МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования
«Санкт-Петербургский государственный университет аэрокосмического приборостроения»

КАФЕДРА №33

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

по дисциплине: Технологии и методы программирования

РАБОТУ ВЫПОЛНИЛ

Студент гр. № 3331		Е.А. Курочкина
	подпись, дата	инициалы, фамилия

Санкт-Петербург

2025

ЦЕЛЬ РАБОТЫ

Создание системы авторизации и реализация интеграции с Telegram. Проект представляет собой простое веб-приложение для управления задачами, интегрированное с Telegram.

ОСНОВНЫЕ ФУНКЦИИ

- CRUD (создание, чтение, обновление, удаление задач)
- Аутентификацией пользователей
- Интеграцией с Telegram

ХОД РАБОТЫ

Подготовка

1. Определение функциональных возможностей
2. **Настройка среды**:
    - Создание Firebase-проекта для аутентификации
    - Регистрация Telegram-бота
    - Настройка MySQL-базы данных
3. **Реализация клиентской части**

1. **Структура HTML/CSS**:
    - Адаптивный интерфейс с таблицей задач
    - Форма добавления/редактирования задач
    - Панель аутентификации

КОД ПРОЕКТА

```
<!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;
        max-width: 800px;
        margin: 0 auto;
        padding: 20px;
        background-color: #f5f5f5;
    }
    .auth-panel {
        display: flex;
        justify-content: flex-end;
        margin-bottom: 20px;
        gap: 10px;
    }
    .auth-btn {
        padding: 8px 16px;
        border: none;
        border-radius: 4px;
        cursor: pointer;
        color: white;
    }
    .google-auth {
        background-color: #4285F4;
    }
    .telegram-auth {
        background-color: #0088cc;
    }
```

```
.logout-btn {
    background-color: #e74c3c;
}
.hidden {
    display: none;
}
#todoList {
    border-collapse: collapse;
    width: 100%;
    margin: 20px 0;
    box-shadow: 0 2px 8px rgba(0,0,0,0.1);
    background-color: white;
}
#todoList th, #todoList td {
    border: 1px solid #ddd;
    padding: 12px;
    text-align: left;
}
#todoList th {
    background-color: #3498db;
    color: white;
}
.action-buttons button {
    padding: 6px 10px;
    margin: 0 3px;
    cursor: pointer;
    border: none;
```

```
    border-radius: 3px;
    color: white;
}
.edit-btn {
    background-color: #f39c12;
}
.remove-btn {
    background-color: #e74c3c;
}
.add-form {
    margin-top: 20px;
    display: flex;
    gap: 10px;
}
.add-form input {
    flex: 1;
    padding: 10px;
    border: 1px solid #ddd;
    border-radius: 4px;
}
.add-form button {
    padding: 10px 20px;
    background-color: #2ecc71;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
```

```
        }
        .user-info {
            display: flex;
            align-items: center;
            gap: 10px;
            margin-bottom: 20px;
        }
        .user-avatar {
            width: 40px;
            height: 40px;
            border-radius: 50%;
        }
    </style>
</head>
<body>
    <div class="auth-panel">
        <button id="googleAuth" class="auth-btn google-auth">Login with Google</button>
        <button id="telegramAuth" class="auth-btn telegram-auth">Login with Telegram</button>
        <button id="logoutBtn" class="auth-btn logout-btn hidden">Logout</button>
    </div>

    <div id="userInfo" class="user-info hidden">
        <img id="userAvatar" class="user-avatar" src="" alt="User Avatar">
        <span id="userName"></span>
    </div>
```

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

  <div id="appContent" class="hidden">
    <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</button>
    </div>
  </div>

  <!-- Firebase SDK -->
  <script src="https://www.gstatic.com/firebasejs/8.10.0/firebase-app.js"></script>
  <script src="https://www.gstatic.com/firebasejs/8.10.0/firebase-auth.js"></script>

  <!-- Telegram WebApp SDK -->
```

```
<script src="https://telegram.org/js/telegram-web-app.js"></script>

<script>
    // Firebase configuration
    const firebaseConfig = {
        apiKey: "YOUR_FIREBASE_API_KEY",
        authDomain: "YOUR_FIREBASE_AUTH_DOMAIN",
        projectId: "YOUR_FIREBASE_PROJECT_ID",
        storageBucket: "YOUR_FIREBASE_STORAGE_BUCKET",
        messagingSenderId: "YOUR_FIREBASE_SENDER_ID",
        appId: "YOUR_FIREBASE_APP_ID"
    };

    // Initialize Firebase
    firebase.initializeApp(firebaseConfig);
    const auth = firebase.auth();

    // Telegram WebApp initialization
    let tg = null;
    if (window.Telegram && window.Telegram.WebApp) {
        tg = window.Telegram.WebApp;
        tg.expand();
    }

    // App state
    let currentUser = null;
    let items = JSON.parse(localStorage.getItem('todoItems')) || [];
```

```
    let nextId = items.length > 0 ? Math.max(...items.map(item => item.id)) + 1 :
1;

    // DOM elements
    const googleAuthBtn = document.getElementById('googleAuth');
    const telegramAuthBtn = document.getElementById('telegramAuth');
    const logoutBtn = document.getElementById('logoutBtn');
    const appContent = document.getElementById('appContent');
    const userInfo = document.getElementById('userInfo');
    const userAvatar = document.getElementById('userAvatar');
    const userName = document.getElementById('userName');

    // Initialize the app
    document.addEventListener('DOMContentLoaded', () => {
      checkAuthState();
      renderList();
    });

    // Firebase auth state listener
    function checkAuthState() {
      auth.onAuthStateChanged(user => {
        if (user) {
          currentUser = {
            uid: user.uid,
            displayName: user.displayName,
            photoURL: user.photoURL,
            provider: 'google'
          };
```

```
            showAppContent();
        } else if (tg && tg.initDataUnsafe && tg.initDataUnsafe.user) {
            // Telegram auth
            const tgUser = tg.initDataUnsafe.user;
            currentUser = {
                uid: `tg_${tgUser.id}`,
                displayName: tgUser.first_name || 'Telegram User',
                photoURL: tgUser.photo_url || '',
                provider: 'telegram'
            };
            showAppContent();
        } else {
            hideAppContent();
        }
    });
}

function showAppContent() {
    document.querySelectorAll('.auth-btn').forEach(btn =>
btn.classList.add('hidden'));
    logoutBtn.classList.remove('hidden');
    appContent.classList.remove('hidden');
    userInfo.classList.remove('hidden');

    // Update user info
    userName.textContent = currentUser.displayName;
    if (currentUser.photoURL) {
        userAvatar.src = currentUser.photoURL;
```

```
        } else {
            userAvatar.src = 'https://via.placeholder.com/40';
        }
    }

    function hideAppContent() {
        document.querySelectorAll('.auth-btn').forEach(btn =>
btn.classList.remove('hidden'));
        logoutBtn.classList.add('hidden');
        appContent.classList.add('hidden');
        userInfo.classList.add('hidden');
    }

    // Auth handlers
    googleAuthBtn.addEventListener('click', () => {
        const provider = new firebase.auth.GoogleAuthProvider();
        auth.signInWithPopup(provider);
    });

    telegramAuthBtn.addEventListener('click', () => {
        if (tg) {
tg.openTelegramLink(`https://t.me/YOUR_BOT_USERNAME?start=webapp_${win
dow.location.hostname}`);
        } else {
            alert('Please open this page in Telegram');
        }
    });
```

```
    logoutBtn.addEventListener('click', () => {
        if (currentUser.provider === 'google') {
            auth.signOut();
        } else if (tg) {
            tg.close();
        }
        currentUser = null;
        hideAppContent();
    });

    // To-Do List functions
    function renderList() {
        const listBody = document.getElementById('listBody');
        listBody.innerHTML = '';

        if (items.length === 0) {
            const row = document.createElement('tr');
            row.innerHTML = `<td colspan="3" style="text-align: center;">No tasks yet. Add one above!</td>`;
            listBody.appendChild(row);
            return;
        }

        items.forEach((item, index) => {
            const row = document.createElement('tr');
            row.innerHTML = `
                <td>${index + 1}</td>
```

```
            <td>${item.text}</td>
            <td>
                <button class="action-btn edit-btn" onclick="editItem(${index})">Edit</button>
                <button class="action-btn remove-btn" onclick="removeItem(${index})">Remove</button>
            </td>
        `;
        listBody.appendChild(row);
    });

    localStorage.setItem('todoItems', JSON.stringify(items));
}

function addItem() {
    const newItemInput = document.getElementById('newItem');
    const newItemText = newItemInput.value.trim();

    if (newItemText) {
        items.push({
            id: nextId++,
            text: newItemText,
            userId: currentUser.uid,
            createdAt: new Date().toISOString()
        });
        newItemInput.value = '';
        renderList();
    }
```

```
        }

        function removeItem(index) {
            if (confirm('Are you sure you want to delete this task?')) {
                items.splice(index, 1);
                renderList();
            }
        }

        function editItem(index) {
            const newText = prompt('Edit task:', items[index].text);
            if (newText !== null && newText.trim() !== '') {
                items[index].text = newText.trim();
                items[index].updatedAt = new Date().toISOString();
                renderList();
            }
        }

        // Handle Enter key press
        document.getElementById('newItem').addEventListener('keypress',
function(e) {
            if (e.key === 'Enter') {
                addItem();
            }
        });
    </script>
</body>
</html>
```

```
const http = require('http');
const fs = require('fs');
const path = require('path');
const mysql = require('mysql2/promise');
const PORT = 3000;

// Database connection settings
const dbConfig = {
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'todolist',
};

// Authentication middleware
async function authenticate(req, res, next) {
    // Here you would implement your authentication logic
    // For example, check session cookies or JWT tokens
    next();
}

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.end();
        return rows;
    } catch (error) {
        console.error('Error retrieving list items:', error);
        throw error;
    }
}

// Modified request handler with authentication
async function handleRequest(req, res) {
    if (req.url === '/') {
        try {
            // This would be replaced with actual user ID from session
            const userId = 'current_user_id';
            const todoItems = await retrieveListItems(userId);

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

            // Generate HTML rows
            const rowsHtml = todoItems.map(item => `
                <tr>
                    <td>${item.id}</td>
                    <td>${item.text}</td>
                    <td>
```

```
                <button class="action-btn edit-btn"
onclick="editItem(${item.id})">Edit</button>
                <button class="action-btn remove-btn"
onclick="removeItem(${item.id})">Remove</button>
            </td>
          </tr>
        `).join('');

        const processedHtml = html.replace('<!-- Tasks will be added here
dynamically -->', rowsHtml);

        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.end(processedHtml);
      } catch (err) {
        console.error(err);
        res.writeHead(500, { 'Content-Type': 'text/plain' });
        res.end('Error loading page');
      }
  } else {
    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}`));
```

РАБОТА С НЕЙРОСЕТЬЮ

В процессе работы нейросеть отвечала на вопросы и решала поставленные задачи.

Проблемы, возникающие при работе с ботом

Из-за нечетких или неточных запросов бот может предлагать неоптимальные решения. Бот при таких запросах мог предлагать несовместимые с уже существующей программой решения.

Пример проблемы при неточном формулирование запроса бот мог предложить одновременное отображение формы авторизации и списка задач.

Решение проблемы: четкое описание того, что должен делать бот.

ИТОГИ РАБОТЫ

- Реализовано полнофункциональное приложение с:
    - Аутентификацией пользователя
    - CRUD (создание, чтение, обновление, удаление задач)
    - Интеграцией с Telegram

Направления развития

1. Реализация PWA-версии
2. Добавление совместного редактирования
3. Интеграция с календарями
4. Машинное обучение для сортировки задач

ВЫВОД

Разработанная и реализованная программа выполняет все требования и поставленные задачи. Обеспечивает интеграцию с телеграммом и авторизацию с возможностью последующего усовершенствования в том числе путем добавления новых функций. В работе возникли проблемы связанные с недопониманием ИИ запроса, отправленного человеком, которые было необходимо устранить (путем уточняющих запросов).