GUI(Grafiksel Kullanıcı Arayüzü) Otomasyonunu Öğrenme (Learning GUI Automation)

1. Giriş

GUI Automation Nedir?

GUI (Graphical User Interface) Automation, yazılımın **kullanıcı arayüzünü test etme işini otomatikleştirme** yöntemidir. Normalde bir QA test mühendisi arayüzde butona tıklar, metin girer veya ekranın doğru açılıp açılmadığını gözle kontrol eder. GUI Automation'da ise bu adımlar kodla veya araçlarla tanımlanır ve bilgisayar bunları otomatik tekrarlar.

Örnek:

- **Manuel Test:** Login ekranına kullanıcı adı gir, şifreyi yaz, login butonuna bas, ana sayfa geldi mi gözle kontrol et.
- **GUI Automation:** Selenium gibi bir araçla aynı adımları kodla yaz, script'i çalıştır, test otomatik login olur ve sonucu kendisi doğrular.

Neden Test Süreçlerinde Önemli?

1. **Zaman kazandırır**
    - Bir senaryo yüzlerce defa koşabilir.
    - Örneğin aynı login testini 5 farklı tarayıcıda 200 defa koşturmak, manuel testte günler alırken otomasyonla birkaç saat sürer.
2. **İnsan hatasını azaltır**
    - Manuel testte dikkatsizlik (yanlış tıklama, eksik input) olabilir.
    - Otomasyonda her zaman aynı adımlar çalışır.
3. **CI/CD entegrasyonu sağlar**
    - Kod push edildiğinde otomatik regression testleri tetiklenir.
    - Yeni bug'lar daha canlıya çıkmadan yakalanır.
4. **Maliyet avantajı sunar**
    - İlk yatırım (framework, altyapı, yazım) maliyetlidir.
    - Ama orta–uzun vadede regresyon yükünü azalttığı için ciddi kazanç sağlar.

Manuel Test vs GUI Automation

Özellik	Manuel Test	GUI Automation
Hız	Yavaş	Çok hızlı

Özellik	Manuel Test	GUI Automation
Tekrar	Zor ve yorucu	Sınırsız tekrar
İnsan hatası	Yüksek	Minimum
Uygunluk	Küçük projeler	Orta–büyük ölçekli projeler

Gerçek Hayatta Kullanım Alanları

- **Web Uygulamaları:** Login, ürün sepete ekleme, ödeme, form doldurma.
- **Mobil Uygulamalar:** iOS/Android testleri (Appium).
- **Desktop Uygulamaları:** WinAppDriver veya Pywinauto ile masaüstü yazılımlar.
- **Regression Testing:** Yeni feature geldiğinde eskilerin bozulmadığını garanti etmek.
- **Cross-browser Testing:** Chrome, Firefox, Safari, Edge üzerinde aynı senaryo.

---

2. BDD Yaklaşımı (Behavior Driven Development)

BDD Nedir?

BDD, yazılım davranışını (behavior) hem teknik hem iş tarafının anlayabileceği şekilde tanımlayan bir yaklaşımdır. Amaç, testlerin teknik terimlerden bağımsız, doğal dile yakın yazılmasıdır.

TDD ile Farkları

- **TDD:** Önce test kodu yazılır → uygulama geliştirilir → test koşar.
- **BDD:** Senaryolar iş kurallarıyla yazılır (Given–When–Then), herkes okuyabilir.

Frameworkler

- **Cucumber (Java, JS, Python)**
- **SpecFlow (.NET)**
- **Behave (Python)**

Given–When–Then Formatı

```
Feature: Login Functionality
  Scenario: Successful Login
    Given Kullanıcı login sayfasında
    When Kullanıcı geçerli bilgilerle giriş yapar
    Then Kullanıcı ana sayfaya yönlendirilir
```

- **Given:** Başlangıç → Kullanıcı login sayfasında.
- **When:** Eylem → Kullanıcı geçerli bilgileri girer.
- **Then:** Beklenen sonuç → Ana sayfa açılır.

GUI Testlerinde Rolü

- İş analistlerinin yazdığı senaryolar doğrudan otomasyon testine çevrilebilir.
- QA, developer ve iş ekibi aynı senaryoya bakıp anlayabilir.
- Gereksinimlerle otomasyon arasında şeffaf köprü sağlar.

---

3. POM (Page Object Model)

POM Nedir?

POM, GUI otomasyonunda kullanılan **tasarım deseni**dir. Her sayfa için ayrı bir class tanımlanır. Bu class sayfadaki elementleri ve onlarla yapılacak işlemleri içerir.

Avantajları

- Tekrar kullanılabilirlik: Aynı sayfa farklı testlerde tekrar kullanılabilir.
- Bakım kolaylığı: UI değişirse sadece ilgili Page Class güncellenir.
- Okunabilirlik: Test Class sadece senaryoyu yazar.

Yapı

- **Page Class:** Sayfa elementleri ve metotları.
- **Test Class:** Page Class kullanılarak senaryonun çalıştırıldığı testler.
- **Utilities:** Ortak araçlar (driver setup, raporlama, logger).

---

4. GUI Automation Demo (Python + Selenium + POM)

Klasör Yapısı

```
TEST/
├─ features/
│  ├─ login.feature
│  └─ environment.py
│
├─ features/steps/
│  └─ login_steps.py
│
├─ pages/
│  └─ login_page.py
│
├─ utils/
│  └─ driver_factory.py
│
└─ tests/
   └─ test_login_basic.py
```

> Not: Paket çözümlemesi için klasörlerin **paket olmasına gerek yok** (Behave böyle istiyor). `tests/` altında PyTest/manuel çalıştırma için dosya var.

---

a) utils/driver_factory.py

Amaç: Sürücüyü tek noktadan oluşturmak (headless/normal, pencere boyutu, log susturma, driver yönetimi).

```
# utils/driver_factory.py
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager

def create_driver(headless: bool = False):
    options = Options()
    if headless:
        options.add_argument("--headless=new")
    options.add_argument("--window-size=1366,768")
    options.add_argument("--no-sandbox")
    options.add_argument("--disable-dev-shm-usage")
    options.add_experimental_option("excludeSwitches", ["enable-logging"])

    service = Service(ChromeDriverManager().install())
    driver = webdriver.Chrome(service=service, options=options)
    driver.implicitly_wait(3)
    return driver
```

Nerede kullanılıyor?

- Behave akışında `features/environment.py` driver'ı buradan üretip `context.driver` içine koyar.
- PyTest/manuel senaryoda `tests/test_login_basic.py` içinden çağrılır.

---

b) pages/login_page.py

Amaç: POM. Sayfadaki elementleri ve işlemleri tek sınıfta toplar. Test/step'ler doğrudan Selenium locator'larına dokunmaz, bu sınıfı kullanır.

```
# pages/login_page.py
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

class LoginPage:
    BASE_URL = "https://the-internet.herokuapp.com/login"

    USERNAME = (By.ID, "username")
    PASSWORD = (By.ID, "password")
    LOGIN_BTN = (By.CSS_SELECTOR, "button.radius")
```

```
    FLASH = (By.ID, "flash")
    SECURE_HEADER = (By.CSS_SELECTOR, "div.example h2")

    def __init__(self, driver, timeout: int = 10):
        self.driver = driver
        self.wait = WebDriverWait(driver, timeout)

    def open(self):
        self.driver.get(self.BASE_URL)

    def set_username(self, username: str):
        self.wait.until(EC.visibility_of_element_located(self.USERNAME)).clear()
        self.driver.find_element(*self.USERNAME).send_keys(username)

    def set_password(self, password: str):
        self.driver.find_element(*self.PASSWORD).clear()
        self.driver.find_element(*self.PASSWORD).send_keys(password)

    def click_login(self):
        self.driver.find_element(*self.LOGIN_BTN).click()

    def login(self, username: str, password: str):
        self.set_username(username)
        self.set_password(password)
        self.click_login()

    def get_flash_message(self) -> str:
        elem = self.wait.until(EC.visibility_of_element_located(self.FLASH))
        return elem.text.strip().replace("×", "").strip()

    def is_secure_area(self) -> bool:
        try:
            self.wait.until(EC.visibility_of_element_located(self.SECURE_HEADER))
            return "/secure" in self.driver.current_url
        except:
            return False
```

Nerede kullanılıyor?

- Hem Behave step'lerinde (`features/steps/login_steps.py`) hem de `tests/test_login_basic.py` içinde.

---

c) features/environment.py

Amaç: Behave yaşam döngüsü. Senaryo başlamadan driver aç, bittikten sonra kapat. Böylece her senaryo izole olur.

```
# features/environment.py
from utils.driver_factory import create_driver
```

```
def before_scenario(context, scenario):
    context.driver = create_driver(headless=False)

def after_scenario(context, scenario):
    if hasattr(context, "driver"):
        context.driver.quit()
```

Nerede kullanılıyor?

- `behave` komutu çalışırken Behave otomatik çağırır. Step dosyalarında `context.driver` hazır bulunur.

---

d) features/login.feature

Amaç: İş dilinde (Given–When–Then) senaryolar. Step'lerin isimleriyle birebir eşleşir.

```
# features/login.feature
Feature: Login Functionality (GUI)
  As a user
  I want to log in to the application
  So that I can access the secure area

  Scenario: Successful login with valid credentials
    Given kullanıcı login sayfasında
    When kullanıcı geçerli kullanıcı adı ve şifre ile giriş yapar
    Then kullanıcı güvenli alanı görmelidir

  Scenario: Unsuccessful login with wrong password
    Given kullanıcı login sayfasında
    When kullanıcı geçersiz şifre ile giriş yapar
    Then kullanıcı hata mesajını görmelidir
```

Nerede kullanılıyor?

- `behave` komutu çalıştırıldığında Behave bu dosyadaki adımları `steps/` altındaki Python fonksiyonlarına eşler.

---

e) features/steps/login_steps.py

Amaç: Feature'daki Given–When–Then cümlelerinin Selenium koduna bağlanması. POM sınıfı kullanılıyor.

```
# features/steps/login_steps.py
from behave import given, when, then
from selenium.webdriver.common.by import By
from pages.login_page import LoginPage

@given("kullanıcı login sayfasında")
def step_open_login(context):
```

```
    context.page = LoginPage(context.driver)
    context.page.open()

@when("kullanıcı geçerli kullanıcı adı ve şifre ile giriş yapar")
def step_login_valid(context):
    context.page.login("tomsmith", "SuperSecretPassword!")

@then("kullanıcı güvenli alanı görmelidir")
def step_assert_secure(context):
    assert context.page.is_secure_area(), "Secure Area görünmedi!"
    flash = context.page.get_flash_message()
    assert "You logged into a secure area!" in flash

@when("kullanıcı geçersiz şifre ile giriş yapar")
def step_login_invalid(context):
    context.page.login("tomsmith", "yanlis_sifre")

@then("kullanıcı hata mesajını görmelidir")
def step_assert_error(context):
    flash = context.page.get_flash_message()
    assert ("Your password is invalid!" in flash) or ("Your username is invalid!"
in flash), \
        f"Beklenen hata mesajı gelmedi. Gelen: {flash}"
```

Nerede kullanılıyor?

- `behave` çalışırken feature'daki adımlar bu fonksiyonlara gelir. Her step içinde **POM** metotları çağrılır.

---

f) tests/test_login_basic.py (opsiyonel)

Amaç: Behave olmadan da aynı POM ile çalıştırmak istersen. Run butonuna basıp hızlı doğrulama yaparsın.

```
# tests/test_login_basic.py
from utils.driver_factory import create_driver
from pages.login_page import LoginPage

def test_login_positive():
    driver = create_driver(headless=False)
    page = LoginPage(driver)
    try:
        page.open()
        page.login("tomsmith", "SuperSecretPassword!")
        assert page.is_secure_area(), "Login başarısız!"
        flash = page.get_flash_message()
        assert "You logged into a secure area!" in flash
        print("[PASS] Pozitif login")
    finally:
        driver.quit()

def test_login_negative():
```

```
    driver = create_driver(headless=True)
    page = LoginPage(driver)
    try:
        page.open()
        page.login("tomsmith", "yanlis_sifre")
        flash = page.get_flash_message()
        assert "Your password is invalid!" in flash or "Your username is invalid!"
in flash
        print("[PASS] Negatif login")
    finally:
        driver.quit()

if __name__ == "__main__":
    test_login_positive()
    test_login_negative()
```

Çalıştırma

Kurulum:

```
pip install selenium webdriver-manager behave
```

Behave ile (BDD):

```
cd TEST
behave
```

Tek dosya (isteğe bağlı):

```
python tests/test_login_basic.py
```

Akışın Mantığı (Demo ile bağ)

1. `behave` dediğinde:

- Behave `features/login.feature` dosyasını okur.
- Her adımı `features/steps/login_steps.py` içindeki fonksiyonlara eşler.
- Senaryo başlamadan `features/environment.py` → `before_scenario` çağrılır, `context.driver` üretilir.
- Step'ler içinde `LoginPage(context.driver)` ile POM kullanılır.
- Doğrulamalar (assertion) POM üzerinden okunur (`is_secure_area`, `get_flash_message`).
- Senaryo bitince `after_scenario` ile tarayıcı kapanır.

2. `test_login_basic.py`:

- Aynı POM sınıfını kullanır ama Behave yerine doğrudan Python ile çalışır.

---

5. Kapanış

Bugün öğrendiklerim:

- GUI Automation'ın manuel testlere göre hız ve güvenilirlik farkı.
- BDD'nin iş kurallarıyla teknik tarafı birleştiren yapısı.
- POM tasarım deseninin bakım ve okunabilirlik avantajı.
- Python + Selenium ile login senaryosunu otomatik çalıştırma.

Katkısı: Artık gerçek projelerde regression yükünü azaltabilecek, CI/CD'ye entegre otomasyon testleri yazabilecek seviyeye geldim. Ayrıca iş ve teknik ekip arasındaki köprüyü kuracak senaryoları yazmayı da daha iyi anlıyorum.