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

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

|  |
| --- |
| ОТЧЁТ О ЛАБОРАТОРОНОЙ РАБОТЕ |
| по дисциплине: ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент гр. № | 3333 |  |  |  | Д. Д. Гарнцев |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

**ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ**

Данный проект представляет собой веб-приложение для ведения списка задач (ToDo), включающее регистрацию и вход пользователей, а также подключение Telegram-бота. С помощью бота пользователь может взаимодействовать с задачами напрямую из мессенджера, при этом все изменения сохраняются в общей базе данных.

Ключевые возможности:

1. Создание аккаунта и вход по логину и паролю.
2. Полный набор операций с задачами: добавление, просмотр, редактирование и удаление.
3. Telegram-бот для управления задачами в чате.
4. Поддержка пользовательских сессий для сохранения состояния авторизации.

**Пример работы программы:**

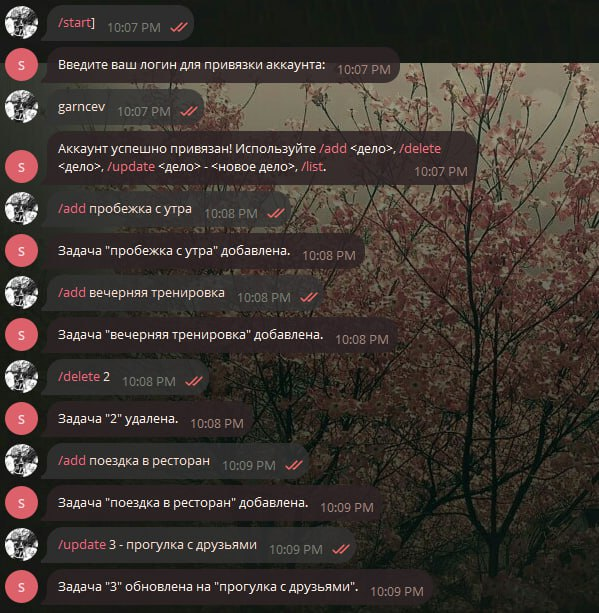


Рисунок 1 – интеграция телеграмма

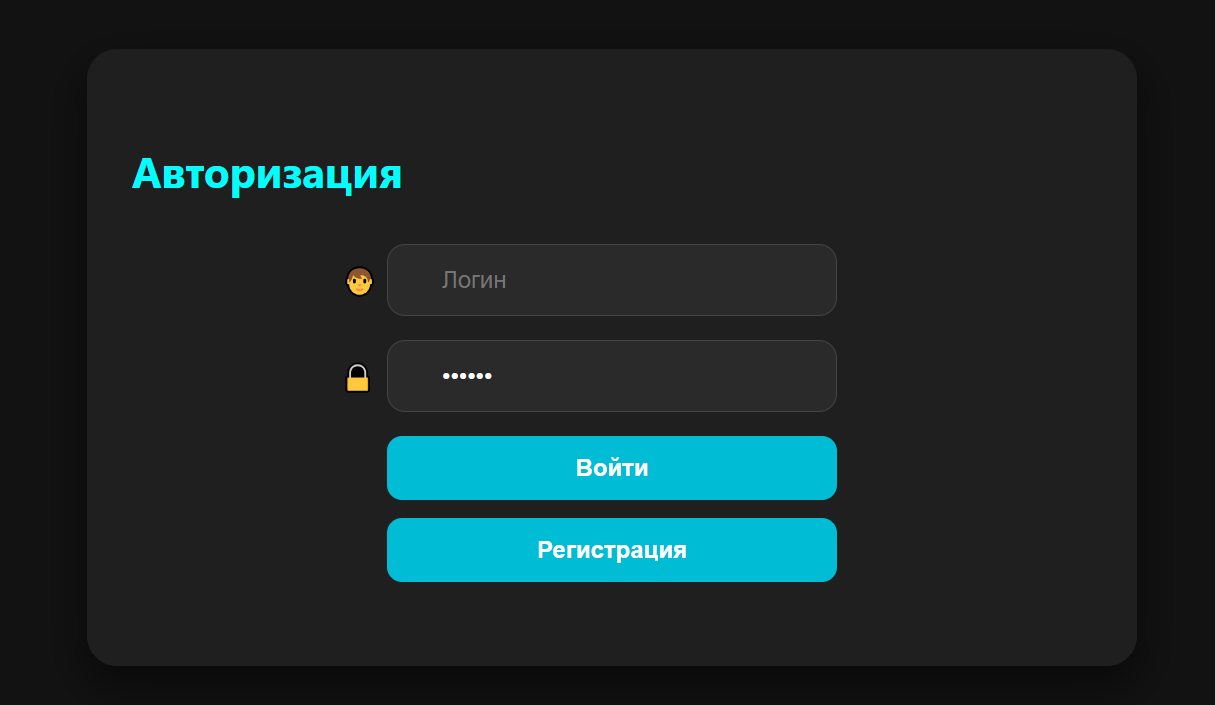


Рисунок 2 – авторизация/регистрация

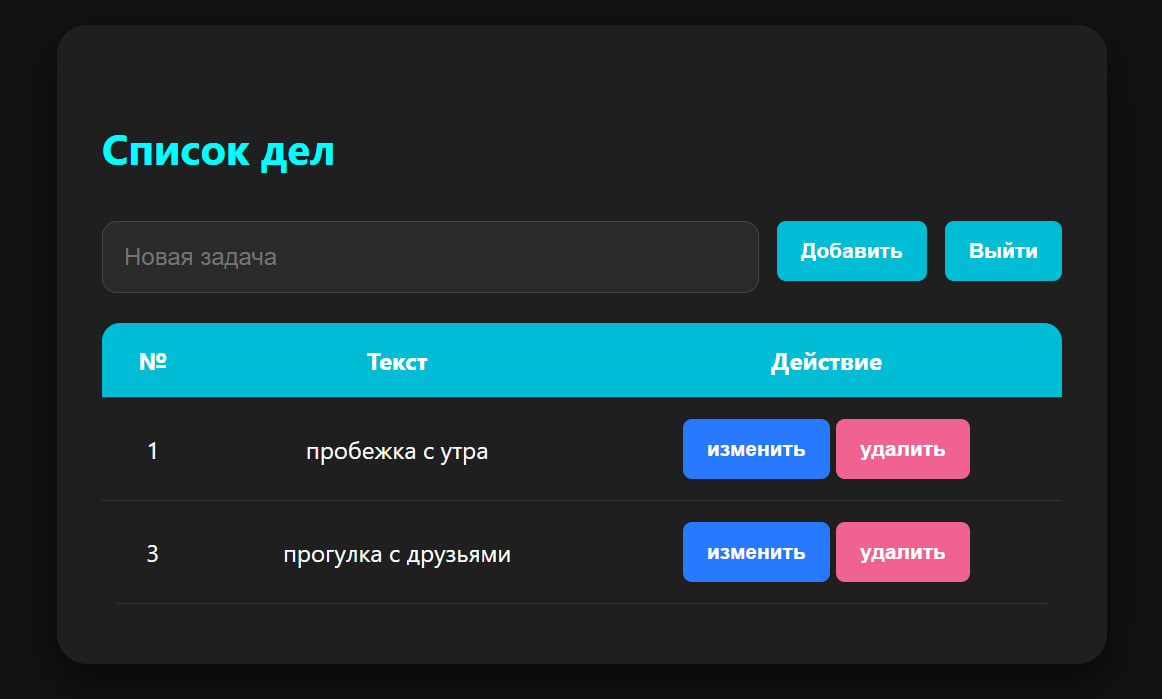


Рисунок 3 – интерфейс

**Проблемы, выявленные по ходу выполнения работы:**

**1. Проблема: Неполные или неоднозначные ответы ИИ**

**Описание:**

Иногда ИИ может предложить неполный или устаревший код, особенно при использовании устаревших версий библиотек или без контекста.

**Решение:**

* Проверять документацию по библиотекам (express, mysql2, bcrypt) вручную.
* Тестировать каждый фрагмент кода и дорабатывать его под актуальную версию Node.js и библиотек.

**2. Проблема: Безопасность данных (в т.ч. Telegram токен и пароли)**

**Описание:** ИИ может предложить хранить конфиденциальные данные прямо в коде (как в serv.js: Telegram токен и пароль MySQL).  
**Решение:**

* Переносить такие данные в .env файл и использовать dotenv для их загрузки.
* Исключить .env из репозитория с помощью .gitignore.

**3. Проблема: Недостаточная защита авторизации**

**Описание:** В реализации регистрации/входа есть потенциальные уязвимости: отсутствие ограничения попыток входа, нет валидации пароля (длина, сложность).  
**Решение:**

* Добавить ограничение количества попыток входа (Rate Limiting).
* Расширить валидацию пароля на клиенте и сервере (например, минимум 8 символов, наличие цифр и букв).
* Добавить helmet и express-rate-limit.

**4. Проблема: Потенциальные ошибки подключения к БД**

**Описание:** Все обращения к базе данных выполняются напрямую без пула подключений и часто без catch-блока внутри try.  
**Решение:**

* Использовать пул подключений из mysql2.
* Оборачивать все SQL-операции в try/catch и логировать ошибки более подробно.

**5. Проблема: Telegram-интеграция ограничена**

**Описание:** Телеграм-бот реализован, но команды /add, /delete, /update требуют точного ввода.  
**Решение:**

* Добавить парсинг с регулярными выражениями и подсказки при ошибках.
* Реализовать inline-кнопки и команду /help через node-telegram-bot-api.

**6. Проблема: Потеря состояния после перезагрузки сервера**

**Описание:** Сессии хранятся в памяти (express-session), что неустойчиво.  
**Решение:**

* Подключить MySQLStore или RedisStore для хранения сессий.

**7. Проблема: Нет защиты от XSS**

**Описание:** Текст задач отображается напрямую в HTML, что создаёт уязвимость XSS.  
**Решение:**

* Использовать библиотеку escape-html (уже установлена) при выводе текста задач.
* Пример: escapeHtml(item.text) при генерации {{rows}}.

**Работа процессов:**

**1. Авторизационный модуль**

**1.1 Создание учётной записи**

1. Пользователь вводит желаемые логин и пароль.
2. Пароль шифруется через bcrypt.hash() (стандартная соль = 10).
3. Логин и зашифрованный пароль сохраняются в таблице users.

-- добавление нового пользователя

INSERT INTO users (username, password)

VALUES (?, ?);

**1.2 Авторизация**

1. Во всплывающей форме авторизации пользователь указывает логин и пароль.
2. Сервер находит в базе строку с таким логином и проверяет пароль функцией bcrypt.compare().
3. Если проверка проходит, создаётся сессия: в req.session.user кладётся id и username.

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

const { username, password } = req.body;

const user = await getUserFromDB(username);

if (user && await bcrypt.compare(password, user.password)) {

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

return res.json({ success: true });

}

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

});

**1.3 Завершение сеанса**

Сессия уничтожается, после чего пользователь перенаправляется на страницу входа.

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

req.session.destroy();

res.redirect('/login');

});

**2. Взаимодействие с Telegram:**

**2.1 Привязка Telegram-аккаунта**

1. Пользователь пишет боту команду /start.
2. Бот запрашивает логин, указанный в веб-приложении.
3. Если логин найден, Telegram-ID сохраняется в поле users.telegram\_id.

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

const chatId = msg.chat.id;

bot.sendMessage(chatId, 'Введите ваш логин:');

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

const username = msg.text.trim();

await linkTelegramId(username, String(chatId));

});

});

**2.2 Управление задачами из чата**

**Добавление**

/add пробежка с утра

Добавить новую задачу

INSERT INTO tasks (text, user\_id) VALUES (?, ?);

**Удаление**

/delete 2

Удалить задачу с id = 2 удаляется у текущего пользователя.

|  |
| --- |
|  |

|  |
| --- |
| DELETE FROM tasks WHERE id = ? AND user\_id = ?; |

**Просмотр списка**

/list

Вывести все задачи пользователя

SELECT id, text FROM tasks WHERE user\_id = ?;

**ВЫВОД**

Разработанное приложение полностью соответствует заданным требованиям. Была реализована стабильная система авторизации и выполнена интеграция с Telegram-ботом, что создаёт хорошую основу для дальнейшего расширения функциональности. В процессе работы возникали трудности, связанные с тем, что искусственный интеллект не всегда точно понимал формулировки запросов. Эти сложности удалось преодолеть путём уточнения вопросов и последовательного взаимодействия, что позволило успешно завершить реализацию проекта.

Приложение 1 – Код serv.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 = '8049995581:AAGUvWHikQUxvJV280vttYBI3xhZB8PMXaY';

const bot = new TelegramBot(telegram\_token, { polling: true });

const dbConfig = {

  host: 'Localhost',

  user: 'root',

  password: 'qwe123!@#',

  database: 'todolist',

};

app.use(express.json());

app.use(session({

  secret: 'your-secret-key',

  resave: false,

  saveUninitialized: false,

  cookie: { secure: false }

}));

async function initDatabase() {

  try {

    const connection = await mysql.createConnection({

      host: dbConfig.host,

      user: dbConfig.user,

      password: dbConfig.password

    });

    console.log('Connected to MySQL server');

    await connection.query('CREATE DATABASE IF NOT EXISTS todolist');

    await connection.query('USE todolist');

    console.log('Database todolist selected');

    await connection.query(`

      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

      )

    `);

    console.log('Table users created');

    await connection.query('DROP TABLE IF EXISTS items');

    await connection.query(`

      CREATE TABLE items (

        id INT AUTO\_INCREMENT PRIMARY KEY,

        text VARCHAR(255) NOT NULL,

        user\_id INT,

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

      )

    `);

    console.log('Table items created');

    await connection.end();

    console.log('Database and tables initialized successfully');

  } catch (error) {

    console.error('Error initializing database:', error);

    throw error;

  }

}

async function retrieveListItems(userId) {

  try {

    const connection = await mysql.createConnection(dbConfig);

    const query = 'SELECT id, text FROM items WHERE user\_id = ?';

    const [rows] = await connection.execute(query, [userId]);

    await connection.close();

    return rows;

  } catch (error) {

    console.error('Ошибка при получении элементов:', error);

    throw error;

  }

}

async function addListItem(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.close();

    return { id: result.insertId, text };

  } catch (error) {

    console.error('Ошибка при добавлении элемента:', error);

    throw error;

  }

}

async function deleteListItem(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.close();

    return result.affectedRows > 0;

  } catch (error) {

    console.error('Ошибка при удалении элемента:', error);

    throw error;

  }

}

async function updateListItem(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.close();

    return result.affectedRows > 0;

  } catch (error) {

    console.error('Ошибка при обновлении элемента:', error);

    throw error;

  }

}

async function getUserByTelegramId(telegramId) {

  try {

    const connection = await mysql.createConnection(dbConfig);

    const [rows] = await connection.execute('SELECT \* FROM users WHERE telegram\_id = ?', [telegramId]);

    await connection.close();

    return rows[0];

  } catch (error) {

    console.error('Ошибка при получении пользователя:', error);

    throw error;

  }

}

async function linkTelegramId(username, telegramId) {

  try {

    const connection = await mysql.createConnection(dbConfig);

    const [userRows] = await connection.execute('SELECT \* FROM users WHERE username = ?', [username]);

    if (userRows.length === 0) {

      await connection.close();

      return { success: false, message: `Пользователь с логином "${username}" не найден. Зарегистрируйтесь на сайте.` };

    }

    const [telegramRows] = await connection.execute('SELECT \* FROM users WHERE telegram\_id = ?', [telegramId]);

    if (telegramRows.length > 0) {

      await connection.close();

      return { success: false, message: `Этот Telegram ID уже привязан к пользователю "${telegramRows[0].username}".` };

    }

    await connection.execute('UPDATE users SET telegram\_id = ? WHERE username = ?', [telegramId, username]);

    await connection.close();

    console.log(`Telegram ID ${telegramId} успешно привязан к пользователю ${username}`);

    return {

      success: true,

      message: 'Аккаунт успешно привязан! Используйте /add <дело>, /delete <дело>, /update <дело> - <новое дело>, /list.'

    };

  } catch (error) {

    console.error('Ошибка при привязке Telegram ID:', error);

    return { success: false, message: 'Ошибка сервера при привязке Telegram ID. Попробуйте позже.' };

  }

}

async function getHtmlRows(userId) {

  const todoItems = await retrieveListItems(userId);

  return todoItems.map(item => `

    <tr>

      <td>${item.id}</td>

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

      <td>

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

        <button class="delete-btn" data-id="${item.id}">удалить</button>

      </td>

    </tr>

  `).join('');

}

function isAuthenticated(req, res, next) {

  if (req.session.user) {

    next();

  } else {

    res.redirect('/login');

  }

}

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

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

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

});

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

  const { username, password } = req.body;

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

    return res.status(400).json({ error: 'Логин и пароль не могут быть пустыми' });

  }

  try {

    const connection = await mysql.createConnection(dbConfig);

    const [rows] = await connection.execute('SELECT \* FROM users WHERE username = ?', [username]);

    await connection.close();

    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: 'Успешный вход' });

    } 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 || username.trim().length === 0 || password.trim().length === 0) {

    return res.status(400).json({ error: 'Логин и пароль не могут быть пустыми' });

  }

  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.close();

    return res.json({ message: 'Пользователь зарегистрирован' });

  } catch (error) {

    console.error(error);

    return res.status(400).json({ error: 'Пользователь уже существует или 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, 'interf.html'), 'utf8');

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

    res.send(processedHtml);

  } catch (err) {

    console.error(err);

    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() === '') {

    res.status(400).json({ error: 'Некорректный или отсутствующий текст' });

    return;

  }

  try {

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

    res.json(newItem);

  } catch (err) {

    console.error(err);

    res.status(500).json({ error: 'Не удалось добавить элемент' });

  }

});

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

  const id = req.query.id;

  if (!id || isNaN(id)) {

    res.status(400).json({ error: 'Некорректный или отсутствующий ID' });

    return;

  }

  try {

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

    if (success) {

      res.json({ message: 'Элемент удален' });

    } else {

      res.status(404).json({ error: 'Элемент не найден' });

    }

  } catch (err) {

    console.error(err);

    res.status(500).json({ error: 'Не удалось удалить элемент' });

  }

});

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() === '') {

    res.status(400).json({ error: 'Некорректный или отсутствующий ID или текст' });

    return;

  }

  try {

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

    if (success) {

      res.json({ message: 'Элемент обновлен', text });

    } else {

      res.status(404).json({ error: 'Элемент не найден' });

    }

  } catch (err) {

    console.error(err);

    res.status(500).json({ error: 'Не удалось обновить элемент' });

  }

});

//бот тг

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

  const chatId = msg.chat.id;

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

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

    const username = msg.text.trim();

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

    bot.sendMessage(chatId, result.message);

  });

});

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

  const chatId = msg.chat.id;

  const text = match[1].trim();

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

  if (!user) {

    bot.sendMessage(chatId, 'Ваш Telegram ID не привязан к аккаунту. Используйте /start для привязки.');

    return;

  }

  try {

    await addListItem(text, user.id);

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

  } catch (error) {

    bot.sendMessage(chatId, 'Ошибка при добавлении задачи.');

  }

});

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

  const chatId = msg.chat.id;

  const text = match[1].trim();

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

  if (!user) {

    bot.sendMessage(chatId, 'Ваш Telegram ID не привязан к аккаунту. Используйте /start для привязки.');

    return;

  }

  try {

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

    if (success) {

      bot.sendMessage(chatId, `Задача "${text}" удалена.`);

    } else {

      bot.sendMessage(chatId, `Задача "${text}" не найдена.`);

    }

  } catch (error) {

    bot.sendMessage(chatId, 'Ошибка при удалении задачи.');

  }

});

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

  const chatId = msg.chat.id;

  const oldText = match[1].trim();

  const newText = match[2].trim();

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

  if (!user) {

    bot.sendMessage(chatId, 'Ваш Telegram ID не привязан к аккаунту. Используйте /start для привязки.');

    return;

  }

  try {

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

    if (success) {

      bot.sendMessage(chatId, `Задача "${oldText}" обновлена на "${newText}".`);

    } else {

      bot.sendMessage(chatId, `Задача "${oldText}" не найдена.`);

    }

  } catch (error) {

    bot.sendMessage(chatId, 'Ошибка при обновлении задачи.');

  }

});

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

  const chatId = msg.chat.id;

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

  if (!user) {

    bot.sendMessage(chatId, 'Ваш Telegram ID не привязан к аккаунту. Используйте /start для привязки.');

    return;

  }

  try {

    const items = await retrieveListItems(user.id);

    if (items.length === 0) {

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

      return;

    }

    const message = items.map(item => `${item.id}. ${item.text}`).join('\n');

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

  } catch (error) {

    bot.sendMessage(chatId, 'Ошибка при получении списка задач.');

  }

});

initDatabase().then(() => {

  app.listen(port, () => console.log(`Сервер запущен на порту ${port}`));

}).catch(err => {

  console.error('Failed to initialize database and start server:', err);

  process.exit(1);

});

Приложение 2 – Код interf.js

<!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>

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

    body {

      margin: 0;

      padding: 0;

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

      background: #121212;

      display: flex;

      align-items: center;

      justify-content: center;

      min-height: 100vh;

    }

    .card {

      background: #1f1f1f;

      padding: 40px 30px;

      border-radius: 20px;

      box-shadow: 0 12px 30px rgba(0, 0, 0, 0.6);

      width: 100%;

      max-width: 700px;

      color: #fff;

    }

    h2 {

      text-align: left;

      margin-bottom: 30px;

      font-size: 28px;

      color: #00ffff;

    }

    .input-icon-group {

      display: flex;

      justify-content: center;

      position: relative;

      margin-bottom: 16px;

    }

    .input-icon-group span {

      position: absolute;

      left: calc(50% - 180px);

      top: 50%;

      transform: translateY(-50%);

      color: #888;

      font-size: 18px;

      pointer-events: none;

    }

    .input-icon-group input {

      width: 300px;

      max-width: 100%;

      padding: 14px 14px 14px 36px;

      border: 1px solid #444;

      border-radius: 12px;

      background: #2a2a2a;

      color: #fff;

      font-size: 16px;

    }

    .auth-buttons {

      display: flex;

      flex-direction: column;

      align-items: center;

      gap: 12px;

    }

    .auth-buttons button {

      width: 300px;

      max-width: 100%;

      padding: 12px;

      background: #00bcd4;

      color: white;

      border: none;

      border-radius: 10px;

      font-weight: bold;

      font-size: 16px;

      cursor: pointer;

      transition: 0.3s ease;

    }

    .auth-buttons button:hover {

      background: #0097a7;

    }

    .error {

      color: #ff5252;

      text-align: center;

      font-weight: bold;

      margin-top: 10px;

    }

  </style>

</head>

<body>

  <div class="card" id="authContainer">

    <h2>Авторизация</h2>

    <form onsubmit="event.preventDefault();">

      <div class="input-icon-group">

        <span>🧑</span>

        <input type="text" id="username" placeholder="Логин">

      </div>

      <div class="input-icon-group">

        <span>🔒</span>

        <input type="password" id="password" placeholder="Пароль">

      </div>

      <div class="auth-buttons">

        <button onclick="login()">Войти</button>

        <button onclick="register()">Регистрация</button>

      </div>

    </form>

    <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>

  <style>

    .todo-container {

      margin-top: 40px;

    }

    .task-input-group {

      display: flex;

      gap: 12px;

      margin-bottom: 20px;

    }

    .task-input-group input {

      flex-grow: 1;

      padding: 14px;

      border: 1px solid #444;

      border-radius: 10px;

      background: #2a2a2a;

      color: white;

      font-size: 16px;

    }

    .edit-btn, .delete-btn, .task-input-group button {

      height: 40px;

      padding: 0 16px;

      font-size: 14px;

      font-weight: bold;

      border: none;

      border-radius: 6px;

      cursor: pointer;

      transition: background 0.2s ease;

    }

    .task-input-group button {

      background-color: #00bcd4;

      color: #fff;

    }

    .task-input-group button:hover {

      background-color: #0097a7;

    }

    .edit-btn {

      background-color: #2979ff;

      color: #fff;

    }

    .edit-btn:hover {

      background-color: #1565c0;

    }

    .delete-btn {

      background-color: #f06292;

      color: #fff;

    }

    .delete-btn:hover {

      background-color: #e91e63;

    }

    table {

      width: 100%;

      border-collapse: collapse;

      margin-top: 20px;

      border-radius: 12px;

      overflow: hidden;

    }

    th, td {

      padding: 14px;

      text-align: center;

      background: #1f1f1f;

      color: white;

      border-bottom: 1px solid #333;

    }

    th {

      background: #00bcd4;

      color: #fff;

    }

    .logout-btn {

      background: transparent;

      border: 2px solid #00bcd4;

      color: #00bcd4;

      height: 40px;

      padding: 0 16px;

      border-radius: 6px;

    }

    .logout-btn:hover {

      background: #00bcd4;

      color: white;

    }

  </style>

  <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.display = 'none';

          document.getElementById('todoContainer').style.display = '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>