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

ОТЧЕТ  
О ЛАБОРАТОРНОЙ РАБОТЕ

|  |
| --- |
| Лабораторная работа  «Создание собственного онлайн сервиса» |
| по дисциплине: Технологии и методы программирования |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент гр. № | 3331 |  | подпись, дата |  | Ю.И. Ефименкова |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

оглавление

1. **Цель работы 3**
2. код сайта 3
3. код бота 37
4. структура базы данных 42
5. Обзор возможностей 42
6. развертывание сайта и бота 44
7. промпты для получения кода 45
8. трудности при использовании GPT 50
9. вывод 50

1. Цель работы

Целью этой лабораторной работы является создание сайта и ботом с общей базой данных, при помощи использования больших языковых моделей. Данная лабораторная работа предполгает развитие навыков «промптинга» у студентов, пониманию работы базы данных и способов общения и взаимодействия между собой сайта и бота с базой данных. В качестве тематики сайта и бота был выбран планировщик задач с возможностью авторизации.

****2. код сайта****

****login.html****

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

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

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

**<title>Login</title>**

**<style>**

**/\* Стили для всей 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'); /\* Фоновое изображение \*/**

**background-size: cover;**

**background-position: center;**

**background-attachment: fixed;**

**margin: 0;**

**padding: 20px;**

**color: #333;**

**}**

**/\* Стили для контейнера формы \*/**

**.login-container {**

**max-width: 400px;**

**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: #333;**

**}**

**/\* Стили для полей ввода \*/**

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

**width: 100%;**

**padding: 10px;**

**margin: 10px 0;**

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

**border-radius: 4px;**

**box-sizing: border-box;**

**}**

**/\* Стили для кнопки \*/**

**button {**

**width: 100%;**

**padding: 10px;**

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

**color: white;**

**border: none;**

**border-radius: 4px;**

**cursor: pointer;**

**font-size: 16px;**

**}**

**button:hover {**

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

**}**

**/\* Стили для сообщений об ошибках \*/**

**.error {**

**color: red;**

**text-align: center;**

**}**

**/\* Стили для значка просмотра пароля \*/**

**.password-wrapper {**

**position: relative;**

**}**

**.password-wrapper input {**

**padding-right: 30px; /\* Место для значка \*/**

**}**

**.password-wrapper .toggle-password {**

**position: absolute;**

**right: 10px;**

**top: 50%;**

**transform: translateY(-50%);**

**cursor: pointer;**

**user-select: none;**

**}**

**</style>**

**</head>**

**<body>**

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

**<h1>Login</h1>**

**<form id="login-form">**

**<input type="text" id="login-input" placeholder="Login" required>**

**<div class="password-wrapper">**

**<input type="password" id="password-input" placeholder="Password" required>**

**<span class="toggle-password">👁️</span>**

**</div>**

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

**</form>**

**<div class="error" id="error-message"></div>**

**</div>**

**<script>**

**const passwordInput = document.getElementById('password-input');**

**const togglePassword = document.querySelector('.toggle-password');**

**togglePassword.addEventListener('mousedown', () => {**

**passwordInput.type = 'text';**

**});**

**togglePassword.addEventListener('mouseup', () => {**

**passwordInput.type = 'password';**

**});**

**// Также обрабатываем случай, если пользователь убирает курсор с значка**

**togglePassword.addEventListener('mouseleave', () => {**

**passwordInput.type = 'password';**

**});**

**document.getElementById('login-form').addEventListener('submit', async (e) => {**

**e.preventDefault();**

**const login = document.getElementById('login-input').value;**

**const password = document.getElementById('password-input').value;**

**try {**

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

**method: 'POST',**

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

**body: JSON.stringify({ login, password })**

**});**

**const result = await response.json();**

**if (result.success) {**

**window.location.href = '/';**

**} else {**

**document.getElementById('error-message').textContent = 'Неверный логин или пароль';**

**}**

**} catch (error) {**

**console.error('Ошибка:', error);**

**document.getElementById('error-message').textContent = 'Ошибка при авторизации';**

**}**

**});**

**</script>**

**</body>**

**</html>**

****index.html****

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

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

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

**<!-- Подключаем jQuery -->**

**<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>**

**<!-- Подключаем клиентский скрипт -->**

**<script src="/client.js" defer></script>**

**<style>**

**/\* 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;**

**}**

**/\* Стили для таблицы \*/**

**table {**

**width: 100%;**

**border-collapse: collapse;**

**margin-bottom: 20px;**

**}**

**th, td {**

**padding: 12px;**

**text-align: left;**

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

**}**

**th {**

**background-color: #f2f2f2;**

**color: #333;**

**}**

**tr:hover {**

**background-color: #f5f5f5;**

**}**

**/\* Стили для кнопок \*/**

**button {**

**padding: 6px 12px;**

**margin: 0 5px;**

**border: none;**

**border-radius: 4px;**

**cursor: pointer;**

**}**

**.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;**

**margin-top: 20px;**

**}**

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

**width: 70%;**

**padding: 10px;**

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

**border-radius: 4px;**

**margin-right: 10px;**

**}**

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

**padding: 10px 20px;**

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

**color: white;**

**border: none;**

**border-radius: 4px;**

**cursor: pointer;**

**}**

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

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

**}**

**/\* Адаптивность \*/**

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

**table, form {**

**width: 100%;**

**}**

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

**width: 60%;**

**}**

**}**

**</style>**

**</head>**

**<body>**

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

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

**{{adminButton}}**

**<button id="logout-btn">Log Out</button>**

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

**<tr>**

**<th>Номер</th>**

**<th>Order Index</th>**

**<th>Text</th>**

**<th>Action</th>**

**</tr>**

**{{rows}}**

**</table>**

**<br>**

**<input type="text" id="new-item" placeholder="Add new item">**

**<button id="add-btn" onclick="addItem()">Add</button>**

**</div>**

**<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>**

**<script>**

**async function addItem() {**

**const text = document.getElementById('new-item').value;**

**if (!text) return;**

**try {**

**const response = await fetch('/add', {**

**method: 'POST',**

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

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

**});**

**const data = await response.json();**

**if (data.success) {**

**location.reload();**

**} else {**

**alert('Ошибка: ' + data.error);**

**}**

**} catch (error) {**

**console.error('Ошибка:', error);**

**}**

**}**

**async function logout() {**

**await fetch('/logout', { method: 'POST' });**

**window.location.href = '/login';**

**}**

**document.querySelectorAll('.edit-btn').forEach(button => {**

**button.addEventListener('click', async () => {**

**const id = button.getAttribute('data-id');**

**try {**

**const response = await fetch(`/getItem?id=${id}`, {**

**method: 'GET',**

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

**});**

**const data = await response.json();**

**if (data.success) {**

**const currentText = data.item.text;**

**const currentOrder = data.item.order\_index;**

**const newText = prompt(`Enter new text (current: "${currentText}")`, currentText);**

**const newOrder = prompt(`Изменить порядок (current: ${currentOrder})`, currentOrder);**

**if (newText && newOrder !== null) {**

**await fetch('/edit', {**

**method: 'POST',**

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

**body: JSON.stringify({ id, text: newText, orderIndex: newOrder })**

**});**

**location.reload();**

**}**

**} else {**

**alert('Ошибка: ' + data.error);**

**}**

**} catch (error) {**

**console.error('Ошибка:', error);**

**}**

**});**

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

**button.addEventListener('click', async () => {**

**const id = button.getAttribute('data-id');**

**try {**

**const response = await fetch('/delete', {**

**method: 'POST',**

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

**body: JSON.stringify({ id })**

**});**

**const data = await response.json();**

**if (data.success) {**

**location.reload();**

**} else {**

**alert('Ошибка: ' + data.error);**

**}**

**} catch (error) {**

**console.error('Ошибка:', error);**

**}**

**});**

**});**

**document.querySelectorAll('.move-up-btn').forEach(button => {**

**button.addEventListener('click', async () => {**

**const id = button.getAttribute('data-id');**

**const currentRow = button.closest('tr');**

**const prevRow = currentRow.previousElementSibling;**

**if (prevRow) {**

**const newOrderIndex = parseInt(prevRow.querySelector('td:nth-child(2)').textContent) - 1;**

**await reorderItem(id, newOrderIndex);**

**}**

**});**

**});**

**document.querySelectorAll('.move-down-btn').forEach(button => {**

**button.addEventListener('click', async () => {**

**const id = button.getAttribute('data-id');**

**const currentRow = button.closest('tr');**

**const nextRow = currentRow.nextElementSibling;**

**if (nextRow) {**

**const newOrderIndex = parseInt(nextRow.querySelector('td:nth-child(2)').textContent);**

**await reorderItem(id, newOrderIndex);**

**}**

**});**

**});**

**async function reorderItem(id, newOrderIndex) {**

**try {**

**const response = await fetch('/reorder', {**

**method: 'POST',**

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

**body: JSON.stringify({ id, newOrderIndex })**

**});**

**const data = await response.json();**

**if (data.success) {**

**location.reload();**

**} else {**

**alert('Ошибка: ' + data.error);**

**}**

**} catch (error) {**

**console.error('Ошибка:', error);**

**}**

**}**

**});**

**document.querySelectorAll('.edit-btn').forEach(button => {**

**button.addEventListener('click', () => {**

**const id = button.getAttribute('data-id');**

**editItem(id);**

**});**

**});**

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

**button.addEventListener('click', () => {**

**const id = button.getAttribute('data-id');**

**deleteItem(id);**

**});**

**});**

**document.querySelectorAll('.move-up-btn').forEach(button => {**

**button.addEventListener('click', async () => {**

**const id = button.getAttribute('data-id');**

**try {**

**const response = await fetch('/moveUp', {**

**method: 'POST',**

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

**body: JSON.stringify({ id })**

**});**

**if (response.ok) {**

**location.reload(); // Перезагрузка страницы для обновления списка**

**} else {**

**console.error('Ошибка при перемещении вверх');**

**}**

**} catch (error) {**

**console.error('Ошибка:', error);**

**}**

**});**

**});**

**document.querySelectorAll('.move-down-btn').forEach(button => {**

**button.addEventListener('click', async () => {**

**const id = button.getAttribute('data-id');**

**try {**

**const response = await fetch('/moveDown', {**

**method: 'POST',**

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

**body: JSON.stringify({ id })**

**});**

**if (response.ok) {**

**location.reload();**

**} else {**

**console.error('Ошибка при перемещении вниз');**

**}**

**} catch (error) {**

**console.error('Ошибка:', error);**

**}**

**});**

**});**

**document.getElementById('addForm').addEventListener('submit', async (e) => {**

**e.preventDefault();**

**const text = document.getElementById('newItem').value;**

**try {**

**const response = await fetch('/add', {**

**method: 'POST',**

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

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

**});**

**if (response.ok) {**

**location.reload();**

**} else {**

**console.error('Ошибка при добавлении');**

**}**

**} catch (error) {**

**console.error('Ошибка:', error);**

**}**

**});**

**async function reorderItem(id, newOrderIndex) {**

**try {**

**const response = await fetch('/reorder', {**

**method: 'POST',**

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

**body: JSON.stringify({ id, newOrderIndex })**

**});**

**const data = await response.json();**

**if (data.success) {**

**location.reload();**

**} else {**

**alert('Ошибка: ' + data.error);**

**}**

**} catch (error) {**

**console.error('Ошибка:', error);**

**}**

**}**

**let isMoving = false;**

**document.querySelectorAll('.move-up-btn').forEach(button => {**

**button.addEventListener('click', async () => {**

**if (isMoving) return; // Если уже идёт перемещение, игнорируем клик**

**isMoving = true;**

**const id = button.getAttribute('data-id');**

**try {**

**const response = await fetch('/moveUp', {**

**method: 'POST',**

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

**body: JSON.stringify({ id })**

**});**

**if (response.ok) {**

**location.reload();**

**} else {**

**console.error('Ошибка при перемещении вверх');**

**}**

**} catch (error) {**

**console.error('Ошибка:', error);**

**} finally {**

**isMoving = false; // Сбрасываем флаг после завершения**

**}**

**});**

**});**

**document.querySelectorAll('.move-down-btn').forEach(button => {**

**button.addEventListener('click', async () => {**

**if (isMoving) return; // Если уже идёт перемещение, игнорируем клик**

**isMoving = true;**

**const id = button.getAttribute('data-id');**

**try {**

**const response = await fetch('/moveDown', {**

**method: 'POST',**

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

**body: JSON.stringify({ id })**

**});**

**if (response.ok) {**

**location.reload();**

**} else {**

**console.error('Ошибка при перемещении вниз');**

**}**

**} catch (error) {**

**console.error('Ошибка:', error);**

**} finally {**

**isMoving = false; // Сбрасываем флаг после завершения**

**}**

**});**

**});**

**</script>**

**</html>**

**admin.html**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

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

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

**<title>Admin Panel</title>**

**<style>**

**body {**

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

**background-color: #f4f4f4;**

**color: #333;**

**margin: 0;**

**padding: 20px;**

**}**

**h1, h2 {**

**color: #007bff;**

**}**

**a {**

**color: #007bff;**

**text-decoration: none;**

**}**

**a:hover {**

**text-decoration: underline;**

**}**

**table {**

**width: 100%;**

**border-collapse: collapse;**

**margin-bottom: 20px;**

**}**

**th, td {**

**padding: 10px;**

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

**text-align: left;**

**}**

**th {**

**background-color: #f8f9fa;**

**font-weight: bold;**

**}**

**form label {**

**display: block;**

**margin-top: 10px;**

**}**

**form input[type="text"], form input[type="password"] {**

**width: 100%;**

**padding: 8px;**

**margin-top: 5px;**

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

**border-radius: 4px;**

**box-sizing: border-box;**

**}**

**form input[type="checkbox"] {**

**margin-right: 5px;**

**}**

**button {**

**background-color: #007bff;**

**color: white;**

**padding: 10px 15px;**

**border: none;**

**border-radius: 4px;**

**cursor: pointer;**

**margin-top: 10px;**

**}**

**button:hover {**

**background-color: #0056b3;**

**}**

**.modal {**

**display: none;**

**position: fixed;**

**top: 0;**

**left: 0;**

**width: 100%;**

**height: 100%;**

**background-color: rgba(0,0,0,0.5);**

**}**

**.modal-content {**

**background-color: white;**

**margin: 15% auto;**

**padding: 20px;**

**border: 1px solid #888;**

**width: 300px;**

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

**}**

**#cancelEdit {**

**background-color: #6c757d;**

**}**

**#cancelEdit:hover {**

**background-color: #5a6268;**

**}**

**.edit-btn {**

**background-color: #007bff;**

**color: white;**

**padding: 5px 10px;**

**border: none;**

**border-radius: 4px;**

**cursor: pointer;**

**}**

**.edit-btn:hover {**

**background-color: #0056b3;**

**}**

**.delete-btn {**

**background-color: #dc3545;**

**color: white;**

**padding: 5px 10px;**

**border: none;**

**border-radius: 4px;**

**cursor: pointer;**

**}**

**.delete-btn:hover {**

**background-color: #c82333;**

**}**

**</style>**

**</head>**

**<body>**

**<h1>Admin Panel</h1>**

**<a href="/">Back to To-Do List</a>**

**<h2>Users</h2>**

**<table>**

**<tr>**

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

**<th>Login</th>**

**<th>Password</th>**

**<th>Is Admin</th>**

**<th>Actions</th>**

**</tr>**

**{{userRows}}**

**</table>**

**<h2>Add New User</h2>**

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

**<label for="login">Login:</label>**

**<input type="text" id="login" required>**

**<label for="password">Password:</label>**

**<input type="password" id="password" required>**

**<label><input type="checkbox" id="isAdmin"> Is Admin</label>**

**<button type="submit">Add User</button>**

**</form>**

**<div id="editModal" class="modal">**

**<div class="modal-content">**

**<h2>Edit User</h2>**

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

**<input type="hidden" id="editId">**

**<label for="editLogin">Login:</label>**

**<input type="text" id="editLogin" required>**

**<label for="editPassword">Password:</label>**

**<input type="text" id="editPassword" required>**

**<label>Is Admin:</label>**

**<label><input type="radio" name="editIsAdmin" value="admin"> Да</label>**

**<label><input type="radio" name="editIsAdmin" value="user"> Нет</label>**

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

**<button type="button" id="cancelEdit">Cancel</button>**

**</form>**

**</div>**

**</div>**

**<script>**

**// Функция для переключения видимости пароля**

**function togglePasswordVisibility(input, button) {**

**button.addEventListener('mousedown', () => {**

**input.type = 'text';**

**});**

**button.addEventListener('mouseup', () => {**

**input.type = 'password';**

**});**

**button.addEventListener('mouseleave', () => {**

**input.type = 'password';**

**});**

**}**

**// Применяем переключение ко всем ячейкам пароля**

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

**document.querySelectorAll('.password-cell').forEach(cell => {**

**const input = cell.querySelector('input');**

**const button = cell.querySelector('.show-password-btn');**

**togglePasswordVisibility(input, button);**

**});**

**});**

**// Обработчик для добавления пользователя**

**document.getElementById('addUserForm').addEventListener('submit', async (e) => {**

**e.preventDefault();**

**const login = document.getElementById('login').value;**

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

**const isAdmin = document.getElementById('isAdmin').checked ? 'on' : 'off';**

**try {**

**const response = await fetch('/addUser', {**

**method: 'POST',**

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

**body: JSON.stringify({ login, password, isAdmin })**

**});**

**const result = await response.json();**

**if (result.success) {**

**location.reload();**

**} else {**

**alert('Ошибка: ' + result.error);**

**}**

**} catch (err) {**

**alert('Ошибка: ' + err.message);**

**}**

**});**

**// Обработчик для удаления пользователя**

**const editModal = document.getElementById('editModal');**

**document.querySelectorAll('.edit-btn').forEach(button => {**

**button.addEventListener('click', async () => {**

**const id = button.getAttribute('data-id');**

**try {**

**const response = await fetch('/getPassword', {**

**method: 'POST',**

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

**body: JSON.stringify({ id })**

**});**

**const result = await response.json();**

**if (result.success) {**

**document.getElementById('editId').value = id;**

**document.getElementById('editLogin').value = button.parentElement.parentElement.children[1].textContent;**

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

**const isAdmin = button.parentElement.parentElement.children[3].textContent === 'Да';**

**document.querySelector('input[name="editIsAdmin"][value="admin"]').checked = isAdmin;**

**document.querySelector('input[name="editIsAdmin"][value="user"]').checked = !isAdmin;**

**editModal.style.display = 'block';**

**} else {**

**alert('Ошибка: ' + result.error);**

**}**

**} catch (err) {**

**alert('Ошибка: ' + err.message);**

**}**

**});**

**});**

**document.getElementById('editUserForm').addEventListener('submit', async (e) => {**

**e.preventDefault();**

**const id = document.getElementById('editId').value;**

**const login = document.getElementById('editLogin').value;**

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

**const isAdmin = document.querySelector('input[name="editIsAdmin"]:checked').value === 'admin' ? 'on' : 'off';**

**try {**

**const response = await fetch('/editUser', {**

**method: 'POST',**

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

**body: JSON.stringify({ id, login, password, isAdmin })**

**});**

**const result = await response.json();**

**if (result.success) {**

**location.reload();**

**} else {**

**alert('Ошибка: ' + result.error);**

**}**

**} catch (err) {**

**alert('Ошибка: ' + err.message);**

**}**

**});**

**document.getElementById('cancelEdit').addEventListener('click', () => {**

**editModal.style.display = 'none';**

**});**

**</script>**

**</script>**

**</body>**

**</html>**

**index.js**

**const http = require('http');**

**const fs = require('fs');**

**const path = require('path');**

**const mysql = require('mysql2/promise');**

**const cookieSession = require('cookie-session');**

**const crypto = require('crypto');**

**const PORT = process.env.PORT || 3000;**

**const dbConfig = {**

**host: process.env.DB\_HOST,**

**user: process.env.DB\_USER,**

**password: process.env.DB\_PASSWORD,**

**database: process.env.DB\_DATABASE,**

**};**

**const pool = mysql.createPool(dbConfig);**

**const session = cookieSession({**

**name: 'session',**

**keys: ['my\_secret\_key'],**

**maxAge: 24 \* 60 \* 60 \* 1000,**

**});**

**function isAuthenticated(req) {**

**return req.session && req.session.userId;**

**}**

**async function isAuthenticatedOrToken(req) {**

**if (req.session && req.session.userId) {**

**return true;**

**}**

**const authHeader = req.headers['authorization'];**

**if (authHeader && authHeader.startsWith('Bearer ')) {**

**const token = authHeader.substring(7);**

**try {**

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

**const [rows] = await connection.execute(**

**'SELECT id FROM users WHERE token = ?',**

**[token]**

**);**

**await connection.end();**

**return rows.length > 0;**

**} catch (error) {**

**console.error('Ошибка при проверке токена:', error);**

**return false;**

**}**

**}**

**return false;**

**}**

**async function executeWithRetry(operation, maxRetries = 3) {**

**for (let attempt = 1; attempt <= maxRetries; attempt++) {**

**try {**

**return await operation();**

**} catch (error) {**

**if (error.code === 'ER\_LOCK\_DEADLOCK' && attempt < maxRetries) {**

**console.warn(`Дедлок на попытке ${attempt}, повторяем...`);**

**await new Promise(resolve => setTimeout(resolve, 100 \* attempt));**

**continue;**

**}**

**throw error;**

**}**

**}**

**}**

**async function retrieveListItems(userId) {**

**try {**

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

**const query = 'SELECT id, text, order\_index FROM items WHERE user\_id = ? ORDER BY order\_index';**

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

**console.log(`Items for user ${userId}:`, rows.map(item => ({id: item.id, order\_index: item.order\_index})));**

**await connection.end();**

**return rows;**

**} catch (error) {**

**console.error('Error retrieving list items:', error);**

**throw error;**

**}**

**}**

**async function getUserIdFromRequest(req) {**

**if (req.session && req.session.userId) {**

**return req.session.userId;**

**}**

**const authHeader = req.headers['authorization'];**

**if (authHeader && authHeader.startsWith('Bearer ')) {**

**const token = authHeader.substring(7);**

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

**const [rows] = await connection.execute(**

**'SELECT id FROM users WHERE token = ?',**

**[token]**

**);**

**await connection.end();**

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

**return rows[0].id;**

**}**

**}**

**throw new Error('Неавторизованный доступ');**

**}**

**async function getHtmlRows(userId) {**

**const todoItems = await retrieveListItems(userId);**

**return todoItems.map((item, index) => `**

**<tr>**

**<td>${index + 1}</td>**

**<td>${item.order\_index}</td>**

**<td>${item.text}</td>**

**<td>**

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

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

**<button class="move-up-btn" data-id="${item.id}">↑</button>**

**<button class="move-down-btn" data-id="${item.id}">↓</button>**

**</td>**

**</tr>**

**`).join('');**

**}**

**async function rebuildOrderIndex(userId) {**

**let connection;**

**try {**

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

**const [items] = await connection.execute(**

**'SELECT id FROM items WHERE user\_id = ? ORDER BY order\_index',**

**[userId]**

**);**

**for (let i = 0; i < items.length; i++) {**

**await connection.execute(**

**'UPDATE items SET order\_index = ? WHERE id = ?',**

**[i + 1, items[i].id]**

**);**

**}**

**} catch (error) {**

**console.error('Error rebuilding order index:', error);**

**throw error;**

**} finally {**

**if (connection) await connection.end();**

**}**

**}**

**async function handleRequest(req, res) {**

**session(req, res, async () => {**

**if (req.url === '/login' && req.method === 'GET') {**

**try {**

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

**res.writeHead(200, { 'Content-Type': 'text/html' });**

**res.end(html);**

**} catch (err) {**

**res.writeHead(500, { 'Content-Type': 'text/plain' });**

**res.end('Error loading login.html');**

**}**

**} else if (req.url === '/login' && req.method === 'POST') {**

**let body = '';**

**req.on('data', chunk => { body += chunk; });**

**req.on('end', async () => {**

**try {**

**const { login, password } = JSON.parse(body);**

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

**const [rows] = await connection.execute(**

**'SELECT id, role FROM users WHERE login = ? AND password = ?',**

**[login, password]**

**);**

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

**const token = crypto.randomBytes(32).toString('hex');**

**await connection.execute(**

**'UPDATE users SET token = ? WHERE id = ?',**

**[token, rows[0].id]**

**);**

**req.session.userId = rows[0].id;**

**req.session.role = rows[0].role;**

**await connection.end();**

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: true, token: token }));**

**} else {**

**await connection.end();**

**res.writeHead(401, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: 'Неверный логин или пароль' }));**

**}**

**} catch (error) {**

**res.writeHead(500, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: 'Ошибка сервера' }));**

**}**

**});**

**} else if (req.url === '/' && req.method === 'GET') {**

**if (!req.session) {**

**console.error('Session not initialized');**

**res.writeHead(500, { 'Content-Type': 'text/plain' });**

**res.end('Session error');**

**return;**

**}**

**if (isAuthenticated(req)) {**

**try {**

**const userRole = req.session.role || 'user';**

**let adminButtonHtml = '';**

**if (userRole === 'admin') {**

**adminButtonHtml = '<button onclick="window.location.href=\'/admin.html\'">Go to Admin Panel</button>';**

**}**

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

**if (!html) {**

**throw new Error('HTML content is empty');**

**}**

**const processedHtml = html.replace('{{adminButton}}', adminButtonHtml || '')**

**.replace('{{userRole}}', userRole)**

**.replace('{{rows}}', await getHtmlRows(req.session.userId));**

**res.writeHead(200, { 'Content-Type': 'text/html' });**

**res.end(processedHtml);**

**} catch (err) {**

**console.error('Error in route /:', err.message);**

**res.writeHead(500, { 'Content-Type': 'text/plain' });**

**res.end('Error loading index.html: ' + err.message);**

**}**

**} else {**

**res.writeHead(302, { 'Location': '/login' });**

**res.end();**

**}**

**} else if (req.url === '/admin.html' && req.method === 'GET') {**

**if (isAuthenticated(req) && req.session.role === 'admin') {**

**try {**

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

**const userRows = await getUserHtmlRows();**

**const processedHtml = html.replace('{{userRows}}', userRows);**

**res.writeHead(200, { 'Content-Type': 'text/html' });**

**res.end(processedHtml);**

**} catch (err) {**

**res.writeHead(500, { 'Content-Type': 'text/plain' });**

**res.end('Ошибка загрузки admin.html');**

**}**

**} else {**

**res.writeHead(403, { 'Content-Type': 'text/plain' });**

**res.end('Доступ запрещен');**

**}**

**} else if (req.method === 'POST' && req.url === '/addUser') {**

**if (!isAuthenticated(req) || req.session.role !== 'admin') {**

**res.writeHead(403, { 'Content-Type': 'text/plain' });**

**res.end('Доступ запрещен');**

**return;**

**}**

**let body = '';**

**req.on('data', chunk => { body += chunk; });**

**req.on('end', async () => {**

**try {**

**const { login, password, isAdmin } = JSON.parse(body);**

**if (!login || !password) {**

**throw new Error('Логин и пароль обязательны');**

**}**

**const role = isAdmin === 'on' ? 'admin' : 'user';**

**const is\_admin = role === 'admin' ? 1 : 0;**

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

**await connection.execute(**

**'INSERT INTO users (login, password, is\_admin, role) VALUES (?, ?, ?, ?)',**

**[login, password, is\_admin, role]**

**);**

**await connection.end();**

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: true }));**

**} catch (error) {**

**res.writeHead(400, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: error.message }));**

**}**

**});**

**} else if (req.method === 'POST' && req.url === '/deleteUser') {**

**if (!isAuthenticated(req) || req.session.role !== 'admin') {**

**res.writeHead(403, { 'Content-Type': 'text/plain' });**

**res.end('Доступ запрещен');**

**return;**

**}**

**let body = '';**

**req.on('data', chunk => { body += chunk; });**

**req.on('end', async () => {**

**try {**

**const { id } = JSON.parse(body);**

**if (!id) throw new Error('ID пользователя обязателен');**

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

**await connection.execute('DELETE FROM users WHERE id = ?', [id]);**

**await connection.end();**

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: true }));**

**} catch (error) {**

**res.writeHead(400, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: error.message }));**

**}**

**});**

**} else if (req.method === 'POST' && req.url === '/editUser') {**

**if (!isAuthenticated(req) || req.session.role !== 'admin') {**

**res.writeHead(403, { 'Content-Type': 'text/plain' });**

**res.end('Доступ запрещен');**

**return;**

**}**

**let body = '';**

**req.on('data', chunk => { body += chunk; });**

**req.on('end', async () => {**

**try {**

**const { id, login, password, isAdmin } = JSON.parse(body);**

**if (!id || !login || !password) {**

**throw new Error('ID, логин и пароль обязательны');**

**}**

**const role = isAdmin === 'on' ? 'admin' : 'user';**

**const is\_admin = role === 'admin' ? 1 : 0;**

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

**await connection.execute(**

**'UPDATE users SET login = ?, password = ?, is\_admin = ?, role = ? WHERE id = ?',**

**[login, password, is\_admin, role, id]**

**);**

**await connection.end();**

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: true }));**

**} catch (error) {**

**res.writeHead(400, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: error.message }));**

**}**

**});**

**} else if (req.method === 'POST' && req.url === '/getPassword') {**

**if (!isAuthenticated(req) || req.session.role !== 'admin') {**

**res.writeHead(403, { 'Content-Type': 'text/plain' });**

**res.end('Доступ запрещен');**

**return;**

**}**

**let body = '';**

**req.on('data', chunk => { body += chunk; });**

**req.on('end', async () => {**

**try {**

**const { id } = JSON.parse(body);**

**if (!id) throw new Error('ID пользователя обязателен');**

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

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

**await connection.end();**

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

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: true, password: rows[0].password }));**

**} else {**

**res.writeHead(404, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: 'Пользователь не найден' }));**

**}**

**} catch (error) {**

**res.writeHead(400, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: error.message }));**

**}**

**});**

**} else if (req.method === 'POST' && req.url === '/add') {**

**if (!(await isAuthenticatedOrToken(req))) {**

**res.writeHead(401, { 'Content-Type': 'text/plain' });**

**res.end('Unauthorized');**

**return;**

**}**

**let body = '';**

**req.on('data', chunk => { body += chunk; });**

**req.on('end', async () => {**

**try {**

**const { text } = JSON.parse(body);**

**if (!text) throw new Error("Текст не передан");**

**const userId = await getUserIdFromRequest(req);**

**const newItemId = await addItem(text, userId);**

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: true, id: newItemId }));**

**} catch (error) {**

**res.writeHead(400, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: error.message }));**

**}**

**});**

**} else if (req.method === 'POST' && req.url === '/delete') {**

**if (!(await isAuthenticatedOrToken(req))) {**

**res.writeHead(401, { 'Content-Type': 'text/plain' });**

**res.end('Unauthorized');**

**return;**

**}**

**let body = '';**

**req.on('data', chunk => { body += chunk; });**

**req.on('end', async () => {**

**try {**

**const { id } = JSON.parse(body);**

**if (!id) throw new Error("ID не передан");**

**const userId = await getUserIdFromRequest(req);**

**await deleteItem(id, userId);**

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: true }));**

**} catch (error) {**

**res.writeHead(400, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: error.message }));**

**}**

**});**

**} else if (req.url.startsWith('/getItem') && req.method === 'GET') {**

**if (!isAuthenticated(req)) {**

**res.writeHead(401, { 'Content-Type': 'text/plain' });**

**res.end('Unauthorized');**

**return;**

**}**

**const urlParams = new URLSearchParams(req.url.split('?')[1]);**

**const id = urlParams.get('id');**

**try {**

**const userId = req.session.userId;**

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

**const [rows] = await connection.execute(**

**'SELECT text, order\_index FROM items WHERE id = ? AND user\_id = ?',**

**[id, userId]**

**);**

**await connection.end();**

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

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: true, item: rows[0] }));**

**} else {**

**res.writeHead(404, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: 'Задача не найдена' }));**

**}**

**} catch (error) {**

**console.error('Ошибка:', error);**

**res.writeHead(500, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: 'Ошибка сервера' }));**

**}**

**} else if (req.method === 'POST' && req.url === '/edit') {**

**if (!isAuthenticated(req)) {**

**res.writeHead(401, { 'Content-Type': 'text/plain' });**

**res.end('Unauthorized');**

**return;**

**}**

**let body = '';**

**req.on('data', chunk => { body += chunk; });**

**req.on('end', async () => {**

**try {**

**const { id, text, orderIndex } = JSON.parse(body);**

**if (!id || !text || orderIndex === undefined) throw new Error("ID, текст или порядок не переданы");**

**const userId = await getUserIdFromRequest(req);**

**await updateItem(id, text, orderIndex, userId);**

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: true }));**

**} catch (error) {**

**res.writeHead(400, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: error.message }));**

**}**

**});**

**} else if (req.method === 'POST' && req.url === '/reorder') {**

**if (!(await isAuthenticatedOrToken(req))) {**

**res.writeHead(401, { 'Content-Type': 'text/plain' });**

**res.end('Unauthorized');**

**return;**

**}**

**let body = '';**

**req.on('data', chunk => { body += chunk; });**

**req.on('end', async () => {**

**try {**

**const { id, newOrderIndex } = JSON.parse(body);**

**if (!id || newOrderIndex === undefined) throw new Error("ID или новый порядок не переданы");**

**const userId = await getUserIdFromRequest(req);**

**await reorderItem(id, newOrderIndex, userId);**

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: true }));**

**} catch (error) {**

**res.writeHead(400, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: error.message }));**

**}**

**});**

**} else if (req.method === 'POST' && req.url === '/logout') {**

**req.session = null;**

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: true }));**

**} else if (req.url === '/api/items' && req.method === 'GET') {**

**const authHeader = req.headers['authorization'];**

**if (authHeader && authHeader.startsWith('Bearer ')) {**

**const token = authHeader.substring(7);**

**try {**

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

**const [rows] = await connection.execute(**

**'SELECT id FROM users WHERE token = ?',**

**[token]**

**);**

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

**const userId = rows[0].id;**

**const items = await retrieveListItems(userId);**

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify(items));**

**} else {**

**res.writeHead(401, { 'Content-Type': 'text/plain' });**

**res.end('Unauthorized');**

**}**

**await connection.end();**

**} catch (error) {**

**console.error('Ошибка:', error);**

**res.writeHead(500, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: 'Error retrieving items' }));**

**}**

**} else {**

**res.writeHead(401, { 'Content-Type': 'text/plain' });**

**res.end('Unauthorized');**

**}**

**} else if (req.method === 'POST' && req.url === '/moveUp') {**

**if (!(await isAuthenticatedOrToken(req))) {**

**res.writeHead(401, { 'Content-Type': 'text/plain' });**

**res.end('Unauthorized');**

**return;**

**}**

**let body = '';**

**req.on('data', chunk => { body += chunk; });**

**req.on('end', async () => {**

**try {**

**const { id } = JSON.parse(body);**

**if (!id) throw new Error("ID не передан");**

**const userId = await getUserIdFromRequest(req);**

**await moveUp(id, userId);**

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: true }));**

**} catch (error) {**

**res.writeHead(400, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: error.message }));**

**}**

**});**

**} else if (req.method === 'POST' && req.url === '/moveDown') {**

**if (!(await isAuthenticatedOrToken(req))) {**

**res.writeHead(401, { 'Content-Type': 'text/plain' });**

**res.end('Unauthorized');**

**return;**

**}**

**let body = '';**

**req.on('data', chunk => { body += chunk; });**

**req.on('end', async () => {**

**try {**

**const { id } = JSON.parse(body);**

**if (!id) throw new Error("ID не передан");**

**const userId = await getUserIdFromRequest(req);**

**await moveDown(id, userId);**

**res.writeHead(200, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: true }));**

**} catch (error) {**

**res.writeHead(400, { 'Content-Type': 'application/json' });**

**res.end(JSON.stringify({ success: false, error: error.message }));**

**}**

**});**

**} else {**

**res.writeHead(404, { 'Content-Type': 'text/plain' });**

**res.end('Route not found');**

**}**

**});**

**}**

**async function addItem(text, userId) {**

**try {**

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

**const query = 'INSERT INTO items (text, user\_id, order\_index) VALUES (?, ?, ?)';**

**const [result] = await connection.execute(query, [text, userId, 0]);**

**await connection.end();**

**await rebuildOrderIndex(userId);**

**return result.insertId;**

**} catch (error) {**

**console.error('Error adding item:', error);**

**throw error;**

**}**

**}**

**async function reorderItem(id, newOrderIndex, userId) {**

**try {**

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

**const [currentItem] = await connection.execute(**

**'SELECT order\_index FROM items WHERE id = ? AND user\_id = ?',**

**[id, userId]**

**);**

**if (currentItem.length === 0) throw new Error('Задача не найдена');**

**const currentOrderIndex = currentItem[0].order\_index;**

**if (newOrderIndex > currentOrderIndex) {**

**await connection.execute(**

**'UPDATE items SET order\_index = order\_index - 1 WHERE user\_id = ? AND order\_index > ? AND order\_index <= ?',**

**[userId, currentOrderIndex, newOrderIndex]**

**);**

**} else if (newOrderIndex < currentOrderIndex) {**

**await connection.execute(**

**'UPDATE items SET order\_index = order\_index + 1 WHERE user\_id = ? AND order\_index >= ? AND order\_index < ?',**

**[userId, newOrderIndex, currentOrderIndex]**

**);**

**}**

**await connection.execute(**

**'UPDATE items SET order\_index = ? WHERE id = ? AND user\_id = ?',**

**[newOrderIndex, id, userId]**

**);**

**await connection.end();**

**await rebuildOrderIndex(userId);**

**} catch (error) {**

**console.error('Error reordering item:', error);**

**throw error;**

**}**

**}**

**async function deleteItem(id, userId) {**

**try {**

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

**const query = 'DELETE FROM items WHERE id = ? AND user\_id = ?';**

**const [result] = await connection.execute(query, [id, userId]);**

**await connection.end();**

**await rebuildOrderIndex(userId);**

**return result;**

**} catch (error) {**

**console.error('Error deleting item:', error);**

**throw error;**

**}**

**}**

**async function updateItem(id, newText, newOrderIndex, userId) {**

**try {**

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

**const query = 'UPDATE items SET text = ?, order\_index = ? WHERE id = ? AND user\_id = ?';**

**const [result] = await connection.execute(query, [newText, newOrderIndex, id, userId]);**

**await connection.end();**

**await rebuildOrderIndex(userId);**

**return result;**

**} catch (error) {**

**console.error('Error updating item:', error);**

**throw error;**

**}**

**}**

**async function getUserHtmlRows() {**

**try {**

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

**const [rows] = await connection.execute('SELECT id, login, password, role FROM users ORDER BY id');**

**await connection.end();**

**return rows.map(user => `**

**<tr>**

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

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

**<td class="password-cell">**

**<input type="password" value="${user.password}" disabled>**

**<button class="show-password-btn">👁️</button>**

**</td>**

**<td>${user.role === 'admin' ? 'Да' : 'Нет'}</td>**

**<td>**

**<button class="edit-btn" data-id="${user.id}">Edit</button>**

**<button class="delete-btn" data-id="${user.id}">Delete</button>**

**</td>**

**</tr>**

**`).join('');**

**} catch (error) {**

**console.error('Error retrieving users:', error);**

**throw error;**

**}**

**}**

**const moveUp = async (id, userId) => {**

**await executeWithRetry(async () => {**

**const connection = await pool.getConnection();**

**try {**

**await connection.beginTransaction();**

**const [itemRows] = await connection.execute(**

**'SELECT order\_index FROM items WHERE id = ? AND user\_id = ? FOR UPDATE',**

**[id, userId]**

**);**

**if (!itemRows.length) throw new Error('Item not found');**

**const currentOrderIndex = itemRows[0].order\_index;**

**const [aboveRows] = await connection.execute(**

**'SELECT id, order\_index FROM items WHERE user\_id = ? AND order\_index < ? ORDER BY order\_index DESC LIMIT 1 FOR UPDATE',**

**[userId, currentOrderIndex]**

**);**

**if (!aboveRows.length) throw new Error('No item above to swap with');**

**const aboveItem = aboveRows[0];**

**// Fix the "move up" logic by ensuring only the direct neighbor is swapped**

**await connection.execute(**

**'UPDATE items SET order\_index = ? WHERE id = ?',**

**[aboveItem.order\_index, id]**

**);**

**await connection.execute(**

**'UPDATE items SET order\_index = ? WHERE id = ?',**

**[currentOrderIndex, aboveItem.id]**

**);**

**await connection.commit();**

**} catch (error) {**

**await connection.rollback();**

**throw error;**

**} finally {**

**connection.release();**

**}**

**});**

**};**

**const moveDown = async (id, userId) => {**

**await executeWithRetry(async () => {**

**const connection = await pool.getConnection();**

**try {**

**await connection.beginTransaction();**

**const [itemRows] = await connection.execute(**

**'SELECT order\_index FROM items WHERE id = ? AND user\_id = ? FOR UPDATE',**

**[id, userId]**

**);**

**if (!itemRows.length) throw new Error('Item not found');**

**const currentOrderIndex = itemRows[0].order\_index;**

**const [belowRows] = await connection.execute(**

**'SELECT id, order\_index FROM items WHERE user\_id = ? AND order\_index > ? ORDER BY order\_index ASC LIMIT 1 FOR UPDATE',**

**[userId, currentOrderIndex]**

**);**

**if (!belowRows.length) throw new Error('No item below to swap with');**

**const belowItem = belowRows[0];**

**await connection.execute(**

**'UPDATE items SET order\_index = ? WHERE id = ?',**

**[belowItem.order\_index, id]**

**);**

**await connection.execute(**

**'UPDATE items SET order\_index = ? WHERE id = ?',**

**[currentOrderIndex, belowItem.id]**

**);**

**await connection.commit();**

**} catch (error) {**

**await connection.rollback();**

**throw error;**

**} finally {**

**connection.release();**

**}**

**});**

**};**

**const server = http.createServer(handleRequest);**

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

3. код бота

**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

**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)**

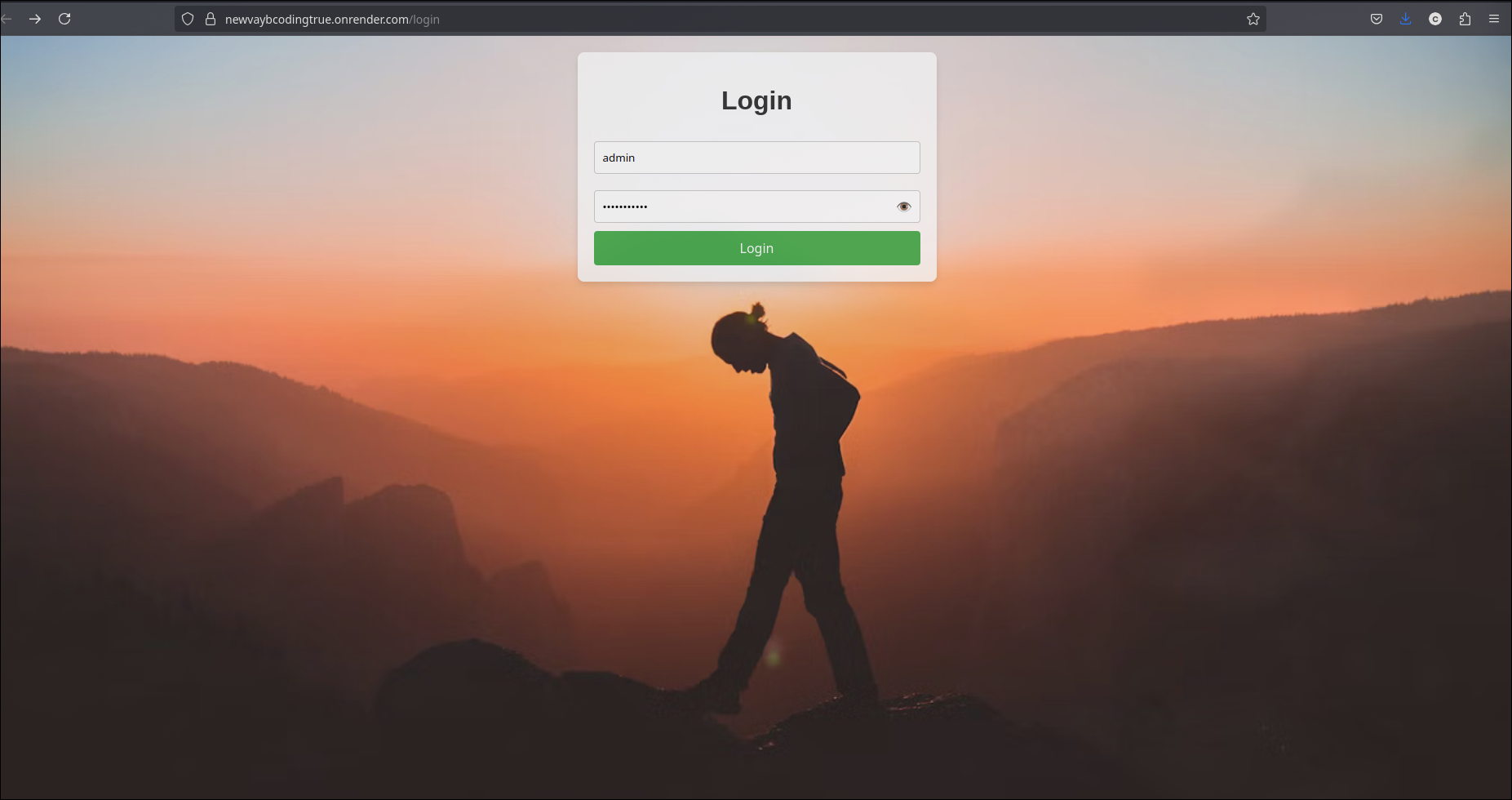
**UPDATE items SET order\_index = id;**

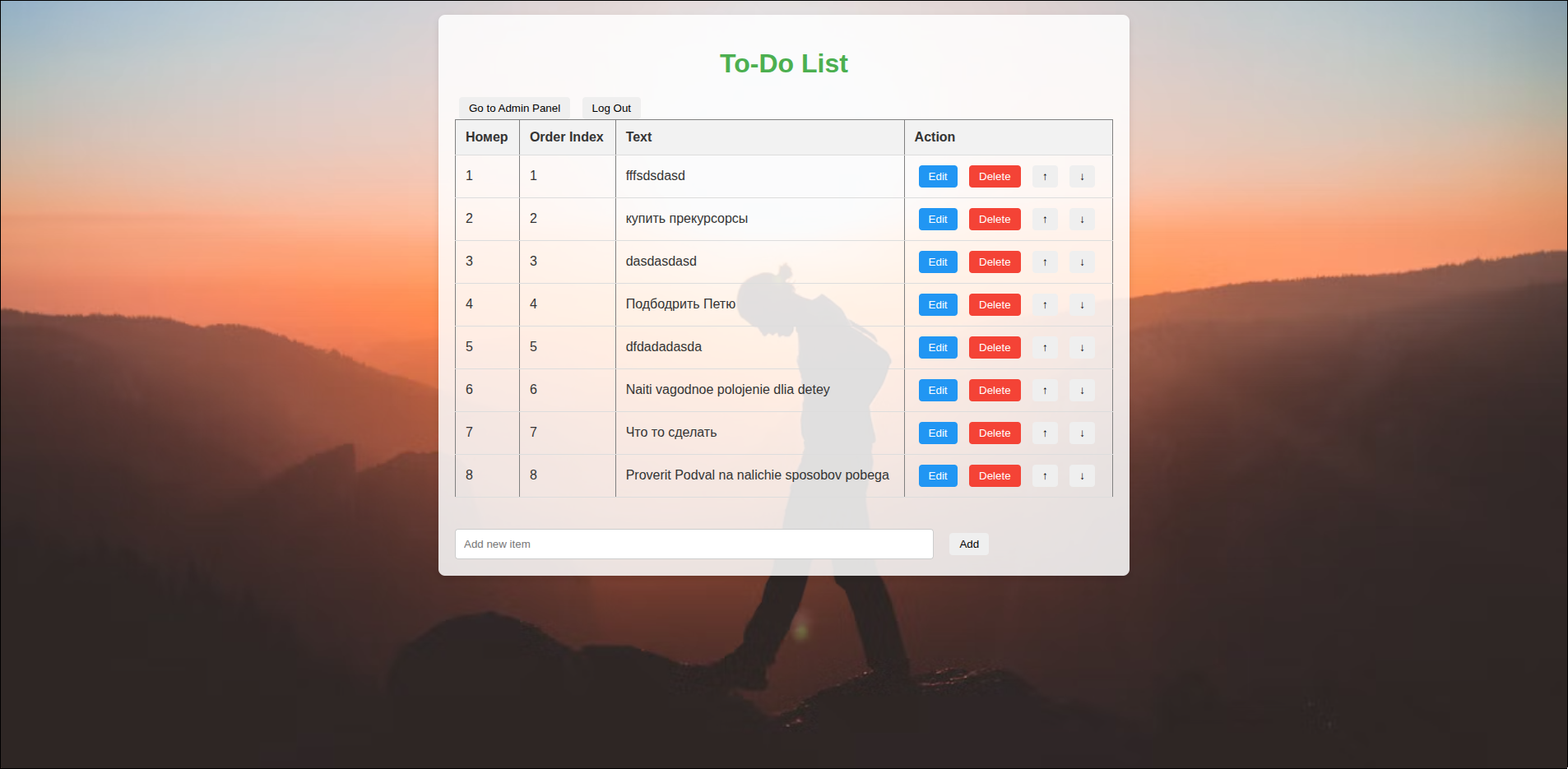
**ALTER DATABASE todolist CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;**

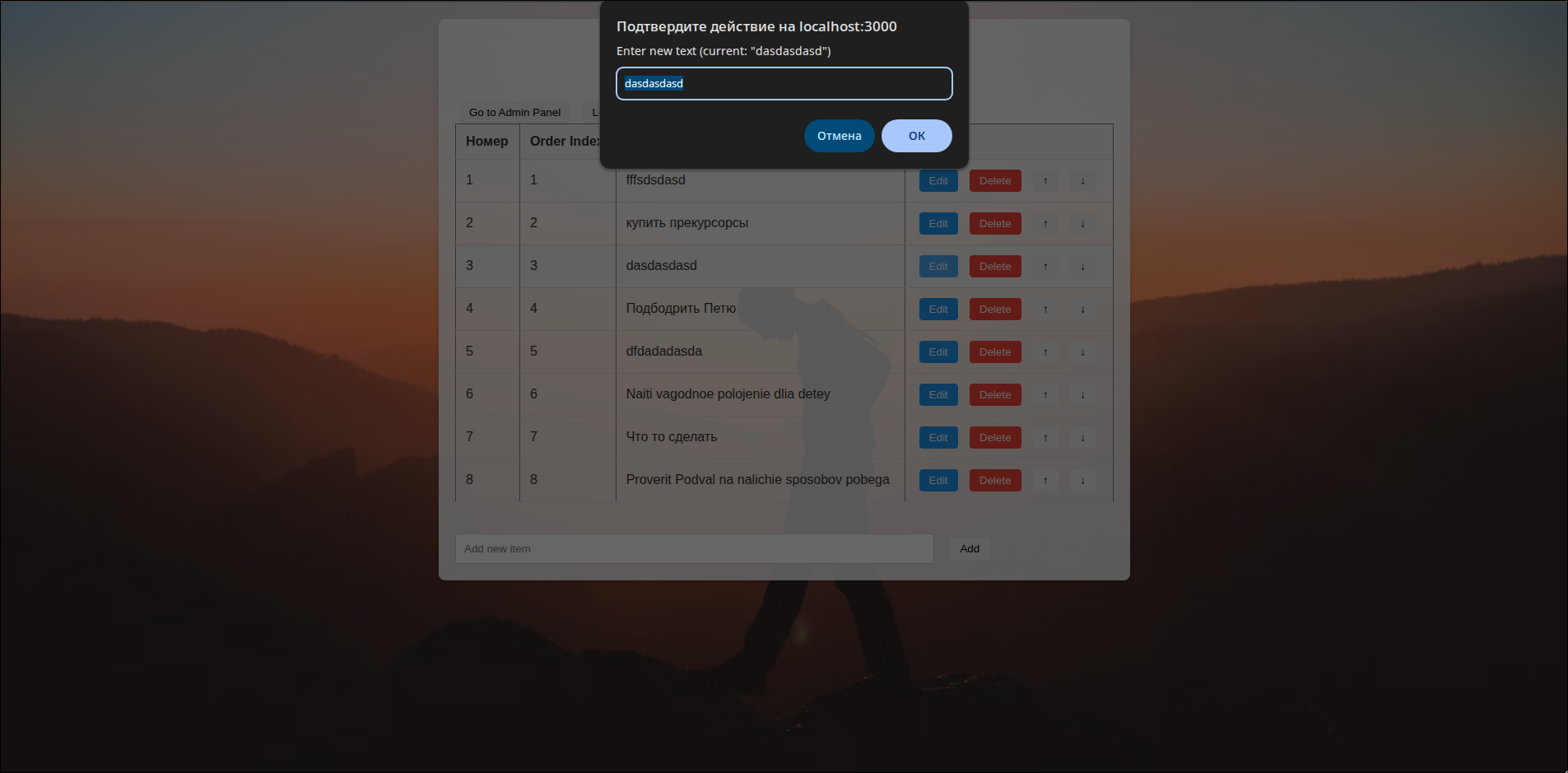
**ALTER TABLE items CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;**

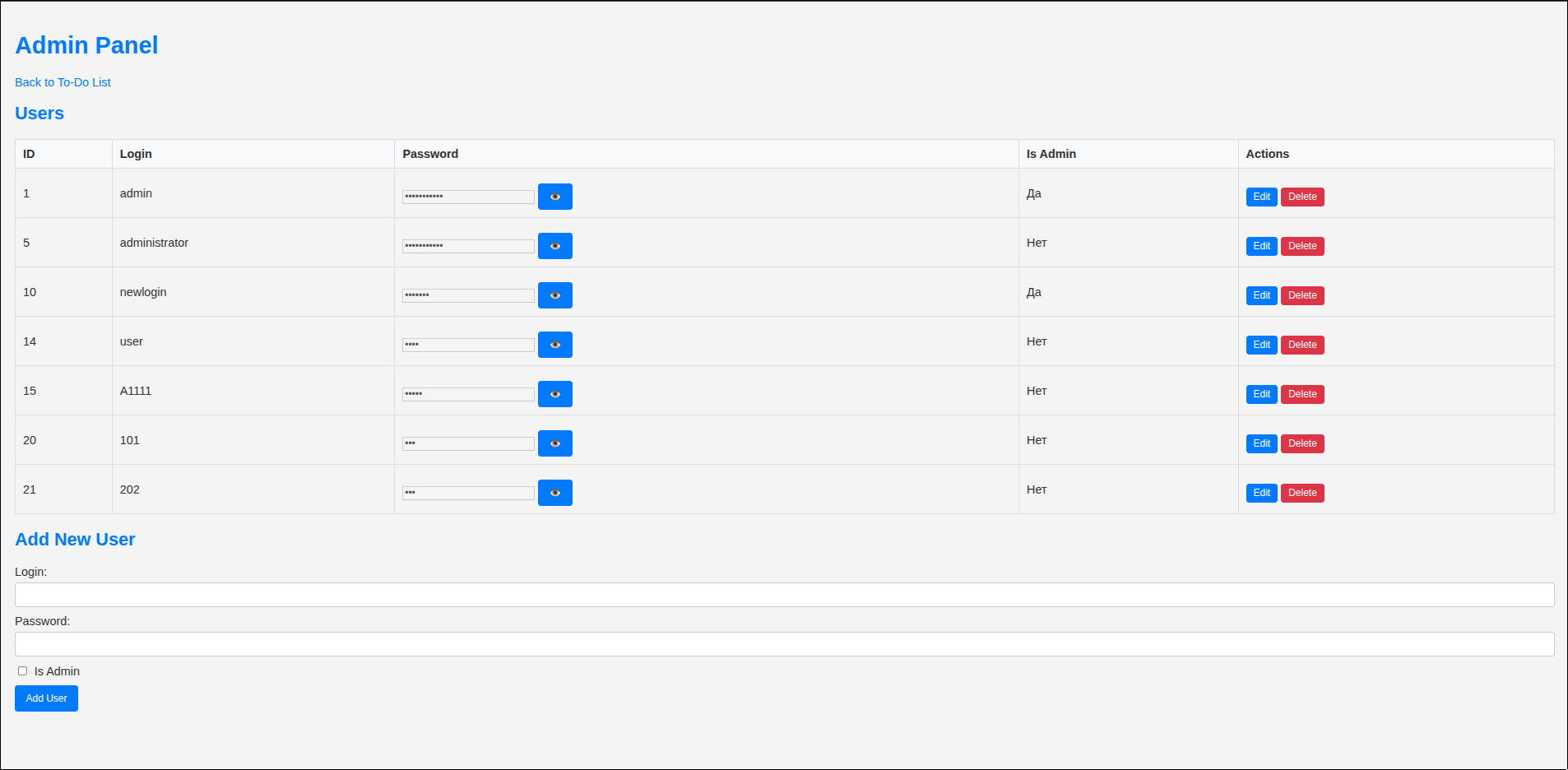
**ALTER TABLE users CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;**

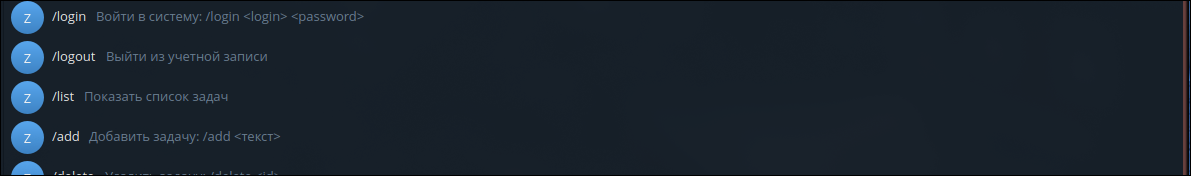
****5. Обзор возможностей сайта****

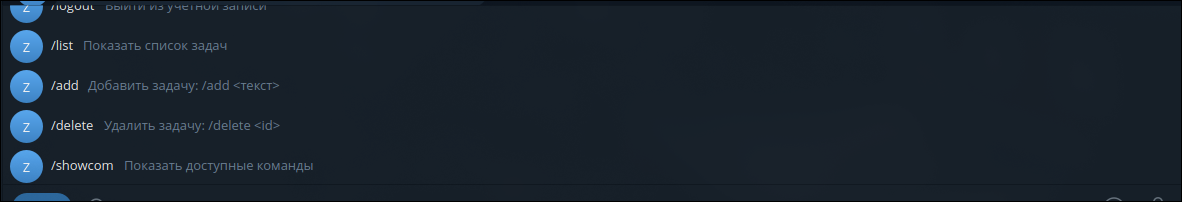
**Рис. 1. Авторизационная страница login.html**

**Рис. 2. Страница списка задач**

**Рис. 3. Возможность редактирования задачи**

**Рис. 4. Страница админ панели**

**Рис. 5. Обзор возможностей бота**

**Рис. 6. Обзор возможностей бота**

****6. развертывание сайта и бота****

**Для развертывания сайта на локалхосте надо в папке проекта:**

**npm install**

**npm install cookie-session**

**node index.js**

**Для развертывания бота на локалхосте надо в папке проекта:**

**npm init -y**

**npm install node-telegram-bot-api axios**

**node bot.js**

****7. промпты для получения кода****

**Промты для бота**

**Промт 1: Общий запрос на создание бота Telegram**

**Запрос**:  
Напишите код для бота Telegram в файле bot.js, который взаимодействует с локальным сервером на http://localhost:3000 для управления списком задач. Бот должен:

1. Использовать библиотеку node-telegram-bot-api с токеном '7671395940:AAHwqDqy-PD8OfhFdjvCIjTE2u2yQ2yZ7wo' и работать в режиме polling.
2. Использовать axios для отправки HTTP-запросов к серверу.
3. Авторизоваться на сервере при запуске, отправляя POST-запрос на /login с данными { login: 'telegram\_bot', password: 'bot\_password' }, сохранять полученный токен и использовать его в заголовке Authorization: Bearer <token> для всех запросов.
4. Поддерживать команды:
   * /list — получать список задач с сервера (GET-запрос на /api/items) и отправлять его пользователю в виде текста (например, "1: Задача 1\n2: Задача 2" или "Список пуст").
   * /add <текст> — добавлять задачу (POST-запрос на /add с { text: <текст> }) и отправлять "Задача добавлена."
   * /delete <id> — удалять задачу по ID (POST-запрос на /delete с { id: <id> }) и отправлять "Задача удалена."
5. Обрабатывать ошибки:
   * Если токен отсутствует, отправлять "Ошибка: бот не авторизован."
   * При ошибках запросов отправлять сообщения вроде "Ошибка получения списка."
6. Выводить в консоль "Бот запущен..." при старте.  
   Предоставьте полный код для файла bot.js.

Промт 2: Настройка бота и аутентификация

**Запрос**:  
Создайте начальную часть кода для файла bot.js. Включите:

* Подключение библиотек node-telegram-bot-api и axios.
* Инициализацию бота с токеном '7671395940:AAHwqDqy-PD8OfhFdjvCIjTE2u2yQ2yZ7wo' в режиме polling.
* Константу baseUrl со значением 'http://localhost:3000'.
* Переменную sessionToken для хранения токена, изначально равную null.
* Асинхронную функцию login(), которая отправляет POST-запрос на http://localhost:3000/login с данными { login: 'telegram\_bot', password: 'bot\_password' }, сохраняет полученный токен в sessionToken и выводит в консоль "Авторизация успешна, токен: <token>" или "Ошибка авторизации: &#x3C;сообщение об ошибке>" при неудаче.</token>
* Вызов функции login() при запуске бота.
* Вывод в консоль "Бот запущен...".

Промт 3: Реализация команды /list

**Запрос**:  
Напишите обработчик для команды /list в боте Telegram. Код должен:

* Проверять наличие sessionToken. Если его нет, отправлять сообщение "Ошибка: бот не авторизован." и завершать выполнение.
* Отправлять GET-запрос с помощью axios на http://localhost:3000/api/items с заголовком Authorization: Bearer <sessionToken>.
* Получать массив задач в формате JSON (например, [{ id: 1, text: 'Задача 1' }, { id: 2, text: 'Задача 2' }]) и формировать текстовое сообщение:
  + Если массив не пуст, преобразовывать его в строку вида "1: Задача 1\n2: Задача 2".
  + Если массив пуст, отправлять "Список пуст."
* Отправлять сформированное сообщение пользователю.
* При ошибке запроса выводить в консоль сообщение об ошибке и отправлять пользователю "Ошибка получения списка."

Промт 4: Реализация команды /add

**Запрос**  
Создайте обработчик для команды /add <текст> в боте Telegram. Код должен:

* Использовать регулярное выражение /\/add (.+)/ для извлечения текста задачи из команды (например, "Купить молоко" из /add Купить молоко).
* Проверять наличие sessionToken. Если его нет, отправлять "Ошибка: бот не авторизован." и завершать выполнение.
* Отправлять POST-запрос с помощью axios на http://localhost:3000/add с данными { text: <извлеченный текст> } и заголовком Authorization: Bearer <sessionToken>.
* При успешном запросе отправлять сообщение "Задача добавлена."
* При ошибке запроса выводить в консоль сообщение об ошибке и отправлять пользователю "Ошибка добавления задачи."

Промт 5: Реализация команды /delete

**Запрос**:  
Реализуйте обработчик для команды /delete <id> в боте Telegram. Код должен:

* Использовать регулярное выражение /\/delete (\d+)/ для извлечения ID задачи из команды (например, "1" из /delete 1).
* Проверять наличие sessionToken. Если его нет, отправлять "Ошибка: бот не авторизован." и завершать выполнение.
* Отправлять POST-запрос с помощью axios на http://localhost:3000/delete с данными { id: <извлеченный ID> } и заголовком Authorization: Bearer <sessionToken>.
* При успешном запросе отправлять сообщение "Задача удалена."
* При ошибке запроса выводить в консоль сообщение об ошибке и отправлять пользователю "Ошибка удаления задачи."

**Промты для сайта**

**Настройка сервера и подключение к базе данных  
Напишите код для создания HTTP-сервера на Node.js, который слушает на порту 3000. Используйте mysql2/promise для подключения к базе данных MySQL с параметрами: хост 'localhost', пользователь 'root', пустой пароль, база данных 'todolist'. Создайте пул соединений с базой данных.**

1. **Настройка сессий**

Настройте сессии с помощью cookie-session. Установите имя сессии 'session', ключ 'my\_secret\_key' и максимальный возраст 24 часа.

1. **Проверка аутентификации**

Создайте функцию isAuthenticated, которая проверяет, есть ли в сессии userId. Также напишите асинхронную функцию isAuthenticatedOrToken, которая проверяет сессию или токен из заголовка 'Authorization' (Bearer токен). Если токен присутствует, проверьте его в базе данных в таблице users.

1. **Обработка дедлоков**

Реализуйте функцию executeWithRetry, которая выполняет операцию с повторными попытками при возникновении дедлоков (ошибка 'ER\_LOCK\_DEADLOCK'). Максимальное количество попыток — 3, с задержкой между попытками.

1. **Получение списка задач**

Напишите асинхронную функцию retrieveListItems, которая получает список задач пользователя из таблицы items, отсортированный по order\_index. Функция должна принимать userId и возвращать массив задач.

1. **Получение userId из запроса**

Создайте асинхронную функцию getUserIdFromRequest, которая возвращает userId из сессии или из токена в заголовке 'Authorization'. Если ни сессии, ни токена нет, бросьте ошибку 'Неавторизованный доступ'.

1. **Генерация HTML строк для задач**

Напишите асинхронную функцию getHtmlRows, которая получает список задач пользователя и возвращает HTML-строки для таблицы с задачами, включая кнопки для редактирования, удаления, перемещения вверх и вниз.

1. **Перестройка order\_index**

Реализуйте асинхронную функцию rebuildOrderIndex, которая пересчитывает order\_index для всех задач пользователя, устанавливая их в порядке от 1 до N.

1. **Обработчик запросов**

Напишите асинхронный обработчик запросов handleRequest, который обрабатывает следующие маршруты:

* + GET /login: отправляет файл login.html.
  + POST /login: обрабатывает вход пользователя, проверяет логин и пароль, генерирует токен, обновляет его в базе данных, устанавливает сессию и возвращает JSON с токеном.
  + GET /: если пользователь аутентифицирован, отправляет index.html с заменой placeholders на данные пользователя и задачи.
  + GET /admin.html: если пользователь администратор, отправляет admin.html с данными пользователей.
  + POST /addUser: добавляет нового пользователя (только для администраторов).
  + POST /deleteUser: удаляет пользователя по ID (только для администраторов).
  + POST /editUser: редактирует пользователя (только для администраторов).
  + POST /getPassword: получает пароль пользователя по ID (только для администраторов).
  + POST /add: добавляет задачу (требует аутентификации).
  + POST /delete: удаляет задачу по ID (требует аутентификации).
  + GET /getItem: получает задачу по ID (требует аутентификации).
  + POST /edit: редактирует задачу (требует аутентификации).
  + POST /reorder: изменяет порядок задачи (требует аутентификации).
  + POST /logout: очищает сессию.
  + GET /api/items: возвращает список задач в JSON (требует токен).
  + POST /moveUp: перемещает задачу вверх (требует аутентификации).
  + POST /moveDown: перемещает задачу вниз (требует аутентификации).

1. **Добавление задачи**

Реализуйте асинхронную функцию addItem, которая добавляет новую задачу в таблицу items с order\_index = 0 и затем перестраивает order\_index для всех задач пользователя.

1. **Изменение порядка задачи**

Напишите асинхронную функцию reorderItem, которая изменяет order\_index задачи и корректирует order\_index других задач пользователя соответственно.

1. **Удаление задачи**

Создайте асинхронную функцию deleteItem, которая удаляет задачу по ID и перестраивает order\_index для оставшихся задач.

1. **Обновление задачи**

Реализуйте асинхронную функцию updateItem, которая обновляет текст и order\_index задачи и перестраивает order\_index для всех задач пользователя.

1. **Генерация HTML строк для пользователей**

Напишите асинхронную функцию getUserHtmlRows, которая получает список пользователей из таблицы users и возвращает HTML-строки для таблицы пользователей в админ-панели.

1. **Перемещение задачи вверх**

Реализуйте функцию moveUp, которая перемещает задачу вверх, меняя её order\_index с предыдущей задачей, используя транзакции и блокировку строк для предотвращения гонок.

1. **Перемещение задачи вниз**

Аналогично, напишите функцию moveDown для перемещения задачи вниз.

1. **Запуск сервера**

Создайте HTTP-сервер с помощью http.createServer(handleRequest) и запустите его на порту 3000, выведя в консоль "Server running on port 3000".

****8. Трудности при использовании GPT****

**Использование GPT при написании кода сопряжено с рядом проблем. Оно часто генерирует код, который кажется правильным, но не учитывает крайние случаи или теряет контекст проекта. Отладка такого кода может быть сложной, так как ошибки бывают незаметными на первый взгляд. Чрезмерная зависимость от GPT может затормозить процесс обучения, поскольку разработчики перестают глубоко разбираться в концепциях. Кроме того, код от GPT может плохо интегрироваться с существующими системами или содержать уязвимости из-за несоблюдения лучших практик безопасности.**

****9. Вывод****

**В ходе данной лабораторной работы была достигнута цель создания сайта и бота с общей базой данных с использованием больших языковых моделей. Этот проект способствовал развитию навыков «промптинга» у студентов, а также углубил их понимание работы баз данных и методов взаимодействия между сайтом, ботом и базой данных. Тематика сайта и бота — планировщик задач с возможностью авторизации — была успешно реализована, демонстрируя практическое применение полученных знаний и навыков.**