API Otomasyonunu Öğrenme (Learning API Automation)

1. Giriş

API Automation Nedir?

API (Application Programming Interface) Automation, **yazılımın arka uç servislerini otomatik test etme** sürecidir.

- **Manuel Test:** QA, Postman veya başka araçla endpoint'e istek atar, response'u kontrol eder.
- **API Automation:** Kod ile aynı isteği tekrar tekrar gönderir, response'ları otomatik doğrular ve raporlar.

Örnek:

- **Manuel Test:** Postman'da GET request gönder → Status Code 200 mü kontrol et.
- **API Automation:** Python + Requests veya Postman Test Scripts ile GET request gönder → Otomatik assertion → Rapor oluştur.

Neden Test Süreçlerinde Önemli?

1. **Tekrarlanabilirlik**
    - Aynı endpoint'i yüzlerce senaryoda test et.
2. **Hız ve Güvenilirlik**
    - Manuel kontrol yerine kodla her zaman aynı sonuç.
3. **CI/CD Entegrasyonu**
    - Kod push edildiğinde API testleri otomatik çalışır.
4. **Hata Yakalama**
    - Endpoint'ler doğru çalışıyor mu? Parametreler doğru mu işleniyor?

API Türleri ve HTTP Methods

- **REST API:** HTTP protokolü üzerinden CRUD (GET, POST, PUT, DELETE) işlemleri.
- **SOAP API:** XML tabanlı mesajlaşma, WSDL ile servis tanımı.

Örnek HTTP Method Kullanımı:

Method	Amaç	Örnek Endpoint
GET	Veri çekme	/users/1
POST	Yeni veri oluşturma	/users

Method	Amaç	Örnek Endpoint
PUT	Mevcut veriyi güncelle	/users/1
DELETE	Veri silme	/users/1

---

2. API Test Araçları ve Yaklaşımı

Postman Tool

- Request oluşturma
- Environment ve collection yönetimi
- Test script yazma (JavaScript ile assertion)
- Collection Runner ile otomatik test koşturma

Python + Requests / PyTest

- Requests ile endpoint'e istek gönderme
- PyTest ile assertion ve raporlama
- CI/CD entegrasyonu için kolay script çalıştırma

---

3. Demo: API Automation (Python + PyTest + Requests)

Klasör Yapısı

```
API_TEST/
├─ tests/
│  ├─ test_users.py
│  └─ test_auth.py
│
├─ utils/
│  └─ api_client.py
│
└─ requirements.txt
```

---

a) utils/api_client.py

Amaç: API çağrılarını merkezi yönetmek, base URL, headers, auth vb. kontrol etmek.

```python
# utils/api_client.py
import requests

class APIClient:
    def __init__(self, base_url: str):
        self.base_url = base_url
```

```
    def get(self, endpoint, headers=None, params=None):
        return requests.get(f"{self.base_url}{endpoint}", headers=headers,
params=params)

    def post(self, endpoint, headers=None, data=None, json=None):
        return requests.post(f"{self.base_url}{endpoint}", headers=headers,
data=data, json=json)

    def put(self, endpoint, headers=None, data=None, json=None):
        return requests.put(f"{self.base_url}{endpoint}", headers=headers,
data=data, json=json)

    def delete(self, endpoint, headers=None):
        return requests.delete(f"{self.base_url}{endpoint}", headers=headers)
```

---

b) tests/test_users.py

Amaç: Users API endpoint'lerini otomatik test etmek.

```
# tests/test_users.py
import pytest
from utils.api_client import APIClient

BASE_URL = "https://jsonplaceholder.typicode.com"
client = APIClient(BASE_URL)

def test_get_user():
    response = client.get("/users/1")
    assert response.status_code == 200
    data = response.json()
    assert data["id"] == 1
    assert "username" in data

def test_create_user():
    payload = {"name": "Harun", "username": "hk", "email": "harun@example.com"}
    response = client.post("/users", json=payload)
    assert response.status_code == 201
    data = response.json()
    assert data["name"] == "Harun"

def test_update_user():
    payload = {"name": "Harun Korkmaz"}
    response = client.put("/users/1", json=payload)
    assert response.status_code == 200
    data = response.json()
    assert data["name"] == "Harun Korkmaz"

def test_delete_user():
```

```
    response = client.delete("/users/1")
    assert response.status_code == 200 or response.status_code == 204
```

---

c) requirements.txt

```
pytest
requests
```

---

d) Çalıştırma

```
# Sanal ortam oluştur
python -m venv venv
source venv/bin/activate  # Linux / Mac
venv\Scripts\activate     # Windows

# Paketleri yükle
pip install -r requirements.txt

# Testleri çalıştır
pytest tests/
```

---

4. Akışın Mantığı (Demo ile Detaylı Açıklama)

4.1 API Test Yapısının Katmanları

1. **utils/api_client.py → API Client Katmanı**
   - Tüm HTTP isteklerini merkezi olarak yönetir.
   - Base URL, headers, authentication token gibi ortak ayarlar burada tutulur.
   - Avantaj: Endpoint değişirse sadece client'i güncellemen yeterli, testleri tekrar yazmana gerek yok.

   Örnek kullanım:

   ```
   client = APIClient("https://jsonplaceholder.typicode.com")
   response = client.get("/users/1")
   ```

2. **tests/ → Test Katmanı**
   - Endpoint'ler için birebir test senaryoları burada yazılır.
   - PyTest test fonksiyonları ile GET, POST, PUT, DELETE işlemleri otomatik yapılır.
   - Assertions ile hem **status code** hem **response içeriği** doğrulanır.

Örnek kullanım:

```
def test_get_user():
    response = client.get("/users/1")
    assert response.status_code == 200
    assert "username" in response.json()
```

3. **requirements.txt → Bağımlılık Katmanı**
    - PyTest ve Requests gibi paketlerin yönetimi burada.
    - Ortam kurulumu ve CI/CD entegrasyonu kolaylaşır.

4.2 Test Akışı Adım Adım

Senaryomuz: "Kullanıcı endpoint'lerini otomatik test et."

1. **Test Başlatılır**

```
* PyTest veya CI/CD pipeline tarafından `tests/` klasörü içindeki testler
çağrılır.
* Her test fonksiyonu bağımsız olarak çalışır.
```

2. **API Client Oluşturulur**

```
* `APIClient` sınıfı üzerinden base URL ve ortak ayarlar tanımlanır.
* Header veya Auth gerekli ise burada eklenir.
```

3. **Request Gönderilir**

```
* Test fonksiyonu endpoint'e GET/POST/PUT/DELETE isteği gönderir.
* Örneğin:

  ```python
  response = client.post("/users", json=payload)
  ```
```

4. **Response Kontrol Edilir (Assertion)**

```
* Status code doğrulanır (200, 201, 204 vb.)
* Response içindeki kritik alanlar kontrol edilir:

  ```python
```
```

```
  assert data["name"] == "Harun"
  ```
```

5. **Test Sonucu Raporlanır**

```
* PyTest çıktısı ile test PASS/FAIL olarak raporlanır.
* CI/CD entegrasyonunda bu sonuçlar pipeline raporuna eklenir.
```

6. **Tüm Testler Tamamlanır**

```
* Tüm testler bittiğinde, herhangi bir hata veya başarısız test loglanır.
* Testler bağımsız olduğundan, bir testin başarısız olması diğerlerini etkilemez.
```

4.3 Örnek Akış – "GET /users/1" Testi

1. `test_get_user()` fonksiyonu çalıştırılır.
2. `APIClient.get("/users/1")` çağrılır → HTTP GET request gönderilir.
3. Server'dan response alınır → status code 200 olmalı.
4. Response JSON parse edilir → "username" alanı var mı kontrol edilir.
5. Assertion başarılı → PASS, başarısız → FAIL ve hata loglanır.

4.4 PyTest ve API Client'ın Avantajları

Katman / Yapı	Avantajları
APIClient	Kod tekrarını önler, merkezi yönetim sağlar.
PyTest test fonksiyonları	Her endpoint için bağımsız, hızlı ve tekrarlanabilir testler.
Assertions	Status code ve response doğrulama, manuel kontrol ihtiyacını ortadan kaldırır.
CI/CD entegrasyonu	Kod push edildiğinde testler otomatik çalışır, regression riskini azaltır.

4.5 Akış Şeması (Mantıksal)

```
Başlat → PyTest test fonksiyonunu çağır
        ↓
APIClient üzerinden request oluştur
        ↓
Request gönder → Response al
        ↓
Status code ve response doğrula
```
```

```
        ↓
PASS / FAIL → Logla / CI/CD raporla
        ↓
Tüm testler tamam → Süreç bitir
```

4.6 Demo Bağlantısı

- GUI Automation'da POM sınıfı ile sayfa elementlerini yönetmiştik.
- API Automation'da **APIClient** POM mantığının karşılığıdır.
- Test fonksiyonları GUI'daki feature adımlarına denk gelir.
- Assertions → GUI'daki `is_secure_area()` veya `get_flash_message()` gibi doğrulamalar.

---

5. Sonuç

Bugün öğrendiklerim:

- API Automation'ın manuel testlere göre hız ve güvenilirlik farkı.
- Python + Requests + PyTest ile otomatik API testi yazabilme.
- CI/CD entegrasyonu için hazır script ve yapı oluşturabilme.
- API client tasarımı ile kod tekrarı azaltıldı ve bakım kolaylığı sağlandı.

Katkısı: Artık gerçek projelerde API regression testlerini hızlı, güvenilir ve tekrar edilebilir şekilde yazabilecek, yeni servisler eklendiğinde kolayca test edebileceğim bir yapı kurdum.