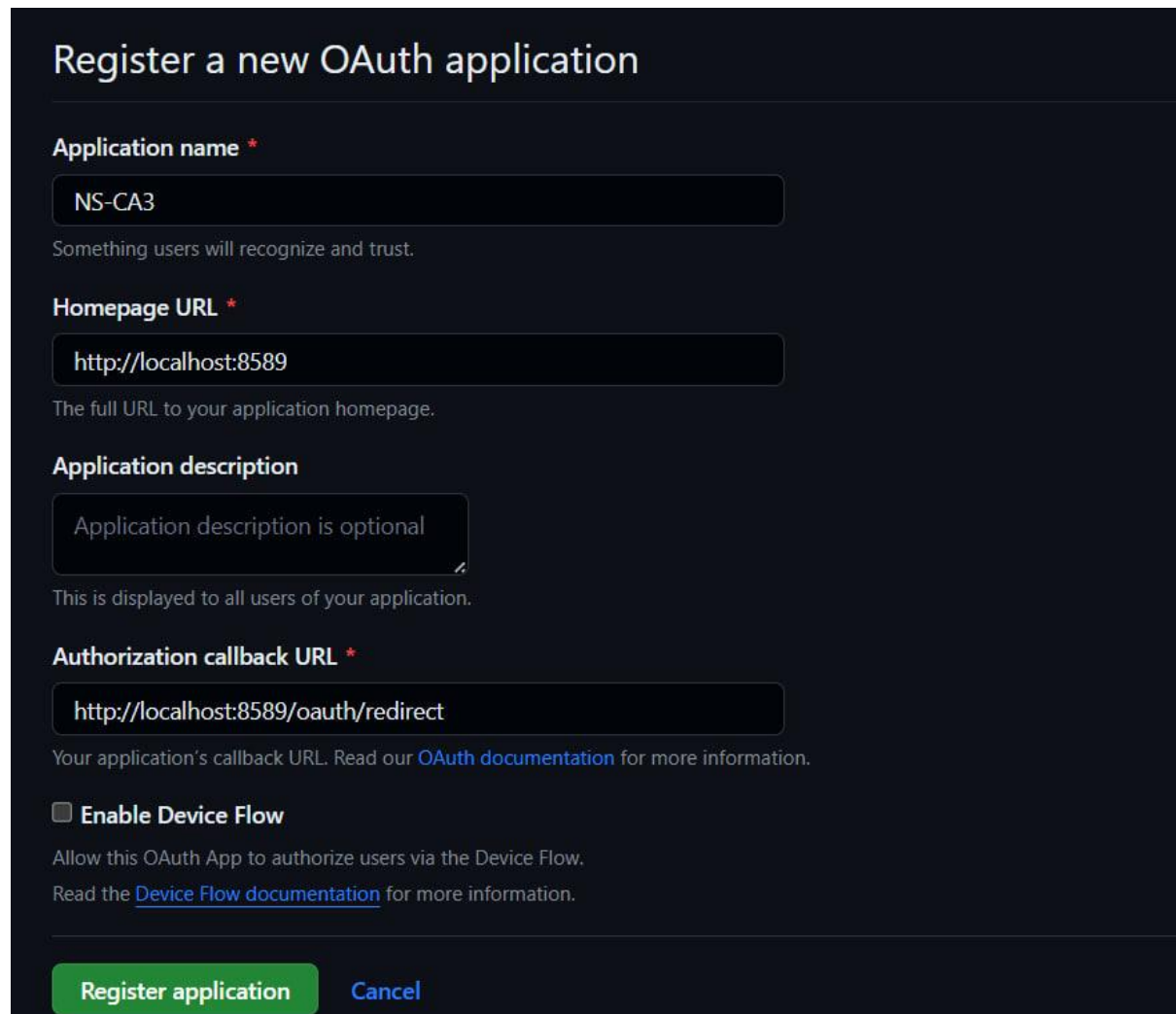


1) دستورالعمل

1. ساخت OAuth Application

در این مرحله، در صفحه گیت‌هاب شخصی، یک OAuth Application ایجاد می‌کنیم:



Register a new OAuth application

Application name *
NS-CA3
Something users will recognize and trust.

Homepage URL *
http://localhost:8589
The full URL to your application homepage.

Application description
Application description is optional
This is displayed to all users of your application.

Authorization callback URL *
http://localhost:8589/oauth/redirect
Your application's callback URL. Read our [OAuth documentation](#) for more information.

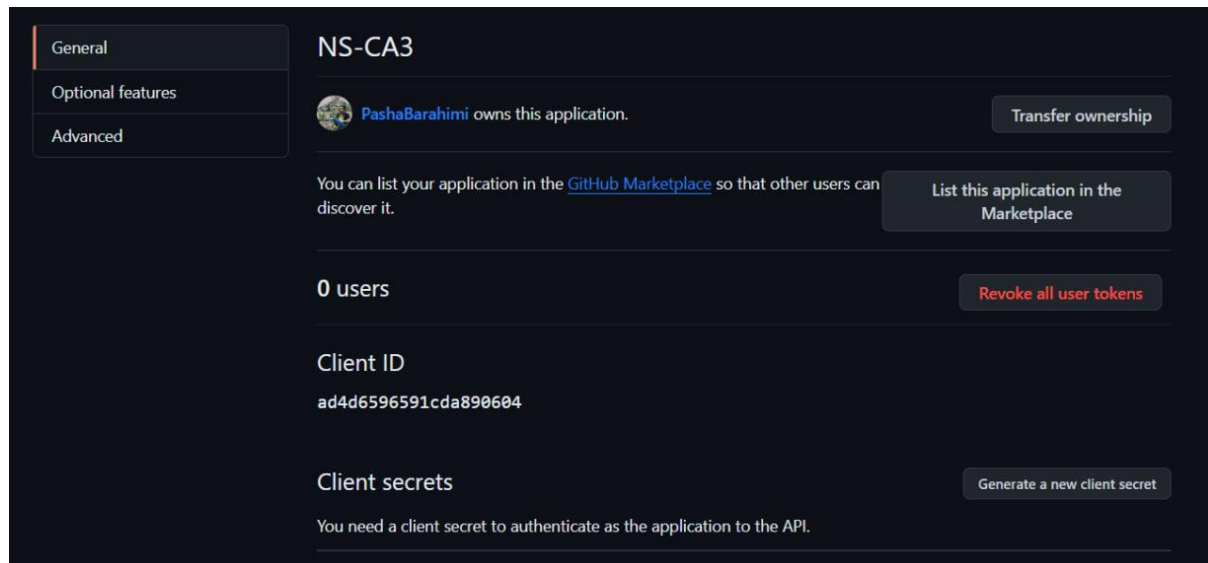
☐ **Enable Device Flow**
Allow this OAuth App to authorize users via the Device Flow.
Read the [Device Flow documentation](#) for more information.

Register application **Cancel**

در این بخش همانطور که در صورت مسئله ذکر شده، فیلد callback URL برابر با مقدار زیر قرار داده شده است:

<http://localhost:8589/oauth/redirect>

در تصویر زیر مشاهده می‌کنیم که این اپلیکیشن ساخته شده است:

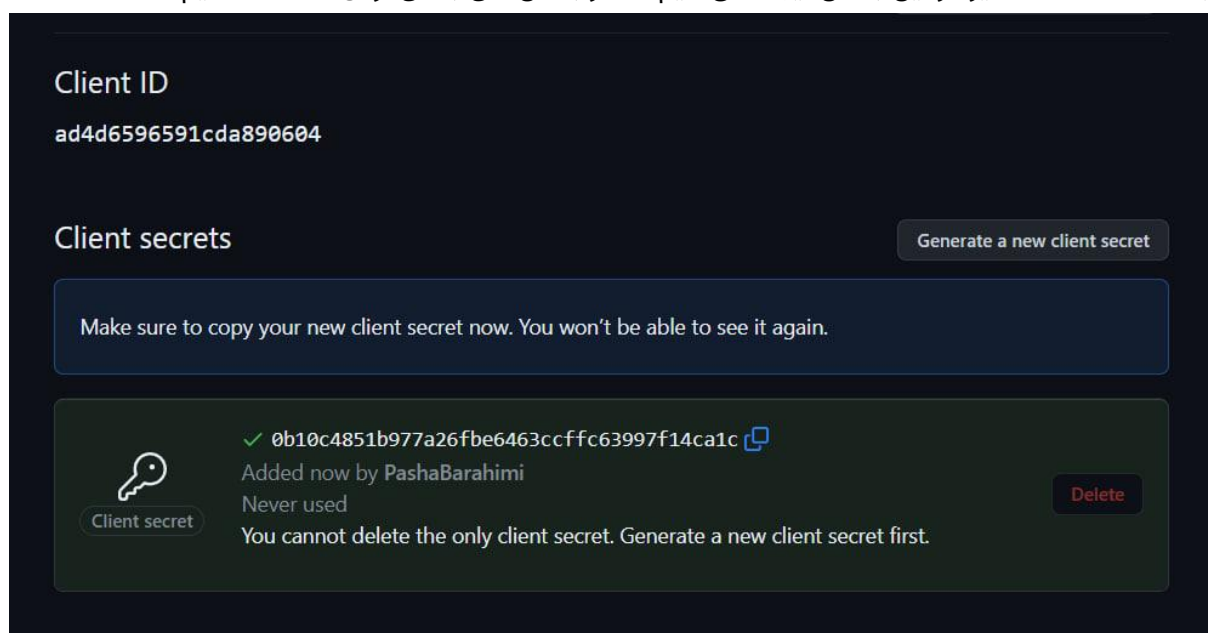


The screenshot shows the 'General' tab of the GitHub application settings for 'NS-CA3'. It indicates that 'PashaBarahimi' owns the application. There are buttons for 'Transfer ownership', 'List this application in the Marketplace', 'Revoke all user tokens', and 'Generate a new client secret'. The 'Client ID' is displayed as 'ad4d6596591cda890604'. A note states that a client secret is needed to authenticate the application to the API.

در این بخش یک Client ID داریم که برابر با مقدار زیر است:

ad4d6596591cda890604

یک client secret نیز در این بخش ایجاد می‌کنیم که در بخش‌های بعدی از آن استفاده کنیم:



The screenshot shows the 'Client secrets' section of the GitHub application settings. It displays the 'Client ID' as 'ad4d6596591cda890604'. There is a button to 'Generate a new client secret'. A message states: 'Make sure to copy your new client secret now. You won't be able to see it again.' Below this, a new client secret is generated: '0b10c4851b977a26fbe6463ccfffc63997f14ca1c'. It is noted as 'Added now by PashaBarahimi' and 'Never used'. A 'Delete' button is present. A warning message states: 'You cannot delete the client secret. Generate a new client secret first.'

مقدار client secret در ادامه قرار داده شده است:

0b10c4851b977a26fbe6463ccfffc63997f14ca1c

2. اجرای سرور

پس از نصب پیش‌نیازهای سرور، آن را روی پورت 8589 اجرا می‌کنیم. در این مرحله، کد سرور به صورت زیر است:

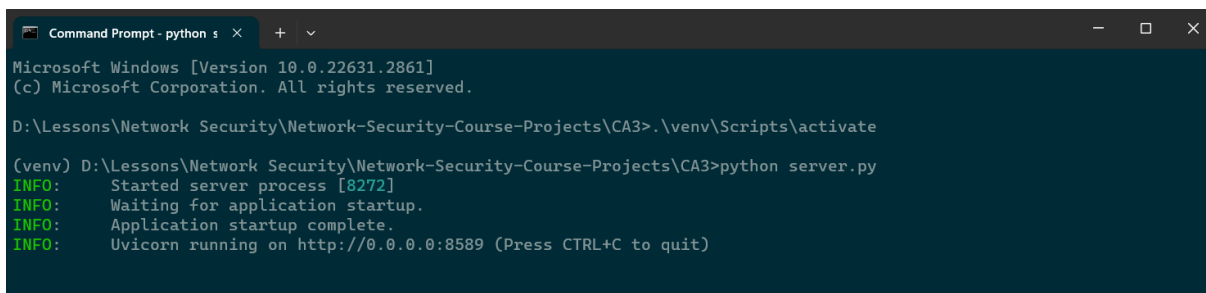
```
import uvicorn
from fastapi import FastAPI

app = FastAPI()

@app.get("/oauth/redirect")
def oauth_redirect(code: str):
    print(f'Github code is: {code}')
    return f'Github code is: {code}'

if (__name__ == '__main__'):
    uvicorn.run(app, host= '0.0.0.0', port = 8589)
```

سرور به صورت زیر اجرا می‌شود:



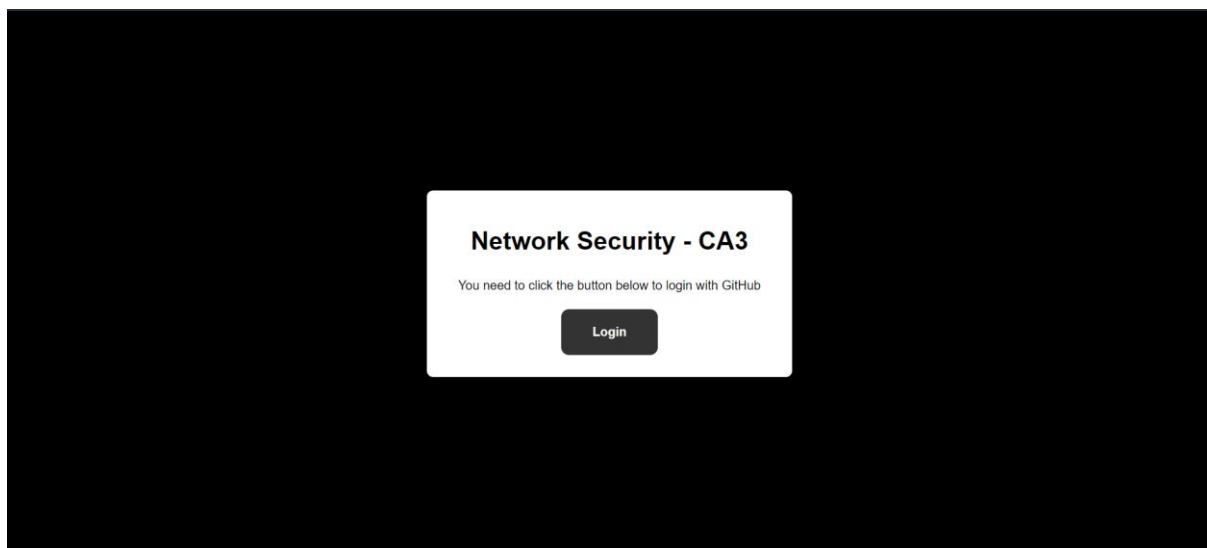
```
Command Prompt - python s
Microsoft Windows [Version 10.0.22631.2861]
(c) Microsoft Corporation. All rights reserved.

D:\Lessons\Network Security\Network-Security-Course-Projects\CA3>.venv\Scripts\activate

(venv) D:\Lessons\Network Security\Network-Security-Course-Projects\CA3>python server.py
INFO: Started server process [8272]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8589 (Press CTRL+C to quit)
```

3. ایجاد صفحه HTML

صفحه HTML به صورت زیر ایجاد شده است:



کد آن نیز به صورت زیر است:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <title>Pasha Barahimi</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #000000;
      margin: 0;
      padding: 0;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
    }

    .container {
      text-align: center;
      max-width: 600px;
      background-color: #fff;
      padding: 25px 40px;
      border-radius: 8px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }

    .container p {
      margin-top: 30px;
      display: block;
    }

    .login-btn {
      display: inline-block;
      background-color: #333;
      color: #fff;
      padding: 20px 40px;
      margin: 5px 0px 3px 0px;
      text-decoration: none;
      border-radius: 10px;
      font-weight: bold;
      transition: background-color 0.3s ease;
    }

    .login-btn img {
      height: 15px;
      width: 15px;
      margin-right: 8px;
    }

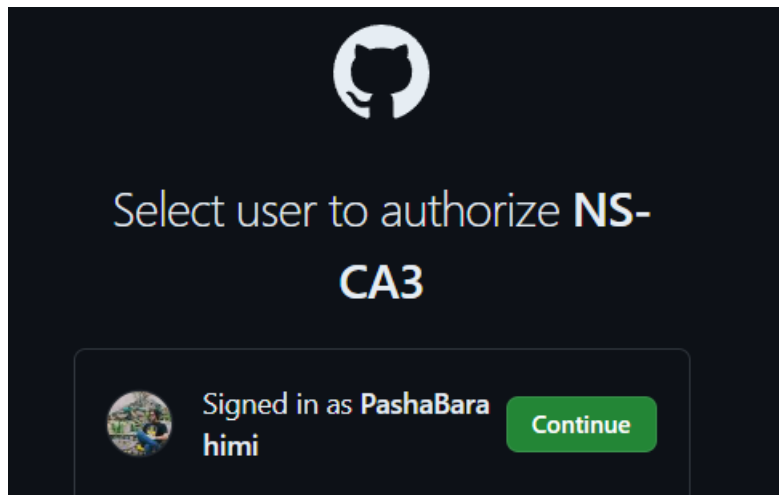
    .login-btn:hover {
      background-color: #555;
    }
  </style>
</head>

<body>
  <div class="container">
    <h1>Network Security - CA3</h1>
    <p>You need to click the button below to login with GitHub</p>
    <a href="https://github.com/login/oauth/authorize?client_id=
ad4d6596591cda890604&redirect_uri=http://localhost:8589/oauth/redirect"
      class="login-btn">
      Login
    </a>
  </div>
</body>

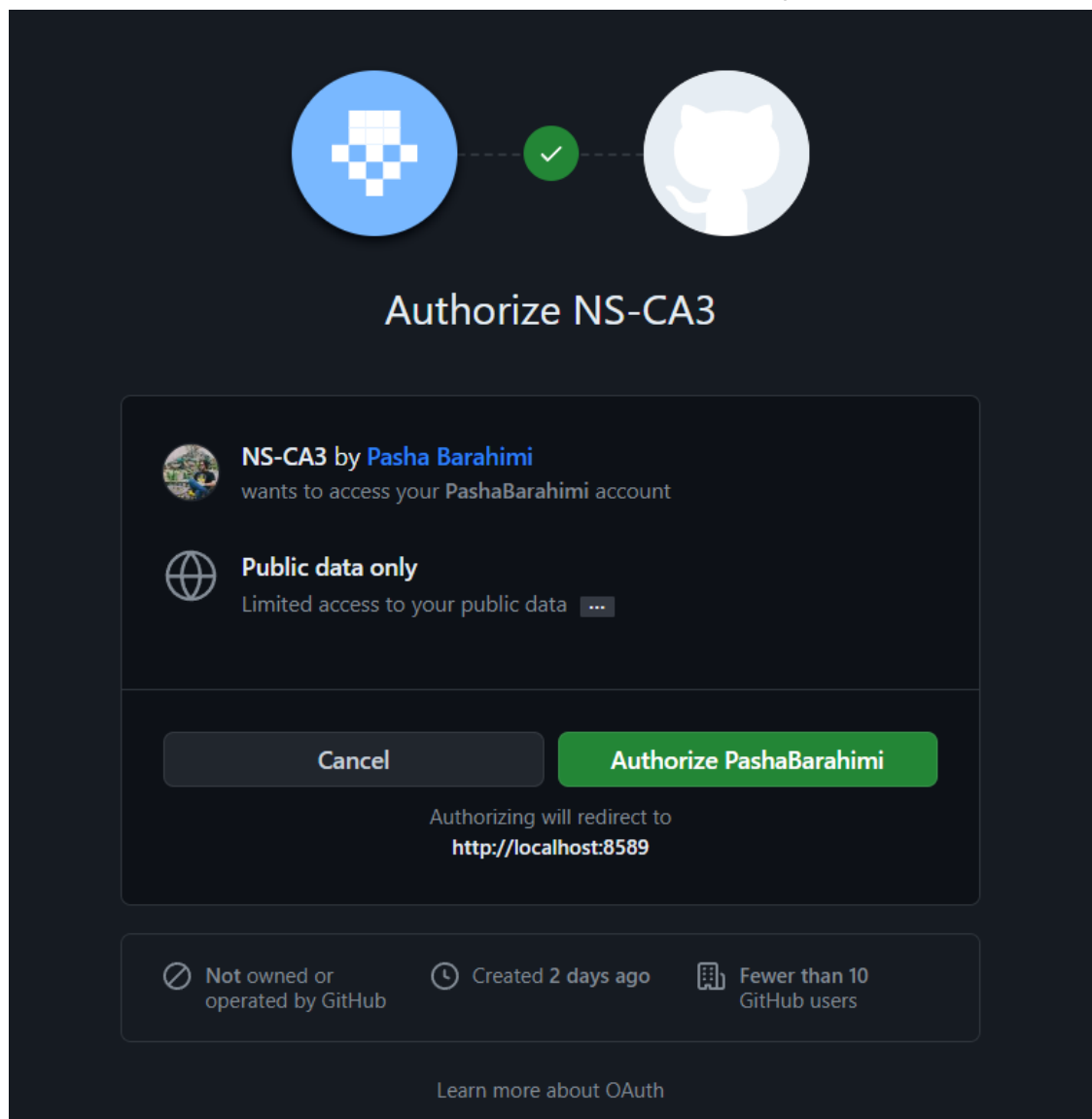
</html>
```

4. چاپ پارامتر code

حال روی دکمه کلیک کرده و مرحله authentication را انجام می‌دهیم:



سپس باید authorization انجام شود:



سپس روی Authorize کلیک می‌کنیم:

```
"Github code is: 14fbc12d01b68206b0f9"
```

مقدار code در ادامه قرار داده شده است:

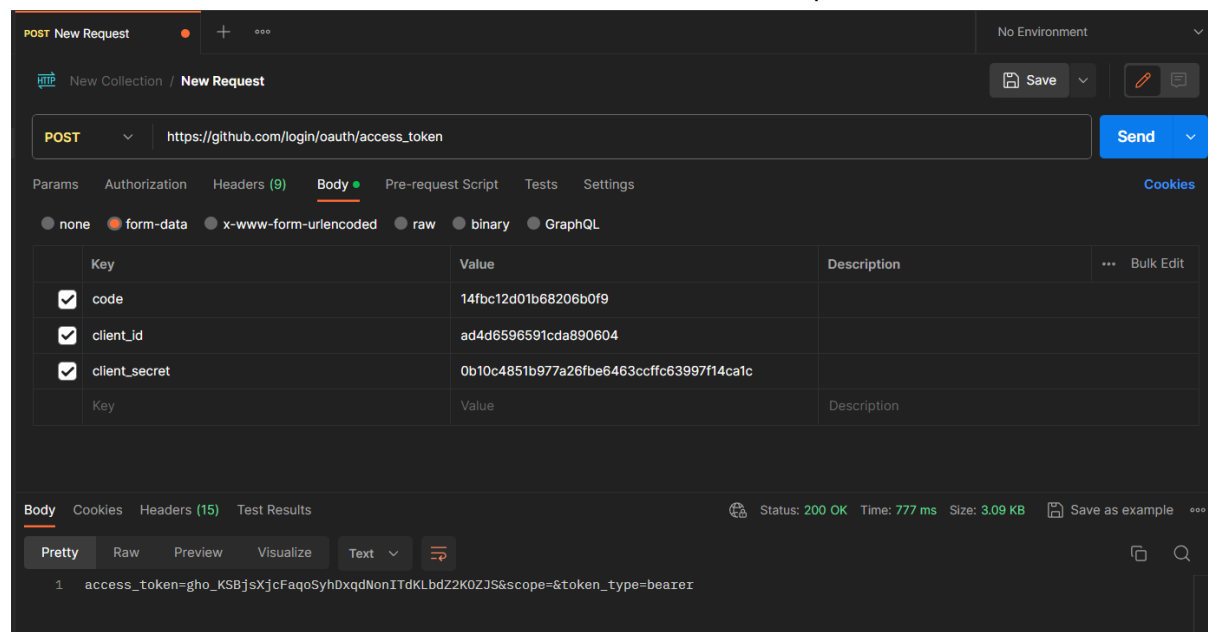
14fbc12d01b68206b0f9

در ترمینال نیز خروجی زیر را خواهیم داشت:

```
Github code is: 14fbc12d01b68206b0f9
INFO: 127.0.0.1:63719 - "GET /oauth/redirect?code=14fbc12d01b68206b0f9 HTTP/1.1" 200 OK
```

5. دریافت Access Token

حال به کمک postman، یک ریکوئست به گیت‌هاب زده و با دادن مقادیر code، client_id و client_secret، یک access token دریافت می‌کنیم:

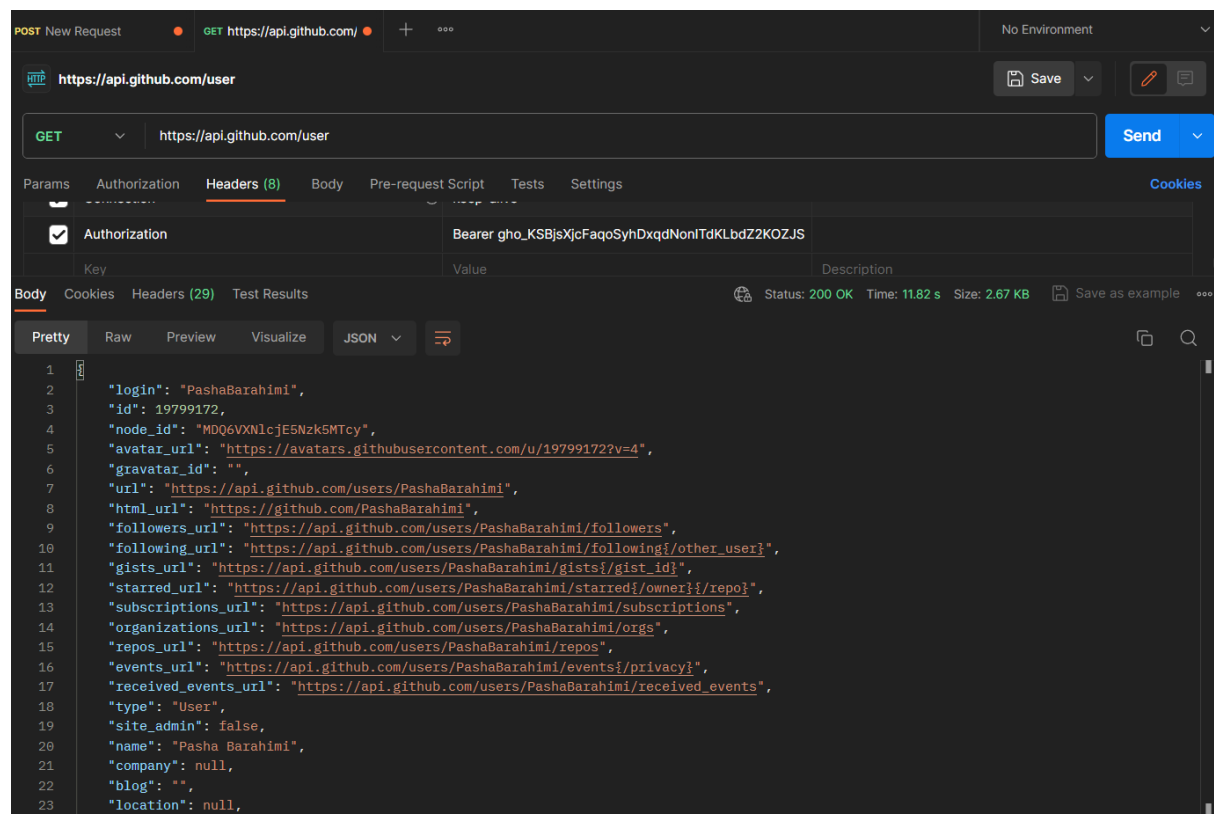


همانطور که دیده می‌شود، این توکن از نوع Bearer است. مقدار آن در ادامه آمده است:

gho_KSBjsXjcFaqoSyhDxqdNonITdKLbdZ2K0ZJS

6. دریافت اطلاعات کاربر از طریق API

حال به کمک api گیت‌هاب و توکن دریافت شده، اطلاعات یوزر را دریافت می‌کنیم:



همانطور که مشاهده می‌شود، اطلاعات یوزر به درستی دریافت شده است.

7. خودکار کردن مراحل

حال با تغییر `server.py` به صورت زیر، مراحل انجام شده را به صورت خودکار انجام خواهیم داد:

```
import requests
import json

import uvicorn
from fastapi import FastAPI

CLIENT_ID = "ad4d6596591cda890604"
CLIENT_SECRET = "0b10c4851b977a26f6e6463ccffc63997f14ca1c"

app = FastAPI()

def get_access_token(code: str) -> str:
    access_token_url = "https://github.com/login/oauth/access_token"
    payload = {
        "client_id": CLIENT_ID,
        "client_secret": CLIENT_SECRET,
        "code": code
    }
    headers = {
        "Accept": "application/json"
    }
    response = requests.post(access_token_url, data=payload,
headers=headers)
    access_token = response.json()[ 'access_token' ]
    return access_token

def get_user_details(access_token: str) -> dict:
    api_url = "https://api.github.com/user"
    headers = {
        "Authorization": f"Bearer {access_token}"
    }
    response = requests.get(api_url, headers=headers)
    return response.json()

@app.get("/oauth/redirect")
def oauth_redirect(code: str) -> dict:
    access_token = get_access_token(code)
    user_details = get_user_details(access_token)
    print(f'Github code is: {code}')
    print(f'Github access token is: {access_token}')
    print(f'Github user details are: {json.dumps(user_details,
indent=4)}')
    return user_details

if (__name__ == '__main__'):
    uvicorn.run(app, host= '0.0.0.0', port = 8589)
```

حال اگر مرحله authentication را انجام دهیم، پاسخ به صورت زیر خواهد بود:

```
{ "login": "PashaBarahimi", "id": 19799172, "node_id": "MDQ6V0lucjE5Nzk5MTcy", "avatar_url": "https://avatars.githubusercontent.com/u/19799172?v=4", "gravatar_id": "", "url": "https://api.github.com/users/PashaBarahimi", "html_url": "https://github.com/PashaBarahimi", "followers_url": "https://api.github.com/users/PashaBarahimi/followers", "following_url": "https://api.github.com/users/PashaBarahimi/following{/other_user}", "gists_url": "https://api.github.com/users/PashaBarahimi/gists{/gist_id}", "starred_url": "https://api.github.com/users/PashaBarahimi/starred{/owner}{/repo}", "subscriptions_url": "https://api.github.com/users/PashaBarahimi/subscriptions", "organizations_url": "https://api.github.com/users/PashaBarahimi/orgs", "repos_url": "https://api.github.com/users/PashaBarahimi/repos", "events_url": "https://api.github.com/users/PashaBarahimi/events{/privacy}", "received_events_url": "https://api.github.com/users/PashaBarahimi/received_events", "type": "User", "site_admin": false, "name": "Pasha Barahimi", "company": null, "blog": "", "location": null, "email": "pashabarahimi@gmail.com", "hireable": null, "bio": "Computer Engineering Student, University of Tehran", "twitter_username": null, "public_repos": 17, "public_gists": 1, "followers": 83, "following": 67, "created_at": "2016-06-07T12:42:13Z", "updated_at": "2023-12-10T11:43:29Z" }
```


در ترمینال نیز خروجی زیر را خواهیم داشت:

```
INFO: Started server process [16240]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8589 (Press CTRL+C to quit)
Github code is: a33b4e22ec6a606e3678
Github access token is: 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
Github user details are: {
  "login": "PashaBarahimi",
  "id": 19799172,
  "node_id": "MDQ6VXNlcjE5Nzk5MTcy",
  "avatar_url": "https://avatars.githubusercontent.com/u/19799172?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/PashaBarahimi",
  "html_url": "https://github.com/PashaBarahimi",
  "followers_url": "https://api.github.com/users/PashaBarahimi/followers",
  "following_url": "https://api.github.com/users/PashaBarahimi/following{/other_user}",
  "gists_url": "https://api.github.com/users/PashaBarahimi/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/PashaBarahimi/starred{/owner}/{/repo}",
  "subscriptions_url": "https://api.github.com/users/PashaBarahimi/subscriptions",
  "organizations_url": "https://api.github.com/users/PashaBarahimi/orgs",
  "repos_url": "https://api.github.com/users/PashaBarahimi/repos",
  "events_url": "https://api.github.com/users/PashaBarahimi/events{/privacy}",
  "received_events_url": "https://api.github.com/users/PashaBarahimi/received_events",
  "type": "User",
  "site_admin": false,
  "name": "Pasha Barahimi",
  "company": null,
  "blog": "",
  "location": null,
  "email": "pashabarahimi@gmail.com",
  "hireable": null,
  "bio": "Computer Engineering Student, University of Tehran",
  "twitter_username": null,
  "public_repos": 17,
  "public_gists": 1,
  "followers": 83,
  "following": 67,
  "created_at": "2016-06-07T12:42:13Z",
  "updated_at": "2023-12-10T11:43:29Z"
}
```

مشاهده می‌شود که مراحل به صورت کامل خودکار شده‌اند.

(2) سوالات

1. مزایای استفاده از Authorization Code Grant Type

برخی فواید استفاده از این روش در ادامه ذکر شده است:

امنیت بهتر

این روش به دلیل جدا کردن مراحل authentication و authorization، امنیت مناسبی را فراهم می‌کند. در این روش، کلاینت ابتدا یک کد authorization دریافت کرده و سپس در ازای آن، می‌تواند access token بگیرد. در واقع عمر کم code باعث می‌شود خطر افشا و سوء استفاده از آن کاهش یابد. همچنین، client secret در دسترس کاربر نیست.

توانایی استفاده عمومی و پشتیبانی مناسب

این روش برای اکثر استفاده‌های عمومی مناسب است و همچنین، در بسیاری از سرویس‌ها پشتیبانی می‌شود.

قابلیت Refresh Token

این روش به ما این امکان را می‌دهد که در هنگام اکسپایر شدن توکن، بدون نیاز به وارد کردن مجدد اطلاعات، بتوانیم توکن را renew کنیم.

توانایی احراز هویت کلاینت

در این روش می‌توانیم خود کلاینت را هم احراز هویت کنیم که می‌تواند امنیت را افزایش دهد.

2. ضعف(های) امنیتی استفاده از Client Credential Grant Type در یک نرم‌افزار تلفن همراه

این روش معمولاً برای ارتباط server-to-server استفاده می‌شود و برای مواقعی است که اپلیکیشن از طرف خودش احراز هویت را انجام می‌دهد و کاربر واقعی را درگیر نمی‌کند. اگر بخواهیم از این روش در نرم‌افزار تلفن همراه استفاده کنیم، می‌تواند سبب مشکلات زیر شود:

عدم احراز هویت توسط کاربر

همانطور که پیش‌تر ذکر شد، یکی از موارد این روش این است که کلاینت از طرف خودش احراز هویت را انجام می‌دهد و کاربر را درگیر نمی‌کند. به همین دلیل، زمانی که از آن در نرم‌افزار تلفن همراه استفاده کنیم، نرم‌افزار می‌تواند بدون اینکه کاربر متوجه شود، از طرف او احراز هویت را انجام دهد. این کار سبب می‌شود امنیت کاربر کاهش یابد. در واقع در این حالت کلاینت بدون اجازه کاربر می‌تواند از منابع او و سرویس مد نظر استفاده کند.

فاش شدن توکن

در این حالت، معمولاً توکن مد نظر در اپلیکیشن قرار گرفته و در صورتی که مهاجم بتواند به باینری اپلیکیشن و یا تلفن همراه دست پیدا کند، می‌تواند سبب سوء استفاده از منابع و سرویس شود.

عدم وجود مکانیزم برای ثابت کردن داشتن توکن

با توجه به اینکه مکانیزمی برای اثبات داشتن توکن در این روش وجود ندارد، این روش می‌تواند حمله replay attack را ممکن کند.

3. نوع و روش تولید این Access Token و توانایی decode آن

توکن ساخته شده از نوع Bearer است و در سرور ساخته شده و به کلاینت داده می‌شود. کلاینت در هر ریکوئست، این توکن را در هدر Authorization قرار می‌دهد که سرور بتواند او را authorize و authenticate کند. در پروتکل OAuth2 هیچ فرمت دقیقی برای ساخت این نوع توکن ذکر نشده و می‌تواند به روش‌های متفاوتی ساخته شود. روشی که به طور معمول استفاده می‌شود، ساخت یک رشته رندوم است که در این حالت، هیچ چیزی برای decode کردن وجود ندارد. اما برای ساخت این توکن می‌توان از روش‌هایی نظیر JWT نیز استفاده کرد. توکن‌های JWT شامل برخی موارد هستند که با Base64 انکود شده‌اند و با decode کردن آن‌ها می‌توانیم به این اطلاعات دست پیدا کنیم. این موارد می‌توانند شامل هویت کاربر و زمان اعتبار توکن و ... باشد که در بخش‌های header و payload قرار می‌گیرند و در انتها، یک signature خواهیم داشت که ساخته شدن توکن توسط سرور و valid بودن آن را اثبات می‌کند. توکن دریافت شده از گیت‌هاب رشته‌ای رندوم بوده و نمی‌توانیم آن را دیکود کنیم.

4. ضعف(های) امنیتی این برنامه در محیط Production

از مشکلات استفاده از این برنامه در محیط Production می‌توان به موارد زیر اشاره کرد:

- با توجه به اینکه Client ID و Client Secret در این برنامه به صورت hardcode شده قرار گرفته‌اند، مهاجم در صورت دسترسی به کلاینت می‌تواند به این اطلاعات دست پیدا کند و از آن سوء استفاده کند. برای رفع محدودی این مشکل می‌توانیم از روش‌های دیگری برای ذخیره امن این اطلاعات استفاده کنیم.
- در حالت فعلی، ارتباط بین گیت‌هاب و کلاینت به صورت HTTP است که باعث می‌شود مهاجم بتواند حمله MIDM انجام دهد یا اطلاعات (توکن‌ها) را مشاهده کند و به این صورت می‌تواند از این اطلاعات سوء استفاده کند. برای حل این مسئله می‌توانیم به کمک یک گواهی SSL، ارتباط را به صورت HTTPS برقرار کنیم.
- در این حالت حمله فیشینگ برای صفحه‌ای که به آن redirect می‌شویم نیز ممکن است که می‌تواند آسیب‌های بزرگی را برای برنامه ما به همراه داشته باشد.