



TEKNOLOJİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ
YAZILIM KALİTE GÜVENCE TEMELLERİ

Github API ve UI Testi

Enes Elyesa Çiçek 170220012

Mert Baytaş 138320067

Ahmet KURT 170421022

Ubeydullah AK 170421852

1. Giriş: Yazılım Test Otomasyonu ve Proje Amacı	2
2. Uygulama ve API Analizi: Fonksiyonlar ve Test Odakları	3
Test Edilmesi Gereken Alanlar ve Seçim Gerekçeleri:	4
Senaryolar Örneği:	4
3. Api Testleri (Otomasyon):	5
Kullanılan Araçlar:	5
a. Public API Testleri:	5
i. Search Api Testi	5
ii. User Api Testi	6
iii. Takipçileri ve Takipleri listeleme apisi	7
iv. Repo listesi apisi	8
v. Repo zipball olarak indirme	9
b. Private API Testleri:	10
4. Selenium ile UI Testleri:	17
a. Giriş: Selenium ve GitHub UI Testi	17
b. Test Edilen Kullanıcı Senaryoları ve Sayfa Akışı	17
i. GitHub Ana Sayfası (https://github.com)	17
ii. Giriş Sayfası (https://github.com/login)	17
iii. Yeni Repo Oluşturma (https://github.com/new)	18
iv. Repo Sayfasına Gitme ve Dosya Yükleme	18
v. Arama ve İlgili Repo Sayfasına Geçiş	18
vi. Dosya İndirme (Raw Buton)	19
vii. Oturum Kapatma	19
c. Başarılı ve Başarısız Assertion Örnekleri	19
5. Selenium Kodunun Aşama Aşama Anlatımı ve Proje Dosya Yapısı	20
a. Selenium Kodunun Aşama Aşama Ne Yaptığı	20
i. Ortam Tanımlamaları ve Tarayıcı Ayarları	20
b. Proje Dosya Yapısı ve İçerikleri	22
requirements.txt	22
.env	22
6. Proje Genel Sonucu ve Değerlendirme	23
7. Sonuç	24
8. Kaynakça	24

1. Giriş: Yazılım Test Otomasyonu ve Proje Amacı

Yazılım Kalite Güvencesi, yazılım geliştirme sürecinde ortaya çıkan hataları en aza indirerek kullanıcıya sorunsuz bir deneyim sunmayı hedefler. Yazılım test otomasyonu ise bu süreci daha hızlı, güvenilir ve tekrarlanabilir hale getiren otomatik test yöntemlerini kapsar. Bu projede hem API test otomasyonu hem de Selenium tabanlı UI (kullanıcı arayüz) testleri gerçekleştirilmiştir.

API testleri, bir uygulamanın istemciden sunucuya veri alışverişi sürecinde beklenen dönüşlerin alınıp alınmadığını kontrol eder. Bu testler genellikle Postman, Curl veya programatik HTTP istekleriyle yapılır. Biz bu projede GitHub'un REST API'lerini kullanarak GET istekleri ile repo arama, kullanıcı bilgisi sorgulama, takipçileri listeleme, zip dosyası indirme gibi fonksiyonlara odaklandık.

Selenium ise web tabanlı uygulamaların kullanıcı arayüzlerini simüle ederek test etmemizi sağlar. Bu testler, bir kullanıcının tarayıcıda yaptığı tüm işlemlerin otomatik olarak gerçekleştirilmesini sağlar. Bu projede GitHub girişi, yeni repo oluşturma, dosya yükleme, repo arama ve indirme gibi temel akışlar Selenium ile test edildi.

Postman, API testlerini manuel veya otomatik olarak yapabilmek için geliştirilmiş yaygın bir test aracıdır. Otomatik testlerde ise scripting desteği sayesinde tekrarlanabilir senaryolar oluşturulabilir.

Bu proje, yazılım testi öğrencileri için hem backend (API) hem de frontend (UI) seviyesinde kalite kontrol mekanizmalarını tanımak için gerçek hayat senaryoları sunmaktadır. Uygulama olarak GitHub'ı seçmemizin nedeni, güçlü ve belgelenmiş API desteği, dinamik ve etkileşimli bir UI yapısına sahip olması ve öğrendiklerimizi doğrudan gerçek bir platform üzerinde test etme fırsatı sunmasıdır.

2. Uygulama ve API Analizi: Fonksiyonlar ve Test Odakları

GitHub, kullanıcıların depo (repo) oluşturmalarını, kod yüklemelerini, takip etmesini ve aramasını sağlayan bir versiyon kontrol platformudur. En çok kullanılan sayfalar ve API fonksiyonları şunlardır:

Sayfa/Endpoint	Fonksiyon	Test Durumu
https://github.com/login	Kullanıcı girişı	UI: Pozitif/Negatif
GET /search/repositories	Anahtar kelime ile repo arama	API: Pozitif/Negatif
GET /users/{username}	Kullanıcı bilgisi getirme	API: Pozitif/Negatif
GET /users/{username}/repos	Kullanıcıya ait repoları listeleme	API: Pozitif/Negatif
GET /repos/{owner}/{repo}/	Repo'yu zip şeklinde indirme	API: Pozitif/Negatif
Web'de "New Repository" sayfası	Yeni repo oluşturma	UI: Pozitif/Negatif
Repo sayfası	Dosya yükleme	UI: Pozitif/Negatif

Test Edilmesi Gereken Alanlar ve Seçim Gerekçeleri:

- **Giriş Sayfası:** Yanlış şifre, boş alan gibi senaryolar UI testleriyle kontrol edildi.
- **Repo Oluşturma:** Selenium ile repo oluşturma ekranına gidilerek form doldurma ve buton etkileşimi test edildi.
- **API Arama:** Anahtar kelime girildiğinde dönen repo sayısı, sıralama ve içerik kontrol edildi.
- **Hatalı API Kullanımı:** Geçersiz username ile 404 yanıtı beklenerek test senaryosuna dahil edildi.
- **Repo Indirme:** API üzerinden zipball endpoint ile çekilen repo'nun indirilebilirliği test edildi.

Senaryolar Örneği:

Senaryo Adı	Test Türü	Giriş Verisi	Beklenen Çıktı
Geçerli giriş	UI	Doğru username/password	Ana sayfaya yönlendirme
Geçersiz giriş	UI	Yanlış password	Hata bildirimi
Repo arama	API	q=todolist	Status 200 + repo listesi

Kullanıcı bilgisi	API	username=elysaccek	Status 200 + JSON içerikte login alanı
Repo indirme	API	repo=Base58-Library	Status 200 + binary dosya indirimi
Dosya yükleme	UI	test.html dosyası	"Commit changes" butonuyla yükleme
Geçersiz username sorgusu	API	username=asdadasd123	Status 404

Bu analiz ve senaryolar, hem uygulamanın kritik fonksiyonlarını hem de çeşitli hata durumlarını kapsayacak şekilde testlerin planlandığını ve uygulandığını göstermektedir.

3. Api Testleri (Otomasyon):

Bu bölümde, GitHub REST API'si kullanılarak temel HTTP istekleri üzerinden çeşitli testler gerçekleştirilmiştir. Testlerde olumlu ve olumsuz senaryolar dikkate alınmış; sistemin doğru ve hatalı girişlere karşı verdiği yanıtlar analiz edilmiştir.

Kullanılan Araçlar:

- Postman (manuel testler ve gözlemlleme)
- Curl (dosya indirme testi)
- REST API (GitHub Public ve Private API)

a. Public API Testleri:

i. Search Api Testi

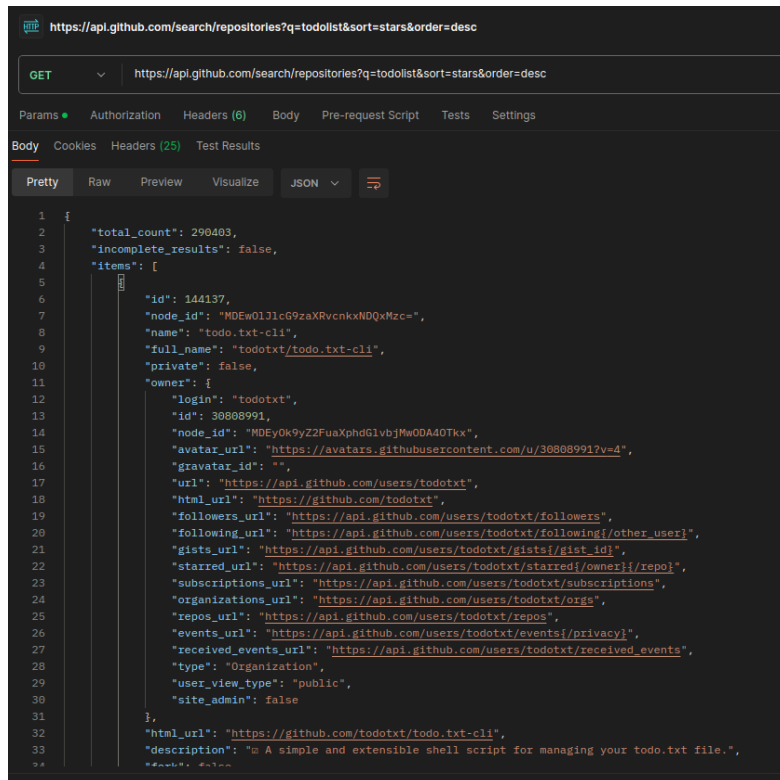
Api URL: <https://api.github.com/search/repositories?>

Parametreler:

Parametre	Örnek	Açıklama
q	todolist	Aranan terim
sort	stars	Arama sonucu neye göre sıralansın?
order	desc	Sıralama yönü ne olsun? desc yüksekten => düşüğe asc düşükten => yükseğe

Postman Sorgusu:

GET <https://api.github.com/search/repositories?q=todolist&sort=stars&order=desc>



ii. User Api Testi

Api URL: <https://api.github.com/users/{username}>

Postman Sorgusu: GET <https://api.github.com/users/elysaccek>

```
https://api.github.com/users/elysaccek

GET https://api.github.com/users/elysaccek

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (26) Test Results

Pretty Raw Preview Visualize JSON

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

{
  "login": "elysaccek",
  "id": 71216237,
  "node_id": "MDQ6VXNlcjcxMjE2MjM3",
  "avatar_url": "https://avatars.githubusercontent.com/u/71216237?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/elysaccek",
  "html_url": "https://github.com/elysaccek",
  "followers_url": "https://api.github.com/users/elysaccek/followers",
  "following_url": "https://api.github.com/users/elysaccek/following{/other_user}",
  "gists_url": "https://api.github.com/users/elysaccek/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/elysaccek/starred{/owner}/{repo}",
  "subscriptions_url": "https://api.github.com/users/elysaccek/subscriptions",
  "organizations_url": "https://api.github.com/users/elysaccek/orgs",
  "repos_url": "https://api.github.com/users/elysaccek/repos",
  "events_url": "https://api.github.com/users/elysaccek/events{/privacy}",
  "received_events_url": "https://api.github.com/users/elysaccek/received_events",
  "type": "User",
  "user_view_type": "public",
  "site_admin": false,
  "name": "Elyesa Çiçek",
  "company": null,
  "blog": "",
  "location": null,
  "email": null,
  "hireable": null,
  "bio": "According To My Preference",
  "twitter_username": null,
  "public_repos": 9,
  "public_gists": 0,
  "followers": 9,
  "following": 8,
  "created_at": "2020-09-13T11:58:29Z",
  "updated_at": "2020-09-13T11:58:29Z"
}
```

iii. Takipçileri ve Takipleri listeleme apisi

Api URL:

<https://api.github.com/users/{username}/followers>

<https://api.github.com/users/{username}/following>

Postman Sorgusu:

GET <https://api.github.com/users/elysaccek/followers>

```
https://api.github.com/users/elysaccek/followers

GET https://api.github.com/users/elysaccek/followers

Params Authorization Headers (6) Body Pre-request Script Tests Settings

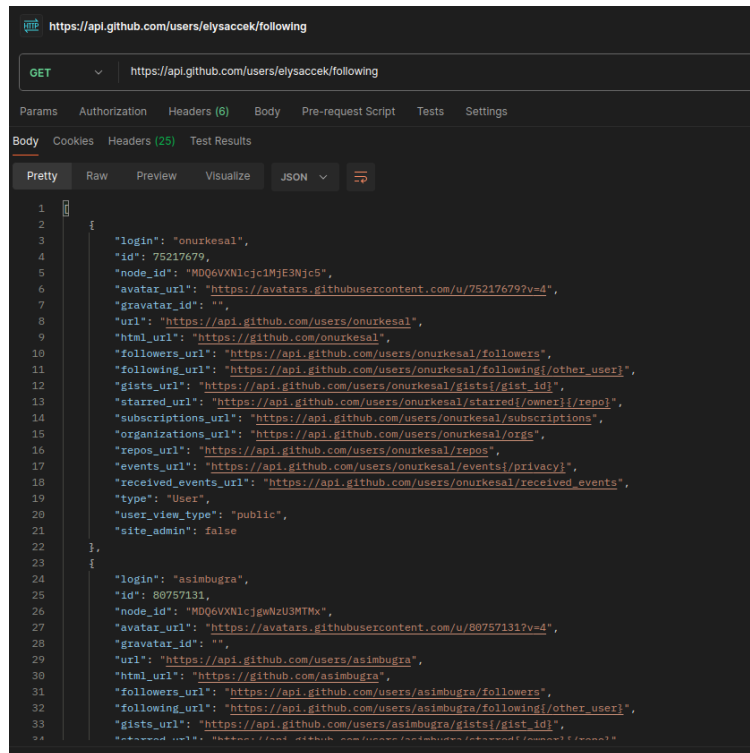
Body Cookies Headers (25) Test Results

Pretty Raw Preview Visualize JSON

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

[
  {
    "login": "bludnic",
    "id": 25831507,
    "node_id": "MDQ6VXNlcjI1ODMxNTA3",
    "avatar_url": "https://avatars.githubusercontent.com/u/25831507?v=4",
    "gravatar_id": "",
    "url": "https://api.github.com/users/bludnic",
    "html_url": "https://github.com/bludnic",
    "followers_url": "https://api.github.com/users/bludnic/followers",
    "following_url": "https://api.github.com/users/bludnic/following{/other_user}",
    "gists_url": "https://api.github.com/users/bludnic/gists{/gist_id}",
    "starred_url": "https://api.github.com/users/bludnic/starred{/owner}/{repo}",
    "subscriptions_url": "https://api.github.com/users/bludnic/subscriptions",
    "organizations_url": "https://api.github.com/users/bludnic/orgs",
    "repos_url": "https://api.github.com/users/bludnic/repos",
    "events_url": "https://api.github.com/users/bludnic/events{/privacy}",
    "received_events_url": "https://api.github.com/users/bludnic/received_events",
    "type": "User",
    "user_view_type": "public",
    "site_admin": false
  },
  {
    "login": "onurkesal",
    "id": 75217679,
    "node_id": "MDQ6VXNlcjMjE3MjE3",
    "avatar_url": "https://avatars.githubusercontent.com/u/75217679?v=4",
    "gravatar_id": "",
    "url": "https://api.github.com/users/onurkesal",
    "html_url": "https://github.com/onurkesal",
    "followers_url": "https://api.github.com/users/onurkesal/followers",
    "following_url": "https://api.github.com/users/onurkesal/following{/other_user}",
    "gists_url": "https://api.github.com/users/onurkesal/gists{/gist_id}",
    "starred_url": "https://api.github.com/users/onurkesal/starred{/owner}/{repo}",
    "subscriptions_url": "https://api.github.com/users/onurkesal/subscriptions",
    "organizations_url": "https://api.github.com/users/onurkesal/orgs",
    "repos_url": "https://api.github.com/users/onurkesal/repos",
    "events_url": "https://api.github.com/users/onurkesal/events{/privacy}",
    "received_events_url": "https://api.github.com/users/onurkesal/received_events",
    "type": "User",
    "user_view_type": "public",
    "site_admin": false
  }
]
```

GET <https://api.github.com/users/elysaccek/following>

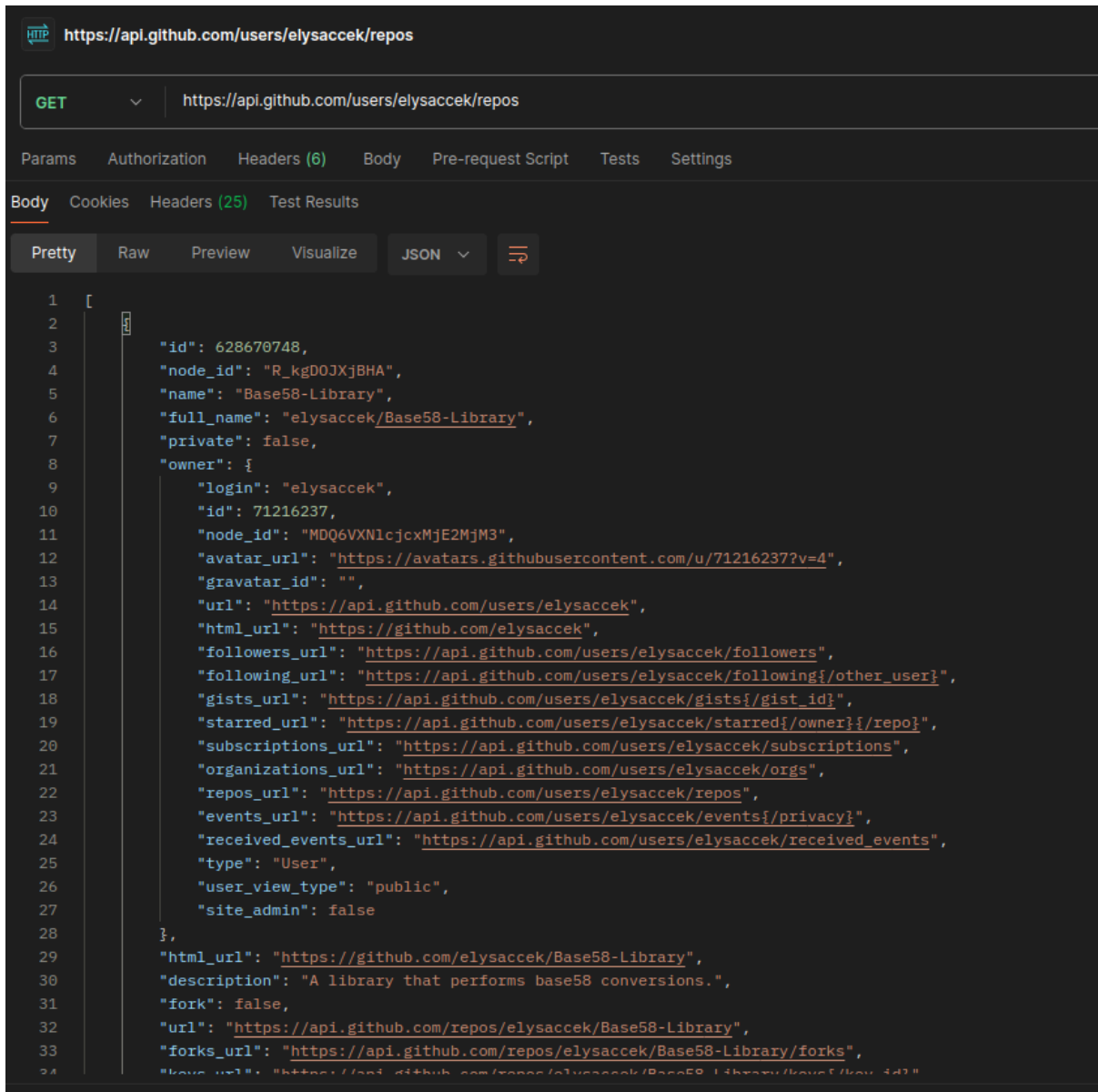


```
1  {
2    "login": "onurkesal",
3    "id": 75217679,
4    "node_id": "MDQ6VXNlcjc1MjE3Njc5",
5    "avatar_url": "https://avatars.githubusercontent.com/u/75217679?v=4",
6    "gravatar_id": "",
7    "url": "https://api.github.com/users/onurkesal",
8    "html_url": "https://github.com/onurkesal",
9    "followers_url": "https://api.github.com/users/onurkesal/followers",
10   "following_url": "https://api.github.com/users/onurkesal/following{/other_user}",
11   "gists_url": "https://api.github.com/users/onurkesal/gists{/gist_id}",
12   "starred_url": "https://api.github.com/users/onurkesal/starred{/owner}/repo",
13   "subscriptions_url": "https://api.github.com/users/onurkesal/subscriptions",
14   "organizations_url": "https://api.github.com/users/onurkesal/orgs",
15   "repos_url": "https://api.github.com/users/onurkesal/repos",
16   "events_url": "https://api.github.com/users/onurkesal/events{/privacy}",
17   "received_events_url": "https://api.github.com/users/onurkesal/received_events",
18   "type": "User",
19   "user_view_type": "public",
20   "site_admin": false
21 },
22 {
23   "login": "asimbugra",
24   "id": 88757131,
25   "node_id": "MDQ6VXNlcjgwMzU3MTMx",
26   "avatar_url": "https://avatars.githubusercontent.com/u/88757131?v=4",
27   "gravatar_id": "",
28   "url": "https://api.github.com/users/asimbugra",
29   "html_url": "https://github.com/asimbugra",
30   "followers_url": "https://api.github.com/users/asimbugra/followers",
31   "following_url": "https://api.github.com/users/asimbugra/following{/other_user}",
32   "gists_url": "https://api.github.com/users/asimbugra/gists{/gist_id}",
33   "starred_url": "https://api.github.com/users/asimbugra/starred{/owner}/repo",
34   "subscriptions_url": "https://api.github.com/users/asimbugra/subscriptions",
35   "organizations_url": "https://api.github.com/users/asimbugra/orgs",
36   "repos_url": "https://api.github.com/users/asimbugra/repos",
37   "events_url": "https://api.github.com/users/asimbugra/events{/privacy}",
38   "received_events_url": "https://api.github.com/users/asimbugra/received_events",
39   "type": "User",
40   "user_view_type": "public",
41   "site_admin": false
42 }
```

iv. Repo listesi apisi

Api URL: <https://api.github.com/users/{username}/repos>

Postman Sorgusu: GET <https://api.github.com/users/elysaccek/repos>

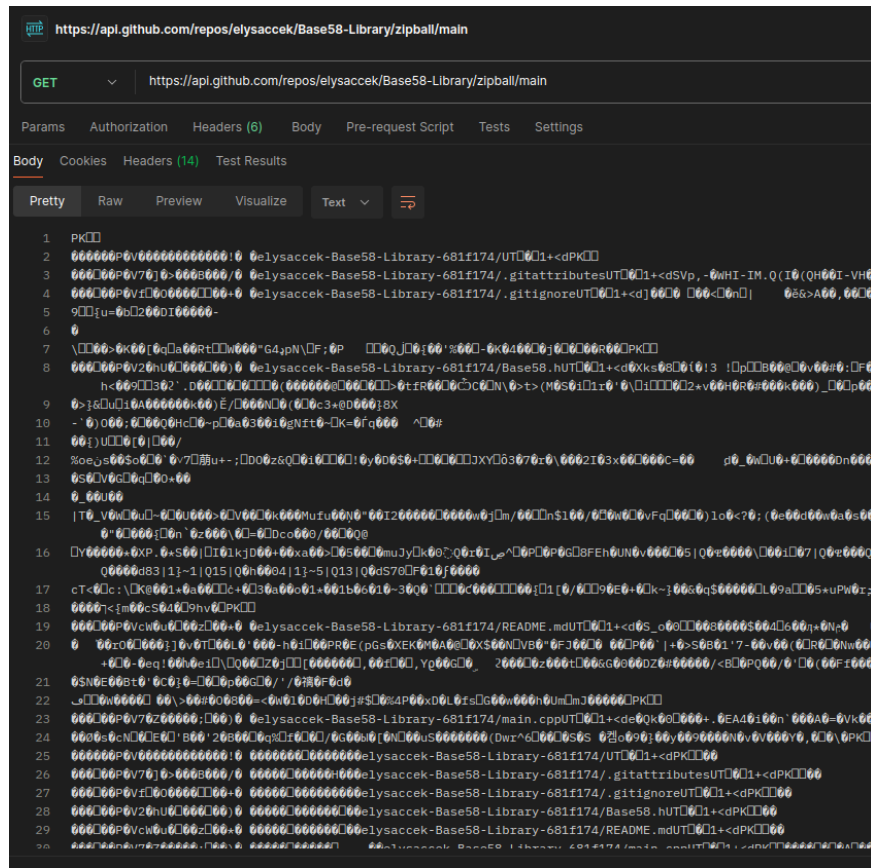


v. Repo zipball olarak indirme

Api URL: <https://api.github.com/repos/{owner}/{repo}/zipball/{ref}>

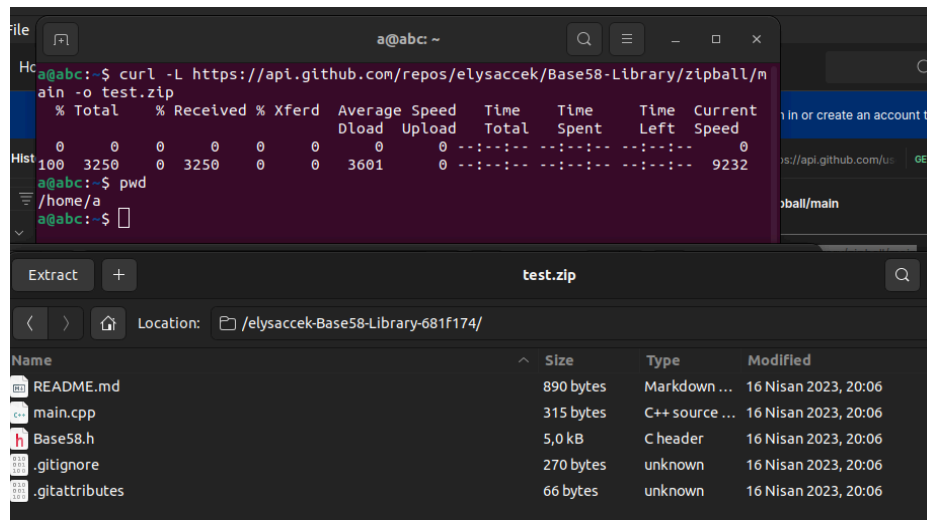
Postman Sorgusu:

GET <https://api.github.com/repos/elysaccek/Base58-Library/zipball/main>



Gelen isteği curl ile zip olarak kaydetmek:

curl -L <https://api.github.com/repos/{owner}/{repo}/zipball/{ref}> -o {output}.zip



b. Private API Testleri:

GitHub'ın REST API'si, kullanıcıların GitHub üzerindeki kaynaklara (repository, followers, stars, issues, vs.) programatik olarak erişmesini sağlar. Ancak bu işlemlerden

bazıları (repo oluşturma, güncelleme, silme gibi) **kimlik doğrulaması** gerektirir. Bunun için GitHub tarafından sağlanan **Personal Access Token (PAT)** kullanılır.

✖ Yetkili API kullanımı nasıl çalışır?

Postman gibi araçlar üzerinden istek atarken:

- **Authorization** sekmesinde “Bearer Token” yöntemi seçilir.
- Token, GitHub > Settings > Developer Settings > Personal Access Tokens sayfasından alınır.
- Bu token sayesinde ilgili kullanıcı adına işlem yapılabilir.

✓ 1. Repository Oluşturma – **POST /user/repos**

🔍 Amaç: Yeni bir GitHub reposu oluşturmak.

🔗 URL

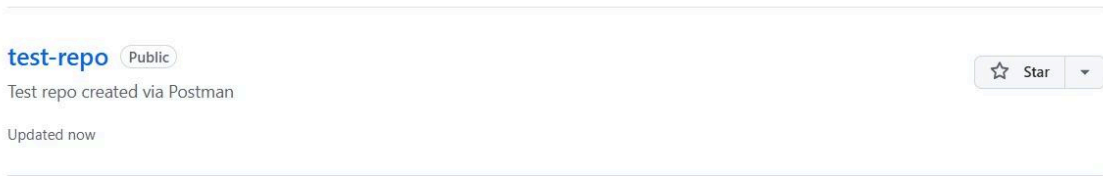
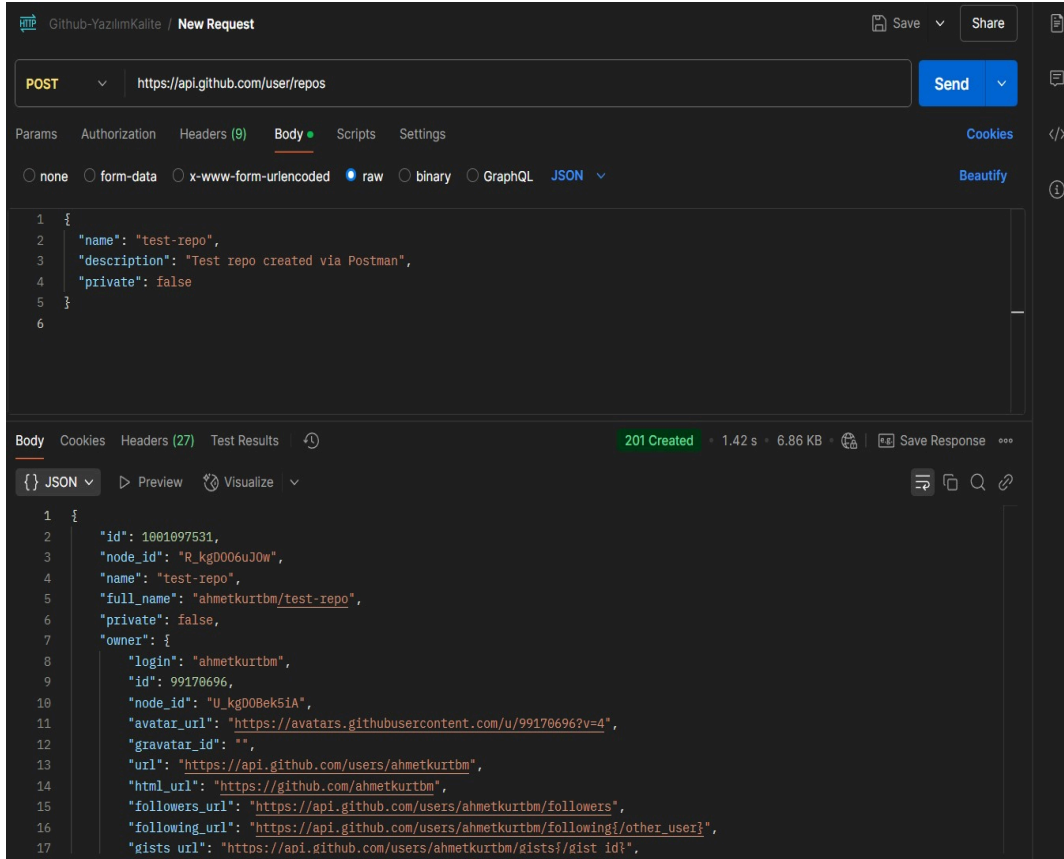
<https://api.github.com/user/repos>

📦 Body

```
{  
  "name": "test-repo",  
  "description": "Test repo created via Postman",  
  "private": false  
}
```

✓ Response

- **201 Created**: Repository başarıyla oluşturuldu.
- **id**, **name**, **owner** bilgileri JSON yanıtında dönüyor.



2. Repository Açıklamasını Güncelleme – PATCH /repos/{owner}/{repo}

Amaç: Mevcut bir repo'nun açıklamasını (description) değiştirmek.

URL

<https://api.github.com/repos/ahmetkurtbm/pythonBot>

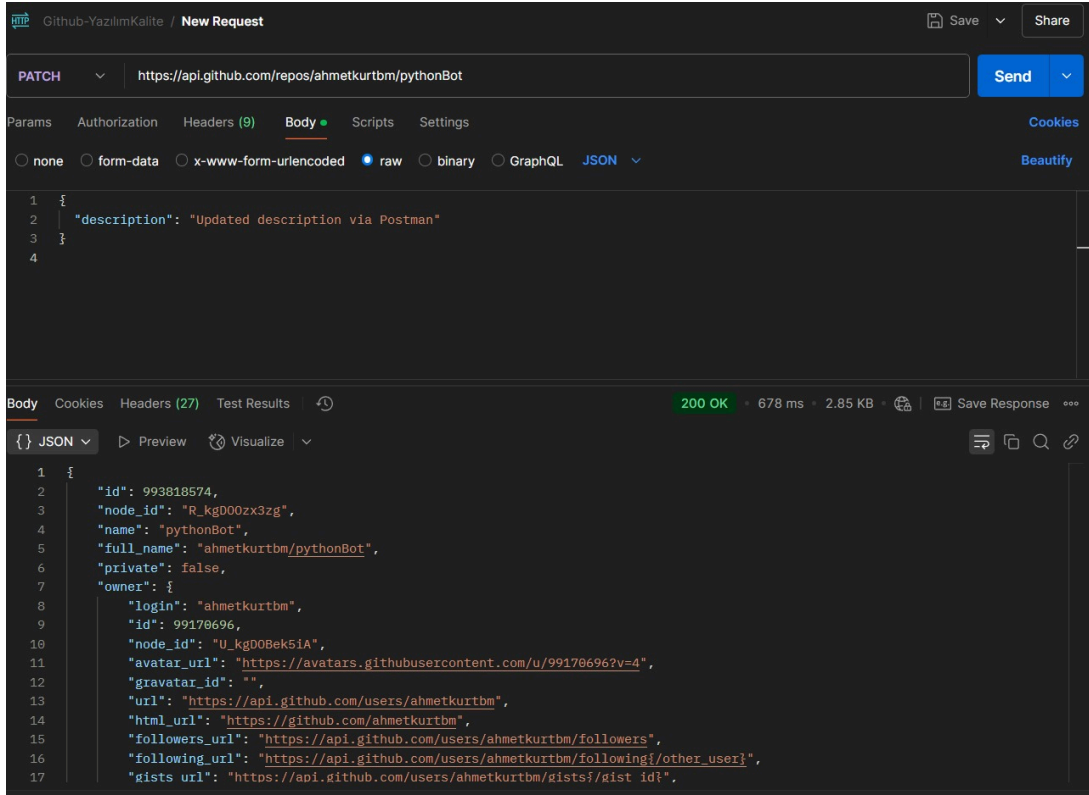
Body

```
{
  "description": "Updated description via Postman"
}
```

✓ Response

- 200 OK: Açıklama başarıyla güncellendi.

- Güncel bilgiler JSON içinde dönüyor.



❌ 3. Geçersiz PUT İsteği – PUT /repos/{owner}/{repo}

🔍 Amaç: Yanlış yöntemle repo üzerinde işlem yapmaya çalışma.

🔗 URL

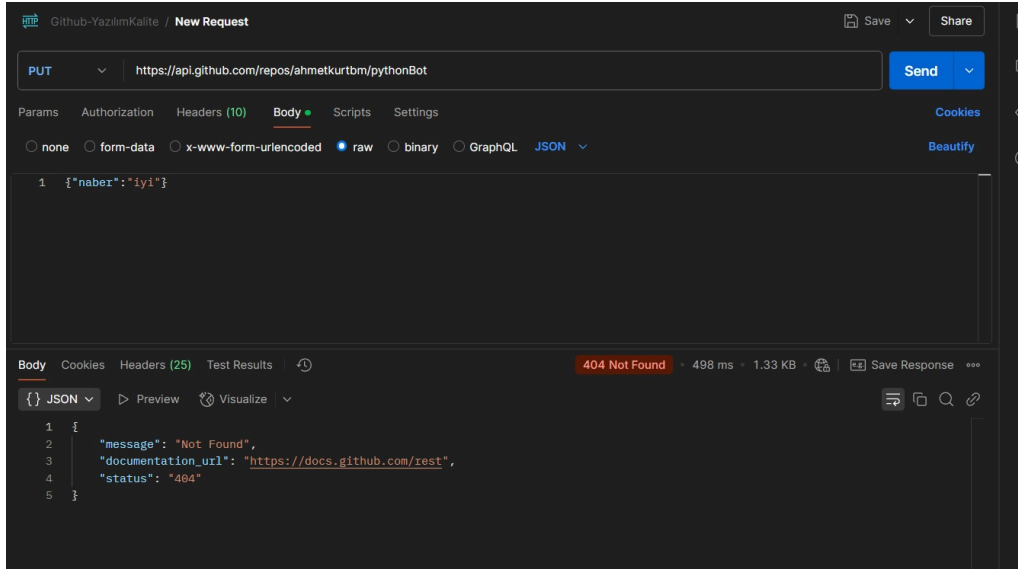
<https://api.github.com/repos/ahmetkurtbm/pythonBot>

📦 Body

`{ "naber": "iyi" }`

❌ Response

- **404 Not Found**: Geçersiz endpoint veya istek gövdesi.



★ 4. Repository’e Yıldız Verme – PUT /user/starred/{owner}/{repo}

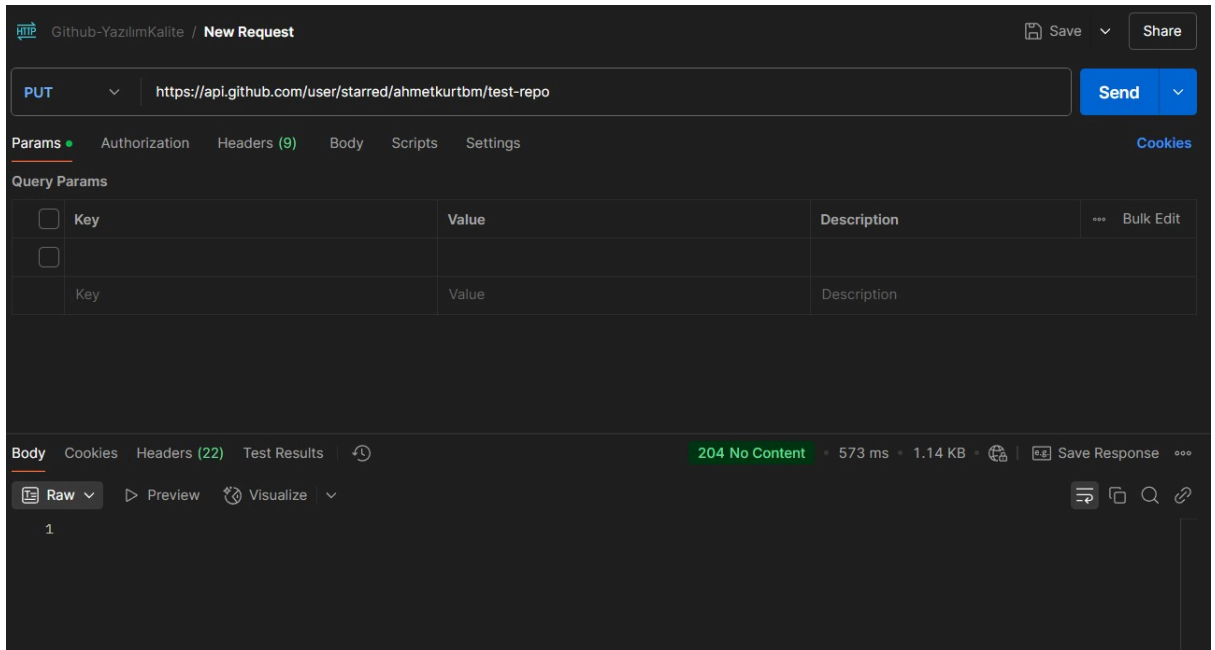
🔍 Amaç: Belirli bir repo’yu kullanıcı adına yıldızlamak (star).

🔗 URL

<https://api.github.com/user/starred/ahmetkurtbm/test-repo>

✅ Response

- 204 No Content: İşlem başarılı, ancak gövde dönmez.

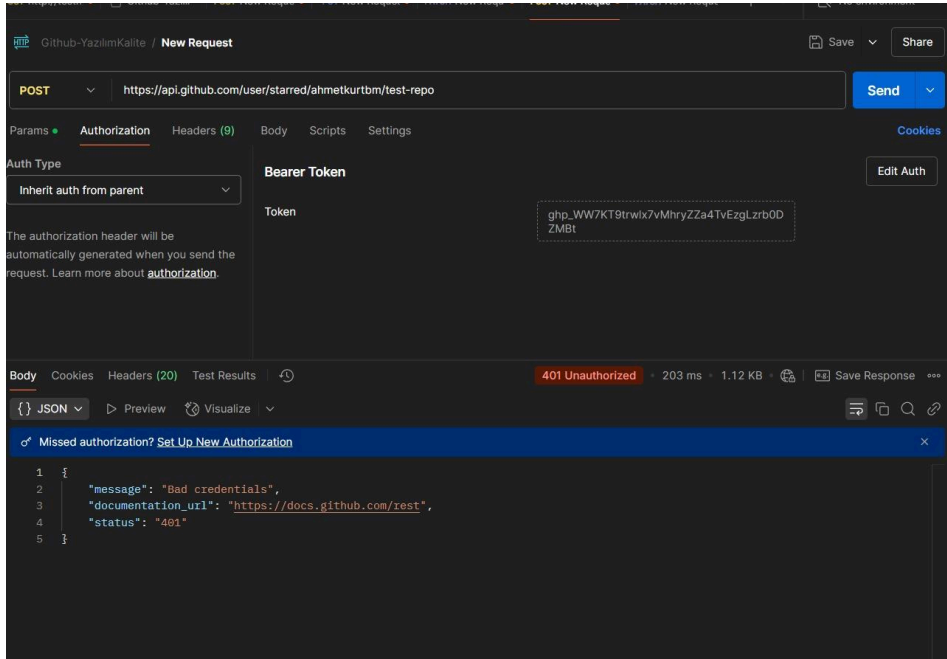


✗ 5. Hatalı Token ile Star Verme – POST /user/starred/...

🔍 Amaç: Token olmadan yetkisiz erişim denemesi.

✗ Response

- 401 Unauthorized: Geçersiz veya eksik token kullanımı.

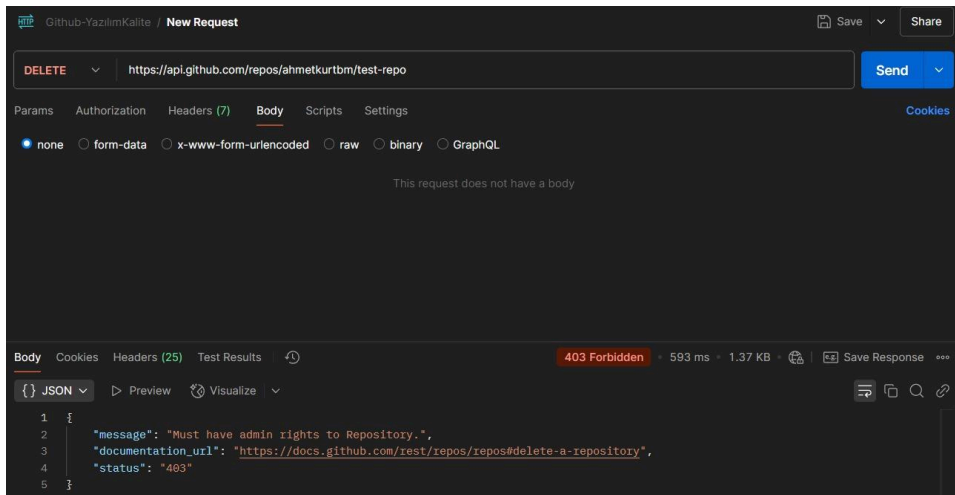


✗ 6. Repository Silme (Admin Yetkisi Yok) – DELETE /repos/...

🔍 Amaç: Kullanıcının sahibi olmadığı veya admin yetkisi olmayan bir repo'yu silmeye çalışmak.

✗ Response

- 403 Forbidden: Admin yetkisi gerektiği için reddedildi.



✓ 7. Repository Bilgilerini Genişletilmiş Güncelleme – PATCH

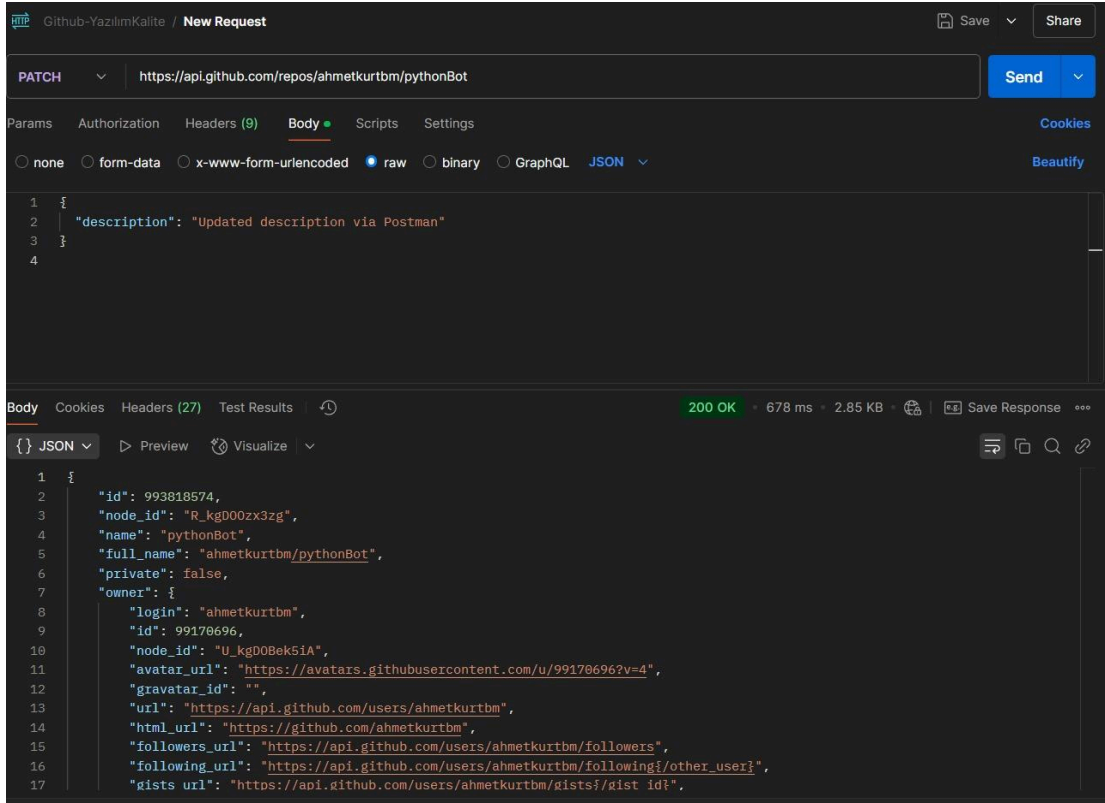
🔍 Amaç: Repo açıklamasını ve ana sayfa (homepage) bilgisini birlikte güncellemek.

📦 Body

```
{  "description": "Yeni açıklama: Postman ile güncellendi",  "homepage": "https://example.com",  "private": false}
```

✓ Response

- 200 OK: Güncelleme başarıyla yapıldı.



pythonBot Public

Yeni açıklama: Postman ile güncellendi

Python ☆ 1 Updated last week

★ Starred

✓ 8. Başarılı DELETE

🔍 Amaç: Kullanıcının sahibi olduğu bir repository'yi kalıcı olarak silmek.

🔗 URL

<https://api.github.com/repos/ahmetkurtbm/test-repo>

🔑 Gereken Yetki

- **admin** yetkisi şart.
- Geçerli bir token kullanılmalı.

✓ Response

- **204 No Content**: Silme işlemi başarıyla tamamlandı. Gövde dönmez.

4. Selenium ile UI Testleri:

a. Giriş: Selenium ve GitHub UI Testi

Selenium, web tabanlı uygulamalar için kullanıcı davranışlarını simüle ederek testler yazmayı sağlayan açık kaynaklı bir otomasyon aracıdır. Özellikle yazılım testi süreçlerinde kullanıcı arayüzünün doğru çalışıp çalışmadığını kontrol etmek için tercih edilir. Form doldurma, buton tıklama, sayfa geçişi gibi etkileşimleri otomatikleştirerek hem zaman kazandırır hem de insan hatasını azaltır.

Bu projede Selenium kullanılarak GitHub web arayüzü üzerinde kullanıcı senaryoları test edilmiştir. Test edilen senaryolar, gerçek bir kullanıcının GitHub üzerinde gerçekleştireceği temel işlemlerden oluşmaktadır. Amaç, GitHub üzerindeki akışların doğruluğunu test etmek ve karşılaşılan durumları otomatik olarak raporlamaktır.

b. Test Edilen Kullanıcı Senaryoları ve Sayfa Akışı

Aşağıda adım adım GitHub üzerinde gerçekleştirdiğimiz UI testleri açıklanmıştır. Her adımda kullanılan sayfa URL'leri, elementlerin tespit yöntemleri ve doğrulama mantıkları detaylı şekilde ele alınmıştır.

i. GitHub Ana Sayfası (<https://github.com>)

Amaç: Sisteme giriş yapmak için gerekli bağlantıyı ("Sign in") bulmak.

Kullanılan Element:

`sign_in_button` → `By.LINK_TEXT` ile "Sign in"

Aksiyon:

`click()` ile login sayfasına geçiş sağlandı.

ii. Giriş Sayfası (<https://github.com/login>)

Amaç: Kullanıcının geçerli bilgilerle giriş yapmasını sağlamak.

Elementler:

`username_input` → `By.NAME`, name="login"

`password_input` → `By.NAME`, name="password"

`login_button` → `By.NAME`, name="commit"

Doğrulamalar:

```
assert username_input.get_attribute("value") ==
```

```
GITHUB_USERNAME
```

```
assert login_button.is_enabled()
```

Geçiş: Başarılı giriş sonrası URL <https://github.com/> ile başlamalıdır.

Hata Durumu: Eğer URL <https://github.com/session> içeriyorsa, oturum hatası var demektir.

iii. Yeni Repo Oluşturma (<https://github.com/new>)

Amaç: Yeni bir depo oluşturmak.

Elementler:

repo_name_input → XPath: //input[@aria-required="true"]

repo_description → XPath: //input[@aria-label="Description"]

repo_available → class='prc-components-ValidationText-jjsBp'

create_button → XPath: /html/body/.../form/div[6]/button

Kontrol:

assert repo_name_input.get_attribute("value") == repo_name

assert "available" not in repo_available.text

assert "Selenium" in repo_description.get_attribute("value")

iv. Repo Sayfasına Gitme ve Dosya Yükleme

Amaç: test.html dosyasını yüklemek.

Sayfa: <https://github.com/{username}?tab=repositories>

Elementler:

repo_link → XPath: repo adını içeren <a> etiketi

upload_link → XPath: "uploading an existing file"

file_input → XPath: //input[@type='file']

commit_button → XPath: "Commit changes"

Kontrol:

assert file_input.is_displayed()

assert "test.html" in driver.page_source

v. Arama ve İlgili Repo Sayfasına Geçiş

Amaç: GitHub'da belirli bir repo ismini arayıp tıklamak.

Elementler:

search_button → XPath ile arama ikonu

search_box → ID: query-builder-test

repo_item → XPath: arama sonucundaki repo bağlantısı

Aksiyon: Arama kutusuna yazıp enter'a basmak, çıkan ilk sonuca gitmek.

vi. Dosya İndirme (Raw Buton)

Amaç: test.html dosyasını indirmek.

Element:

download_button → CSS Selector:
button[data-testid="download-raw-button"]

Kontrol:

assert download_button.is_displayed()

vii. Oturum Kapatma

Amaç: Sistemdeki kullanıcı oturumunu güvenli şekilde sonlandırmak.

Element: Logout formu XPath ile tespit edilir.

Aksiyon: click() işlemiyle çıkış yapılır.

c. Başarılı ve Başarısız Assertion Örnekleri

Adım	Kontrol Noktası	Durum
Giriş sayfası yönlendirme	assert "login" in driver.current_url	✓
Username alanı kontrolü	assert username_input.get_attribute(...)	✓
Giriş butonu durumu	assert login_button.is_enabled()	✓
Oturum hatası kontrolü	assert "session" not in driver.current_url	✓
Repo adı girildi mi?	assert repo_name_input.get_attribute(...)	✓
Repo adı uygun mu?	assert "available" not in repo_available.text.lower()	✓

Açıklama yazıldı mı?	assert "Selenium" in repo_description.get_attri bute(...)	✓
Dosya input görünür mü?	assert file_input.is_displayed()	✓
test.html yüklendi mi?	assert "test.html" in driver.page_source	✓
İndirme butonu var mı?	assert download_button.is_displ ayed()	✓

Başarısız bir durumda örneğin login bilgileri yanlış girilirse, Selenium otomatik olarak URL'nin <https://github.com/session> olduğunu kontrol ederek testin başarısız olduğunu belirtir.

```
if "https://github.com/session" in current_url:
    assert False, "Repo oluşturulamadı. Oturum hatası sayfasına yönlendi."
```

5. Selenium Kodunun Aşama Aşama Anlatımı ve Proje Dosya Yapısı

a. Selenium Kodunun Aşama Aşama Ne Yaptığı

Kodun genel yapısı bir GitHub kullanıcısının manuel olarak yaptığı şu adımları otomatikleştir:

i. Ortam Tanımlamaları ve Tarayıcı Ayarları

```
load_dotenv()
GITHUB_USERNAME = os.getenv("GITHUB_USERNAME")
GITHUB_PASSWORD = os.getenv("GITHUB_PASSWORD")
repo_name = "my-new-repo22"
```

".env" dosyasından kullanıcı adı ve şifresi okunur.
test.html dosyasının bulunduğu dizin belirlenir.

```
options = webdriver.ChromeOptions()
options.add_argument("--start-maximized")
```

Chrome tarayıcı arka planda ve tam ekran başlatılır.
Otomasyon izleri gizlenir (bot algısı azaltılır).

ii. Giriş Sayfasına Git ve Oturum Aç

```
driver.get("https://github.com")
sign_in_button = wait.until(EC.element_to_be_clickable((By.LINK_TEXT,
"Sign in"))))
sign_in_button.click()
```

GitHub ana sayfasına gidilir.

"Sign in" butonuna tıklanarak login ekranına geçilir.

```
username_input.send_keys(GITHUB_USERNAME)
password_input.send_keys(GITHUB_PASSWORD)
login_button.click()
```

Kullanıcı bilgileri girilir, oturum açma butonuna tıklanır.

Oturumun başarılı olup olmadığı URL kontrolüyle test edilir.

iii. Yeni Repository Oluştur

```
driver.get("https://github.com/new")
repo_name_input.send_keys(repo_name)
repo_description.send_keys("This is my new repository created using
Selenium.")
create_button.click()
```

Yeni repo sayfasına gidilir.

Repo adı ve açıklama girilerek yeni repo oluşturulur.

iv. Repo Sayfasına Git ve Dosya Yükle

```
driver.get(f"https://github.com/{GITHUB_USERNAME}?tab=repositories")
repo_link.click()
upload_link.click()
file_input.send_keys(file_path)
commit_button.click()
```

Kullanıcının repo listesine gidilir.

Yeni oluşan repo bulunur, "dosya yükleme" ekranına geçilir.

test.html yüklenir ve commit edilir.

v. Arama Yap ve Sonuçlara Git

```
search_button.click()
search_box.send_keys("repo:ahmetkurt-dev/ahmetkurt-dev ")
search_box.send_keys(Keys.ENTER)
repo_item.click()
```

GitHub üzerinden repo arama fonksiyonu test edilir.
Arama kutusuna repo bilgisi girilir ve ilk sonuca gidilir.

vi. Dosya İndir

```
download_button.click()
```

"Download raw" butonuna tıklanarak dosya indirme başlatılır.

vii. Oturumu Kapat

```
driver.get("https://github.com/logout")  
logout_button.click()
```

Kullanıcı oturumu sonlandırılır.

b. Proje Dosya Yapısı ve İçerikleri

```
github_ui_test_projesi/  
├── .env # GITHUB_USERNAME ve PASSWORD tanımları  
burada  
├── test.html # Test için yüklenecek örnek dosya  
├── selenium_test.py # Tüm Selenium kodları bu dosyada yazılı  
├── requirements.txt # Kullanılan Python kütüphaneleri  
├── README.md # Projenin amacı, çalıştırma talimatları  
└── test_senaryolari_raporu.md # Senaryolar ve sonuçların yazıldığı test raporu
```

requirements.txt

```
selenium  
python-dotenv
```









.env

```
GITHUB_USERNAME=kullaniciadi  
GITHUB_PASSWORD=sifre
```

Bu dosya yapısı, kodun kolay yönetilmesini ve testlerin tekrar edilebilir olmasını sağlar. Bu kapsamlı UI testiyle GitHub üzerinde temel kullanıcı işlemleri başarıyla simüle edilmiş ve doğruluğu test edilmiştir. Projeyi başka biri çalıştırdığında sıfırdan ayar yapmak yerine sadece **.env** dosyasını girerek sistemi ayağa kaldırabilir. Her adımda Selenium'un bekleme (**WebDriverWait**), etkileşim (**click**, **send_keys**) ve doğrulama (**assert**) yetenekleri kullanılmıştır. Bu proje sayesinde yazılım testlerinin hem teknik detayları hem de pratik uygulaması derinlemesine öğrenilmiştir.

6. Proje Genel Sonucu ve Değerlendirme

Bu proje, yazılım kalite güvencesi ve otomasyon testleri kapsamında hem API seviyesinde hem de UI seviyesinde kapsamlı bir test süreci uygulayarak başarıyla tamamlanmıştır. Test edilen uygulama olan GitHub, gerçek dünyada yaygın şekilde kullanılan bir sistem olduğundan dolayı test senaryolarının uygulanabilirliği yüksek olmuştur. Yapılan çalışmalar, hem bireysel hem de ekip temelli test süreçlerinde aşağıdaki kazanımları sağlamıştır:

-  GitHub üzerinde başarılı bir şekilde oturum açıldı.
-  Yeni bir repository oluşturuldu.
-  HTML dosyası sisteme yüklenerek "commit" işlemi gerçekleştirildi.
-  Arama alanında belirli bir repoya ulaşıldı.
-  Sisteme yüklenen dosya raw formatında indirildi.
-  Oturum güvenli bir şekilde kapatıldı.
-  API seviyesinde; kullanıcı bilgisi, repo arama, takipçi listesi, repo indirme gibi testler başarılı şekilde çalıştırıldı.
-  API ve UI testleri başarı/başarısızlık durumlarını doğru şekilde raporladı.
- Selenium ile çok adımlı UI otomasyon senaryoları yazıldı.
- Postman üzerinden API testleri yapılırken parametre çeşitliliği (q, sort, order) kullanıldı.
- assert yapısı ile her adıma özel doğrulamalar gerçekleştirildi.
- Hatalı input senaryoları ile sistemin dayanıklılığı kontrol edildi.
- test.html dosyasının yüklenmesi ve dosya sisteminde görünmesi simüle edildi.
- Dosya indirme akışı download-raw-button üzerinden tespit edilerek kullanıcıya gerçek deneyim kazandırıldı.

7. Sonuç

Bu proje, yazılım test otomasyonunun hem teknik hem de uygulamalı yönlerini öğreterek öğrencilere gerçek dünya test süreçlerini deneyimleme fırsatı sunmuştur. API ve UI testlerinin birbirini tamamlayan yapıları üzerinden geliştirilen bu test planı, test mühendisliği yaklaşımına uygun olarak hazırlanmıştır.

8. Kaynakça

- <https://stateful.com/blog/github-search-api>
- <https://docs.github.com/en/rest/using-the-rest-api/getting-started-with-the-rest-api?apiVersion=2022-11-28>
- <https://ksdevware.com/github-users-api/>
- <https://docs.github.com/en/rest/users/followers?apiVersion=2022-11-28>
- <https://stateful.com/blog/github-api-list-repositories>