

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

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

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

по курсу: ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ

СТУДЕНТ ГР. №	3333	_____	А.Е. Чернолоз
	номер группы	подпись, дата	инициалы, фамилия

Санкт-Петербург
2025

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Создать полнофункциональное веб-приложение для управления списком задач с возможностью взаимодействия через Telegram-бота. Реализовать авторизацию, хранение данных в базе и обработку задач с клиентской и серверной сторон.

Задачи, решаемые в ходе работы:

1. Разработка пользовательского интерфейса (веб-страницы) для регистрации, входа в систему и отображения задач в виде списка.
2. Реализация серверной части на Node.js с обработкой запросов от клиента.
3. Настройка и работа с базой данных MySQL для хранения информации о пользователях и их задачах..
4. Реализация Telegram-бота, способного выполнять команды по добавлению, удалению, редактированию и отображению задач.
5. Обеспечение взаимодействия всех компонентов системы, включая обработку ошибок, тестирование работы Telegram-бота, отладку SQL-запросов и тестовое наполнение базы данных..

Пример работы программы:

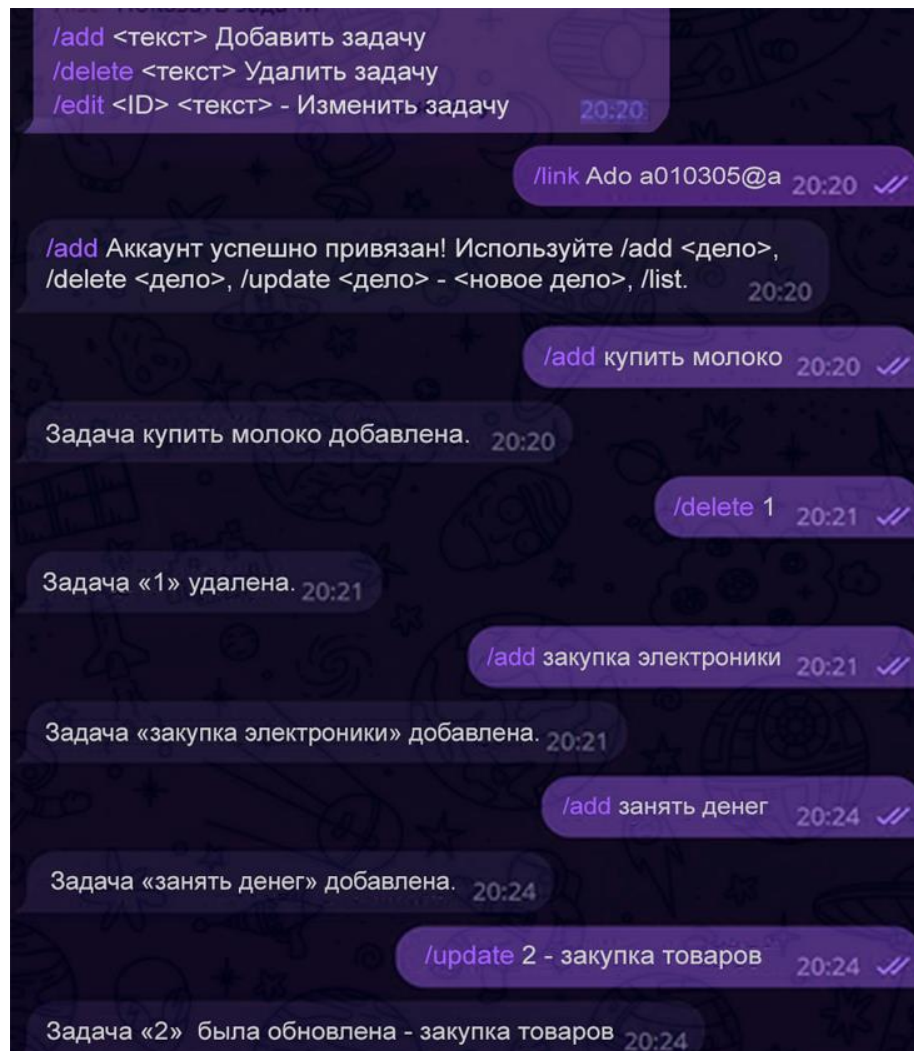
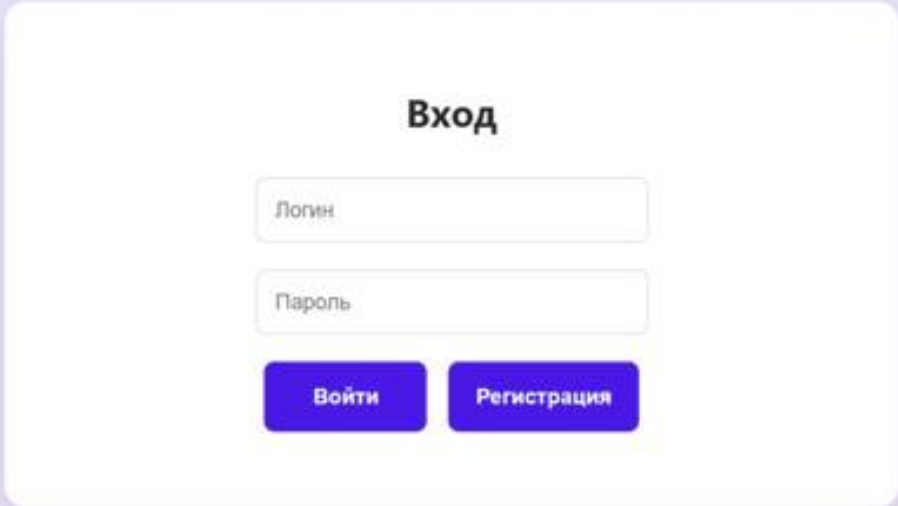


Рисунок 1 – интеграция телеграмма



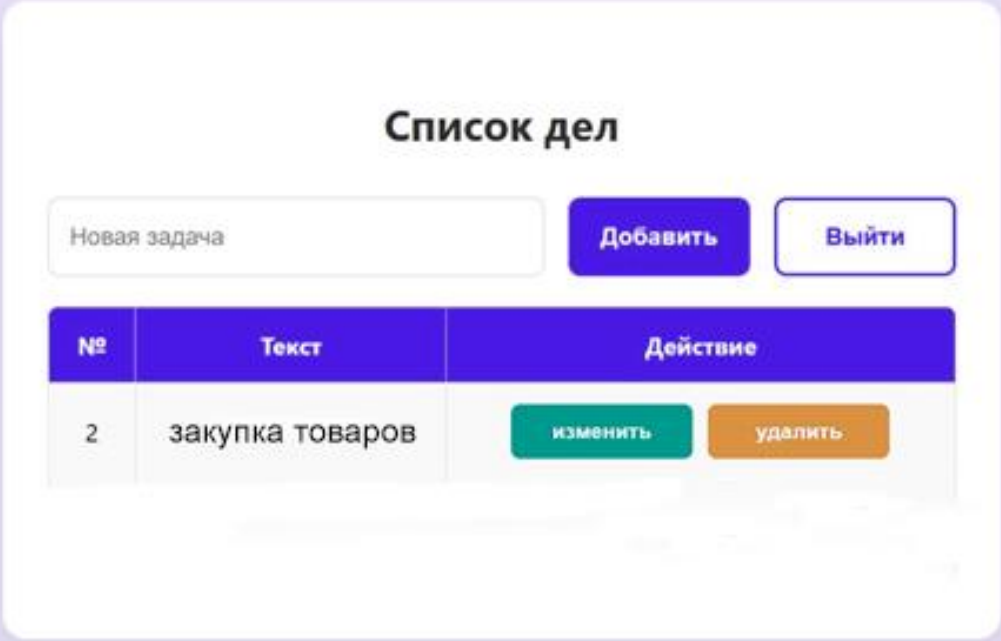
Вход

Логин

Пароль

Войти Регистрация

Рисунок 2 – авторизация/регистрация



Список дел

Новая задача

Добавить Выйти

№	Текст	Действие
2	закупка товаров	изменить удалить

Рисунок 3 – интерфейс

Работа процессов:

1. Авторизация и привязка Telegram-аккаунта

1.1 Привязка через /start

1. Пользователь запускает бота с командой /start.
2. Бот запрашивает логин, который должен быть заранее зарегистрирован в таблице users (вручную или через другой интерфейс).
3. При совпадении логина Telegram ID (msg.chat.id) записывается в поле telegram_id соответствующего пользователя.
4. После этого все действия в чате будут привязаны к конкретному пользователю.

SQL-запрос:

```
UPDATE users SET telegram_id = ? WHERE username = ?;
```

JS-код:

```
bot.onText(/\/start/, async (msg) => {
  const chatId = msg.chat.id;
  bot.sendMessage(chatId, 'Введите ваш логин:');

  bot.once('message', async (response) => {
    const username = response.text.trim();
    try {
      const [rows] = await pool.query(
        'UPDATE users SET telegram_id = ? WHERE username = ?',
        [chatId.toString(), username]
      );
      if (rows.affectedRows > 0) {
        bot.sendMessage(chatId, `Логин "${username}" успешно привязан.`);
      } else {
        bot.sendMessage(chatId, 'Логин не найден. Обратитесь к администратору.');      }
    } catch (err) {
```

```
        console.error(err);
        bot.sendMessage(chatId, 'Произошла ошибка при привязке.');
```

}

```
});
```

});

2. Работа с задачами (таблица items)

Во всех командах используется проверка Telegram ID — бот находит user.id из таблицы users по telegram_id.

2.1 Добавление задачи

Команда:

/add Купить молоко

SQL-запрос:

```
INSERT INTO items (text, user_id) VALUES (?, ?);
```

2.2 Удаление задачи

Команда:

/delete 5

Удаляется задача с id = 5, только если она принадлежит текущему пользователю.

SQL-запрос:

```
DELETE FROM items WHERE id = ? AND user_id = ?;
```

2.3 Редактирование задачи

Команда:

`/edit 5 Купить хлеб`

SQL-запрос:

`UPDATE items SET text = ? WHERE id = ? AND user_id = ?;`

2.4 Просмотр задач

Команда:

`/list`

Бот показывает список всех задач пользователя.

SQL-запрос:

`SELECT id, text FROM items WHERE user_id = ?;`

Пример ответа:

Ваши задачи:

1. Купить молоко
2. Сделать дз по математике

3. Проверка привязки Telegram

Перед выполнением любой команды (кроме `/start`) бот выполняет проверку:

1. Ищет пользователя в таблице `users` по `telegram_id`.
2. Если не найден — сообщает, что нужно авторизоваться через `/start`.

Пример кода:

```
async function getUserByTelegramId(telegramId) {  
  const [rows] = await pool.query(  
    'SELECT telegram_id, id FROM users WHERE telegram_id = ?'
```

```

        'SELECT * FROM users WHERE telegram_id = ?',
        [telegramId]
    );
    return rows[0];
}

```

4. Структура базы данных

Таблица users

Поле	Тип	Описание
id	INT	Уникальный идентификатор пользователя
username	VARCHAR(50)	Уникальное имя пользователя
password_hash	VARCHAR(255)	Захешированный пароль (если нужен)
telegram_id	VARCHAR(50)	ID Telegram-пользователя (чат ID)

Таблица items

Поле	Тип	Описание
id	INT	Уникальный ID задачи
text	VARCHAR(255)	Текст задачи
user_id	INT	ID пользователя из users
created_at	TIMESTAMP	Дата создания задачи

Вывод:

Разработанное приложение соответствует поставленной цели: реализована система авторизации пользователей и успешно выполнена интеграция с Telegram-ботом для управления задачами. Все ключевые функции — добавление, удаление, редактирование и просмотр задач — работают корректно и надёжно. В процессе разработки возникали трудности, связанные с реализацией логики Telegram-бота и взаимодействием с базой данных. Также пришлось уточнять формулировки запросов к ИИ, так как не всегда удавалось получить нужный результат с первого раза. Однако благодаря поэтапному подходу и экспериментам над кодом удалось преодолеть возникающие сложности и довести проект до рабочего состояния. Финальный результат можно рассматривать как устойчивую основу для возможного дальнейшего расширения функционала.

Приложение 1 – Код программы – bot.js

```
const TelegramBot = require('node-telegram-bot-api');
const mysql = require('mysql2/promise');
const bcrypt = require('bcrypt');

// Конфигурация базы данных
const dbConfig = {
  host: 'localhost',
  user: 'todo_user',
  password: '1234',
  database: 'todolist',
  waitForConnections: true,
  connectionLimit: 10,
  queueLimit: 0
};

// Создаем пул соединений вместо отдельных подключений
const pool = mysql.createPool(dbConfig);

// Токен вашего Telegram бота
const TOKEN = '8137640522:AAEH-Ro6cZv3T5N0qFXA0BX2yM5m1RLXgwI';
const bot = new TelegramBot(TOKEN, {
  polling: true,
  request: {
    timeout: 10000
  }
});

// Функция для получения пользователя по Telegram ID
async function getUserByTelegramId(telegramId) {
  let connection;
  try {
    connection = await pool.getConnection();
    const [rows] = await connection.query(
      'SELECT id, username FROM users WHERE telegram_id = ?',
      [telegramId]
    );
  } catch (error) {
    console.error('Error in getUserByTelegramId:', error);
  }
}
```

```

    );
    return rows[0] || null;
} catch (error) {
    console.error('Database error in getUserByTelegramId:', error);
    throw error;
} finally {
    if (connection) connection.release();
}
}

// Основной обработчик команд
bot.on('message', async (msg) => {
    const chatId = msg.chat.id;
    const telegramId = msg.from.id;
    const text = msg.text;

    // Игнорируем сообщения без текста или не начинающиеся с /
    if (!text || !text.startsWith('/')) {
        return;
    }

    try {
        // Разбиваем команду на части
        const parts = text.split(' ').filter(p => p.trim());
        const command = parts[0].toLowerCase();
        const args = parts.slice(1);

        // Обработка разных команд
        switch (command) {
            case '/start':
                await handleStart(chatId);
                break;

            case '/link':
                if (args.length < 2) {
                    await bot.sendMessage(chatId, 'Неверный формат. Используйте шаблон: /link <логин> <пароль>');
                }
            }
        }
    }
});

```

```

        return;
    }
    await handleLink(chatId, telegramId, args[0], args[1]);
    break;

case '/list':
    await handleList(chatId, telegramId);
    break;

case '/add':
    if (args.length < 1) {
        await bot.sendMessage(chatId, 'Неверный формат.
Используйте шаблон: /add <текст задачи>');
        return;
    }
    await handleAdd(chatId, telegramId, args.join(' '));
    break;

case '/delete':
    if (args.length < 1 || isNaN(args[0])) {
        await bot.sendMessage(chatId, 'Неверный формат.
Используйте шаблон: /delete <ID задачи>');
        return;
    }
    await handleDelete(chatId, telegramId, parseInt(args[0]));
    break;

case '/edit':
    if (args.length < 2 || isNaN(args[0])) {
        await bot.sendMessage(chatId, 'Неверный формат.
Используйте шаблон: /edit <ID> <новый текст>');
        return;
    }
    await handleEdit(chatId, telegramId, parseInt(args[0]),
args.slice(1).join(' '));
    break;

```

```

        default:
            await bot.sendMessage(chatId, 'Неизвестная команда.
Доступные команды:\n\n' +
                '/start - Начало работы\n' +
                '/link - Привязать аккаунт\n' +
                '/list - Показать задачи\n' +
                '/add - Добавить задачу\n' +
                '/delete - Удалить задачу\n' +
                '/edit - Изменить задачу');
        }
    } catch (error) {
        console.error('Error handling message:', error);
        await bot.sendMessage(chatId, '⚠️ Произошла ошибка при обработке
команды. Пожалуйста, попробуйте позже.');
```

```

    });

// Обработчики конкретных команд
async function handleStart(chatId) {
    await bot.sendMessage(
        chatId,
        `👋 To-Do List Bot\n\n` +
        `Этот бот помогает управлять вашими задачами.\n\n` +
        `Сначала привяжите свой аккаунт:\n` +
        `/link <логин> <пароль>\n\n` +
        `После привязки доступны команды:\n` +
        `/list - Показать задачи\n` +
        `/add <текст> - Добавить задачу\n` +
        `/delete <ID> - Удалить задачу\n` +
        `/edit <ID> <текст> - Изменить задачу`
    );
}

```

```

async function handleLink(chatId, telegramId, username, password) {
    let connection;
    try {
        connection = await pool.getConnection();
    }
}

```

```

// Проверяем пользователя
const [users] = await connection.query(
    'SELECT id, password_hash FROM users WHERE username = ?',
    [username]
);

if (users.length === 0) {
    await bot.sendMessage(chatId, 'Пользователь не найден. Проверьте логин.');
```

Проверьте логин.');

```

    return;
}

const user = users[0];
const isValid = await bcrypt.compare(password,
user.password_hash);

if (!isValid) {
    await bot.sendMessage(chatId, 'Неверный пароль. Попробуйте снова.');
```

снова.');

```

    return;
}

// Привязываем Telegram ID
await connection.query(
    'UPDATE users SET telegram_id = ? WHERE id = ?',
    [telegramId, user.id]
);

await bot.sendMessage(
    chatId,
    'Аккаунт успешно привязан!\n\n' +
    'Теперь вы можете управлять задачами:\n' +
    '/list - Показать все задачи\n' +
    '/add [текст] - Добавить задачу\n' +
    '/delete [ID] - Удалить задачу\n' +
    '/edit [ID] [текст] - Изменить задачу'
```

```

    );
  } catch (error) {
    console.error('Error in link command:', error);
    if (error.code === 'ER_DUP_ENTRY') {
      await bot.sendMessage(chatId, 'Этот Telegram аккаунт уже
привязан.');
```

```

    } else {
      throw error;
    }
  } finally {
    if (connection) connection.release();
  }
}

```

```

async function handleList(chatId, telegramId) {
  const user = await getUserByTelegramId(telegramId);
  if (!user) {
    await bot.sendMessage(chatId, 'Сначала привяжите аккаунт командой
/link');
```

```

    return;
  }
}

```

```

let connection;
try {
  connection = await pool.getConnection();
  const [tasks] = await connection.query(
    'SELECT id, text FROM items WHERE user_id = ? ORDER BY
created_at DESC',
    [user.id]
  );

  if (tasks.length === 0) {
    await bot.sendMessage(chatId, 'Список задач пуст. Добавьте
первую задачу командой /add');
```

```

    return;
  }
}

```

```

        const tasksList = tasks.map(task => `#${task.id}:
        ${task.text}`).join('\n\n');
        await bot.sendMessage(
            chatId,
            `Ваши задачи (${tasks.length}): \n\n${tasksList}`
        );
    } catch (error) {
        console.error('Error in list command:', error);
        throw error;
    } finally {
        if (connection) connection.release();
    }
}

```

```

async function handleAdd(chatId, telegramId, text) {
    const user = await getUserByTelegramId(telegramId);
    if (!user) {
        await bot.sendMessage(chatId, 'Сначала привяжите аккаунт командой
/link');
        return;
    }

```

```

    if (!text || text.trim().length === 0) {
        await bot.sendMessage(chatId, 'Текст задачи не может быть пустым');
        return;
    }

```

```

    let connection;
    try {
        connection = await pool.getConnection();
        await connection.query(
            'INSERT INTO items (text, user_id) VALUES (?, ?)',
            [text.trim(), user.id]
        );
    }

    await bot.sendMessage(
        chatId,

```

```

        `Задача успешно добавлена:\n"${text.trim()}"`
    );
} catch (error) {
    console.error('Error in add command:', error);
    throw error;
} finally {
    if (connection) connection.release();
}
}

async function handleDelete(chatId, telegramId, taskId) {
    const user = await getUserByTelegramId(telegramId);
    if (!user) {
        await bot.sendMessage(chatId, 'Сначала привяжите аккаунт командой /link');
        return;
    }

    let connection;
    try {
        connection = await pool.getConnection();
        const [result] = await connection.query(
            'DELETE FROM items WHERE id = ? AND user_id = ?',
            [taskId, user.id]
        );

        if (result.affectedRows > 0) {
            await bot.sendMessage(chatId, `Задача #${taskId} удалена`);
        } else {
            await bot.sendMessage(chatId, `Задача #${taskId} не найдена`);
        }
    } catch (error) {
        console.error('Error in delete command:', error);
        throw error;
    } finally {
        if (connection) connection.release();
    }
}

```



```
}
```

```
async function handleEdit(chatId, telegramId, taskId, newText) {
  const user = await getUserByTelegramId(telegramId);
  if (!user) {
    await bot.sendMessage(chatId, 'Сначала привяжите аккаунт командой /link');
    return;
  }

  if (!newText || newText.trim().length === 0) {
    await bot.sendMessage(chatId, 'Текст задачи не может быть пустым');
    return;
  }

  let connection;
  try {
    connection = await pool.getConnection();
    const [rows] = await connection.execute(
      'SELECT * FROM users WHERE telegram_id = ?',
      [telegramId]
    );
    const user = rows[0];

    if (!user) {
      await bot.sendMessage(chatId, 'Сначала зарегистрируйтесь с помощью /start.');
```

помощью /start.');

```
      return;
    }

    const [result] = await connection.execute(
      'UPDATE tasks SET description = ? WHERE id = ? AND user_id = ?',
      [newText.trim(), taskId, user.id]
    );
  }
```

```

        if (result.affectedRows > 0) {
            await bot.sendMessage(chatId, `Задача #${taskId} успешно
обновлена:\n"${newText.trim()}"`);
        } else {
            await bot.sendMessage(chatId, `Задача #${taskId} не
найдена или вы не являетесь её автором.`);
        }
    } catch (error) {
        console.error('Error in edit command:', error);
    } finally {
        if (connection) connection.release();
    }
}

};

// Обработка ошибок бота
bot.on('polling_error', (error) => {
    console.error('Polling error:', error);
});

process.on('unhandledRejection', (error) => {
    console.error('Unhandled rejection:', error);
});

console.log('Бот запущен - вводите команды');

```

Приложение 2 – Код index.js

```

const http = require('http');
const fs = require('fs');
const path = require('path');
const mysql = require('mysql2/promise');
const url = require('url');
const querystring = require('querystring');
const bcrypt = require('bcrypt');
const cookie = require('cookie');
const crypto = require('crypto');

const PORT = 3000;

```

```

const dbConfig = {
  host: 'localhost',
  user: 'todo_user',
  password: '1234',
  database: 'todolist',
};

// Сессионное хранилище
const sessions = {};

// Middleware для проверки аутентификации
function checkAuth(req) {
  const cookies = cookie.parse(req.headers.cookie || '');
  const sessionId = cookies.sessionId;

  if (!sessionId || !sessions[sessionId]) {
    return null;
  }

  return sessions[sessionId].userId;
}

// Функции для работы с пользователями
async function createUser(username, password) {
  const hashedPassword = await bcrypt.hash(password, 10);
  const connection = await mysql.createConnection(dbConfig);
  const query = 'INSERT INTO users (username, password_hash) VALUES (?, ?)';
  const [result] = await connection.execute(query, [username, hashedPassword]);
  await connection.end();
  return result.insertId;
}

async function verifyUser(username, password) {
  const connection = await mysql.createConnection(dbConfig);
  const query = 'SELECT id, password_hash FROM users WHERE username = ?';
  const [rows] = await connection.execute(query, [username]);
  await connection.end();

  if (rows.length === 0) {
    return null;
  }
}

```

```

    const user = rows[0];
    const isValid = await bcrypt.compare(password, user.password_hash);

    return isValid ? user.id : null;
}

// Функции для работы с задачами
async function retrieveListItems(userId) {
    try {
        const connection = await mysql.createConnection(dbConfig);
        const query = 'SELECT id, text FROM items WHERE user_id = ? ORDER
BY id';
        const [rows] = await connection.execute(query, [userId]);
        await connection.end();
        return rows;
    } catch (error) {
        console.error('Error retrieving list items:', error);
        throw error;
    }
}

async function addItemToDB(text, userId) {
    try {
        const connection = await mysql.createConnection(dbConfig);
        const query = 'INSERT INTO items (text, user_id) VALUES (?, ?)';
        const [result] = await connection.execute(query, [text, userId]);
        await connection.end();
        return result.insertId;
    } catch (error) {
        console.error('Error adding item:', error);
        throw error;
    }
}

async function deleteItemFromDB(id, userId) {
    try {
        const connection = await mysql.createConnection(dbConfig);
        const query = 'DELETE FROM items WHERE id = ? AND user_id = ?';
        const [result] = await connection.execute(query, [id, userId]);
        await connection.end();
        return result.affectedRows > 0;
    } catch (error) {
        console.error('Error deleting item:', error);
        throw error;
    }
}

```

```

}

async function updateItemInDB(id, newText, userId) {
  try {
    const connection = await mysql.createConnection(dbConfig);
    const query = 'UPDATE items SET text = ? WHERE id = ? AND user_id
= ?';
    const [result] = await connection.execute(query, [newText, id,
userId]);
    await connection.end();
    return result.affectedRows > 0;
  } catch (error) {
    console.error('Error updating item:', error);
    throw error;
  }
}

```

```

async function getHtmlRows(userId) {
  const todoItems = await retrieveListItems(userId);
  return todoItems.map(item => `
<tr data-id="${item.id}">
<td>${item.id}</td>
<td class="item-text">${item.text}</td>
<td>
<button class="edit-btn">Edit</button>
<button class="delete-btn" data-id="${item.id}">x</button>
</td>
</tr>
`).join('');
}

```

```

async function serveLoginPage(res, error = null) {
  try {
    let html = await fs.promises.readFile(path.join(__dirname,
'login.html'), 'utf8');
    if (error) {
      html = html.replace('<!-- ERROR_PLACEHOLDER -->',
'<div class="error">Invalid username or
password</div>');
    }
    res.writeHead(200, { 'Content-Type': 'text/html' });
    return res.end(html);
  } catch (err) {
    console.error(err);
    res.writeHead(500, { 'Content-Type': 'text/plain' });
  }
}

```

```

        return res.end('Error loading login page');
    }
}

async function serveRegisterPage(res, error = null) {
    try {
        let html = await fs.promises.readFile(path.join(__dirname,
            'register.html'), 'utf8');
        if (error) {
            html = html.replace('<!-- ERROR_PLACEHOLDER -->',
                '<div class="error">Registration failed.
Username may be taken.</div>');
        }
        res.writeHead(200, { 'Content-Type': 'text/html' });
        return res.end(html);
    } catch (err) {
        console.error(err);
        res.writeHead(500, { 'Content-Type': 'text/plain' });
        return res.end('Error loading register page');
    }
}

async function handleRequest(req, res) {
    const parsedUrl = url.parse(req.url, true);

    // Обработка статических файлов
    if (req.method === 'GET' && parsedUrl.pathname === '/styles.css') {
        try {
            const css = await fs.promises.readFile(path.join(__dirname,
                'styles.css'), 'utf8');
            res.writeHead(200, { 'Content-Type': 'text/css' });
            return res.end(css);
        } catch (err) {
            res.writeHead(404);
            return res.end();
        }
    }

    // Обработка маршрутов аутентификации
    if (req.method === 'GET' && parsedUrl.pathname === '/login') {
        return serveLoginPage(res, parsedUrl.query.error);
    }

    if (req.method === 'POST' && parsedUrl.pathname === '/login') {
        let body = '';

```

```

    req.on('data', chunk => body += chunk.toString());
    req.on('end', async () => {
        const { username, password } = querystring.parse(body);

        try {
            const userId = await verifyUser(username, password);
            if (userId) {
                const sessionId =
crypto.randomBytes(16).toString('hex');
                sessions[sessionId] = { userId };

                res.writeHead(302, {
                    'Location': '/',
                    'Set-Cookie': cookie.serialize('sessionId',
sessionId, {
                        httpOnly: true,
                        maxAge: 60 * 60 * 24 * 7 // 1 week
                    })
                });
                return res.end();
            } else {
                res.writeHead(302, { 'Location': '/login?error=1' });
                return res.end();
            }
        } catch (error) {
            console.error(error);
            res.writeHead(500, { 'Content-Type': 'text/plain' });
            return res.end('Login error');
        }
    });
    return;
}

if (req.method === 'GET' && parsedUrl.pathname === '/register') {
    return serveRegisterPage(res, parsedUrl.query.error);
}

if (req.method === 'POST' && parsedUrl.pathname === '/register') {
    let body = '';
    req.on('data', chunk => body += chunk.toString());
    req.on('end', async () => {
        const { username, password } = querystring.parse(body);

        try {
            await createUser(username, password);

```

```

        res.writeHead(302, { 'Location': '/login' });
        return res.end();
    } catch (error) {
        console.error(error);
        res.writeHead(302, { 'Location': '/register?error=1' });
        return res.end();
    }
});
return;
}

if (req.method === 'GET' && parsedUrl.pathname === '/logout') {
    const cookies = cookie.parse(req.headers.cookie || '');
    const sessionId = cookies.sessionId;

    if (sessionId && sessions[sessionId]) {
        delete sessions[sessionId];
    }

    res.writeHead(302, {
        'Location': '/login',
        'Set-Cookie': cookie.serialize('sessionId', '', {
            httpOnly: true,
            expires: new Date(0)
        })
    });
    return res.end();
}

// Проверка аутентификации для защищенных маршрутов
const userId = await checkAuth(req);

if (!userId && parsedUrl.pathname !== '/login' && parsedUrl.pathname
!== '/register') {
    res.writeHead(302, { 'Location': '/login' });
    return res.end();
}

// Новый endpoint для проверки привязки Telegram
if (req.method === 'GET' && parsedUrl.pathname === '/check-telegram')
{
    try {
        const connection = await mysql.createConnection(dbConfig);
        const query = 'SELECT telegram_id FROM users WHERE id = ?';
        const [rows] = await connection.execute(query, [userId]);
    }
}

```



```

        await connection.end();

        res.writeHead(200, { 'Content-Type': 'application/json' });
        return res.end(JSON.stringify({ hasTelegram:
!!rows[0]?.telegram_id }));
    } catch (error) {
        console.error(error);
        res.writeHead(500, { 'Content-Type': 'application/json' });
        return res.end(JSON.stringify({ error: 'Database error' }));
    }
}

// Обработка защищенных маршрутов
if (req.method === 'GET' && parsedUrl.pathname === '/') {
    try {
        const html = await fs.promises.readFile(path.join(__dirname,
'index.html'), 'utf8');
        const processedHtml = html.replace('{{rows}}', await
getHtmlRows(userId));

        res.writeHead(200, { 'Content-Type': 'text/html' });
        return res.end(processedHtml);
    } catch (err) {
        console.error(err);
        res.writeHead(500, { 'Content-Type': 'text/plain' });
        return res.end('Error loading index.html');
    }
}
else if (req.method === 'POST' && parsedUrl.pathname === '/add') {
    let body = '';
    req.on('data', chunk => body += chunk.toString());
    req.on('end', async () => {
        const text = new URLSearchParams(body).get('text');
        if (text && text.trim()) {
            try {
                await addItemToDB(text.trim(), userId);
                res.writeHead(302, { 'Location': '/' });
                return res.end();
            } catch (error) {
                console.error(error);
                res.writeHead(500, { 'Content-Type': 'text/plain' });
                return res.end('Error adding item');
            }
        }
    }
} else {
    res.writeHead(400, { 'Content-Type': 'text/plain' });

```

```

        return res.end('Invalid input');
    }
});
return;
}
else if (req.method === 'POST' && parsedUrl.pathname === '/delete') {
    let body = '';
    req.on('data', chunk => body += chunk.toString());
    req.on('end', async () => {
        const { id } = querystring.parse(body);
        if (id) {
            try {
                const success = await deleteItemFromDB(id, userId);
                if (success) {
                    res.writeHead(302, { 'Location': '/' });
                    return res.end();
                } else {
                    res.writeHead(404, { 'Content-Type': 'text/plain'
});
                        return res.end('Item not found');
                    }
                } catch (error) {
                    console.error(error);
                    res.writeHead(500, { 'Content-Type': 'text/plain' });
                    return res.end('Error deleting item');
                }
            } else {
                res.writeHead(400, { 'Content-Type': 'text/plain' });
                return res.end('Invalid ID');
            }
        }
    });
    return;
}
else if (req.method === 'POST' && parsedUrl.pathname === '/update') {
    let body = '';
    req.on('data', chunk => body += chunk.toString());
    req.on('end', async () => {
        const { id, text } = querystring.parse(body);
        if (id && text && text.trim()) {
            try {
                const success = await updateItemInDB(id, text.trim(),
userId);
                res.writeHead(200, { 'Content-Type':
'application/json' });
                return res.end(JSON.stringify({ success }));
            }

```

```

        } catch (error) {
            console.error(error);
            res.writeHead(500, { 'Content-Type':
'application/json' });
            return res.end(JSON.stringify({ success: false,
error: 'Error updating item' }));
        }
    } else {
        res.writeHead(400, { 'Content-Type': 'application/json'
});
        return res.end(JSON.stringify({ success: false, error:
'Invalid input' }));
    }
    });
    return;
}
else {
    res.writeHead(404, { 'Content-Type': 'text/plain' });
    return res.end('Not Found');
}
}

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

```

Приложение 3 – 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 дел</title>
    <style>
body {
margin: 0;
padding: 0;
    font-family: 'Segoe UI', Tahoma, Geneva,
Verdana, sans-serif;    background: #dbeafe;
display: flex;    align-items: center;
justify-content: center;
    min-height: 100vh;
}

    .card {
background:
white;

```

```
padding: 40px
30px;
border-radius:
16px;
    box-shadow: 0 12px 30px
rgba(0,0,0,0.1);    width: 100%;
    max-width: 600px;
}
```

```
h2 {
    text-align: center;
margin-bottom: 30px;
font-size: 28px;
    color: #333;
}
```

```
#authContainer form
{
    display: flex;
flex-direction: column;
    align-items: center;
}
```

```
#authContainer input {
max-width: 260px;
    width: 100%;
padding: 14px;
margin-bottom:
20px;
    border: 1px solid #ccc;
    border-
radius: 8px;
font-size: 16px;
}
```

```
.button-group {
display: flex;
    justify-content: center;
gap: 16px;
}
```

```
button {
padding: 0 20px;
min-width: 120px;
height: 52px;
display: flex;
align-items:
center;
justify-content:
```

```
center;
background:
#007bff;
color: white;
border: none;
border-radius: 8px;
font-weight: bold;
font-size: 16px;
cursor: pointer;
transition: 0.3s
ease;
}
```

```
    button:hover {
background: #0056b3;
}
```

```
    .logout-btn {
background: white;
border: 2px solid
#007bff;          color:
#007bff;          font-
weight: bold;
transition: .3s ease;
}
```

```
    .logout-
btn:hover {
background:
#007bff;
color: white;    }
```

```
    .todo-container {
margin-top: 40px;
}
```

```
    .task-input-group {
display: flex;
justify-content:
center; align-items:
stretch;
    gap: 16px;
margin-bottom:
20px; }
```

```
    .task-input-group
input {    flex: 1 1
auto;
```

```
padding:
14px;      border:
1px solid #ccc;
border-radius: 8px;
font-size: 16px;
}
```

```
table {
width: 100%;
border-collapse:
collapse;
background: #f9f9f9;
border-radius: 8px;
overflow: hidden;
}
```

```
th, td {
padding: 14px;
text-align:
center;
border: 1px solid #ddd;
}
```

```
th {
background:
#007bff;
color: white;
}
```

```
.edit-btn,
.delete-btn {
display: inline-
block;
width: 80px;
height: 36px;
line-height: 36px;
padding: 0;
font-size: 14px;
font-weight: bold;
border-radius:
6px;      color:
white;
cursor: pointer;
}
```

```
.edit-btn {
background-color:
#28a745;
margin-right: 6px;
```

```

    }

.delete-btn {
  background-color: #dc3545;
}

.error {
  color: red;
  text-align:
  center;
  font-weight:
  bold;
  margin-top:
  10px;
}
</style>
</head>
<body>
  <div class="card" id="authContainer">
    <h2>Вход</h2>
    <form onsubmit="event.preventDefault();">
      <input type="text" id="username" placeholder="Логин">
      <input type="password" id="password" placeholder="Пароль">
    </form>
    <div class="button-group">
      <button onclick="login()">Войти</button>
      <button onclick="register()">Регистрация</button>
    </div>
    <p id="authError" class="error"></p>
  </div>

  <div class="card todo-container" id="todoContainer" style="display:
  none;">
    <h2>Список дел</h2>
    <div class="task-input-group">
      <input type="text" id="taskInput"
placeholder="Новая задача">
      <button
onclick="addTask()">Добавить</button>
      <button class="logout-btn" onclick="logout()">Выйти</button>
    </div>
    <p id="taskError" class="error"></p>
    <table id="taskTable">
      <tr>
        <th>№</th>
        <th>Текст</th>
        <th>Действие</th>
      </tr>
      {{rows}}
    </table>
  </div>

```

```
    </table>
</div>
```

```
    <script>      async function
showError(id, message) {
const el =
document.getElementById(id);
el.textContent = message;
el.style.display = 'block';
    setTimeout(() => { el.style.display = 'none'; }, 3000);
}

async function login() {
const username =
document.getElementById('username').value.trim(); const
password =
document.getElementById('password').value.trim();
    if (!username || !password) return showError('authError',
'Заполните все поля');    try {
        const res = await
fetch('/login', {          method:
'POST',          headers: { 'Content-
Type': 'application/json' },
        body: JSON.stringify({ username, password })
    });
    const data = await res.json();
    if (res.ok) {

document.getElementById('authContainer').style.displa
y = 'none';
document.getElementById('todoContainer').style.displa
y = 'block';          loadTasks();          } else {
        showError('authError', data.error || 'Ошибка входа');
    }
    } catch {
        showError('authError', 'Сервер недоступен');
    }
}

    async function register() {          const username =
document.getElementById('username').value.trim();          const
password = document.getElementById('password').value.trim();
    if (!username || !password) return showError('authError',
'Заполните все поля');    try {
        const res = await
fetch('/register', {          method:
'POST',          headers: { 'Content-
Type': 'application/json' },
```



```

        body: JSON.stringify({ username, password })
    });
    const data = await res.json();
    showError('authError', data.message || data.error);
} catch {
    showError('authError', 'Сервер недоступен');
}
}

async function
logout() {      await
fetch('/logout');
    document.getElementById('todoContainer').style.display = 'none';
document.getElementById('authContainer').style.display = 'block';
}

    async function loadTasks() {      const
res = await fetch('/');      const html =
await res.text();      const parser = new
DOMParser();      const doc =
parser.parseFromString(html, 'text/html');
const newTable =
doc.querySelector('#taskTable');
    document.querySelector('#taskTable').innerHTML = newTable.innerHTML;
attachEventListeners();
}

    async function addTask() {      const text =
document.getElementById('taskInput').value.trim
();      if (!text) return showError('taskError',
'Введите задачу');      try {
        const res = await fetch('/add', {
            method: 'POST',
            headers: { 'Content-Type':
'application/json' },      body:
JSON.stringify({ text })
        });
        const data = await
res.json();      if
(res.ok) loadTasks();
        else showError('taskError', data.error || 'Ошибка');
    } catch {
        showError('taskError', 'Ошибка сервера');
    }
}

    async function deleteTask(id) {
        try {

```

```

        const response = await
fetch(`/delete?id=${id}`, {                                method:
'DELETE'
    });
    const data = await response.json();
    if (response.ok) {
        document.querySelector(`tr td[data-
id="${id}"]`).parentElement.remove();
    } else {
        showError('taskError', data.error || 'Ошибка удаления');
    }
} catch {
    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) {
        textCell.textContent = data.text;
    } else {
        showError('taskError', data.error || 'Ошибка редактирования');
    }
} catch {
    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>
```