

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

должность, уч. степень, звание

подпись, дата

Жиданов К.А.

инициалы, фамилия

ОТЧЕТ
О ЛАБОРАТОРНОЙ РАБОТЕ №1

по дисциплине: Основы программирования

РАБОТУ ВЫПОЛНИЛ

Студент гр. № 3331

подпись, дата

Меркулова С.М.

инициалы, фамилия

Санкт-Петербург 2025

Цель лабораторной работы

Разработать веб-приложение To-Do List с авторизацией пользователей и интеграцией Telegram-бота.

Задачи:

- Реализовать регистрацию и вход для нескольких пользователей.
- Реализовать на сайте добавление, удаление и редактирование элементов.
- Обеспечить сохранение задач для каждого пользователя отдельно.
- Интегрировать Telegram-бота для просмотра задач через чат.

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 with Auth</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 20px;
    }
    .auth-container {
      max-width: 400px;
      margin: 0 auto;
      padding: 20px;
      border: 1px solid #ddd;
      border-radius: 5px;
      margin-top: 50px;
    }
    .todo-container {
      display: none;
      max-width: 800px;
      margin: 0 auto;
    }
    #todoList {
      border-collapse: collapse;
      width: 100%;
      margin: 20px 0;
    }
    #todoList th, #todoList td {
      border: 1px solid #ddd;
      padding: 8px;
      text-align: left;
```

```

    }
    #todoList th {
        background-color: #f0f0f0;
    }
    .action-buttons {
        display: flex;
        gap: 5px;
    }
    .add-form {
        margin: 20px 0;
    }
    .add-form input[type="text"] {
        padding: 8px;
        width: 70%;
    }
    .add-form button {
        padding: 8px 15px;
    }
    .edit-input {
        width: 100%;
        padding: 5px;
        box-sizing: border-box;
    }
    .error-message {
        color: red;
        margin-top: 10px;
    }
    .user-info {
        float: right;
    }
</style>
</head>
<body>

<div id="authContainer" class="auth-container">
    <h2>Login / Register</h2>
    <div>
        <input type="text" id="username" placeholder="Username" required><br><br>
        <input type="password" id="password" placeholder="Password"
required><br><br>
        <button onclick="login()">Login</button>
        <button onclick="register()">Register</button>
    </div>
    <div id="authError" class="error-message"></div>
</div>

<div id="todoContainer" class="todo-container">
    <div class="user-info">
        Welcome, <span id="currentUser"></span> |
        <button onclick="logout()">Logout</button>
    </div>

```

```

<h2>Your To-Do List</h2>

<table id="todoList">
  <thead>
    <tr>
      <th>#</th>
      <th>Task</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody id="listBody"></tbody>
</table>

<div class="add-form">
  <input type="text" id="newItem" placeholder="Enter new task">
  <button onclick="addItem()">Add Task</button>
</div>
</div>

<script>
  let currentUserId = null;

  // Check if user is already logged in
  checkSession();

  async function checkSession() {
    try {
      const response = await fetch('/check-auth');
      const data = await response.json();
      if (data.loggedIn) {
        currentUserId = data.userId;
        document.getElementById('currentUser').textContent =
data.username;
        document.getElementById('authContainer').style.display = 'none';
        document.getElementById('todoContainer').style.display = 'block';
        loadItems();
      }
    } catch (error) {
      console.error('Session check error:', error);
    }
  }

  async function login() {
    const username = document.getElementById('username').value.trim();
    const password = document.getElementById('password').value.trim();

    try {
      const response = await fetch('/login', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',

```

```

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

    const data = await response.json();

    if (data.success) {
        currentUserId = data.userId;
        document.getElementById('currentUser').textContent = username;
        document.getElementById('authContainer').style.display = 'none';
        document.getElementById('todoContainer').style.display = 'block';
        document.getElementById('authError').textContent = '';
        loadItems();
    } else {
        document.getElementById('authError').textContent = data.message;
    }
} catch (error) {
    console.error('Login error:', error);
    document.getElementById('authError').textContent = 'Login failed';
}

}

async function register() {
    const username = document.getElementById('username').value.trim();
    const password = document.getElementById('password').value.trim();

    try {
        const response = await fetch('/register', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({ username, password }),
        });

        const data = await response.json();

        if (data.success) {
            document.getElementById('authError').textContent = 'Registration
successful! Please login.';
            document.getElementById('authError').style.color = 'green';
        } else {
            document.getElementById('authError').textContent = data.message;
            document.getElementById('authError').style.color = 'red';
        }
    } catch (error) {
        console.error('Registration error:', error);
        document.getElementById('authError').textContent = 'Registration
failed';
    }
}

```

```

async function logout() {
  try {
    await fetch('/logout', { method: 'POST' });
    currentUserId = null;
    document.getElementById('authContainer').style.display = 'block';
    document.getElementById('todoContainer').style.display = 'none';
    document.getElementById('username').value = '';
    document.getElementById('password').value = '';
    document.getElementById('authError').textContent = '';
  } catch (error) {
    console.error('Logout error:', error);
  }
}

async function loadItems() {
  try {
    const response = await fetch('/items');
    const items = await response.json();
    renderList(items);
  } catch (error) {
    console.error('Error loading items:', error);
  }
}

function renderList(items) {
  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>
        <div class="action-buttons">
          <button class="edit-btn" data-
id="${item.id}">Edit</button>
          <button class="delete-btn" data-
id="${item.id}">x</button>
        </div>
      </td>
    `;
    listBody.appendChild(row);
  });

  // Add event listeners
  document.querySelectorAll('.delete-btn').forEach(btn => {
    btn.addEventListener('click', async function() {
      const id = this.getAttribute('data-id');
      await deleteItem(id);
    });
  });
}

```

```

    });
  });

  document.querySelectorAll('.edit-btn').forEach(btn => {
    btn.addEventListener('click', function() {
      const id = this.getAttribute('data-id');
      enableEdit(id);
    });
  });
}

async function addItem() {
  const text = document.getElementById('newItem').value.trim();
  if (!text) return;

  try {
    const response = await fetch('/items', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ text }),
    });

    if (response.ok) {
      document.getElementById('newItem').value = '';
      loadItems();
    }
  } catch (error) {
    console.error('Error adding item:', error);
  }
}

async function deleteItem(id) {
  try {
    const response = await fetch(`/items/${id}`, {
      method: 'DELETE',
    });

    if (response.ok) {
      loadItems();
    }
  } catch (error) {
    console.error('Error deleting item:', error);
  }
}

function enableEdit(id) {
  const row = document.querySelector(`.edit-btn[data-id="${id}"]`).closest('tr');
  const textCell = row.querySelector('.item-text');

```

```

const currentText = textCell.textContent;

textCell.innerHTML = `
  <input type="text" class="edit-input" value="${currentText}">
  <button class="save-btn" data-id="${id}">Save</button>
  <button class="cancel-btn">Cancel</button>
`;

const saveBtn = textCell.querySelector('.save-btn');
const cancelBtn = textCell.querySelector('.cancel-btn');

saveBtn.addEventListener('click', async function() {
  const newText = textCell.querySelector('.edit-input').value.trim();
  if (newText) {
    await updateItem(id, newText);
  }
});

cancelBtn.addEventListener('click', () => loadItems());
}

async function updateItem(id, newText) {
  try {
    const response = await fetch(`/items/${id}`, {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ text: newText }),
    });

    if (response.ok) {
      loadItems();
    }
  } catch (error) {
    console.error('Error updating item:', error);
  }
}
</script>
</body>
</html>

```

```

index.js
const http = require('http');
const fs = require('fs');
const path = require('path');
const url = require('url');
const sqlite3 = require('sqlite3').verbose();
const crypto = require('crypto');
const cookie = require('cookie');
const TelegramBot = require('node-telegram-bot-api');

```



```

const axios = require('axios');

// Telegram bot token
const TELEGRAM_TOKEN = '7730296042:AAFvOdbRI37_dz3UsRNeH3SvEDCyGrnRZOo';
const bot = new TelegramBot(TELEGRAM_TOKEN, {polling: true});

// Database setup
const db = new sqlite3.Database('./todo.db', (err) => {
  if (err) {
    console.error('Database error:', err);
  } else {
    console.log('Connected to SQLite database');
    initializeDatabase();
  }
});

function initializeDatabase() {
  db.serialize(() => {
    db.run(`CREATE TABLE IF NOT EXISTS users (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      username TEXT UNIQUE,
      password TEXT,
      salt TEXT,
      telegram_chat_id TEXT
    )`);

    db.run(`CREATE TABLE IF NOT EXISTS items (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      user_id INTEGER,
      text TEXT,
      FOREIGN KEY(user_id) REFERENCES users(id)
    )`);
  });
}

function dbQuery(query, params = []) {
  return new Promise((resolve, reject) => {
    db.all(query, params, (err, rows) => {
      if (err) reject(err);
      else resolve(rows);
    });
  });
}

function dbRun(query, params = []) {
  return new Promise((resolve, reject) => {
    db.run(query, params, function(err) {
      if (err) reject(err);
      else resolve(this);
    });
  });
}

```

```

}

function hashPassword(password, salt) {
  return crypto.pbkdf2Sync(password, salt, 1000, 64, 'sha512').toString('hex');
}

const sessions = {};

function createSession(userId, username) {
  const sessionId = crypto.randomBytes(16).toString('hex');
  sessions[sessionId] = { userId, username };
  return sessionId;
}

function getSession(sessionId) {
  return sessions[sessionId];
}

function deleteSession(sessionId) {
  delete sessions[sessionId];
}

// HTTP Server
const server = http.createServer(async (req, res) => {
  const parsedUrl = url.parse(req.url, true);
  const cookies = cookie.parse(req.headers.cookie || '');

  // Serve static files
  if (req.url === '/') {
    try {
      const html = await fs.promises.readFile(path.join(__dirname, 'index.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 index.html');
    }
    return;
  }

  // API endpoints
  if (req.method === 'POST' && parsedUrl.pathname === '/login') {
    let body = '';
    req.on('data', chunk => body += chunk.toString());
    req.on('end', async () => {
      try {
        const { username, password } = JSON.parse(body);
        const user = await dbQuery('SELECT * FROM users WHERE username = ?', [username]);

```

```

        if (user.length === 0) {
            res.writeHead(401, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify({ success: false, message: 'Invalid
username or password' }));
            return;
        }

        const hashedPassword = hashPassword(password, user[0].salt);
        if (hashedPassword !== user[0].password) {
            res.writeHead(401, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify({ success: false, message: 'Invalid
username or password' }));
            return;
        }

        const sessionId = createSession(user[0].id, user[0].username);
        res.writeHead(200, {
            'Content-Type': 'application/json',
            'Set-Cookie': cookie.serialize('sessionId', sessionId, {
                httpOnly: true,
                maxAge: 60 * 60 * 24 * 7
            })
        });
        res.end(JSON.stringify({
            success: true,
            userId: user[0].id,
            username: user[0].username
        }));
    } catch (error) {
        res.writeHead(500, { 'Content-Type': 'application/json' });
        res.end(JSON.stringify({ success: false, message: 'Login failed'
})));
    }
    });
    return;
}

if (req.method === 'POST' && parsedUrl.pathname === '/register') {
    let body = '';
    req.on('data', chunk => body += chunk.toString());
    req.on('end', async () => {
        try {
            const { username, password } = JSON.parse(body);

            if (!username || !password) {
                res.writeHead(400, { 'Content-Type': 'application/json' });
                res.end(JSON.stringify({ success: false, message: 'Username
and password are required' }));
                return;
            }

```

```

        const salt = crypto.randomBytes(16).toString('hex');
        const hashedPassword = hashPassword(password, salt);

        try {
            await dbRun(
                'INSERT INTO users (username, password, salt) VALUES (?, ?, ?)',
                [username, hashedPassword, salt]
            );

            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify({ success: true }));
        } catch (err) {
            if (err.message.includes('UNIQUE constraint failed')) {
                res.writeHead(400, { 'Content-Type': 'application/json' });
                res.end(JSON.stringify({ success: false, message: 'Username already exists' }));
            } else {
                throw err;
            }
        } catch (error) {
            res.writeHead(500, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify({ success: false, message: 'Registration failed' }));
        }
    }

    return;
}

if (req.method === 'POST' && parsedUrl.pathname === '/logout') {
    if (cookies.sessionId) {
        deleteSession(cookies.sessionId);
    }
    res.writeHead(200, {
        'Content-Type': 'application/json',
        'Set-Cookie': cookie.serialize('sessionId', '', {
            httpOnly: true,
            expires: new Date(0)
        })
    });
    res.end(JSON.stringify({ success: true }));
    return;
}

if (req.method === 'GET' && parsedUrl.pathname === '/check-auth') {
    const session = cookies.sessionId ? getSession(cookies.sessionId) : null;
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify({
        loggedIn: !!session,
    }));
}

```

```

        userId: session?.userId,
        username: session?.username
    }));
    return;
}

// Protected routes
const session = cookies.sessionId ? getSession(cookies.sessionId) : null;
if (!session) {
    res.writeHead(401, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify({ error: 'Unauthorized' }));
    return;
}

if (req.method === 'GET' && parsedUrl.pathname === '/items') {
    try {
        const items = await dbQuery('SELECT id, text FROM items WHERE user_id = ?', [session.userId]);
        res.writeHead(200, { 'Content-Type': 'application/json' });
        res.end(JSON.stringify(items));
    } catch (error) {
        res.writeHead(500, { 'Content-Type': 'application/json' });
        res.end(JSON.stringify({ error: 'Failed to fetch items' }));
    }
    return;
}

if (req.method === 'POST' && parsedUrl.pathname === '/items') {
    let body = '';
    req.on('data', chunk => body += chunk.toString());
    req.on('end', async () => {
        try {
            const { text } = JSON.parse(body);
            await dbRun(
                'INSERT INTO items (user_id, text) VALUES (?, ?)',
                [session.userId, text]
            );
            res.writeHead(201, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify({ success: true }));
        } catch (error) {
            res.writeHead(500, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify({ error: 'Failed to add item' }));
        }
    });
    return;
}

if (req.method === 'PUT' && parsedUrl.pathname.startsWith('/items/')) {
    const id = parsedUrl.pathname.split('/')[2];
    let body = '';
    req.on('data', chunk => body += chunk.toString());

```

```

    req.on('end', async () => {
      try {
        const { text } = JSON.parse(body);
        const result = await dbRun(
          'UPDATE items SET text = ? WHERE id = ? AND user_id = ?',
          [text, id, session.userId]
        );

        if (result.changes === 0) {
          res.writeHead(404, { 'Content-Type': 'application/json' });
          res.end(JSON.stringify({ error: 'Item not found or not owned
by user' }));
        } else {
          res.writeHead(200, { 'Content-Type': 'application/json' });
          res.end(JSON.stringify({ success: true }));
        }
      } catch (error) {
        res.writeHead(500, { 'Content-Type': 'application/json' });
        res.end(JSON.stringify({ error: 'Failed to update item' }));
      }
    });
    return;
  }

  if (req.method === 'DELETE' && parsedUrl.pathname.startsWith('/items/')) {
    const id = parsedUrl.pathname.split('/')[2];
    try {
      const result = await dbRun(
        'DELETE FROM items WHERE id = ? AND user_id = ?',
        [id, session.userId]
      );

      if (result.changes === 0) {
        res.writeHead(404, { 'Content-Type': 'application/json' });
        res.end(JSON.stringify({ error: 'Item not found or not owned by
user' }));
      } else {
        res.writeHead(200, { 'Content-Type': 'application/json' });
        res.end(JSON.stringify({ success: true }));
      }
    } catch (error) {
      res.writeHead(500, { 'Content-Type': 'application/json' });
      res.end(JSON.stringify({ error: 'Failed to delete item' }));
    }
    return;
  }

  res.writeHead(404, { 'Content-Type': 'text/plain' });
  res.end('Not found');
});

```

```
// Telegram Bot Logic
const userAuthStates = {}; // Stores temporary auth data for users

bot.onText(/\sstart/, (msg) => {
  const chatId = msg.chat.id;
  bot.sendMessage(chatId, 'Добро пожаловать в To-Do List бот! Для доступа к
вашим задачам сначала авторизуйтесь:\n\n1. Введите команду /login\n2. Затем
введите ваш логин и пароль в формате: логин:пароль');
});

// Хранилище для состояний пользователей
const userSessions = {};

bot.onText(/\slogin/, (msg) => {
  const chatId = msg.chat.id;
  userSessions[chatId] = { state: 'awaiting_credentials' };
  bot.sendMessage(chatId, 'Пожалуйста, введите ваш логин и пароль в
формате:\nлогин:пароль\n\nНапример: myusername:mypassword');
});

bot.on('message', async (msg) => {
  const chatId = msg.chat.id;
  const text = msg.text;

  if (text.startsWith('/')) return;

  if (userSessions[chatId] && userSessions[chatId].state ===
'awaiting_credentials') {
    try {
      const [username, password] = text.split(':');

      if (!username || !password) {
        bot.sendMessage(chatId, 'Неверный формат. Пожалуйста, введите в
формате: логин:пароль');
        return;
      }

      const user = await dbQuery('SELECT * FROM users WHERE username = ?',
[username.trim()]);

      if (user.length === 0) {
        bot.sendMessage(chatId, '✗ Пользователь не найден');
        return;
      }

      const hashedPassword = hashPassword(password.trim(), user[0].salt);
      if (hashedPassword !== user[0].password) {
        bot.sendMessage(chatId, '✗ Неверный пароль');
        return;
      }
    }
  }
});
```

```

        // Сохраняем ID пользователя в сессии
        userSessions[chatId] = {
            userId: user[0].id,
            username: user[0].username,
            state: 'authenticated'
        };

        bot.sendMessage(chatId, '✅ Авторизация успешна! Теперь вы можете
использовать команду /get_todos для получения вашего списка задач.');
```

```

    } catch (error) {
        console.error('Auth error:', error);
        bot.sendMessage(chatId, '⚠️ Произошла ошибка при авторизации');
    }
}
});

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

    // Проверяем, авторизован ли пользователь в этом чате
    if (!userSessions[chatId] || userSessions[chatId].state !== 'authenticated')
    {
        bot.sendMessage(chatId, '❌ Вы не авторизованы. Пожалуйста, используйте
/login для авторизации.');
```

```

        return;
    }

    try {
        // Получаем задачи только для текущего авторизованного пользователя
        const todos = await dbQuery('SELECT text FROM items WHERE user_id = ?',
[userSessions[chatId].userId]);

        if (todos.length === 0) {
            bot.sendMessage(chatId, '📋 Ваш список задач пуст. Добавьте задачи
на сайте!');
```

```

            return;
        }

        let message = `📋 Ваши задачи (пользователь:
${userSessions[chatId].username}): \n\n`;
        todos.forEach((todo, index) => {
            message += `${index + 1}. ${todo.text} \n`;
        });

        bot.sendMessage(chatId, message);
    } catch (error) {
        console.error('Error getting todos:', error);
        bot.sendMessage(chatId, '⚠️ Произошла ошибка при получении задач');
```

```

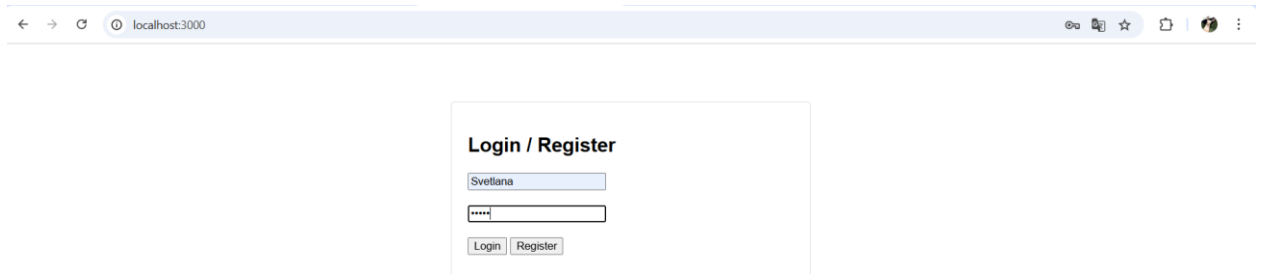
    }
});
});

```



```
// Start server
const PORT = 3000;
server.listen(PORT, () => console.log(`Server running on
http://localhost:${PORT}`));
console.log('Telegram bot запущен!');
```

Интерфейс программы:




Login / Register

Svetlana

Login Register

Рисунок 1 – Регистрация пользователя



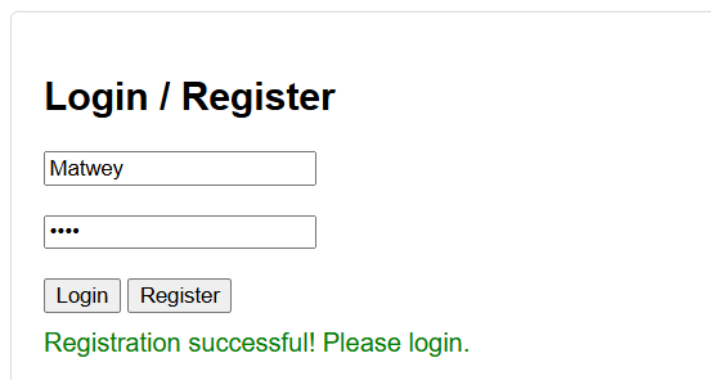
Your To-Do List Welcome, Svetlana | Logout

#	Task	Actions
1	Сделать лабораторную	Edit x
2	Постирать вещи	Edit x
3	Покупать	Edit x

Save Cancel

Enter new task Add Task

Рисунок 2 – Составление списка задач



Login / Register

Matwey

Login Register

Registration successful! Please login.

Рисунок 3 – Успешная регистрация

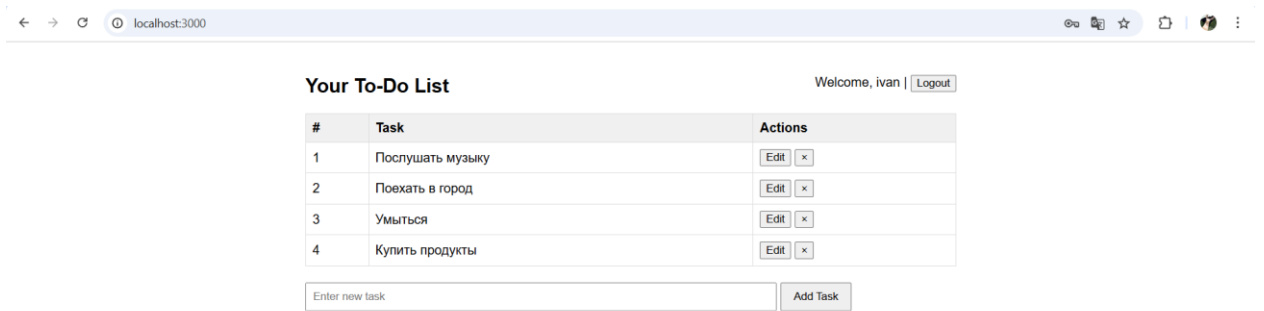


Рисунок 4 – Список задач второго зарегистрированного пользователя

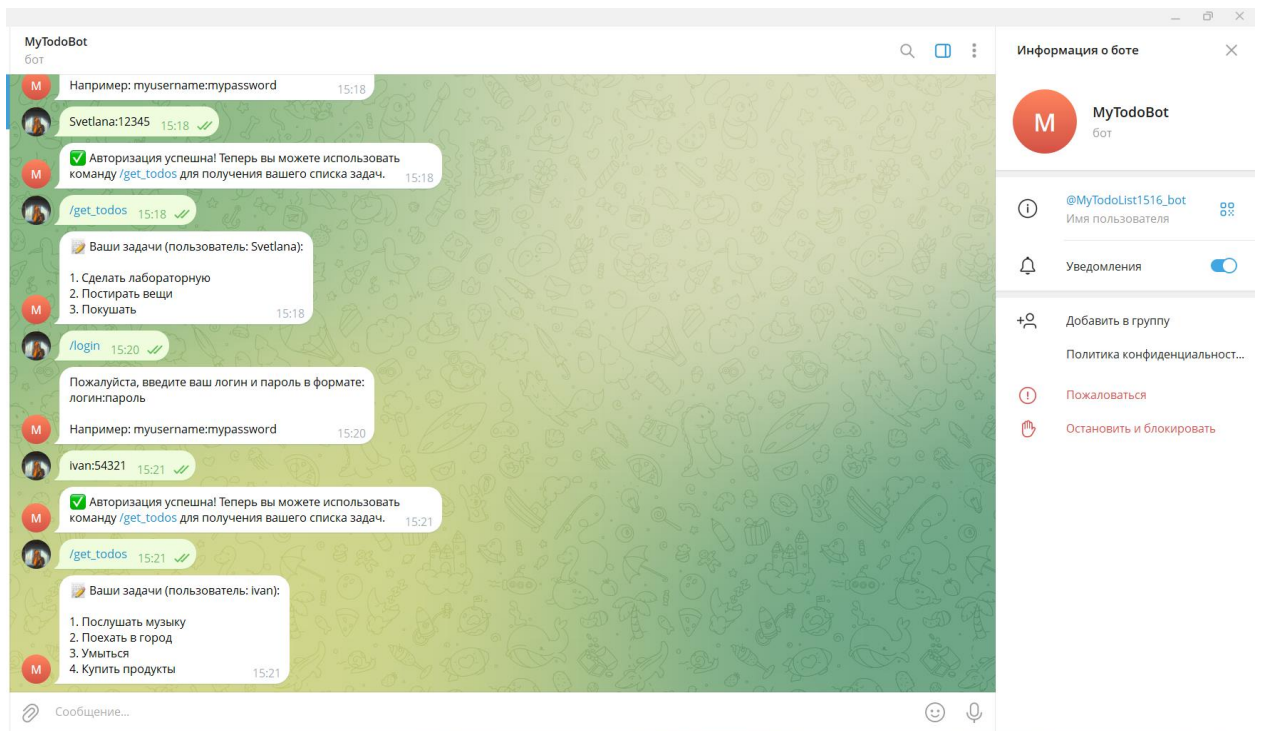


Рисунок 5 – работа бота

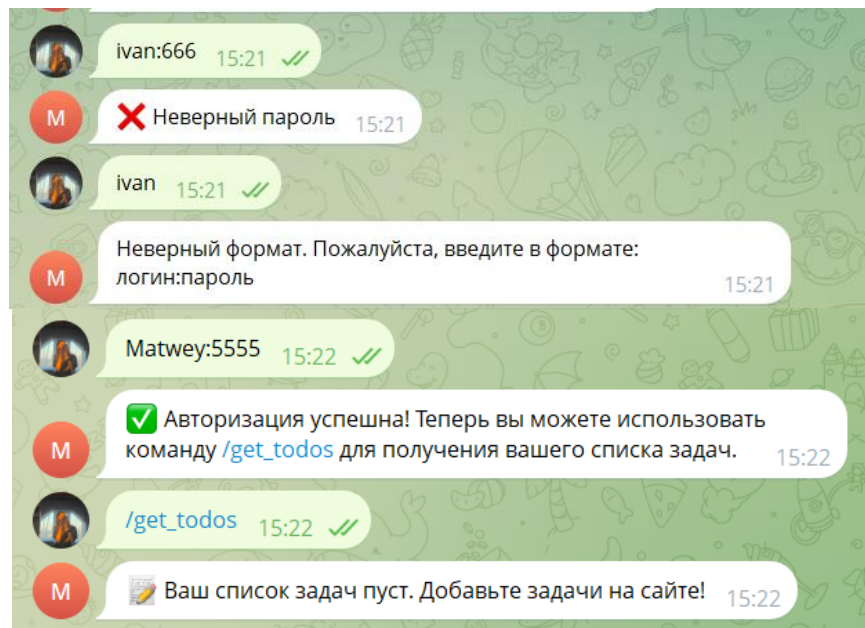


Рисунок 6 – ошибки, который обрабатывает бот

Описание работы:

Для начала вбиваем в браузер «<http://localhost:3000>», откроется сайт, на котором изначально необходимо зарегистрироваться. Все пароли и логины сохраняются и каждому отдельному пользователю соответствует отдельный список, который он создал на сайте. Так же добавлена интеграция с телеграммом. Существует бот, который по вашему логину и паролю выводит список задач, которые вы ввели на сайте. Сайт не допустит вас к списку задач, если введен неверный логин или пароль. Бот также недопускает к списку, если пароль и логин не совпадают. Если список пользователя пуст или логин и пароль введены не в том формате, бот также выдает сообщение об этом.

Для работы использовался чат GPT «DeepSeek».

Промты:

Сначала вводятся два файла в GPT «`index.html`» и «`index.js`».

«А ты можешь для этого кода добавить еще авторизацию пользователя, чтобы при этом сохранялись данные?»

Это задание он сделал с первой попытки, авторизация выполнялась так, как нужно, все данные сохранялись и проблем не возникало. Также бот объяснял

все пошагово, каждый пункт прописывал, что и как установить, как вводить данные.

А можешь для этого кода сделать интеграцию с телеграммом через IP адрес?

Нужно чтобы список дел приходил в сообщения в телеграмме

Здесь уже начались проблемы с установкой доступа к серверу. Каждую ошибку, которая возникала в процессе работы, я вводила в чат, чтобы он объяснил в чем проблема и как это исправить.

«Все сделала по инструкции, но бот мне не отвечает, и по ссылке

<http://95.123.456.78:3000> с моим IP адресом тоже не открывается, когда запускала сайт просто ввела node index.js в папке проекта без каких либо дополнительных команд. Как решить эту проблему?»

«Через телефон и этот компьютер я могу открыть сайт по ссылке с моим IP,

но бот по прежнему не отвечает, может после установки и внесения

изменений нужно было снова вводить какие то данные в командную строку?

Можешь еще раз пошагово описать то, что мне нужно вводить в командную строку уже после того как я настроила роутер»

«При регистрации выдает вот эту ошибку, что значит?»

«Я установила старую версию по твоей ссылке, но выдает данную ошибку»

«Так, я зарегистрировалась на сайте, получила токен, что сделать если не использовать старую версию? Можешь заново все расписать по пунктам максимально подробно»

Однако чем больше ошибок возникало и чем больше запросов было, я все сильнее путалась. Возникали проблемы с тем, что бот при любых моих запросах выводил сообщение: «Ошибка при получении задач».

«Бот отвечает, но выдает каждый раз "ошибка при получении задач", что с этим делать?»

Особого понимания в чем именно проблема не было, чат предположил, что дело в сервере, однако бот работал при запуске сервера, но просто не получал список. Все его инструкции не помогали. Поэтому я ввела еще один промт,

чтобы заново найти все проблемы, подытожить всю проделанную работу и уменьшить путаницу. Создала новый чат, заново ввела оба файла и написала промт:

«Давай начнем с начала и подытожим. У меня есть два файла, с кодом, который ты мне написал. Мне нужно сделать на этом сайте авторизацию, чтобы можно было регистрироваться несколькими пользователями, составлять списки и у каждого пользователя они сохранялись. Также мне нужна интеграция с телеграммом, чтобы был бот, который сначала запрашивает логин и пароль авторизованного пользователя, а затем высылает ему список тех задач, которые пользователь ввел на сайте. В целом бот работает тогда, когда запущен сервер, но список задач он не видит, каждый раз присылает сообщение "Ошибка при получении задач". Можешь исправить все ошибки, переделать код так, чтобы все работало и объяснить как что запускать, какие правила запуска, и по пункту сказать все шаги работы, объясняя до мелочей»

После этого чат выдал уже работающий код, однако возникла проблема с тем, что при введении разных пользователей в бот (т.е. при введении логина и пароля второго пользователя) чат выводил список дел пользователя, который первым пользовался ботом.

«Бот работает и отвечает, но список задач выводит только одного пользователя, если я захочу узнать список задач другого и зарегистрируюсь, то выведет первого»

После этого запроса бот прислал работающий без ошибок код, список задач высылался ровно в зависимости от того, какой пользователь хочет его узнать, если список обновлялся на сайте, он также обновлялся и в боте. Никаких проблем больше не возникало. Код прошел все проверки.

Вывод: программа работает и успешно справляется с заданием. Чат GPT полностью справился с задачей, хоть и не с первой попытки. На все мои вопросы отвечал лаконично и понятно, решал большинство проблем. Однако сам запрос нужно задавать максимально понятно и подробно, чтобы не возникало путаницы и недопониманий. Обращаться можно с любой

проблемой, так как большинство ошибок, которые возникали в процессе работы, чат решил. Финальная версия приложения представляет собой полноценную систему, где пользователи могут регистрироваться, создавать и управлять своими списками задач через веб-интерфейс, а также получать доступ к своим задачам через Telegram-бота. Для этого бот запрашивает учетные данные пользователя, после чего выводит актуальный список его задач.