



به نام خدا  
دانشگاه تهران  
پردیس دانشکدگان فنی  
دانشکده مهندسی برق و کامپیوتر



تمرین کامپیوتری شماره 3 درس مبانی امنیت شبکه‌های کامپیوتری

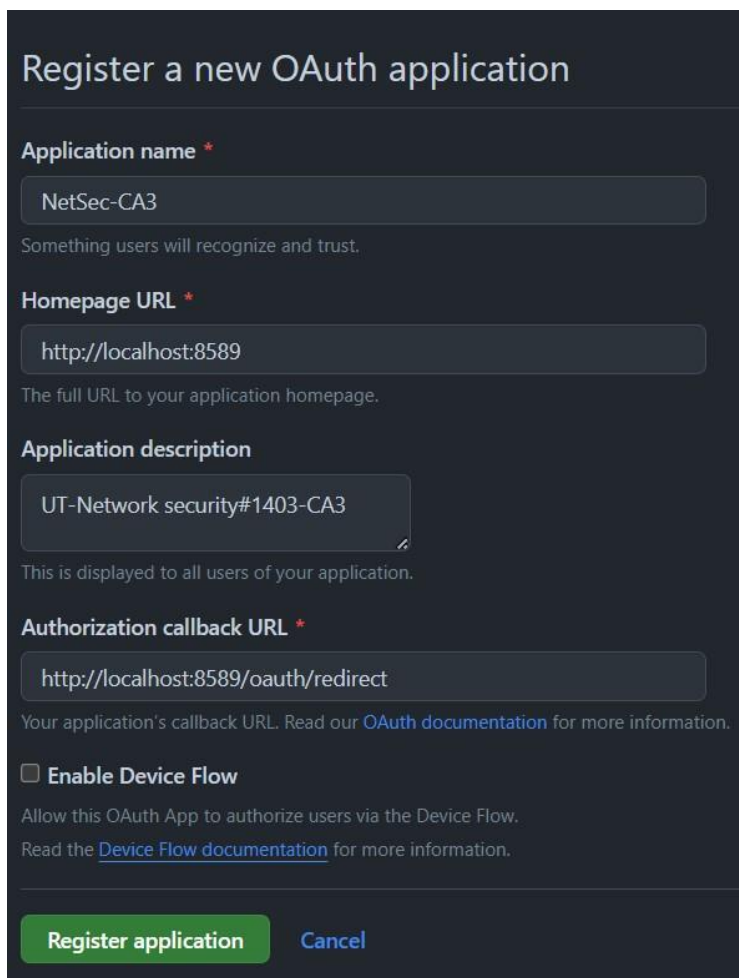
نام استاد : دکتر صیادحقیقی

نام دانشجو : علی مهرانی

شماره دانشجویی : 810198542

## 1. ایجاد OAuth App در حساب Github

مطابق شکل زیر یک OAuth App در گیتهاب با نام NetSec-CA3 ایجاد کرده و اطلاعات مربوط به URL را نیز در اختیار آن قرار می‌دهیم.



Register a new OAuth application

**Application name \***

NetSec-CA3

Something users will recognize and trust.

**Homepage URL \***

http://localhost:8589

The full URL to your application homepage.

**Application description**

UT-Network security#1403-CA3

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.

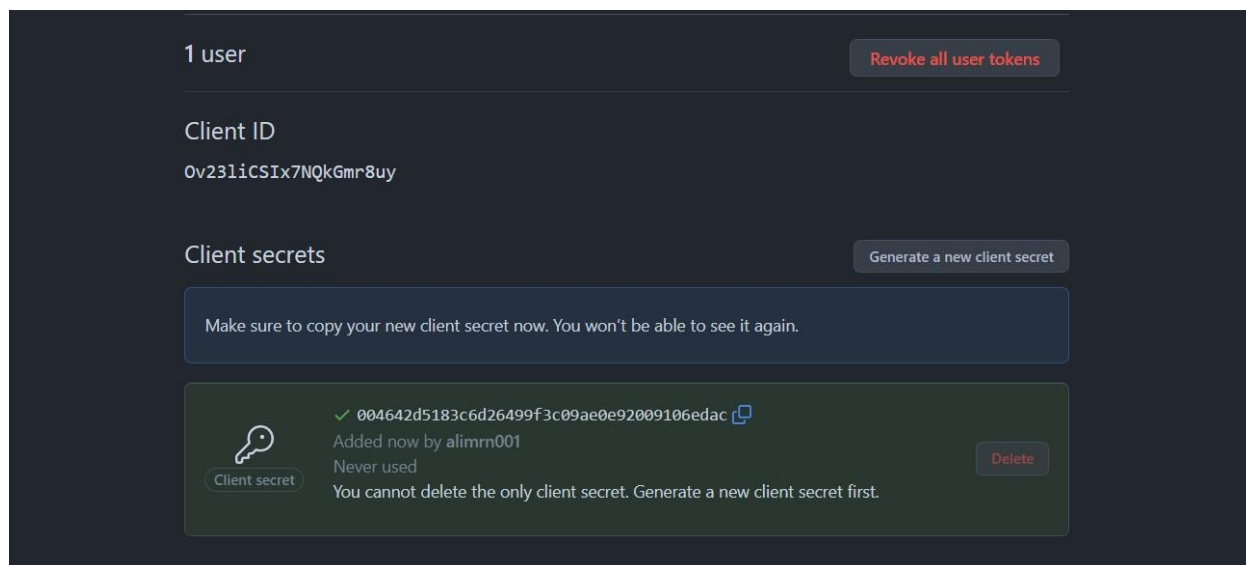
☐ **Enable Device Flow**

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

**Register application** [Cancel](#)

شکل 1- ایجاد OAuth App

سپس برنامه ذکر شده را register می‌کنیم. در ادامه پس از register کردن برنامه ذکر شده client\_id مربوط به این برنامه نشان داده می‌شود و همچنین در ادامه یک client\_secret نیز ایجاد می‌کنیم که شکل زیر نشان‌دهنده آن می‌باشد.



شکل 2- نمایش client\_id و client\_secret

## 2. اجرای برنامه Server

پس از موفقیت در ساخت برنامه OAuth برنامه سرور داده شده را اجرا می‌کنیم. Package های fastapi و uvicorn با استفاده از دستورات ذکر شده با موفقیت بر روی سیستم نصب شدند. با نصب موفقیت‌آمیز آن‌ها، فایل server بدون خطا اجرا می‌شود اما برای کار کردن درست آن باید در ادامه موارد ذکرشده را انجام دهیم.

## 3. ایجاد صفحه login

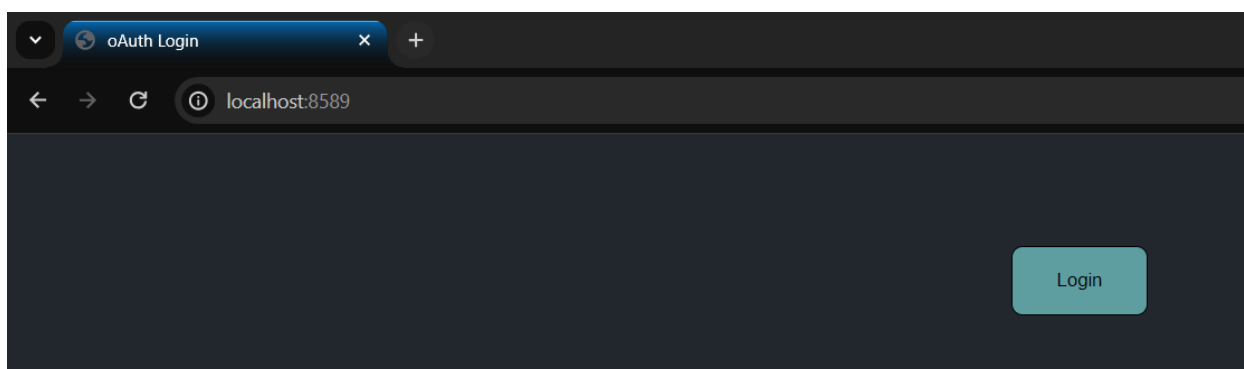
برای امکان login با استفاده از احراز هویت Github باید یک صفحه ساده login بسازیم که کاربر با کلیک برروی دکمه متناظر، وارد صفحه احراز هویت Github شود. شکل زیر صفحه HTML ایجاد شده را نمایش می‌دهد.

```

    .main-container {
      display: flex;
      justify-content: center;
    }
  </style>
</head>
<body>
  <div class="main-container">
    <button class="login-button">
      <a
        class="oauth-link"
        href="https://github.com/login/oauth/authorize?client_id=0v23liCSIx7NQkGmr8uy"
      >Login</a>
    </button>
  </div>
</body>
</html>

```

شکل 3- فایل login.html



شکل 4- serve شدن فایل login.html بر روی سرور

همانگونه که در شکل بالا نیز مشخص است، فایل login.html ایجاد شده بر روی index سرور مربوطه serve و نمایش داده شده است. برای انجام این کار و serve کردن فایل مذکور موارد زیر انجام داده شده‌اند.

باید از یک Template Engine برای serve کردن و نمایش دادن فایل‌های html استفاده کنیم که در اینجا از Jinja2 استفاده کرده‌ایم که بخشی از مازول templating در fastapi است. در ابتدا آدرس فایل‌های static یا template خود را مشخص می‌کنیم و سپس فایل مورد نظر را در endpoint مورد نظر نمایش می‌دهیم که شکل زیر کدهای مربوط به این فرآیند را نمایش می‌دهد.

```

app = FastAPI()
templates = Jinja2Templates(directory="templates")

> def get_token_from_encoded_url(response): ...

> def get_auth_token(github_code): ...

> def get_user_data_with_token(token): ...

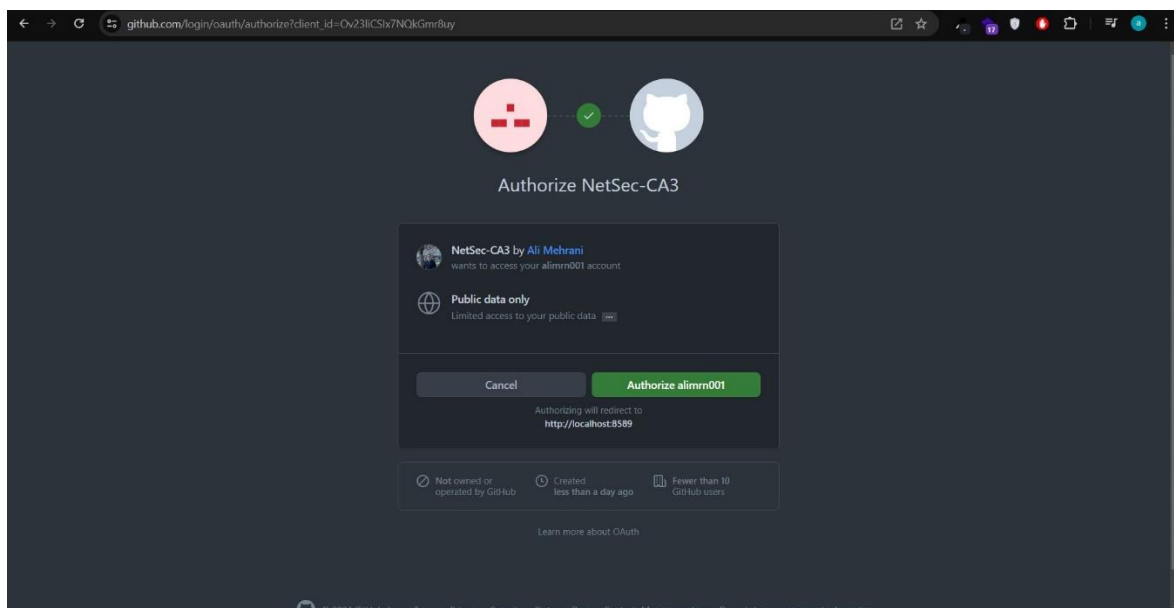
@app.get("/oauth/redirect")
> def oauth_redirect(request: Request, code: str): # add request as parameter

@app.get("/", response_class=HTMLResponse)
def index(request: Request):
    return templates.TemplateResponse("login.html", {"request": request})

```

شکل 4- نمایش فایل‌های template با کمک Jinja2Templates

با کلیک بر روی دکمه Login کاربر به صفحه احراز هویت گیتهاب وارد می‌شود که به شکل زیر می‌باشد:



شکل 5- صفحه احراز هویت گیتهاب

با Authorize کردن کاربر، کد ایجاد شده در سرور و همچنین در مرورگر نمایش داده می‌شود و کاربر به آدرس درج شده در callback URL ارجاع داده می‌شود. که شکل زیر آن را نشان می‌دهد.



شکل 6- نمایش کد ایجاد شده در مرورگر

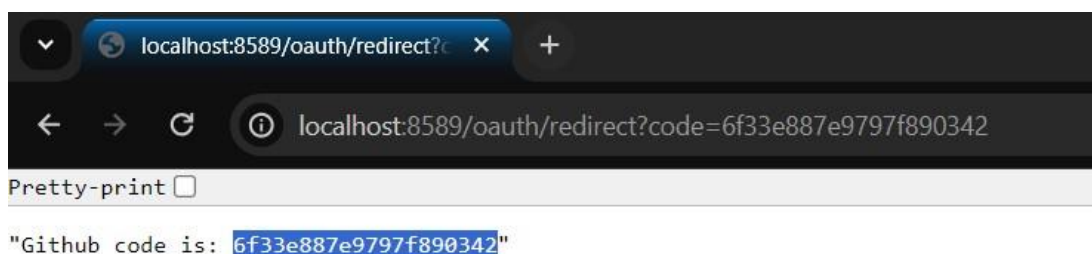


شکل 7- نمایش کد ایجاد شده در ترمینال سرور

## 4. درخواست Access\_token و دریافت اطلاعات کاربر

با دریافت مقدار code و همچنین مقادیر client\_id و client\_secret می‌توانیم از سرویس GitHub درخواست Access token مربوطه را انجام دهیم. در این بخش این درخواست را به صورت دستی و با استفاده از ابزار postman انجام می‌دهیم. شکل زیر تصاویر مربوط به این فرآیند را نمایش می‌دهد

(لازم به ذکر است که کد ایجاد شده در بخش قبلی expire شده و کد جدیدی ایجاد و استفاده شده)



شکل 8- کد جدید ایجاد شده

```
INFO: 127.0.0.1:14805 - "GET /oauth/redirect?code=6f33e887e9797f890342 HTTP/1.1" 200 OK
Github code is: 6f33e887e9797f890342
INFO: 127.0.0.1:14805 - "GET /oauth/redirect?code=6f33e887e9797f890342 HTTP/1.1" 200 OK
```

شکل 9- کد جدید ایجاد شده

حال از postman برای فرستادن درخواست استفاده می‌کنیم

The screenshot shows the Postman interface for a POST request to `https://github.com/login/oauth/access_token?client_id=Ov23liCSix7NQkGmr8uy&client_secret=004642d5183c6d26499f3c09ae0e92009106edac&code=6f33e887e9797f890342`. The request body is a JSON object with the following parameters:

Key	Value	Description
<input checked="" type="checkbox"/> client_id	Ov23liCSix7NQkGmr8uy	
<input checked="" type="checkbox"/> client_secret	004642d5183c6d26499f3c09ae0e92009106edac	
<input checked="" type="checkbox"/> code	6f33e887e9797f890342	

The response body shows the following JSON object:

```
{  "access_token": "gho_BgFBCft6H9xkyD1fJhZiv8JhM0Y1zm0VCkxA",  "scope": "read:user"}
```

شکل 10- دریافت مقدار access\_token با استفاده از code و client\_id و client\_secret با استفاده از postman

## 5. استفاده از API های Github برای دریافت اطلاعات کاربر

با داشتن Access\_token میتوانیم از سرویس API های GitHub برای دریافت اطلاعات کاربر استفاده کنیم و با فرستادن یک HTTP Request و قرار دادن Access\_token در قسمت مربوط به Authorization میتوانیم اطلاعات مورد نظر خود را دریافت کنیم. شکل زیر فرآیند و نتیجه انجام این کار در محیط Postman را نمایش می‌دهد.

The screenshot shows the Postman interface for a GET request to `https://api.github.com/user`. The request is configured with the following headers:

Key	Value	Description
Authorization	Bearer gho_BgFbCft6H9xkyDifJhZrv8JhMOYIzm0VCkxA	
Postman-Token	<calculated when request is sent>	
Host	<calculated when request is sent>	
User-Agent	PostmanRuntime/7.36.0	
Accept	/*/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	

The response is shown in the Body tab, formatted as JSON:

```
1 {
2   "login": "alimrn001",
3   "id": 100947783,
4   "node_id": "U_kgD0BgRXRw",
5   "avatar_url": "https://avatars.githubusercontent.com/u/100947783?v=4",
6   "gravatar_id": "",
7   "url": "https://api.github.com/users/alimrn001",
8   "html_url": "https://github.com/alimrn001",
9   "followers_url": "https://api.github.com/users/alimrn001/followers",
10  "following_url": "https://api.github.com/users/alimrn001/following{/other_user}",
11  "gists_url": "https://api.github.com/users/alimrn001/gists{/gist_id}"
```

شکل 11- مشاهده اطلاعات کاربر در محیط Postman با استفاده از API گیت‌هاب

اطلاعات دقیق کاربر نیز در شکل زیر قابل مشاهده می‌باشد.



```

{
  "login": "alimrn001",
  "id": 100947783,
  "node_id": "U_kgDOBgRXRw",
  "avatar_url": "https://avatars.githubusercontent.com/u/100947783?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/alimrn001",
  "html_url": "https://github.com/alimrn001",
  "followers_url": "https://api.github.com/users/alimrn001/followers",
  "following_url": "https://api.github.com/users/alimrn001/following{/other_user}",
  "gists_url": "https://api.github.com/users/alimrn001/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/alimrn001/starred{/owner}/{/repo}",
  "subscriptions_url": "https://api.github.com/users/alimrn001/subscriptions",
  "organizations_url": "https://api.github.com/users/alimrn001/orgs",
  "repos_url": "https://api.github.com/users/alimrn001/repos",
  "events_url": "https://api.github.com/users/alimrn001/events{/privacy}",
  "received_events_url": "https://api.github.com/users/alimrn001/received_events",
  "type": "User",
  "site_admin": false,
  "name": "Ali Mehrani",
  "company": "University of Tehran",
  "blog": "",
  "location": "Tehran, Iran",
  "email": null,
  "hireable": null,
  "bio": "Computer Engineering student at the University of Tehran |\r\nInterested in",
  "twitter_username": null,
  "public_repos": 23,
  "public_gists": 0,
  "followers": 26,
  "following": 34,
  "created_at": "2022-03-04T14:02:56Z",
  "updated_at": "2024-05-24T11:05:19Z"
}

```

شکل 12- اطلاعات دریافت شده کاربر با API گیتهاب

## 6. خودکارسازی فرآیند دریافت Access\_token و اطلاعات کاربر

در ادامه با پیاده‌سازی کدهای لازم در فایل `server.py`، فرآیندهای مراحل بالا که شامل دریافت `Access_token` و استفاده از آن برای دریافت اطلاعات کاربر می‌باشد را خودکارسازی می‌کنیم و در نهایت پس از `Authorize` کردن کاربر، اطلاعات او را در یک صفحه تحت وب پس از `redirect` شده، نمایش می‌دهیم.

شکل‌های زیر کدهای نوشته‌شده را نمایش می‌دهند:

ابتدا در تابع `oauth_redirect` مقدار `github code` را دریافت کرده و آن را به تابعی با نام `get_auth_token` می‌فرستیم. وظیفه این تابع دریافت `access_token` با داشتن `github code` و همچنین مقادیر `client_key` و `secret_key` که در برنامه مشخص شده‌اند، می‌باشد. سپس با دریافت `access_token` در تابع `oauth_redirect` آن را به تابعی با نام `get_user_data_with_token` می‌دهیم که وظیفه این تابع، همانطور که از نامش پیداست، دریافت اطلاعات کاربر با دریافت `access_token` می‌باشد. در نهایت می‌توانیم اطلاعات دریافت شده را در قالب یک `raw JSON` برگردانیم یا در قالب یک `template` آن را در یک صفحه `html` نمایش دهیم که در این‌جا مورد دوم صورت می‌گیرد. شکل زیر تابع `oauth_redirect` را نمایش می‌دهد:

```
61 @app.get("/oauth/redirect")
62 def oauth_redirect(request: Request, code: str): # add request as parameter to be able to use context for template engine
63     print(f'Github code is: {code}')
64
65     try:
66         access_token = get_auth_token(code)
67         user_data = get_user_data_with_token(access_token)
68         # return user_data # uncomment if you want to display user data in raw json
69
70         return templates.TemplateResponse("user_data.html", {"request": request, "user_data": user_data}) # uncomment if y
71
72     except Exception as e:
73         return f'Server Error: {e}', 500
74
```

شکل 13- تابع `oauth_redirect`

در ادامه در تابع `get_auth_token` با فرستادن یک `HTTP Request` با استفاده از پکیج `Requests` و ارسال مقادیر `code` و `client_id` و `client_secret` در `payload` و ارسال درخواست، مقدار `access_token` را دریافت کرده و برمی‌گردانیم. همچنین در صورت بروز هرگونه مشکل، با استفاده از `raise_for_status`، `exception` های لازم را `throw` می‌کنیم. لازم به ذکر است که `reponse` داده شده پس از فرستادن درخواست `HTTP`، مقدار `token` نیست بلکه خروجی ساختاری مانند یک `encoded URL` دارد که `token` قسمتی از آن است و با استفاده از تابع `get_token_from_encoded_url`، آن را استخراج می‌کنیم که تابع ذکر شده از پکیج `parse_qs` استفاده می‌کند که برای `parse` کردن یک `URL` به `component` های مختلف استفاده می‌شود. شکل زیر دو تابع ذکر شده را نمایش می‌دهد.

```

18 def get_token_from_encoded_url(response):
19     if not response.text:
20         raise Exception("Server Error")
21
22     response_txt_data = parse_qs(response.text)
23     access_token = response_txt_data.get('access_token', [None])[0]
24     if access_token:
25         return access_token
26     else:
27         raise Exception("Server Error")

```

شکل 14- تابع get\_token\_from\_encoded\_url

```

30 def get_auth_token(github_code):
31     try:
32         token_request_payload = {
33             'code': github_code,
34             'client_id': CLIENT_ID,
35             'client_secret': CLIENT_SECRET
36         }
37
38         auth_response = requests.post(
39             ACCESS_TOKEN_RETRIEVAL_URL, data=token_request_payload)
40
41         auth_response.raise_for_status()
42         access_token = get_token_from_encoded_url(auth_response)
43         print(f'access_token: {access_token}')
44
45         return access_token
46
47     except Exception as e:
48         raise Exception(f'Error: Failed to retrieve access token: {e}')
49

```

شکل 15- تابع get\_auth\_token

```

USER_DATA_RETRIEVAL_URL = "https://api.github.com/user"
ACCESS_TOKEN_RETRIEVAL_URL = "https://github.com/login/oauth/access_token/"
CLIENT_SECRET = "004642d5183c6d26499f3c09ae0e92009106edac"
CLIENT_ID = "Ov23liCSIx7NQkGmr8uy"

```

شکل 16- اطلاعات مربوط به OAuth App

در نهایت در تابع `get_user_data_with_token`، مقدار `token` را دریافت کرده و یک `http request` ایجاد و ارسال میکنیم که در قسمت `header` آن و در قسمت `authorization` از `header` مقدار `token` را قرار می‌دهیم و `request` را ارسال می‌کنیم. در صورت بروز هرگونه خطا، `exception` های لازمه را `throw` میکنیم و در غیر اینصورت، اطلاعات کاربر را دریافت و برمی‌گردانیم.

شکل زیر این تابع را نمایش می‌دهد:

```
51 def get_user_data_with_token(token):
52     auth_header = {
53         'Authorization': f'Bearer {token}'
54     }
55     data_response = requests.get(USER_DATA_RETRIEVAL_URL, headers=auth_header)
56     data_response.raise_for_status()
57
58     return data_response.json()
```

شکل 17- تابع `get_user_data_with_token`

در نهایت با دریافت اطلاعات کاربر، `template` مربوط به صفحه اطلاعات کاربر را ایجاد کرده و آن اطلاعات را در قالب `context` به `template` ارسال میکنیم و صفحه را ایجاد میکنیم. نام این صفحه `user_data.html` نام دارد.

```
return templates.TemplateResponse("user_data.html", {"request": request, "user_data": user_data})
```

شکل 18- ارسال اطلاعات در قالب `context` به `template` و ایجاد `user_data.html`

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>user: {{ user_data.login }}</title>
7   <style> ...
36  </style>
37 </head>
38 <body>
39   <div class="main-container d-flex">
40     <div class="profile-section-1">
41       
46       <h3 class="mt-1">Username: {{ user_data.login }}</h3>
47       <h3 class="mt-1">Name: {{ user_data.name }}</h3>
48       <a href="{{ user_data.html_url }}" class="mt-1">Visit Profile</a>
49     </div>
50     <div class="profile-section-2">
51       <p class="mt-1">{{ user_data.bio }}</p>
52       <p class="mt-1">Company: {{user_data.company}}</p>
53       <p class="mt-1">Location: {{user_data.location}}</p>
54     </div>
55   </div>
56 </body>
57 </html>

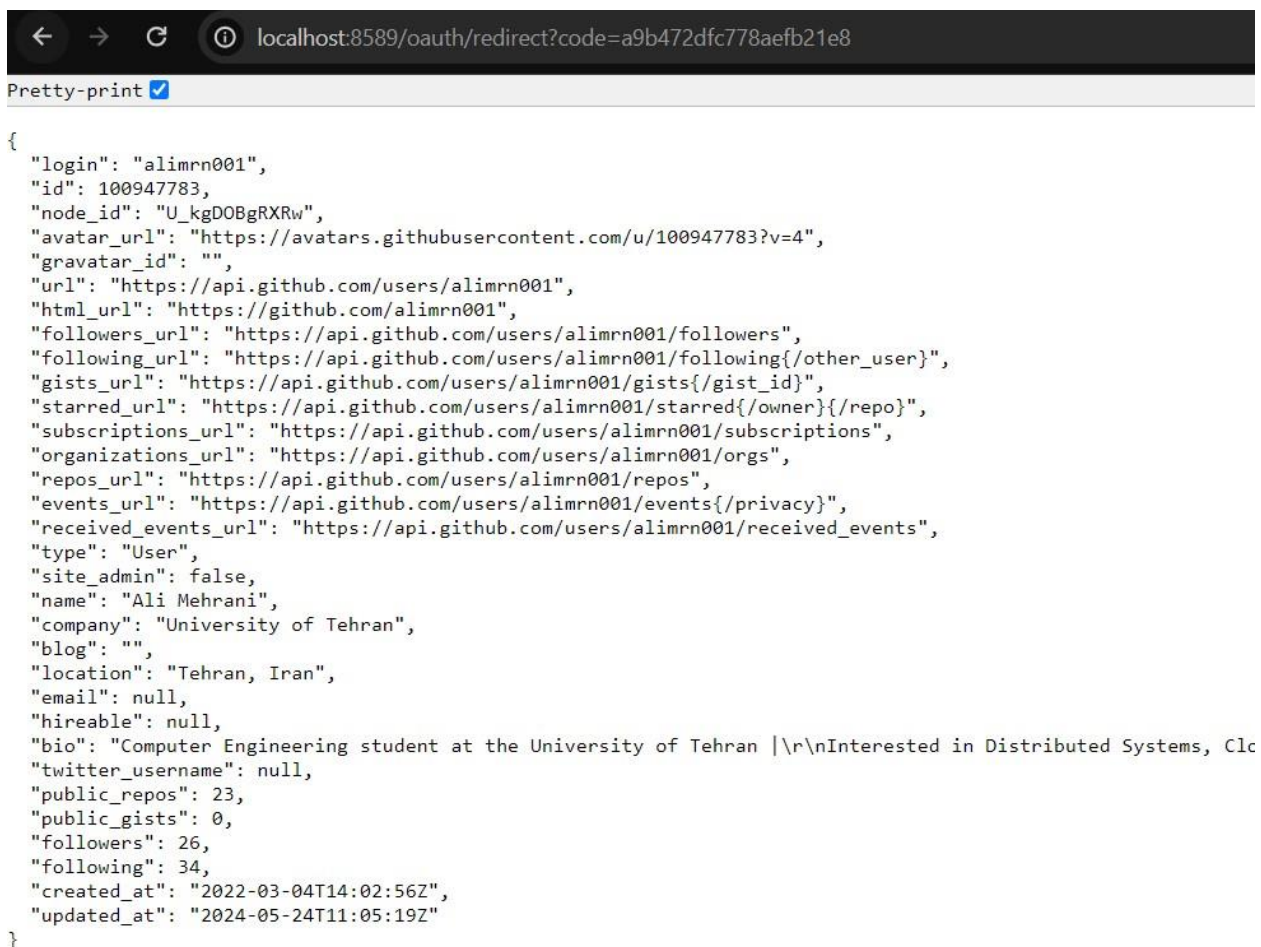
```

شکل 19- فایل user\_data.html

در نهایت پس از Authorize کردن کاربر، او به صفحه ای هدایت که اطلاعات او در آن نمایش داده شده. شکل زیر آن را نشان می‌دهد:



شکل 20- نمایش اطلاعات کاربر در قالب HTML



شکل 21- نمایش اطلاعات کاربر در قالب raw JSON

## پاسخ سوالات

### 1.

مزایای آن عبارتند از:

در آن می‌توانیم وظیفه تامین سرویس‌های Authentication و Authorization را بر عهده سیستم‌های Third party خارجی مورد اطمینان قرار دهیم.

می‌توانیم امنیت سیستم را با استفاده از Proof Key for Code Exchange یا PKCE افزایش دهیم که با کمک آن می‌توان از حملات CSRF یا حملات Authorization code injection جلوگیری کرد.

این روش در مقایسه با Grant Type های دیگر در مواردی مانند web app ها یا mobile app ها کمی بهتر است چون یک مرحله اضافه برای دریافت access\_token با code را هم دارد.

### 2.

استفاده از روش کلاینت برای این نوع برنامه‌ها مناسب نیست چون که این نرم‌افزارها نمی‌توانند client credential را مخفی نگه دارند، یک Attacker می‌تواند با برداشتن client credential و جا زدن برنامه خودش به جای آن client به منابع سیستم دسترسی پیدا کند. استفاده از این روش بیشتر برای ارتباط سرورهای مختلف با یکدیگر یا سرویس‌ها با یکدیگر (در معماری‌های مبتنی بر سرویس یا SOA) مناسب است.

### 3. (منبع استفاده شده برای پاسخ: <https://github.blog/2021-04-05-behind-githubs-new-authentication-token-formats>)

Access token نباید قابل حدس یا دست‌کاری شدن باشد که از ویژگی‌های آن در پروتکل است همچنین باید در شبکه امن مانند https ارسال شود تا قابل خواندن توسط attacker نباشد همچنین ساختار access token های github به شکل زیر می‌باشد:

ابتدا سه کاراکتر می‌آیند که تعیین‌کننده نوع token می‌باشد و برای OAuth access token به فرم gho می‌باشد (یا مثلاً برای refresh token به فرم ghr) در ادامه یک \_ underline می‌آید که یک کاراکتر base64 نیست و اگر بر روی یک رشته که شامل آن underline است double-click کنید، کل رشته select می‌شود. در ادامه یک checksum به طول 32 بیت در 6 کاراکتر آخر token می‌آید و در کل از حروف a تا z (lowercase و uppercase) و اعداد 0 تا 9 استفاده می‌شود که باعث می‌شود امکان تولید token تکراری کم باشد و قابل decode نیز نیست. رشته access token در اصل encoded است و دارای آنتروپی بالا و randomness است و دارای complexity مناسب است و فرمت base64 دارد و قابل حدس نیست.

#### .4

ابتدا باید به این اشاره کرد که چون از HTTPS برای سرور استفاده نشده هنگام فرستادن درخواست‌ها مقدار token کاربر قابل مشاهده است و Attacker می‌تواند آن را استفاده کند و خودش را به جای کاربر جا بزند.

همچنین برای token تاریخ انقضا یا expire و روش refresh شدن آن نیز مشخص نشده و فرد دیگری می‌تواند به session کاربر دسترسی پیدا بکند که برای رفع این مشکل باید زمان expire شدن token و همچنین نحوه refresh آن را مشخص کنیم.