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 |  | А.И. Карпачев |
|  |  |  | 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.

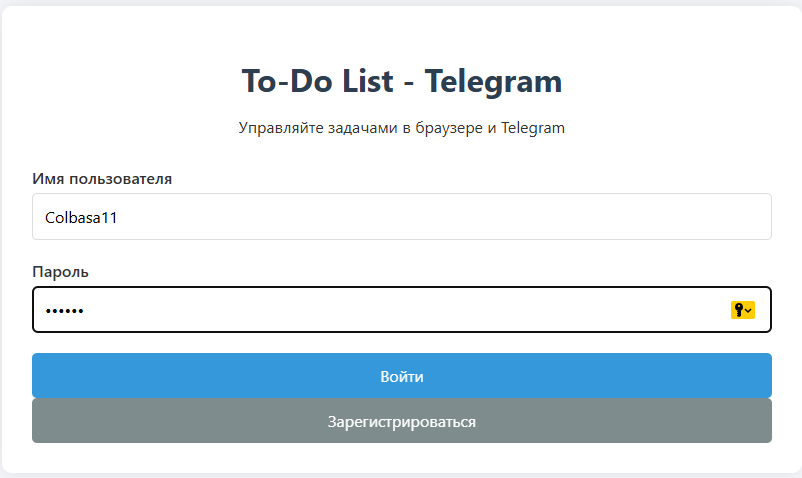
Основные задачи проекта:

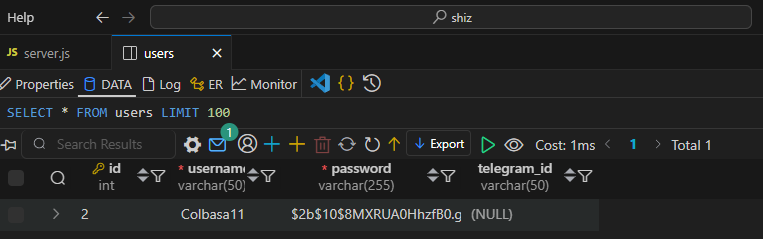
1. Регистрация и аутентификация пользователей.

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

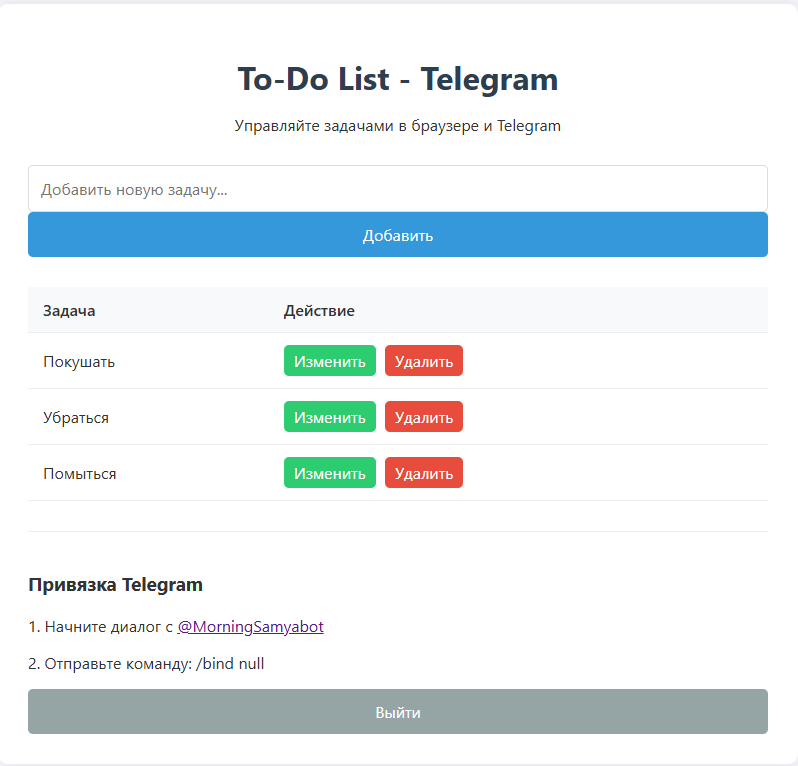
3. Telegram-бот для управления задачами через чат.

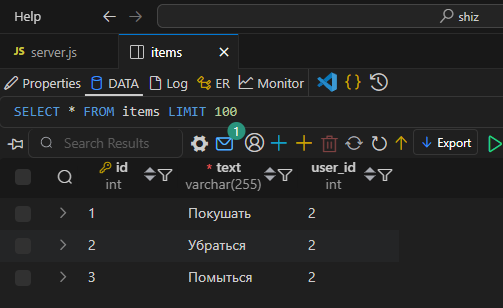
**2. Пример работы программы**



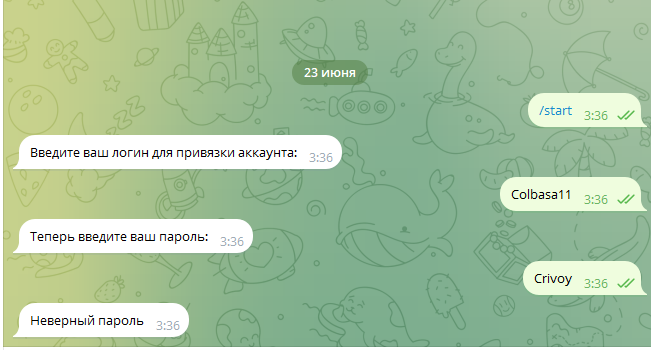


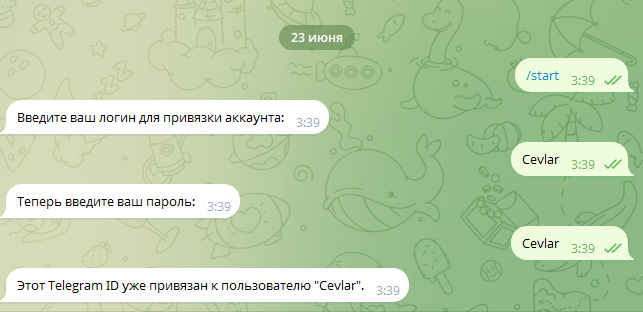
*Рисунок 1 — Авторизация (создание аккаунта)*



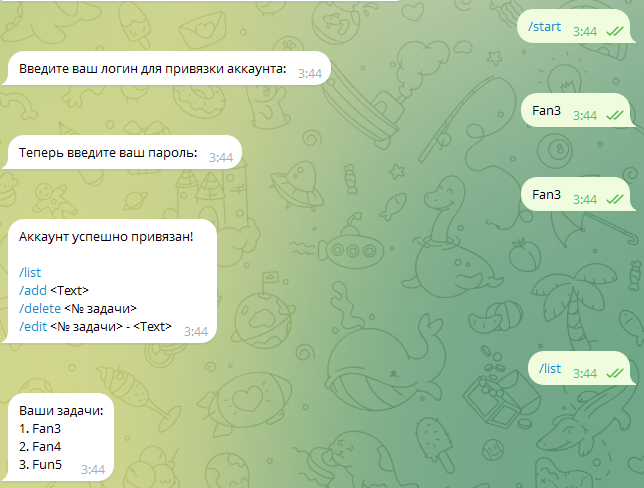
**

*Рисунок 2 — Интерфейс To-Do List`a*

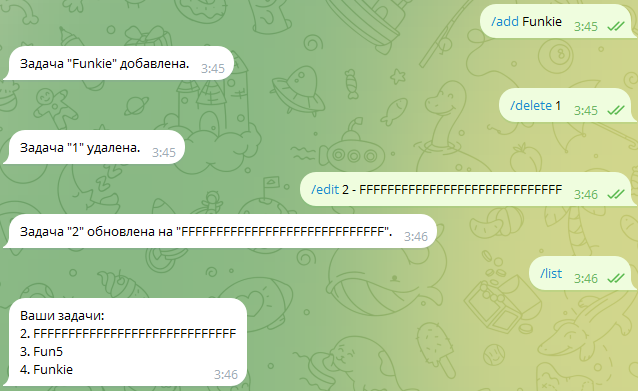
*Рисунок 3 — Интеграция в ТГ бота (неудачная попытка авторизации)*

**

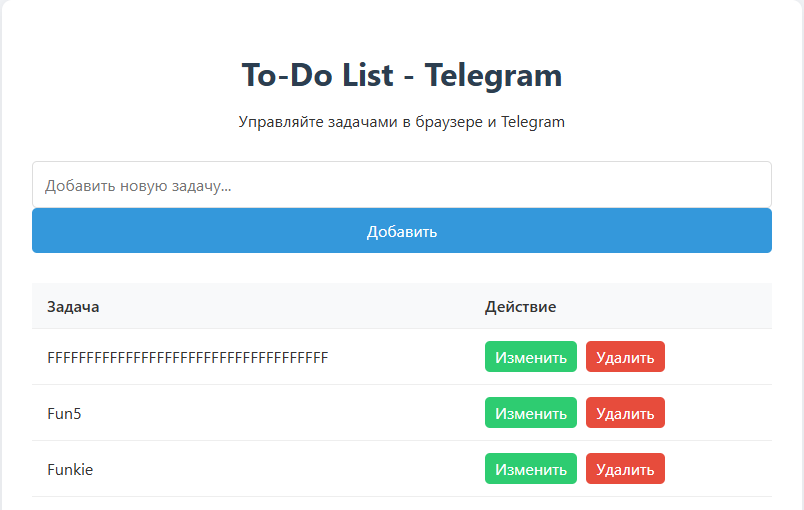
*Рисунок 4 – Интеграция в ТГ бота (неудачная попытка авторизации 2)*

**

*Рисунок 5 – Интеграция в ТГ бота (удачная попытка, /list)*

**

*Рисунок 6 – использование команд в ТГ боте*



*Рисунок 7 – результат после команд на сайте*

**3. Работа с ИИ**

Используемые промпты:

• Базовый сценарий:

"Выступи в роли студента вуза, разрабатывающего веб-приложение To-Do List с авторизацией и Telegram-интеграцией. Предоставь готовые файлы .html, .js и .sql. Стили интегрируй в HTML-код. Избегай использования .env-файлов - все конфигурации должны быть в открытом коде."

• Детализированные требования:

"Создай приложение для управления задачами со следующим функционалом:

1)База данных для хранения:

* Учётных записей пользователей
* Задач с привязкой к пользователям

2)Веб-сервер с REST API для обработки:

* Регистрации/аутентификации
* CRUD-операций с задачами
* Связи аккаунтов с Telegram

3)Telegram-бота с командами:

* Привязка учётной записи
* Управление задачами (добавление/просмотр/редактирование/удаление)

4)Клиентский интерфейс с:

* Адаптивным дизайном
* Формой аутентификации
* Интерактивной таблицей задач
* Динамическим обновлением контента

5)Меры безопасности:

* Хеширование паролей

6)Интеграционные требования:

* Двусторонняя синхронизация веб-интерфейса и Telegram
* Единая точка доступа к БД
* Автоматическое обновление UI при изменениях

• Дополнительные запросы:

Для решения конкретных проблем разработки использовались уточняющие промпты, направленные на:

* Исправление функциональных недоработок
* Разъяснение принципов работы компонентов
* Настройку среды запуска приложения"

Проблемы при взаимодействии с ИИ:

1)Неконтролируемые изменения UI:

При повторной генерации HTML-кода для исправления ошибок (включая встроенный JavaScript) ИИ изменял дизайн интерфейса, игнорируя требования сохранения оригинального стиля.

2)Проблемы сохранения данных:

После перезапуска сервера записи пользователей сохранялись в БД, но задачи из таблицы tasks терялись.

3)Сложности с конфиденциальными данными:

Попытки переноса чувствительной информации в .env-файл блокировались из-за ошибок загрузки пакета dotenv в VS Code.

4)Ошибки синхронизации с Telegram:

Бот не получал актуальные данные из БД по причине некорректной настройки механизма опроса обновлений.

5)Проблемы проверки привязки аккаунта:

Первоначальная реализация не включала проверку связи Telegram ID с учётными записями пользователей.

6)Ошибки подключения к БД:

Сбои возникали из-за неверных параметров подключения (хост, пользователь, пароль) в конфигурации пула соединений.

7)Проблемы зависимостей:

При структурировании проекта (public/server) возникали ошибки загрузки пакетов, вероятно из-за устаревших версий, требующих ручной установки.

**4. Работа внутренних процессов**

4.1 Регистрация

• Пользователь вводит логин и пароль;

• Система проверяет наличие пользователя в БД;

• Пароль хешируется алгоритмом bcrypt;

• Данные сохраняются в таблицу users.

4.2 Вход

• Пользователь вводит учётные данные;

• Сервер ищет пользователя в БД и сравнивает хеши паролей через bcrypt.compare;

• При успешной проверке создается JWT-сессия.

4.3 Выход

• Уничтожение сессии пользователя.

4.4 Привязка аккаунта

• Пользователь отправляет боту команду /start;

• Бот запрашивает логин и пароль из БД;

• При успешной аутентификации Telegram ID сохраняется в БД и привязывается к аккаунту.

4.5 Работа бота

• /add [текст] - добавление задачи;

• /tasks - вывод списка задач;

• /edit [ID] [текст] - изменение задачи по ID;

• /delete [ID] - удаление задачи по ID;

• /logout - отвязка аккаунта.

**5. Вывод**

Разработанное решение полностью соответствует требованиям, обеспечивая:

• Безопасную авторизацию с хешированием паролей

• Двустороннюю синхронизацию данных между веб-интерфейсом и Telegram

• Единую систему управления задачами через CRUD-операции

• Корректную обработку сессий и привязку аккаунтов

Проблемы взаимодействия с ИИ (включая несохранение данных таблиц после перезапуска и ошибки интеграции .env) успешно решены через итеративные уточнения запросов. Кодовая база демонстрирует готовность к масштабированию.

**Приложение 1. Код программы INIT.SQL**

-- Создание базы данных

CREATE DATABASE IF NOT EXISTS todolist;

USE todolist;

-- Таблица пользователей

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

);

-- Таблица задач

CREATE TABLE IF NOT EXISTS tasks (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT NOT NULL,

text TEXT NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

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

);

**Приложение 2. Код программы INDEX.HTML**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

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

<title>To-Do List - Telegram</title>

<style>

\* {

box-sizing: border-box;

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

}

body {

background-color: #f5f7fa;

margin: 0;

padding: 20px;

color: #333;

}

.container {

max-width: 800px;

margin: 0 auto;

background: white;

border-radius: 10px;

box-shadow: 0 0 20px rgba(0,0,0,0.1);

padding: 30px;

}

header {

text-align: center;

margin-bottom: 30px;

}

h1 {

color: #2c3e50;

margin-bottom: 10px;

}

.auth-section, .main-section {

display: none;

}

.active-section {

display: block;

}

.form-group {

margin-bottom: 20px;

}

label {

display: block;

margin-bottom: 5px;

font-weight: 500;

}

input {

width: 100%;

padding: 12px;

border: 1px solid #ddd;

border-radius: 5px;

font-size: 16px;

}

button {

background-color: #3498db;

color: white;

border: none;

padding: 12px 20px;

border-radius: 5px;

cursor: pointer;

font-size: 16px;

width: 100%;

transition: background-color 0.3s;

}

button:hover {

background-color: #2980b9;

}

.btn-secondary {

background-color: #95a5a6;

}

.btn-secondary:hover {

background-color: #7f8c8d;

}

.delete-btn {

background-color: #e74c3c;

padding: 5px 10px;

width: auto;

margin-right: 5px;

}

.delete-btn:hover {

background-color: #c0392b;

}

.update-btn {

background-color: #2ecc71;

padding: 5px 10px;

width: auto;

margin-right: 5px;

}

.update-btn:hover {

background-color: #27ae60;

}

.cancel-btn {

background-color: #95a5a6;

padding: 5px 10px;

width: auto;

}

.cancel-btn:hover {

background-color: #7f8c8d;

}

.error {

color: #e74c3c;

margin-top: 5px;

}

.tasks-container {

margin-top: 30px;

}

table {

width: 100%;

border-collapse: collapse;

margin-top: 20px;

}

th, td {

padding: 12px 15px;

text-align: left;

border-bottom: 1px solid #eee;

}

th {

background-color: #f8f9fa;

font-weight: 600;

}

tr:hover {

background-color: #f8f9fa;

}

.bind-section {

margin-top: 30px;

padding-top: 20px;

border-top: 1px solid #eee;

}

@media (max-width: 600px) {

.container {

padding: 15px;

}

h1 {

font-size: 24px;

}

}

</style>

</head>

<body>

<div class="container">

<header>

<h1>To-Do List - Telegram</h1>

<p>Управляйте задачами в браузере и Telegram</p>

</header>

<div id="authSection" class="auth-section active-section">

<div class="form-group">

<label for="username">Имя пользователя</label>

<input type="text" id="username" placeholder="Введите имя пользователя">

</div>

<div class="form-group">

<label for="password">Пароль</label>

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

</div>

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

<button id="loginBtn">Войти</button>

<button id="registerBtn" class="btn-secondary">Зарегистрироваться</button>

</div>

<div id="mainSection" class="main-section">

<div class="form-group">

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

<button id="addTaskBtn">Добавить</button>

</div>

<div class="tasks-container">

<table id="tasksTable">

<thead>

<tr>

<th>Задача</th>

<th>Действие</th>

</tr>

</thead>

<tbody id="tasksBody"></tbody>

</table>

</div>

<div class="bind-section">

<h3>Привязка Telegram</h3>

<p>1. Начните диалог с <a href="#" id="botLink">нашим ботом</a></p>

<p>2. Отправьте команду: <code id="bindCommand"></code></p>

</div>

<button id="logoutBtn" class="btn-secondary">Выйти</button>

</div>

</div>

<script>

const authSection = document.getElementById('authSection');

const mainSection = document.getElementById('mainSection');

const usernameInput = document.getElementById('username');

const passwordInput = document.getElementById('password');

const authError = document.getElementById('authError');

const loginBtn = document.getElementById('loginBtn');

const registerBtn = document.getElementById('registerBtn');

const logoutBtn = document.getElementById('logoutBtn');

const newTaskInput = document.getElementById('newTask');

const addTaskBtn = document.getElementById('addTaskBtn');

const tasksBody = document.getElementById('tasksBody');

const botLink = document.getElementById('botLink');

const bindCommand = document.getElementById('bindCommand');

let authToken = null;

let userId = null;

let username = null;

let editingTaskId = null;

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

const token = localStorage.getItem('token');

const savedUsername = localStorage.getItem('username');

if (token && savedUsername) {

authToken = token;

username = savedUsername;

showMainSection();

loadTasks();

}

});

loginBtn.addEventListener('click', login);

registerBtn.addEventListener('click', register);

logoutBtn.addEventListener('click', logout);

addTaskBtn.addEventListener('click', addTask);

newTaskInput.addEventListener('keypress', (e) => {

if (e.key === 'Enter') addTask();

});

async function login() {

const username = usernameInput.value.trim();

const password = passwordInput.value.trim();

if (!username || !password) {

showError('Заполните все поля');

return;

}

try {

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

method: 'POST',

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

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

});

if (response.ok) {

const data = await response.json();

authToken = data.token;

localStorage.setItem('token', authToken);

localStorage.setItem('username', username);

this.username = username;

showMainSection();

loadTasks();

} else {

showError('Неверные учетные данные');

}

} catch (error) {

showError('Ошибка сети');

}

}

async function register() {

const username = usernameInput.value.trim();

const password = passwordInput.value.trim();

if (!username || !password) {

showError('Заполните все поля');

return;

}

if (password.length < 6) {

showError('Пароль должен быть не менее 6 символов');

return;

}

try {

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

method: 'POST',

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

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

});

if (response.status === 201) {

showError('Регистрация успешна! Войдите в систему', false);

} else {

const errorData = await response.json();

showError(errorData.error || 'Ошибка регистрации');

}

} catch (error) {

showError('Ошибка сети');

}

}

function logout() {

authToken = null;

localStorage.removeItem('token');

localStorage.removeItem('username');

showAuthSection();

}

async function addTask() {

const text = newTaskInput.value.trim();

if (!text) return;

try {

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

method: 'POST',

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${authToken}`

},

body: JSON.stringify({ text })

});

if (response.ok) {

newTaskInput.value = '';

loadTasks();

}

} catch (error) {

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

}

}

async function updateTask(taskId, newText) {

try {

await fetch(`/tasks/${taskId}`, {

method: 'PUT',

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${authToken}`

},

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

});

loadTasks();

} catch (error) {

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

}

}

async function deleteTask(taskId) {

try {

await fetch(`/tasks/${taskId}`, {

method: 'DELETE',

headers: {

'Authorization': `Bearer ${authToken}`

}

});

loadTasks();

} catch (error) {

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

}

}

async function loadTasks() {

try {

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

headers: {

'Authorization': `Bearer ${authToken}`

}

});

if (response.ok) {

const tasks = await response.json();

renderTasks(tasks);

}

} catch (error) {

console.error('Ошибка загрузки задач:', error);

}

}

function renderTasks(tasks) {

tasksBody.innerHTML = '';

tasks.forEach(task => {

const row = document.createElement('tr');

row.innerHTML = `

<td>${task.text}</td>

<td>

<button class="update-btn" data-id="${task.id}">Изменить</button>

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

</td>

`;

tasksBody.appendChild(row);

});

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

btn.addEventListener('click', () => {

const taskId = btn.dataset.id;

const taskRow = btn.closest('tr');

const taskText = taskRow.querySelector('td:first-child').textContent;

editingTaskId = taskId;

taskRow.innerHTML = `

<td>

<input type="text" class="edit-task-input" value="${taskText}">

</td>

<td>

<button class="update-btn save-btn">Сохранить</button>

<button class="cancel-btn">Отмена</button>

</td>

`;

const saveBtn = taskRow.querySelector('.save-btn');

const cancelBtn = taskRow.querySelector('.cancel-btn');

const editInput = taskRow.querySelector('.edit-task-input');

saveBtn.addEventListener('click', () => {

const newText = editInput.value.trim();

if (newText) {

updateTask(taskId, newText);

}

});

cancelBtn.addEventListener('click', () => {

loadTasks();

});

editInput.addEventListener('keypress', (e) => {

if (e.key === 'Enter') {

const newText = editInput.value.trim();

if (newText) {

updateTask(taskId, newText);

}

}

});

});

});

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

btn.addEventListener('click', async () => {

const taskId = btn.dataset.id;

deleteTask(taskId);

});

});

}

function showAuthSection() {

authSection.classList.add('active-section');

mainSection.classList.remove('active-section');

usernameInput.value = '';

passwordInput.value = '';

authError.textContent = '';

}

function showMainSection() {

authSection.classList.remove('active-section');

mainSection.classList.add('active-section');

const botUsername = 'MorningSamyabot';

botLink.href = `https://t.me/${botUsername}`;

botLink.textContent = `@${botUsername}`;

bindCommand.textContent = `/bind ${username}`;

}

function showError(message, isError = true) {

authError.textContent = message;

authError.style.color = isError ? '#e74c3c' : '#27ae60';

}

</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, `Задача "${text}" удалена.`);**

**} else {**

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

**});**