МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №33

ОТЧЕТ ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

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

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |
| --- | --- | --- | --- |
| Студент гр. № | 3333 |  | Н.М.Сазонов |
|  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

### ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Создать полнофункциональное веб-приложение для управления списком задач с возможностью взаимодействия через Telegram-бота. Реализовать авторизацию, хранение данных в базе и обработку задач с клиентской и серверной сторон.

**Перечень решаемых задач в процессе выполнения работы:**

1. Разработка интерфейса для регистрации, входа и управления задачами пользователем.
2. Создание серверной логики на платформе Node.js для обработки клиентских запросов.
3. Настройка и использование реляционной базы данных SQL для хранения учётных записей и списка задач.
4. Подключение и настройка Telegram-бота для выполнения операций по управлению задачами (добавление, редактирование, удаление, просмотр).
5. Настройка взаимодействия всех компонентов системы и проведение комплексной проверки её работоспособности.

# Пример работы программы

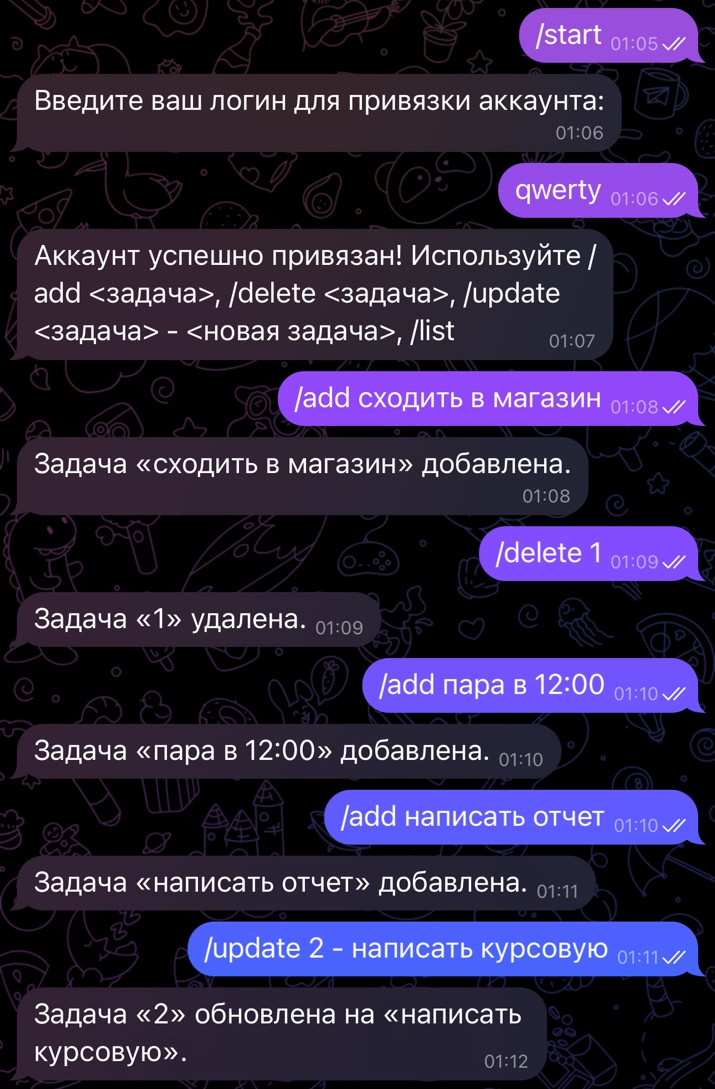
****

Рисунок 1 – интеграция телеграмма

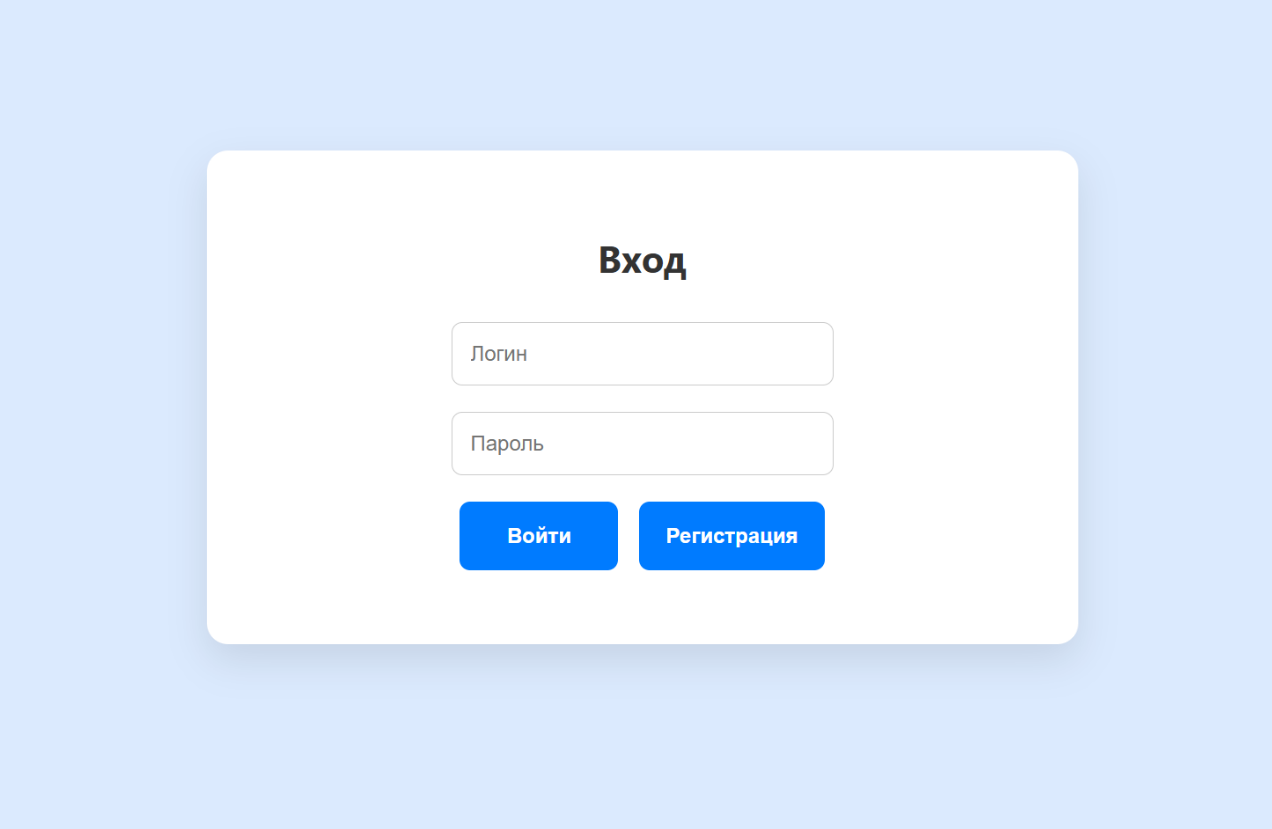


Рисунок 2 – авторизация/регистрация

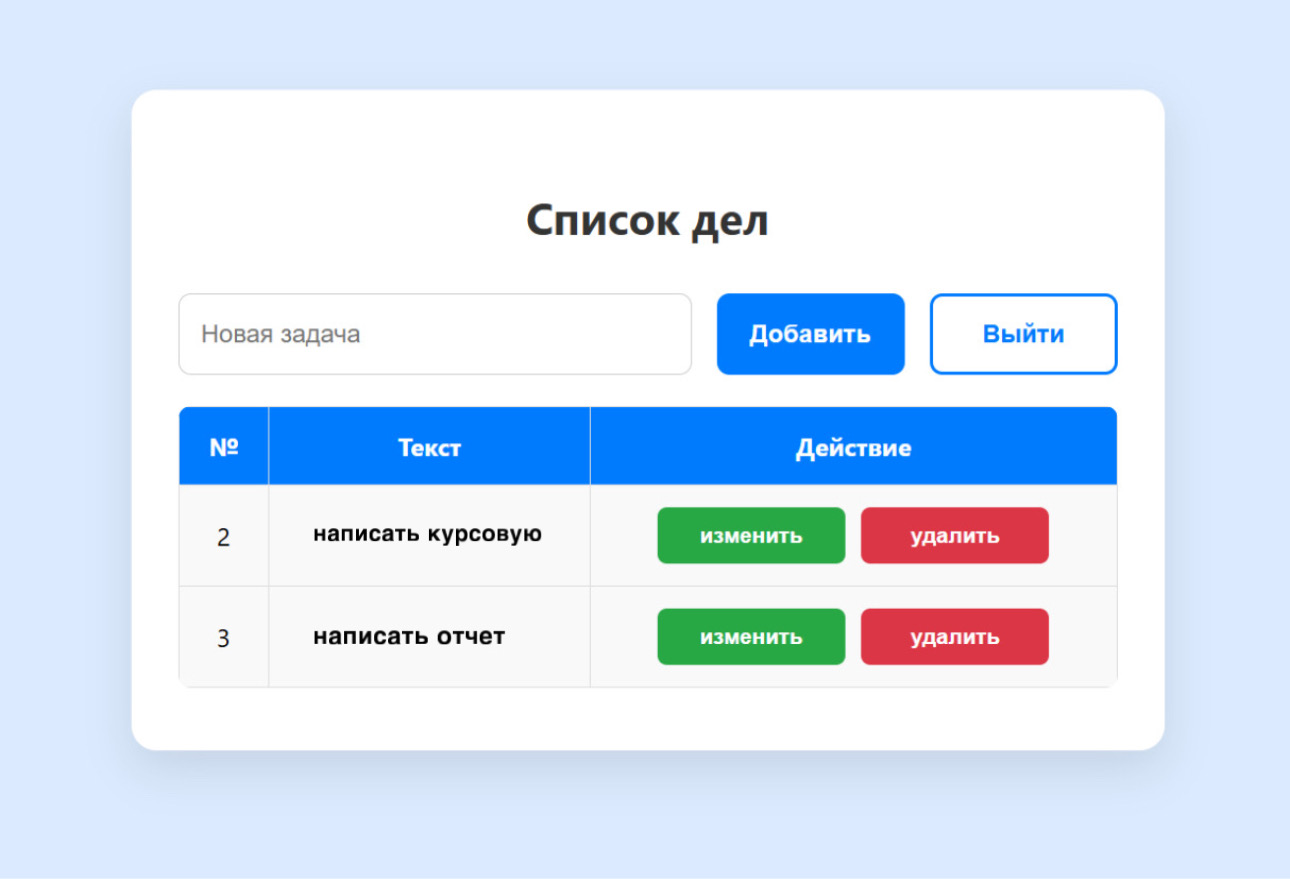


Рисунок 3 – интерфейс

# Вывод:

Разработанное приложение успешно соответствует поставленным требованиям. Реализована стабильная авторизация пользователей и полноценная интеграция с Telegram- ботом, что создаёт базу для последующей доработки и расширения функционала. В ходе выполнения проекта возникали сложности, связанные с тем, что ИИ не всегда точно интерпретировал сформулированные запросы. Эти трудности удалось преодолеть путём уточнения формулировок и последовательного диалога с ИИ, что в конечном итоге позволило получить корректные решения и завершить реализацию.

## Приложение 1 – Код index.js

const express = require('express');

const mysql = require('mysql2/promise'); const session = require('express-session'); const bcrypt = require('bcrypt');

const path = require('path'); const fs = require('fs').promises;

const TelegramBot = require('node-telegram-bot-api'); const app = express();

const port = 3000;

const telegram\_token = '8136358504:AAFYF6NFQtOZ4HvzAfOXcveXivkR\_a44Zv4'; const bot = new TelegramBot(telegram\_token, { polling: true });

const dbConfig = { host: 'Localhost', user: 'root',

password: 'qwe123!@#', database: 'todolist',

};

app.use(express.json()); app.use(session({

secret: 'your-secret-key', resave: false, saveUninitialized: false, cookie: { secure: false }

}));

async function initDatabase() { try {

const connection = await mysql.createConnection({ host: dbConfig.host,

user: dbConfig.user, password: dbConfig.password

});

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

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

console.log('Database todolist selected');

await connection.query(`

CREATE TABLE IF NOT EXISTS users (

id INT AUTO\_INCREMENT PRIMARY KEY, username VARCHAR(50) NOT NULL UNIQUE, password VARCHAR(255) NOT NULL,

telegram\_id VARCHAR(50) UNIQUE

)

`);

console.log('Table users created');

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

CREATE TABLE items (

id INT AUTO\_INCREMENT PRIMARY KEY, text VARCHAR(255) NOT NULL,

user\_id INT,

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

)

`);

console.log('Table items created');

await connection.end();

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

} catch (error) {

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

}

}

async function retrieveListItems(userId) { try {

const connection = await mysql.createConnection(dbConfig);

const query = 'SELECT id, text FROM items WHERE user\_id = ?'; const [rows] = await connection.execute(query, [userId]);

await connection.close(); return rows;

} catch (error) {

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

}

}

async function addListItem(text, userId) { try {

const connection = await mysql.createConnection(dbConfig);

const query = 'INSERT INTO items (text, user\_id) VALUES (?, ?)'; const [result] = await connection.execute(query, [text, userId]); await connection.close();

return { id: result.insertId, text };

} catch (error) {

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

}

}

async function deleteListItem(id, userId) { try {

const connection = await mysql.createConnection(dbConfig);

const query = 'DELETE FROM items WHERE id = ? AND user\_id = ?'; const [result] = await connection.execute(query, [id, userId]);

await connection.close(); return result.affectedRows > 0;

} catch (error) {

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

}

}

async function updateListItem(id, newText, userId) { try {

const connection = await mysql.createConnection(dbConfig);

const query = 'UPDATE items SET text = ? WHERE id = ? AND user\_id = ?'; const [result] = await connection.execute(query, [newText, id, userId]);

await connection.close(); return result.affectedRows > 0;

} catch (error) {

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

}

}

async function getUserByTelegramId(telegramId) { try {

const connection = await mysql.createConnection(dbConfig);

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

await connection.close(); return rows[0];

} catch (error) {

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

}

}

async function linkTelegramId(username, telegramId) { try {

const connection = await mysql.createConnection(dbConfig);

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

if (userRows.length === 0) { await connection.close();

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

Зарегистрируйтесь на сайте.` };

}

const [telegramRows] = await connection.execute('SELECT \* FROM users WHERE telegram\_id

= ?', [telegramId]);

if (telegramRows.length > 0) { await connection.close();

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

}

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

await connection.close();

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

success: true,

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

<дело> - <новое дело>, /list.'

};

} catch (error) {

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

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

}

}

async function getHtmlRows(userId) {

const todoItems = await retrieveListItems(userId); return todoItems.map(item => `

<tr>

<td>${item.id}</td>

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

<td>

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

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

</td>

</tr>

`).join('');

}

function isAuthenticated(req, res, next) { if (req.session.user) {

next();

} else { res.redirect('/login');

}

}

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

const html = await fs.readFile(path.join( dirname, 'index.html'), 'utf8'); res.send(html.replace('{{rows}}', ''));

});

app.post('/login', async (req, res) => { const { username, password } = req.body;

if (!username || !password || username.trim().length === 0 || password.trim().length === 0) { return res.status(400).json({ error: 'Логин и пароль не могут быть пустыми' });

}

try {

const connection = await mysql.createConnection(dbConfig);

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

await connection.close();

if (rows.length === 0) {

return res.status(401).json({ error: 'Пользователь не найден' });

}

const user = rows[0];

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

if (match) {

req.session.user = { id: user.id, username: user.username }; return res.json({ message: 'Успешный вход' });

} else {

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

}

} catch (error) { console.error(error);

return res.status(500).json({ error: 'Ошибка сервера' });

}

});

app.post('/register', async (req, res) => { const { username, password } = req.body;

if (!username || !password || username.trim().length === 0 || password.trim().length === 0) { return res.status(400).json({ error: 'Логин и пароль не могут быть пустыми' });

}

try {

const connection = await mysql.createConnection(dbConfig); const hashedPassword = await bcrypt.hash(password, 10);

await connection.execute('INSERT INTO users (username, password) VALUES (?, ?)', [username, hashedPassword]);

await connection.close();

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

} catch (error) { console.error(error);

return res.status(400).json({ error: 'Пользователь уже существует или ошибка сервера' });

}

});

app.get('/logout', (req, res) => { req.session.destroy(); res.redirect('/login');

});

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

const html = await fs.readFile(path.join( dirname, 'index.html'), 'utf8');

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

} catch (err) { console.error(err);

res.status(500).send('Ошибка загрузки страницы');

}

});

app.post('/add', isAuthenticated, async (req, res) => { const { text } = req.body;

if (!text || typeof text !== 'string' || text.trim() === '') {

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

}

try {

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

} catch (err) { console.error(err);

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

}

});

app.delete('/delete', isAuthenticated, async (req, res) => { const id = req.query.id;

if (!id || isNaN(id)) {

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

}

try {

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

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

} else {

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

}

} catch (err) { console.error(err);

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

}

});

app.put('/update', isAuthenticated, async (req, res) => { const id = req.query.id;

const { text } = req.body;

if (!id || isNaN(id) || !text || typeof text !== 'string' || text.trim() === '') { res.status(400).json({ error: 'Некорректный или отсутствующий ID или текст' }); return;

}

try {

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

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

} else {

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

}

} catch (err) { console.error(err);

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

}

});

//бот тг

bot.onText(/\/start/, async (msg) => { const chatId = msg.chat.id;

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

bot.once('message', async (msg) => { const username = msg.text.trim();

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

});

});

bot.onText(/\/add (.+)/, async (msg, match) => { const chatId = msg.chat.id;

const text = match[1].trim();

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

if (!user) {

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

return;

}

try {

await addListItem(text, user.id);

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

} catch (error) {

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

}

});

bot.onText(/\/delete (.+)/, async (msg, match) => { const chatId = msg.chat.id;

const text = match[1].trim();

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

if (!user) {

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

return;

}

try {

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

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

} else {

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

}

} catch (error) {

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

}

});

bot.onText(/\/update (.+) - (.+)/, async (msg, match) => { const chatId = msg.chat.id;

const oldText = match[1].trim(); const newText = match[2].trim();

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

if (!user) {

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

return;

}

try {

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

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

} else {

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

}

} catch (error) {

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

}

});

bot.onText(/\/list/, async (msg) => { const chatId = msg.chat.id;

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

if (!user) {

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

return;

}

try {

const items = await retrieveListItems(user.id); if (items.length === 0) {

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

}

const message = items.map(item => `${item.id}. ${item.text}`).join('\n'); bot.sendMessage(chatId, `Ваши задачи:\n${message}`);

} catch (error) {

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

}

});

initDatabase().then(() => {

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

}).catch(err => {

console.error('Failed to initialize database and start server:', err); process.exit(1);

});

## Приложение 2 – Код программы index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>To-Do List</title>

<link rel="stylesheet" href="/styles.css">

<style>

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 20px;

}

header {

display: flex;

justify-content: space-between;

align-items: center;

margin-bottom: 20px;

padding-bottom: 10px;

border-bottom: 1px solid #ddd;

}

#todoList {

border-collapse: collapse;

width: 70%;

margin: 0 auto;

}

#todoList th, #todoList td {

border: 1px solid #ddd;

padding: 8px;

text-align: left;

}

#todoList th {

background-color: #f0f0f0;

}

.add-form {

margin: 20px auto;

width: 70%;

text-align: center;

}

.add-form input[type="text"] {

padding: 8px;

width: 70%;

}

.add-form button {

padding: 8px 15px;

margin-left: 5px;

}

.edit-mode .item-text {

padding: 0;

}

.edit-input {

width: 70%;

padding: 5px;

}

.error {

color: red;

margin: 10px 0;

text-align: center;

}

button {

cursor: pointer;

}

.delete-btn {

background: #ff4444;

color: white;

border: none;

border-radius: 50%;

width: 25px;

height: 25px;

}

.edit-btn, .save-btn, .cancel-btn {

padding: 5px 10px;

margin: 0 2px;

}

.user-info {

font-size: 0.9em;

}

</style>

</head>

<body>

<header>

<h2>To-Do List</h2>

<div class="user-info">

Welcome, <span id="username"></span> | <a href="/logout">Logout</a>

</div>

</header>

<table id="todoList">

<thead>

<tr>

<th>ID</th>

<th>Text</th>

<th>Actions</th>

</tr>

</thead>

<tbody id="listBody">

{{rows}}

</tbody>

</table>

<div class="add-form">

<input type="text" id="newItem" placeholder="Enter new item">

<button onclick="addItem()">Add</button>

</div>

<script>

// Функция добавления новой задачи

async function addItem() {

const newItemInput = document.getElementById('newItem');

const newItemText = newItemInput.value.trim();

if (newItemText) {

try {

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

method: 'POST',

headers: {

'Content-Type': 'application/x-www-form-urlencoded',

},

body: `text=${encodeURIComponent(newItemText)}`

});

if (response.redirected) {

window.location.reload();

}

} catch (error) {

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

alert('Failed to add item');

}

newItemInput.value = '';

}

}

// Обработчики для кнопок удаления и редактирования

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

// Установка имени пользователя

const cookies = document.cookie.split(';').reduce((cookies, cookie) => {

const [name, value] = cookie.split('=').map(c => c.trim());

cookies[name] = value;

return cookies;

}, {});

if (cookies.username) {

document.getElementById('username').textContent = cookies.username;

}

// Обработчик кликов по таблице

document.getElementById('listBody').addEventListener('click', async (e) => {

// Удаление задачи

if (e.target.classList.contains('delete-btn')) {

const id = e.target.getAttribute('data-id');

if (confirm('Are you sure you want to delete this item?')) {

try {

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

method: 'POST',

headers: {

'Content-Type': 'application/x-www-form-urlencoded',

},

body: `id=${id}`

});

if (response.redirected) {

window.location.reload();

}

} catch (error) {

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

alert('Failed to delete item');

}

}

}

// Редактирование задачи

if (e.target.classList.contains('edit-btn')) {

startEdit(e.target.closest('tr'));

}

});

});

// Функции для редактирования задачи

function startEdit(row) {

const id = row.dataset.id;

const textCell = row.querySelector('.item-text');

const originalText = textCell.textContent;

// Переход в режим редактирования

row.classList.add('edit-mode');

textCell.innerHTML = `

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

<button class="save-btn">Save</button>

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

`;

// Фокусировка на поле ввода

textCell.querySelector('.edit-input').focus();

// Обработчик сохранения

textCell.querySelector('.save-btn').addEventListener('click', async () => {

const newText = textCell.querySelector('.edit-input').value.trim();

if (newText && newText !== originalText) {

try {

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

method: 'POST',

headers: {

'Content-Type': 'application/x-www-form-urlencoded',

},

body: `id=${id}&text=${encodeURIComponent(newText)}`

});

if (response.ok) {

const result = await response.json();

if (result.success) {

window.location.reload();

} else {

alert('Error updating item');

}

}

} catch (error) {

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

alert('Failed to update item');

}

}

cancelEdit(row, originalText);

});

// Обработчик отмены

textCell.querySelector('.cancel-btn').addEventListener('click', () => {

cancelEdit(row, originalText);

});

}

function cancelEdit(row, originalText) {

row.classList.remove('edit-mode');

const textCell = row.querySelector('.item-text');

textCell.textContent = originalText;

}

</script>

</body>

</html>

## Приложение 3 – Код программы – init.sql

CREATE DATABASE IF NOT EXISTS todolist;

USE todolist;

CREATE TABLE IF NOT EXISTS users (

id INT AUTO\_INCREMENT PRIMARY KEY, username VARCHAR(50) NOT NULL UNIQUE, password VARCHAR(255) NOT NULL,

telegram\_id VARCHAR(50) UNIQUE

);

CREATE TABLE IF NOT EXISTS items (

id INT AUTO\_INCREMENT PRIMARY KEY, text VARCHAR(255) NOT NULL,

user\_id INT,

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

);