

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего
образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №33

ОТЧЁТ ЗАЩИЩЁН С ОЦЕНКОЙ _____

ПРЕПОДАВАТЕЛЬ

Старший преподаватель

Жиданов К. А.

должность, уч. степень, звание

подпись, дата

инициалы, фамилия

ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №1

по курсу: Технологии и методы программирования

СТУДЕНТ
ГР. №

3331

Сазонов Д. А.

номер группы

подпись, дата

инициалы, фамилия

Санкт-Петербург
2025

Введение

В рамках данной лабораторной работы был разработан веб-сервис для управления списком дел (to-do list) с использованием Node.js и MySQL. Приложение позволяет пользователю авторизоваться, просматривать, добавлять, редактировать и удалять задачи через веб-интерфейс. Кроме того, реализована интеграция с Telegram-ботом, что обеспечивает возможность управления списком дел прямо из мессенджера. Такой подход демонстрирует современные методы создания интерактивных веб-приложений и их интеграции с внешними сервисами.

Структура проекта

Структура проекта представлена на рисунке 1.

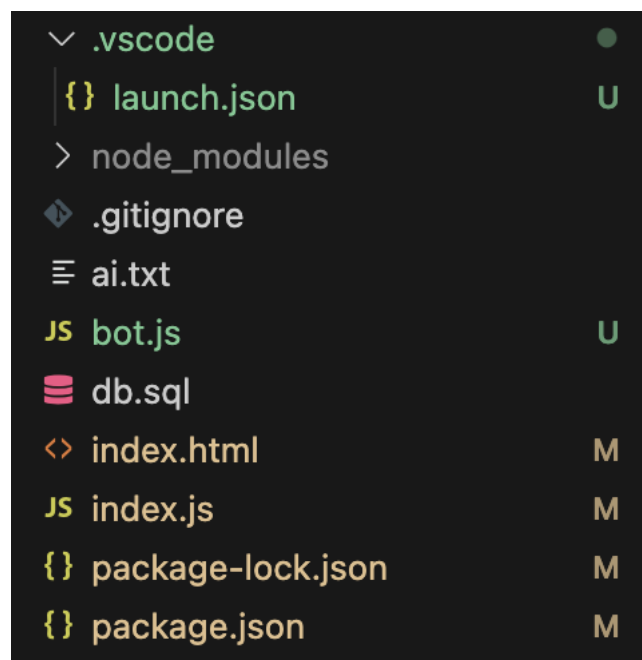


Рисунок 1 – структура проекта

Краткое описание основных файлов:

1. index.js

Главный серверный файл на Node.js.

Реализует:

- запуск HTTP-сервера,

- обработку маршрутов для авторизации, получения, добавления, редактирования и удаления задач,
- работу с MySQL-базой данных для хранения списка дел,
- простую систему сессий для авторизации пользователей.

2. index.html

Основной клиентский файл (веб-страница).

Реализует:

- интерфейс для просмотра, добавления, редактирования и удаления задач,
- форму авторизации пользователя,
- взаимодействие с сервером через fetch-запросы к REST API.

3. bot.js

Файл Telegram-бота на Node.js.

Реализует:

- авторизацию пользователя через Telegram,
- просмотр, добавление, редактирование и удаление задач через команды бота,
- интеграцию с сервером (index.js) для работы с тем же списком дел.

4. package.json

Файл, который управляет зависимостями проекта, например:

- telegram API
- mysql2
- node-fetch

Код проекта

Код файла index.js

```
const http = require('http');  
const fs = require('fs');  
const path = require('path');
```

```
const mysql = require('mysql2/promise');
const crypto = require('crypto');

const PORT = 3000;

// Настройки подключения к БД
const dbConfig = {
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'todolist',
};

// Простое хранение сессий в памяти
const SESSIONS = {};

// Создать новую сессию
function createSession() {
  const sessionId =
crypto.randomBytes(16).toString('hex');
  SESSIONS[sessionId] = { authenticated: true };
  return sessionId;
}

// Получить сессию из cookie
function getSession(req) {
  const cookie = req.headers.cookie;
  if (!cookie) return null;
  const match = cookie.match(/session=([a-f0-9]+)/);
```

```

    if (match && SESSIONS[match[1]]) {
        return { id: match[1], data: SESSIONS[match[1]] };
    }
    return null;
}

// Установить cookie сессии
function setSessionCookie(res, sessionId) {
    res.setHeader('Set-Cookie', `session=${sessionId};
HttpOnly; Path=/`);
}

// Удалить сессию
function destroySession(sessionId) {
    delete SESSIONS[sessionId];
}

// Получить все элементы из БД
async function retrieveListItems() {
    const connection = await
mysql.createConnection(dbConfig);
    const query = 'SELECT id, text FROM items ORDER BY id';
    const [rows] = await connection.execute(query);
    await connection.end();
    return rows;
}

// Добавить элемент в БД
async function addItemToDB(text) {

```

```
    const connection = await
mysql.createConnection(dbConfig);
    const query = 'INSERT INTO items (text) VALUES (?)';
    const [result] = await connection.execute(query,
[text]);
    await connection.end();
    return result.insertId;
}
```

// Обновить элемент в БД

```
async function updateItemInDB(id, text) {
    const connection = await
mysql.createConnection(dbConfig);
    const query = 'UPDATE items SET text = ? WHERE id = ?';
    const [result] = await connection.execute(query, [text,
id]);
    await connection.end();
    return result.affectedRows > 0;
}
```

// Удалить элемент из БД

```
async function deleteItemFromDB(id) {
    const connection = await
mysql.createConnection(dbConfig);
    const query = 'DELETE FROM items WHERE id = ?';
    const [result] = await connection.execute(query, [id]);
    await connection.end();
    return result.affectedRows > 0;
}
```

```
// Обработчик запросов
async function handleRequest(req, res) {

    if (req.url === '/login' && req.method === 'POST') {
        // Авторизация
        let body = '';
        req.on('data', chunk => body += chunk);
        req.on('end', () => {
            try {
                const { login, password } =
JSON.parse(body);
                if (login === 'danil' && password ===
'guap3331') {

                    const sessionId = createSession();
                    setSessionCookie(res, sessionId);
                    res.writeHead(200, { 'Content-Type':
'application/json' });
                    res.end(JSON.stringify({ success: true
})));
                } else {
                    res.writeHead(200, { 'Content-Type':
'application/json' });
                    res.end(JSON.stringify({ success: false
})));
                }
            } catch {
                res.writeHead(400, { 'Content-Type':
'application/json' });
            }
        });
    }
}
```

```

        res.end(JSON.stringify({ success: false }));
    }
    });
    return;
}

if (req.url === '/check-auth' && req.method === 'GET') {
    const session = getSession(req);
    res.writeHead(200, { 'Content-Type':
'application/json' });
    res.end(JSON.stringify({ authenticated: !(session
&& session.data.authenticated) }));
    return;
}

// Защищённый маршрут: получить список дел
if (req.url === '/items' && req.method === 'GET') {
    const session = getSession(req);
    if (!session || !session.data.authenticated) {
        res.writeHead(401, { 'Content-Type':
'application/json' });
        res.end(JSON.stringify({ error: 'Unauthorized'
}));
        return;
    }
    try {
        const items = await retrieveListItems();
        res.writeHead(200, { 'Content-Type':
'application/json' });

```



```

        res.end(JSON.stringify(items));
    } catch (err) {
        res.writeHead(500, { 'Content-Type':
'application/json' });
        res.end(JSON.stringify({ error: 'DB error' }));
    }
    return;
}

// Добавление нового элемента
if (req.url === '/items' && req.method === 'POST') {
    const session = getSession(req);
    if (!session || !session.data.authenticated) {
        res.writeHead(401, { 'Content-Type':
'application/json' });
        res.end(JSON.stringify({ success: false, error:
'Unauthorized' }));
        return;
    }
    let body = '';
    req.on('data', chunk => body += chunk);
    req.on('end', async () => {
        try {
            const { text } = JSON.parse(body);
            if (!text || typeof text !== 'string' ||
!text.trim()) {
                res.writeHead(400, { 'Content-Type':
'application/json' });

```

```

        res.end(JSON.stringify({ success: false,
error: 'Invalid text' }));
        return;
    }
    const id = await addItemToDB(text.trim());
    res.writeHead(200, { 'Content-Type':
'application/json' });
    res.end(JSON.stringify({ success: true, id
}));
    } catch {
        res.writeHead(500, { 'Content-Type':
'application/json' });
        res.end(JSON.stringify({ success: false,
error: 'Server error' }));
    }
    });
    return;
}

```

```

// Обновление элемента
if (req.url.startsWith('/items/') && req.method ===
'PUT') {
    const session = getSession(req);
    if (!session || !session.data.authenticated) {
        res.writeHead(401, { 'Content-Type':
'application/json' });
        res.end(JSON.stringify({ success: false, error:
'Unauthorized' }));
        return;
    }
}

```

```

    }
    const id = parseInt(req.url.split('/')[2], 10);
    if (!id) {
        res.writeHead(400, { 'Content-Type':
'application/json' });
        res.end(JSON.stringify({ success: false, error:
'Invalid ID' }));
        return;
    }
    let body = '';
    req.on('data', chunk => body += chunk);
    req.on('end', async () => {
        try {
            const { text } = JSON.parse(body);
            if (!text || typeof text !== 'string' ||
!text.trim()) {
                res.writeHead(400, { 'Content-Type':
'application/json' });
                res.end(JSON.stringify({ success: false,
error: 'Invalid text' }));
                return;
            }
            const updated = await updateItemInDB(id,
text.trim());
            res.writeHead(200, { 'Content-Type':
'application/json' });
            res.end(JSON.stringify({ success: updated
}));
        } catch {

```

```

        res.writeHead(500, { 'Content-Type':
'application/json' });
        res.end(JSON.stringify({ success: false,
error: 'Server error' }));
    }
    });
    return;
}

// Удаление элемента
if (req.url.startsWith('/items/') && req.method ===
'DELETE') {
    const session = getSession(req);
    if (!session || !session.data.authenticated) {
        res.writeHead(401, { 'Content-Type':
'application/json' });
        res.end(JSON.stringify({ success: false, error:
'Unauthorized' }));
        return;
    }
    const id = parseInt(req.url.split('/')[2], 10);
    if (!id) {
        res.writeHead(400, { 'Content-Type':
'application/json' });
        res.end(JSON.stringify({ success: false, error:
'Invalid ID' }));
        return;
    }
    try {

```

```

        const deleted = await deleteItemFromDB(id);
        res.writeHead(200, { 'Content-Type':
'application/json' });
        res.end(JSON.stringify({ success: deleted }));
    } catch {
        res.writeHead(500, { 'Content-Type':
'application/json' });
        res.end(JSON.stringify({ success: false, error:
'Server error' }));
    }
    return;
}

// Главная страница – отдаём index.html с пустым
{{rows}} (список подгружается через JS)
if ((req.url === '/' || req.url === '/index.html') &&
req.method === 'GET') {
    try {
        const html = await
fs.promises.readFile(path.join(__dirname, 'index.html'),
'utf8');
        const processedHtml = html.replace('{{rows}}',
'');
        res.writeHead(200, { 'Content-Type': 'text/html'
});
        res.end(processedHtml);
    } catch (err) {
        res.writeHead(500, { 'Content-Type':
'text/plain' });

```

```

        res.end('Error loading index.html');
    }
    return;
}

// Если ничего не подошло – 404
res.writeHead(404, { 'Content-Type': 'text/plain' });
res.end('Route not found');
}

const server = http.createServer(handleRequest);

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

```

Код файла index.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1" />
    <title>To-Do List</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }
        #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;
    }
    #todoList th:first-child, #todoList th:last-child {
        width: 5%;
    }
    #todoList th:nth-child(2) {
        width: 90%;
    }
    .add-form {
        margin-top: 20px;
        width: 70%;
        margin: 20px auto;
    }
    .add-form input[type="text"] {
        padding: 8px;
        width: 70%;
    }
    .add-form button {
        padding: 8px;
```

```
        width: 20%;
    }
    .edit-input {
        width: 100%;
        padding: 8px;
        border: 1px solid #ccc;
    }
    /* Стили для формы логина */
    #loginForm {
        max-width: 300px;
        margin: 100px auto;
        text-align: center;
    }
    #loginForm input {
        width: 90%;
        padding: 8px;
        margin: 8px 0;
        box-sizing: border-box;
    }
    #loginForm button {
        padding: 8px 16px;
        width: 100%;
    }
    #loginError {
        color: red;
        margin-top: 10px;
        min-height: 18px;
    }
</style>
```



```

</head>
<body>

<!-- Форма логина -->
<div id="loginForm" style="display: none;">
    <h2>Login</h2>
    <input type="text" id="login" placeholder="Login"
autocomplete="username" />
    <input type="password" id="password"
placeholder="Password" autocomplete="current-password" />
    <button onclick="submitLogin()">Login</button>
    <div id="loginError"></div>
</div>

<!-- Основной To-Do List -->
<div id="todoApp" style="display: none;">

    <h2 style="text-align: center;">To-Do List</h2>

    <table id="todoList">
        <thead>
            <tr>
                <th>Number</th>
                <th>Text</th>
                <th>Action</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>

</div>

<script>
    // Переменные для списка дел
    let items = [];
    let nextId = 1;
    let editing = null;

    // Функция отображения формы логина
    function showLogin() {
        document.getElementById('loginForm').style.display =
'';
        document.getElementById('todoApp').style.display =
'none';
        document.getElementById('loginError').textContent =
'';
    }

    // Функция отображения To-Do List
    function showTodo() {
```

```
        document.getElementById('loginForm').style.display =  
'none';  
        document.getElementById('todoApp').style.display =  
'';  
        fetchItemsFromServer();  
    }  
  
    // Проверка авторизации при загрузке страницы  
    window.onload = function() {  
        fetch('/check-auth')  
            .then(res => res.json())  
            .then(data => {  
                if (data.authenticated) {  
                    showTodo();  
                } else {  
                    showLogin();  
                }  
            })  
            .catch(() => showLogin());  
    }  
  
    // Отправка логина и пароля на сервер  
    function submitLogin() {  
        const login =  
document.getElementById('login').value.trim();  
        const password =  
document.getElementById('password').value.trim();  
        if (!login || !password) {
```

```
document.getElementById('loginError').textContent = 'Please  
enter login and password';
```

```
    return;  
  }  
  fetch('/login', {  
    method: 'POST',  
    headers: {'Content-Type': 'application/json'},  
    body: JSON.stringify({ login, password })  
  })  
  .then(res => res.json())  
  .then(data => {  
    if (data.success) {  
      showTodo();  
    } else {
```

```
document.getElementById('loginError').textContent = 'Invalid  
login or password';
```

```
    }  
  })  
  .catch(() => {
```

```
document.getElementById('loginError').textContent = 'Server  
error';
```

```
    });  
  }
```

```
// Загрузка списка дел с сервера  
function fetchItemsFromServer() {
```

```

    fetch('/items')
      .then(res => res.json())
      .then(data => {
        items = data;
        nextId = items.length > 0 ?
Math.max(...items.map(i => i.id)) + 1 : 1;
        renderList();
      })
      .catch(() => {
        alert('Failed to load to-do items from
server');
      });
  }

```

```

// Отрисовка списка дел
function renderList() {
  const listBody =
document.getElementById('listBody');
  listBody.innerHTML = '';

  items.forEach((item, index) => {
    const row = document.createElement('tr');
    row.innerHTML = `
      <td>${index + 1}</td>
      <td class="item-text">${item.text}</td>
      <td>
        <button
onclick="editItem(${index})">Edit</button>

```

```
        <button  
onclick="removeItem(${index})">Remove</button>  
    </td>
```

```
    `;  
    listBody.appendChild(row);  
  });  
}
```

```
// Добавление нового элемента  
function addItem() {  
    const newItemInput =  
document.getElementById('newItem');  
    const newItemText = newItemInput.value.trim();  
  
    if (!newItemText) return;  
  
    // Отправляем на сервер  
    fetch('/items', {  
        method: 'POST',  
        headers: {'Content-Type': 'application/json'},  
        body: JSON.stringify({ text: newItemText })  
    })  
    .then(res => res.json())  
    .then(data => {  
        if (data.success) {  
            items.push({ id: data.id, text: newItemText  
});  
  
            newItemInput.value = '';  
            renderList();
```

```

        } else {
            alert('Failed to add item');
        }
    })
    .catch(() => alert('Server error'));
}

```

// Удаление элемента

```

function removeItem(index) {
    const item = items[index];
    fetch(`/items/${item.id}`, { method: 'DELETE' })
        .then(res => res.json())
        .then(data => {
            if (data.success) {
                items.splice(index, 1);
                renderList();
            } else {
                alert('Failed to remove item');
            }
        })
        .catch(() => alert('Server error'));
}

```

// Редактирование элемента

```

function editItem(index) {
    if (editing !== null) return;

    const row = document.querySelectorAll('#listBody
tr')[index];

```

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

const input = document.createElement('input');
input.type = 'text';
input.className = 'edit-input';
input.value = text;

textCell.textContent = '';
textCell.appendChild(input);
input.focus();

editing = { index, input, originalText: text };

input.addEventListener('keyup', (e) => {
    if (e.key === 'Enter') saveEdit(index);
    else if (e.key === 'Escape') cancelEdit();
});

input.addEventListener('blur', () =>
saveEdit(index));
}

// Сохранение редактирования
function saveEdit(index) {
    if (!editing) return;

    const { input, originalText } = editing;
    const newValue = input.value.trim();
```



```
if (newValue === originalText) {
    cancelEdit();
    return;
}

if (!newValue) {
    // Если пустое, удаляем
    removeItem(index);
    cancelEdit();
    return;
}

const item = items[index];
fetch(`/items/${item.id}`, {
    method: 'PUT',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({ text: newValue })
})
.then(res => res.json())
.then(data => {
    if (data.success) {
        items[index].text = newValue;
        renderList();
    } else {
        alert('Failed to update item');
    }
    cancelEdit();
})
```

```

        .catch(() => {
            alert('Server error');
            cancelEdit();
        });
    }

    // Отмена редактирования
    function cancelEdit() {
        if (editing) {
            const { input } = editing;
            if (input.parentNode) {
                input.parentNode.textContent =
editing.originalText;
            }
            editing = null;
        }
    }
}
</script>

</body>
</html>

```

Код файла bot.js

```

const TelegramBot = require('node-telegram-bot-api');
const fetch = require('node-fetch');
const TOKEN =
'7111754190:AAHHUI0RFlopmk6knGdBdj5FbK0ADseIK1Q';
const API_URL = 'http://localhost:3000';

```

```
const userStates = {};  
const userCookies = {};  
  
const bot = new TelegramBot(TOKEN, { polling: true });  
  
// Старт работы с ботом  
bot.onText(/\sstart/, (msg) => {  
    const chatId = msg.chat.id;  
    userStates[chatId] = { step: 'login' };  
    bot.sendMessage(chatId, 'Пожалуйста, введите логин:');  
});  
  
// Обработка всех текстовых сообщений  
bot.on('message', async (msg) => {  
    const chatId = msg.chat.id;  
    const text = msg.text;  
  
    // Игнорируем команды  
    if (text.startsWith('/')) return;  
  
    const state = userStates[chatId];  
    if (!state) {  
        bot.sendMessage(chatId, 'Напишите /start для  
начала.');        return;  
    }  
  
    // === Логин/Пароль ===  
    if (state.step === 'login') {
```

```
    state.login = text;
    state.step = 'password';
    bot.sendMessage(chatId, 'Введите пароль:');
    return;
}

if (state.step === 'password') {
    state.password = text;
    // Пытаемся авторизоваться на сервере
    try {
        const response = await fetch(`${API_URL}/login`,
{
            method: 'POST',
            headers: { 'Content-Type':
'application/json' },
            body: JSON.stringify({ login: state.login,
password: state.password }),
        });

        const data = await response.json();
        const setCookie = response.headers.get('set-
cookie');

        if (data.success && setCookie) {
            userCookies[chatId] = setCookie;
            state.step = 'authenticated';
            bot.sendMessage(
                chatId,
```

```

        'Авторизация успешна! Вот ваши
задачи:\n\n' +

        'Доступные команды:\n' +
        '/add – добавить задачу\n' +
        '/edit – редактировать задачу\n' +
        '/delete – удалить задачу\n' +
        '/tasks – показать задачи'

    );
    await sendTasks(chatId);
  } else {
    bot.sendMessage(chatId, 'Неверный логин или
пароль. Попробуйте снова.\nВведите логин:');
    state.step = 'login';
  }
} catch (e) {
  bot.sendMessage(chatId, 'Ошибка сервера.
Попробуйте позже.');
```

```

    state.step = 'login';
  }
  return;
}

// === Ожидание ввода для добавления задачи ===
if (state.step === 'add_item') {
  const newText = text.trim();
  if (!newText) {
    bot.sendMessage(chatId, 'Текст задачи не может
быть пустым. Введите текст задачи:');
    return;
  }

```

```

    }
    try {
        const response = await fetch(`${API_URL}/items`,
{
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Cookie': userCookies[chatId],
            },
            body: JSON.stringify({ text: newText }),
        });
        const data = await response.json();
        if (data.success) {
            bot.sendMessage(chatId, 'Задача
добавлена!');

            state.step = 'authenticated';
            await sendTasks(chatId);
        } else {
            bot.sendMessage(chatId, 'Ошибка при
добавлении задачи.');
```

```

// === Ожидание выбора задачи для редактирования ===
if (state.step === 'edit_choose') {
    const num = parseInt(text.trim(), 10);
    if (isNaN(num) || !state.tasks || num < 1 || num >
state.tasks.length) {
        bot.sendMessage(chatId, 'Некорректный номер.
Введите номер задачи для редактирования:');
        return;
    }
    state.editIndex = num - 1;
    state.step = 'edit_text';
    bot.sendMessage(
        chatId,
        `Введите новый текст для
задачи:\n"${state.tasks[state.editIndex].text}"`
    );
    return;
}

```

```

// === Ожидание нового текста для редактирования ===
if (state.step === 'edit_text') {
    const newText = text.trim();
    if (!newText) {
        bot.sendMessage(chatId, 'Текст задачи не может
быть пустым. Введите новый текст:');
        return;
    }
    const task = state.tasks[state.editIndex];
    try {

```

```

        const response = await
fetch(`${API_URL}/items/${task.id}`, {
    method: 'PUT',
    headers: {
        'Content-Type': 'application/json',
        'Cookie': userCookies[chatId],
    },
    body: JSON.stringify({ text: newText }),
});
const data = await response.json();
if (data.success) {
    bot.sendMessage(chatId, 'Задача успешно
изменена!');

    state.step = 'authenticated';
    await sendTasks(chatId);
} else {
    bot.sendMessage(chatId, 'Ошибка при
редактировании задачи.');
```

```

    }
} catch (e) {
    bot.sendMessage(chatId, 'Ошибка сервера при
редактировании задачи.');
```

```

    }
    state.step = 'authenticated';
    return;
}

// === Ожидание выбора задачи для удаления ===
if (state.step === 'delete_choose') {
```



```
    const num = parseInt(text.trim(), 10);
    if (isNaN(num) || !state.tasks || num < 1 || num >
state.tasks.length) {
        bot.sendMessage(chatId, 'Некорректный номер.
Введите номер задачи для удаления:');
        return;
    }
    const task = state.tasks[num - 1];
    try {
        const response = await
fetch(`${API_URL}/items/${task.id}`, {
            method: 'DELETE',
            headers: {
                'Cookie': userCookies[chatId],
            },
        });
        const data = await response.json();
        if (data.success) {
            bot.sendMessage(chatId, 'Задача удалена!');
            state.step = 'authenticated';
            await sendTasks(chatId);
        } else {
            bot.sendMessage(chatId, 'Ошибка при удалении
задачи.');
```

```

        state.step = 'authenticated';
        return;
    }

    // === Уже авторизован, но не в процессе команды ===
    if (state.step === 'authenticated') {
        bot.sendMessage(
            chatId,
            'Доступные команды:\n' +
            '/add – добавить задачу\n' +
            '/edit – редактировать задачу\n' +
            '/delete – удалить задачу\n' +
            '/tasks – показать задачи'
        );
    }
});

// Команда: показать задачи
bot.onText(/\tasks/, async (msg) => {
    const chatId = msg.chat.id;
    const state = userStates[chatId];
    if (!state || state.step !== 'authenticated') {
        bot.sendMessage(chatId, 'Сначала авторизуйтесь через /start');
        return;
    }
    await sendTasks(chatId);
});

```

```
// Команда: добавить задачу
bot.onText(/\//add/, (msg) => {
  const chatId = msg.chat.id;
  const state = userStates[chatId];
  if (!state || state.step !== 'authenticated') {
    bot.sendMessage(chatId, 'Сначала авторизуйтесь через /start');
    return;
  }
  state.step = 'add_item';
  bot.sendMessage(chatId, 'Введите текст новой задачи:');
});
```

```
// Команда: редактировать задачу
bot.onText(/\//edit/, async (msg) => {
  const chatId = msg.chat.id;
  const state = userStates[chatId];
  if (!state || state.step !== 'authenticated') {
    bot.sendMessage(chatId, 'Сначала авторизуйтесь через /start');
    return;
  }
  // Получаем список задач
  const tasks = await getTasks(chatId);
  if (!tasks || tasks.length === 0) {
    bot.sendMessage(chatId, 'Список задач пуст.');
```

```

    let message = 'Выберите номер задачи для
редактирования:\n';
    tasks.forEach((item, i) => {
        message += `${i + 1}. ${item.text}\n`;
    });
    state.tasks = tasks;
    state.step = 'edit_choose';
    bot.sendMessage(chatId, message);
});

// Команда: удалить задачу
bot.onText(/\/delete/, async (msg) => {
    const chatId = msg.chat.id;
    const state = userStates[chatId];
    if (!state || state.step !== 'authenticated') {
        bot.sendMessage(chatId, 'Сначала авторизуйтесь через
/start');
        return;
    }
    // Получаем список задач
    const tasks = await getTasks(chatId);
    if (!tasks || tasks.length === 0) {
        bot.sendMessage(chatId, 'Список задач пуст.');
```

```
state.tasks = tasks;
state.step = 'delete_choose';
bot.sendMessage(chatId, message);
});

// Получение и отправка списка задач пользователю
async function sendTasks(chatId) {
  const tasks = await getTasks(chatId);
  if (!tasks) {
    bot.sendMessage(chatId, 'Ошибка при получении задач.');
```

задач.');

```
    return;
  }
  if (tasks.length === 0) {
    bot.sendMessage(chatId, 'Список задач пуст.');
```

return;

```
  }
  let msg = 'Ваши задачи:\n';
  tasks.forEach((item, i) => {
    msg += `${i + 1}. ${item.text}\n`;
  });
  bot.sendMessage(chatId, msg);
}
```

```
// Получить задачи с сервера
async function getTasks(chatId) {
  try {
    const response = await fetch(`${API_URL}/items`, {
      method: 'GET',
```

```

        headers: {
            'Cookie': userCookies[chatId],
        }
    });
    if (response.status === 401) {
        bot.sendMessage(chatId, 'Ваша сессия устарела. Пожалуйста, авторизуйтесь снова через /start.');
```

Пожалуйста, авторизуйтесь снова через /start.');

```

        userStates[chatId] = { step: 'login' };
        return null;
    }
    const items = await response.json();
    return Array.isArray(items) ? items : [];
} catch (e) {
    return null;
}
}

```

Пример работы программы

Для начала введём неверный логин и пароль, чтобы проверить выдаст ли программа ошибку:

Login

неправильно

••••••

Login

Invalid login or password

Рисунок 2 – ошибка при вводе неверного логина или пароля

Далее введем корректные данные, после чего добавим задачи в наш список на сайте:

To-Do List		
Number	Text	Action
1	Купить хлеб	<div>Edit</div> <div>Remove</div>
2	Встретить брата	<div>Edit</div> <div>Remove</div>
3	Встретить курьера	<div>Edit</div> <div>Remove</div>
<div>Enter new item</div>		<div>Add</div>

Рисунок 3 – добавление задач через сайт

Теперь проверим отображается ли наш список в телеграм-боте:

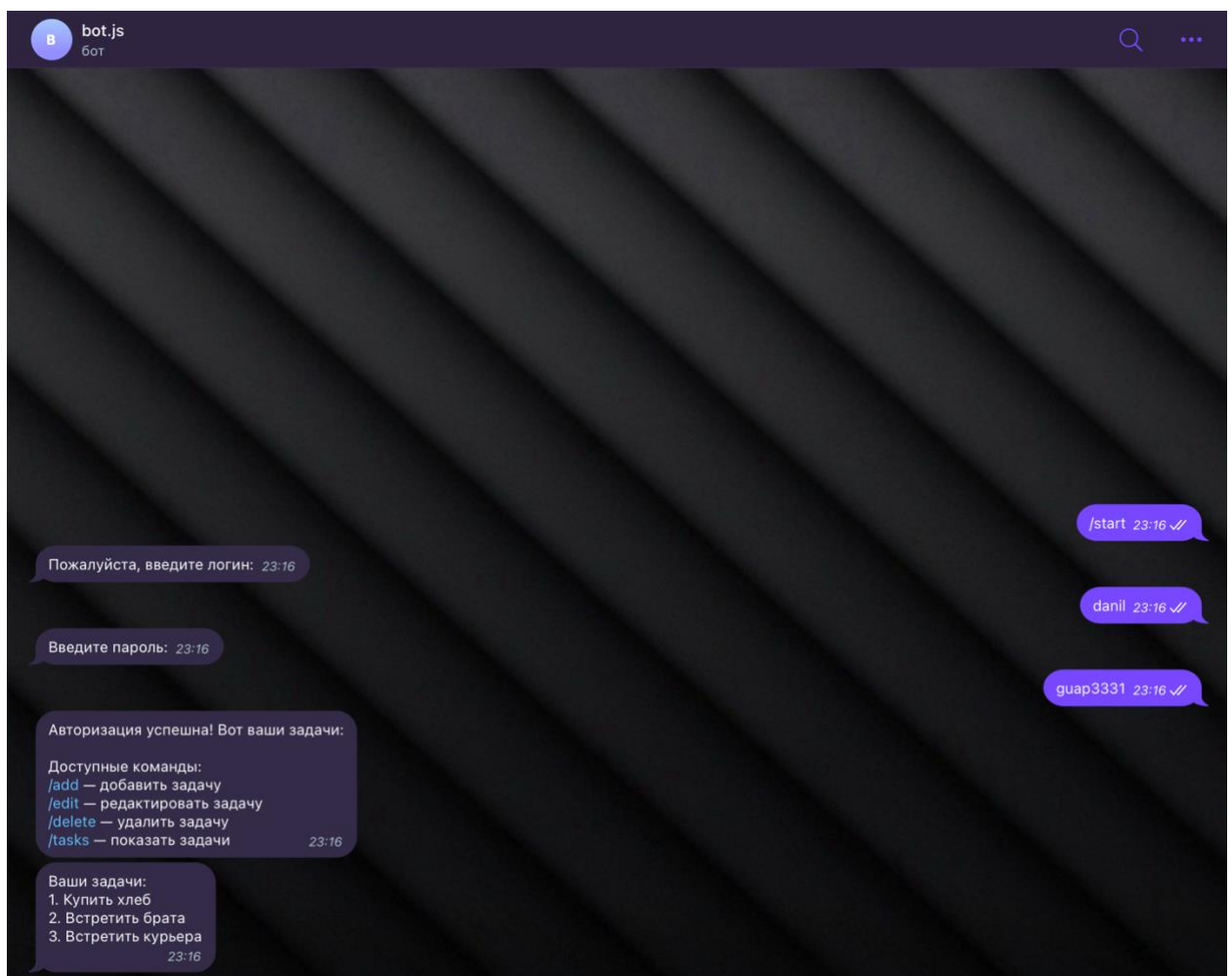


Рисунок 4 – список задач в телеграм-боте

Далее проверим как работают функции редактирования и удаления через сайт, изменим задачу «купить хлеб» на «купить молоко», а после удалим задачу «купить молоко»:

To-Do List

Number	Text	Action
1	Купить молоко	<div>Edit</div> <div>Remove</div>
2	Встретить брата	<div>Edit</div> <div>Remove</div>
3	Встретить курьера	<div>Edit</div> <div>Remove</div>

Enter new item

Add

Рисунок 5 – редактирование задач

To-Do List

Number	Text	Action
1	Встретить брата	<div>Edit</div> <div>Remove</div>
2	Встретить курьера	<div>Edit</div> <div>Remove</div>

Enter new item

Add

Рисунок 6 – удаление задач

Теперь обновим список задач в телеграм-боте, чтобы отобразить изменения, которые мы внесли на сайте:

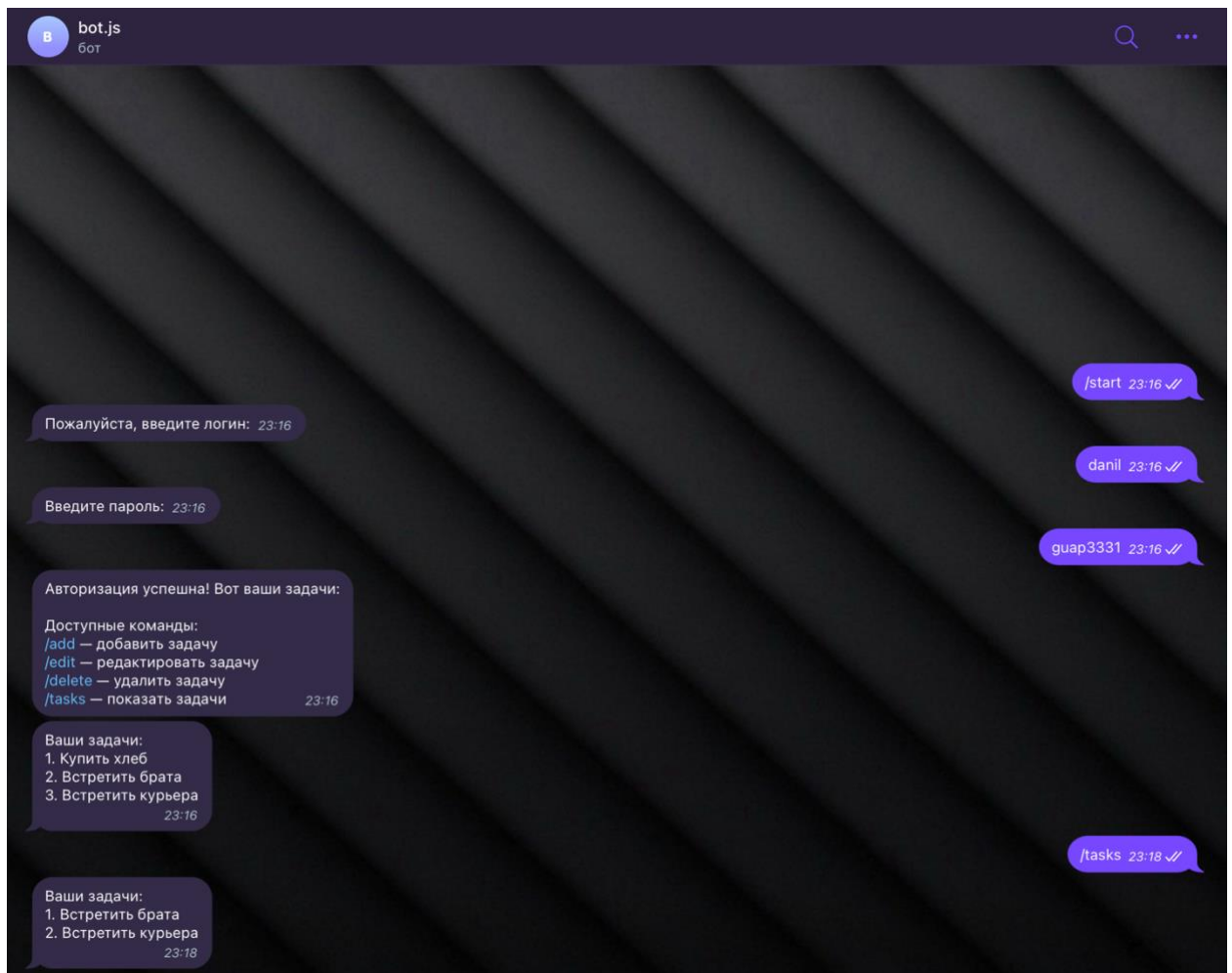


Рисунок 7 – обновлённый список задач в телеграм-боте

Теперь повторим все действия (добавление, удаление и редактирование элементов списка) через телеграм-бот и проверим изменения на сайте:

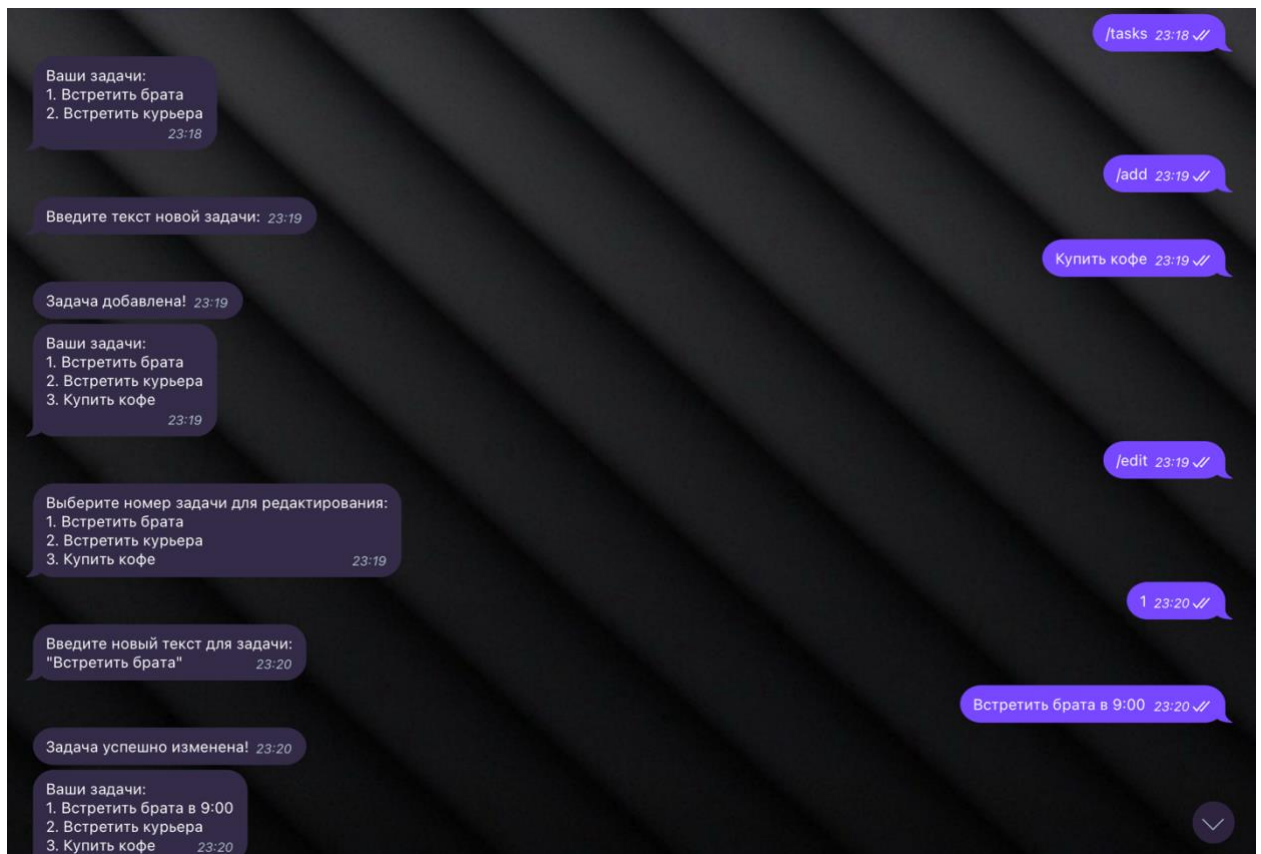


Рисунок 8 – добавление и редактирование элементов списка через телеграм-бот

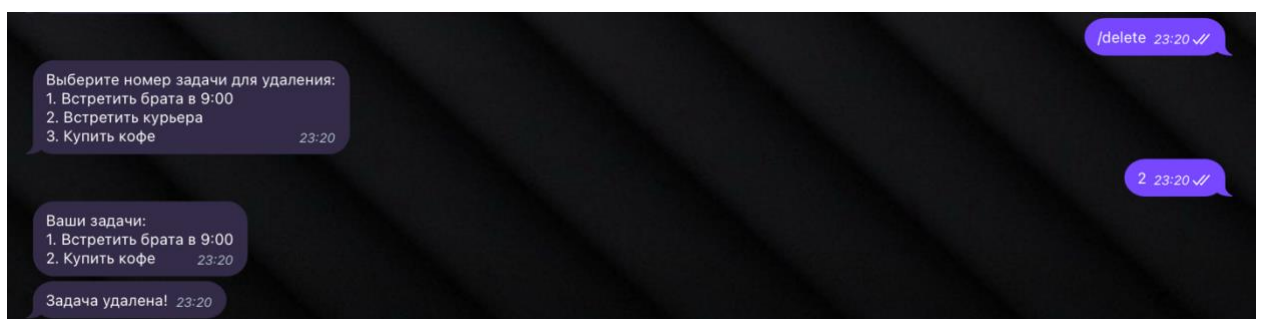


Рисунок 9 – удаление элементов через телеграм-бот

To-Do List

Number	Text	Action
1	Встретить брата в 9:00	<div>Edit</div> <div>Remove</div>
2	Купить кофе	<div>Edit</div> <div>Remove</div>

Рисунок 10 – результаты на сайте

Пример работы с нейросетью

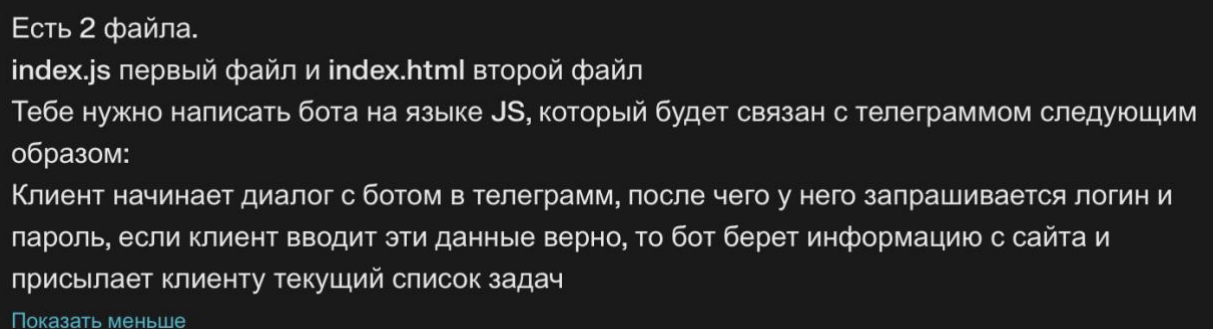
Что я просил сделать нейросеть, примеры:

1. Помочь с запуском и структурой node.js сервера
2. Объяснить, как выполнить добавление, редактирование и удаление элементов списка
3. Создать телеграм-бота, который выводит список задач, синхронизированный с сайтом
4. Помочь с написанием кода для авторизации на сайте и в боте
5. Объяснить код каждого файла для лучшего понимания и написания отчета
6. Проверить готовый проект на наличие ошибок и их исправление
7. Предоставление инструкции по использованию терминала для запуска и настройки сервера, а также базы данных

Как помогла нейросеть:

1. Подробно объясняла ход решения каждой задачи
2. Давала готовые решения с комментариями
3. Подсказывала в чем заключаются ошибки и как их исправить

Пример запроса:



Есть 2 файла.
index.js первый файл и index.html второй файл
Тебе нужно написать бота на языке JS, который будет связан с телеграммом следующим образом:
Клиент начинает диалог с ботом в телеграмм, после чего у него запрашивается логин и пароль, если клиент вводит эти данные верно, то бот берет информацию с сайта и присылает клиенту текущий список задач
[Показать меньше](#)

Рисунок 11 – пример запроса нейросети

На этот запрос нейросеть сразу выдала готовый ответ:

1. Установка зависимостей

Вам понадобится библиотека `node-telegram-bot-api` для работы с Telegram API и `node-fetch` для HTTP-запросов к вашему серверу.

```
bash
npm install node-telegram-bot-api node-fetch@2
```

2. Логика Telegram-бота (bot.js)

```
js
const TelegramBot = require('node-telegram-bot-api');
const fetch = require('node-fetch'); // v2
const TOKEN = 'ВАШ_ТОКЕН_БОТА'; // замените на токен вашего бота

const bot = new TelegramBot(TOKEN, { polling: true });

// Для хранения состояний пользователей
const userStates = {};
// Для хранения cookie сессии
const userCookies = {};

// Запуск диалога
```

Рисунок 12 – часть готового ответа нейросети

Сложности при работе с нейросетью:

1. Многие нейросетевые модели имеют ограничение по количеству запросов, прикреплению файлов и картинок в бесплатных версиях, что сильно усложняет поиск оптимальной модели
2. Большинство нейросетей не дописывают текст или код до конца из-за ограничения по символам, что затрудняет работу с большим кодом
3. Также существенно затрудняет работу ограничение по символам на промпт для нейросети, так как иногда приходится вставлять код в виде текста, если добавление файлов недоступно или ограничено
4. Многие нейросети выдают неправильные ответы, нерабочие коды и т.д., поэтому результаты требуется перепроверять

Пример ошибки нейросети:

```

body = await req.json();
});
req.on('end', async () => {
const { username, password } = JSON.parse(body);

if (await checkUser(username, password)) {
req.session.isAuthenticated = true;
res.writeHead(302, { Location: '/' });
res.end();
} else {
res.writeHead(302, { Location: '/login?error=1' });
res.end();
}
});
} else if (req.url === '/login' || req.url === '/login/') {
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 page');
}
} else if (req.url

```

Рисунок 13 – пример ошибки нейросети

В данном случае была попытка написания кода для авторизации через нейросеть, однако возникла следующая ошибка: нейросеть выдала код, но он был написан не полностью, так как получился объёмным и дописать его до конца не позволило ограничение по символам для ответа нейросети.

Вывод

В ходе выполнения лабораторной работы был реализован современный веб-сервис для управления списком дел с интеграцией Telegram-бота. В процессе разработки активно использовались нейросетевые инструменты для генерации и оптимизации программного кода.

Следует отметить, что несмотря на значительную помощь, которую оказывают нейросети при написании кода — ускоряя процесс разработки, предлагая рабочие решения и помогая автоматизировать рутинные задачи — на текущем этапе развития технологий искусственного интеллекта они не способны полностью заменить человека-программиста. Нейросети часто допускают синтаксические и логические ошибки, требуют проверки, доработки и адаптации с учётом специфики проекта. Поэтому участие разработчика остаётся необходимым на всех этапах создания программного продукта.

Тем не менее, опыт работы с нейросетевыми инструментами в рамках данной лабораторной работы показал, что их использование действительно упрощает и ускоряет процесс разработки, позволяет быстрее находить решения и повышает продуктивность. Такой подход способствует формированию современных навыков и расширяет возможности при создании новых IT-продуктов.