МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский государственный университет аэрокосмического приборостроения»

кАФЕДРА 33

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

ОТЧЕТ  
О ЛАБОРАТОРНОЙ РАБОТЕ

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

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент гр. № | 3333 |  |  |  | Исаев А.С. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

**1.ЦЕЛЬ РАБОТЫ**

Цель данной лабораторной работы — усовершенствование исходного кода приложения «To-Do List», реализованного на Node.js с использованием базы данных MySQL. В ходе выполнения проекта требуется поэтапно реализовать полный набор функций для управления задачами: добавление, удаление, редактирование, а также обеспечить систему авторизации пользователей и связать веб-приложение с Telegram-ботом. Разрешалось использовать большие языковые модели (LLM) для генерации отдельных участков кода и подготовки сопровождающей документации.

****2. код сайта****

**app.py**

**from flask import Flask, render\_template, request, redirect, url\_for, session, flash, jsonify**

**import mysql.connector**

**from werkzeug.security import generate\_password\_hash, check\_password\_hash**

**app = Flask(\_\_name\_\_)**

**app.secret\_key = 'your\_secret\_key' # Замените на безопасный ключ**

**# Проверка прав администратора**

**def is\_admin():**

**if 'user\_id' in session:**

**conn = get\_db\_connection()**

**cur = conn.cursor(dictionary=True)**

**cur.execute("SELECT is\_admin FROM users WHERE id = %s", (session['user\_id'],))**

**user = cur.fetchone()**

**cur.close()**

**conn.close()**

**if user:**

**print(f"Пользователь найден: is\_admin = {user['is\_admin']}")**

**return user['is\_admin'] == 1**

**else:**

**print("Пользователь не найден в базе данных")**

**return False**

**return False**

**# Контекстный процессор для шаблонов**

**@app.context\_processor**

**def utility\_processor():**

**return dict(is\_admin=is\_admin)**

**# Маршрут админ-панели**

**@app.route('/admin', methods=['GET'])**

**def admin\_panel():**

**if not is\_admin():**

**flash('У вас нет прав для доступа к этой странице')**

**return redirect(url\_for('task\_list'))**

**conn = get\_db\_connection()**

**cur = conn.cursor(dictionary=True)**

**cur.execute("SELECT id, login, password, is\_admin, role FROM users")**

**users = cur.fetchall()**

**cur.close()**

**conn.close()**

**return render\_template('admin\_panel.html', users=users)**

**# Добавление пользователя**

**@app.route('/add\_user', methods=['POST'])**

**def add\_user():**

**if not is\_admin():**

**flash('У вас нет прав для доступа к этой странице')**

**return redirect(url\_for('task\_list'))**

**data = request.get\_json()**

**if data and 'username' in data and 'password' in data:**

**username = data['username']**

**password = data['password']**

**admin\_status = data.get('is\_admin', 0)**

**conn = get\_db\_connection()**

**cur = conn.cursor()**

**cur.execute("INSERT INTO users (login, password, is\_admin, role) VALUES (%s, %s, %s, %s)",**

**(username, password, admin\_status, 'user'))**

**conn.commit()**

**cur.close()**

**conn.close()**

**return jsonify({'status': 'success'}), 200**

**else:**

**return jsonify({'status': 'error', 'message': 'Неверные данные'}), 400**

**# Редактирование пользователя**

**@app.route('/edit\_user/<int:user\_id>', methods=['GET', 'POST'])**

**def edit\_user(user\_id):**

**if not is\_admin():**

**flash('У вас нет прав для доступа к этой странице')**

**return redirect(url\_for('task\_list'))**

**conn = get\_db\_connection()**

**cur = conn.cursor(dictionary=True)**

**if request.method == 'POST':**

**data = request.get\_json()**

**if data and 'username' in data and 'is\_admin' in data:**

**username = data['username']**

**is\_admin\_status = data['is\_admin'] # Переименованная локальная переменная**

**password = data.get('password')**

**if password:**

**cur.execute("UPDATE users SET login = %s, password = %s, is\_admin = %s WHERE id = %s",**

**(username, password, is\_admin\_status, user\_id))**

**else:**

**cur.execute("UPDATE users SET login = %s, is\_admin = %s WHERE id = %s",**

**(username, is\_admin\_status, user\_id))**

**conn.commit()**

**return jsonify({'status': 'success'}), 200**

**else:**

**return jsonify({'status': 'error', 'message': 'Неверные данные'}), 400**

**cur.execute("SELECT id, login, is\_admin FROM users WHERE id = %s", (user\_id,))**

**user = cur.fetchone()**

**cur.close()**

**conn.close()**

**if user:**

**return render\_template('edit\_user.html', user=user)**

**flash('Пользователь не найден')**

**return redirect(url\_for('admin\_panel'))**

**# Удаление пользователя**

**@app.route('/delete\_user/<int:user\_id>', methods=['POST'])**

**def delete\_user(user\_id):**

**if not is\_admin():**

**flash('У вас нет прав для доступа к этой странице')**

**return redirect(url\_for('task\_list'))**

**conn = get\_db\_connection()**

**cur = conn.cursor()**

**cur.execute("DELETE FROM users WHERE id = %s", (user\_id,))**

**conn.commit()**

**cur.close()**

**conn.close()**

**flash('Пользователь успешно удален')**

**return redirect(url\_for('admin\_panel'))**

**# Функция подключения к базе данных**

**def get\_db\_connection():**

**conn = mysql.connector.connect(**

**host=os.getenv('DB\_HOST'),**

**user=os.getenv('DB\_USER'),**

**password=os.getenv('DB\_PASSWORD'),**

**database=os.getenv('DB\_NAME'),**

**charset='utf8mb4',**

**collation='utf8mb4\_unicode\_ci'**

**)**

**return conn**

**# Главная страница**

**@app.route('/')**

**def home():**

**if 'user\_id' in session:**

**return redirect(url\_for('task\_list'))**

**return redirect(url\_for('login'))**

**# Страница входа**

**@app.route('/login', methods=['GET', 'POST'])**

**def login():**

**if request.method == 'POST':**

**username = request.form['username']**

**password = request.form['password']**

**print(f"Попытка входа: {username}")**

**conn = get\_db\_connection()**

**cur = conn.cursor(dictionary=True)**

**cur.execute("SELECT \* FROM users WHERE login = %s", (username,))**

**user = cur.fetchone()**

**if user:**

**print(f"Пользователь найден: {user['login']}")**

**if user['password'] == password:**

**print("Пароль верен")**

**session['user\_id'] = user['id']**

**print(f"Сессия установлена: user\_id = {session['user\_id']}")**

**return redirect(url\_for('task\_list'))**

**else:**

**print("Неверный пароль")**

**flash('Неверное имя пользователя или пароль')**

**else:**

**print("Пользователь не найден")**

**flash('Неверное имя пользователя или пароль')**

**cur.close()**

**conn.close()**

**return render\_template('login.html')**

**# Страница регистрации**

**@app.route('/register', methods=['GET', 'POST'])**

**def register():**

**if request.method == 'POST':**

**username = request.form['username']**

**password = request.form['password']**

**print(f"Попытка регистрации: {username}")**

**conn = get\_db\_connection()**

**cur = conn.cursor()**

**cur.execute("SELECT \* FROM users WHERE login = %s", (username,))**

**if cur.fetchone():**

**print("Пользователь уже существует")**

**flash('Имя пользователя уже занято')**

**else:**

**cur.execute("INSERT INTO users (login, password, is\_admin, role) VALUES (%s, %s, %s, %s)",**

**(username, password, 0, 'user'))**

**conn.commit()**

**print("Пользователь успешно зарегистрирован")**

**flash('Регистрация прошла успешно, войдите в систему')**

**return redirect(url\_for('login'))**

**cur.close()**

**conn.close()**

**return render\_template('register.html')**

**# Выход из системы**

**@app.route('/logout')**

**def logout():**

**session.pop('user\_id', None)**

**flash('Вы вышли из системы')**

**return redirect(url\_for('login'))**

**# Список задач**

**@app.route('/tasks')**

**def task\_list():**

**if 'user\_id' not in session:**

**return redirect(url\_for('login'))**

**user\_id = session['user\_id']**

**conn = get\_db\_connection()**

**cur = conn.cursor(dictionary=True)**

**cur.execute("SELECT \* FROM items WHERE user\_id = %s ORDER BY order\_index", (user\_id,))**

**tasks = cur.fetchall()**

**cur.close()**

**conn.close()**

**return render\_template('task\_list.html', tasks=tasks)**

**# Добавление задачи**

**@app.route('/add\_task', methods=['POST'])**

**def add\_task():**

**if 'user\_id' not in session:**

**return redirect(url\_for('login'))**

**data = request.get\_json()**

**if data and 'text' in data:**

**text = data['text']**

**user\_id = session['user\_id']**

**conn = get\_db\_connection()**

**cur = conn.cursor()**

**cur.execute("SELECT MAX(order\_index) FROM items WHERE user\_id = %s", (user\_id,))**

**max\_index = cur.fetchone()[0]**

**new\_index = 1 if max\_index is None else max\_index + 1**

**cur.execute("INSERT INTO items (text, user\_id, order\_index) VALUES (%s, %s, %s)",**

**(text, user\_id, new\_index))**

**conn.commit()**

**cur.close()**

**conn.close()**

**return jsonify({'status': 'success'}), 200**

**else:**

**return jsonify({'status': 'error', 'message': 'Неверные данные'}), 400**

**# Редактирование задачи**

**@app.route('/edit\_task/<int:task\_id>', methods=['GET', 'POST'])**

**def edit\_task(task\_id):**

**if 'user\_id' not in session:**

**return redirect(url\_for('login'))**

**user\_id = session['user\_id']**

**conn = get\_db\_connection()**

**cur = conn.cursor(dictionary=True)**

**if request.method == 'POST':**

**text = request.form['text']**

**cur.execute("UPDATE items SET text = %s WHERE id = %s AND user\_id = %s",**

**(text, task\_id, user\_id))**

**conn.commit()**

**if cur.rowcount > 0:**

**flash('Задача успешно обновлена')**

**else:**

**flash('Задача не найдена или у вас нет прав')**

**cur.close()**

**conn.close()**

**return redirect(url\_for('task\_list'))**

**cur.execute("SELECT \* FROM items WHERE id = %s AND user\_id = %s", (task\_id, user\_id))**

**task = cur.fetchone()**

**cur.close()**

**conn.close()**

**if task:**

**return render\_template('edit\_task.html', task=task)**

**flash('Задача не найдена или у вас нет прав')**

**return redirect(url\_for('task\_list'))**

**# Удаление задачи**

**@app.route('/delete\_task/<int:task\_id>', methods=['POST'])**

**def delete\_task(task\_id):**

**if 'user\_id' not in session:**

**return redirect(url\_for('login'))**

**user\_id = session['user\_id']**

**conn = get\_db\_connection()**

**cur = conn.cursor()**

**cur.execute("DELETE FROM items WHERE id = %s AND user\_id = %s", (task\_id, user\_id))**

**conn.commit()**

**if cur.rowcount > 0:**

**flash('Задача успешно удалена')**

**else:**

**flash('Задача не найдена или у вас нет прав')**

**cur.close()**

**conn.close()**

**return redirect(url\_for('task\_list'))**

**# Реорганизация задач**

**@app.route('/reorder\_tasks', methods=['POST'])**

**def reorder\_tasks():**

**if 'user\_id' not in session:**

**return redirect(url\_for('login'))**

**data = request.get\_json()**

**if data and 'task\_ids' in data:**

**task\_ids = data['task\_ids']**

**conn = get\_db\_connection()**

**cur = conn.cursor()**

**for index, task\_id in enumerate(task\_ids):**

**cur.execute("UPDATE items SET order\_index = %s WHERE id = %s AND user\_id = %s",**

**(index + 1, task\_id, session['user\_id']))**

**conn.commit()**

**cur.close()**

**conn.close()**

**return jsonify({'status': 'success'}), 200**

**else:**

**return jsonify({'status': 'error', 'message': 'Неверные данные'}), 400**

**base.html**

**<!DOCTYPE html>**

**<html lang="ru">**

**<head>**

**<meta charset="UTF-8">**

**<title>Планировщик задач</title>**

**<link rel="stylesheet" href="{{ url\_for('static', filename='css/style.css') }}">**

**<style>**

**/\* Стили для всплывающих оповещений \*/**

**.flash-message {**

**position: fixed;**

**top: 20px;**

**right: 20px;**

**background-color: #4CAF50;**

**color: white;**

**padding: 10px 20px;**

**border-radius: 5px;**

**box-shadow: 0 0 10px rgba(0,0,0,0.5);**

**opacity: 1;**

**transition: opacity 0.5s ease-in-out;**

**z-index: 1000;**

**}**

**.flash-message.hide {**

**opacity: 0;**

**}**

**</style>**

**</head>**

**<body>**

**<div class="container">**

**<!-- Контейнер для всплывающих оповещений -->**

**{% with messages = get\_flashed\_messages() %}**

**{% if messages %}**

**{% for message in messages %}**

**<div class="flash-message">{{ message }}</div>**

**{% endfor %}**

**{% endif %}**

**{% endwith %}**

**{% block content %}{% endblock %}**

**</div>**

**<script>**

**// Скрытие оповещений через 3 секунды**

**document.addEventListener('DOMContentLoaded', function() {**

**const flashes = document.querySelectorAll('.flash-message');**

**flashes.forEach(function(flash) {**

**setTimeout(function() {**

**flash.classList.add('hide');**

**}, 3000);**

**// Удаление элемента после анимации**

**flash.addEventListener('transitionend', function() {**

**flash.remove();**

**});**

**});**

**});**

**</script>**

**</body>**

**</html>**

**login.html**

**{% extends "base.html" %}**

**{% block content %}**

**<h1>Вход</h1>**

**<form method="POST" action="/login">**

**<input type="text" name="username" placeholder="Имя пользователя" required>**

**<input type="password" name="password" placeholder="Пароль" required>**

**<button type="submit">Войти</button>**

**</form>**

**<a href="{{ url\_for('register') }}">Регистрация</a>**

**{% endblock %}**

**task\_list.html**

**{% extends "base.html" %}**

**{% block content %}**

**<h1>Ваши задачи</h1>**

**<table>**

**<thead>**

**<tr>**

**<th>Задача</th>**

**<th>Действия</th>**

**</tr>**

**</thead>**

**<tbody id="task-list">**

**{% for task in tasks %}**

**<tr data-id="{{ task.id }}">**

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

**<td>**

**<button onclick="openEditTaskModal('{{ task.id }}', '{{ task.text }}')">Редактировать</button>**

**<form method="post" action="{{ url\_for('delete\_task', task\_id=task.id) }}" style="display:inline">**

**<button type="submit">Удалить</button>**

**</form>**

**<button onclick="moveTaskUp('{{ task.id }}')">↑</button>**

**<button onclick="moveTaskDown('{{ task.id }}')">↓</button>**

**</td>**

**</tr>**

**{% endfor %}**

**</tbody>**

**</table>**

**{% if is\_admin() %}**

**<a href="{{ url\_for('admin\_panel') }}">Админ-панель</a>**

**{% endif %}**

**<button onclick="openAddTaskModal()">Добавить задачу</button>**

**<a href="{{ url\_for('logout') }}">Выйти</a>**

**<!-- Модальное окно для добавления задачи -->**

**<div id="addTaskModal" style="display:none; position: fixed; top: 50%; left: 50%; transform: translate(-50%, -50%); background: white; padding: 20px; border: 1px solid #ccc; box-shadow: 0 0 10px rgba(0,0,0,0.5);">**

**<div>**

**<h2>Добавить задачу</h2>**

**<form id="addTaskForm">**

**<input type="text" id="addTaskText" name="text" placeholder="Текст задачи" required>**

**<button type="button" onclick="addTask()">Добавить</button>**

**<button type="button" onclick="closeAddTaskModal()">Отмена</button>**

**</form>**

**</div>**

**</div>**

**<!-- Модальное окно для редактирования задачи -->**

**<div id="editTaskModal" style="display:none; position: fixed; top: 50%; left: 50%; transform: translate(-50%, -50%); background: white; padding: 20px; border: 1px solid #ccc; box-shadow: 0 0 10px rgba(0,0,0,0.5);">**

**<div>**

**<h2>Редактировать задачу</h2>**

**<form id="editTaskForm">**

**<input type="hidden" id="taskId" name="task\_id">**

**<input type="text" id="editTaskText" name="text" required>**

**<button type="button" onclick="saveTask()">Сохранить</button>**

**<button type="button" onclick="closeEditTaskModal()">Отмена</button>**

**</form>**

**</div>**

**</div>**

**<script>**

**function openAddTaskModal() {**

**document.getElementById('addTaskModal').style.display = 'block';**

**}**

**function closeAddTaskModal() {**

**document.getElementById('addTaskModal').style.display = 'none';**

**}**

**function addTask() {**

**var taskText = document.getElementById('addTaskText').value;**

**fetch('/add\_task', {**

**method: 'POST',**

**headers: {**

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

**},**

**body: JSON.stringify({ text: taskText })**

**}).then(response => {**

**if (response.ok) {**

**location.reload();**

**} else {**

**alert('Ошибка при добавлении задачи');**

**}**

**});**

**}**

**function openEditTaskModal(taskId, taskText) {**

**document.getElementById('taskId').value = taskId;**

**document.getElementById('editTaskText').value = taskText;**

**document.getElementById('editTaskModal').style.display = 'block';**

**}**

**function closeEditTaskModal() {**

**document.getElementById('editTaskModal').style.display = 'none';**

**}**

**function saveTask() {**

**var taskId = document.getElementById('taskId').value;**

**var taskText = document.getElementById('editTaskText').value;**

**fetch('/edit\_task/' + taskId, {**

**method: 'POST',**

**headers: {**

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

**},**

**body: JSON.stringify({ text: taskText })**

**}).then(response => {**

**if (response.ok) {**

**location.reload();**

**} else {**

**alert('Ошибка при сохранении задачи');**

**}**

**});**

**}**

**// Функции для перемещения задач**

**function moveTaskUp(taskId) {**

**const taskList = document.getElementById('task-list');**

**const taskRow = taskList.querySelector(`tr[data-id="${taskId}"]`);**

**const prevRow = taskRow.previousElementSibling;**

**if (prevRow) {**

**taskList.insertBefore(taskRow, prevRow);**

**saveOrder();**

**}**

**}**

**function moveTaskDown(taskId) {**

**const taskList = document.getElementById('task-list');**

**const taskRow = taskList.querySelector(`tr[data-id="${taskId}"]`);**

**const nextRow = taskRow.nextElementSibling;**

**if (nextRow) {**

**taskList.insertBefore(nextRow, taskRow);**

**saveOrder();**

**}**

**}**

**// Сохранение порядка задач**

**function saveOrder() {**

**const taskIds = Array.from(document.querySelectorAll('#task-list tr')).map(tr => tr.dataset.id);**

**fetch('/reorder\_tasks', {**

**method: 'POST',**

**headers: {**

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

**},**

**body: JSON.stringify({ task\_ids: taskIds })**

**}).then(response => {**

**if (!response.ok) {**

**alert('Ошибка при обновлении порядка задач');**

**}**

**});**

**}**

**</script>**

**{% endblock %}**

**register.html**

**{% extends "base.html" %}**

**{% block content %}**

**<h1>Регистрация</h1>**

**<form method="POST" action="/register">**

**<input type="text" name="username" placeholder="Имя пользователя" required>**

**<input type="password" name="password" placeholder="Пароль" required>**

**<button type="submit">Зарегистрироваться</button>**

**</form>**

**{% endblock %}**

**admin\_panel.html**

**{% extends "base.html" %}**

**{% block content %}**

**<h1>Админ-панель</h1>**

**<table border="1">**

**<thead>**

**<tr>**

**<th>ID</th>**

**<th>Логин</th>**

**<th>Пароль (хеш)</th>**

**<th>Администратор</th>**

**<th>Действия</th>**

**</tr>**

**</thead>**

**<tbody>**

**{% for user in users %}**

**<tr>**

**<td>{{ user.id }}</td>**

**<td>{{ user.login }}</td>**

**<td>{{ user.password }}</td>**

**<td>{{ 'Да' if user.is\_admin else 'Нет' }}</td>**

**<td>**

**<button onclick="openEditUserModal('{{ user.id }}', '{{ user.login }}', '{{ user.is\_admin }}')">Редактировать</button>**

**<form method="post" action="{{ url\_for('delete\_user', user\_id=user.id) }}" style="display:inline;">**

**<button type="submit">Удалить</button>**

**</form>**

**</td>**

**</tr>**

**{% endfor %}**

**</tbody>**

**</table>**

**<button onclick="openAddUserModal()">Добавить пользователя</button>**

**<!-- Кнопка для возврата к списку задач -->**

**<a href="{{ url\_for('task\_list') }}" class="button">Вернуться к задачам</a>**

**<!-- Модальное окно для добавления пользователя -->**

**<div id="addUserModal" style="display:none; position: fixed; top: 50%; left: 50%; transform: translate(-50%, -50%); background: white; padding: 20px; border: 1px solid #ccc; box-shadow: 0 0 10px rgba(0,0,0,0.5);">**

**<div>**

**<h2>Добавить пользователя</h2>**

**<form id="addUserForm">**

**<input type="text" id="username" name="username" placeholder="Логин" required>**

**<input type="password" id="password" name="password" placeholder="Пароль" required>**

**<label>**

**<input type="checkbox" id="isAdmin" name="is\_admin"> Администратор**

**</label>**

**<button type="button" onclick="addUser()">Добавить</button>**

**<button type="button" onclick="closeAddUserModal()">Отмена</button>**

**</form>**

**</div>**

**</div>**

**<!-- Модальное окно для редактирования пользователя -->**

**<div id="editUserModal" style="display:none; position: fixed; top: 50%; left: 50%; transform: translate(-50%, -50%); background: white; padding: 20px; border: 1px solid #ccc; box-shadow: 0 0 10px rgba(0,0,0,0.5);">**

**<div>**

**<h2>Редактировать пользователя</h2>**

**<form id="editUserForm">**

**<input type="hidden" id="userId" name="user\_id">**

**<input type="text" id="editUsername" name="username" placeholder="Логин" required>**

**<input type="password" id="editPassword" name="password" placeholder="Новый пароль">**

**<label>**

**<input type="checkbox" id="editIsAdmin" name="is\_admin"> Администратор**

**</label>**

**<button type="button" onclick="saveUser()">Сохранить</button>**

**<button type="button" onclick="closeEditUserModal()">Отмена</button>**

**</form>**

**</div>**

**</div>**

**<script>**

**function openAddUserModal() {**

**document.getElementById('addUserModal').style.display = 'block';**

**}**

**function closeAddUserModal() {**

**document.getElementById('addUserModal').style.display = 'none';**

**}**

**function addUser() {**

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

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

**var isAdmin = document.getElementById('isAdmin').checked ? 1 : 0;**

**fetch('/add\_user', {**

**method: 'POST',**

**headers: {**

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

**},**

**body: JSON.stringify({ username: username, password: password, is\_admin: isAdmin })**

**}).then(response => {**

**if (response.ok) {**

**location.reload();**

**} else {**

**alert('Ошибка при добавлении пользователя');**

**}**

**});**

**}**

**function openEditUserModal(userId, username, isAdmin) {**

**document.getElementById('userId').value = userId;**

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

**document.getElementById('editIsAdmin').checked = isAdmin == '1';**

**document.getElementById('editUserModal').style.display = 'block';**

**}**

**function closeEditUserModal() {**

**document.getElementById('editUserModal').style.display = 'none';**

**}**

**function saveUser() {**

**var userId = document.getElementById('userId').value;**

**var username = document.getElementById('editUsername').value;**

**var password = document.getElementById('editPassword').value;**

**var isAdmin = document.getElementById('editIsAdmin').checked ? 1 : 0;**

**fetch('/edit\_user/' + userId, {**

**method: 'POST',**

**headers: {**

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

**},**

**body: JSON.stringify({ username: username, password: password, is\_admin: isAdmin })**

**}).then(response => {**

**if (response.ok) {**

**location.reload();**

**} else {**

**alert('Ошибка при сохранении пользователя');**

**}**

**});**

**}**

**</script>**

**{% endblock %}**

**style.css**

**/\* Общие стили для страницы \*/**

**body {**

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

**background-image: url('https://images.unsplash.com/photo-1506748686214-e9df14d4d9d0?ixlib=rb-4.0.3&auto=format&fit=crop&w=1350&q=80'); /\* Красивое фоновое изображение \*/**

**background-size: cover;**

**background-position: center;**

**background-attachment: fixed;**

**margin: 0;**

**padding: 20px;**

**color: #333;**

**}**

**/\* Контейнер для содержимого \*/**

**.container {**

**max-width: 800px;**

**margin: 0 auto;**

**background-color: rgba(255, 255, 255, 0.85); /\* Полупрозрачный белый фон \*/**

**padding: 20px;**

**border-radius: 8px;**

**box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);**

**}**

**/\* Стили для заголовка \*/**

**h1 {**

**text-align: center;**

**color: #4CAF50;**

**margin-bottom: 20px;**

**}**

**/\* Стили для таблицы \*/**

**table {**

**width: 100%;**

**border-collapse: collapse;**

**margin-bottom: 20px;**

**background-color: #fff;**

**border-radius: 8px;**

**overflow: hidden;**

**}**

**th, td {**

**padding: 12px;**

**text-align: left;**

**border-bottom: 1px solid #ddd;**

**}**

**th {**

**background-color: #f2f2f2;**

**color: #333;**

**font-weight: bold;**

**}**

**tr:hover {**

**background-color: #f5f5f5;**

**}**

**/\* Стили для кнопок \*/**

**button {**

**padding: 6px 12px;**

**margin: 0 5px;**

**border: none;**

**border-radius: 4px;**

**cursor: pointer;**

**font-size: 14px;**

**}**

**.edit-btn {**

**background-color: #2196F3;**

**color: white;**

**}**

**.delete-btn {**

**background-color: #f44336;**

**color: white;**

**}**

**button:hover {**

**opacity: 0.8;**

**}**

**/\* Стили для формы \*/**

**form {**

**display: flex;**

**justify-content: center;**

**align-items: center;**

**margin-top: 20px;**

**}**

**input[type="text"] {**

**width: 70%;**

**padding: 10px;**

**border: 1px solid #ccc;**

**border-radius: 4px;**

**margin-right: 10px;**

**font-size: 14px;**

**}**

**button[type="submit"] {**

**padding: 10px 20px;**

**background-color: #4CAF50;**

**color: white;**

**border: none;**

**border-radius: 4px;**

**cursor: pointer;**

**font-size: 14px;**

**}**

**button[type="submit"]:hover {**

**background-color: #45a049;**

**}**

**/\* Адаптивность \*/**

**@media (max-width: 600px) {**

**table, form {**

**width: 100%;**

**}**

**input[type="text"] {**

**width: 60%;**

**}**

**button {**

**font-size: 12px;**

**padding: 8px 16px;**

**}**

**}**

3. код бота

Бот реализован с использованием библиотеки python-telegram-bot и предоставляет функции авторизации, добавления, удаления и просмотра задач. Все команды интегрированы с базой данных, которая общая с сайтом.

**bottg.py**

**import os**

**import mysql.connector**

**from telegram import Update, BotCommand**

**from telegram.ext import Application, CommandHandler, ContextTypes**

**from aiohttp import web**

**import asyncio**

**# Настройка токена и конфигурации базы данных**

**TOKEN = '7671395940:AAHwqDqy-PD8OfhFdjvCIjTE2u2yQ2yZ7wo'**

**db\_config = {**

**'host': 'sql7.freesqldatabase.com',**

**'user': 'sql7784455',**

**'password': 'xxB1ERVxEi',**

**'database': 'sql7784455',**

**'port': 3306**

**}**

**session\_token = None # Хранит ID пользователя после авторизации**

**# Функция для подключения к БД**

**def get\_db\_connection():**

**return mysql.connector.connect(\*\*db\_config)**

**# Команда /login - вход в систему**

**async def login(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> None:**

**global session\_token**

**if len(context.args) != 2:**

**await update.message.reply\_text('Использование: /login <login> <password>')**

**return**

**login, password = context.args**

**try:**

**conn = get\_db\_connection()**

**cursor = conn.cursor(dictionary=True)**

**cursor.execute("SELECT \* FROM users WHERE login = %s AND password = %s", (login, password))**

**user = cursor.fetchone()**

**cursor.close()**

**conn.close()**

**if user:**

**session\_token = user['id']**

**await update.message.reply\_text('Авторизация успешна.')**

**else:**

**await update.message.reply\_text('Ошибка авторизации. Неверный логин или пароль.')**

**except Exception as e:**

**await update.message.reply\_text(f'Ошибка: {str(e)}')**

**# Команда /logout - выход из учетной записи**

**async def logout(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> None:**

**global session\_token**

**session\_token = None**

**await update.message.reply\_text('Вы успешно вышли из учетной записи.')**

**# Команда /showcom - показать доступные команды**

**async def showcom(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> None:**

**commands = """**

**Доступные команды:**

**- /login <login> <password> - Войти в систему**

**- /logout - Выйти из учетной записи**

**- /list - Показать список задач**

**- /add <текст> - Добавить задачу**

**- /delete <id> - Удалить задачу по ID**

**- /showcom - Показать доступные команды**

**"""**

**await update.message.reply\_text(commands)**

**# Команда /list - показать список задач**

**async def list\_tasks(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> None:**

**global session\_token**

**if not session\_token:**

**await update.message.reply\_text('Ошибка: вы не авторизованы. Используйте /login <login> <password>.')**

**return**

**try:**

**conn = get\_db\_connection()**

**cursor = conn.cursor(dictionary=True)**

**cursor.execute("SELECT id, text FROM items WHERE user\_id = %s", (session\_token,))**

**items = cursor.fetchall()**

**cursor.close()**

**conn.close()**

**if items:**

**message = '\n'.join([f"{index + 1} / {item['text']} / [{item['id']}]" for index, item in enumerate(items)])**

**else:**

**message = 'Список пуст.'**

**await update.message.reply\_text(message)**

**except Exception as e:**

**await update.message.reply\_text(f'Ошибка: {str(e)}')**

**# Команда /add - добавить задачу**

**async def add\_task(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> None:**

**global session\_token**

**if not session\_token:**

**await update.message.reply\_text('Ошибка: вы не авторизованы. Используйте /login <login> <password>.')**

**return**

**if len(context.args) < 1:**

**await update.message.reply\_text('Использование: /add <текст>')**

**return**

**text = ' '.join(context.args)**

**try:**

**conn = get\_db\_connection()**

**cursor = conn.cursor()**

**cursor.execute("INSERT INTO items (text, user\_id) VALUES (%s, %s)", (text, session\_token))**

**conn.commit()**

**cursor.close()**

**conn.close()**

**await update.message.reply\_text('Задача добавлена.')**

**except Exception as e:**

**await update.message.reply\_text(f'Ошибка: {str(e)}')**

**# Команда /delete - удалить задачу по ID**

**async def delete\_task(update: Update, context: ContextTypes.DEFAULT\_TYPE) -> None:**

**global session\_token**

**if not session\_token:**

**await update.message.reply\_text('Ошибка: вы не авторизованы. Используйте /login <login> <password>.')**

**return**

**if len(context.args) != 1:**

**await update.message.reply\_text('Использование: /delete <id>')**

**return**

**task\_id = context.args[0]**

**try:**

**conn = get\_db\_connection()**

**cursor = conn.cursor()**

**cursor.execute("DELETE FROM items WHERE id = %s AND user\_id = %s", (task\_id, session\_token))**

**conn.commit()**

**cursor.close()**

**conn.close()**

**await update.message.reply\_text('Задача удалена.')**

**except Exception as e:**

**await update.message.reply\_text(f'Ошибка: {str(e)}')**

**# Установка команд в меню Telegram**

**async def set\_commands(application):**

**commands = [**

**BotCommand("login", "Войти в систему: /login <login> <password>"),**

**BotCommand("logout", "Выйти из учетной записи"),**

**BotCommand("list", "Показать список задач"),**

**BotCommand("add", "Добавить задачу: /add <текст>"),**

**BotCommand("delete", "Удалить задачу: /delete <id>"),**

**BotCommand("showcom", "Показать доступные команды")**

**]**

**await application.bot.set\_my\_commands(commands)**

**# Простой HTTP-сервер для Render.com**

**async def run\_http\_server():**

**app = web.Application()**

**app.add\_routes([web.get('/', lambda request: web.Response(text="Bot is running"))])**

**runner = web.AppRunner(app)**

**await runner.setup()**

**site = web.TCPSite(runner, '0.0.0.0', int(os.getenv("PORT", 8080))) # Используем PORT от Render**

**await site.start()**

**print("HTTP-сервер запущен на порту", os.getenv("PORT", 8080))**

**# Основная функция для запуска бота и HTTP-сервера**

**async def main():**

**# Создаем приложение Telegram**

**application = Application.builder().token(TOKEN).build()**

**# Добавление обработчиков команд**

**application.add\_handler(CommandHandler("login", login))**

**application.add\_handler(CommandHandler("logout", logout))**

**application.add\_handler(CommandHandler("showcom", showcom))**

**application.add\_handler(CommandHandler("list", list\_tasks))**

**application.add\_handler(CommandHandler("add", add\_task))**

**application.add\_handler(CommandHandler("delete", delete\_task))**

**# Установка команд в меню**

**await set\_commands(application)**

**# Запуск HTTP-сервера**

**await run\_http\_server()**

**# Инициализация и запуск polling'а**

**await application.initialize()**

**await application.start()**

**await application.updater.start\_polling()**

**# Бесконечный цикл для удержания приложения**

**while True:**

**await asyncio.sleep(3600) # Спим 1 час, чтобы не нагружать CPU**

**# Запуск приложения**

**if \_\_name\_\_ == '\_\_main\_\_':**

**loop = asyncio.get\_event\_loop()**

**try:**

**loop.run\_until\_complete(main())**

**except KeyboardInterrupt:**

**loop.run\_until\_complete(application.updater.stop())**

**loop.run\_until\_complete(application.stop())**

**loop.close()**

**requirements.txt**

**python-telegram-bot**

**requests**

**mysql-connector-python**

**aiohttp**

4. структура базы данных

База данных состоит из двух таблиц: users (пользователи) и items (задачи). Предусмотрены внешние ключи, уникальные логины, роли пользователей, индексы сортировки задач.

**CREATE DATABASE IF NOT EXISTS todolist;**

**USE todolist;**

**CREATE TABLE IF NOT EXISTS items (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**text VARCHAR(255) NOT NULL,**

**user\_id INT NOT NULL,**

**CONSTRAINT fk\_user FOREIGN KEY (user\_id) REFERENCES users(id)**

**);**

**CREATE TABLE IF NOT EXISTS users (**

**id INT AUTO\_INCREMENT PRIMARY KEY,**

**login VARCHAR(50) NOT NULL UNIQUE,**

**password VARCHAR(255) NOT NULL,**

**is\_admin TINYINT(1) NOT NULL DEFAULT 0,**

**role VARCHAR(50) NOT NULL DEFAULT 'user',**

**token VARCHAR(255) DEFAULT NULL**

**);**

**INSERT INTO users (login, password, is\_admin, role) VALUES ('admin', 'password123', 1, 'admin');**

**-- Добавляем столбец order\_index**

**ALTER TABLE items ADD COLUMN order\_index INT NOT NULL DEFAULT 0;**

**-- Задаём начальные значения для существующих записей (например, равные ID)**

**UPDATE items SET order\_index = id;**

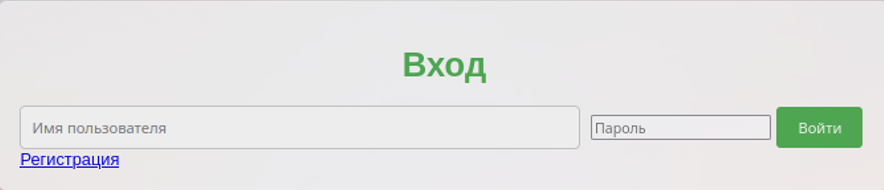
**ALTER DATABASE todolist CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;**

**ALTER TABLE items CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;**

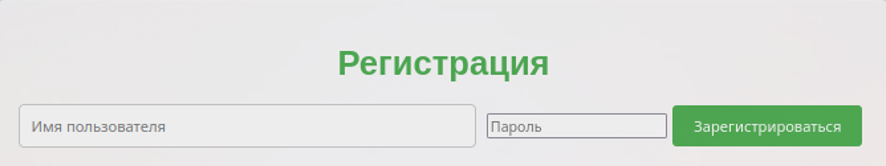
**ALTER TABLE users CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;**

****5. Обзор ОСНОВНЫХ ФУНКЦИЙ сайта****

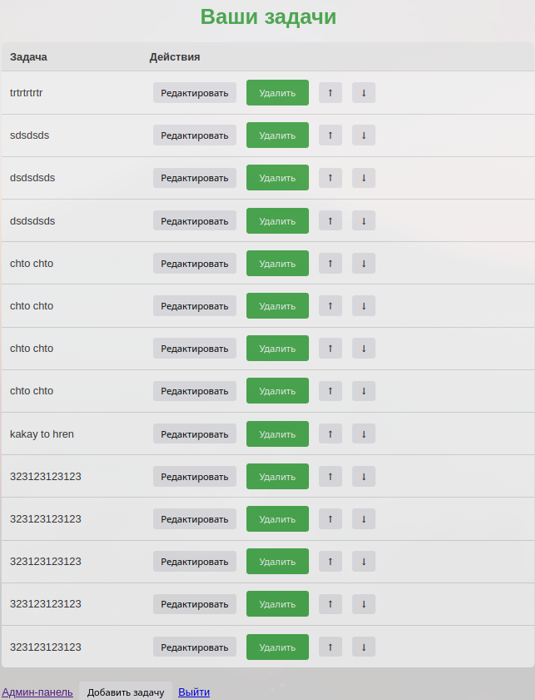
Интерфейс включает:  
- авторизацию пользователя (login.html);  
- регистрацию нового пользователя;  
- отображение списка задач;  
- возможность добавления, удаления, редактирования задач;  
- интерфейс для администратора (админ-панель).



**Рис. 1. Окно авторизации**



**Рис. 2. Окно регистрации пользователя**



**Рис. 3. Страница списка задач**

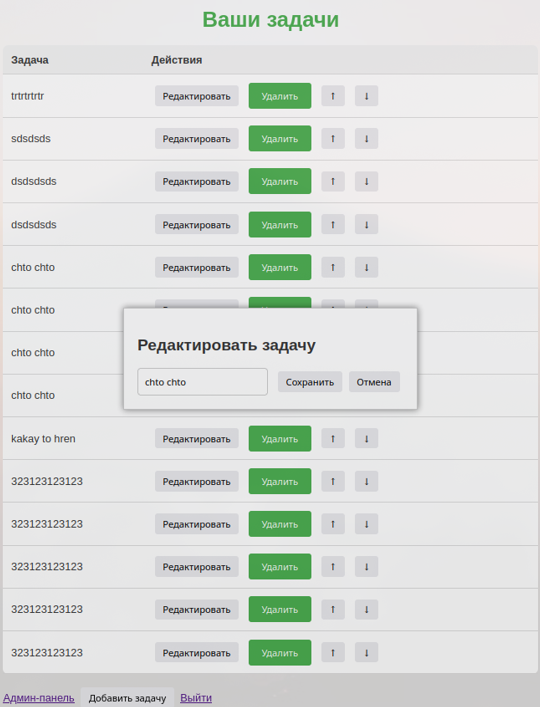


Рис. 4. Редактирование задач

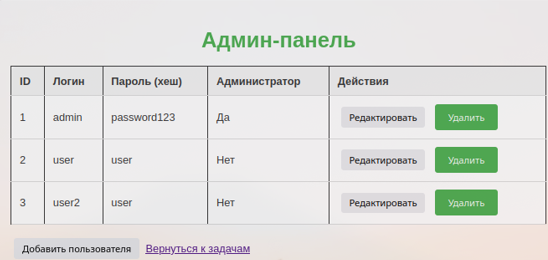


Рис. 5. Страница админ-панели

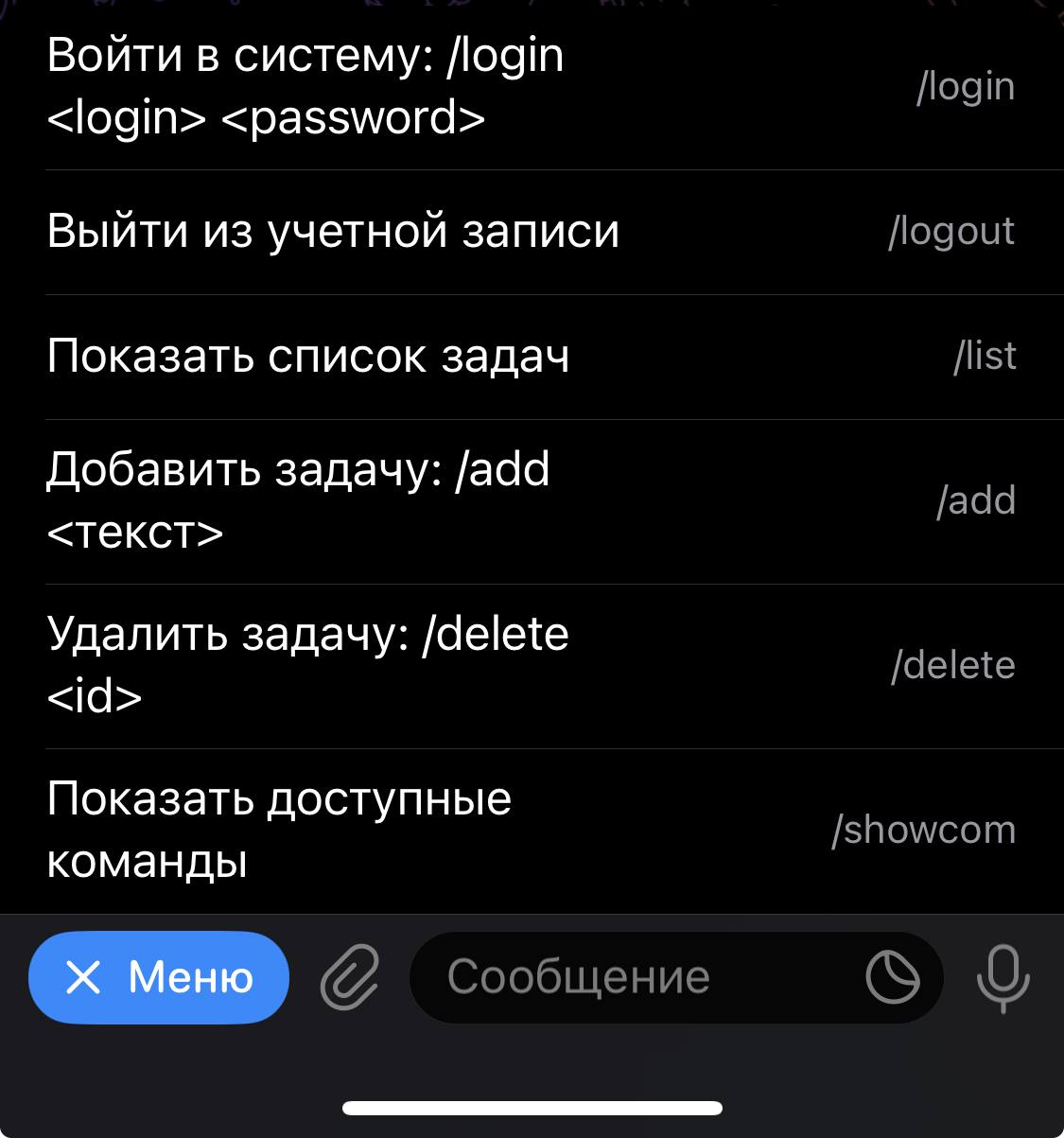


Рис. 6. Обзор функций бота

****6. ФОРМУЛИРОВКИ ЗАПРОСОВ ДЛЯ ГЕНЕРАЦИИ КОДА****

В процессе реализации проекта использовалась генерация кода с помощью больших языковых моделей. Для этого были разработаны последовательные и точные формулировки задач (prompt-запросы), включавшие:

* Для Telegram-бота:
* - Авторизация через /login с сохранением ID пользователя.
* - Завершение сессии через /logout.
* - Вывод списка команд через /showcom.
* - Получение и вывод задач через /list.
* - Добавление задач через /add.
* - Удаление задач через /delete.
* Для веб-приложения:
* “Я пишу Todolist на nodejs, база данных через mysql. мне нужно, чтобы ты добавил функцию добавления дел в index.js, убери из html <script> (Приложил файлы db.sql, index.html, index.js)”
* - Настройка маршрутов фреймворка на Python, включая авторизацию и регистрацию.
* - Подключение к MySQL, создание таблиц users и items.
* - Оформление шаблонов для задач и админ-панели.
* - Реализация логики добавления, удаления, редактирования задач.
* - Настройка ролей пользователей и административных прав.

****7. ТРУДНОСТИ ПРИ ИСПОЛЬЗОВАНИИ НЕЙРОСЕТИ****

Применение нейросети для генерации кода сопровождается рядом ограничений. Хотя создаваемые фрагменты могут выглядеть корректно, они нередко не охватывают граничные случаи и теряют связь с контекстом проекта. Это затрудняет отладку, поскольку ошибки могут быть неочевидными. Кроме того, чрезмерное полагание на нейросетевые решения может негативно сказаться на обучении — разработчики рискуют упустить важные концептуальные основы. Также сгенерированный код не всегда хорошо интегрируется с уже существующими системами и может содержать уязвимости из-за нарушения стандартов безопасности.

****8. ВЫВОД****

В рамках данной работы была успешно разработана система, включающая веб-приложение и Telegram-бота, объединённых общей СУБД. Использование языковых моделей позволило сгенерировать часть кода и ускорить реализацию. Программа поддерживает регистрацию, работу с задачами и разграничение прав доступа. Все требования были выполнены.