="fas fa-tasks"></i> Мои задачи</h2>

        <div class="status-indicator">

            <div class="status-dot"></div>

            <div class="status-text">Синхронизировано с Telegram</div>

        </div>

        <button class="logout-btn" onclick="logout()"><i class="fas fa-sign-out-alt"></i> Выйти</button>

        <div class="task-input-group">

            <input type="text" id="taskInput" placeholder="Добавить новую задачу...">

            <button class="btn-primary" onclick="addTask()"><i class="fas fa-plus"></i> Добавить</button>

        </div>

        <p id="taskError" class="error"></p>

        <table id="taskTable">

            <thead>

                <tr>

                    <th>№</th>

                    <th>Задача</th>

                    <th>Действия</th>

                </tr>

            </thead>

            <tbody>

                {{rows}}

            </tbody>

        </table>

    </div>

    <script>

        async function showError(elementId, message) {

            const errorElement = document.getElementById(elementId);

            errorElement.textContent = message;

            errorElement.style.display = 'block';

            setTimeout(() => { errorElement.style.display = 'none'; }, 3000);

        }

        async function login() {

            const username = document.getElementById('username').value.trim();

            const password = document.getElementById('password').value.trim();

            if (username.length === 0 || password.length === 0) {

                showError('authError', 'Логин и пароль не могут быть пустыми');

                return;

            }

            try {

                const response = await fetch('/login', {

                    method: 'POST',

                    headers: { 'Content-Type': 'application/json' },

                    body: JSON.stringify({ username, password })

                });

                const data = await response.json();

                if (response.ok) {

                    document.getElementById('authContainer').style.display = 'none';

                    document.getElementById('todoContainer').style.display = 'block';

                    loadTasks();

                } else {

                    showError('authError', data.error || 'Ошибка входа');

                }

            } catch (err) {

                showError('authError', 'Ошибка сервера');

            }

        }

        async function register() {

            const username = document.getElementById('username').value.trim();

            const password = document.getElementById('password').value.trim();

            if (username.length === 0 || password.length === 0) {

                showError('authError', 'Логин и пароль не могут быть пустыми');

                return;

            }

            try {

                const response = await fetch('/register', {

                    method: 'POST',

                    headers: { 'Content-Type': 'application/json' },

                    body: JSON.stringify({ username, password })

                });

                const data = await response.json();

                if (response.ok) {

                    showError('authError', 'Регистрация успешна, войдите');

                } else {

                    showError('authError', data.error || 'Ошибка регистрации');

                }

            } catch (err) {

                showError('authError', 'Ошибка сервера');

            }

        }

        async function logout() {

            try {

                await fetch('/logout');

                document.getElementById('todoContainer').style.display = 'none';

                document.getElementById('authContainer').style.display = 'block';

                document.getElementById('username').value = '';

                document.getElementById('password').value = '';

            } catch (err) {

                showError('taskError', 'Ошибка выхода');

            }

        }

        function updateRowNumbers() {

            const rows = document.querySelectorAll('#taskTable tbody tr');

            rows.forEach((row, index) => {

                row.cells[0].textContent = index + 1;

            });

        }

        async function loadTasks() {

            try {

                const response = await fetch('/');

                const html = await response.text();

                const parser = new DOMParser();

                const doc = parser.parseFromString(html, 'text/html');

                const table = doc.querySelector('#taskTable');

                document.querySelector('#taskTable tbody').innerHTML = table.querySelector('tbody').innerHTML;

                attachEventListeners();

                updateRowNumbers();

            } catch (err) {

                showError('taskError', 'Ошибка загрузки задач');

            }

        }

        async function addTask() {

            const text = document.getElementById('taskInput').value.trim();

            if (!text) {

                showError('taskError', 'Введите текст задачи');

                return;

            }

            try {

                const response = await fetch('/add', {

                    method: 'POST',

                    headers: { 'Content-Type': 'application/json' },

                    body: JSON.stringify({ text })

                });

                const data = await response.json();

                if (response.ok) {

                    document.getElementById('taskInput').value = '';

                    loadTasks();

                } else {

                    showError('taskError', data.error || 'Ошибка добавления');

                }

            } catch (err) {

                showError('taskError', 'Ошибка сервера');

            }

        }

        async function deleteTask(id) {

            try {

                const response = await fetch(`/delete?id=${id}`, {

                    method: 'DELETE'

                });

                const data = await response.json();

                if (response.ok) {

                    loadTasks();

                } else {

                    showError('taskError', data.error || 'Ошибка удаления');

                }

            } catch (err) {

                showError('taskError', 'Ошибка сервера');

            }

        }

        async function editTask(id, textCell) {

            const newText = prompt('Введите новый текст задачи:', textCell.textContent);

            if (newText === null || newText.trim() === '') return;

            try {

                const response = await fetch(`/update?id=${id}`, {

                    method: 'PUT',

                    headers: { 'Content-Type': 'application/json' },

                    body: JSON.stringify({ text: newText.trim() })

                });

                const data = await response.json();

                if (response.ok) {

                    loadTasks();

                } else {

                    showError('taskError', data.error || 'Ошибка редактирования');

                }

            } catch (err) {

                showError('taskError', 'Ошибка сервера');

            }

        }

        function attachEventListeners() {

            document.querySelectorAll('.delete-btn').forEach(button => {

                button.onclick = () => deleteTask(button.dataset.id);

            });

            document.querySelectorAll('.edit-btn').forEach(button => {

                button.onclick = () => {

                    const textCell = document.querySelector(`td[data-id="${button.dataset.id}"]`);

                    editTask(button.dataset.id, textCell);

                };

            });

        }

        document.addEventListener('DOMContentLoaded', () => {

            attachEventListeners();

        });

    </script>

</body>

</html>

Приложение 3. Код программы db.sql

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,

    telegram\_id VARCHAR(50) UNIQUE

);

CREATE TABLE IF NOT EXISTS items (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    text VARCHAR(255) NOT NULL,

    user\_id INT,

    FOREIGN KEY (user\_id) REFERENCES users(id)

);