* 1. Аутентификация

Аутентификация

Вы можете реализовать аутентификацию пользователя ("Войти с помощью книпки X") в вашем приложении Flet, используя сторонних поставщиков идентификации, таких как GitHub, Google, Azure, Auth0, LinkedIn и другие.

Поставщик идентификации должен поддерживать поток авторизации OAuth 2.0 для получения токена доступа к API.

Встроенный логин в Flet с помощью учетных данных и управление пользователями планируется в будущих версиях. Если у вас есть требование создавать и управлять учетными записями пользователей в вашем приложении, вы можете реализовать это самостоятельно или использовать поставщика идентификации *Auth0*, который предлагает обширный бесплатный тариф.

Функции аутентификации Flet:

* + - Работает с десктопными, веб- и мобильными приложениями Flet.
    - Использование нескольких поставщиков аутентификации в одном приложении.
    - Встроенные поставщики OAuth с автоматическим извлечением данных пользователя:

1. GitHub
2. Azure
3. Google
4. Auth0
   * Опциональное извлечение групп.
   * Автоматическое обновление токена.
   * Вход с сохраненным токеном ("Запомнить меня").
   * Пользовательские (кастомные) поставщики OAuth.

# Обзор процесса логина (входа)

* + Настройте поставщика OAuth (встроенный или общего назначения) с помощью идентификатора клиента (Client ID), секретного ключа клиента (Client secret), URL-адреса перенаправления (Redirect URL).
  + Вызовите page.login(provider) , чтобы начать веб-поток OAuth.
  + Пользователь перенаправляется на веб-сайт поставщика OAuth.
  + На сайте поставщика пользователь входит в систему и дает согласие на доступ к API службы с запрошенными областями действия.
  + Сайт поставщика перенаправляет на URL-адрес обратного вызова OAuth Flet с кодом авторизации.
  + Flet обменивает код авторизации на токен и вызывает обработчик событий page.on\_login .
  + Приложение Flet может получить токен API из свойства и данные пользователя из page.auth.user .

page.auth.token

# Настройка поставщика OAuth

Flet имеет следующие встроенные поставщики OAuth:

* + GitHub
  + Azure
  + Google
  + Auth0

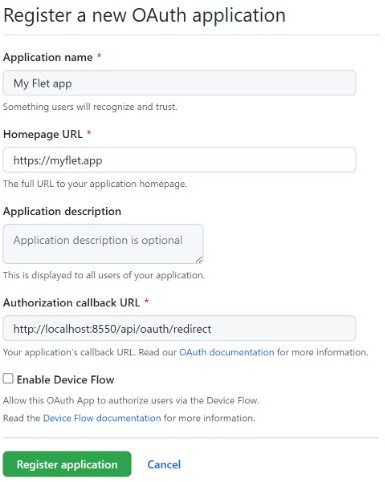
Кроме того, вы можете настроить общего поставщика OAuth и предоставить конечные точки авторизации, токена и информации пользователя.

В этом уроке мы настроим страницу логина (входа в приложение) Flet с учетной записью GitHub.

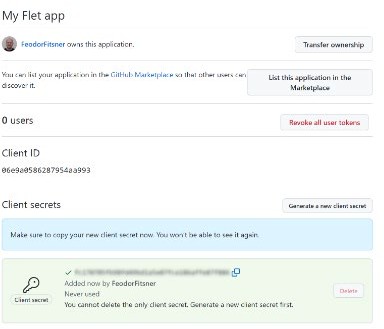
Для интеграции аутентификации Flet с GitHub сначала должно быть зарегистрировано новое приложение OAuth GitHub (Настройки профиля → Настройки разработчика → Приложения OAuth).

URL обратного вызова для авторизации должен быть в формате:

{application-url}/api/oauth/redirect



На странице "Детали приложения OAuth" нажмите кнопку "Сгенерировать новый клиентский секрет". Скопируйте значения "Client ID" и "Client secret" в надежное место - они понадобятся вам в приложении Flet.



# Вход с помощью поставщика OAuth

import os

import flet as ft

from flet.auth.providers.github\_oauth\_provider import

GitHubOAuthProvider

GITHUB\_CLIENT\_ID = os.getenv("GITHUB\_CLIENT\_ID")

assert GITHUB\_CLIENT\_ID, "set GITHUB\_CLIENT\_ID environment variable"

GITHUB\_CLIENT\_SECRET = os.getenv("GITHUB\_CLIENT\_SECRET")

assert GITHUB\_CLIENT\_SECRET, "set GITHUB\_CLIENT\_SECRET environment

variable"

def main(page: ft.Page):

provider = GitHubOAuthProvider(

client\_id=GITHUB\_CLIENT\_ID,

client\_secret=GITHUB\_CLIENT\_SECRET,

redirect\_url=["http://localhost:8550/api/oauth/redirect"](http://localhost:8550/api/oauth/redirect),

)

def login\_click(e):

page.login(provider)

def on\_login(e):

print("Ошибка входа:", e.error)

print("Токен доступа:", page.auth.token.access\_token)

print("ID пользователя:", page.auth.user.id)

page.on\_login = on\_login

page.add(ft.ElevatedButton("Войти с помощью GitHub",

on\_click=login\_click))

ft.app(target=main, port=8550, view=ft.WEB\_BROWSER)

*ВНИМАНИЕ*

*Обратите внимание, что мы получаем идентификатор клиента приложения OAuth и его секрет из переменных окружения. Не вставляйте секреты в исходный код во избежание случайного попадания в открытый доступ!*

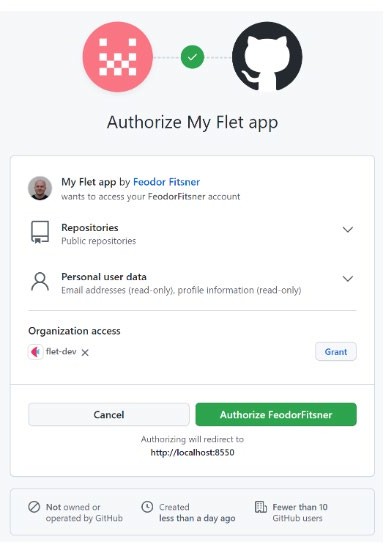
Перед запуском приложения установите секретные переменные среды в командной строке:

$ export GITHUB\_CLIENT\_ID="<client\_id>"

$ export GITHUB\_CLIENT\_SECRET="<client\_secret>"

Запустите программу и нажмите кнопку "Войти с помощью GitHub". Страница авторизации GitHub будет открыта:

* + **Desktop app** - в новом окне или вкладке браузера.
  + **Web app** - в новом всплывающем окне (убедитесь, что блокировщик всплывающих окон выключен).
  + **Mobile app** - во встроенном веб-браузере.



# Переадресация URL

Мы использовали <http://localhost:8550/api/oauth/redirect> в качестве URL-адреса переадресации при регистрации приложения OAuth GitHub. Обратите внимание, что он имеет фиксированный порт 8550 . Чтобы запустить ваше приложение Flet на

фиксированном порту, используйте аргумент порта при вызове flet.app :

ft.app(target=main, port=8550)

# Область Применения

Большинство поставщиков OAuth позволяют приложениям запрашивать одну или несколько областей применения, чтобы ограничить доступ приложения к учетной записи пользователя.

Встроенные поставщики Flet по умолчанию запрашивают области примннения для доступа к профилю пользователя, но вы можете запросить дополнительные области

применения в методе логина, например, примере:

public\_repo

в приведенном выше

page.login(

provider,

scope=["public\_repo"])

Метод аутентификации:

page.login()

fetch\_user (bool)

По умолчанию

имеет ряд аргументов для контроля процесса

- извлечь ли данные пользователя в page.auth.user .

.

True

* + fetch\_groups (bool) - извлечь ли группы пользователя в page.auth.user.groups . По умолчанию False .
  + scope - список областей применения для запроса.
  + saved\_token - снимок JSON

page.auth.token

авторизации. Токен можно сериализовать с

для восстановления

помощью

page.auth.token.to\_json()

в page.client\_storage .

, зашифровать и сохранить

* + on\_open\_authorization\_url - обратный вызов для открытия браузера с URL авторизации.

complete\_page\_html

* + - пользовательский HTML контент страницы "Вы успешно прошли аутентификацию. Закройте эту страницу".

redirect\_to\_page (bool)

* + - используется только с веб-приложением Flet,

когда страница авторизации открывается в той же вкладке браузера.

Authorization

Результат вызова следующими полями:

page.login()

* это экземпляр класса со
  + - токен OAuth, используемый для доступа к API поставщика.

|  |  |
| --- | --- |
| token | |
| user | - |

* + детали пользователя с обязательным полем id и другими полями, специфичными для поставщика OAuth.

provider

* + - экземпляр поставщика OAuth, используемый для авторизации.

Ссылка на последнюю авторизацию сохраняется в свойстве page.auth.

Если ваше приложение позволяет авторизацию с помощью нескольких поставщиков OAuth, вы можете сохранить авторизации в сессии, например:

page.session["github\_auth"] = page.login(github\_provider)

page.session["google\_auth"] = page.login(google\_provider)

# Проверка результатов аутентификации

После успешной или неудачной авторизации вызывается обработчик событий page.on\_login .

Аргумент обработчика событий e является экземпляром класса с следующими свойствами:

LoginEvent

Описание ошибки OAuth.



- Ошибка OAuth.

* error\_description (str) -

error (str)

Если error - это пустая строка, авторизация прошла успешно.

Вы можете использовать этот обработчик событий для переключения интерфейса вошел/вышел пользователь, например:

import os

import flet as ft

from flet import ElevatedButton, LoginEvent, Page

from flet.auth.providers import GitHubOAuthProvider

def main(page: Page):

provider = GitHubOAuthProvider(

client\_id=os.getenv("GITHUB\_CLIENT\_ID"),

client\_secret=os.getenv("GITHUB\_CLIENT\_SECRET"),

redirect\_url=["http://localhost:8550/api/oauth/redirect"](http://localhost:8550/api/oauth/redirect),

)

def login\_button\_click(e):

page.login(provider, scope=["public\_repo"])

def on\_login(e: ft.LoginEvent):

if not e.error:

toggle\_login\_buttons()

def logout\_button\_click(e):

page.logout()

def on\_logout(e):

toggle\_login\_buttons()

def toggle\_login\_buttons():

login\_button.visible = page.auth is None

logout\_button.visible = page.auth is not None

page.update()

login\_button = ElevatedButton("Войти с помощью GitHub",

on\_click=login\_button\_click)

logout\_button = ElevatedButton("Выйти",

on\_click=logout\_button\_click)

toggle\_login\_buttons()

page.on\_login = on\_login

page.on\_logout = on\_logout

page.add(login\_button, logout\_button)

ft.app(target=main, port=8550, view=ft.WEB\_BROWSER)

# Доступ к деталям пользователя

Если метод

вызван с параметром

(по

умолчанию), профиль пользователя будет присвоен page.auth.user .

page.login()

fetch\_user=True

Все встроенные поставщики OAuth реализуют свойство - уникальный

user.id

идентификатор пользователя - значение которого зависит от поставщика (число,

Guid или email) и может использоваться в вашем приложении в качестве ключа пользователя.

Остальные свойства профиля пользователя зависят от поставщика и могут быть доступны через индексатор. Например, для вывода некоторых свойств пользователя GitHub:

print("Имя:", page.auth.user["name"])

print("Логин:", page.auth.user["login"])

print("Email:", page.auth.user["email"])

# Использование токена OAuth

После успешной авторизации в

page.auth.token

будет содержаться токен OAuth,

который можно использовать для доступа к API поставщика. Объект Token имеет следующие свойства:

* - токен доступа, используемый как токен авторизации в заголовке запроса API.

access\_token

|  |  |
| --- | --- |
| scope |  |
| token\_type | |
| expires\_in | |

* - область применения токена.
* - тип токена доступа, например Bearer.
* - опциональное количество секунд до истечения срока действия токена доступа.

expires\_at

* - опциональное время ( time.time() + expires\_in ), когда

истекает срок действия токена доступа.

* - опциональный токен обновления, который используется для получения нового токена доступа, когда старый истекает.

refresh\_token

Обычно, для вызова API поставщика требуется только page.auth.token.access\_token. Например, чтобы перечислить репозитории пользователя на GitHub:

import requests

headers = {"Authorization": "Bearer

{}".format(page.auth.token.access\_token)}

repos\_resp = requests.get("https://api.github.com/user/repos",

headers=headers)

user\_repos = json.loads(repos\_resp.text)

for repo in user\_repos:

print(repo["full\_name"])

### ЗАМЕЧАНИЕ

*Не сохраняйте ссылку на где-либо в вашем коде, а*

*page.auth.token*

*вызывайте page.auth.token каждый раз, когда вам нужно получить токен доступа. page.auth.token - это свойство, которое автоматически обновляет токен OAuth, если/когда он истекает.*

*Правильный код:*

access\_token = page.auth.token.access\_token

*Неправильный код:*

token = page.auth.token

# какой-то другой код

access\_token = token.access\_token # токен мог истечь к этому моменту

# Сохранение и восстановление токена аутентификации

Чтобы реализовать постоянный вход ("Запомнить меня" на странице входа), вы можете сохранить токен аутентификации в клиентском хранилище и использовать его для входа в следующий раз, когда пользователь открывает ваше приложение Flet.

Чтобы сериализовать токен аутентификации в JSON:

jt = page.auth.token.to\_json()

### ВНИМАНИЕ

*Шифруйте конфиденциальные данные перед отправкой их в клиентское хранилище.*

Flet включает в себя утилиты для шифрования текстовых данных с использованием симметричного алгоритма (где один и тот же ключ используется для шифрования и дешифрования). Они используют реализацию Fernet из пакета cryptography, которая представляет собой AES 128 с некоторым дополнительным усилением, плюс PBKDF2 для получения ключа шифрования из пароля пользователя.

Чтобы зашифровать JSON токен:

import os

from flet.security import encrypt, decrypt

secret\_key = os.getenv("MY\_APP\_SECRET\_KEY")

# возвращает строку в кодировке base64

ejt = encrypt(jt, secret\_key)

*ВНИМАНИЕ*

*Обратите внимание, мы извлекаем секретный ключ (также известный как пароль, пароль и т.д.) из переменной окружения. Не встраивайте какие-либо секреты в исходный код, чтобы избежать случайного раскрытия информации публике!*

Перед запуском приложения установите секрет в командной строке:

$ export MY\_APP\_SECRET\_KEY="<secret>"

Теперь зашифрованное значение можно сохранить в клиентском хранилище:

page.client\_storage.set("myapp.auth\_token", ejt)

В следующий раз, когда пользователь открывает приложение, вы можете прочитать зашифрованный токен из клиентского хранилища и, если он существует, расшифровать его и использовать в методе page.login() :

ejt = page.client\_storage.get("myapp.auth\_token")

if ejt:

jt = decrypt(ejt, secret\_key)

page.login(provider, saved\_token=jt)

# Выход из системы

Вызов

сбрасывает ссылку

и запускает обработчики

событий page.on\_logout .

page.auth

page.logout()

Вы можете удалить сохраненный токен в методе выхода, например:

def logout\_button\_click(e):

page.client\_storage.remove(AUTH\_TOKEN\_KEY)

page.logout()

[Полный пример приложения.](https://github.com/flet-dev/examples/blob/main/python/apps/authentication/github-oauth-with-listing-repos.py)

# Настройка потока авторизации

По умолчанию, поток авторизации OAuth происходит либо в новом окне/вкладке браузера (десктопное приложение), всплывающем окне браузера (веб) или внутри приложения в веб-виде (мобильные устройства).

По завершении процесса авторизации пользователь перенаправляется на страницу

обратного вызова OAuth ( ), которая пытается закрыть

/api/oauth/redirect

окно/вкладку браузера с помощью JavaScript и предоставляет пользователю инструкции по закрытию окна вручную, если закрытие с помощью JavaScript не сработало.

Данный раздел применим только к десктопным и веб-приложениям Flet, так как веб- просмотр в мобильных приложениях может быть закрыт Flet без использования JavaScript.

Содержимое страницы "Авторизация завершена" можно настроить, например, в методе page.login() :

complete\_page\_html = """

<!DOCTYPE html>

<html>

<head>

<title>Вход в MyApp выполнен</title>

</head>

<body>

<script type="text/javascript">

window.close();

</script>

<p>Вы успешно вошли в систему! Теперь вы можете закрыть эту вкладку

или окно.</p>

</body>

</html>

"""

page.login(

provider,

complete\_page\_html=complete\_page\_html,

)

Также вы можете изменить веб-приложение так, чтобы страница авторизации провайдера открывалась в той же вкладке, что может быть более привычно для

ваших пользователей и избавит их от необходимости иметь дело с блокировщиками всплывающих окон:

page.login(

provider,

on\_open\_authorization\_url=lambda url: page.launch\_url(url,

web\_window\_name="\_self"),

redirect\_to\_page=True

)

Чтобы открыть поток в новой вкладке (обратите внимание,

\_self

заменяется

на \_blank ):

page.login(

provider,

on\_open\_authorization\_url=lambda url: page.launch\_url(url,

web\_window\_name="\_blank")

)

# Настройка кастомного поставщика OAuth

Вы можете настроить любого совместимого с OAuth провайдера аутентификации в вашем приложении с помощью

класса flet.auth.oauth\_provider.OAuthProvider .

Следуя инструкциям в руководстве [LinkedIn Authorization Code Flow,](https://learn.microsoft.com/en-us/linkedin/shared/authentication/authorization-code-flow?context=linkedin%2Fcontext&tabs=HTTPS) мы получаем все необходимые параметры для настройки поставщика OAuth LinkedIn, чтобы позволить пользователям вашего приложения Flet входить в систему с использованием их учетных записей LinkedIn:

import os

import flet as ft

from flet import ElevatedButton, Page

from flet.auth import OAuthProvider

def main(page: Page):

provider = OAuthProvider(

client\_id=os.getenv("LINKEDIN\_CLIENT\_ID"),

client\_secret=os.getenv("LINKEDIN\_CLIENT\_SECRET"),

authorization\_endpoint=["https://www.linkedin.com/oauth/v2/authorizatio](http://www.linkedin.com/oauth/v2/authorizatio)

n",

token\_endpoint=["https://www.linkedin.com/oauth/v2/accessToken"](http://www.linkedin.com/oauth/v2/accessToken),

redirect\_url=["http://localhost:8550/api/oauth/redirect"](http://localhost:8550/api/oauth/redirect),

user\_endpoint="https://api.linkedin.com/v2/me",

user\_scopes=["r\_liteprofile", "r\_emailaddress"],

user\_id\_fn=lambda u: u["id"]

)

def login\_click(e):

page.login(provider)

def on\_login(e):

if e.error:

raise Exception(e.error)

print("ID пользователя:", page.auth.user.id)

print("Токен доступа:", page.auth.token.access\_token)

page.on\_login = on\_login

page.add(ElevatedButton("Войти через LinkedIn",

on\_click=login\_click))

ft.app(target=main, port=8550, view=ft.WEB\_BROWSER)

Обязательные настройки провайдера:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| client\_id |  | | | |
| client\_secret | | |  | |
| authorization\_endpoint | | | | |
| token\_endpoint | | | |  |
| redirect\_url | |  | | |

По аналогии с другими примерами, идентификатор клиента и секретный ключ клиента извлекаются из переменных среды.