T.C.

EGE ÜNİVERSİTESİ

Fen Bilimleri Enstitüsü

YAZILIM GELİŞTİRME VE TEST SÜREÇLERİ HAKKINDA İNCELEME

Yüksek Lisans Tezi

Gizem İREN

Bilgisayar Mühendisliği Anabilim Dalı

İzmir

2020

T.C.

EGE ÜNİVERSİTESİ

Fen Bilimleri Enstitüsü

YAZILIM GELİŞTİRME VE TEST SÜREÇLERİ HAKKINDA İNCELEME

Gizem İREN

Danışman : Prof. Dr. Aylin KANTARCI

Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Yüksek Lisans Programı

İzmir
2020

Gizem İREN tarafından Yüksek Lisans tezi olarak sunulan “Yazılım Geliştirme ve Test Süreçleri Hakkında İnceleme” başlıklı bu çalışma EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliği ile EÜ Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi’nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve .....tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:		**İmza**
Jüri Başkanı	**: .....**	**.....**
Raportör Üye	**: .....**	**.....**
Üye	**: .....**	**.....**

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Yüksek Lisans Tezi olarak sunduğum "Yazılım Geliştirme ve Test Süreçleri Hakkında İnceleme" başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

15/06/2020

İmzası

Gizem İREN

ÖZET

YAZILIM GELİŞTİRME VE TEST SÜREÇLERİ HAKKINDA İNCELEME

İREN, Gizem

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Anabilim Dalı
Tez Danışmanı: Prof. Dr. Aylin Kantarcı
Haziran 2020, 55 sayfa

Yazılım testi yazılım mühendisliğinde tasarım ve uygulama, bakım, süreç ve yönetim içerikleri gibi pek çok teknik ve teknik olmayan alanı kapsayan geniş bir alandır. Yazılım testi süreçlerinin hem kurumların hem de müşterilerin beklenti ve ihtiyaçlarının doğru bir şekilde karşılanması için önemli bir rol oynadığını söyleyebiliriz. Yazılım testleri, geliştirilen yazılımın müşteriler tarafından talep edilen ihtiyaçları karşılayıp karşılamadığını anlamak amacıyla yapılmaktadır. Testler sayesinde yazılımda var olan hatalar tespit edilip düzeltilebilir ve müşteri gereksinimlerine uygun hale getirilmektedir.

Bu tezde, yazılım kalite ve testlerinin öneminden yola çıkılarak test yöntemleri ve araçları incelenmiştir. Öncelikle literatür araştırması ile günümüzde var olan yazılım test yöntemleri ve bu konu ile ilgili yapılmış çalışmalar incelenmiş ardından popüler yazılım test araçları ile yazılım testlerinin nasıl yapıldığına dair bilgi verilmiştir. Bu kapsamda tez sürecinde yapılan test uygulamaları web servis testleri, fonksiyonel ve veri tabanlı testler olarak üç parçaya ayrılmaktadır. Web servis testleri Rest Assured, fonksiyonel testler Selenium ve veri tabanlı testler ise Selenium ve Apache POI araçları ile gerçekleştirilmiştir.

Anahtar sözcükler: Yazılım Test, Fonksiyonel Test, Web Servis Testi, Veri Tabanlı Test, Rest Assured, Selenium, Apache POI

ABSTRACT

REVIEW OF SOFTWARE DEVELOPMENT AND TESTING PROCESSES

İREN, Gizem

MSc in Computer Eng.
Supervisor: Prof. Dr. Aylin KANTARCI
Jıne 2020, 55 pages

Software testing is a broad field covering many technical and non technical areas such as design, implementation, maintenance, process and management. We can say that software testing processes play an important role in fullfilling the expectations and requirements of both institutions and customers. Software tests are carried out to see if the developed software meets the needs demanded by the customers. With the help of these tests, the existing defects in the software can be detected and corrected according the customer requirements.

In this thesis, test methods and tools are examined based on the importance of software quality. Firstly, the current software test methods and the studies on this subject were examined with the literatüre research and then information was given on how to perform software tests with contemporary software testing tools. In this context, test applications carried out in the thesis are divided into three parts as web service, functional and data driven tests. While web service tests were performed with Rest Assured, functional tests were operated with Selenium. Finally, Selenium and Apache POI were used when executing the data driven tests

Keywords: Software Test, Functional Test, Web Service Test, Data Driven Test, Rest Assured, Selenium, Apache POI

ÖNSÖZ

Kullanıcı ihtiyaçlarına bağlı olarak geliştirilen çok amaçlı yazılımların artmasıyla yazılım kalite ve sınama konuları da ön plana çıkmaya başlamıştır. Bu kapsamda yazılımın son kullanıcıya ulaştırılmadan önce detaylı olarak test edilmesi ve hatalardan arınmış olması büyük önem arz etmektedir. Bunu sağlamak amacıyla çeşitli yazılım kalite test yöntemleri ve araçları ortaya çıkmıştır. Tez içeriğinin oluşturulmasında bu konular büyük rol oynamaktadır. Tez boyunca yazılım yaşam döngüsünün en önemli evresinde uygulanan olası teknikler öncelikle araştırılmış olup bu tekniklerin güncel olarak hangi araçlarla nasıl gerçekleştirildiği hakkında uygulamalı olarak bilgi verilmiştir. Bu çalışmanın özellikle yazılım test konusuyla ilgilenenler için önemli bir kaynak olabileceğine inanıyor ve tezimi bu noktada sonuçlandırıyorum.

İZMİR

15/06/2020

Gizem İREN

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam)

Sayfa

İÇİNDEKİLER (devam)

Sayfa

ŞEKİLLER DİZİNİ

Şekil Sayfa

ŞEKİLLER DİZİNİ (devam)

ŞEKİLLER DİZİNİ (devam)

ŞEKİLLER DİZİNİ (devam)

TABLOLAR DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

Kısaltmalar

4GL	Dördüncü nesil diller
IDE	Integrated Development Environment
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
API	Application Programming Interface
RC	Remote Control
UFT	Unified Functional Testing
HTML	Hyper Text Markup Language
URI	Uniform Resource Identifier
XML	Extensible Markup Language
REST	Representational State Transfer
JSON	Javascript Object Notation

1. GİRİŞ

Günümüzde dünya çapındaki şirketlerin birçoğu için yazılım testi en önemli alanlardan biri haline gelmektedir. Yazılım testi yazılım mühendisliğinde şartname, tasarım ve uygulama, bakım, süreç ve yönetim içerikleri gibi birçok teknik ve teknik olmayan alanı kapsayan geniş bir alandır. Yazılım testi süreçlerinin hem kurumların hem de müşterilerin beklenti ve ihtiyaçlarının doğru bir şekilde karşılanması için hayati bir önem arz etmekte olduğunu söyleyebiliriz.

Dijitalleşen iş dünyasında, yazılım süreçlerine karşılık gelen maliyetler ve potansiyel güvenlik tehlikeleri düşünüldüğünde yazılım testleri kapsamında hata tespitleri ve hatalara yönelik çözüm uygulamaları şirketler için oldukça önemlidir (Saini and Rai, 2013). Uygulama sürecini hızlandıran dördüncü nesil dillerin (4GL) geliştirilmesiyle teste ayrılan zaman giderek artmıştır. Dolayısıyla, test sürecine dair uygulamaların bilgisayar bilimi içinde gelecekte çok daha önemli bir hale gelmesi muhtemeldir (Luo, 2001).

Çağımızda teknolojik gelişmelerin çok hızlı ilerleyişi web dünyasını da kaçınılmaz bir şekilde etkilemiştir. Global istatistiklere baktığımızda 10 yıl önceye göre 2 katı aşkın daha fazla internet kullanıcısı bulunmaktadır (Statista, 2019). Internet kullanıcılarının artışı bir kenara, kullanılan ve hali hazırda tasarlanan web sitelerinin sayısı da oldukça çarpıcı rakamlara ulaşmış durumdadır. Araştırmalar göstermektedir ki kullanıcıların web sitelerine olan ilgileri mevcut durumda 3-4 saniye içerisinde önemli ölçüde düşmektedir. Bunun potansiyel sebepleri ise kötü tasarım, karmaşık yapı ya da bilgi eksikliği olmaktadır (Quan, 2018). Sonuç olarak, web sitelerinin hızlı, güvenilir ve doğru çalışması konuları gittikçe önem kazanmaktadır. Tarayıcılardaki bilgi ve veri akışı düşünüldüğünde test sürecinin önceliğinin açık olduğu kesin bir mesele olarak karşımıza çıkmaktadır. Genel anlamda, zaman, para ve çaba maliyetlerini göz önüne aldığımızda ise yazılım otomasyon testlerinin önemini anlamak zor olmayacaktır.

Yazılım testlerindeki en temel amaç yazılımın optimize edilmesine yönelik eksikliklere ve ilgili çözümlere odaklanarak yazılımın kalitesini ölçmektir (Mahajan, 2016). Başka bir deyişle, yazılım testleri, temelde gerçek sonuçlarla beklenen sonuçların uyumluluğunu kontrol edip gerekli olduğu durumlarda yazılımdaki hatalara yönelik çözümler üretmeyi amaçlamaktadır. Beklenen sonuçlar gereksinim belirtiminde tanımlanan, başka bir deyişle daha önceden belirlenen parametrelere dayanmaktadır. Gerçek sonuçlar ise kontrollü test ortamında

test sürecinden sonra elde edilen gerçek parametrelere dayanmaktadır. Genel olarak tamamlanmış bir yazılım testi bir yazılım ürününün bütün ömrünü kapsayacak nitelikte olmalıdır (Saini and Rai, 2013).

Tezin 2. Bölümünde sırasıyla yazılım test tekniklerinin tarihçesinden başlayarak, yazılım yaşam döngüsü, temel prensipler, test modelleri, otomasyon testleri ve bu tezin uygulama aşamasında kullanılan Selenium detaylandırılacaktır. Tezin 3. Bölümünde ise sırasıyla Java Restful API ile ilgili bilgi verilip daha sonra Rest Assured test aracı ile web servis test uygulamaları yapılarak test sonuçları gösterilmiştir. Bir sonraki başlık altında Selenium otomasyon aracı ile bir web uygulamasının fonksiyonel testleri gerçekleştirilmiş olup test sonuçları gösterilmiştir. 3. Bölümün son başlığında ise hem Selenium hem de Apache POI araçları kullanılarak veri temelli testler yapılarak sonuçları hakkında bilgi verilmiştir. Tezin 4. Bölümünde Sonuç ve Öneriler'e yer verilmiştir.

2. GENEL BİLGİLER

2.1 Yazılım Test Tekniklerinin Tarihçesi

2.1.1 Test Tekniklerinin Gelişim Aşamaları

Yazılım testinin tanım ve hedeflerinin netleşmesi zaman içerisinde test teknikleri üzerine çalışmayı tetiklemiştir. Tekniklerin geliştirilmesi tabi ki dönemler içerisinde ve kümülatif bir yaklaşımla gerçekleşmiştir. Test tekniklerinin gelişim sürecine ilişkin beş temel dönem vardır denilebilir.

Gelişim sürecini anlamak adına aşağıda bu dönemlerin özellikleri özetlenmiştir (Luo, 2001).

- **1. Aşama: 1956'dan önce ''Hata Ayıklama Odaklı Dönem''**
  1950'lerde Turing, program testlerinde ilk olarak kabul edilen ünlü bir makale yayınladı. Makale temelde bir programın zekâ gösterdiğini nasıl bilebiliriz sorununu ele almaktadır. Diğer bir deyişle, bir programın gereksinimleri karşıladığını nasıl bilebiliriz sorusu makalenin ana temasını oluşturmaktadır. Bu dönemde hata ayıkla süreci ile test süreci oldukça bütünleşikti. Bu dönemde tanımlanan ve Turing diye bilinen operasyonel test bir referans sisteminin (bir insan) davranışının bir sorgulayıcı (testi gerçekleştiren kişi) için ayırt edilemez olmasını baz alıyordu. Günümüzde bilinen fonksiyonel testin en erken versiyonu

bu testtir denilebilir. Test, hata ayıklama vb. gibi kavramlar o dönem net olarak tanımlanabilen kavramlar değildi.

- **2. Aşama: 1957 – 1978 ‘’Gösteri Odaklı Dönem’’**

  Bir önceki dönemde program kontrolü adı verilen testler, hata ayıklama kavramından farklılaştı. 1957 yılında Charles Baker en temel iki hedeften bahsetti: programın çalıştığından emin olmak ve programın sorunu çözdüğünden emin olmak. 1970’lere gelindiğinde yazılım testleri üzerine kapsamlı araştırmaların yaygınlaştığını söyleyebiliriz.

- **3. Aşama: 1979-1982 ‘’Yıkıma Yönelik Dönem’’**

  1979 yılında Myers, daha etkili test tekniği tasarımı için temel oluşturan ‘’The Art of Software Testing’’ kitabını yazdı. Bu şekilde, ilke kez yazılım testi hata bulma amacıyla bir program uygulama süreci olarak tanımlanmış oldu. Test, bir programın hata içerdiğini göstermek istiyorsa seçilen test senaryolarının bunları algılama olasılığı yükselir ve test daha başarılı olur. Bu yaklaşım değişikliği aslında testlerin diğer doğrulama aktiviteleri ile erken ilişkilendirilmesine yol açmıştır.

- **4. Aşama: 1983-1987 ‘’Değerlendirme Odaklı Dönem’’**

  Uluslararası Standartlar Bürosu Bilgisayar Birimi Bilimleri ve Teknolojisi Enstitüsü tarafından 1983 yılında bilgisayar yazılımının yaşam döngüsü geçerliliği, doğrulanması ve test edilmesi süreçlerine dair bir kılavuz yayınlandı. Test süreçlerine dair hataların analizi ve değerlendirilmesi odaklı bu dönemde kaliteli yazılımların geliştirilmesi ve bakımının sağlanması konusunda oldukça önemli gelişmeler kaydedilmiştir.

- **5. Aşama: 1988’den günümüze ‘’Önleme Odaklı Dönem’’**

  Yazılım test tekniklerine dair en önemli kaynaklardan birinin sahibi olan Beizer, “test tasarlama eyleminin bilinen en etkili hata önleyicilerden biri olduğunu” ve test tanımına hata önleme kavramını ilave ettiğini belirtmiştir. Dolayısıyla, adından da anlaşılacağı üzere 1988’den günümüze yazılım test teknikleri adına daha çok önleme odaklı bir eğilim olduğu söylenebilir. Test sürecinin bugüne en yakın örneği testlerin ve test ortamlarının planlanması, tasarlanması, oluşturulması, sürdürülmesi ve yürütülmesi aktivitelerini içeren 1991 Hetzel tanımlamasına aittir.

Özetlemek gerekirse test teknikleri zaman içerisinde hatalardan yola çıkarak birikimler ve deneyimler doğrultusunda ilerledi. Hata ayıklama odağından önleme odağına gelen döneme kadar birçok bilim adamı ve bu alanda çalışan insanlar hem teorik hem de metodolojik sayısızca çalışma yürüttüler. Dolayısıyla teoriyi ve/veya metodolojiyi kapsayan zengin bir literatür havuzuna sahip olduk.

2.1.2 Teori ve Metot Bazlı Çalışmaların Önemli Sonuçları ve Kazanımlar

Genel olarak literatür çalışmalarına baktığımızda temelde iki kategori vardır: teorik ve metodolojik. Yazılım test tekniklerine üzerine yapılan çalışmaların önemli sonuçları ve kazanımlar Tablo 2.1'de özetlenmektedir.

Tablo 2.1. Zaman içerisinde yürütülen yazılım test teori ve metot çalışmalarının önemli sonuçları ve kazanımlar (Luo, 2011)

		Teori ve Metot	
Zaman	**Konsept**	**Fonksiyonel**	**Yapısal**
2000		2001: Bileşen tabanlı yazılım için entegre teknik	
		2000: UML tabanlı entegrasyon testi	
		1997: Olasılıksal fonksiyonel test	1997: Mimari açıklamaya dayalı entegrasyon testi
			1994: Güvenilirlik tahmini için kapsama dayalı model
	1991: Yazılım yaşam döngüsü boyunca entegrasyon testi		
1990		1990: Mantık tabanlı test	
		1989: Biçimsel ve uygulamalı testleri resmi yöntemlerle kullanma	
			1985: Veri akışı odaklı strateji
	1983: Hata önleyici testler		
1980		1980: Fonksiyonel test ve tasarım soyutlamaları	1980: Alan stratejileri (Domain)
	1979: Testin hata tespitine bağlanması		
			1976: Testin seçimine giden yol yaklaşımı

	1975: Test verisi seçiminin temel teorisi		1975: Test seçimine kenar yaklaşımı, prob yerleştirme
1970			
1960	1957: Hata ayıklamayı testten ayırma süreci		
1950	1950: Bir programı öğrenmek için yapılan testlerin gereksinimleri karşılaması anlayışı		

2.1.3 Yazılım Mühendisliği Alanındaki Tarihsel Çerçeve, Gerçekleşen Gelişmeler ve Değişimler

Yazılım mühendisliğine ait ilk tanım 1969 yılında Fritz Bauer tarafından ‘’Gerçek makineler üzerinde etkin ve güvenilir çalışan ekonomik yazılımlar geliştirilmesi amacıyla mühendislik ilkelerinin kullanılmasıdır.’’ şeklinde tanımlanmıştır (Kuday, 2014). Yazılım testlerinin gelişim süreci, yazılım mühendisliği alanındaki faaliyetlerden ayrı düşünülemez. Tablo 2.2’de yazılım mühendisliği alanında önemli değişimler özetlenmektedir.

Tablo 2.2. Yazılım mühendisliği alanındaki değişimlerin ve gelişmelerin tarihsel çerçevede özeti (Luo, 2001).

Zaman Aralığı	**Odak Konular**
1960 +/- 5	Anımsatıcılar, düz yazıların kullanımı Küçük programlara yönelim Yapıyı temsilen sembolik bilgiler Kontrol akışının temel olarak anlaşılması
1970 +/- 5	Basit düzeyde girdi ve çıktı özellikleri Algoritmalara vurgu Veri yapıları ve türleri Programların 1 kez uygulanması ve sonlandırılması
1980 +/- 5	Karmaşık özelliklere sahip sistemler Sistem entegrasyonuna vurgu, yönetim Uzun ömürlü veritabanları Program derlemelerinin sürekli uygulanması
1990 +	Donanımla entegre yazılım Yönetim sürecinin iyileştirilmesine ve sistem yapısına vurgu Sistem tasarımı için soyutlamalar Yüksek düzeyde etkileşimli sistemler, multimedya

Odak noktalarının kümülatif birikimine en büyük katkıyı sağlayan gelişmeler 1965-1980 yılları arasında üniversitelerin ve iş alanlarının güçlü bilgisayar merkezleri kurma girişimleri ve yazılım evlerinin emeklemeye başlaması olmuştur. 1970'lerle birlikte veri yapısı ve türlerinin incelenmesi, algoritmalara yapılan vurgular ve programların denenmesi gibi temel aşamalar uygulamaların öngörülen sürelerde tamamlanamaması ve maddi zararlara sebep olmuştur. 1968 yılında Almanya'da NATO Bilim Kurulunun desteklediği Münih Teknik Üniversitesi ev sahipliğinde ''Büyük Yazılım Projesi'' geliştirme ve yöntemleri tartışılmış ve sonuçta ''Yazılım Mühendisliği'' kavramının bir disiplin olarak ortaya çıkışının miladı olmuştur (Kılan, 2015). Bu tarihten itibaren, globalleşen dünyada kurumların değişimi, mevcut sistemlerin trendlere ve değişen dünyaya ayak uydurmak için adeta yeniden yapılandırılması gibi birçok parametrenin etkisiyle yazılım mühendisliği içeriği ve kapsamı her geçen gün evrilerek değişmeye devam ettiğini ve edeceğini söyleyebiliriz.

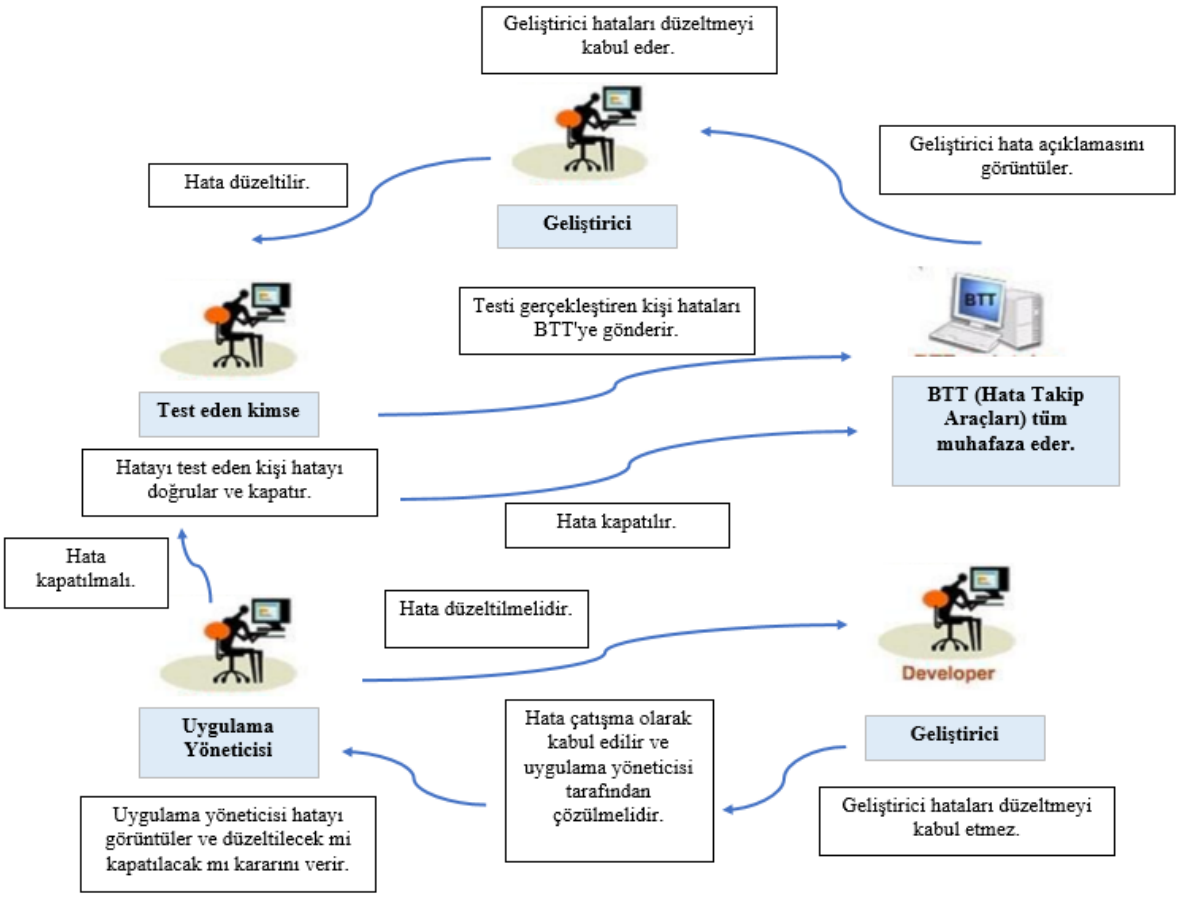
2.2 Hata Yaşam Döngüsü

2.2.1 Hata Yaşam Döngüsü Genel Akışı

Genel kapsamda belirtildiği üzere, hata konusu aslında yazılım testlerinin en temel ve en önemli parçası durumundadır. Bu nedenle yazılım hatası yaşam döngüsünü ve içeriğini anlamak tezin kapsamı açısından önemlidir.

Bir yazılım hatası, yazılım yapmayı amaçladığı şeyi yapmadığında ya da yapması amaçlanmayan bir şeyi yaptığında ortaya çıkar. Yazılım hatası, bilgisayar programında beklenmeyen bir hataya veya kusura neden olan bir kodlama sorunu olarak tanımlanabilir (Chintakayala, 2013). Başka bir deyişle, bir program amaçladığı gibi çalışmazsa, büyük olasılıkla bir hatadır. Hata yaşam döngüsü, yazılım hatasıyla başlar ve hata düzeltildiğinde sonlanır.

Genel anlamda hata yaşam döngüsünün işleyişi aşağıdaki şekilde özetlenmiştir.



Şekil 2.1. Hata Yaşam Döngüsü (Chintakayala, 2013).

2.2.2 Hata Tipleri ve Kapsamı

Genel anlamda yazılımdaki kusurlar hata (bug) olarak tanımlanmaktadır. Test ve/veya kalite mühendisleri tarafından yürütülen süreçlerde beklenmedik şekilde oluşan davranışlar hata olarak işaretlenir (Acun ve Bilgin, 2015). Yaşam döngüsüne dahil olan pek çok farklı tipte hata vardır. Bu hata tipleri ve kısaca içerikleri şu şekilde özetlenebilir (Chintakayala, 2006).

- Kullanıcı arayüz hataları: eksik veya yanlış işlemlerden ötürü kullanıcının beklediği işlemlerin gerçekleşmemesi, eksik veya yanıltıcı bilgi, uygunsuz hata mesajları, performans sorunları (zayıf yanıt, çıktı yönlendirilememesi veya tuş takımının uygunsuz kullanımı vb.)
- Sınırlarla ilgili hatalar: döngü, zaman, uzay ve bellek içerisindeki sınırlar, sınırlar dışında kalan vakaların yanlış kullanımı

- Hesaplama hataları: kötü mantık, aritmetik, veri üzerindeki dönüşümler, yanlış formül ve yaklaşım gibi nedenler

- Kontrol akışı hataları: yanlış geri dönüş durumu, istisnai durumlar, hatalı veya yanlış sonuçlar, veri türü hataları ve kontrol akışına dahil olan aktivitelerin beklenen performansı gösterememesi gibi hatalar
- Veri işleme veya yorumlamadaki hatalar: sonlandırılmamış boş diziler, bir hata çıktıktan veya kullanıcı iptal edildikten sonra hatalı dosya üzerine yazılmaya devam edilmesi
- Donanım: yanlış veya kullanılamayan aygıt, aygıt zekâsının yetersiz kullanımı, yanlış işlem ve talimat kodları
- Test hataları: bozuk veri dosyaları, yanlış yorumlanmış özellikler ve belgeler, sorunun nasıl çözüleceğini açıkça ortaya koymamak, bir problemi fark etmemek veya bildirmemek, düzeltmelerin doğrulanamaması, özet raporun verilememesi

Yaşam döngüsü içinde aynı anda birden fazla hata tipi ortaya çıkabilir, bir tip hatanın varlığı farklı tip bir hatayı tetikleyebilir. Bu noktada, potansiyel olarak ilişkili olabilecek hataların tespiti, çözüm aksiyonları tekrarlı hataların ortaya çıkmasını önlemede oldukça önemlidir. Hata veritabanı madenciliği diye adlandırılan kapsamda bu hataların analizi, öngörülmesine yönelik çalışmalar ve test süreciyle alakalı aktiviteler vardır. Özellikle hata düzeltme sürecinin iyi yönetilmesi, tekrarlı hataların incelenmesi, raporlanmış bir hataya doğru şiddeti (severity) atamak oldukça kritik bir öneme sahiptir (Acun ve Bilgin, 2015). Çünkü yazılım testinden beklenen çıktılar ve çıktıların kalitesi büyük oranda hata tipleri, ortaya çıkma sıklıklarına ve şiddetlerine bağlı olarak şekillenmektedir. Dolayısıyla, yaşam döngüsü içinde hataya dokunan her aktivite aslında genel performansı potansiyel olarak etkileyen kritik öneme sahip aktivitelerdir diyebiliriz.

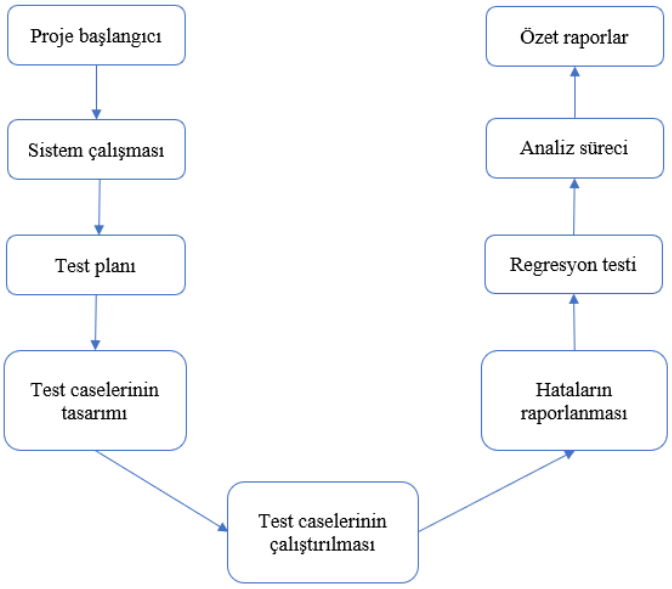
2.3 Yazılım Testi Yaşam Döngüsü

2.3.1 Yazılım Testi Yaşam Döngüsü Aktiviteleri

Yazılım testi süreci, yazılım geliştirme yaşam döngüsünde yaklaşık toplam zamanın %50'sini oluşturur (Devi et al., 2017). Bütçe açısından bakıldığında test süreci maliyeti toplam bütçenin yaklaşık olarak %40'ını kapsar (Tuteja and Dubey, 2012). Test sürecinin zamanı kullanılan algoritmaya, programlama diline, kaç satır kod olduğuna, fonksiyon noktalarına, harici ve dahili arayüzlere bağlı olarak değişmektedir.

Genel olarak yazılım testleri yaşam döngüsü içerisinde yaşayan ve sisteme her noktada dokunan yapılar şeklinde geliştirilmelidir. Başka bir deyişle, yazılım testleri sistem geliştirmede ayrı bir aşama olmamalıdır, tasarım geliştirme ve bakım aşamalarında uygulanabilir olmalıdır (Chaurasia and Bahl, 2018).

Yazılım testi yaşam döngüsü Şekil 2.2'deki gibi özetlenebilir.



Şekil 2.2. Yazılım Testi Yaşam Döngüsü Aktiviteleri (Chaurasia and Bahl, 2018)

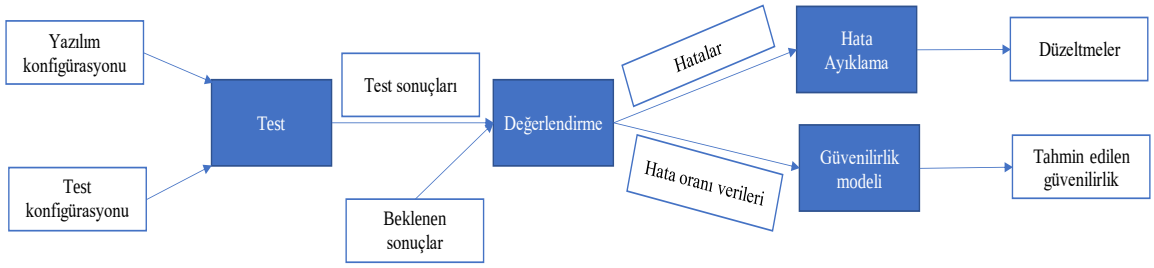
Projenin başlangıcıyla başlayan süreç test planının tanımlanması, test senaryolarının oluşturulması ve gerçekleştirilmesi, raporlama ve analiz aktiviteleri şeklinde devam etmektedir.

Planlama aşaması üst düzey test planları, kalite hedeflerinin planlanması, raporlama prosedürleri, problemlerin sınıflandırılması, kabul kriterlerinin belirlenmesi, proje ölçümleri ve proje test sürecinin zamanlamasının planlanması gibi aktiviteleri içerir (Chintakayala, 2006). Analiz süreci ise iş gereksinimlerine göre fonksiyonel doğrulama, test senaryolarının yazılması, zaman tahminleri ve öncelik atamaları ile senaryo formatı, test döngülerinin matrisler ve zaman çizelgeleri ile geliştirilmesi, eğer mevcutsa otomatize edilebilecek test senaryolarının tanımlanması, yedekleme, geri yükleme ve doğrulama dokümanlarının prosedürlerinin belirlenmesi gibi adımları içerir.

Öncelik kapsamı konusu yazılım testleri süreçlerinde oldukça önemli bir konudur. Tam anlamıyla kapsamlı bir test yapmak imkânsızdır denilebilir. Giriş izinleri, geçersiz girişler ve işlevsel olmayan gereksinimler önceliklere bağlı olarak test süreçlerine dahil olan konulardır. Denge konusu yazılım süreçlerinde önemli olan başka bir konudur. Şöyle ki, yazılı olarak belirlenmiş gereklilikler, gerçek dünyada var olan teknik kısıtlamalar ve kullanıcı beklentileri yazılım testi süreçlerinde dengelenebiliyor olmalıdır. Test sonuçları test cihazından bağımsız olmalıdır ki test süreci ve sonuçları tutarlı ve tarafsız olarak kabul edilebilsin. Sonuçlar yine test cihazından bağımsız olarak tekrarlanabilir olmalıdır.

Test sürecindeki her bir adımın ve her sonucun belgelenmesi testin izlenilebilir olmasına katkı sağlayacaktır (Ghuman, 2014).

Yaşam döngüsü içinde aktiviteler baz alındığında bir bilgi akışı süreci mevcuttur. Şekil 2.3'de test sürecinin bilgi akışı özetlenmiştir.



Şekil 2.3. Test Süreci Bilgi Akışı (Ghuman, 2014)

Yazılım ve test konfigürasyonlarının rehberliğinde başlayan test süreci, gerçekte alınan sonuçların ve beklenen sonuçların karşılaştırılması şeklinde bir değerlendirme sürecine gider. Değerlendirme sonucunda hatalar ayıklanır ve ilgili düzeltmeler için çalışmalar yapılır. Hata oranı verileri ise güvenilirlik modeline gider ve yazılımın belirli alt süreçlerinin ya da aktivitelerinin konu olduğu modelde güvenilirlik düzeyi tahmin edilir (Luo, 2001).

2.3.2 Yazılım Testi Temel Prensipleri

Her yazılım testinde büyük ölçüde kullanılan temel prensipler vardır. Bu prensipler test sürecininin hem operasyonel, hem stratejik hem de yönetimsel açılardan daha sağlıklı yürütülmesine katkıda bulunur. Genel olarak temel prensipler aşağıdaki şekilde özetlenebilir (Saini and Rai, 2013).

- Bağımsız bir ekip test sürecini gerçekleştirmelidir.
- Test sürecine kalifiye personel ya da personeller dahil edilmelidir.
- Testte kesin ya da çok büyük ihtimalle hata bulunmayacağı varsayımları olmamalıdır.
- Beklenmeyen ve geçersiz olan girdilerle geçerli olan koşullar test edilmelidir.
- Test etme sürecinin yazılımdaki hataları bulmayı amaçladığı unutulmamalıdır.
- Test süreci esnasında yazılım statik olarak kalmalıdır.
- Test vakaları ve sonuçlar belgelenmelidir.

Yazılım testi sürecinin izlenilebilirliği, ölçülebilirliği, çıktıları ve testin kalitesi direkt olarak yukarıda bahsedilen temel prensiplerde tanımlanmış rol, görev ve prosedürlere bağlıdır.

2.3.3 Yazılım Testi Yaşam Döngüsü Test Tipleri

Yazılım testlerinde farklı tiplerde test prosedürleri ve uygulamaları yaygın olarak kullanılmaktadır. Farklı kırımlarla test sürecini gerçekleştirmek iş ihtiyaçlarına, mevcut duruma, testten beklenen çıktılara, süreç içerisindeki kısıtlayıcı faktörlere vb. bağlıdır. Dolayısıyla, farklı kırılımlar kullanılarak kategorize edilen birçok test tipi mevcuttur.

Test süreci kapsamındaki analizler statik ve dinamik analizler olmak üzere iki kategoride incelenebilir (Luo, 2001). Statik analizler herhangi bir uygulama olmaksızın yazılım kalitesini belirlemek ve tahmin etmek üzere gerçekleştirilen aktiviteleri içerir. Bu aktiviteler kod denetimi, program analizi, sembolik analiz ve model kontrolünü içerir. Dinamik analizler ise uygulama yoluyla gerçek verilerin gerçek ve/veya simüle edilmiş koşullar altında yazılım kalitesinin belirlenmesine dair aktiviteleri içerir. Test prosedürlerinin kullanımı, otomasyon aktiviteleri, komutların uygulama yoluyla analizi gibi aktiviteler dinamik analizler kapsamındadır.

2.3.3.1 Birim, Entegrasyon ve Sistem Testleri

Birim, entegrasyon ve sistem testleri genel olarak aşağıdaki gibi özetlenebilir (Saini and Rai, 2013; Ghuman, 2014; Babbar, 2017; Hooda and Chhillar, 2015).

- Birim testi, yazılım süreçlerinde en küçük birimlerin doğrulanması ile ilgili uygulamaları içeren testtir. Yazılım geliştirme süreçlerinde ayrıntılı tasarım aşamasında birim testleri tanımlanır ve aynı anda birden fazla birim için testler yapılabilir.
- Entegrasyon testlerinin temel amacı yazılım yapısına dair doğrulamaları yapmaktır. Entegrasyon testleri için çeşitli yaklaşımlar mevcuttur: büyük patlama, yukarıdan aşağıya, aşağıdan yukarıya ve sandviç (karışık) yaklaşım. Entegrasyon testlerinin temel amacı sistemin farklı kısımlarının nasıl gruplandırıldığını veya birlikte çalıştığını test etmektir.
- Sistem testi, yazılım entegrasyonundan sonra genel olarak sistemin işlevini ve performansını test etmeye yönelik yüksek dereceli test setlerinin gerçekleştirilme süreçlerini içerir. Temel amaç, tüm sistem öğelerini yazılım gereksinimleri şartnamesine uygunluğunu doğrulamaktır.

Mocha, Tape ve Jasmine birim ve entegrasyon testleri için kullanılan popüler araçlardan bazılarıdır.

2.3.3.2 Kara ve Beyaz Kutu Testleri

Kara kutu ve beyaz kutu testi yazılım testlerinde en çok kullanılan yöntemler olarak sıralanabilir (Babbar, 2017; Luo, 2001; Hooda and Chhillar, 2015).

- Kara kutu testlerinde diğer bir adıyla fonksiyonel testlerde, veri öğelerinin yapısı ve detayları kullanıcısı tarafından bilinmez veya erişilebilir değildir. Yani test edilen yazılım programı ya da sistem bir kara kutu olarak algılanır. Bu tip testlerde, senaryoların seçimi tasarım spesifikasyonlarına ve ihtiyaçlara bağlı olarak gerçekleşir. Fonksiyonel test, yazılım varlığının dış davranışlarını vurgular. Test senaryoları giriş / çıkış değerinden üretilir veya tasarlanır, aynı zamanda tasarım/kod bilgisi gerekmez. Kara kutu testleri işlevsel olmayacağı gibi işlevsel de olabilir.
- Beyaz kutu testlerinde diğer bir adıyla yapısal testler, veri öğelerinin yapısı ve detayları kullanıcı tarafından bilinir ve erişilebilir. Yani test edilen yazılım programı ya da sistem bir beyaz kutu olarak algılanır. Uygulamaya dayalı olarak seçilen test senaryoları vardır. Spesifik alt kodların çalıştırılması ve performans ölçümü gibi durumlar yapısal testlerde sıkça rastlanan bir durumdur. Beklenen sonuçlar farklı tipte kriterler üzerinden değerlendirilir. Yapısal testler, yazılımın iç yapısına odaklanır. Test caseleri koda dayalı olarak gerçekleştirilir.

2.4 Yazılım Geliştirme Modelleri ve Testin Rolü

Günümüzde kullanılan çok çeşitli yazılım geliştirme yaklaşımları vardır ve bu yaklaşımlar ‘’Yazılım Geliştirme Süreç Modelleri’’ şeklinde de adlandırılmaktadır. Şelale, prototip, spiral, çevik, evrimsel geliştirme, artımlı ve V modeller şeklinde sıralanabilir. (Tuteja and Dubey, 2012; Kuday, 2014). Her bir yazılım geliştirme modeli için test aşaması önemli bir rol oynamaktadır. Yazılım testlerinin yapılma amacı sırasıyla müşteriye teslim etmeden önce ürünün kalitesini garantilemek, düzeltme maliyetini azaltmak, hataların sonraki aşamalara yayılmasını önlemek, müşterileri memnuniyetini ve ürün talebini daha da arttırmaktır.

Bu bölümde yazılım modelleri, bir diğer deyişle yazılım geliştirme süreç modelleri detaylandırılacaktır.

2.4.1 Şelale Modeli

Şelale modeli yazılım projelerinde sıklıkla kullanılan bir süreç modelidir (Kuday, 2014). Bu modelde belli başlı ardışık olarak ilerleyen temel aşamalar vardır. Bu aşamalar aşağıdaki gibi sıralanabilir (Tuteja and Dubey, 2012).

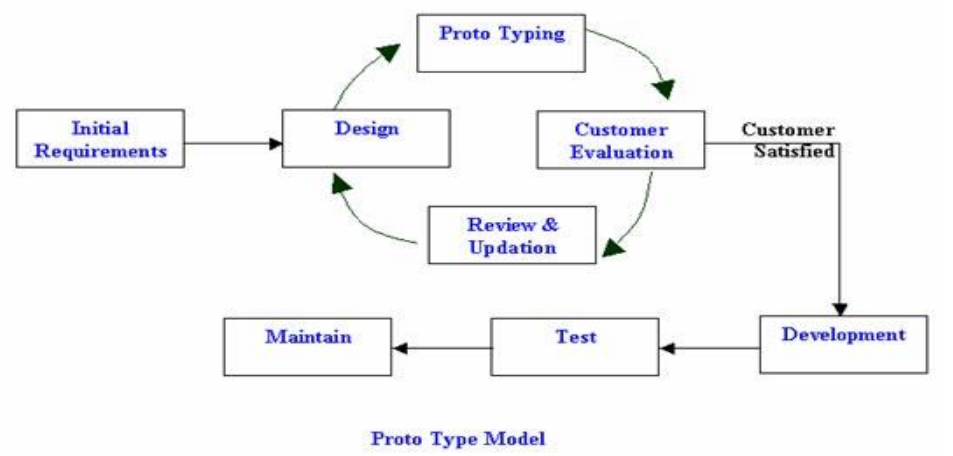
- Gereksinim özelliklerinin belirlenmesi
- Yazılım tasarımı

- Uygulama süreci
- Test süreci
- Sistemin işletilmesi aşaması ve bakım

Her aşama sırasıyla uygulandığından proje süresi daha kısa olan küçük projeler için daha uygundur. Bir aşamanın çıktısı bir sonraki aşamanın girdisini oluşturduğundan belli bir aşamada hata oluştuğunda o hata son aşamaya kadar taşınır. Bir başka deyişle ardışık ilerleyen bir yapı olduğu için ilerleyen aşamalarda karşılaşılan sorunlar için önceki aşamalara dönmek zordur, ek maliyet ve ek zaman gerektirir (Kuday, 2014).

2.4.2 Prototip Modeli

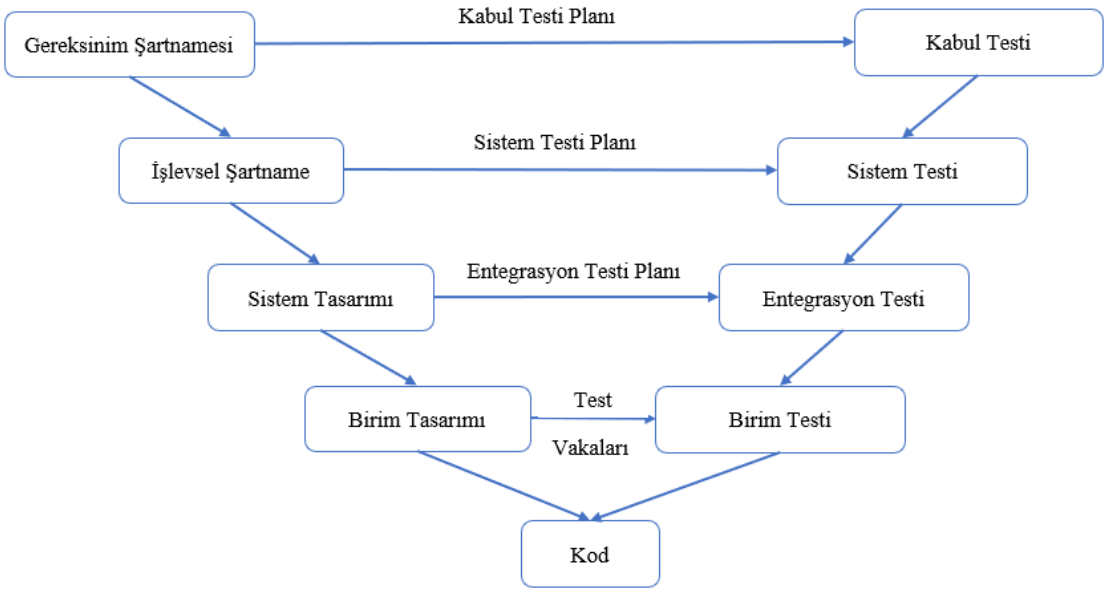
Prototip modelinde ise sistemin girdi, ihtiyaç ve çıktılarına dair bilgi bulunmadığı durumda sıkça kullanılır. Çünkü kullanıcıların iş gereksinimlerinin tam olarak belirli olmadığı durumlarda prototip oluşturmak işlerin hızlanması açısından avantajlı olacaktır (Kuday, 2014). Ancak, prototip oluşturma aşaması, kullanıcıların katılımını gerektirebilir. Yazılımı girdi, çıktı ve iş ihtiyaçlarının netleştirilmesi ve detaylandırılması hem geliştiricilerin hem de kullanıcıların gerçekleştirecekleri toplantılarda fikir birliğine varılmasına bağlıdır. Bu nedenle, kullanıcı geri bildirimlerine dayanarak daha az zaman ve maliyetle spesifikasyonlara uygun iyi bir prototip elde edilebilir (Tuteja and Dubey, 2012). Sadece prototipe odaklanmak bazen projenin tamamını doğru bir şekilde analiz etmeyi engelleyebilir. Bu modelde Şekil 2.4'den görüleceği üzere ürün tasarımı oluşturulduktan sonra ortaya bir prototip model çıkarılmakta ve müşterinin değerlendirilmesine sunulmaktadır. Eğer müşteri beklentileri ile prototip uyuşursa geliştirme aşamasına, uyuşmaz ise tekrardan düzeltme ve ardından tasarlama aşamalarına geçilmektedir (Timah, 2011).



Şekil 2.4. Prototip Modeli (Timah, 2011)

2.4.3 V Modeli

Yaygın olarak kullanılan modellerden biri de V modelidir. V modeli şelale modelinin genişletilmiş bir versiyonudur denilebilir (Kuday, 2014). Model en baştan yazılımın dağıtımına kadar olan süreçteki tüm operasyonları test etmeyi amaçlamaktadır (Mahajan, 2016). V modelinin yapısı Şekil 2.4'de özetlenmiştir.



Şekil 2.5. V Modeli Yapısı (Mahajan, 2016)

V modelinin en büyük avantajlarından biri sağ tarafta yer alan doğrulama odaklı aksiyonları tanımlayarak başlamasıdır. Bu durum projeye büyük kazanımlar sağlar (Kuday, 2014).

2.4.4 Çevik Modeller

Klasik modellerde olduğu gibi fazlaca belgeleme aktiviteleri ve ardışık gelen aşamaları tamamen alt üst eden çevik modeller 1990'ların ortasında ortaya çıkmıştır. Temelde yazılım geliştirme sürecinin hızlandırmayı amaçlayan çevik modeller için hız, devamlılık ve kullanışlı yazılımlar üretebilmek en önemli ilkelerdir (Kuday, 2014). Bu kapsamda scrum, çevik birleştirilmiş süreç, test güdümlü geliştirme gibi birçok farklı türde çevik model geliştirilme sürecine devam edilmektedir. Kısa dönemli planlar ve geliştirmeler ile yazılım geliştirmenin hem değişikliklere uyumu kolaylaştırdığı bir gerçektir.

Çevik modellerde genel olarak ihtiyaçların belirlenmesinin ardından süreç planlama tahmin aktiviteleri başlar. Bu aktivitelere bağlı olarak yüksek seviyede bir başlangıç planı oluşturulur. Yüksek seviye planın ışığında çevik modelin yaşam döngüsü başlar. Sırasıyla senaryoların oluşturulması, işlevsel ve kabul testleri, konumlandırma, kalite güncelleme ve son kontroller döngüler halinde yinelenerek ilerler. Son kontrol aşamasından sonra sistem test edilir ve hazır hale gelir (Kuday, 2014).

2.4.5 Spiral Modeli

Spiral yaşam döngüsü olarak da bilinen spiral model prototipleme ve şelale modellerinin özelliklerini birleştirir. Büyük ve daha karmaşık projeler için kullanımı uygundur. Oldukça özelleştirilmiş olduğundan yeniden kullanımı sınırlı olabilir, fakat çok daha gerçekçi bir bakış açısı sağlar (Tuteja and Dubey, 2012). Spiral modelde tekrarlanan dört temel aktivite vardır (Kuday, 2014). Bunlar aşağıdaki şekilde özetlenebilir.

- Kaynak ve zaman çizelgeleri göz önünde bulundurularak yapılacak planlama
- Teknik ve yönetimsel risklerim çözümlenmesi
- Yazılımın tasarım, kodlama ve hayata geçiş sürecindeki aktivitelerin gerçekleştirilmesi
- Kullanıcı geri bildirimlerinin alınması, yazılımın geçerliliğinin ve beklendiği gibi çalıştığının değerlendirilmesi

2.4.6 Evrimsel Geliştirme Modelleri

Araştırmacı geliştirme ve atılabilir prototipleme şeklinde iki ana kategoriye ayrılan evrimsel geliştirme modelleri yazılımın ilk versiyonun üzerinden sürekli iyileştirme yapmayı temel alır (Kuday, 2014). Araştırmacı geliştirmede yeni özellikler eklenerek devam eden bir aksiyon süreci söz konusudur. Atılabilir prototiplemede ise hedef yazılımı müşteri gözünden keşfetmek için prototipler halinde geliştirmeler yapılır. Dolayısıyla başlangıç, orta seviye ve final versiyonlar şeklinde farklı prototipler gözlemlenebilir.

2.4.7 Artımlı Model

Şelale modelinin tekrarlı olarak gerçekleştirilmesiyle belirli süreler dahilinde yazılımı sürümler halinde geliştirmeye dayanır. Her bir sürümde bir önceki sürüme ek işlevler ya da özellikler eklenmesini baz alır. En temel avantajı az sayıda geliştiriciyle işin yürütülmesi ve takibine olanak sağlamasıdır (Kuday, 2014).

2.5 Otomasyon Testleri Genel Kapsamı

Otomasyon test uzmanlarının temel amacı tekrarlanan işleri ve bu işlere harcanan zamanların minimuma indirmektir. Uzun vadede zaman, para ve kalite açısından oldukça

avantajlı olmakla birlikte iyi bir planlama süreci gerektirir (Kuday, 2014). Bunun yanı sıra insan kaynaklı hataların minimuma indirilmesine fırsat verdiğinden ve gelişen teknolojiyi var olan sisteme entegre etme durumlarından dolayı daha iyi yönetilebilir bir süreç elde edimi söz konusu olmaktadır (Kökten, 2019).

2.5.1 Genel Bileşenler

Otomasyon testlerinin geçmişine baktığımıza yaklaşık 25 seneyi aşan bir süreçten bahsedebiliriz. Genel anlamda otomasyon testleri çok sık tekrarlanan, fazla miktarda efor ve zaman harcamayı gerektiren ve kritik olan süreçler üzerinde sıklıkla kullanılmaktadır (Büyükyumukoğlu vd., 2017). Bu bağlamda yazılım yaşam döngüsünde otomasyon testleri aslında manuel testlerin tamamını kaldırmayı hedefleyen bir yaklaşım değildir. Otomasyon kapsamında genel bileşenler aşağıdaki şekilde özetlenebilir (Madan and Kakkar, 2017; Kuday, 2014).

1. **Test edilecek sistem:**
   Sistem içerisindeki alt sistemler bir bütünlük içerisinde ve stabil olmalıdır. Otomasyon test sürecinde test edilecek sistemde eğer bütün ve stabil alt sistemler söz konusu değilse maliyet açısından etkin bir test süreci mümkün olmayacaktır.
2. **Test platformu:**
   Otomasyon testlerinde test ya da testlerin gerçekleştirdiği platformun proje ihtiyaçlarına ve beklenen çıktılara uygun bir şekilde seçilmesi gerekir.
3. **Senaryo kütüphanesi:**
   Tekrarlı olarak kullanılabilecek kütüphanelerin derlenmesi önemli bir konudur.
4. **Otomatik Uygulama Pratikleri:**
   Test sürecinde kullanılan araçların ve kütüphanelerin test sürecinde dahil oldukları aktivite ve kontrol aşamalarının belgelendirilmesi gibi konuları içerir.
5. **Test Araçları:**
   Test komut dosyalarının geliştirilmesi için birçok farklı tipte test araçları bulunmaktadır.
6. **Test yöneticisi:**
   Test sürecindeki kütüphanelerin, platformların ve araçların yönetilmesi, testi gerçekleştiren kişilere komut dosyalarının geliştirilmesi konusunda rehberlik eden ve bilgi sağlayan kişidir.

Her bir bileşen test sürecinde kritik öneme sahiptir. Hedeflenen kalitede hedeflenen çıktıların elde edilebilmesi bu bileşenler arasındaki ilişkilerin doğru tanımlanmasına ve iyi yönetilmesine bağlı olarak değişir.

2.5.2 Yaygın Olarak Kullanılan Yapılar

Otomasyon testlerinde yaygın olarak kullanılan altı tip yapı vardır. Birbirlerine benzedikleri ve farklılaştıkları noktalar olmakla birlikte aşağıdaki yapıların özellikleri özetlenmektedir (Madan and Kakkar, 2017).

1- **Modül tabanlı test yapısı:**
Yüksek oranda birimlere ayrıştırma şansı nedeniyle maliyet açısından avantajlıdır. Bunun yanı sıra ölçeklendirilebilir ve hiyerarşik bir yapısı vardır. Değişiklikler yapılırken sadece ilgili komut parçası üzerinde çalışma imkânı sağlar. Her bir modül için açık bir şekilde komut dosyaları belirtilse de, test verisi test komutuna gömülüdür. Bu yüzden, farklı veri seti ile test gerçekleştirildiğinde çok sayıda manipülasyonla uğraşmak gerekir. Komut dosyaları yazılıma direkt olarak bağımlıdır.

2- **Kütüphane temelli test yapısı:**
Yüksek düzeyde modülleştirme söz konusudur. Bu yüzden maliyet açısından avantajlıdır ve ölçeklendirilebilir. Yüksek düzeyde yeniden kullanılabilir bir yapı mevcuttur. Dezavantajlara bakıldığında, test verilerindeki değişiklikler komut dosyalarında da değişiklik yapmayı gerektirir. Kütüphane temelli olduğu için yapının kompleks olduğu söylenebilir. Test verisi veritabanından çıkarılamaz. Test komutlarının planlanması ve hazırlanması süreci uzun olabilir. Kütüphane bazlı işleyen sistemde test altındaki uygulama komut dosyaları yerine yöntem ve fonksiyonlara (veya kurulum diliyle ilgili öğeler ve yöntemler) bölünür. Bu yüzden, test altında uygulama modüllerini, bölümlerini ve fonksiyonlarını temsil eden kütüphane dosyalarının oluşturulması bir esastır. Bu dosyalar test sürecinde doğrudan test senaryolarına ait komut dosyalarından çağrılır. Sonuç olarak yapısal komut dosyalarında yüksek düzeyde kodun yeniden kullanımını sağlamakla birlikte maliyet avantajı ve daha kolay komut dosyası bakımı avantajı sağlar. Teknik uzmanlık gerektirdiği, test senaryolarının planlama ve hazırlanma sürecinin zaman aldığı yadsınamaz bir gerçektir (Raju and Vaidhehi, 2017).

3- **Veri temelli test yapısı:**
Test verisindeki değişiklikler test komut kodlarını engellemez, herhangi bir değişikliğe ihtiyaç duyulmaz. Bu, esneklik ve sürdürülebilirlik açısından oldukça önemli bir faktördür. Tek bir test senaryosu bile test verisi değerleri değiştirilerek tekrarlı bir şekilde çalıştırılabilir. Toplam komut dosyalarının sayısı bu yapı ile oldukça düşük düzeyde tutulabilir. Dezavantajlara bakıldığında, sürecin karmaşık olduğu ve fazla gayret gerektirdiği söylenebilir. Programlama dillerinde uzmanlık

gerektirdiği de aşikârdır. Test sürecinde bazen aynı fonksiyonları ve işlevleri tekrarlı olarak test etmek gerekebilir. Farklı giriş verileri kullanılarak tekrarlanan test sürecinde, bu farklı giriş veri setlerinin ana test komut dosyalarına girmesi istenilen bir durum değildir. Veriye dayalı bir sistemde programda gezinme, veri dosyalarında veri okuma ve anlama, test durumu ve bilgilerinin yüklenmesi gibi işlemler test komut dosyalarında kodlanır. Bu noktada, komut dosyasında veriler için yalnızca bir sürücü veya dağıtım mekanizmasıdır. Bu sistemi tüm olası test senaryosu kombinasyonlarını kapsadığından daha az kodlama eforunun yanı sıra, test verilerinde yapılan değişiklik test komut dosyalarını etkilemeyeceğinden büyük ölçüde esneklik sağlar. Öte yandan, süreç veri kaynaklarının oluşturulması ve okunma mekanizmaları için ekstra çaba gerektirdiğinden karmaşıktır (Raju and Vaidhehi, 2017).

4- **Anahtar kelime temelli yapı:**

Komut dosyaları bilgisine sahip olmak bir zorunluluk değildir ve tek bir anahtar kelime çok sayıda komut dosyası arasında kullanılabilir. Anahtar sözcük yaratma mekanizmaları konusunda uzmanlık gerektirir, çünkü yeni anahtar sözcüklerin eklenmesiyle yapı büyüyecek ve karmaşık hale gelecektir.

5- **Hibrit yapı:**

Modüler, veri temelli ve anahtar sözcük temelli yapıların tüm avantajları hibrit için de geçerlidir. Kodların tekrar kullanılabilirlik düzeyi oldukça yüksektir. En hızlı ve en iyi maliyetli yapı olduğu söylenilebilir. Kullanıcı arayüzü ile veritabanı arasındaki veri alışverişini sağlayan algoritma ve ortak kodlar için herhangi ayrı işlevsel kütüphaneler mevcut değildir. Dolayısıyla, teknik uzmanlık gerektirir.

6- **Davranış temelli yapı:**

Sistemin incelenmesine fırsat veren ve doğru test setlerinin kolayca otomatize edilmesine izin veren bir yapıdır. İhtiyaçlar konusunda daha kesin çıkarımlarda bulunulmasını sağlar, fakat teknik ve teknik olmayan paydaşlar arasında sürekli olan bir iletişim ağını gerektirir.

2.5.3 Geliştiricilerin Rolü

Literatürde test süreci yaşam döngüsü, hata yaşam döngüsü ve modellerin yanı sıra geliştiricilerin rol ve görevleri üzerine de birçok çalışma bulunmaktadır. Özellikle otomasyon testlerinin yaşam döngüsünde geliştiriciler sürekliliği ve yaşayan bir otomasyon test sürecini kurmakta kritik öneme sahiptir. Bu görev ve roller aşağıdaki şekilde özetlenebilir (Quan, 2018).

- **Araştırma yapılması ve test planı tasarımı:**
  Geliştiricilerin test sürecine dair araştırma yapmaları, şirketin geçmiş test senaryo ve uygulamalarını bilmeleri test süreci açısında oldukça faydalı olacaktır. Akabinde, taslak bir test planı oluşturmak testin amacını ve adımlarını ilk etapta belirlerken kullanılacak en önemli kaynaklardan biridir. Proje yöneticisi ve test liderinin de katılımıyla taslak test planı üzerinde gerekli görüldüğü noktalarda revizyonlar gerçekleştirilmeli ve final test planı oluşturulmalıdır. Ek olarak, risk faktörlerini ve etkilerini ön görebilmek için risk değerlendirme çalışması yapılmalıdır. Test planının risklere ve beklenmedik durumlara karşı kalite kontrol sürecine ihtiyacı vardır. Örneğin, eğer bazı işlemler tahmin edildiğinden daha fazla zaman alıyorsa test planının gidişatı ve potansiyel değişiklikler konusunda geliştirici proje yöneticisini bilgilendirmekle sorumludur. Gerekli revizyonlar müşteriye proje yöneticisi ya da test lideri tarafından iletilir. Genel anlamda, koordineli ekip çalışması herkesin takip edebildiği ve iyi yönetilen bir test sürecini destekler.

- **Test analizi ve tasarım aşaması:**
  Test koşullarının ve senaryolarının belirlenmesi test uzmanlarının sorumluluğundadır. Örneğin, kullanıcının karakter tipi bir alana sayısal bir değer girmesi durumunda bir uyarı alması gerekir. Başlangıçta, bu senaryolar zararsız görülebilir, fakat ilerleyen aşamalarda performansa ciddi düzeyde zarar verebilirler. Bu senaryolar, genellikle ilk etapta projenin üst seviye ihtiyaçları kapsamında olmazlar. Fakat her ihtiyacın aslında mümkün olduğunca spesifik olması başarıyı getiren en önemli faktörlerden biridir. Bu yüzden, test uzmanların senaryolar ve ihtiyaçlar konusundaki deneyim ve bilgi düzeyleri oldukça önemli bir rol oynamaktadır. Test analizi aşamasından sonra test kullanıcılarının, test ortamının, veritabanının, lisansların, donanımın vb. teslim edildiği tasarım aşamasına geçilir.

- **Uygulama aşaması:**
  Bu aşama, test uzmanlarının test plan ve analizine dahil olan test senaryolarını uygularlar. Test senaryoları belli bir girdi veya değişiklikten sonra beklenen davranışı tanımlar.
  Örneğin:
  - Test senaryosu 1: Kullanıcı üç veya daha fazla kez yanlış şifre giriyor, bir uyarı görülmelidir.
    - Test örneği 1: Kullanıcı üç kez yanlış şifre girdi, uyarı almalıdır.
    - Test örneği 2: Kullanıcı beş kez yanlış şifre girdi, uyarı almalıdır.

- Test senaryosu 2: Kullanıcı iki kez yanlış şifre girdi, bir uyarı görülmemelidir.
  - Test örneği 1: Kullanıcı bir kez yanlış şifre girdi, uyarı almamalıdır.
  - Test örneği 2: Kullanıcı iki kez yanlış şifre girdi, uyarı almamalıdır.

Yukarıda örneklendirildiği gibi, bütün tasarlanan test senaryoları toplanır ve bir test ortamında uygulanır. Bu aşamada en önemli faktörlerden biri yazılımın devreye alındıktan sonra gerçek davranışları taklit edebilmesi için çok iyi bir test ortamında uygulanması gerekir. Test ortamında tamamlanan uygulama sonuçları test uzmanları tarafından toplanır ve onaylanır.

- **Test raporunu toplama ve değerlendirme:**
  Proje yöneticisi ya da test lideri test uygulamasının yeterli olup olmadığını değerlendirir. Test raporu, başarılı test senaryolarının, başarısız test senaryolarının ve hataların yüzdesini içerir. Bu tip metrikler, projenin durumunun takip edilmesi ve performansın değerlendirilmesi açısından oldukça önemlidir. Her test aşamasından sonra test raporu hazırlanır ve test raporlarının kendi içlerinde karşılaştırması yapılarak proje performansı konusunda çıkarımlarda bulunulur.

Otomasyon testlerinin genel kapsamını özetlemek gerekirse, testlerin gerçekleştirilme süresinin kısalması, tekrarlı testlerin manuel efor ve zamanından tasarruf sağlaması, kaynakların daha verimli kullanımına olanak sağlaması tekrar edilebilirlik ve güvenilirlik açısından oldukça avantajlıdır (Kökten, 2019). Bunun yanı sıra otomasyon testlerinden tüm hataları bulmasını, hatasız test yapmasını, değişikliklere ihtiyaç duyulmayacağını, uyum sürecinin hızlı ve kusursuz olacağını beklemek yanlıştır. Yukarıda bahsedildiği üzere birçok bileşen, yapı ve geliştiricini var olduğu bir sistemde uyum ve revizyon süreçleri zaman alabilir.

2.6 Selenium

2.6.1 Selenium'un Kısaca Tarihi

Selenium 2004 yılında Thought Works'de çalışan Jason Huggins tarafından geliştirilmiştir (Yadav and Kumar, 2015). Bu girişimin arkasındaki hikâye şöyledir: O dönem Jason düzenli olarak test süreci gerektiren bir web uygulaması üzerinde çalışıyordu. Bu çalışmaları sırasında manuel olarak yapılması gereken test süreçlerinin arttığını ve artışın verimsizliğe neden olduğunu fark etti. İlk olarak, tarayıcı aksiyonlarını otomatik olarak kontrol edebilen bir JavaScript programı oluşturdu ve bu programa JavaScript Test Runner ismini verdi.

Sonrasında, open-source olarak bu programı tamamladı ve Selenium Core adını verdi (Yadav and Kumar, 2015).

Selenium, birçok farklı tarayıcı ve işletim sisteminde çalışabiliyor olması, farklı programlama dilleri ve test sürecinde dair yapılar tarafından kontrol edilebilir olması ve çok farklı sayıda test fonksiyonlarını içerisinde barındırıyor olması nedeniyle günümüzde oldukça popülerdir ve yaygın kullanım alanına sahiptir.

2.6.2 Selenium Genel Kapsamı ve Yapısı

Otomasyon testinin en önemli özelliklerinden biri test sürecinde insan etkileşiminden dolayı oluşabilecek hataları önlemektir (Raju and Vaidhehi, 2017). Selenium test senaryolarını otomotize hale getirmek ve performansı arttırmak için kullanılan açık kaynaklı ve web tabanlı bir test aracıdır (Chandraprabha, 2015). Özellikle web uygulamalarında sıkça kullanılan Selenium, bu uygulamaların günümüz dünyasında alanının genişlemesiyle birlikte test süreçlerinin karmaşıklaşmasından dolayı önemli hale gelmiştir. Hemen hemen her çeşit web uygulamasının test edilmesi süreci için oldukça geniş bir test fonksiyon setini içerir. Yüksek düzeyde esnek operasyonları farklı tarayıcı platformlarında çok çeşitli test fonksiyonlarıyla gerçekleştirmek mümkündür. Selenium'da yazılan komutlar "Selenese" olarak adlandırılır. Örnek vermek gerekirse, eylemleri hedefleyen komutlar genellikle durum değişikliği söz konusuysa kullanılır. Örneğin, "Bu bağlantıyı tıklayın" veya "Bu seçeneği seçin" gibi eylemlerin test edilmesi sürecinde eylem odaklı komutlar rol alır. Bu tür eylemler eğer başarısız olursa veya bir hata söz konusu olursa, testin yürütülmesi süreci durdurulur.

Selenium'un web uygulamaları için tam bir otomasyon kapsamında kullanılabilen ve her birinin farklı rolü olan üç sürümü vardır (Yadav and Kumar, 2015; Gojare et al., 2015).

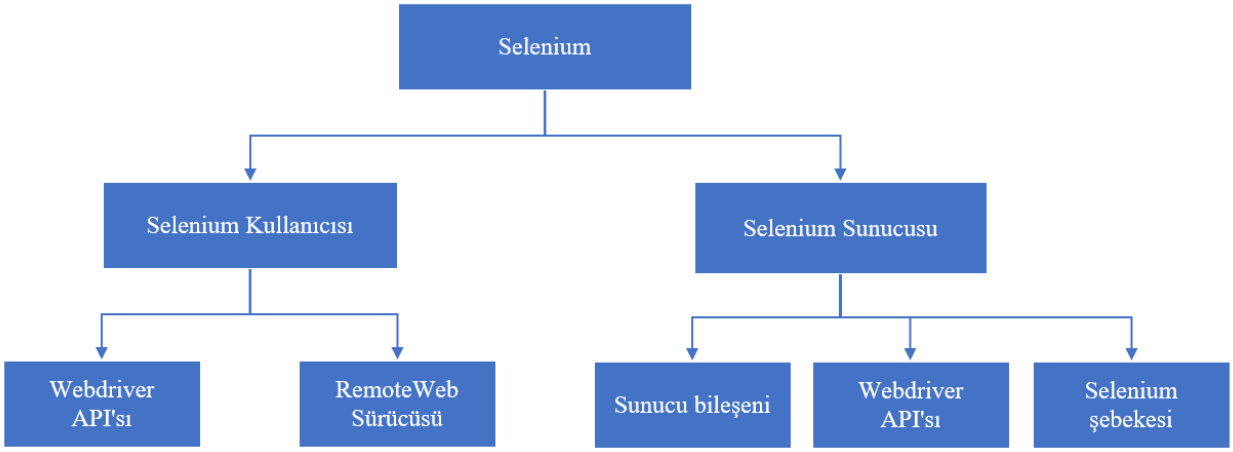
- **Selenium IDE :**
  Testler için entegre bir geliştirme ortamı şeklinde tanımlanabilir. İlk olarak Shinya Kasatani tarafından tasarlanan Selenium IDE'nin kayıt özelliği vardır ve Firefox uzantısı olarak kullanılmaktadır. Test senaryolarının düzenlenmesi ve hataların ayrılması bu ortamda gerçekleşir.
- **Selenium Core :**
  Web uygulamaları için bir test aracı olan Selenium Core, Seleniumun daha basit formu şeklinde nitelendirilebilir. Fakat, http ve https protokolleri arasında geçiş yapabilmek ve dosya işleme gibi aksiyonları gerçekleştiremez.

- **Selenium RC:**

Yalın tarayıcı aksiyonlarından ve doğrusal uygulamalardan daha fazlasını gerektiren testler için iyi bir çözümdür. Dosya okuma ve yazma, veritabanını sorgulama ve test sonuçlarını e-posta ile gönderme gibi göreceli olarak daha kompleks testler için uygundur. Test komut dosyalarının çalıştırılmasından önce sunucunun başlatılıyor olması Selenium RC için bir gerekliliktir.

Selenium WebDriver, bir diğer deyişle Selenium 2.0, Webdriver tarayıcı ile doğrudan iletişime geçer, dolayısıyla Selenium RC'den çok daha hızlı olduğu söylenebilir (Gojare et al., 2015). Test dosyalarını yazmak için birçok dil desteğini bünyesinde barındıran Selenium WebDriver, farklı tarayıcılar ve işletim sistemleri arayıcılığıyla işlev görme yeterliliklerine sahiptir fakat masaüstü tabanlı uygulamaları test etme yeterliliği mevcut değildir (Devi et al., 2017). Bunun yanı sıra, test sonuçlarını üretme özelliği mevcut değildir (Gojare et al., 2015).

Selenium'un genel yapısı aşağıdaki şekilde özetlenmiştir.

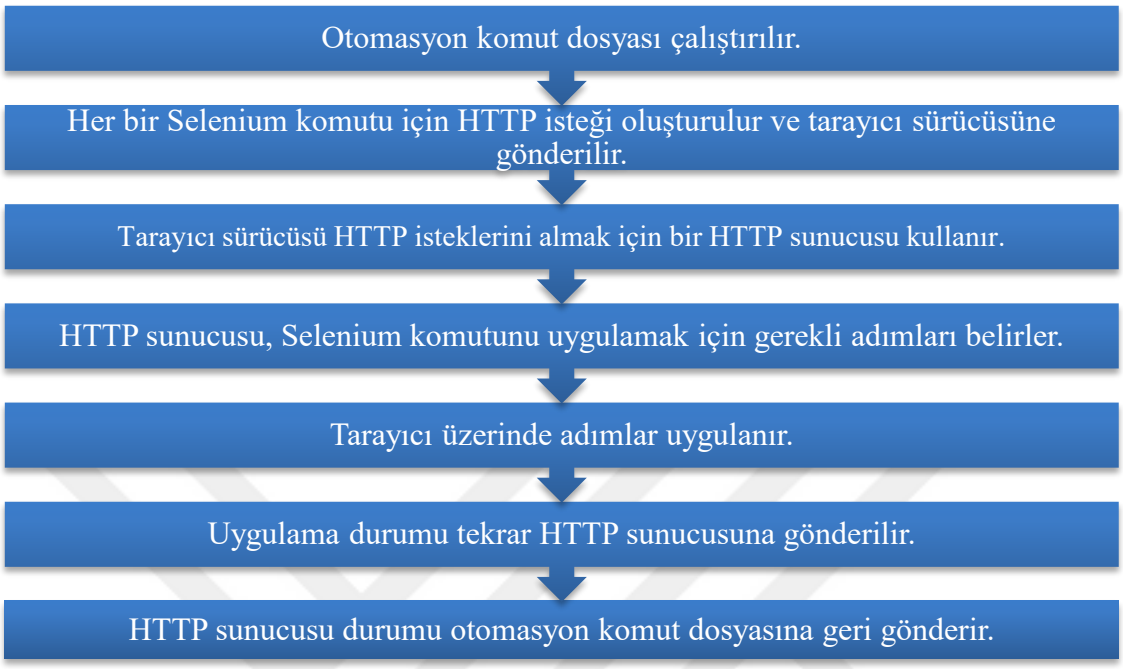


Şekil 2.6. Selenium Genel Yapısı (Gojare et al., 2015)

Şekilde görüldüğü üzere Selenium kullanıcısı ve sunucusu iki temel bileşen olarak karşımıza çıkmaktadır. Her iki bileşen de Webdriver API'sını içerir. Webdriver API web sayfası ve diğer uygulama öğeleriyle etkileşim kurmak ve test komut dosyaları oluşturmak amacıyla kullanılır. Ek olarak, uzak bir Selenium sunucusuyla iletişim kuran uzak (remote) web sürücüsü sınıfını da içermektedir. Selenim sunucusu, Selenium kullanıcısının RemoteWeb sürücüsünden gelen istekleri kabul etmek için kullanılan bir sunucu bileşeni, web tarayıcı testlerini yürütmek için kullanılan Webdriver API, ve benzer olmayan makineler ile tarayıcılar aracılığıyla paralel testleri çalıştırmak için kullanılan Selenium şebekesinden oluşmaktadır (Devi et al., 2017).

2.6.3 Selenium Operasyonlarının Genel Akışı

Şekil 2.6'da Selenium WebDriver operasyonlarının iş akışı görülmektedir (Quan, 2018).



Şekil 2.7. Selenium WebDriver operasyonlarının genel akışı (Quan, 2018)

Selenium WebDriver PHP, C# (Nunit), Java (TestNG, JUnit), Perl, Ruby (Rspec) ve Python (Robot, PyUnit, Unittest) gibi bir çok farklı tipte müşteri kitaplığını destekler. Bunun yanı sıra IR, Chrome, FireFox, Safari ve PhantomJS gibi web tarayıcılarında da çalışmaktadır.

2.7 Literatürde Selenium Uygulama Makaleleri

Bu bölümde bu tezin uygulama kısmına destek sağlayan literatürdeki Selenium uygulama makalelerinin kapsamı ve içeriği özetlenecektir. Teorinin yanı sıra otomasyon testlerinde Selenium'un nasıl kullanıldığına dair çıkarımlar açısından faydalı olacağı ön görülmektedir. Ek olarak tez çalışmasının yapılma amacı açıklanacaktır.

1. Lariya ve diğ. (2018), çalışmalarında Selenium WebDriver'ı kullanarak havayolu uygulamasında hesaba giriş (başarılı veya başarısız) ve uçuş arama fonksiyonlarını test etmişlerdir. Temel olarak problem analizi ile başlayan süreç akışı; giriş sayfası ve arama sayfası için test senaryolarının yazılması, sonuçların raporlanması ve değerlendirilmesi ile sonlanmaktadır.
2. Neethidevan ve Chandrasekaran (2018) öğrencilerin sınav kayıtları üzerinde bir uygulama çalışması gerçekleştirmişlerdir. Uygulamada öğrencilerin kullanıcı adı, üniversite adı ve ders adı gibi bir çok veri alanı web sayfası giriş alanında ayrıntılı olarak girilmektedir. Bu veriler MySQL'de saklanmaktadır. Çalışmada Selenium Web Driver kullanılarak MySQL veritabanına bağlanan ve kullanıcı tarafından girilen son kaydı alan komut dosyaları geliştirilmiştir. Test komut dosyası, kullanıcı tarafından girilen veri alanlarının veritabanına doğru bir şekilde aktarılıp aktarılmadığını test etmiştir.
3. Chandraprabha ve diğ. (2015) çalışmalarında Selenium WebDriver kullanarak otomasyon testi gerçekleştirmişlerdir. Çalışmada verileri tekrarlı olarak kullanabilmek adına veri odaklı bir çalışma çerçevesi çizilmiştir. İlk aşamada test verilerini depolayan harici bir dosya oluşturulmuş, hemen ardından ikinci aşamada verileri otomasyon test komut dosyasına gönderme işlemi gerçekleştirilmiştir. Çalışma veri odaklı bir çerçevenin, büyük veri üzerinde otomasyon testlerinde efor ve zaman açısından oldukça avantajlı olduğunu vurgulamaktadır.
4. Quan (2018) çalışmasında konferans merkezleri, oteller, mekanlar vb. için online rezervasyon alan bir şirketi ele almıştır. Kullanıcıların şirketin web sayfasında doğru kimlik bilgileriyle başarılı bir şekilde oturum açma süreci test edilmiştir. Test sürecinde zamanlama açısından Selenium WebDriver ile yaklaşık %72'lik bir tasarruf sağlandığı belirtilmektedir. Zaman ve kaynak tasarrufunun yanı sıra sonuçların geçerliliği konusunda üst düzey bir verimlilik ve etkinlik sağlanabileceği çıkarımında bulunulmuştur.
5. Ramya ve diğ. (2017) bir websitesinin giriş sayfasını Selenium Web Driver kullanarak test etmişlerdir. Özellikle çoklu test verisi kullanarak çıktı raporlarını farklı şekillerde (e-mail, excel vb.) sunmaları açısından uygulamada önemli bir kaynak olarak kaydedilebilir.

6. Sharma ve Angmo (2014) çalışmalarında çeşitli web otomasyon test araçlarını incelemişlerdir. Selenium, Watir, Hp-Qtp, Test Complete, Test Ng, Hp Load Runner, Silkest, FitNesse ve Win Runner arasından web sitelerini test etmede en kolay ve en iyi otomasyon yönteminin Selenium olduğu çıkarımına varmışlardır.
7. Singla ve Kaur (2014) çalışmalarında Selenium Webdriver yazılım test aracını kullanarak web uygulamalarında otomasyon testi yaparken kelime odaklı bir çerçeve geliştirmişlerdir.
8. Sualim ve diğ. (2016) çalışmalarında üç farklı aracı kabul testi otomasyonunda karşılaştırmalı olarak incelemişlerdir. TestComplete, Selenium Webdriver ve Watir Webdriver çalışmaya konu olan üç araçtır. Selenium Webdriver için kullanıcı deneyiminin gerekliliği vurgulanan konulardan biri olmuştur. Bunun yanı sıra senaryoların gerçekleştirilme zamanının kısa olması açısından verimlilik konusu Selenium Webdriver için belirtilmiştir.
9. Kunte ve Mane (2017) bir başka karşılaştırmalı çalışmanın sahipleridirler. Otomasyon testi yaptıkları çalışmalarında Selenium ve UFT'yi karşılaştırmalı olarak incelemişlerdir. Maliyet açısından Seleniumun açık kaynak kodlu olması, UFT'nin lisans gerektirmesi Selenium'u avantajlı hale getiren bir parametre olmuştur. UFT'nin yalnızca Windows platformunda kullanılabiliyor olması, Selenium'un ise birçok farklı platformda kullanılabilir olması bir başka önemli parametre olarak karşımıza çıkmaktadır. UFT'de yapılan testlerin bir kerelik ve tek bir makinada yapılabilir olması Selenium için bir başka avantaj olarak raporlanmıştır. Selenium aynı anda birden fazla testi simultane bir şekilde gerçekleştirebildiği için daha etkili ve verimli olduğu yönünde çıkarımda bulunmuşlardır.

Tez çalışması kapsamında web servis, işlevsellik ve veri temelli testler gibi farklı türdeki testler ve bu testlerin hangi araçlar ile yapılabileceği araştırılmış sonrasında ise ilgili testler gerçekleştirilip sonuçları ile beraber birtakım bilgiler verilmiştir.

3. YAZILIM TESTLERİNİN GERÇEKLEŞTİRİLMESİ

3.1 Java Restful Servis Testlerinin Rest Assured Aracı ile Gerçekleştirilmesi

3.1.1 Java Restful API Nedir ?

Java Restful servisler veritabanından verileri çekip uygulamalara aktarmamızı sağlayan önemli hizmetler olarak düşünülebilir. Kullanıcı, uygulaması ile öncelikle servise HTTP talepte bulunur. Servis isteği işleyip veritabanına bağlanarak gerekli verileri elde eder. Daha sonra kullanıcı uygulamasına ilgili verileri geri gönderir. Servis aracılığıyla uygulamada verileri kullanabilmek için önemli 2 tür metot kullanılmaktadır:

- GET: Servis ile veritabanından okuma işlemini yapar.
- POST: Servis ile veritabanına ekleme işlemini yapar.

Restful API'ler 4 temel ilke üzerine kurulmuştur. Bu ilkeler aşağıda anlatılmaktadır:

- **İstemci – Sunucu (Client – Server):** Bu madde Rest mimarisinin ilk temel kısıtlaması olarak kabul edilmektedir. Bu kısıtlamaya göre bir uygulama istemci – sunucu şeklinde modellenmelidir. Başka bir deyişle uygulama veriden ayrı olarak bir kullanıcı arayüzüne sahip olmalıdır. Yani uygulamada ön taraf (kullanıcı arayüzü) ve arka taraf (veritabanı) olmak üzere 2 temel bileşen bulunmaktadır. İstemci – sunucu mimarisinde kullanıcı arayüzü istemci rolünde iken arka taraf ise sunucu rolünü oynamaktadır. İstemci ve sunucu bileşenlerinin birbirinden bağımsız olarak gelişebilmesi bu mimarinin sağladığı bir fayda olarak nitelendirilebilir. Ek olarak; bir sunucu birden fazla istemciye hizmet verebilir. İstemciler ise farklı teknolojilerde olup aynı sunucudan faydalanabilir.
- **Durumsuz (Stateless):** Rest mimarisinin ikinci kısıtlaması olarak kabul edilmektedir. Bu kısıtlamaya göre bir sunucu istemciye ait bilgileri veya içerikleri kendi bünyesinde depolamamalıdır. İstemci tarafından gönderilen her bir istek, yine aynı istemci tarafından daha önce yapılmış istekler ile ilişkilendirilmez. Yani özet olarak, sunucu tarafında istemci ile ilgili oturum bilgileri tutulmamaktadır. İstemci sunucuya bir talep yaparken sunucuya gerekli bilgileri her defasında taşımaktadır. Bu yapının bir avantajı da görünürlüğü sağlamasıdır. Çünkü sunucu yapılan talebi anlamlandırırken sadece talepteki bilgilere bakmaktadır. Ayrıca sunucunun kaynak yönetimi yapmasına da gerek yoktur. Yapının dezavantajı ise her isteğe sunucunun ihtiyaç duyduğu bilgilerin eklenmesi ağ trafiğini arttırmakta ve gelen isteklerin doğrulanması sunucuya ekstra bir yük getirmektedir.
- **Önbellek (Cache):** Rest mimarisinin üçüncü kısıtlaması olarak belirlenmiştir. Sunucu tarafından gönderilen yanıtların istemci tarafında önbelleğe alınıp alınmayacaklarına ilişkin birtakım bilgileri içermesi gerekmektedir. Bu genellikle yanıt içinde gönderilen başlık bilgisi ile yapılmaktadır. Bu kısıtlama önbelleklenebilir yanıtlar için istemci verimliliğini arttırmaktadır. İstemci tekrar tekrar sunucuya talepte bulunmaz ve ilgili yanıt için önbelleğini kontrol etmektedir. Bu mekanizma ise ağın bant genişliği ve istemcinin işlem gücünden tasarruf sağlamaktadır.

- **Tek Tip Arayüz (Uniform Interface):** İstemci ve sunucu arasındaki arayüzü tanımlayan kısıtlama olarak tanımlanmıştır. Her bir parçanın birbirinden bağımsız olarak gelişmesine olanak sağlayan bu madde servis mimarisini basitleştirmektedir. Tek tip arayüz kısıtlamasına rehberlik eden birkaç ana ilke bulunmaktadır.

  Kaynak Tabanlı: RESTFUL servis yapısında kaynaklar isteklerde URI'ler kullanılarak tanımlanmaktadırlar. Kaynaklar kavramsal olarak istemciye geri gönderilen gösterimlerden farklıdırlar. Örnek olarak; sunucu veritabanını istemciye göndermemekle beraber veritabanı kayıtlarını temsil eden HTML, XML veya JSON veri formatlarını yanıtlarında kullanmaktadır.

  Kendinden Açıklamalı Mesajlar: Her mesaj kendisinin nasıl işleneceğine ilişkin yeterli düzeyde bir bilgi içermektedir. Örnek olarak hangi ayrıştırıcının kullanılacağı Internet Medya tipi tarafından (MIME türü) belirtilmektedir.

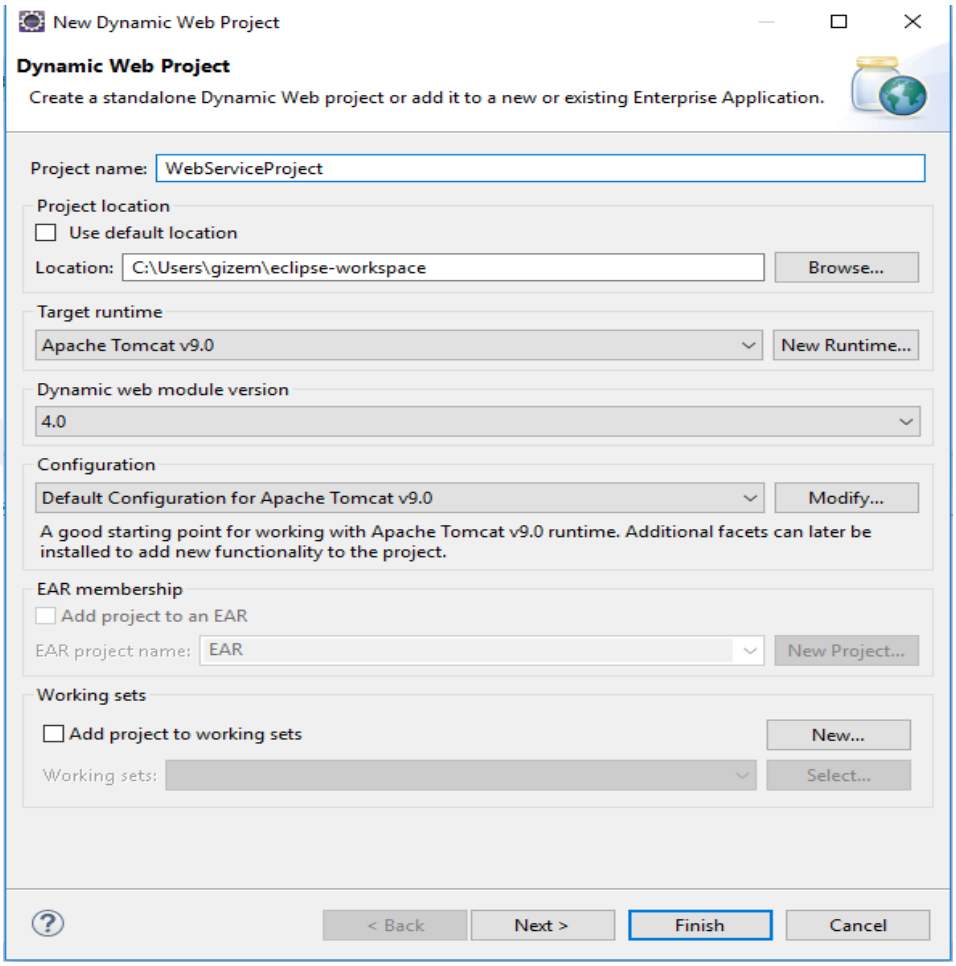
  Uygulama Durumunun Motoru olarak Hiperortam: İstemciler kendi durumlarını gövde içerikleri, sorgu dizesi parametreleri, istek başlık bilgileri ve talep ettikleri URI ile ifade ederken sunucular ise durumlarını yine gövde içerikleri, yanıt kodları ve yanıt başlıkları aracılığıyla istemcilere sunmaktadırlar. Bu teknik olarak Hiperortam olarak adlandırılmaktadır. Bunun yanında sunucunun istemciye döndürdüğü cevapta ilgili nesneye nasıl ulaşılacağına ilişkin bir link bulunuyorsa bu da Hiperortam konusu içinde yer almaktadır.

  Katmanlı Sistem: RESTFUL mimarisinin bu kısıtlamasına göre sistem uygulaması katmanlı bir yapıda olmalıdır. Her bir katman genel olarak sistemin belirli bir fonksiyonunu soyutlamaktadır. Katmanlar, doğrudan etkileşime girdiği katmanlar haricinde diğer katmanların varlığından haberdar olmamalıdırlar. Katmanlı yazılım tasarımı da REST konusu dışında yaygın olarak kullanılan bir tekniktir. Bu kısıtlama sistem içinde var olan farklı bileşenlerin genel karmaşıklığını azaltmaktadır.

  Talep Üzerine Kod: Bu madde RESTFUL mimarisinin isteğe bağlı olarak getirdiği bir kısıtlama olarak düşünülebilir. Bu kısıtlamaya göre bir istemci sunucudan belirli bir kod indirerek kendi işlevini genişletebilmektedir. Bu dolaylı olarak sistemin de genişlemesine olanak sağlamaktadır. Bu kod, uygulamalar (Applet) veya komut dosyaları (Script) formunda olabilir. Bu maddeye örnek olarak Javascript verilebilir. Diğer yandan sunucunun istemciye kod parçacığı göndermesi güvenlik sorunlarını da beraberinde getirebilmektedir (Sharma, 2017; Soni and Ranga, 2019)

3.1.2 Eclipse'de Restful Web Servis Oluşturulması ve Test Aşamaları

Eclipse'de Restful web servisinin uygulanması için öncelikle yeni bir dinamik web projesi aşağıdaki gibi oluşturulmuştur.



Şekil 3.1. Web Servis Projesinin Oluşturulması

Bu web servis projesinde kullanıcı veritabanında bulunan kullanıcı tablosundan ilgili kayıtları bazı kriterlere göre elde etmek amaçlanmaktadır. Ek olarak yine web sayfası üzerinden ilgili değerlerin girilmesi ve web servisi aracılığıyla yeni bir kullanıcının oluşturulup kullanıcı tablosuna eklenmesi de proje kapsamında gerçekleştirilen bir fonksiyondur. Kullanıcı tablosu aşağıdaki SQL kod parçacığı ile oluşturulmuştur.

```
CREATE TABLE [dbo].[User](
    [ID] [int] IDENTITY(1,1) NOT NULL,
    [name] [nvarchar](20) NOT NULL,
    [surname] [nvarchar](20) NOT NULL,
    [age] [int] NOT NULL,
 CONSTRAINT [PK_User] PRIMARY KEY CLUSTERED
(
    [ID] ASC
```

Şekil 3.2. Kullanıcı Tablosunun Oluşturulması

Dinamik web servis projesini oluşturduktan sonra öncelikle proje ile veritabanı bağlantısını sağlayabilmek için DbUtil.java sınıfı aşağıdaki gibi yazılmıştır.

```
package dbOperations;

import java.sql.Connection;

public class DbUtil {

    private static String url = "jdbc:sqlserver://GIZEMIREN\\SQLEXPRESS:1433; databaseName=USER; integratedSecurity=true;";

    public static PreparedStatement getPreparedStatement(String sql) throws ClassNotFoundException, SQLException
    {
        PreparedStatement ps = null;
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
        Connection con = DriverManager.getConnection(url);
        ps= con.prepareStatement(sql);

        return ps;
    }
}
```

Şekil 3.3. Eclipse'de Veritabanı Bağlantı Sınıfının Oluşturulması

Bu sınıfta veritabanı bağlantı linkinin ve SQL sunucu sürücüsünün tanıtılmasının ardından veritabanına bağlantı yapılıp ilgili SQL sorgu cümlesinin hazırlanması gibi işlemler yapılmaktadır. Veritabanı tablosundaki özelliklerin program içinde eşleştirilmesi için UserModel.java isimli sınıf yine aşağıdaki gibi yazılmıştır. Bu sınıfta ID, name, surname, age gibi özellikler özel olarak tanımlanmış olup yeni bir nesnenin yaratılmasını sağlamak amacıyla yapıcı metot oluşturulmuştur.

```
package models;

public class UserModel {

    private int ID;
    private String name;
    private String surname ;
    private int age;

    public UserModel(int ID, String name, String surname, int age)
    {
        this.ID = ID;
        this.name = name;
        this.surname = surname;
        this.age = age;
    }
```

Şekil 3.4. Eclipse'de UserModel Sınıfının Yazılması

Restful web servisin tanımlanması için yine ayrı bir sınıf yaratılmış olup bu sınıf içinde servis ayarlamaları yapılmıştır. Java'da javax.ws.rs kütüphanesi web servis uygulamaları için yazılmış olup bu kütüphanenin sınıfları manuel olarak tanımlanan web servis sınıfları için

kullanılmaktadır. Örnek olarak; yeni bir web servis sınıfı oluşturulurken Application ata sınıfından bu sınıfın alt sınıf olarak aşağıdaki gibi türetilmesi gerekmektedir. @ApplicationPath anahtar sözcüğü web servislere özgü olup web servis yolunun projede belirtilmesi için başvurulmaktadır.

```
package dbOperations;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("webservice")
public class WebServiceApplication extends Application {

}
```

Şekil 3.5. Web Servis Sınıfının Yazılması

Web servis sınıfı oluşturulduktan sonra veritabanı tablosundan çekilen kayıtların programda tutulması ve bu kayıtlara erişimin sağlanması için BusinessUtil.java sınıfı aşağıdaki gibi yazılmıştır. Bu sınıfta tablodan çekilen kayıtlar model sınıfı tipindeki ArrayList veri yapısında saklanmakta olup gerektiğinde tüm kayıtlara bu liste vasıtası ile ulaşılmaktadır.

```
package BusinessUtil;

import java.sql.ResultSet;

public class BusinessUtil {

    ArrayList<UserModel> userList;

    public BusinessUtil()
    {
        userList = new ArrayList<UserModel>();
    }

    public void addUser(ResultSet rs) throws SQLException {

        UserModel newlyCreatedUser;

        while (rs.next()) {

            newlyCreatedUser = new UserModel();
            newlyCreatedUser.setID(rs.getInt("ID"));
            newlyCreatedUser.setName(rs.getString("name"));
            newlyCreatedUser.setSurname(rs.getString("surname"));
            newlyCreatedUser.setAge(rs.getInt("age"));
            userList.add(newlyCreatedUser);
        }
    }

     public List<UserModel> getUserList()
     {
         return userList;
     }
```

Şekil 3.6. BusinessUtil.java Sınıfının Yazılması

Web servis üzerinden yapılan veritabanı işlemleri ayrı bir sınıfta toplanmış olup bu işlemler WebServiceOperations.java sınıfı içinde yazılmıştır. Öncelikle tablodan tüm kayıtların çekilebilmesi için **queryAllUsers** metodu aşağıdaki gibi oluşturulmuştur. Bu metotta ilgili SQL sorgusu yaratılıp çalıştırıldıktan sonra sonuçlar programda ResultSet yapısı içinde tutulup çağırıldığı yere geri dönmektedir.

```
package dbOperations;

import java.sql.PreparedStatement;

public class WebServiceOperations {

     ResultSet rs = null;

    public ResultSet queryAllUsers() throws SQLException {

        String query;
        query = "select * from [User]";

        try
        {
          PreparedStatement ps = DbUtil.getPreparedStatement(query);
          rs = ps.executeQuery();
        }

        catch(ClassNotFoundException ex)
        {
            Logger logger = Logger.getLogger(WebServiceOperations.class.getName());
            logger.log(Level.SEVERE, null, ex);
        }

        catch(SQLException ex)
        {
            Logger logger = Logger.getLogger(WebServiceOperations.class.getName());
            logger.log(Level.SEVERE, null, ex);
        }

        return rs;

    }
```

Şekil 3.7. Veritabanından Tüm Kullanıcıların Çekilmesi

Yine aynı şekilde kullanıcılar id, ad, soyad ve yaş filtrelerine göre web servis üzerinden veritabanına bağlanılıp aşağıdaki gibi çekilmektedir.

```
public ResultSet queryAllUsersById(int id) throws SQLException {

        String query;
        query = "select * from [User] where id like '" + id + "%'";

        try
        {
          PreparedStatement pst = DbUtil.getPreparedStatement(query);
          rs = pst.executeQuery();
        }
```

Şekil 3.8. Veritabanından Kullanıcıların Id Kriterine Göre Çekilmesi

```
public ResultSet queryAllUsersByAge(int age) throws SQLException {

    String query;
    query = "select * from [User] where age like '" + age + "%'";

    try
    {
      PreparedStatement ps = DbUtil.getPreparedStatement(query);
      rs = ps.executeQuery();
    }
```

Şekil 3.9. Veritabanından Kullanıcıların Yaş Kriterine Göre Çekilmesi

```
public ResultSet queryAllUsersByName(String name) throws SQLException {

    String query;
    query = "select * from [User] where name like '" + name + "%'";

    try
    {
      PreparedStatement pst = DbUtil.getPreparedStatement(query);
      rs = pst.executeQuery();
    }
```

Şekil 3.10. Veritabanından Kullanıcıların Ad Kriterine Göre Çekilmesi

```
public ResultSet queryAllUsersBySurname(String surname) throws SQLException {

    String query;
    query = "select * from [User] where name like '" + surname + "%'";

    try
    {
      PreparedStatement pst = DbUtil.getPreparedStatement(query);
      rs = pst.executeQuery();
    }
```

Şekil 3.11. Veritabanından Kullanıcıların Soyad Kriterine Göre Çekilmesi

Şekil 3.12'da ise uygun sorgu kullanılarak kullanıcılar ad ve yaş kriteri ile beraber tablodan getirilmektedir.

```
public ResultSet queryAllUsersByAgeAndName(String name, int age) throws SQLException {

    String query;
    query = "select * from [User] where name like '" + name + "%' and age like '" + age + "%'";

    try
    {
      PreparedStatement pst = DbUtil.getPreparedStatement(query);
      rs = pst.executeQuery();
    }
```

Şekil 3.12. Veritabanından Kullanıcıların Ad ve Yaş Kriterlerine Göre Çekilmesi

Veritabanı işlemleri oluşturulduktan sonra bu işlemlerin kullanılıp sonuçların web servisleri üzerinden döndürülmesi ayrı ayrı metotlar halinde düzenlenmiş olup farklı bir sınıfta tutulmaktadır. İlk olarak tüm kullanıcıların web servisi üzerinden veritabanından çekilip sonuçların web sayfasından gösterilmesi amaçlanmıştır. Bu işlem bir servis metodu içinde

gerçekleştirilmiştir (Şekil 3.13). Ana web servis sınıfının yapısı şekilden görüleceği üzere bir @Path notasyonu ile başlayıp bu belirteç ile web servis sınıfının yolu belirtilmiştir. Ardından servis metodunun da projede kendine ait bir yolu olacağı için yine metot yazılmadan önce aynı notasyon kullanılmıştır. Bunun yanında servis metotlarının başında farklı notasyonlar da yazılabilmektedir. Örnek olarak metot tarafından elde edilen verinin web sayfasında hangi veri formatında gözükeceği @Produces notasyonu ile belirtilmektedir. Sonuçların JSON veri tipinde gönderilmesi için APPLICATION_JSON değeri medya tipi olarak seçilmelidir. Son olarak ise metodun GET ya da POST metodu olup olmadığının belirtilmesi için @GET ve @POST notasyonları özel olarak yazılmaktadır.

```
package dbOperations;

import java.sql.ResultSet;

@Path("/getUserService")
public class WebServiceMain {

    WebServiceOperations wso;
    BusinessUtil businessUtil;
    String result;

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("/getAllUsers")
    public List<UserModel> start() throws Exception {

        wso = new WebServiceOperations();
        businessUtil = new BusinessUtil();

        ResultSet myRs = wso.queryAllUsers();
        businessUtil.addUser(myRs);

        return businessUtil.getUserList();
    }
```

Şekil 3.13. Web Servis Metodunun Tanımlanması

Kullanıcıların web servis üzerinden yaş kriterine göre çekilmesi aşağıdaki gibi yapılmaktadır. Bu işlem için yaş parametresi kullanılacağı için bu parametrenin metotta verilmesi gerekmektedir. Bu kapsamda @PathParam notasyonu ile uygun parametre ayarı yapılmıştır.

```
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("/age/{age}")
public List<UserModel> start(@PathParam("age") int userAge) throws Exception {

    wso = new WebServiceOperations();
    businessUtil = new BusinessUtil();

    ResultSet myRs = wso.queryAllUsersByAge(userAge);
    businessUtil.addUser(myRs);
    return businessUtil.getUserList();
}
```

Şekil 3.14. Kullanıcıların Yaş Kriterine Göre Servisten Çekilmesi

Benzer olarak kullanıcıların ad, soyad ve id filtrelerine göre servis üzerinden elde edilmesi aşağıdaki şekillerden görülmektedir.

```
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("/name/{name}")
public List<UserModel> getName(@PathParam("name") String userName) throws Exception {

    wso = new WebServiceOperations();
    businessUtil = new BusinessUtil();

    ResultSet myRs = wso.queryAllUsersByName(userName);
    businessUtil.addUser(myRs);
    return businessUtil.getUserList();
}
```

Şekil 3.15. Kullanıcıların Ad Kriterine Göre Servisten Çekilmesi

```
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("/surname/{surname}")
public List<UserModel> surname(@PathParam("surname") String userSurname) throws Exception {

    wso = new WebServiceOperations();
    businessUtil = new BusinessUtil();

    ResultSet myRs = wso.queryAllUsersBySurname(userSurname);
    businessUtil.addUser(myRs);
    return businessUtil.getUserList();
}
```

Şekil 3.16. Kullanıcıların Soyad Kriterine Göre Servisten Çekilmesi

```
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("/id/{id}")
public List<UserModel> getId(@PathParam("id") int userId) throws Exception {

    wso = new WebServiceOperations();
    businessUtil = new BusinessUtil();

    ResultSet myRs = wso.queryAllUsersById(userId);
    businessUtil.addUser(myRs);
    return businessUtil.getUserList();
}
```

Şekil 3.17. Kullanıcıların Id Kriterine Göre Servisten Çekilmesi

Java Restful Web Servis projesini oluşturup başarılı bir şekilde yerelde (lokal) çalıştırdıktan sonra servis metotlarının test edilmesi işlemleri sırasıyla yapılmıştır. Test aşamasında Java'nın Rest-Assured kütüphanesinden faydalanılmıştır. Rest-Assured Java'ya özgü bir araç olup XML ve JSON web servis yanıtlarının ayrıştırılmasını sağlamaktadır. Servis yanıt kodu, yanıt satırı veya geri dönen yanıtın veri değerine göre çeşitli şekillerde test etmeyi desteklemektedir. İlk olarak Şekil 3.13'de tanıtılan ve tablodaki tüm kullanıcıların web sayfasında gösterilmesini sağlayan servis metodu test edilmiştir.

```
import org.testng.Assert;

public class TEST_GET_ALL_USERS {

    @Test
    public void getAllUsers()
    {
        //writing base URL
        RestAssured.baseURI = "http://localhost:8080/ServiceProject/webservice/getUserService";

        RequestSpecification httpRequest = RestAssured.given();

        Response response = httpRequest.request(Method.GET,"/getAllUsers");

        String responseBody = response.getBody().asString();
        System.out.println("Response body is "+responseBody);

        int statusCode = response.getStatusCode();
        System.out.println("Status code is "+statusCode);
        Assert.assertEquals(statusCode, 200);

        String statusLine = response.getStatusLine();
        System.out.println("Status line is "+statusLine);
        Assert.assertEquals(statusLine, "HTTP/1.1 200 ");

        Assert.assertEquals(responseBody.contains("1"), true);
        Assert.assertEquals(responseBody.contains("25"), true);
        Assert.assertEquals(responseBody.contains("gizem"), true);
        Assert.assertEquals(responseBody.contains("iren"), true);
```

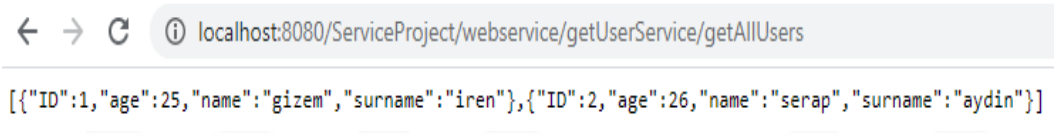
Şekil 3.18. Servis Metodunun Test Edilmesi

Öncelikle Rest-Assured ile test edilecek web servisin linki RestAssured sınıfının baseURI özelliği ile programda tanıtılmaktadır. Daha sonra web servise manuel olarak bir HTTP talep yapılabilmesi için RestAssured sınıfının given metodundan faydalanılmaktadır. Bu metot geriye RequestSpecification tipinden bir değişken döndürmektedir. Bu değişken HTTP talep değişkeni olup web servis sınıfının getAllUsers metodunu çağırmaktadır. Bu metot çalıştığında ise geriye bir servis yanıtı döndürmektedir. Eğer servis metodu başarılı bir şekilde çalıştıysa yanıtın durum kodu 200 olarak ortaya çıkmaktadır. Test işlemine ilk olarak durum kodu ile başlanmıştır. Assert sınıfının eşitlik metodu yardımı ile yanıt durum kodunun 200'e eşit olup olmadığı kontrol edilmiştir. Ardından yanıt durum satırının testi gerçekleştirilmiştir. Yanıt durum satırı kullanıcıya HTTP Protokol versiyonu, durum kodu ve durum mesajı hakkında bilgi vermektedir. Aynı şekilde servis metodu sorunsuz çalıştıysa yanıt durum satırının "HTTP/1.1 200 OK" olması beklenmektedir. Bu kontrol de yine Assert sınıfının eşitlik metodu ile yapılmıştır. Yanıt gövdesi ise kullanıcı tarafından web servisinden talep edilen veri olarak nitelendirilmektedir. Bu veri ise JSON veri formatı tipinde web sayfasında aşağıdaki gibi gösterilmektedir. Son olarak ise yanıt verisi test edilmiş olup beklenen değerleri içerip içermediği incelenmiştir. Testlerin çalıştırılması için TestNG yapısına başvurulmuştur. TestNG açık kaynak otomatik bir test çerçevesi olup JUnit ve NUnit test yapılarından esinlenilerek geliştirilmiştir. Fakat bu test yapıları ile karşılaştırıldığında TestNG'ye yeni işlevsellikler eklendiği için daha güçlü olduğu söylenebilmektedir. Yazılım geliştiricilerin daha esnek ve

güçlü test durumları yazmalarına olanak sağlamaktadır. TestNG çatısı ile çalıştırılıp sonuçlar aşağıdaki gibi çalıştırılmış olup servis metodunun düzgün çalıştığı görülmüştür. Yani testlerimiz hata vermeden geçmiştir.

```
<terminated> TEST_GET_ALL_USERS [TestNG] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (5 Oca 2020 20:44:51)
[RemoteTestNG] detected TestNG version 7.0.0
Response body is [{"ID":1,"age":25,"name":"gizem","surname":"iren"},{"ID":2,"age":26,"name":"serap","surname":"aydin"}]
Status code is 200
Status line is HTTP/1.1 200
PASSED: getAllUsers

===============================================
    Default test
    Tests run: 1, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 1, Passes: 1, Failures: 0, Skips: 0
===============================================
```

Şekil 3.19. Test Sonuçlarının Gösterilmesi



Şekil 3.20. Web Servis Yanıtının Görüntülenmesi

Yine web servis testi kapsamında farklı metotların testleri gerçekleştirilmiştir. Örnek olarak yaşı 25'e eşit olan tüm kullanıcıların aşağıdaki web servis metodu ile getirilmesi kontrol edilmiştir.

```
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("/age/{age}")
public List<UserModel> start(@PathParam("age") int userAge) throws Exception {

    wso = new WebServiceOperations();
    businessUtil = new BusinessUtil();

    ResultSet myRs = wso.queryAllUsersByAge(userAge);
    businessUtil.addUser(myRs);
    return businessUtil.getUserList();
}
```

Şekil 3.21. Yaş Kriterine Göre Kullanıcıları Çeken Web Servis Metodu

Web sayfasında uygun URL yazarak yaşı 25'e eşit olan tüm kullanıcılar aşağıdaki gibi başarılı bir şekilde getirilmiştir.

Şekil 3.22. Yaş Kriteri ile İlişkili Web Servis Metodunun Çalıştırılması

Rest Assured kullanılarak ilgili metot Şekil 3.23'deki gibi test edilmiştir.

```
public class TEST_GET_USERS_BY_AGE {
        @Test
        public void getAllUsersByAge()
        {
            //writing base URL
            RestAssured.baseURI = "http://localhost:8080/ServiceProject/wel

            RequestSpecification httpRequest = RestAssured.given();

            Response response = httpRequest.request(Method.GET,"/age/25");

            String responseBody = response.getBody().asString();
            System.out.println("Response body is "+responseBody);

            int statusCode = response.getStatusCode();
            System.out.println("Status code is "+statusCode);
            Assert.assertEquals(statusCode, 200);

            String statusLine = response.getStatusLine();
            System.out.println("Status line is "+statusLine);
            Assert.assertEquals(statusLine, "HTTP/1.1 200 ");

            Assert.assertEquals(responseBody.contains("1"), true);
            Assert.assertEquals(responseBody.contains("25"), true);
            Assert.assertEquals(responseBody.contains("gizem"), true);
            Assert.assertEquals(responseBody.contains("iren"), true);

            Assert.assertEquals(responseBody.contains("4"), true);
            Assert.assertEquals(responseBody.contains("25"), true);
            Assert.assertEquals(responseBody.contains("sema"), true);
            Assert.assertEquals(responseBody.contains("ok"), true);
        }
```

Şekil 3.23. Yaş Kriteri ile İlişkili Web Servis Metodunun Test Edilmesi

Testler başarılı bir şekilde geçmiş olup aşağıdaki gibi sonuçlar görülmektedir.

```
<terminated> TEST_GET_USERS_BY_AGE [TestNG] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (14 Oca 2020 10:40:46)
[RemoteTestNG] detected TestNG version 7.0.0
Response body is [{"ID":1,"age":25,"name":"gizem","surname":"iren"},{"ID":4,"age":25,"name":"sema","surname":"ok"}]
Status code is 200
Status line is HTTP/1.1 200
PASSED: getAllUsersByAge

===============================================
    Default test
    Tests run: 1, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 1, Passes: 1, Failures: 0, Skips: 0
===============================================
```

Şekil 3.24. Yaş Kriteri ile İlişkili Web Servis Metodunun Test Sonuçları

Test sonucunun başarısız olduğunu görmek geçersiz bir değer kontrol edilmiştir (kullanıcı ismi "yıldız" olarak kontrol edilmiştir). Servis yanıtı, değeri içermediğinden test başarısız olarak sonuçlandırılacaktır.

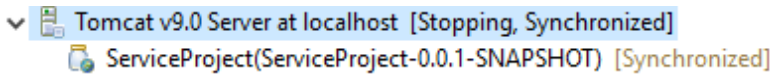
```
Assert.assertEquals(responseBody.contains("1"), true);
Assert.assertEquals(responseBody.contains("25"), true);
Assert.assertEquals(responseBody.contains("yıldız"), true);
Assert.assertEquals(responseBody.contains("iren"), true);
```

Şekil 3.25 Testte Geçersiz Değer Kontrolünün Yapılması

```
[RemoteTestNG] detected TestNG version 7.0.0
Response body is [{"ID":1,"age":25,"name":"gizem","surname":"iren"},{"ID":4,"age":25,"name":"sema","surname":"ok"}]
Status code is 200
Status line is HTTP/1.1 200
FAILED: getAllUsersByAge
java.lang.AssertionError: expected [true] but found [false]

===============================================
    Default test
    Tests run: 1, Failures: 1, Skips: 0
===============================================
```

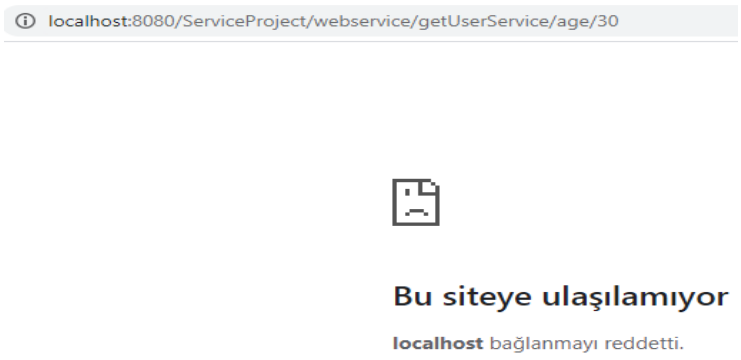
Şekil 3.26. Test Sonucunun Başarısız Olması

Ek olarak sunucuda herhangi bir problem olduğunda da web servis yanıt veremeyeceğinden yanıt kodu 200'e eşit olmayacaktır. Bu durum da test edilmiştir. Projenin sunucusu Apache Tomcat olup testler çalıştırılmadan önce sunucunun bağlantısı kesilmiştir.



Şekil 3.27. Sunucu Bağlantısının Kesilmesi

Tekrar web sayfasında herhangi bir servis metodunun linki yazıldığında geriye aşağıdaki gibi hata sayfası dönmektedir.



Şekil 3.28. Web Servise Ulaşılamaması

Aynı şekilde servis metodu test edildiğinde aşağıdaki gibi başarısız olmuştur.

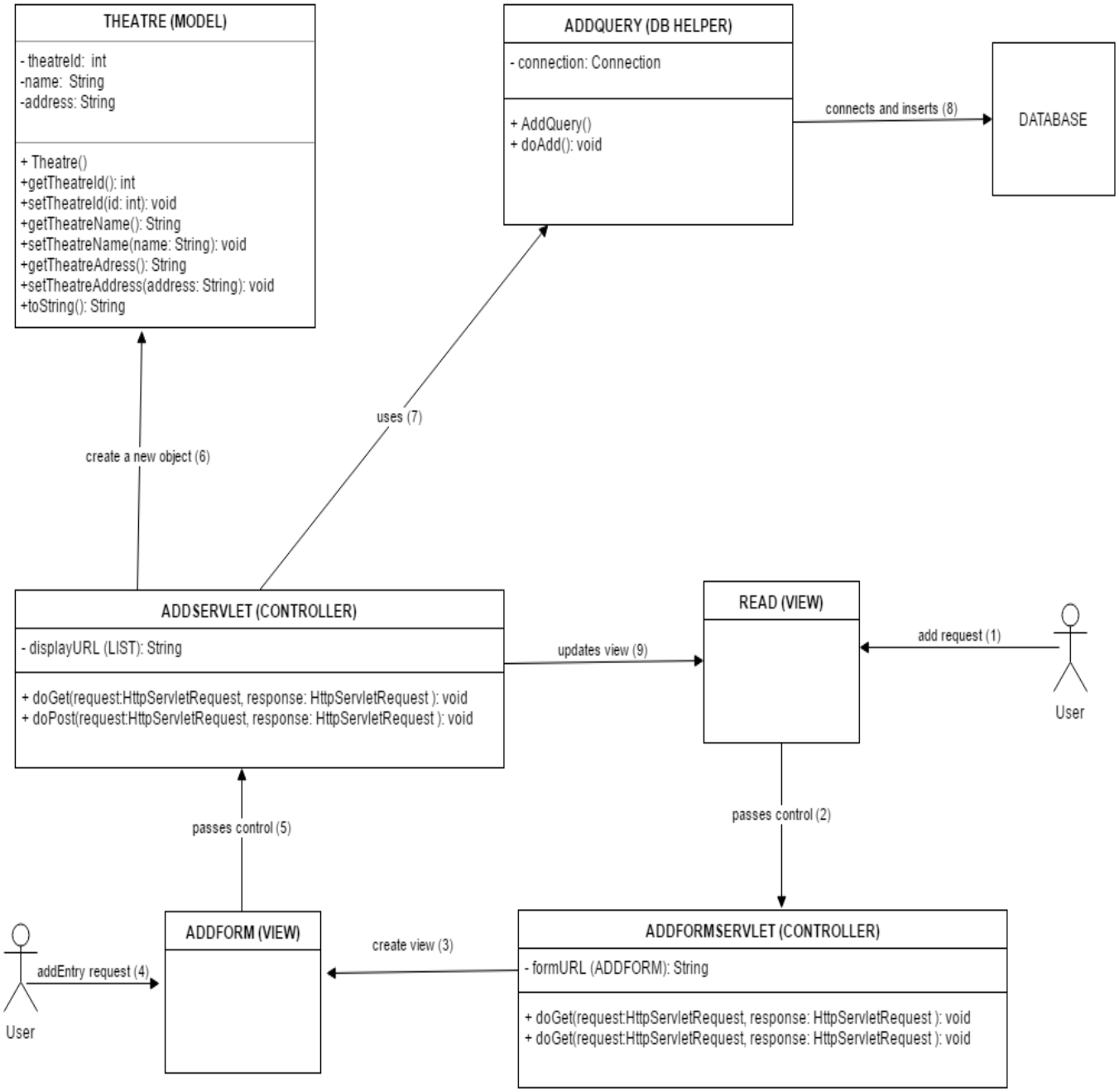
```
<terminated> TEST_GET_USERS_BY_AGE [TestNG] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (14 Oca 2020 11:29:45)
[RemoteTestNG] detected TestNG version 7.0.0
FAILED: getAllUsersByAge
java.net.ConnectException: Connection refused: connect

===============================================
    Default test
    Tests run: 1, Failures: 1, Skips: 0
===============================================
```

Şekil 3.29. Servis Metodu Test Sonucunun Başarısız Olması

3.2 Selenium Tabanlı Fonksiyonel Testlerin Gerçekleştirilmesi

Tez kapsamında fonksiyonel testlerin otomatize edilip gerçekleştirilmesi için Selenium test aracı kullanılmıştır. Java dilinde yazılmış bir Web uygulaması üzerinden ekleme ve çıkarma işlemleri test edilmiştir. Web uygulaması Java sunucu sayfaları ve Servlet sınıfları ile oluşturulmuş olup ekleme işleminin mimarisi aşağıdaki sınıf diyagramı ile gösterilmiştir. Buna göre adımlar aşağıdaki şekildedir:

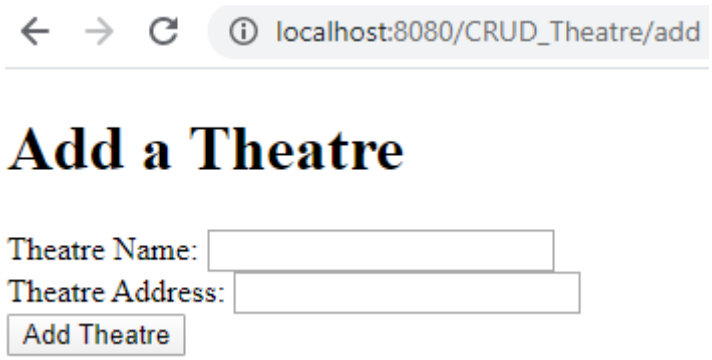
1. Kullanıcı tüm tiyatroların listelendiği web sayfasından (read.jsp) yeni bir tiyatro ekleme talebinde bulunmaktadır.
2. Talep yapıldıktan sonra tiyatro ekleme işleminin yapıldığı Formu açacak olan Servlet (AddFormServlet.java) sınıfına kontrol geçmektedir.
3. Servlet sorumlu olduğu formu (addForm.jsp) açtıktan sonra kullanıcı sayfadan tiyatro nesnesine ait bilgileri girmektedir.
4. Bilgileri girip ekleme butonuna bastıktan sonra kontrol veritabanına ekleme yapacak olan Servlet'e geçmektedir (AddServlet.java)
5. Bu Servlet içinde kullanıcının girdiği bilgilerle Tiyatro sınıfından yeni bir Tiyatro nesnesi yaratılıp veritabanına ekleme işlemi ilgili veritabanı sınıfı içinde yapılmaktadır (AddQuery.java).



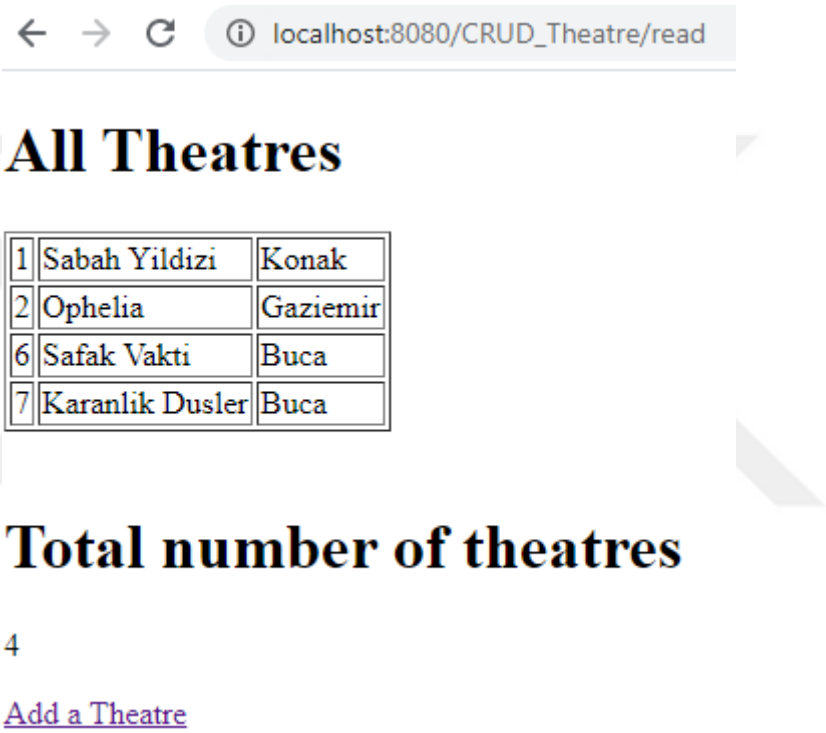
Şekil 3.30. Web Uygulaması Ekleme İşleminin Mimarisi

6. Veritabanına ekleme işlemi başarılı bir şekilde yapıldıktan sonra kullanıcı yine tüm tiyatroların listelendiği sayfaya yönlendirilmektedir (read.jsp).

Ekleme işleminin yapıldığı form ve tüm tiyatroları gösteren sayfa aşağıdaki gibi görülmektedir.



Şekil 3.31. Tiyatro Ekleme Formu



Şekil 3.32. Tüm Tiyatroların Listelendiği Web Sayfası

Projede her bir işlemi test eden ayrı ayrı test sınıfları oluşturulmuştur. Ekleme işlemini test eden sınıf aşağıda görülmektedir. Selenium ile test yaparken öncelikle web sayfasının otomatik olarak açılması gerekmektedir. Bu işlemi ise Şekil 3.33'de görülen **openApplication** metodu yürütmektedir. İlgili sayfanın Google Chrome tarayıcıda açılabilmesi için chromedriver çalıştırılabilir dosya yolunun projede tanıtılması gerekmektedir. Daha sonra ChromeDriver sınıfından bir nesne oluşturularak açılacak sayfanın linki verilmiştir. Sayfa açıldıktan sonra formda yine otomatik bir şekilde textbox'lara değer göndermek için **addTheatre** metodu yazılmıştır. Her bir textbox'a değer atanırken textbox'ların isim özelliğinden faydalanılmıştır. Textbox'lara değer gönderildikten sonra ise ekleme butonu id ile sayfada aratılıp ardından hazır click metodu çağırılmıştır.

```
public class TEST_THEATRE_ADD {

    public static WebDriver driver;

    public static void openApplication() {

      System.setProperty("webdriver.chrome.driver","C:/Users/gizem.iren/Downloads/chromedriver.exe");
      driver = new ChromeDriver();
      driver.get("http://localhost:8080/CRUD_Theatre/add");
    }
```

Şekil 3.33. Ekleme İşlemini Test Eden Sınıf

```
    public static void addTheatre()
    {
      driver.findElement(By.name("theatreName")).sendKeys("Gecenin Kuşları");
      driver.findElement(By.name("theatreAddress")).sendKeys("Alsancak");
      driver.findElement(By.id("addBtn")).click();
    }
```

Şekil 3.34. Ekleme İşlemini Otomatik Yapan Metot

Son olarak main metodu içinde işlem testinin geçip geçmediği test edilmiştir. Bu test ise tiyatroları listeleyen sayfada tüm kayıtların sayacını tutan web elemanına bağlı olarak yapılmaktadır. Web sayfası açılıp tiyatro eklendikten sonra tüm tiyatroları listeleyen sayfaya kullanıcı yönlendirilmektedir. Fakat kullanıcının yönlendirildiği sayfa internet hızına göre hemen açılmayabilir. Açılmadığı için de sayfada var olan web elemanına erişilemez. Bu durumu önlemek için Selenium'da var olan bekleme metotlarına başvurulmuştur. Ağ koşullarında herhangi bir aksilik olduğunda bir sonraki sayfanın açılıp ilgili web elemanının görünürlüğü için belirli bir süre beklenecektir. Bu bekleme ise maksimum 10 saniye olacaktır.

```
    public static void main(String[] args) throws InterruptedException {

        //Open the application
        openApplication();

        //Send values to textboxes
        addTheatre();

        //Wait for the page to load by 15 seconds
        WebElement countOfRecords = driver.findElement(By.id("rowCount"));
        WebDriverWait wait = new WebDriverWait(driver,10);
        wait.until(ExpectedConditions.visibilityOf(countOfRecords));

        int valueOfCountOfRecords = Integer.parseInt(countOfRecords.getText());

        System.out.println(countOfRecords.getText());

        if (valueOfCountOfRecords == 5)
        {
          System.out.println("Test Case: "+ "Theatre Add Operation Passed");
        }

        else
        {
          System.out.println("Test Case: "+ "Theatre Add Operation Failed");
        }
    }
```

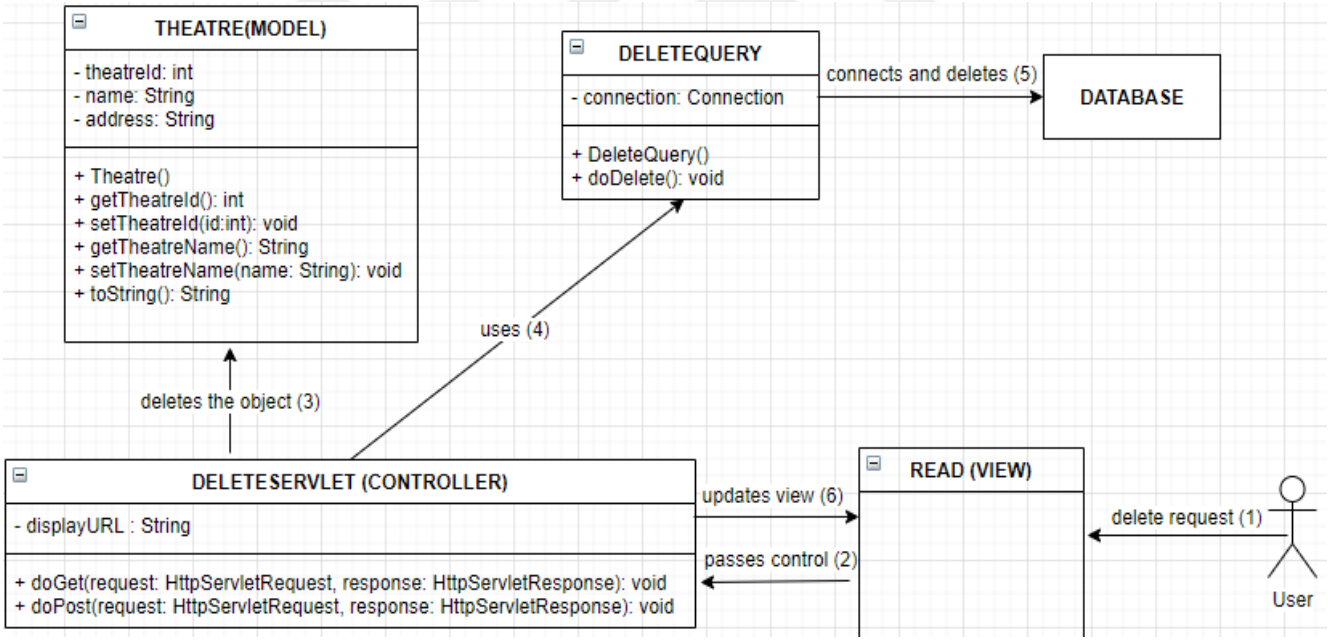
Şekil 3.35. Ekleme İşleminin Test Süreci ve Geçme Koşulu

Web sayfası başarılı bir şekilde açılırsa tüm tiyatroların sayacını tutan web elemanı ile beklenen toplam değer birbiriyle karşılaştırılmakta ve eğer sonuçlar birbirine eşit ise test başarılı bir şekilde sonuçlanmaktadır. Sonuçlar birbirine eşit değilse ekleme işleminde bir problem olduğu anlaşılacak ve test başarısız olarak sonuçlanacaktır. Ekleme işlemi testinin başarılı sonucu ise aşağıda gösterilmektedir.

```
<terminated> TEST_THEATRE_ADD [Java Application]
Starting ChromeDriver 79.0.3945.36 (3582d
Only local connections are allowed.
Please protect ports used by ChromeDriver
Oca 23, 2020 5:14:24 PM org.openqa.seleni
INFO: Detected dialect: W3C
5
Test Case: Theatre Add Operation Passed
```

Şekil 3.36. Ekleme İşleminin Test Sonucunun Başarılı Olması

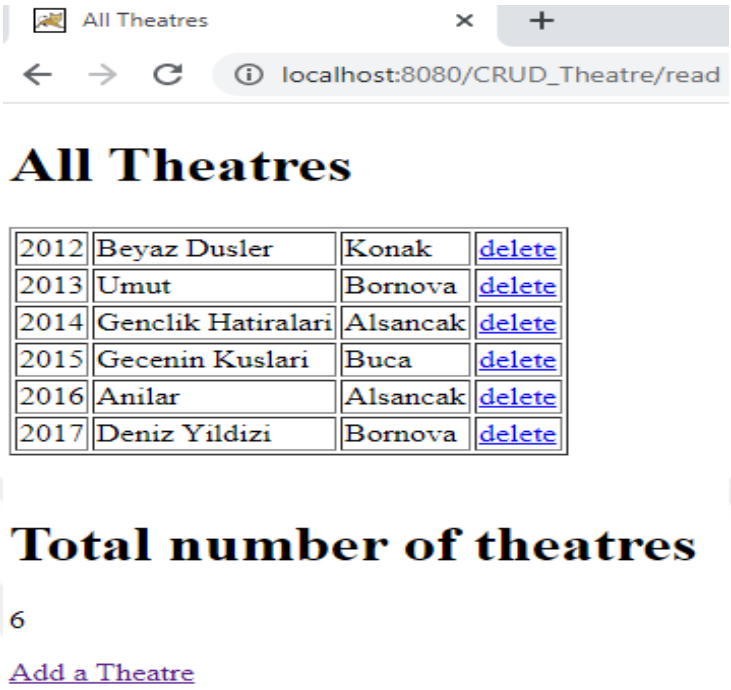
Aynı şekilde web sayfasından belirli bir tiyatro silme işlemi de test edilmiştir. Silme işleminin çalışma mekanizması Şekil 3.37'den görülmektedir.



Şekil 3.37. Web Uygulamasında Silme İşleminin Çalışma Mekanizması

1. Kullanıcı tüm tiyatroların listelendiği web sayfasından (read.jsp) spesifik bir tiyatroyu silme talebinde bulunmaktadır.
2. Talep yapıldıktan sonra Servlet (DeleteServlet.java) sınıfına kontrol geçmektedir.
3. Bu Servlet içinde veritabanı işlem sınıfının metodu çağrılarak ilgili kaydın tablodan silinmesi sağlanmaktadır.

Uygulamayı çalıştırınca yine aşağıdaki gibi tüm kayıtların listelendiği web sayfası görülmektedir.



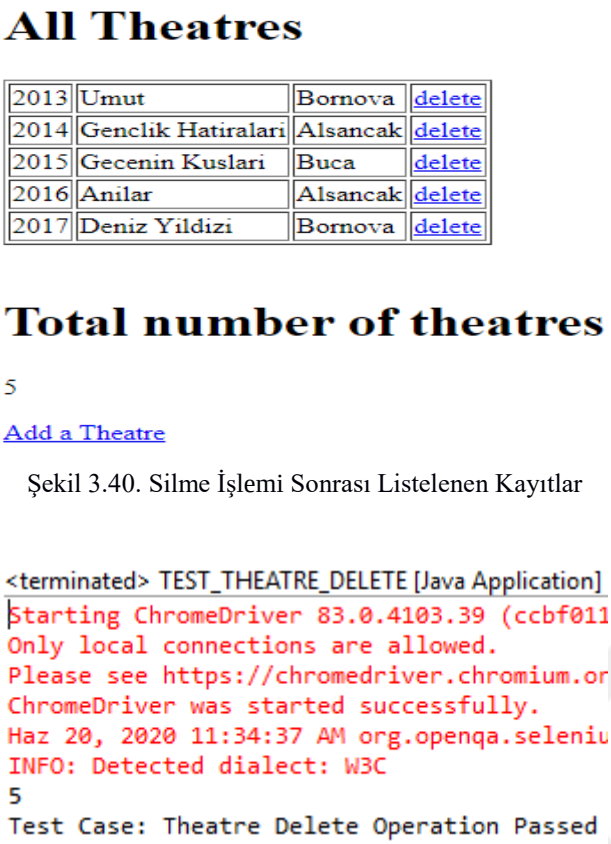
Şekil 3.38. Tüm Kayıtların Listelendiği Web Sayfası

Silme işleminin test edilmesi için yine ayrı bir sınıf oluşturulmuş olup sınıfın ana metodu aşağıda görülmektedir. Test için web sayfasında görülen ikinci kayıt kullanılmıştır. Bu kapsamda Selenium Web Driver'ın web elemanlarına ulaşabilmemiz için bize sağlamış olduğu hazır metotlardan faydalanılmıştır. Silme işlemi kullanıcının LinkText web elemanına basmasıyla yapıldığı için öncelikle etiket ismi aracılığı ile sayfadaki tüm LinkText web elemanlarına ulaşılmış ve ilk olanı tıklanmak üzere seçilmiştir.

```
public static void deleteTheatre()
{
  List<WebElement> linkList = driver.findElements(By.tagName("a"));
  linkList.get(0).click();
}
```

Şekil 3.39. Delete LinkText Butonuna Otomatik Basılması

Id'si 2012 olan kayıt (Beyaz Düşler) silindiğinde yine aşağıdaki gibi ilgili kaydın başarılı bir şekilde silindiği ve testin de başarılı bir şekilde geçtiği görülmektedir (Şekil 3.40 ve Şekil 3.41).



Şekil 3.40. Silme İşlemi Sonrası Listelenen Kayıtlar

Şekil 3.41. Silme İşleminin Test Sonucunun Başarılı Olması

Test sınıfının main metodunda ise bir önceki test sınıfında olduğu gibi yine uygulama otomatik olarak açılmakta (openApplication metodu) ve silme butonuna otomatik olarak tıklanmaktadır (deleteTheatre metodu). Testin geçip geçmeyeceği ise yine web sayfasında gösterilen toplam kayıt sayısına göre belirlenmektedir.

```
public static void main(String[] args) throws InterruptedException {

    //Open the application
    openApplication();

    //click first delete link button (delete first record)
    deleteTheatre();

    //Wait for the page to load by 10 seconds
    WebElement countOfRecords = driver.findElement(By.id("rowCount"));
    WebDriverWait wait = new WebDriverWait(driver,10);
    wait.until(ExpectedConditions.visibilityOf(countOfRecords));

    int valueOfCountOfRecords = Integer.parseInt(countOfRecords.getText());

    System.out.println(countOfRecords.getText());

    if (valueOfCountOfRecords == 5)
    {
      System.out.println("Test Case: "+ "Theatre Delete Operation Passed");
    }

    else
    {
      System.out.println("Test Case: "+ "Theatre Delete Operation Failed");
    }
```

Şekil 3.42. Silme İşleminin Test Süreci ve Geçme Koşulu

3.3 Selenium ve Apache POI ile Veri Tabanlı Testlerin Gerçekleştirilmesi

3.3.1 Apache POI Nedir ?

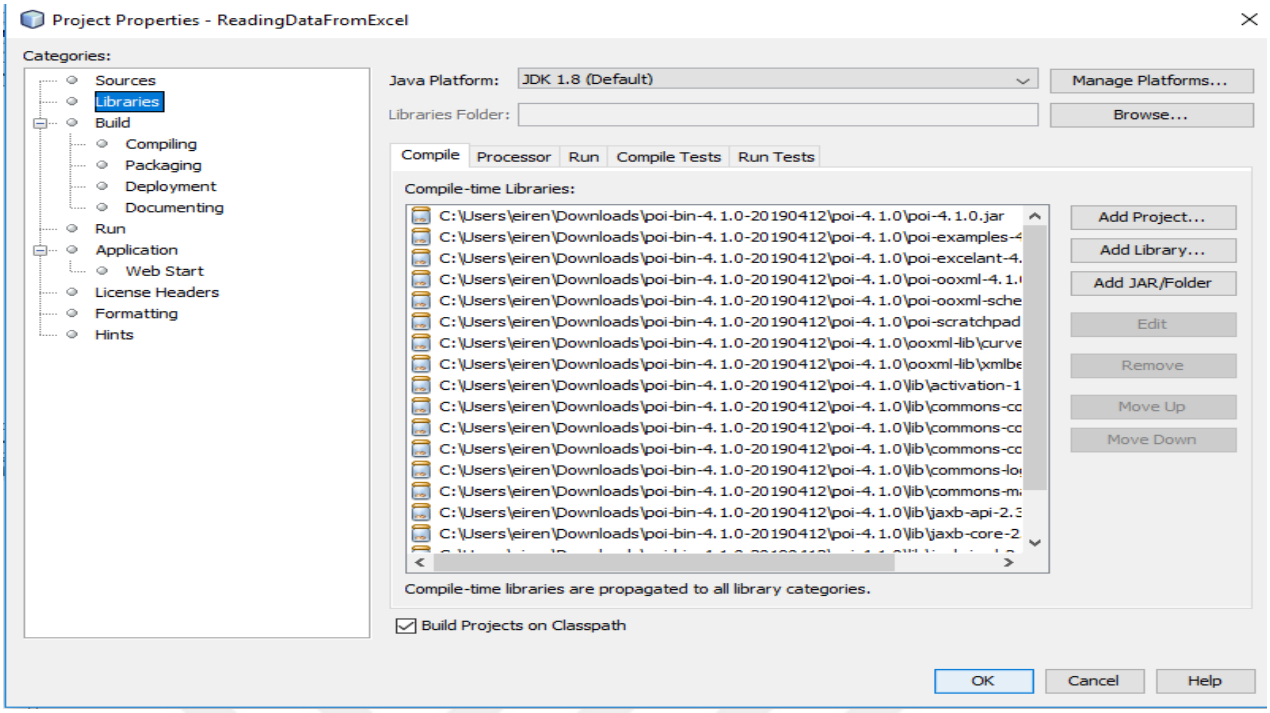
Apache POI, programcıların Java programlarını kullanarak MS Office dosyaları oluşturmasına, değiştirmesine ve görüntülemesine izin veren popüler bir API'dir. Açık kaynak kodlu bir kütüphane olup Apache Yazılım Kuruluşu tarafından geliştirilmekte ve dağıtılmaktadır. Apache POI, MS Office belgeleri ile çalışabilmek için değişik sınıf ve metotlar barındırmaktadır. Servis aşağıdaki gibi farklı bileşenlere ayrılmaktadır:

- **HSSF (Horrible Spreadsheet Formatı):** MS Excel dosyalarının xls biçimlerini okuyup yazabilmek için kullanılmaktadır.
- **XSSF (XML Spreadsheet Formatı):** MS Excel dosyalarının xlsx biçimlerini okuyup yazabilmek için kullanılmaktadır.
- **HWPF (Horrible Word Processor Formatı):** MS Word dosyalarının doc biçimlerini okuyup yazabilmek için kullanılmaktadır.
- **XWPF (XML Word Processor Formatı):** MS Word dosyalarının docx biçimlerini okuyup yazabilmek için kullanılmaktadır.
- **HSLF (Horrible Slide Layout Formatı):** Powerpoint gösterimlerini okumak, güncellemek veya oluşturmak için kullanılmaktadır.
- **HDGF (Horrible DiaGram Formatı):** MS- Visio dosyaları için gerekli metot ve sınıfları içermektedir.

Apache POI gibi farklı firmalar tarafından piyasaya sürülen araçlar da mevcuttur. Bu araçlar Aspose cells for Java (Aspose), JXL (Common Libraries) ve JExcel (Team Dev) olarak sıralanabilir (Tutorialspoint, 2020).

3.3.2 Selenium ile Veri Tabanlı Excel Testlerinin Gerçekleştirilmesi

Veri tabanlı Excel testlerini gerçekleştirmek üzere Netbeans aracında yeni bir proje oluşturulmuştur. Projenin build dizinine ilgili jar dosyalarının Şekil 3.44'de görüldüğü gibi eklenmesi gerekmektedir. Ek olarak testlerin gerçekleştirimi için yine TestNG aracına başvurulmuştur. Test etmek üzere bir Excel listesi oluşturulmuştur. Bu listede kullanıcı mail adresi ve şifre gibi bilgiler tutulmaktadır. Projede Excel dosyasından kullanıcı mail adresleri ve şifreler sırasıyla okunarak Gmail giriş sayfasına yönlendirilmektedir. Excel verilerini proje içerisinden okumak için Apache POI servisinden faydalanılmıştır. Verilerin doğruluğunu test ederken Selenium kütüphanesinin metotları kullanılmıştır.



Şekil 3.43. Apache POI Kütüphanesinin Projeye Eklenmesi

Excel dosyasından ilgili verileri çekebilmek için aşağıdaki gibi ayrı bir Excel işlem sınıfı oluşturulmuştur. Bu sınıfta hedef Excel dosyasına ve verilerin bulunduğu çalışma kâğıdına erişmek için Apache POI kütüphanesinde tanımlanmış sınıflardan bazı değişkenler yaratılmıştır. Sınıfın yapıcı metoduna Excel dosyasının dizini ve Excel çalışma kağıdının parametreleri verilmiştir. Kullanıcı bu sınıftan bir nesne oluştururken uygun parametre değerlerini belirtmektedir. Metot içinde ise Excel dosya dizini parametresi ile FileInputStream nesnesi tanımlanıp ardından bu nesne ile Excel çalışma kitabına ulaşılmaktadır. Daha sonra Excel çalışma kitabı nesnesi yardımı ile verilerin saklandığı çalışma kâğıdına erişilmektedir.

```
public class ApachePOIExcelOperations {

        public String path;
        public FileInputStream fis = null;
        public FileOutputStream fileOut = null;
        private XSSFWorkbook workbook = null;
        private XSSFSheet sheet = null;

        private Object[][] excelData;

        public ApachePOIExcelOperations(String path, String sheetName) {

                this.path = path;
                try {
                        fis = new FileInputStream(path);
                        workbook = new XSSFWorkbook(fis);
                        sheet = workbook.getSheet(sheetName);
                }

                catch (Exception e) {
                        e.printStackTrace();
                }
        }
```

Şekil 3.44. Excel İşlem Sınıfının Oluşturulması

Bu bölümde Excel'de bulunan tüm veriler iki boyutlu bir nesne dizisinde tutulmaktadır. Döngü ile her bir satıra ait tüm sütun değerleri çekilmiş olup metot içinde bu dizi geri döndürülmektedir.

```
public Object[][] getExcelData() throws IOException
{
      int rowNumber = sheet.getLastRowNum();
      int columnNumber = sheet.getRow(0).getLastCellNum();
      excelData = new Object[rowNumber][columnNumber];

      for(int i= 0; i<rowNumber;i++)
      {
          for(int j = 0;j<columnNumber;j++)
          {
              System.out.println(sheet.getRow(rowNumber).getCell(j).getStringCellValue());
              excelData[i][j] = sheet.getRow(rowNumber).getCell(j).getStringCellValue();
          }
      }
      workbook.close();
      fis.close();
      return excelData;
}
```

Şekil 3.45. Excel Verilerinin Çekilmesi

Veri temelli test için ayrı bir test sınıfı oluşturulmuş olup Şekil 3.46'dan gözükmektedir. Burada ChromeDriver tanımlanarak uygulamanın otomatik olarak yönlendirileceği web sayfası belirtilmiştir. TestNG'de test otomasyonu öncesi çalıştırılacak bir metot var ise ilgili metodun üzerine @BeforeTest notasyonu ekleyerek bu gerçekleştirilmektedir. Test öncesi gerekli ayarlamaların yapıldığı init metodu test öncesi çalıştırılacağı için bu notasyon metot başına eklenmiştir.

```
public class DataDrivenTest {

    WebDriver driver;

    @BeforeMethod
    public void init()
    {
        System.setProperty("webdriver.chrome.driver","C:\\Users\\eiren\\Downloads\\chromedriver.exe");
        driver = new ChromeDriver();
        driver.get("http://gmail.com");
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    }
```

Şekil 3.46. Veri Temelli Test Sınıfı

Veri temelli test için veri sağlayıcısına ihtiyaç duyulmuş olup bu veri seti Şekil 45'de Excel verileri ile doldurulan dizi ile oluşturulmaktadır. Veri sağlayıcısına özel isim verilmiştir.

```
    @DataProvider(name="ExcelDataForTest")
    public Object[][] getExcelData() throws IOException
    {
        ApachePOIExcelOperations operation = new ApachePOIExcelOperations("Credentials.xlsx","Credentials");
        return operation.getExcelData();
    }
```

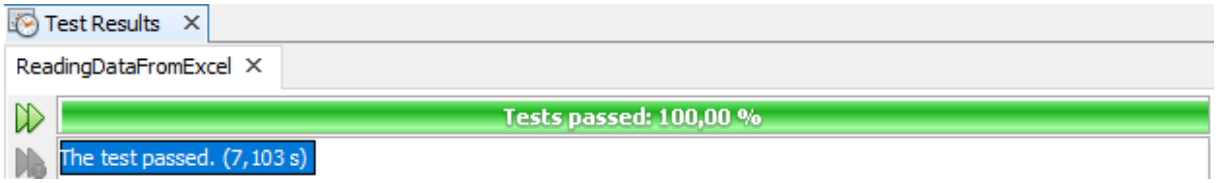
Şekil 3.47. Veri Setinin Oluşturulması

Test ederken hangi veri setinin kullanılacağı test metodundan önce Şekil 3.48'deki gibi veri seti ismi ile belirtilmiştir. Daha sonra veri setinde bulunan hem kullanıcı adı hem de şifre ile web sayfasına girişin kontrolü için web sayfasında bulunan uygun alanların belirlenmesi gerekmektedir. Bu nokta da yine Selenium'un hazır metotlarından faydalanılmıştır. Alanlar belirlendikten sonra veriler alanlar gönderilmiş olup web sayfasına giriş işlemi otomatik olarak yapılmıştır. Testin başarılı olması ise web sayfasının URL bilgisi ile belirlenmektedir. O anki mevcut URL, beklenen URL ile karşılaştırılmakta ve her ikisi de birbirine eşitse test başarılı olarak kabul edilmektedir. Aksi taktirde test başarısız olacaktır.

```
@Test(dataProvider="ExcelDataForTest")
public void authenticateLogin(String userName, String userPassword) throws InterruptedException
{
   String expectedURL = "https://mail.google.com/mail/u/0/#inbox";
   WebElement email_phone = driver.findElement(By.xpath("//input[@id='identifierId']"));
   email_phone.sendKeys(userName);
   driver.findElement(By.id("identifierNext")).click();
   WebElement password = driver.findElement(By.xpath("//input[@name='password']"));
   Thread.sleep(1000);
   password.sendKeys(userPassword);
   driver.findElement(By.id("passwordNext")).click();
   Thread.sleep(5000);
   String actualURL = driver.getCurrentUrl();
   Assert.assertEquals(actualURL, expectedURL);
}
```

Şekil 3.48. Veri Tabanlı Test Metodu

Test sınıfı çalıştırıldığında testin aşağıdaki gibi başarılı bir şekilde geçtiği görülerek Excel verilerinin geçerli ve doğru veri olduğu anlaşılmıştır.



Şekil 3.49. Veri Tabanlı Testin Başarılı Olması

4. DEĞERLENDİRME VE ÖNERİLER

Tez çalışması kapsamında farklı yönlerden gerçekleştirilebilecek test uygulamaları çalıştırılıp sonuçları incelenmiştir. İlk olarak web servis testi yapabilmek amacıyla web servis uygulaması ve testleri yazılmış olup Rest Assured test aracı ile testler kontrol edilmiştir. Ardından Java Servlet web uygulaması oluşturulmuş olup Selenium otomasyon test aracı ile fonksiyonel testler gerçekleştirilmiştir. Son olarak ise yine Selenium ve Apache POI teknolojileri kullanılarak ve Excel'den ilgili veriler çekilip kontrol edilerek veri temelli testler yapılmıştır. Uygulamalarda başvurulan tüm test araçları ücretsiz olup ihtiyaca göre kullanıcılar tarafından kolay bir şekilde kurulup test amaçlı kullanılabilir.

Yazılım testleri yazılım projelerinde maliyet, zaman ve müşteri memnuniyeti açısından büyük önem arz etmektedir. Yazılımın müşteriye sunulmadan önce kalitesini maksimum düzeyde garanti etmek adına yazılım testlerine zaman ayrılmalıdır. Bu nedenle ürünlerin doğruluğunu ve müşteri beklentilerini karşıladığını görmek için testler önemli rol oynamaktadır. Diğer yandan uygulamada çıkan hatalar firmalara maliyet olarak geri dönmektedir. Çünkü hataların düzeltilmesi masraflara yol açmaktadır. Testlerin maliyetleri minimuma indirmek adına düzgün bir biçimde ve kapsamlı olarak yapılması gerekmektedir. Böylece test uzmanları tarafından belirlenen hatalar ilk aşamalarda düzeltilerek ileriki aşamalara yayılmamaktadır. Bu da zaman ve maliyet açısından kuruma fayda sağlamaktadır. Sonuç olarak geliştirilen yazılım projelerinde mutlaka test uzmanlarına yer verilmeli ve yazılım kullanıcıya çıkmadan önce farklı yönlerden test edilmelidir.

Gerçekleştirilen tez çalışması aynı zamanda test alanında çalışan veya çalışmak isteyen kişilere de bu konu ile ilgili bilgi vermeyi amaçlamaktadır. Çalışmada test çeşitleri ve uygulamaları ayrıntılı bir şekilde anlatılmış olup çalışma ilgilenenler tarafından rehber bir kaynak olarak seçilebilir.

KAYNAKLAR DİZİNİ

Acun, G. ve Bilgin, T.T., 2015, Yazılım Hata Logları Kullanılarak Veri Madenciliği Uygulamasının Gerçekleştirilmesi, Marmara Fen Bilimleri Dergisi, 1:14-20.

Avcıoğlu, A., 2015, Bilgisayar Tabanlı Sistemlerde Test Otomasyonunun Tasarlanması ve Gerçeklenmesi, Yüksek Lisans Tezi, Haccettepe Üniversitesi, Elektrik Elektronik Mühendisliği Anabilim Dalı.

Babbar, H., 2017, Software Testing: Techniques and Test Cases, International Journal of Resarch in Computer Applications and Robotics, 5(3): 44-53.

Büyükyumukoğlu, G., Ersoy, E., Özdemir, M.Ş., Bağrıyanık, S. ve Karahoca, A., 2017, Test Otomasyonunda Karşılaşılan Zorluklar ve Öğrenilen Dersler: Telekomünikasyon Sektöründen Deneyimler, Conference Paper: Turkish National Software Engineering Symposium.

Chintakyala, P., 2013, 'Beginners Guide To Software Testing', pp.1–41.

Chandraprabha, Kumar, A. and Saxena, S., 2015, Data Driven Testing Framework using Selenium WebDriver, International Journal of Computer Applications, 118:18-23.

Chaurasia, N. and Bahl, K., 2018, Analytical Study on Manual vs. Automation Testing Using Selenium, International Journal of Innovative Science, Engineering & Technology, 5(10): 40-43.

Devi, J. ve diğ., 2017, A Study on Functioning of Selenium Automation Testing Structure, International Journal of Advanced Research in Computer Science and Software Engineering, 7(5): 855-862.

Ghuman, S.S, 2014, Software Testing Techniques, International Journal of Computer Science and Mobile Computing, 3(10): 988-993.

Gojare, S., Joshi, R. and Gaigaware, D., 2015, Analysis and Design of Selenium Webdriver Automation Testing Framework, Prodecia Computer Science – 2nd International Symposium on Big Data and Cloud Computing (ISBCC'15), 50: 341-346.

Hooda, I. and Chhillar, R.S., 2015, Software Test Process, Testing Types and Techniques, International Journal of Computer Applications, 111:10-14.

Kılan, N.K., 2015, Yazılım Mühendisliğinin Gelişimine Bir Bakış, Türkiye Bilişim Vakfı Bilgisayar Bilimleri ve Mühendisliği Dergisi (TBV), 8(2): 31-36.

Kökten, F., 2019, Davranış Odaklı Geliştirme Yaklaşımıyla Bir Test Otomasyonunun Geliştirilmesi ve Sürecin Değerlendirilmesi, Yüksek Lisans Tezi, Afyon Kocatepe Üniversitesi Fen Bilimleri Enstitüsü.

Kuday, G., 2014, Yazılım Mühendisliği Yöntemleriyle Yazılım Test Süreci, Yüksek Lisans Tezi, Haliç Üniversitesi Fen Bilimleri Enstitüsü.

Kuntei P. and Mane, D., 2017, Automation Testing of Web based application with Selenium and HP UFT (QTP), International Research Journal of Engineering and Technology (IRJET), 4(6):2579-2583.

Lariya, S., Shrivastava, S. and Nemai S., 2018, Automation Testing using Selenium Web Driver & Behavior Driven Development (BDD), International Journal for Scientific Research & Development (IJSRD), 6(3): 2014-2017.

Luo, L., 2001, Software Testing Techniques, Technology Maturation and Research Strategy Class Report for 17-939A.

Madan, S. and Kakkar, A., 2017, Test Automation as Framework for Web Applications, Indian Journal of Computer Science and Engineering (IJCSE), 8(3): 479-486.

Mahajan, P., 2016, Different Types of Testing in Software Testing, International Research Journal of Engineering and Technology (IRJET), 3(4): 1661-1664.

Neethidevan, V. and Chandrasekaran, G., 2018, Database Testing using Selenium Web Driver – A Case Study, International Journal of Pure and Applied Mathematics, 118: 559-566.

Quan, D. M., 2018, Implementing test automation with Selenium WebDriver, Lathi University of Applied Sciences, Degree Programme in Business Information Technology, Bachelor's Thesis.

Ramya, P., Sindhura, V. and Sagar, P.V., 2017, Testing using Selenium Web Driver, 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT), 1-7.

Sabastian Raju, J. and Vaidhehi, V., 2017, Design and Implementation of Hybrid Test Automation Framework for Web Based Application, International Journal of Innovative Research in Advanced Engineering (IJIRAE), 4(3):108-112.

Saini, G. and Rai, K., 2013, An Analysis on Objectives, Importance and Types of Software Testing, International Journal of Computer Science and Mobile Computing, 2(9): 18-23.

Sharma L. 2017, "What is Rest ?", https://www.toolsqa.com/rest-assured/what-is-rest/ (Erişim Tarihi: 17.01.2020)

Sharma, M. and Angmo, R., 2014, Web-based automation testing and tools, International Journal of Computer Science and Information Technologies, 5:908-912.

Singla, S. and Kaur, H., 2014, Selenium Keyword Driven Automation Testing Framework, International Journal of Advanced Research in Computer Science and Software Engineering Issn, 2277, 6: 125-130.

Soni, A. and Ranga V., 2019, API Features Individualizing of Web Services: REST and SOAP, International Journal of Innovative Technology and Exploring Engineering (IJITEE), 8(9S): 664-671.

Statista, 2019, Number of internet users worldwide from 2005 to 3019, Retrieved from https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/

Sualim, S.A., Yassin, N.M. and Mohamad, R., 2016, Comparative Evaluation of Automated User Acceptance Testing Tool for Web Based Application, International Journal of Software Engineering and Technology, 2(2): 1-6.

Timah, V. 2011. "Where Software Meets Hardware", http://csebrules.blogspot.com/2011/01/assignment-2-task-2-prototyping-model.html (Erişim Tarihi: 04.02.2020)

Tuteja, M. and Dubey, G., 2012, A Research Study on importance of Testing and Quality Assurance in Software Development Life Cycle (SDLC) Models, International Journal of Soft Computing and Engineering (IJSCE), 2(3): 251-257.

Tutorialspoint. 2020. "Apache POI - Overview", https://www.tutorialspoint.com/apache_poi/apache_poi_overview.htm (Erişim Tarihi: 04.02.2020)

Yadav, P. and Kumar, A., 2015, An Automation Testing Using Selenium Tool, International Journal of Emerging Trends and Technology in Computer Science (IJETTCS), 4:5(2): 68-71.

TEŞEKKÜR

Bu çalışma süresince kıymetli görüşlerinden yararlandığım, yakın ilgisini esirgemeyen ve tezin biçimlenmesinde yardımcı olan sayın Prof. Dr. Aylin Kantarcı'ya, teşekkürü bir borç bilirim. Ayrıca tez çalışması boyunca bana sürekli destek veren aileme de teşekkürlerimi sunarım.

15/06/2020

Gizem İREN

ÖZGEÇMİŞ

Ad Soyad : Gizem İREN

Doğum tarihi : 21.09.1994

Doğum yeri : Trabzon (TR)

Adres :Emlak Bankası Konutları 668. Sok. Ata-2 Sitesi No:2 3.Etap Evleri K.5 D.12, Gaziemir/İzmir

Telefon :(+90) 539 209 83 66

E-mail : gizem.iren@hotmail.com

Eğitim :

- 2012–2017 İzmir Ekonomi Bilgisayar Mühendisliği Bölümü
- 2008–2012 Konak Anadolu Lisesi

İş Tecrübeleri :

- 08/2018 – 09/2019 Yazılım Destek Uzmanı (Etiya)
- 09/2019 – halen Yazılım Kalite ve Test Mühendisi (Veripark)