УДК 004.896

Разработка приложения для интеграции учетных записей сети интернет

Акентьев Данила Денисович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В современном информационном обществе интеграция учетных записей в сети Интернет является важным аспектом повседневной жизни пользователей и бизнес-организаций. В данной статье рассматривается разработка приложения для интеграции учетных записей различных онлайн-платформ с целью оптимизации процесса управления данными пользователей, повышения безопасности хранения личной информации и предоставления удобного интерфейса. Применяются методы веб-разработки с использованием языка программирования Python и фреймворка Flask, а также функциональное и безопасное тестирование для проверки работоспособности и надежности приложения.
Ключевые слова: Интеграция учетных записей, веб-разработка, безопасность данных, Python, Flask, API социальных сетей, функциональное тестирование, OAuth.

Development of an application for the integration of Internet accounts

Akentev Danila Denisovich
Sholom-Aleichem Priamursky State University
Student

Abstract
In today's information society, the integration of Internet accounts is an important aspect of the daily lives of users and business organizations. This article discusses the development of an application for integrating accounts of various online platforms in order to optimize the process of managing user data, increase the security of storing personal information and provide a user-friendly interface. Web development methods using the Python programming language and the Flask framework are used, as well as functional and secure testing to verify the operability and reliability of the application.
Keywords: Account integration, Web development, Data security, Python, Flask, Social Media API, Functional testing, OAuth.

1 Введение

1.1 Актуальность

В современном информационном обществе интеграция учетных записей в сети Интернет является важным аспектом повседневной жизни

пользователей и бизнес-организаций. С ростом популярности цифровых технологий и широкого использования онлайн-сервисов существует потребность в создании приложений, обеспечивающих удобство и безопасность управления учетными записями в различных онлайн-платформах.

1.2 Обзор исследований

А.Н. Юров помогает освоить основы разработки приложений на языке python [1]. Опытом использования Flask в написание веб-страниц делится Т.А. Халевин [2]. Р.И. Круглик привел пример интеграции авторизации на сайте через Google OAuth API [3]. Как FastAPI позволяет достичь большего за меньшее время и с меньшим количеством кода продемонстрировал в своей публикации Б. Любанович [4]. А.В. Неустроев и Р.В. Наумов в своей статье показал и рассказал, как использовать библиотеку OAuth в Python [5].

1.3 Цель исследования

Цель данной статьи заключается в разработке приложения для интеграции учетных записей в сети Интернет. Проект ориентирован на оптимизацию процесса управления данными пользователей, повышение безопасности хранения личной информации и обеспечение пользователей удобным интерфейсом для управления своими учетными записями.

2 Материалы и методы

Для проведения исследования по разработке приложения для интеграции учетных записей в сети Интернет будут использованы следующие методы: Разработка приложения: Использование метода прототипирования для создания начальной версии приложения с учетом выявленных требований. Применение методов веб-разработки с использованием языка программирования Python и фреймворка Flask. Тестирование: Проведение функционального тестирования приложения для проверки его работоспособности и соответствия требованиям. Оценка безопасности приложения с использованием методов тестирования на проникновение и анализа уязвимостей. Анализ результатов: Обработка результатов тестирования и сравнение их с изначальными требованиями и ожиданиями пользователей. Идентификация возможных улучшений и дополнительного функционала.

Использование комбинации данных методов позволит создать комплексный исследовательский подход, учитывая как теоретические аспекты, так и практические результаты разработки приложения.

3 Результаты и обсуждения

Для начала разработки приложения необходимо установить Python (рис.1).

Рисунок 1– Скачивание и установки Python

После установки Python, рекомендуется установить Visual Studio Code для редактирования кода (рис.2).

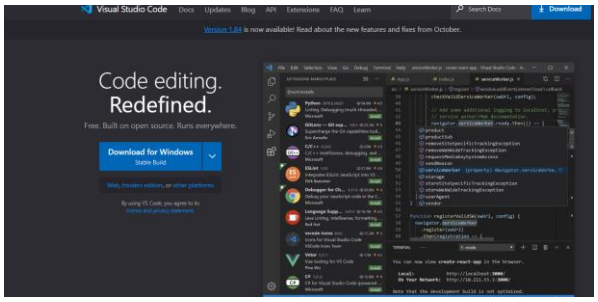

Рисунок 2 – Установка VS Code

Затем создаем рабочую среду в формате, представленном ниже, и открываем ее (рис.3). В терминале выполняем следующие команды (рис.4).

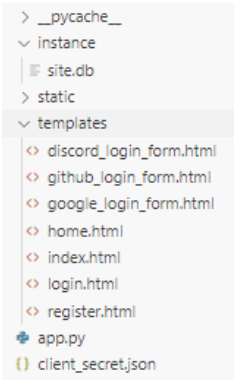

Рисунок 3 – Формат рабочей среды

```
>> pip install Flask Flask-Dance Flask-Login
>> pip install Flask-SQLAlchemy Flask-WTF google-auth
>> pip install google-auth-oauthlib oauthlib requests
>> pip install requests-oauthlib SQLAlchemy urllib3
>> pip install URLObject Werkzeug WTForms Zenora
```

Рисунок 4 – Установка необходимых библиотек

Интеграция Google

Для интеграции учетной записи через Google, переходим на веб-сайт Google Console [6].

Выбираем "Select a project", затем "New Project". Задаем имя проекта и нажимаем "Create". В левой колонке на веб-сайте переходим в раздел "APIs & Services" и выбираем "Library".

В разделе поиска находим Google+ API и Google People API, после чего активируем их, нажимая на опцию "Enable".

Затем снова переходим в левое меню, открываем раздел "APIs & Services" и переходим в "OAuth consent screen". Выбираем "External" и создаем новый экран согласия, нажимая "Create".

Указываем имя и выбираем электронную почту, затем прокручиваем вниз, указываем адрес электронной почты для контактной информации разработчика и нажимаем "Save and Continue". Дополнительные данные вводить не требуется.

Затем в левом меню переходим в раздел "Credentials". Затем в верхней части экрана выбираем "Create Credentials" и далее "OAuth client ID".

Выбираем тип приложения "web application". Добавляем Redirect URIs, в нашем случае это. После этого создаем клиента.

Копируем Client ID и загружаем JSON-файл. Переносим данные из файла в client_secret.json. Внесем эти данные в наш код и создадим объект flow для аунтефикации (рис.5).

```
GOOGLE_CLIENT_ID = "…"
client_secrets_file = os.path.join(pathlib.Path(__file__).parent, "client_secret.json")
flow = Flow.from_client_secrets_file(
    client_secrets_file=client_secrets_file,
    scopes=["https://www.googleapis.com/auth/userinfo.profile", "https://www.googleapis.com/auth/userinfo.email", "openid"],
    redirect_uri="http://localhost:5000/callback"
)
```

Рисунок 5 – Конфигурация Google

Далее создадим маршрут для инициации процесса аутентификации Google (рис.6).

```
@app.route("/login_google")
def login_google():
    authorization_url, state = flow.authorization_url()
    session["state"] = state
    return redirect(authorization_url)
```

Рисунок 6 – Обработчик запуска процесса аутентификации Google

Теперь создадим маршрут обратного вызова после успешной аутентификации для получения информации о пользователе с помощью токена, а также перенаправление пользователя на страницу создания пользователя в случае отсутствия информации в базе данных или авторизация пользователя под своей учетной записью в случае успешного поиска по google_id аккаунта (рис.7).

```python
@app.route("/callback")
def callback():
    flow.fetch_token(authorization_response=request.url)
    if not session["state"] == request.args["state"]:
        abort(500)
    credentials = flow.credentials
    request_session = requests.session()
    cached_session = cachecontrol.CacheControl(request_session)
    token_request = google.auth.transport.requests.Request(session=cached_session)
    id_info = id_token.verify_oauth2_token(
        id_token=credentials._id_token,
        request=token_request,
        audience=GOOGLE_CLIENT_ID
    )
    session["google_id"] = id_info.get("sub")
    session["email"] = id_info.get("email")
    # Проверяем, существует ли пользователь с таким google_id в базе данных
    user = User.query.filter_by(google_id=session["google_id"]).first()
    if user:
        login_user(user)
        return redirect(url_for('home'))
    else:
        if current_user.is_authenticated:
            # Обновляем данные пользователя
            current_user.email = session["email"]
            current_user.google_id = session["google_id"]
            current_user.email_confirmed = 1
            # Добавляем изменения в базу данных
            db.session.commit()
            login_user(current_user)
            return redirect(url_for('home'))
        else:
            return redirect(url_for("google_login_form"))
```

Рисунок 7 – Маршрут обратного вызова Google

В случае отсутствия учетной записи со схожим google_id пользователя перенаправит на страницу с формой входа (рис.8). Внешний вид страницы (рис.9).

```html
<h2 class="text-center mb-4"> Вход с помощью Google </h2> <!-- Форма входа -->
    <form method="POST" action="{{ url_for('google_login_form') }}" class="needs-validation" novalidate>
        {{ form.hidden_tag() }} <!-- Поле для ввода имени пользователя -->
        <div class="form-group">
            {{ form.username.label(class="font-weight-bold") }}
            {{ form.username(class="form-control", placeholder="Введите ваше имя пользователя", required=True) }}
            <div class="invalid-feedback"> Пожалуйста, введите ваше имя пользователя. </div>
        </div><!-- Поле для ввода пароля -->
        <div class="form-group">
            {{ form.password.label(class="font-weight-bold") }}
            {{ form.password(class="form-control", placeholder="Введите ваш пароль", required=True) }}
            <div class="invalid-feedback"> Пожалуйста, введите ваш пароль. </div>
        </div> <!-- Кнопка отправки формы -->
        <div class="form-group">{{ form.submit(class="btn btn-outline-dark") }}</div>
    </form>
```

Рисунок 8 – Форма входа с помощью Google

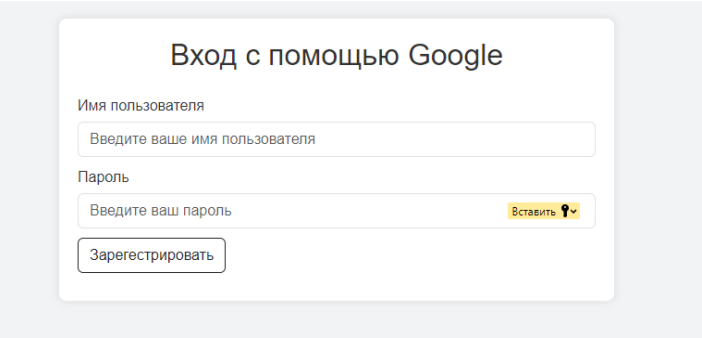

Рисунок 9 – Отображение формы входа с помощью Google

Далее создадим маршрут регистрации и входа пользователя с введёнными данными в форму (рис.10).

```python
@app.route("/google_login_form", methods=['GET', 'POST'])
def google_login_form():
    form = GoogleLoginForm()
    if form.validate_on_submit():
        # Получить данные из формы
        username = form.username.data
        password = form.password.data
        new_user = User(username=username,
                        password=password,
                        email=session["email"],
                        google_id=session["google_id"],
                        email_confirmed=True)
        db.session.add(new_user)
        db.session.commit()
        # Выполнить вход нового пользователя
        login_user(new_user)
        return redirect(url_for('home'))

    return render_template('google_login_form.html', form=form)
```

Рисунок 10 – Маршрут обработки данных из формы, регистрации и входа Google

После этого пользователь попадет на домашнюю страницу с полной информацией о своей учетной записи (с подтвержденной интеграцией google аккаунта) (рис.11).

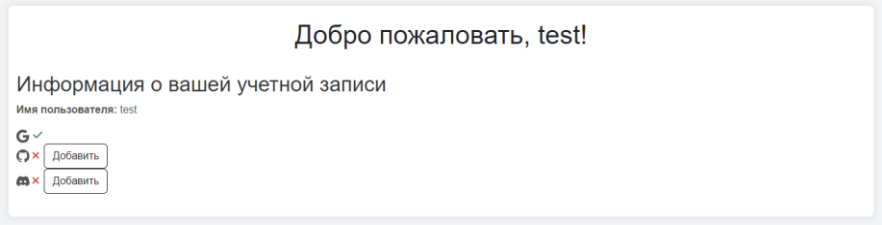

Рисунок 11 – Отображение домашней страницы

Интеграция GitHub

Теперь перейдем к интеграции учетной записи GitHub. Переходим в настройки на официальном сайте [7] и выбираем "Developer settings".

Затем переходим в раздел "OAuth Apps" и создаем новое приложение OAuth, выбрав опцию "New OAuth App".

Вводим название приложения, указываем URL домашней страницы и URL обратного вызова для авторизации. После этого регистрируем созданное приложение.

Сохраняем Client ID и создаем новый клиентский секрет. Теперь установим конфигурацию для этих данных (рис.12).

```python
app.config['GITHUB_CLIENT_ID'] = '…'
app.config['GITHUB_CLIENT_SECRET'] = '…'
```

Рисунок 12 – Конфигурация GitHub

Затем настроим обработчик для запуска процесса аутентификации на GitHub (рис.13).

```python
@app.route('/git_login')
def git_login():
    return redirect(f'https://github.com/login/oauth/authorize?client_id=
                    {app.config["GITHUB_CLIENT_ID"]}&scope=user:email')
```

Рисунок 13 – Обработчик запуска процесса аутентификации GitHub

На этом этапе создадим обратный вызов для получения информации о пользователе с использованием токена после успешной аутентификации. В случае отсутствия информации в базе данных, пользователь будет перенаправлен на страницу создания аккаунта. В случае нахождения соответствия по github_id, пользователь автоматически войдет под своей учетной записью (рис.14).

```python
@app.route('/git_login/authorized')
def authorized():
    code = request.args.get('code')
    if not code:
        return 'Access denied: missing authorization code'
    response = requests.post('https://github.com/login/oauth/access_token', params={
        'client_id': app.config['GITHUB_CLIENT_ID'],
        'client_secret': app.config['GITHUB_CLIENT_SECRET'],
        'code': code,
    }, headers={'Accept': 'application/json'})
    if response.status_code != 200:
        return f'Error exchanging code for access token: {response.text}'
    data = response.json()
    if 'access_token' not in data:
        return f'Access token not found in response: {data}'
    session['github_token'] = data['access_token']
    user_data = requests.get('https://api.github.com/user', headers={'Authorization': f'Bearer {data["access_token"]}'})
    user_data.raise_for_status()
    session['github_id'] = user_data.json().get('id')
    if not session['github_id']:
        return 'Error fetching GitHub user ID'
    # Проверяем, существует ли пользователь с таким github_id в базе данных
    user = User.query.filter_by(github_id=session['github_id']).first()
    if user:
        login_user(user)
        return redirect(url_for('home'))
    else:
        if current_user.is_authenticated:
            # Обновляем данные пользователя
            current_user.github_id = session['github_id']
            # Добавляем изменения в базу данных
            db.session.commit()
            login_user(current_user)
            return redirect(url_for('home'))
        else:
            return redirect(url_for("github_login_form"))
```

Рисунок 14 – Маршрут обратного вызова Github

Если учетной записи с аналогичным github_id не существует, пользователь будет перенаправлен на страницу с формой входа (рис.15). Внешний вид формы входа через GitHub (рис.16).

```html
<h2 class="text-center mb-4"> Вход с помощью GitHub </h2>
    <form method="POST" action="{{ url_for('github_login_form') }}" class="needs-validation" novalidate>
        {{ form.hidden_tag() }} <!-- Поле для ввода имени пользователя -->
        <div class="form-group">
            {{ form.username.label(class="font-weight-bold") }}
            {{ form.username(class="form-control", placeholder="Введите ваше имя пользователя", required=True) }}
            <div class="invalid-feedback"> Пожалуйста, введите ваше имя пользователя. </div>
        </div> <!-- Поле для ввода пароля -->
        <div class="form-group">
            {{ form.password.label(class="font-weight-bold") }}
            {{ form.password(class="form-control", placeholder="Введите ваш пароль", required=True) }}
            <div class="invalid-feedback"> Пожалуйста, введите ваш пароль. </div>
        </div> <!-- Кнопка отправки формы -->
        <div class="form-group">{{ form.submit(class="btn btn-outline-dark") }}</div>
    </form>
```

Рисунок 15 – Форма входа с помощью GitHub

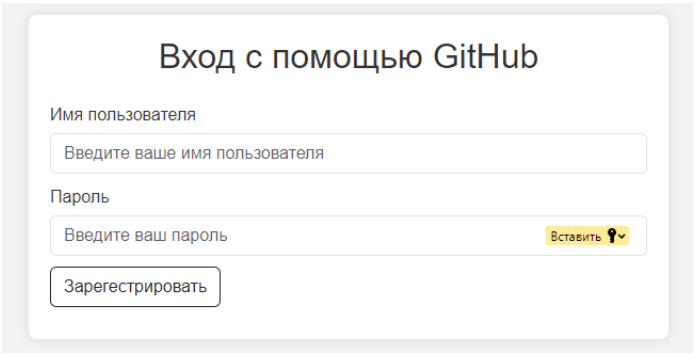

Рисунок 16 – Отображение формы входа с помощью GitHub

Далее настроим маршрут для регистрации и входа, обрабатывая введенные данные из формы (рис.17).

```
@app.route("/github_login_form", methods=['GET', 'POST'])
def github_login_form():
    form = GitHubLoginForm()
    if form.validate_on_submit():
        # Получить данные из формы
        username = form.username.data
        password = form.password.data
        # Если пользователь не существует, создать нового пользователя
        new_user = User(username=username,
                        password=password,
                        github_id=session['github_id'] )
        db.session.add(new_user)
        db.session.commit()
        # Выполнить вход нового пользователя
        login_user(new_user)
        return redirect(url_for('home'))
    return render_template('github_login_form.html', form=form)
```

Рисунок 17 – Маршрут обработки данных из формы, регистрации и входа GitHub

После успешного входа пользователь будет перенаправлен на домашнюю страницу с полной информацией о своей учетной записи, подтверждая успешную интеграцию с аккаунтом GitHub (рис.18).

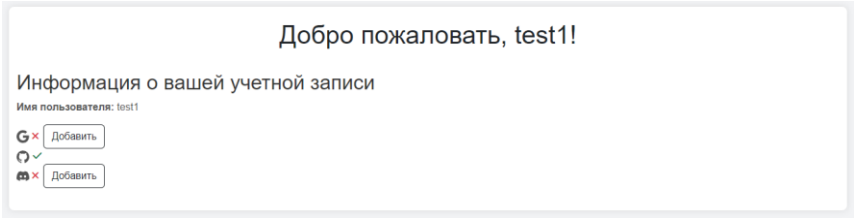

Рисунок 18 – Отображение домашней страницы

Интеграция Discord

Теперь осталось интегрировать учетную запись Discord. Для этого переходим на веб-сайт Discord Developer [8] и выбираем "New Application". Вводим имя приложения и создаем его.

В разделе "OAuth2 (General)" левого меню выбираем опцию "Reset Secret", сохраняем полученный секрет. После этого добавляем ссылку для перенаправления. Переходим к следующему шагу "URL Generator",

выбираем область "identify" и генерируем URL. Сохраните сгенерированный URL. Затем переходим к настройке бота, выбираем "Bot" и выполним сброс токена, после чего сохраним его.

Добавим данные полученные при создании приложения в код (рис.19).

```
TOKEN = "…"
DIS_CLIENT_ID = "…"
CLIENT_SECRET = "…"
REDIRECT_URI = "http://localhost:5000/oauth/callback"
client = APIClient(TOKEN,client_secret=CLIENT_SECRET)
```

Рисунок 19 – Конфигурация Discord

Продолжим, создав маршрут для начала процесса аутентификации в Discord (рис.20).

```
@app.route('/discord_login')
def discord_login():
    return redirect(f"https://discord.com/api/oauth2/authorize?client_id=
                    {parse.quote(DIS_CLIENT_ID)}&response_type=code&redirect_uri=
                    {parse.quote(REDIRECT_URI)}&scope=identify")
```

Рисунок 20 – Обработчик запуска процесса аутентификации Discord

На следующем этапе настроим обратный вызов для получения информации о пользователе с использованием токена после успешной аутентификации в Discord. Если информации о пользователе нет в базе данных, пользователь будет перенаправлен на страницу создания аккаунта. В случае успешного поиска по discord_id, пользователь автоматически войдет под своей учетной записью (рис.21).

```
@app.route("/oauth/callback")
def callback_dis():
    code = request.args['code']
    acces_token = client.oauth.get_access_token(code,REDIRECT_URI).access_token
    bearer_client= APIClient(acces_token, bearer=True)
    user_info = bearer_client.users.get_current_user()
    session['discord_id'] = user_info.id
    # Проверяем, существует ли пользователь с таким github_id в базе данных
    user = User.query.filter_by(discord_id=session['discord_id']).first()
    if user:
        login_user(user)
        return redirect(url_for('home'))
    else:
        if current_user.is_authenticated:
            # Обновляем данные пользователя
            current_user.discord_id = session['discord_id']
            # Добавляем изменения в базу данных
            db.session.commit()
            login_user(current_user)
            return redirect(url_for('home'))
        else:
            return redirect(url_for("discord_login_form"))
```

Рисунок 21 – Маршрут обратного вызова Discord

Если учетной записи с аналогичным discord_id не существует, пользователь будет перенаправлен на страницу входа (рис.22). Внешний вид страницы (рис.23).

```
<h2 class="text-center mb-4"> Вход с помощью Discord </h2> <!-- Форма входа -->
    <form method="POST" action="{{ url_for('discord_login_form') }}" class="needs-validation" novalidate>
        {{ form.hidden_tag() }} <!-- Поле для ввода имени пользователя -->
        <div class="form-group">
            {{ form.username.label(class="font-weight-bold") }}
            {{ form.username(class="form-control", placeholder="Введите ваше имя пользователя", required=True) }}
            <div class="invalid-feedback"> Пожалуйста, введите ваше имя пользователя. </div>
        </div> <!-- Поле для ввода пароля -->
        <div class="form-group">
            {{ form.password.label(class="font-weight-bold") }}
            {{ form.password(class="form-control", placeholder="Введите ваш пароль", required=True) }}
            <div class="invalid-feedback"> Пожалуйста, введите ваш пароль. </div>
        </div> <!-- Кнопка отправки формы -->
        <div class="form-group">{{ form.submit(class="btn btn-outline-dark") }}</div>
    </form>
```

Рисунок 22 – Форма входа с помощью Discord

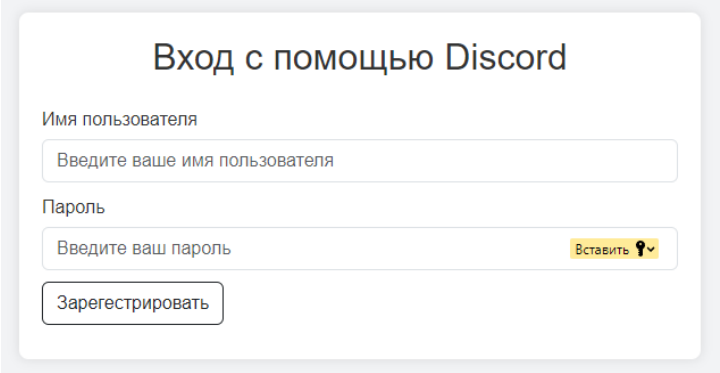

Рисунок 23 – Форма входа с помощью Discord

Затем определим маршрут для процессов регистрации и входа, обрабатывая введенные данные из соответствующей формы (рис.24).

```
@app.route("/discord_login_form", methods=['GET', 'POST'])
def discord_login_form():
    form = DiscordLoginForm()
    if form.validate_on_submit():
        # Получить данные из формы
        username = form.username.data
        password = form.password.data
        new_user = User(username=username,
                        password=password,
                        discord_id=session['discord_id'])
        db.session.add(new_user)
        db.session.commit()
            # Выполнить вход нового пользователя
        login_user(new_user)
        return redirect(url_for('home'))
    return render_template('discord_login_form.html', form=form)
```

Рисунок 24 – Маршрут обработки данных из формы, регистрации и входа Discord

После успешного входа пользователя система автоматически направит его на домашнюю страницу, где будет представлена полная информация о

его учетной записи (рис.25). Это подтвердит успешную интеграцию с аккаунтом Discord.

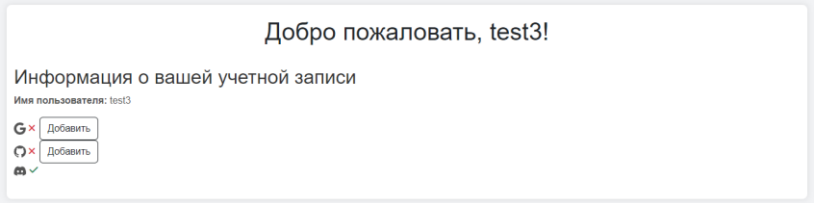

Рисунок 25 – Отображение домашней страницы

Выводы

В процессе разработки приложения для интеграции учетных записей сети интернет были успешно решены ключевые задачи, начиная от создания основы приложения и взаимодействия с API социальных сетей, и заканчивая реализацией функционала, обеспечивающего эффективную интеграцию учетных записей пользователей.

На первом этапе были настроены основные параметры приложения, создана удобная структура и определены основные цели интеграции. Затем, с применением технологии работы с API различных социальных сетей, включая обработку авторизации и получение необходимых данных, был обеспечен плавный и безопасный обмен информацией между приложением и учетными записями пользователей.

Использование современных технологий веб-разработки, таких как фреймворк Flask (или аналогичные), позволило эффективно реализовать функциональность приложения. Кроме того, применение шаблонов для отображения данных пользователю сделало интерфейс интуитивно понятным и легким в использовании.

Библиографический список

1. Юров А.Н. Разработка приложений на python. Воронеж: Воронежский государственный технический университет, 2023
2. Халевин Т.А. Основы фреймворка flask для написания web-страниц в Python // Дневник науки. 2023. № 6(78)
3. Круглик Р.И. Авторизация на сайте через google oauth api // Постулат. 2020. №1(51). С.98
4. Любанович Б. Fastapi: веб-разработка на python // Системный администратор. 2024. № 4(257). С.66-78
5. Неустроев А.В., Наумов Р.В. Использование библиотеки django oauth toolkit в django python // Проблемы современной науки и образования. 2017. №1(83). С.57-58
6. URL: https://console.cloud.google.com
7. URL: https://github.com/
8. URL: https://discord.com/developers