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 |  | 23.06.25 |  | К. А. Жиданов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |
|  |  |  |  |  |

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

|  |
| --- |
| **ВЕБ-ПРИЛОЖЕНИЕ «TO-DO LIST»** |
| по дисциплине: ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент гр. № | 3333 |  | 23.06.25 |  | И. С. Махин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург  
2025

**1. Цель работы**

Создать веб-приложение для управления задачами (To-Do List) с возможностью аутентификации и интеграции приложения в Telegram-бота.

Бот позволяет управлять задачами через мессенджер, синхронизируя данные с основной базой данных. Также, взаимодействовать с задачами можно и через сайт.

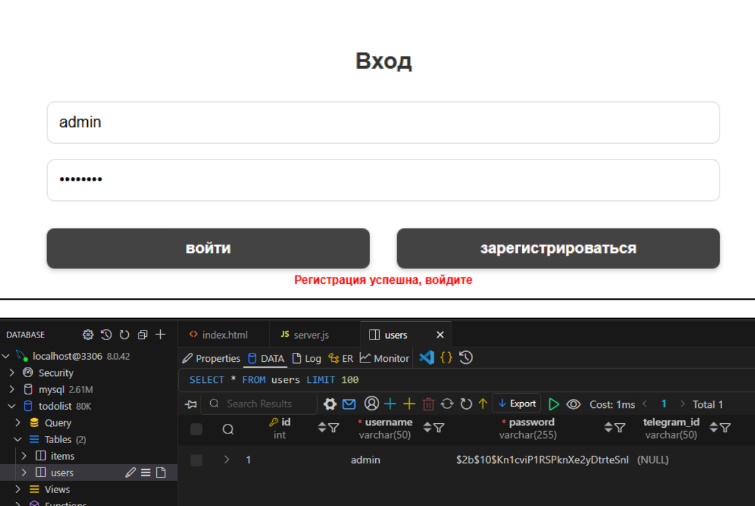
Основные задачи проекта:

1. Регистрация и аутентификация пользователей.

2. Создание, изменение, удаление, обновление списка задач.

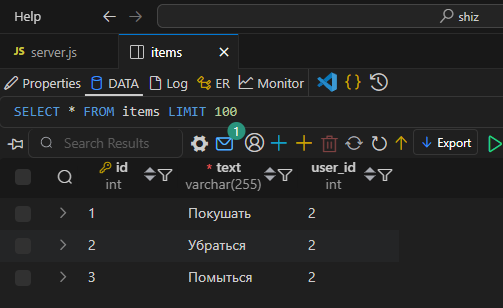
3. Telegram-бот для управления задачами через чат.

**2. Пример работы программы**

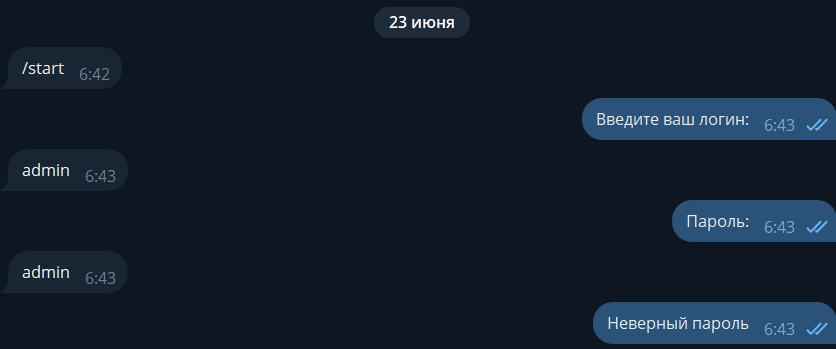


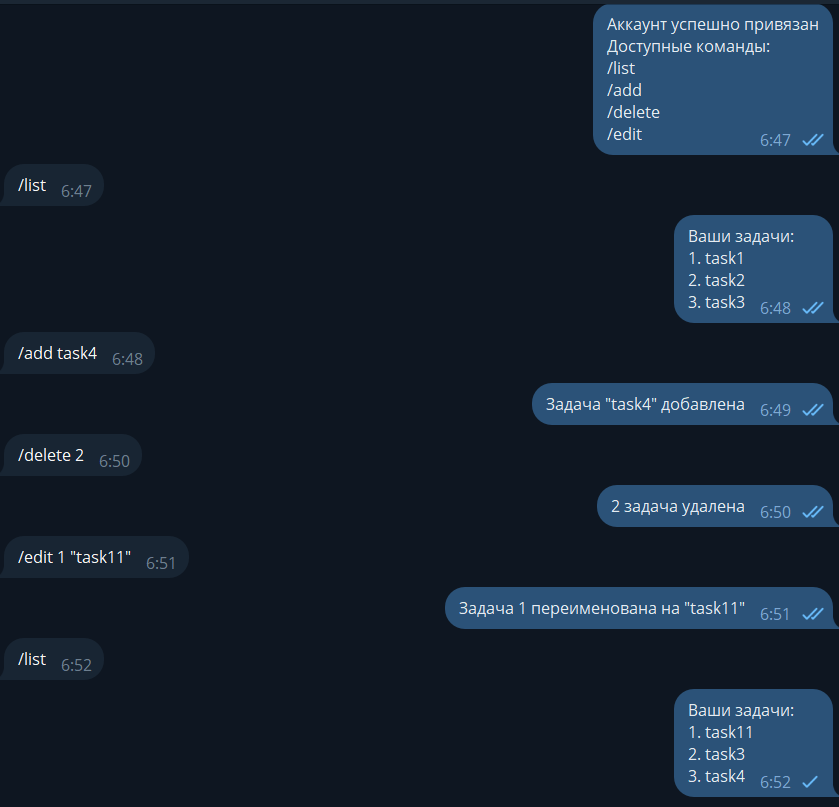
*Рисунок 1 — Авторизация (создание аккаунта)*

**

**

*Рисунок 2 — Интерфейс To-Do List*

*Рисунок 3 — Интеграция в ТГ бота (неудачная попытка авторизации)*

**

*Рисунок 4 – использование команд в ТГ боте*

**3. Работа с ИИ**

Используемые промпты:

* Представь, что ты студент ВУЗА и тебя попросили написать код для веб-приложения, которое представляет собой To-Do List с авторизацией и интеграцией в ТГ бота. От тебя требуется файлы с расширением html, js, sql. Стили можешь прописать в html. Не используй env, пиши в открытый код.
* Разработай web-приложение для управления задачами с авторизацией пользователей и Telegram-интеграцией. Включает:

1. Базу данных для хранения пользователей и их задач
2. Веб-сервер с REST API для:
   * Регистрации/входа через логин-пароль
   * CRUD операций над задачами
   * Привязки Telegram-аккаунтов
3. Telegram-бота с командами:
   * Привязка аккаунта
   * Добавление/удаление/редактирование/просмотр задач
4. Клиентский интерфейс с:
   * Адаптивным дизайном
   * Формой входа
   * Интерактивной таблицей задач
   * Динамическим обновлением без перезагрузки
5. Безопасность:
   * Хеширование паролей
6. Интеграцию компонентов:
   * Синхронизацию данных между вебом и Telegram
   * Единую БД для всех операций
   * Автоматическое обновление интерфейса

* Были также использованы более мелкие по масштабу запросы для ИИ, в которых я уточнял некоторые недоработки, принципы работы и способы запуска веб-приложения.

Проблемы при работе с ИИ:

1) При каждом запросе переписать код html, для исправления какой-либо ошибки (поскольку script.js тоже расположен внутри html файла), ИИ меняла дизайн сайта несмотря на то, что я просил этого не делать.

2) После перезапуска сервера, пользователи сохранялись в базе, однако их данные из таблицы – нет.

3) Возникла проблема с переносом данных, которые должны быть скрыты, в файл с расширением «env», поскольку VS Code не мог подгрузить пакет «Dotenv».

4) Бот ТГ не получал обновления из базы данных из-за неправильной конфигурации.

5) Изначально бот также не проверял, привязан ли Telegram ID к пользователю

6) Некорректные настройки подключения (host, user, password) могли вызывать ошибки.

7) При попытке составления правильной структуры проекта (public data + server data), не подгружались нужные пакеты (возможно связано с тем, что они устарели и их надо ставить вручную).

**4. Работа внутренних процессов**

4.1 Регистрация

• Пользователь вводит логин и пароль;

• проверка на наличие такого пользователя в системе;

• пароль хешируется;

• данные сохраняются в таблицу «users».

4.2 Вход

• Пользователь вводит логин и пароль;

• сервер ищет пользователя в БД и сравнивает хеши (bcrypt.compare);

• при совпадении паролей – создается сессия.

4.3 Выход

• Уничтожение сессии.

4.4 Привязка аккаунта

• Пользователь пишет боты «/start»;

• бот просит ввести пользователя логин и пароль из БД;

• в случае совпадения – TG ID также сохраняется в БД, связывая аккаунт.

4.5 Работа бота

• «/add» - команда для добавления элемента в таблицу (просто текст);

• «/list» - выводит список элементов из таблицы вместе с их ID;

• «/edit» - производит замену элемента по его ID;

• «/delete» - производит удаление элемента из БД по его ID.

**5. Вывод**

Разработанное решение успешно соответствует требованиям задачи, обеспечивая корректную интеграцию Telegram и авторизацию, с потенциалом для дальнейшего развития. При работе возникла проблема неверной интерпретации запросов искусственным интеллектом, которую удалось решить через диалог с уточняющими вопросами, что привело к положительному результату.

**Приложение 1. Код программы INIT.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)

);

**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=2.0">

  <title>Список задач</title>

  <style>

    body {

      font-family: 'Arial', sans-serif;

      max-width: 700px;

      margin: 0 auto;

      padding: 300px;

      background-color: #ffffff00;

      color: black;

    }

    h2 {

      color: #333;

      margin-top: 0;

      text-align: center;

      font-size: 24px;

      margin-bottom: 20px;

    }

    .input-group {

      margin-bottom: 15px;

    }

    input {

      padding: 12px;

      margin: 8px 0;

      border: 1px solid #ddd;

      border-radius: 8px;

      width: calc(100% - 26px);

      box-sizing: content-box;

      font-size: 16px;

    }

    .button-group {

      display: flex;

      justify-content: space-between;

      margin-top: 20px;

    }

    button {

      padding: 12px 20px;

      background-color: #212121d8;

      color: white;

      border: none;

      border-radius: 8px;

      cursor: pointer;

      transition: all 0.3s;

      font-size: 16px;

      font-weight: bold;

      box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);

      width: 48%;

    }

    button:hover {

      background-color: #212121d8;

      transform: translateY(-2px);

      box-shadow: 0 4px 8px rgba(0, 0, 0, 0.3);

    }

    table {

      table-layout: fixed;

      width: 100%;

      border-collapse: collapse;

      margin-top: 20px;

      background-color: #f3f3f3;

      border-radius: 4px;

      overflow: hidden;

    }

    th:nth-child(1) { width: 5%; }

    th:nth-child(2) { width: 55%; }

    th:nth-child(3) { width: 25%; }

    th, td {

      border: 1px solid #6665658f;

      padding: 14px;

      text-align: center;

      color: #000000;

    }

    th {

      background-color: #C0C0C0;

      font-weight: bold;

    }

    .error {

      color: red;

      display: none;

      margin-top: 5px;

      font-weight: bold;

      text-align: center;

    }

    .logout-btn {

      background-color: #212121d8;

      width: auto;

      margin-bottom: 15px;

      position: absolute;

      top: 100px;

      right: 100px;

    }

    .edit-btn {

      width: 70px;

      margin-right: 5px;

      align-items: center;

      justify-content: center;

      display: inline-flex;

      height: 30px;

    }

    .delete-btn {

      width: 70px;

      align-items: center;

      justify-content: center;

      display: inline-flex;

      height: 30px;

    }

    .task-input-group {

      display: flex;

      gap: 10px;

      margin-bottom: 10px;

    }

    .task-input-group button {

      margin-top: 8px;

      height: 44px;

    }

    #taskInput {

      flex-grow: 1;

    }

  </style>

</head>

<body>

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

    <h2>Вход</h2>

    <div class="input-group">

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

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

    </div>

    <div class="button-group">

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

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

    </div>

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

  </div>

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

    <h2>To-Do List</h2>

    <button class="logout-btn" onclick="logout()">выйти</button>

    <div class="task-input-group">

      <input type="text" id="taskInput" placeholder="Новая задача">

      <button onclick="addTask()">add</button>

    </div>

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

    <table id="taskTable">

      <tr>

        <th>№</th>

        <th>Text</th>

        <th>Action</th>

      </tr>

      {{rows}}

    </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 || !password) {

        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';

          await loadTasks();

        } else {

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

        }

      } catch {

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

      }

    }

    async function register() {

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

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

      if (!username || !password) {

        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 {

        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 {

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

      }

    }

    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 newTableRows = doc.querySelector('#taskTable').innerHTML;

        document.querySelector('#taskTable').innerHTML = newTableRows;

        attachEventListeners(); // << ВАЖНО: обновляем обработчики после загрузки

      } catch {

        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 = '';

          await loadTasks(); // Перезагружаем таблицу с новыми данными и событиями

        } else {

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

        }

      } 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) {

          await loadTasks();

        } 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) {

          await loadTasks();

        } 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(); // для подставленных {{rows}}, если сервер их встраивает напрямую

    });

  </script>

</body>

</html>

**Приложение 3. Код программы SERVER.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 = '** **7751087406:AAFAKJ\_xpOIstfj0azxHdfccMOqCJl2sCSw';**

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

**const dbConfig = {**

**host: 'localhost',**

**user: 'root',**

**password: 'root1234',**

**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 or already exists');**

**await connection.query(`**

**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)**

**)**

**`);**

**console.log('Table items created or already exists');**

**await connection.end();**

**console.log('Database initialization completed');**

**} 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 getUserByUsername(username) {**

**try {**

**const connection = await mysql.createConnection(dbConfig);**

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

**await connection.close();**

**return rows[0];**

**} catch (error) {**

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

**throw error;**

**}**

**}**

**async function linkTelegramId(username, password, telegramId) {**

**try {**

**const user = await getUserByUsername(username);**

**if (!user) {**

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

**}**

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

**if (!match) {**

**return { success: false, message: 'Неверный пароль' };**

**}**

**const [telegramRows] = await (await mysql.createConnection(dbConfig))**

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

**if (telegramRows.length > 0) {**

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

**}**

**await (await mysql.createConnection(dbConfig))**

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

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

**return {**

**success: true,**

**message: 'Аккаунт успешно привязан!\n\n/list\n/add <Text>\n/delete <№ задачи>\n/edit <№ задачи> - <Text>',**

**};**

**} 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}">edit</button>**

**<button class="delete-btn" data-id="${item.id}">delete</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, 'index.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, 'index.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('/edit', 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' });**

**}**

**});**

**// Telegram bot handlers**

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

**const chatId = msg.chat.id;**

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

**let username = '';**

**const onUsername = async (msg) => {**

**username = msg.text.trim();**

**const user = await getUserByUsername(username);**

**if (!user) {**

**bot.sendMessage(chatId, `Пользователь с логином "${username}" не найден. Зарегистрируйтесь на сайте.`);**

**bot.removeListener('message', onUsername);**

**return;**

**}**

**bot.sendMessage(chatId, 'Теперь введите ваш пароль:');**

**bot.removeListener('message', onUsername);**

**const onPassword = async (msg) => {**

**const password = msg.text.trim();**

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

**bot.sendMessage(chatId, result.message);**

**bot.removeListener('message', onPassword);**

**};**

**bot.once('message', onPassword);**

**};**

**bot.once('message', onUsername);**

**});**

**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, `"${text}" задача удалена.`);**

**} else {**

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

**}**

**} catch (error) {**

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

**}**

**});**

**bot.onText(/\/edit (.+) - (.+)/, 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 "${oldText}" переименована на "${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);**

**});**