T.C.

Fırat Üniversitesi

Teknoloji Fakültesi
Yazılım Mühendisliği Bölümü

YAZILIM KALİTE GÜVENCESİ VE TESTİ DENEY FÖYÜ

14 Haftalık Uygulamalı Selenium Temelli Ders Deneyleri
(Kod Örnekleri, Sorular ve Açıklamalar ile)

Dersin Öğretim Üyesi:
Doç. Dr. Muhammed BAYKARA
Asistan: Hüseyin Alperen Dağdöğen

2025 - 2026 Güz Dönemi

Yazılım Kalite Güvencesi ve Testi - Deney Föyü

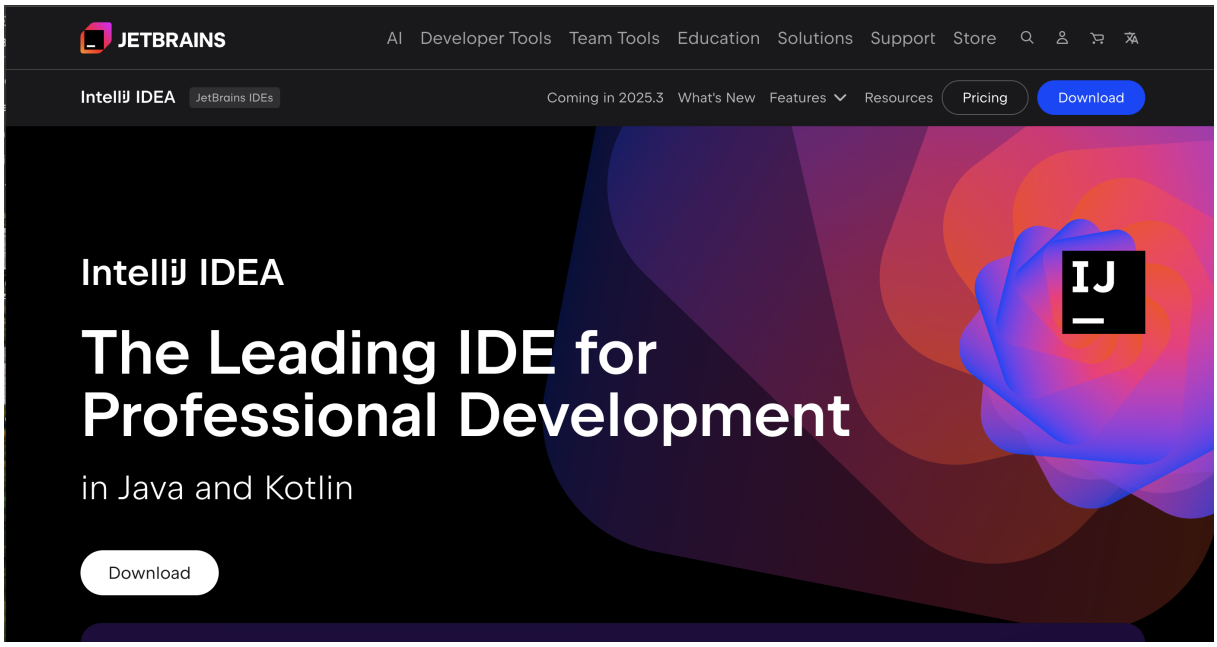
1. Hafta Selenium'a Giriş

Amaç: Modern web uygulamaları üzerinde test otomasyonu gerçekleştirmek için Selenium WebDriver'ın kurulumu, mimarisi ve temel prensiplerinin öğrenilmesi.

Kapsam: WebDriver kurulumu, ChromeDriver/GeckoDriver kullanımı, DOM yapısı analizi, element bulma yöntemleri (id, class, name, xpath, cssSelector), tarayıcı yönetimi.

Kod Örneği:

```
System.setProperty("webdriver.chrome.driver", "drivers/
    chromedriver.exe");
WebDriver driver = new ChromeDriver();
driver.get("https://example.com");
WebElement search = driver.findElement(By.name("q"));
search.sendKeys("Selenium␣WebDriver");
search.submit();
driver.quit();
```



Sorular:

1. WebDriver'ın Remote Server yapısı ile istemci arasındaki iletişimi ve HTTP üzerinden komut aktarımını açıklayınız.
2. Single Page Application (SPA) tabanlı sayfalarda test senaryolarında karşılaşılabilecek senkronizasyon problemleri nelerdir ve nasıl çözülür?

3. Dinamik ID içeren elementler için en güvenli XPath oluşturma tekniklerini örnekle gösteriniz.

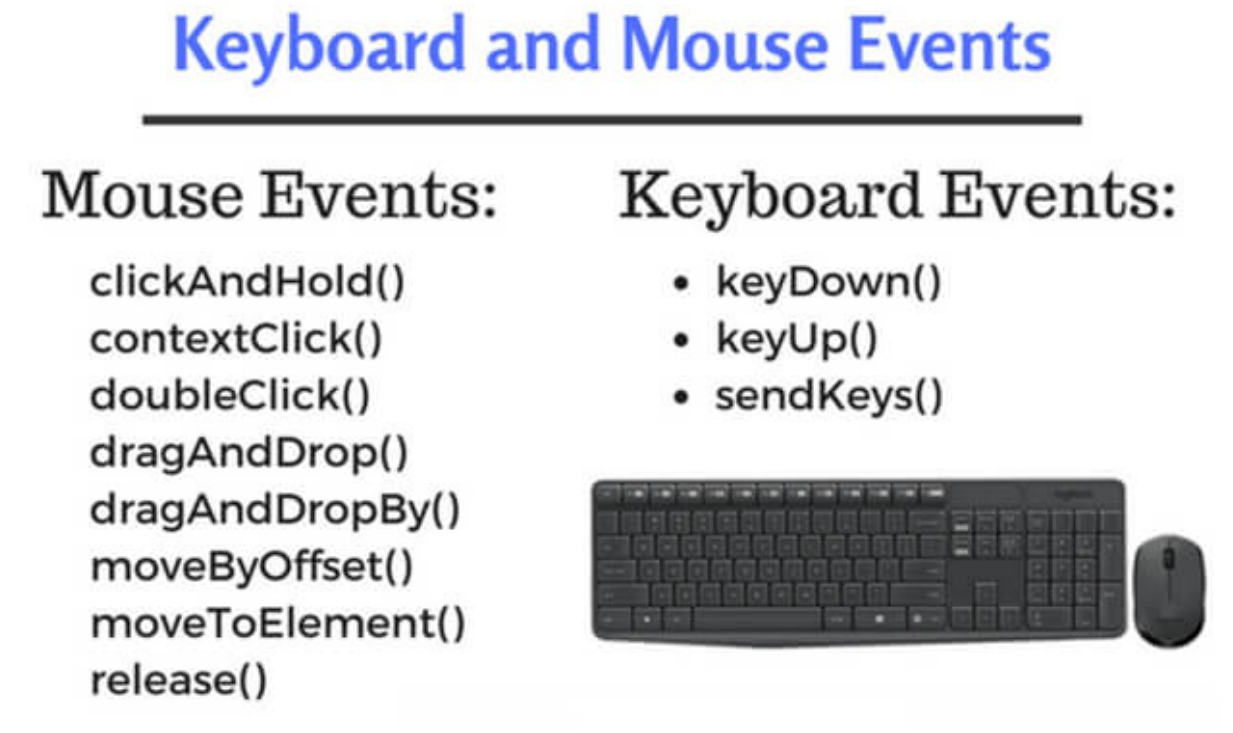
2. Hafta Selenium ile Etkileşimli Senaryolar

Amaç: Web elementleriyle ileri düzey etkileşim sağlayarak gerçek kullanıcı senaryolarının otomasyonunu sağlamak.

Kapsam: Click, SendKeys, Clear, Submit, Action Class, Keyboard/Mouse işlemleri, Scroll, Alert yönetimi.

Kod Örneği:

```
Actions actions = new Actions(driver);
actions.moveToElement(driver.findElement(By.id("menu"))).click().
   perform();
Alert alert = driver.switchTo().alert();
alert.accept();
```



Sorular:

1. Explicit wait ve Fluent wait arasındaki farkları açıklayın ve her biri için gerçek bir kullanım senaryosu tanımlayınız.
2. Mouse hover ve sağ tıklama gibi kullanıcı davranışları Selenium'da nasıl simüle edilir? Kodla açıklayınız.
3. JavaScript ile oluşturulmuş custom alert kutularını nasıl kontrol edersiniz?

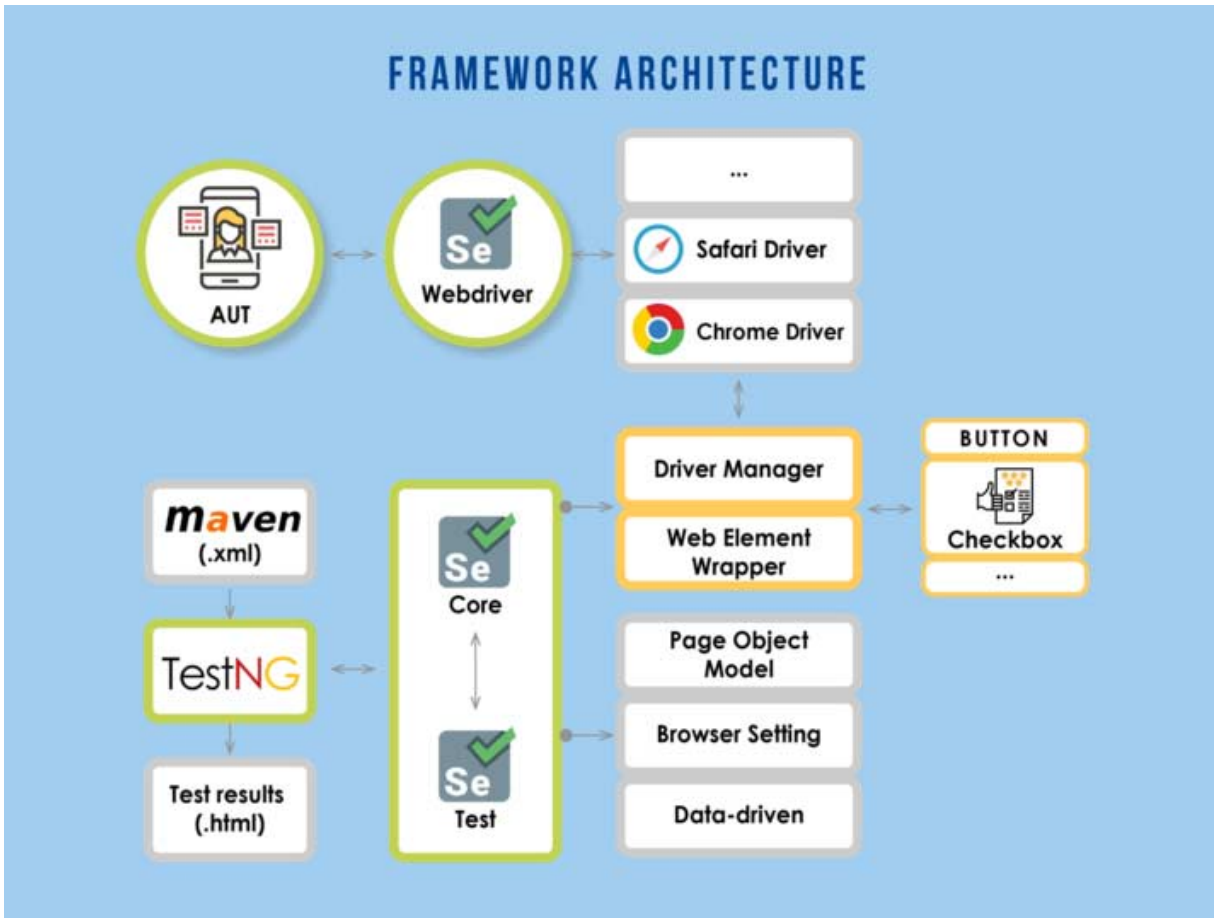
3. Hafta Selenium'da İleri Seviye Uygulamalar

Amaç: Karmaşık yapıdaki web uygulamaları üzerinde çerçeve geçişleri, pencere yönetimi ve JavaScript etkileşimi gibi ileri düzey işlemleri uygulamak.

Kapsam: iframe, window handling, JavaScriptExecutor, File upload/download, Shadow DOM erişimi.

Kod Örneği:

```
driver.switchTo().frame("iframeName");
String mainWindow = driver.getWindowHandle();
for(String handle : driver.getWindowHandles()) {
  if (!handle.equals(mainWindow)) {
    driver.switchTo().window(handle);
  }
}
```



Sorular:

1. Birden fazla pencereyle çalışan bir test senaryosunda, her pencereye özel işlemler nasıl yapılır? Kod ile gösteriniz.
2. iframe içindeki iframe'lerde element bulma problemi nasıl çözülür?
3. JavaScriptExecutor kullanarak sayfadaki görünmeyen bir elementi görünür hale getirme işlemi nasıl yapılır?

4. Hafta JUnit ile Test Otomasyonu

Amaç: Java ortamında test otomasyonu için en yaygın kullanılan JUnit framework'ü ile temel test sınıfları oluşturmak.

Kapsam: JUnit kurulumu, @Test anotasyonu, Assertions, test başarısızlık yönetimi.

Kod Örneği:

```
@Test
public void testToplama() {
  int sonuc = Matematik.topla(3, 4);
  assertEquals(7, sonuc);
}
```

```
1  package demo.tests;
2
3  import static org.junit.Assert.*;
8
9  public class JUnitProgram {
10
11     @Before
12     public void setUp() throws Exception {
13     }
14
15     @After
16     public void tearDown() throws Exception {
17     }
18
19     @Test
20     public void test() {
21         fail("Not yet implemented");
22     }
23
24 }
```

Sorular:

1. JUnit'te exception fırlatan metotların test edilmesi için hangi yapılar kullanılır?
2. assertThat() metodunun avantajları nelerdir?
3. Bir metodun testten önce ve sonra çalışması nasıl sağlanır? Hangi senaryolarda kullanılır?

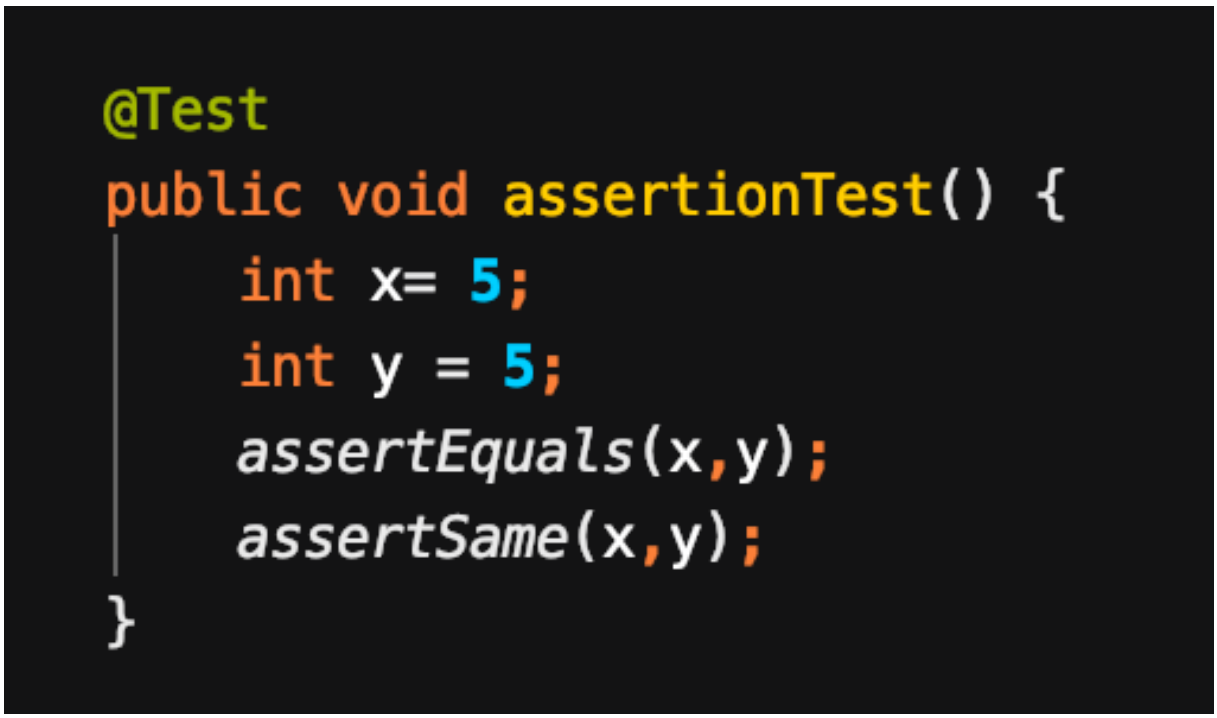
5. Hafta JUnit Test Suites ve Parametreli Testler

Amaç: Test sınıflarını gruplayarak testleri topluca yönetmek ve veriyle çalışan testlerin kapsamını genişletmek.

Kapsam: @RunWith, @Suite.SuiteClasses, Parameterized testler, CSV/JSON veri kaynakları.

Kod Örneği:

```
@RunWith(Parameterized.class)
public class CiftlikTest {
  @Parameters
  public static Collection<Object[]> veriler() {
    return Arrays.asList(new Object[][] {{2, true}, {3, false}});
  }
}
```



Sorular:

1. Parametreli testlerin avantajları nelerdir? Hangi durumlarda tercih edilmemelidir?
2. TestSuite yapısının büyük projelerde modülerlik açısından faydaları nelerdir?
3. External veri kaynaklarıyla test beslemenin riskleri ve çözüm yöntemleri nelerdir?

6. Hafta 6. Hafta: TestNG ile Gelişmiş Test Yönetimi

Amaç: Java tabanlı projelerde TestNG çerçevesi kullanarak testleri organize etmek, bağımlılıklar tanımlamak, paralel test çalıştırmak ve raporlama sürecini otomatikleştirmektir.

Kapsam:

- TestNG kurulumu ve yapılandırma
- Anotasyonlar: `@BeforeClass`, `@AfterClass`, `@BeforeMethod`, `@AfterMethod`
- Test gruplama (`groups`)
- Test bağımlılıkları (`dependsOnMethods`)
- Öncelik sıralaması (`priority`)
- Parametre geçişi ve `@DataProvider` kullanımı
- XML dosyası ile test senaryosu yönetimi (`testng.xml`)
- Paralel test çalıştırma (multi-thread)
- Raporlama (`emailable-report.html`, `index.html`)
- Jenkins veya CI/CD ortamında TestNG entegrasyonu

Örnek Uygulama:

```
import org.testng.annotations.*;

public class LoginTest {

    @BeforeClass
    public void setup() {
        System.out.println("Tarayıcı başlatıldı");
    }

    @Test(priority = 1)
    public void openLoginPage() {
        System.out.println("Login sayfası açıldı");
    }

    @Test(priority = 2, dependsOnMethods = {"openLoginPage"})
    public void performLogin() {
        System.out.println("Kullanıcı giriş yaptı");
    }

    @DataProvider(name="userData")
    public Object[][] provideData(){
        return new Object[][] {{"admin","12345"}, {"user","abcde"}};
    }
```

```
    @Test(dataProvider = "userData")
    public void testLogin(String username, String password){
        System.out.println("Kullanıcı: " + username + " | Şifre: " + password);
    }

    @AfterClass
    public void tearDown() {
        System.out.println("Tarayıcı kapatıldı");
    }
}
```

Deney Adımları:

1. IDE'nizde yeni bir Maven projesi oluşturun ve `testng` bağımlılığını `pom.xml` dosyasına ekleyin.
2. Yukarıdaki kodu oluşturun ve her anotasyonun çalıştırılma sırasını gözlemleyin.
3. Aynı sınıf içinde birden fazla test metodu tanımlayarak `priority` ve `dependsOnMethods` etkilerini inceleyin.
4. `@DataProvider` ile dinamik veri geçişi yapın.
5. `testng.xml` dosyası oluşturun ve testleri gruplar halinde çalıştırın.
6. Paralel test çalıştırmayı (`parallel="methods"`) etkinleştirin.
7. Çalıştırma sonrası oluşan HTML raporlarını inceleyin.

Örnek Sorular:

1. TestNG, JUnit'e göre hangi ek avantajları sağlar?
2. `@DataProvider` anotasyonu ne işe yarar? Ne tür senaryolarda kullanılır?
3. `dependsOnMethods` ile test bağımlılığı nasıl yönetilir?

Ek Görev: Jenkins veya GitHub Actions ortamında bir TestNG projesini CI/CD sürecine entegre ederek otomatik test tetiklemesi gerçekleştirin.

7. Hafta TestNG İleri Özellikler

Amaç: TestNG'nin ileri seviye özelliklerini kullanarak testlerin yönetilebilirliğini artırmak, gruplama ve bağımlılık senaryolarını kurgulamak.

Kapsam: Test grupları, öncelik sıralaması (priority), test bağımlılıkları (dependsOnMethods), parameterizasyon, veri sağlayıcılar (DataProvider).

Kod Örneği:

```
@Test(groups = {"login"}, priority = 1)
public void kullaniciGiris() {
    System.out.println("Kullan c ␣giri ␣yapt ");
}

@Test(groups = {"dashboard"}, priority = 2, dependsOnMethods = "
   kullaniciGiris")
public void panelGorunurluguTesti() {
    System.out.println("Dashboard␣kontrol␣edildi");
}

@DataProvider(name = "veriSaglayici")
public Object[][] veriSeti() {
    return new Object[][] {
        {"admin", "1234"},
        {"test", "4567"}
    };
}

@Test(dataProvider = "veriSaglayici")
public void parametreliGirisTesti(String kullanici, String sifre)
    {
    System.out.println("Giri ␣yap lan␣kullan c :␣" +
       kullanici);
}
```

Sorular:

1. Test gruplarını build sistemi (Maven/Gradle) ile entegre ederek sadece belirli testlerin çalışmasını nasıl sağlarsınız?
2. DataProvider yapısında runtime'da veriyi dış dosyadan (CSV/JSON) almak için hangi ek Java sınıfları gerekir?
3. dependsOnMethods özelliği başarısız bir testte ne tür etkiler doğurur? Exception handling nasıl optimize edilir?

8. Hafta Page Object Model (POM)

Amaç: Web otomasyon projelerinde sürdürülebilirliği artırmak için POM tasarım desenini uygulamak ve kod tekrarını minimize etmek.

Kapsam: Page class, test class ayrımı, reusable metotlar, sayfa nesnesi oluşturma, locator soyutlaması.

Kod Örneği:

```
// LoginPage.java
public class LoginPage {
    WebDriver driver;

    @FindBy(id = "username")
    WebElement txtUsername;

    @FindBy(id = "password")
    WebElement txtPassword;

    @FindBy(id = "login")
    WebElement btnLogin;

    public LoginPage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }

    public void login(String user, String pass) {
        txtUsername.sendKeys(user);
        txtPassword.sendKeys(pass);
        btnLogin.click();
    }
}

// LoginTest.java
@Test
public void dogruGirisTesti() {
    LoginPage loginPage = new LoginPage(driver);
    loginPage.login("admin", "1234");
    Assert.assertTrue(driver.getTitle().contains("Panel"));
}
```

Sorular:

1. POM'da page class'ların reusable olmasını sağlayacak mimari tasarım ilkeleri nelerdir?

2. Eğer UI bileşenleri sürekli değişiyorsa locator stratejisi nasıl modüler hale getirilebilir?

3. Page sınıflarında wait işlemlerinin soyutlanması ne tür avantajlar sağlar?

9. Hafta PageFactory ile Element Yönetimi

Amaç: Web elementlerinin tanımını sadeleştirmek, init işlemlerini merkezi hale getirerek kod okunabilirliğini ve yönetilebilirliğini artırmak.

Kapsam: PageFactory.initElements, @FindBy anotasyonu, lazy loading, element caching.

Kod Örneği:

```
public class DashboardPage {
    WebDriver driver;

    @FindBy(xpath = "//div[@class='welcome']")
    WebElement welcomeMessage;

    public DashboardPage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }

    public boolean mesajGorunurMu() {
        return welcomeMessage.isDisplayed();
    }
}
```

Sorular:

1. PageFactory kullanımı ile constructor üzerinden yapılan element yükleme arasında ne gibi performans ve bakım farkları oluşur?
2. Lazy initialization kullanılan elementler DOM'dan silinirse ne gibi exception oluşur? Bu nasıl yakalanır?
3. PageFactory sınıfında kullanılan annotation'lar harici frameworklerle (Allure, Spring, vs.) nasıl entegre edilebilir?

10. Hafta Davranışa Dayalı Geliştirme (BDD) ve Cucumber'a Giriş

Amaç: BDD (Behavior Driven Development) yaklaşımı ile teknik ve teknik olmayan paydaşların ortak anlayış geliştirebildiği otomasyon senaryoları yazmayı öğrenmek.

Kapsam: Gherkin dili, Feature dosyası, Step Definition sınıfı, Given-When-Then yapısı, Feature-Selenium entegrasyonu.

Kod Örneği:

```
Feature: Kullanici Giris
  Scenario: Gecerli kullanici bilgileriyle giris yapilmasi
    Given kullanici giris sayfasindadir
    When kullanici "admin" kullanici adi ve "1234" sifresi ile
       giris yapar
    Then kullanici yonetim panelini gorur
```

```
// StepDefinition
@When("kullanici␣{string}␣kullanici␣adi␣ve␣{string}␣sifresi␣ile␣
   giris␣yapar")
public void login(String username, String password) {
    loginPage.login(username, password);
}
```

Sorular:

1. Gherkin dili ile yazılmış bir senaryonun farklı platformlarda çalışabilmesi için dikkat edilmesi gereken standartlar nelerdir?

2. Step Definition sınıfı içinde reusable method yazımının riskleri ve önerilen desenler nelerdir?

3. Feature dosyalarının modülerleştirilmesi neden önemlidir? Büyük projelerde hangi stratejiler izlenmelidir?

11. Hafta Cucumber İleri Konular: Scenario Outline, Hooks, Tags

Amaç: Cucumber'ın veri kontrollü testler (data-driven), yaşam döngüsü yönetimi ve test filtreleme özelliklerini uygulamak.

Kapsam: Scenario Outline, Examples ile test verisi geçme, @Before/@After Hooks, test etiketleme (@tag), Runner konfigürasyonu.

Kod Örneği:

```
Scenario Outline: Hatali kullanici girisleri
  Given kullanici login sayfasindadir
  When kullanici <kullaniciAdi> ve <sifre> bilgilerini girer
  Then "Hatali giris" uyarisi gosterilir

Examples:
  | kullaniciAdi | sifre  |
  | admin        | wrong  |
  | guest        | 0000   |
```

```
// Hooks.java
@Before
public void tarayiciBaslat() {
    driver = new ChromeDriver();
}

@After
public void ekranGoruntusuAl(Scenario scenario) {
    if (scenario.isFailed()) {
        TakesScreenshot ts = (TakesScreenshot) driver;
        File src = ts.getScreenshotAs(OutputType.FILE);
        // dosya kopyalama islemi
    }
    driver.quit();
}

}
```

Sorular:

1. Scenario Outline kullanımı hangi tip testlerde hataya yol açabilir? Validasyon eksikliklerine örnek veriniz.

2. Hooks sınıfında senaryo başarısız olduğunda yapılacak işlemler hangi yapılarla koşula bağlanabilir?

3. Tag yapısıyla testlerin kategorilere ayrılması CI/CD sürecinde nasıl avantaj sağlar?

12. Hafta Kalite Güvencesinde Test Raporlama ve Loglama

Amaç: Otomasyon test sonuçlarını analiz edilebilir formatta sunmak, loglama ile hata kök nedenlerini takip edebilmek.

Kapsam: Allure Report, Extent Report, ekran görüntüsü alma, log4j/logback entegrasyonu, test çıktısı saklama.

Kod Örneği:

```
@AfterMethod
public void testSonrasi(ITestResult sonuc) {
  if (sonuc.getStatus() == ITestResult.FAILURE) {
    File ekran = ((TakesScreenshot) driver).getScreenshotAs(
       OutputType.FILE);
    // ekran goruntusunu kaydet
  }
  driver.quit();
}

// log4j.properties
log4j.rootLogger=INFO, FILE
log4j.appender.FILE.File=logs/test.log
```

Sorular:

1. Raporlama araçlarının CI/CD ile entegrasyonunda dikkat edilmesi gereken dosya formatları nelerdir?
2. Otomatik ekran görüntüsü alma işlemi hem rapora hem de dosya sistemine nasıl eşzamanlı kaydedilir?
3. Log seviyesi (info/debug/error) seçimi hangi kriterlere göre belirlenmelidir?

13. Hafta Güvenlik Testlerine Giriş (Pentest)

Amaç: Web uygulamalarında en sık rastlanan güvenlik açıklarını tespit etmek ve temel pentest araçlarını uygulamak.

Kapsam: OWASP Top 10, SQL Injection, XSS, CSRF, ZAP Proxy, Burp Suite, manuel ve otomatik testler.

Kod Örneği:

```
Proxy proxy = new Proxy();
proxy.setHttpProxy("localhost:8080");
ChromeOptions options = new ChromeOptions();
options.setProxy(proxy);
WebDriver driver = new ChromeDriver(options);

// sayfada basit bir SQL injection ornegi
WebElement input = driver.findElement(By.id("user"));
input.sendKeys("' OR 1=1 --");
```

Sorular:

1. SQL Injection testi sırasında beyaz kutu ve kara kutu yaklaşımları nasıl ayrışır?
2. ZAP proxy ile güvenlik testi yapılırken DOM tabanlı XSS nasıl tespit edilir?
3. Pentest çıktılarının yazılım geliştirme ekiplerine iletilmesi sürecinde en kritik metrikler nelerdir?

14. Hafta Genel Değerlendirme ve Final Uygulaması

Amaç: Tüm modülleri entegre eden kapsamlı bir test projesi geliştirerek uçtan uca otomasyon yapısı kurmak.

Kapsam: Selenium + TestNG + Cucumber + POM entegrasyonu, ekran görüntüsü, loglama, Allure raporları, Git sürüm kontrolü, CI/CD entegrasyonu.

Kod Örneği:

```
public class BaseTest {
  protected WebDriver driver;

  @BeforeClass
  public void setup() {
    driver = new ChromeDriver();
    PageFactory.initElements(driver, this);
  }

  @AfterClass
  public void teardown() {
    driver.quit();
  }
}

@RunWith(Cucumber.class)
@CucumberOptions(
  features = "src/test/resources/features",
  glue = "stepdefinitions",
  plugin = {"pretty", "html:target/report.html", "json:target/
     report.json"},
  tags = "@kritik"
)
public class TestRunner {}
```

Sorular:

1. Tüm test katmanlarını içeren entegre bir mimari tasarımı şematik olarak çiziniz.
2. Jenkins veya GitHub Actions ile test entegrasyonu kurulurken çevresel değişkenler nasıl yönetilir?
3. Geliştirici ekip ile otomasyon test ekibi arasında sürekli entegrasyon süreci nasıl organize edilir?