**API**

1. **API Nedir?**

Bağlantı anlamına gelir. Yani API, istekleri alan ve bir sisteme ne yapmak istediğinizi söyleyen habercidir ve daha sonra yanıtı size geri döndürür.

API, Uygulama Programlama Arayüzü’ nün (yazılım aracısı olan) kısaltmasıdır.

API'nin açılımı olan Application Programming Interface, Uygulama Programlama Arabirimi anlamına gelir. API'ler bağlamında Uygulama sözcüğü, ayrı bir işlevi bulunan her türlü yazılımı ifade eder. Arabirim, iki uygulama arasındaki hizmet sözleşmesi gibi düşünülebilir. Bu sözleşme, ikisinin istekler ve yanıtlar kullanarak birbiriyle nasıl iletişim kuracağını tanımlar. İlişkili API belgeleri, geliştiricilerin bu istek ve yanıtları nasıl yapılandırması gerektiğine dair bilgiler içerir.

1. **API vs Webservices?**

Bir API (Uygulama Programlama Arayüzü), yazılım uygulamaları oluşturmak ve entegre etmek için bir dizi kural ve protokoldür. Yazılım bileşenlerinin nasıl etkileşime girmesi gerektiğini belirtir ve API'ler farklı sistemler arasında iletişime izin verir.

Öte yandan bir web hizmeti, bir ağ üzerinden makineler arası birlikte çalışabilir iletişimi desteklemek üzere tasarlanmış bir yazılım sistemidir. Makine tarafından işlenebilir bir formatta (özellikle WSDL) tanımlanan bir arayüze sahiptir.

* Daha basit bir ifadeyle, bir web hizmeti farklı uygulamaların birbirleriyle konuşmasına ve kendi aralarında veri ve hizmet paylaşmasına izin verirken, bir API bir geliştiricinin bir uygulama veya hizmetin işlevselliğine erişmesine izin verir.

API = browser: Selenium WebDriver, database: JDBC, MsOffice: Apache POI

Webservices = eğer bir API iletişim için interneti kullanıyorsa, bu bir webservices'tir.

\*Tüm web servisleri API'dir.

UI (kullanıcı arayüzü) yok -> UI ile web uygulaması ve Selenium Webdriver kullanıyoruz

● We use:

○ SOAP → XML

○ REST → JSON, XML, TEXT

○ Postman, Rest Assured Library

**API (Application Programming Interface)**

=> Uygulama Programlama Arayüzü

- API testi, bir uygulamanın işlevselliği, güvenirliği, performansı ve güvenliği açısından beklentileri karşılayıp karşılamadığını kontrol etmek için yapılır.

**-> API ve Web Service Arasındaki Fark?**

Api ve Web Service, uygulamalar arasında iletişim oluştururlar

**Web Service**

- Aralarındaki tek fark Web Service iki uygulama arasındaki iletişimi INTERNET aracılığı ile yapar.

**API**

- Microsoft Word gibi masaüstü uygulamaları, INTERNETE'e ihtiyaç duymadan kendi API'larını kullanır.

Bu yüzden Word kullandığımızda Microsoft API kullanıyoruz.

=> NOT !!!

Bütün Web Service'ler API'dir. Ancak bütün API'lar Web Service DEGILDIR.

Ortak HTTP Status Kodlar

200(OK) => Talep karşılandı. Request gerçekleşti.

201(CREATED) => Database'e gönderilen datanın oluşturulduğunu söyler (200 olarak da gelebilir)

204(NO CONTENT) => Talep yerine getirildi ancak size cevap (response) dönemiyoruz

400(BAD REQUEST) => Syntax yanlış, yüksek boyutlu data ya da diğer kullanıcı hatası

403(FORBIDDEN) => Bu data'ya erişim izniniz yok (400 olarak da gelebilir)

404(NOT FOUND) => Aranılan Data bulunamadı

500(INTERNAL SERVER ERROR) => Server kaynaklı bir hatadır, server'in düzelmesini bekleriz

**=> Api Testlerinde hangi programları kullanırız?**

**1- Postman**

**2- SOAP UI**

Xml formatında çalışır

**3-REST/ RestFul**

REST=> Representational State Transfer

Rest Api, xml ve json(Java Script Object Notation) formatlarının ikisini de kullanabilir, kullanımı daha yaygındır

**Bir REST Request'i Nelerden Oluşur?**

GET, dataları okumak içindir

POST, yeni data olusturmak içindir

PUT, dataları tamamen güncellemek için

PATCH, dataları kısmen güncellemek için

DELETE, dataları silmek için

API Endpoint Nedir?

Api'in istek gönderdiği ve kaynağın yasadığı yer, Endpoint'tir.

**Swagger Nedir?**

- API'nizin, zayıf kaynak kodu bilgisine sahip olsalar bile geliştiriciler ve testerlar tarafından okunması ve anlaşılması kolay ortak bir dil kullanarak tanımlamak için bir framework'tur.

- API belgeleri için SWAGGER kullanılır. Swagger bana API endpoint'leri verir ve bunların nasıl kullanılacağı hakkında bilgilendirir.

1. **SoapUI nedir ve mevcut projenizde nasıl kullandınız?**

SoapUI, SOAP ve REST web hizmetleri için açık kaynaklı, platformlar arası bir işlevsel test aracıdır. Geliştiricilerin web hizmetleri için karmaşık işlevsel ve yük testleri oluşturmasına, yürütmesine ve analiz etmesine olanak tanır ve test için onay testi, veri odaklı test ve test otomasyonu gibi çok çeşitli işlevler sağlar. Ayrıca, test senaryoları oluşturmayı ve yönetmeyi ve ayrıntılı test raporları oluşturmayı kolaylaştıran kullanıcı dostu bir arayüze sahiptir.

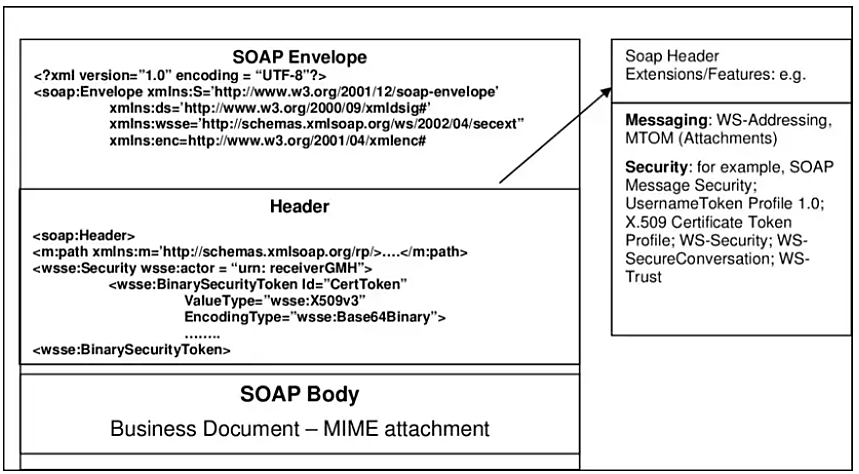
* SOAP UI, önde gelen açık kaynaklı platformlar arası API Test aracıdır
* SOAPUI, test uzmanlarının farklı WebAPI'lerde otomatik işlevsel, regresyon, uyumluluk ve yük testleri gerçekleştirmesine olanak tanır.
* SOAPUI, her türlü API'yi test etmek için tüm standart protokolleri ve teknolojileri destekler.
* SOAPUI arayüzü hem teknik hem de teknik olmayan kullanıcıların sorunsuz bir şekilde kullanmasını sağlayan basittir

SOAP protokolünde, veri formatı olarak XML kullanır.

SOAP genelde HTTP (Hyper Text Transfer Protocol) protokolü bazı zamanlarda ise TCP/IP kullanılarak gönderilir.

SOAP, kurumsal projelerde, veri hataları ve servisin nasıl kullanılacağıyla ilgili iletişimi azaltmak istenildiğinde ve .Net, Java gibi dillerle ile iletişim kuran entegrasyonlarda sıklıkla kullanılmaktadır.

SOAP 4 ana bölümden oluşmaktadır.



**Envelop**: SOAP yapısının root elemanıdır. En üstte bulunur ve kullanılması zorunludur.

**Header**: HTML’deki head etiketine benzemektedir. Authentication, SOAP ile ilgili işlemlerin yapıldığı alandır. Bu alan zorunlu değildir. Geliştiricinin insiyatifine bırakılmıştır.

**Body**: SOAP’ın en önemli alanıdır. Bu alanda çeşitli fonksiyonların sonuçları ve bilgileri tutulur.

**Fault**: SOAP içerisinde oluşabilecek hataların yönetildiği alandır. Bu alanda hata mesajları, hata bilgileri bulunur.

* SoapUI'nin bir projede nasıl kullanılabileceğine dair örnekler:
* Belirli bir konum için hava durumu tahmini sağlayan bir web hizmetiniz var.
* Bu web hizmeti için test senaryoları oluşturmak üzere SoapUI'yi kullanabilirsiniz. Örneğin, web hizmetine belirli bir konumla bir istek gönderen ve doğru hava tahmini için yanıtı kontrol eden bir test senaryosu oluşturabilirsiniz.
* Web hizmetini birden fazla konumla test etmek ve yanıtların doğruluğunu kontrol etmek için SoapUI'nin veri odaklı test özelliğini de kullanabilirsiniz.
* SoapUI'yi web hizmetinin performansını test etmek amacıyla farklı yükleri simüle etmek için de kullanabilirsiniz. Örneğin, yoğun kullanım altında nasıl performans gösterdiğini görmek için kısa bir süre içinde web hizmetine çok sayıda istek gönderen bir yük testi oluşturabilirsiniz.
* Test senaryolarını yürüttüğünüzde SoapUI, başarılı ve başarısız test senaryolarının sayısı, yanıt süresi ve hata mesajları gibi bilgileri içeren ayrıntılı test raporları sağlayacaktır.
* Bu bilgileri web hizmetiyle ilgili sorunları belirlemek ve gerekli iyileştirmeleri yapmak için kullanabilirsiniz.
* Bunun üst düzey bir örnek olduğunu ve SoapUI'nin güvenlik testi, işlevsel test ve test otomasyonu gibi bundan daha fazla yetenek sunduğunu belirtmek gerekir.

1. REST tabanlı mimaride yaygın olarak kullanılan bazı HTTP yöntemlerinin adı nedir?

REST (Representational State Transfer) mimarisinde yaygın olarak kullanılan bazı HTTP yöntemleri şunlardır:

GET: Bir kaynağın bilgisini almak için kullanılır.

POST: Bir kaynak oluşturmak veya güncellemek için kullanılır.

PUT: Bir kaynağın tamamını güncellemek için kullanılır.

DELETE: Bir kaynağı silerek işlem yapmak için kullanılır.

PATCH: Bir kaynağın belirli bir bölümünü güncellemek için kullanılır.

GET: getir

-->id ve benzer verileri inceleyebiliriz

-->Pretty'den kalıbı kopyalayabiliriz

POST: Yeni kayıt oluştur

--> Kopyaladığımız kalıpla body'de JSON dili ile yeni kayıt oluşturabiliriz

--> Mevcut id değişiyor doğal olarak

PATCH: Yama yapma (Yani düzeltme)

--> Mevcut datanın bir kısmını güncellemek için kullanılır

PUT: Bütün datayı güncelleme

--> Mevcut id değişmeden data'da komple değişiklik yapilabilir

DELETE: Kayıt silme

--> Mevcut kaydı silmek icin kullanılır

* -Bütün Web Service'ler API'dir ama her API bir Web Service değildir.

1) GET: Sunucudan sadece veri çekmek(okuma) istiyorsak yani veri üzerinde herhangi bir değişiklik (ekleme, silme, modifiye) yapılmayacaksa GET metodunu kullanmamız tavsiye ediliyor.

CRUD operasyonlarından Read’e karşılık geldiğini söyleyebiliriz.

Ör: GET /students kullandığımızda bize öğrenciler listesini dönmesi.

2) POST: Server Api’e body kısmını doldurarak ve veri üzerinde değişiklik yapmak istediğimizde kullanabiliriz

Değişiklik yapmak ile kastedilen CRUD operasyonlarından Create ve Update kısımlarını kapsar.

Ör: Post /createUser ile body kısmına kullanıcı bilgileri girip veritabanında bir kullanıcı oluşturulması istenmesi

3) PUT: Post isteğinin özelliklerine sahiptir. Yani CRUD operasyonlarından Create ve Update operasyonlarını yapmak istediğimizde kullanıyoruz.

Post’dan ayrılan tarafı Put isteğinin idempotent ve not cacheable olarak tanımlanması

4) DELETE: CRUD operasyonlarından Delete’e karşılık gelir. Bir veriyi silmek istediğimizde kullanılması tavsiye ediliyor.

* Mevcut projenizde nasıl kullandınız?

**1**- Postman’i, manuel API testleri icin kullandık.

**2**- Otomasyon testleri için de Rest Assuret Librari kullanıldı. Bunun için PomXml’e api seviyesinde testler yapmak ve birçok http işlemlerini yönetebilmek için rest-assured yüklendi.

**3**- Otomasyon testlerimizi yaparken aşağıdaki adımları izledik.

a) Gereksinimleri anlamak,

b) Test Case yazıyoruz

i) Test Case yazımında Gherkin dilini kullanacağız. Bizler yazılım diline hâkim olsak da karsımızdaki kişiler hâkim olmayabilir ama Gherkin ile yazılan testleri anlamakta zorluk çekmeyeceklerdir.

Gherkin dilinde kullandığımız keywordler;

- Given: Ön koşullar

- When: Yapılacak aksiyonlar için (get(), put(), post(), patch() ve delete() )

- Then: İstek yaptıktan (request gönderdikten sonra) doğrulama

- And: Çoklu işlemlerde kullanacağız

c) Test kodlarımızı yazmaya başlayacağı

i) Set the URL, (İşlem yapılacak olan Url girilir.)

ii) Set the expected Data (beklenen datanın oluşturulması, post, put, patch)

Get işlemlerinde bizden post, put ya da patch istenmediği zaman bu case de işlem yapmaya gerek kalmıyor.

iii) Type code to send request (Talep göndermek için kod yazımı)

iv) Do Assertion () doğrulama yapmak

1. **HTML Status Codes?**

200 OK – İşlem başarıyla gerçekleşirse kullanılır.

201 CREATED – İşlemi başarıyla gerçekleşirse kullanılır. Eklenen veri döndürülür.

204 NO CONTENT – İşlemi başarıyla gerçekleşirse kullanılır. Eklenen, silinen veri döndürülmez.

400 BAD REQUEST – Hatalı istek veya yetkilendirme hatası için kullanılır.

401 UNAUTHORIZED – Yetkilendirme hatası için kullanılır.

403 FORBIDDEN – Yetkilendirme ve erişim hatası için kullanılır.

404 NOT FOUND – İstenilen işlem yoksa kullanılır.

405 METHOD NOT ALLOWED – İzin verilmeyen HTTP yöntemi kullanıldığında bilgi vermek için kullanılır.

409 CONFLICT – Aynı veri üzerinde birden fazla kişinin işlem yaptığını bildirmek için kullanılır.

500 INTERNAL SERVER ERROR – Sunucuda herhangi bir hata olduğunda kullanılır.

1. **What first thing you check when you getresponse? (Yanıt aldığınızda ilk kontrol ettiğiniz şey nedir?)**

İsteğin başarılı olup olmadığını gösteren yanıtın HTTP durum kodu.

Döndürülen verilerin beklendiği gibi olduğundan emin olmak için yanıtın içeriği.

Yanıtta yer alan ve istek hakkında ek bilgiler içerebilecek tüm başlıklar.

Hataların, zaman aşımlarının ve beklenmedik sonuçların da kontrol edilmesi önerilir.

● Status quote (200 her zaman tamam anlamına gelir)

● We always check the 404 means not found (Her zaman 404'ün bulunamadı anlamına gelip gelmediğini kontrol ederiz)

● rest-assured.io==> for automation to find the ECS machine in search type remote Desktop (ECS makinesini arama türünde bulmak için otomasyon için uzak Masaüstü)

1. **Http methods and requesttypes (Http yöntemleri ve istek türleri)**

HTTP (Hypertext Transfer Protocol) internet üzerinde iletişim için kullanılan protokoldür. Belirli bir kaynak üzerinde gerçekleştirilmesi istenen eylemi belirten bir dizi yöntem veya fiil tanımlar. En yaygın HTTP yöntemleri şunlardır:

**GET**: Bir kaynak hakkında bilgi alır. Bu en yaygın kullanılan yöntemdir ve sunucudaki herhangi bir veriyi değiştirmeden sunucudan bilgi almak için kullanılır.

**POST**: Daha fazla işlem için bilgileri sunucuya gönderir. Bu yöntem sunucuda bir kaynak oluşturmak veya güncellemek için kullanılır.

**PUT**: Mevcut bir kaynağı yeni bir kaynakla değiştirir. Bu yöntem sunucudaki bir kaynağı güncellemek için kullanılır.

**DELETE**: Bir kaynağı sunucudan siler.

**HEAD**: Mesaj gövdesi olmadan bir kaynağın başlık bilgilerini alır.

**OPTIONS** (SEÇENEKLER): Bir kaynak için izin verilen yöntemleri alır.

**PATCH** (YAMA): Sunucudaki bir kaynağı kısmen günceller.

**CONNECT**: BAĞLANIR: İstek bağlantısını şeffaf bir TCP/IP tüneline dönüştürür.

**TRACE**: İstek başlıklarını istemciye geri yankılar.

PURGE, COPY, LOCK, UNLOCK, PROPFIND gibi diğer yöntemler daha az kullanılır.

Tüm yöntemlerin her sunucu ve API tarafından desteklenmediğini ve desteklenen belirli yöntemlerin etkileşimde bulunduğunuz API veya hizmete bağlı olarak değişebileceğini unutmamak önemlidir.

● Get does not requires body (Get için gövde gerekmez)

● Put requires body means UPDATE information (Koymak için gövde gerekir, bu da bilgileri GÜNCELLE anlamına gelir)

● Post requires body means CREATE information (Gönderi gövde gerektirir, bilgi OLUŞTURMAK anlamına gelir)

● Delete does not requires body (Silme işlemi gövde gerektirmez)

● GET-> READ, POST-> CREATE, PUT -> UPDATE, DELETE ->DELETE

● POST VS PUT

1. **Parameters API**

PATH PARAMETER (VALUE WILL BE PART OF THE URL) QUERY/REQUEST [PATH PARAMETRESI (DEĞER URL'NIN BIR PARÇASI OLACAKTIR) QUERY/REQUEST]

PARAMETERS (KEY+VALUE FORMAT) [PARAMETRELER (ANAHTAR+DEĞER FORMATI)]

API parametreleri, istenen eylemi veya bilgiyi belirtmek için bir API çağrısına aktarılan değerlerdir. Bu parametreler, döndürülen verilerin biçimi, döndürülecek sonuç sayısı ve API'nin davranışını kontrol eden diğer seçenekler gibi şeyleri içerebilir.

Genellikle URL'de veya API isteğinin gövdesinde anahtar-değer çiftleri şeklinde iletilirler. Yaygın parametrelere örnek olarak döndürülecek sonuç sayısını belirtmek için "limit", veri kümesinde bir başlangıç noktası belirtmek için "offset" ve arama terimlerini belirtmek için "query" verilebilir.

* İki ana API parametresi türü vardır:
* Path parameters (yol parametreleri) ve Query/request parameters (sorgu/istek parametreleri).

**Path parameters**: Bunlar, URL yolunun bir parçası olarak dahil edilen ve bir kaynak veya uç noktayı belirtmek için kullanılan parametrelerdir. Genellikle bir koleksiyondaki belirli bir kaynağı veya öğeyi tanımlamak için kullanılırlar.

Örneğin, "https://api.example.com/users/123" URL'sinde "123", 123 ID'li kullanıcıyı belirten bir yol parametresidir (path parameter).

**Query/request parameters** (Sorgu/istek parametreleri): Bunlar, URL'nin sorgu dizesinde veya istek gövdesinde geçirilen parametrelerdir. API isteği için seçenekleri veya filtrelemeyi belirtmek için kullanılırlar. Bu parametreler genellikle API'nin davranışını kontrol etmek için kullanılır, örneğin döndürülecek sonuç sayısını veya sonuçların sıralama düzenini belirtmek gibi.

Örneğin, "https://api.example.com/users?limit=10&offset=20" URL'sinde "limit" ve "offset", sonuç sayısını ve veri kümesindeki başlangıç noktasını belirtmek için kullanılan sorgu parametreleridir.

1. **Hamcrest Matcher ne içindir?**

Hamcrest, 'eşleşme' kurallarının bildirimsel olarak tanımlanmasına olanak tanıyan eşleştirici nesneleri yazmak için bir çerçevedir.

Hamcrest, değerlerle esnek ve okunabilir bir şekilde eşleştirmek için birleştirilebilen bir eşleştirici nesneleri (kısıtlamalar veya yüklemler olarak da bilinir) kütüphanesidir. Bu eşleştiriciler, anlamlı ve okunması kolay birim testleri oluşturmak için kullanılabileceği gibi, değerleri eşleştirmek için esnek bir yola sahip olmanın yararlı olduğu diğer bağlamlarda da kullanılabilir.

Hamcrest Matcher, bir Java kütüphanesidir ve test sırasında nesnelerin beklenen değerlere eşit olduğunu veya beklenen özellikleri taşıdığını kontrol etmek için kullanılır. Örnek olarak, bir metodun geri dönüş değerinin belirli bir değerle eşit olmasını veya bir nesnenin belirli bir özelliğinin belirli bir değerle eşit olmasını kontrol edebilirsiniz.

Aşağıda bir örnek verilmiştir:

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu örnekte, değişken 'value'nin 5'e eşit olduğu kontrol edilir. Hamcrest Matcher, assertThat() metodunun ikinci parametresi olarak kullanılır. Eğer değer 5'e eşit değilse, bir hata oluşur.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

* // ilk argümanın ikinciye eşit olup olmadığını doğrula

assertThat(str1, is("Kunkka"));

assertThat(str1, is(str2));

* // ilk argümanın ikinciye eşit DEĞİL olup olmadığını doğrula

assertThat(str1, is(not("Tidehunter"));

* // görmezden gelme durumunu karşılaştır

assertThat(str1, equalToIgnoringCase("kunkka"));

* // önceki ve sonraki boşlukları yok sayarak karşılaştır

assertThat(str1, equalToIgnoringWhiteSpace(" Kunkka "));

* // sayıları karşılaştır

assertThat(10, greaterThan(9));

assertThat(10, lessThan(11));

assertThat(10, lessThanOrEqualTo(11));

* // null olmadığını doğrulayın

assertThat(str1, notNullValue());

List<String> liste = Arrays.asList("one", "too", "tree");

assertThat(liste, hasSize(3));

assertThat(liste, containsInAnyOrder("too", "tree", "one"));

assertThat(liste, hasItems("one", "too"));

List<Integer> numbers = Arrays.asList(11, 12, 13);

assertThat(sayılar, everyItem(greaterThan(9)));

1. **RestAssured Log Logging Logs (RestAssured Günlük Kaydı Günlükleri)**

RestAssured, REST API testleri yazmak için kullanılan bir Java kütüphanesidir. Logging, RestAssured tarafından yapılan istek ve yanıtların detaylı olarak kaydedilmesini sağlar. Bu, test sırasında oluşabilecek hata veya sorunların daha kolay takip edilmesini sağlar.

RestAssured, istek ve yanıtları kaydetmek için aşağıdaki metodları kullanabilirsiniz:

* given().log().all() : Bu metod, istek bilgileri (URL, query parametreleri, vb.) ve yanıt bilgileri (status code, body, vb.) dahil olmak üzere tüm bilgileri kaydeder.
* given().log().ifValidationFails(): Bu metod sadece doğrulama başarısız olduğunda istek ve yanıt bilgilerini kaydeder.
* given().log().ifError(): Bu metod sadece yanıtta hata kodu döndüğünde istek ve yanıt bilgilerini kaydeder.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

* Bu örnekte, given().log().all() metodu kullanılarak, istek ve yanıt bilgileri konsola yazdırılır.
* **Request Logging (İstek Günlüğü)**

given().log().all() // Parametreler, üstbilgiler ve gövde dahil olmak üzere tüm istek belirtimi ayrıntılarını günlüğe kaydet

given().log().params() // Yalnızca isteğin parametrelerini günlüğe kaydet

given().log().body() // Yalnızca istek gövdesini günlüğe kaydet

given().log().headers() // Yalnızca istek başlıklarını günlüğe kaydet

given().log().cookies() // Yalnızca istek çerezlerini günlüğe kaydet

given().log().method() // Yalnızca istek yöntemini günlüğe kaydet

given().log().path() // Yalnızca istek yolunu günlüğe kaydet

* **Response Logging (Yanıt Günlüğü)**

get("/x").then().log().body()

get("/x").then().log().ifError()

get("/x").then().log().all()

get("/x").then().log().statusLine() // Yalnızca durum satırını günlüğe kaydet

get("/x").then().log().headers() // Yalnızca yanıt başlıklarını günlüğe kaydet

get("/x").then().log().cookies() // Yalnızca yanıt çerezlerini günlüğe kaydet

get("/x").then().log().ifStatusCodeIsEqualTo(302)

// Yalnızca durum kodu 302'ye eşitse günlüğe kaydet

get("/x").then().log().ifStatusCodeMatches(matcher)

// Yalnızca durum kodu verilen Hamcrest eşleştiricisiyle eşleşirse günlüğe kaydedin

1. **Serialization ve Deserialization**

* Serialization; bir Java nesnesini API JSON formatına eşlediğimizde (JAVA NESNESİNİ JSON'A DÖNÜŞTÜR).
* Java nesnesi (POJO (Plain Old Java Object), BEANS) → APIJSON/XML ile eşleme
* Bir sınıftan bir nesne aldığımızda ve bunu RESTful API'mizde JSON formatına eşlediğimizde

{make: “Toyota”,

Model: “Camry” }

Car car = new Car();

car.setMake(“Toyota”);

car.setModel(“Camry”);

given().body(car).when().post(uri)

* Deserialization; API JSON/XML → Map Java Nesnesine Eşleme (JSON TO JAVAOBJECT)

Car car2 = new Car();

car2=when().get(uri).body.as(car.class);

car.setMake(“Toyota”);

car.setModel(“Camry”);

Serialization, bir nesnenin durumunu bayt akışına dönüştürme işlemidir. Deserialization , nesneyi bayt akışından yeniden yapılandırma işlemidir. Bu genellikle nesneleri ağ üzerinden göndermek veya bir dosyada saklamak için kullanılır. Bayt akışı XML, JSON veya ikili gibi çeşitli formatlarda olabilir. Serialization ve deserialization genellikle bir kütüphane veya çerçeve tarafından gerçekleştirilir, manuel olarak uygulanmaz. Örneğin, Java dillerinde serileştirme için "Serializable" arayüzünü kullanabilir ve serileştirilmiş nesneyi okumak için "ObjectInputStream" sınıfını kullanabilirsiniz. Örnek olarak bir nesnenin serileştirilmesi aşağıdaki gibidir:

**public class SerializationExample {**

**public static void main(String[] args) {**

**Employee employee = new Employee();**

**employee.setName("John Doe");**

**employee.setAge(30);**

**try {**

**FileOutputStream fileOut = new FileOutputStream("employee.ser");**

**ObjectOutputStream out = new ObjectOutputStream(fileOut);**

**out.writeObject(çalışan);**

**out.close();**

**fileOut.close();**

**System.out.println("Serialized data is saved in employee.ser");**

**} catch (IOException i) {**

**i.printStackTrace();}}}**

* Bu örnekte, bir "Employee" nesnesi oluşturuldu ve serileştirildi. Serileştirilmiş nesnenin okunması işlemi aşağıdaki gibidir:

public class DeserializationExample {

public static void main(String[] args) {

Employee employee = null;

try {

FileInputStream fileIn = new FileInputStream("employee.ser");

ObjectInputStream in = new ObjectInputStream(fileIn);

employee = (Employee) in.readObject();

in.close();

fileIn.close();

} catch (IOException i) {

i.printStackTrace();

return;

} catch (ClassNotFoundException c) {

System.out.println("Employee class not found");

c.printStackTrace();

return; }

System.out.println("Deserialized Employee...");

System.out.println("Name: " + employee.getName());

System.out.println("Age: " + employee.getAge());}}

* Bu örnekte, serialized nesne "employee.ser" dosyasından okunur ve yeniden oluşturulur. Son olarak, nesnenin adı ve yaşı ekrana yazdırılır. Serialization ve deserialization, nesneleri bir ağ üzerinden göndermek veya bir dosyada saklamak gibi uygulamalarda kullanılabilir. Ayrıca, serialization ve deserialization işlemleri çok sayıda programlama dili için mevcuttur ve genellikle kütüphaneler veya çerçeveler tarafından desteklenir.

1. **RestAssured Kütüphanesi ile API/Webservices?**

Rest-Assured, RESTful API'ler için okunabilir ve bakımı yapılabilir testler yazmanıza olanak tanıyan bir Java kütüphanesidir. HTTP tabanlı API'lerle etkileşim için basit, alana özgü bir dil sağlayarak API uç noktalarınız için testler oluşturmayı kolaylaştırır. Rest-Assured ile GET, POST, PUT ve DELETE gibi çeşitli HTTP istekleri gerçekleştirebilir ve yanıtın beklenen durum koduna, başlıklara ve gövdeye sahip olduğunu doğrulayabilirsiniz. Ayrıca, yanıttan verileri kolayca ayıklayıp doğrulayabilir ve isteğe kimlik doğrulama ve diğer başlıkları ekleyebilirsiniz. Genel olarak Rest-Assured, Java ortamında RESTful web hizmetlerini ve API'leri test etmek için güçlü bir araçtır.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. **EndPoint nedir?**

**Endpoint**: API üzerinde belirli bir amaç için oluşturulmuş metodunuza verilen isim.

**Endpoint(sorgu adresi)**: API de sorgu yapılırken sorgu adresi farklı bir yapıda olur. Mesela Twitter adresi twitter.com dur. Fakat https://api.twitter.com/1.1/statuses/update.json endpoitini yani sorgu adresini kullanır.

<protocol>://<service-name>/<ResourceType>/ResourceID -> URI (Uniform Resource Identifier)

Base URI / resource ? Parameters

(http://www.google.com/search?source=book…) -> ? -> query parameters

1. **Authorization vs Authentication**

* Authentication, bir kullanıcının bir sisteme giriş yaparken kimliğinin doğrulanmasıdır. Örneğin, bir kullanıcının kullanıcı adı ve şifresi kullanarak bir web sitesine giriş yapması authentication işlemidir.
* Authorization, bir kullanıcının sisteme erişim haklarının belirlenmesidir. Örneğin, bir kullanıcının sadece okuma yetkisi olsa bile, yazma yetkisi olmayabilir. Bu authorization işlemidir.
* Ayrıca bir örnekte; Bir kullanıcının sisteme giriş yaptıktan sonra kullanıcının hangi işlemleri yapabileceği belirlenir. Bu işlem authorization işlemidir. Örneğin, bir banka müşterisi sadece hesabının bakiyesini görüntüleyebilirken, yatırım yapma yetkisi yoksa bu authorization işlemidir.

Authorization ve Authentication

authentication --> sen kimsin

authorization --> hangi haklara sahipsiniz

authentication kullanıcı ve paroladır

authorization türleri vardır:

○ authorization yok

○ Temel authorization

○ Taşıyıcı Token

○ Inherit Auth from parent

1. **RESTful Web Hizmeti / API**

* RESTful web hizmeti, Representational State Transfer (REST) prensiplerine dayalı bir web hizmeti türüdür. RESTful web hizmeti, HTTP protokolü kullanarak verileri alıp verir. RESTful web hizmetleri, URL yolu ile erişilen ve HTTP metotları (GET, POST, PUT, DELETE vb.) kullanılarak yapılan istekleri yanıtlar.
* API (Application Programming Interface), bir uygulamanın veya sistemin iç yapısının dışarıya açılmasını sağlar. RESTful web hizmeti, bir API olarak kullanılabilir. Örneğin, bir e-ticaret web sitesi, müşterilerin ürünleri arama, inceleme ve satın alma işlemlerini yapmasını sağlamak için RESTful web hizmeti/API oluşturabilir.
* RESTful web hizmetleri, genellikle XML veya JSON formatında verileri döndürür. Bu, farklı platformlar ve programlama dilleri arasında veri alışverişini kolaylaştırır. Örneğin, bir mobil uygulama ile bir web uygulaması arasında veri alışverişi yapmak için RESTful web hizmeti/API kullanılabilir.

REST, Temsili Durum Aktarımı anlamına gelir

RESTful, REST mimari konsepti uygulanarak yazılan web servisleri için kullanılır.

RESTful'da CRUD işlemlerini gerçekleştirmek için GET, POST, PUT, DELETE gibi web servis http yöntemleri kullanılabilir.

CRUD = Oluştur-> Oku-> Güncelle-> Sil

* **RestApi için en etkili toollar ve framework hangisidir?**
* REST API için kullanabileceğiniz birçok araç ve framework mevcuttur. Önerilebilecek bazı popüler seçenekler şunlardır:
* Express.js: Node.js için kullanabileceğiniz popüler bir frameworktür. Kullanımı kolaydır ve çok sayıda eklenti ve modül mevcuttur.
* Flask: Python için kullanabileceğiniz bir frameworktür. Basit ve esnek yapısı nedeniyle popülerdir.
* Spring: Java için kullanabileceğiniz bir frameworktür. Özellikle büyük ve karmaşık projeler için uygun bir seçenektir.
* Django Rest Framework: Python için kullanabileceğiniz bir frameworktür. Django web framework ile birlikte kullanılması önerilir.
* FastAPI: Python için kullanabileceğiniz bir frameworktür. Performans ve hız açısından diğer alternatiflerden daha iyidir.
* **Java için REST API oluşturmak için kullanabileceğiniz popüler araçlar ve frameworkler şunlardır:**
* **Spring**: Java için en yaygın olarak kullanılan ve özellikle büyük ve karmaşık projeler için uygun bir frameworktür. Spring Boot ve Spring MVC gibi alt frameworkleri de içermektedir.
* **Jersey**: Java için bir RESTful web servis frameworktür. Spring'e alternatif olarak kullanılabilir.
* **JAX-RS** (Java API for RESTful Web Services): Java için bir standarttır ve Spring ve Jersey gibi diğer frameworkler tarafından desteklenmektedir.
* **Dropwizard**: Java için bir RESTful web servis frameworktür. Performans ve hız açısından üst düzeydir.
* **Restlet**: Java için bir RESTful web servis frameworktür. Basit ve esnek yapısı nedeniyle popülerdir.
* Her projenin ihtiyacı farklıdır, bu nedenle en uygun framework veya araç için projenin ihtiyacına göre değerlendirmeniz gerekir.

1. **Response body bir değeri nasıl doğrularsınız?**

Örn: Kimliğin doğru numarayı içerdiğini doğrulama

o Hamster Matcher bir assertion kütüphanesidir.

**Response.then().assertThat().body("Id",Matchers.equalTo(123)).**

o JsonPath'e ayrıştırın ve Id değerini okumak için getInt(), getList(), getString() yöntemlerini kullanın.

Ve JUnit Assertion kullanabilirim:

**String body = ...thenReturn().body().asString().**

**JsonPath json = new JsonPath(body).**

**assertEquals(123,json.getInt("Id")).**

o Bir (POJO) nesnesine (veya Nesne Eşlemesine) serileştirme

**POJO myPojo = ... when().post(url).thenReturn().body().as(Pojo.class).**

**assertEquals(123,myPojo.getId( ) ).**

Ve JUnit Assertion kullanabilirim.

1. **API Authentication Türleri**

**- Basic**

**Pre-emptive**

Eğer bir hizmet pre-emptive olarak yapılandırılmışsa, istemciden kimlik bilgisi gerektirmesine rağmen kimlik bilgisi talep etmeyecektir.

Bir istek kimlik bilgileri içermiyorsa, 401 Yetkisiz durum kodunu döndürür.

**Challenged**

Request API'ye ulaştığında, API kimlik bilgileri gerektirdiğini söyleyecek ve ardından istemci kimlik bilgilerini sağlayacaktır.

**oauth** --> kimlik doğrulaması için 3. taraflardan anahtarların ve belirteçlerin kullanıldığı kimlik doğrulama türleri. 2 tür oauth vardır:

**oauth1** -> uygulaması zor

**oauth2** -> daha güvenli

**Digest**

Temelden daha şifrelidir. https...

* API’larda doğrulama için kullanılan yöntemler dörde ayrılabilir bunlar:
* Basic
* Bearer
* Digest
* OAuth ve diğerleri…

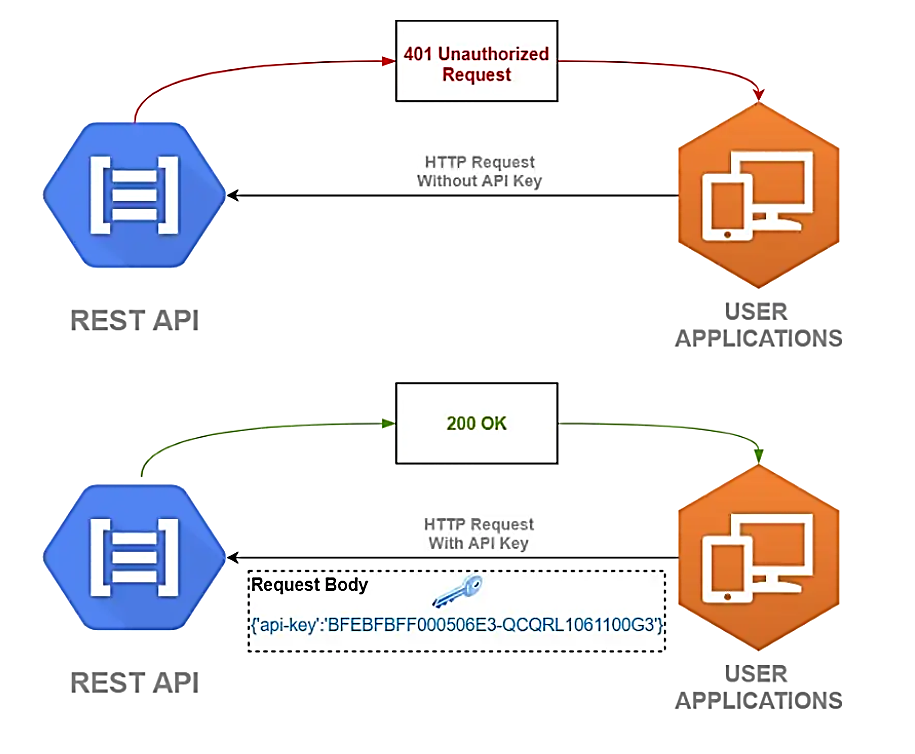
**Temel Doğrulama (Basic Authentication)**

Doğrulama yöntemlerinden en basit olanıdır. Bir HTTP kullanıcı olarak kullanıcı adı ve şifre sağlanmasıyla doğrulamanmasına(Bilgi Faktörü) dayanır. Herhangi bir ekstra bilgi tutmaya gerek yoktur. HTTP Header’ından gönderildiği için ekstra el sıkışmalarına (handshake) veya karmaşık sistemlere ihtiyaç duymaz.

**API Anaharı ile Doğrulama (API Key Authentication)**

Kullanıcı bir API’ya kendisini doğrulattıktan sonra kendisine (MAC, IP veya bazen ikisi de alınarak) ait benzersiz(unique) bir anahtar oluşturulur ve bu anahtar ile sonraki işlemlerinde kendisini sisteme doğrulatır. Bu doğrulama esnasında ilgili anahtar farklı şekillerde gönderilebilir. Bu yöntemler günümüze kadar:

* Body (Gövde) içinde bir alan olarak,
* Header içerisinde (Authorization: Apikey BFEBFBFF000506E3-QCQRL1061100G3),
* Basic içerisinde ( Authorization: Basic BFEBFBFF000506E3-QCQRL1061100G3),
* Query String (?apikey=BFEBFBFF000506E3-QCQRL1061100G3),
* İle gönderildiği gözlemlenmiştir.



API anahtarlarını kullanmak için bir çok sebep vardır lakin en temelde kullanımının kolay olması bizleri cezbeder. Örneğin sadece “read” (okuma) işlemlerini barındıran bir API’da API anahtarlama ile doğrulama yapmak performans ve doğrulama açısından uygun bir yöntem olabilir lakin içerisinde “update” güncelleme, “create” ekleme, “delete” silme işlemleri yapılan ve güvenliğin yüksek öncelikli olmadığı bir API için tercih edilebilir.

<https://medium.com/@thejengo/api-i%CC%87stemcisi-do%C4%9Frulama-y%C3%B6ntemleri-c29e3826daea>

1. **SOAP kullanmanın avantajı nedir?**

* REST çok çeşitli veri formatlarına izin verirken SOAP yalnızca XML'e izin verir.
* JSON ile birleştiğinde (genellikle verilerle daha iyi çalışır ve daha hızlı ayrıştırma sunar), REST genellikle kabul edilir ile çalışmak daha kolay.
* JSON sayesinde REST, tarayıcı istemcileri için daha iyi destek sunar.
* REST, özellikle değiştirilmeyen ve dinamik olmayan bilgiler için önbelleğe alma yoluyla üstün performans sağlar
* Yahoo, Ebay, Amazon ve hatta Google gibi büyük hizmetler için en sık kullanılan protokoldür.
* REST genellikle daha hızlıdır ve daha az bant genişliği kullanır. Ayrıca, mevcut web sitelerine entegre edilmesi daha kolaydır ve site altyapısını yeniden düzenler. Bu, geliştiricilerin bir siteyi sıfırdan yeniden yazmak için zaman harcamak yerine daha hızlı çalışmasını sağlar.
* Bunun yerine, basitçe ek işlevler ekleyebilirler.

1. **SOAP ve RESTful web hizmetleri arasındaki fark nedir?**

* RESTful JSON, XML, TEXT'i destekler, ancak SOAP yalnızca XML'i destekler
* REST, SOAP tabanlı web hizmetlerinden daha hızlıdır
* SOAP XML veri tipini desteklerken REST istenen veri türüyle işlem yapabilir. JSON veri tipi ile XML’den çok daha düşük boyutlarla veri tutulabildiği için REST ile daha hızlı işlem yapılabilir.
* Ayrıca SOAP için WSDL ile tanımlama yapmak gerekirken REST için böyle bir zorunluluk yoktur. (WADL REST için kullanılan WSDL’e benzer bir yapıdır fakat kullanma zorunluluğu yoktur.) Bir dile ihtiyaç duymadan HTTP metodlarıyla tasarlanabildiği için REST’i kullanması ve tasarlaması daha kolaydır.
* SOAP için birçok geliştirme aracı mevcuttur, REST için geliştirme araçlarına ihtiyaç duyulmaz, tasarlaması kolaydır.
* SOAP; XML-Scheme kullanırken REST; URI-scheme kullanır yani metotlar için URI’ler tanımlanır.
* Her ikisi de HTTP protokolünü kullanırlar. Fakat REST için HTTP zorunluluğu varken SOAP; TCP, SMTP gibi başka protokollerle de çalışabilir.
* Test ve hata ayıklama aşaması REST için daha kolaydır. Çünkü HTTP hatalarını döndürür ve bunlar bir toola ihtiyaç duyulmadan görülebilir. SOAP için hata ayıklama araçları gerekebilir.
* REST basit HTTP GET metodunu kullandığı için cacheleme işlemi daha kolaydır. SOAP ile cacheleme yapabilmek için karmaşık XML requestleri yapılmalıdır.
* İkisi de HTTPS destekler, SOAP için WS-SECURITY adlı bir eklenti mevcuttur.
* Güvenlik açısından SOAP daha gelişmiştir çünkü hazır yapılar bulunmaktadır.
* Dokümantasyon bakımından SOAP daha gelişmiştir ve daha fazla kaynak bulunmaktadır.

1. **URI nedir, amacı ve formatı nedir?**

URI (UNİFORM RESOURCE IDENTİFİER)

En yaygın tanımıyla URI, bir adı veya kaynağı tanımlamak için kullanılan bir karakter dizisidir. Bir kaynağı konuma, ada veya her ikisine göre tanımlar. Eğer bir URI bir kaynağı konuma göre tanımlarsa aynı zamanda bir URL, ada göre tanımlarsa aynı zamanda bir URN’dır. Kısacası URN ve URL, URI’ ın alt kümeleridir.

* URI, Tekdüzen Kaynak Tanımlayıcısı anlamına gelir
* URI'nin amacı, web hizmetini barındıran sunucu üzerindeki bir kaynağı bulmaktır.
* Bir URI aşağıdaki formattadır:
* ○ <protocol>://<service-name>/<ResourceType>/<ResourceID>

1. **Projenizde hangi WebServices'i kullanıyorsunuz?**

Temsili Aktarım Durumu olan Restful'u kullanıyorum ve XML ve JSON ile iletişim kuruyor, ancak şu anki proje JSON kullanıyor.

* Projede Rest kullanıldı.

1. **XML nedir?**

Bilgi işlemde, Genişletilebilir Biçimlendirme Dili (XML), kodlama için bir dizi kural tanımlayan bir biçimlendirme dilidir hem insan tarafından okunabilir hem de makine tarafından okunabilir bir formatta belgeler.

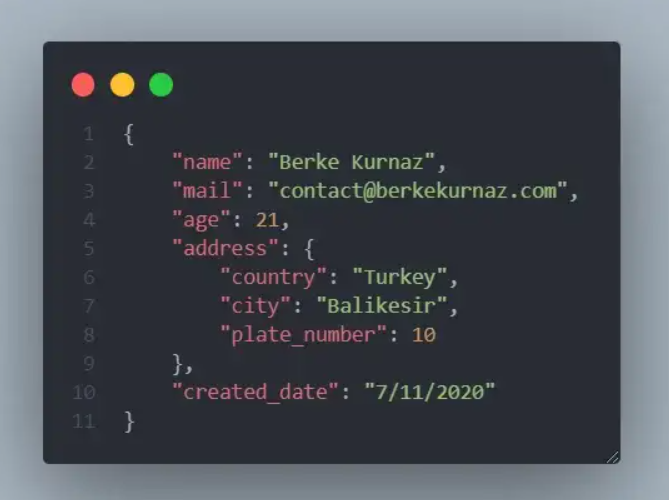
1. **JSON nedir?**

JSON tam adıyla JavaScript Object Notation bütün programlama dilleri arasında, yapılandırılmış veri değişimini kolaylaştıran bir metin biçimidir. Programlama dillerinden bağımsız çalışan JSON dosyalarını istediğiniz programlama dilinde alıp kullanabilir veya var olan verileri JSON haline dönüştürebilirsiniz.

JSON özellikle key-value yapısına göre çalışabilmesi, parçalaması kolay olması ve okunabilirliği yüksek olması itibariyle XML’e göre son zamanlarda oldukça popüler hale geldi. Oldukça hızlı çalışabilen JSON formatı günümüz dünyasında birçok api başta olmak üzere açık kaynak hale getirilen veri setlerinde de sıklıkla kullanılmaktadır.

Örnek JSON Dosyası

JSON dosyası süslü parantezler içerisinde { } yazılır ve her key ile value değerleri sırasıyla alt alta “key”: “value” şeklinde yazılır. Aşağıya eklediğim örnek ile bu anlatımı çok daha rahat şekilde anlayabilirsiniz.



* JavaScript Object Notation'dır (verileri yapılandırmak için minimal, okunabilir bir formattır).
* Öncelikle XML'e alternatif olarak bir sunucu ve web uygulaması arasında veri iletmek için kullanılır.
* Temel olarak XML'in hafif bir versiyonudur.
* Anahtar'da: Değer biçimi
* Anahtar her zaman çift tırnak içinde ve değer dizeyse çift tırnak içinde, sayılarsa tırnak içinde değildir
* Tamamen http protokolüne dayanır, - bu nedenle tarayıcıdaki bağlantıya vurur ve sonuçları görür

1. **Swagger biliyor musun? Swagger nedir**

Web API geliştirmede en önemli ihtiyaçlardan biri dokümantasyon ihtiyacıdır. Çünkü API methodlarının ne işe yaradığı ve nasıl kullanıldığı dokümantasyon içeresinde anlaşılır olması gerekir. Api dokümantasyonunu el emeği ile yazmak hem zordur hemde güncel tutması imkansızdır. Bir biçimde bu dokümantasyonu güncel olarak üretmek gerekir. Burada imdadımıza swagger yetişiyor.

Swagger’ın önemli bir amacı RestApi ler için bir arayüz sağlamaktır. Bu hem insanların hemde bilgisayarlara kaynak koda erişmeden RestApi lerin özelliklerini görmesine, incelemesine ve anlamasına olanak sağlar.

* Swagger, geliştiricilerin RESTful Web hizmetlerini tasarlamasına, oluşturmasına, belgelemesine ve tüketmesine yardımcı olan geniş bir araç ekosistemi tarafından desteklenen açık kaynaklı bir yazılım çerçevesidir.
* Swagger, makinelerin okuyabilmesi için API'lerinizin yapısını tanımlamanıza olanak tanır.
* API'lerin kendi yapılarını tanımlama yeteneği, Swagger'daki tüm harikaların temelidir
* xml şemasına benzer ancak Json için

1. **json vs gson**

GSON, Java Nesnelerini JSON temsillerine dönüştürmek için kullanılabilen bir Java kitaplığıdır. Bir JSON dizesini eşdeğer bir Java nesnesine dönüştürmek için de kullanılabilir. GSON, kaynak koduna sahip olmadığınız önceden var olan nesneler de dahil olmak üzere keyfi Java nesneleri ile çalışabilir. JSON ise insanlar tarafından okunabilen ve çok çeşitli diller tarafından desteklenen bir veri formatıdır.

* JSON, anahtar ve değerlere sahip bir formattır
* GSON bir dönüştürme sürecidir
* ○ java'dan json'a (serileştirme),
* ○ json'dan java'ya (deserialization)

1. **Request nasıl ve nereye gönderiyorsunuz?**

Rest kullandığım için uç noktaları var. Geliştiricilerim genel URL'ler oluşturur ve istekler buURL'ye gönderilir.

1. **Web hizmetleri dışında herhangi bir API kullanıyor musunuz?**

Tarayıcı için Selenium API, veritabanı için JDBC ve API için RestAssured kullanıyorum

1. **API'niz için API dokümantasyon web siteniz var mı?**

Evet, api dokümantasyonumuz için swagger kullanıyoruz ve API uç noktalarının açıklamaları ve yönergeleri burada yer alıyor.

1. **Projenizde API'yi nasıl test ediyorsunuz?**

* API testlerini öncelikle manuel olarak Postmande yaptıktan sonra JUnit, Cucumber ve RestAssured kütüphanelerini kullanarak otomatize ettim.
* API testi JSON formatı kullanılarak ObjectMapper ve deserialization ile gerçekleştirildi.

Şu anki projemde sadece şirketimizin api'sini değil, diğer harici api'leri de test ediyoruz.

Örneğin, yetkili son kullanıcının bilgilerini veritabanımıza kolayca aktarmak için LinkedIn api'sini kullanıyoruz.

Bir test uzmanı olarak bir API isteği gönderiyoruz ve durum kodunu, yanıt gövdesini doğruluyoruz ve api URL'sinin uç noktalarının beklendiği gibi çalıştığını kontrol ediyoruz.

Örneğin, projemde API 57'nin Pozitif/Negatif testini de yapıyorum

Positive - Geçerli istekler, başlıklar, parametreler ve Json gövdesi gönderiyorum ve yanıtın 200/201 olduğunu doğruluyorum

Negative- Geçersiz istekler, başlıklar, parametreler ve gövde gönderiyorum, durumun 200 olmamasını bekliyorum

1. **Rest api'yi nasıl test kullanıyorsunuz?**

Her REST API uç noktasının beklendiği gibi çalışıp çalışmadığını doğrularım.

Manuel API testi için POSTMAN kullanıyorum ve otomasyon için Java'da RESTASSURED kütüphanesini kullanıyorum.

POST, PUT, GET, DELETE türü istekleri gönderiyorum ve yanıt durum kodunu ve yanıt gövdesini, başlığını doğruluyorum.

Ayrıca API'nin pozitif ve negatif testlerini yapıyorum.

Pozitif test yaptığımda, geçerli istek parametreleri, geçerli başlıklar, geçerli istek json gövdesi gönderiyorum ve şunları doğruluyorum

* response status code 200 başarılı ve Json response body verileri de beklenenle eşleşiyor.

Negatif test yaptığımda, geçersiz istek (invalid request) parametreleri veya geçersiz başlıklar veya geçersiz istek json gövdesi gönderiyorum ve bu yanıtı doğrulayın

* status code 200 değil ve Json yanıt gövdesi hata mesajı içeriyor.

1. **Tüm API endpoints tüm Http protokollerini kullanabilir mi?**

Değişir, API geliştiricim bu URL'nin GET, POST, PUT veya DELETE istekleri ile çalışıp çalışmadığına karar verir.

1. **API'nizi manuel olarak nasıl test ediyorsunuz?**

Postman kullanıyorum → REST API URL'yi test eden bir REST API istemci aracıdır

1. **API testi için hangi araçları kullanıyorsunuz?**

Manuel test için Postman

Rest Assured kütüphane

<https://medium.com/kodgemisi/rest-api-tasarim-rehberi-2f004a87426d>

1. **Rest API'de İstek türleri nelerdir?**

Get, Post, Put ve Delete istekleri vardır

○ Get: Okunan verileri al

○ Post: Gönderi veri oluşturur

○ Put: Veri güncellemelerini koy

○ Delete: Sil verileri siler

1. **REST API'de başlıklar nelerdir?**

Accept.(Content Type.JSON) türünü kullanıyorum - aldığım şeyin JSON veya XML biçiminde olması gerektiğini kontrol eder

Ve ContentType.(Contenttype.Json) - gönderdiğim şeyin JSON biçiminde olması gerektiğini kontrol eder

1. **RestAssured Kütüphanesi nedir?**

BDD formatında olan ve java kodunu deserialization ve serialization kullanarak entegre etmeye yardımcı olan web hizmeti olmayan bir api Json'dan veri ayıklamak ve verileri depolamak, doğrulamak ve geçerli kılmak için bir java nesnesine dönüştürmek bekleniyor.

**Terimler**

* Gherkin : Genellikle BDD yapılı testlerde kullanılır. Given, when, then keywordleri ile aşamalı olarak scriptleri daha anlaşılır halde kullanmaya yarayan syntax yapısıdır.
* Source : Http requestinin oluşturulup hedefe yollandığı makine.
* Destination : Http requestinin kaynaktan geldiğini kabul edip onaylayan ve yanıt dönen makine.
* Chain method : when().get().then().body() gibi zincirleme çalışabilen methodlara verilen genel isim.

<https://medium.com/bili%C5%9Fim-hareketi/java-rest-assured-ile-api-t-6e2516dcd801>

1. **Projenizde Enum'u nasıl kullanıyorsunuz?**

Yanıt türümün JSON biçimi olduğundan emin olmak için content Type kullanıyorum

1. **JsonPath nedir?**

Response bodyi doğrulamanın başka bir yolu

JsonPath j=response.jsonpath.

1. **Response datanın boyutunu doğrulamak için hangi yöntemleri kullanıyorsunuz?**

Hamcrest'in Eşleştiricilerini kullanıyorum

**○ hasItems()**

**○ equalTo()**

1. **Response interface nasıl kullanabilirim?**

* API gönderilen request sonucunda dönen cevabın Responce interface’i türünde oluşturulan objeye atama yapılır.

µ **Raporlama (Reporting)**

Mvn Verify başarısız olsa bile testleri çalıştıracaktır (başarısızlığı yok sayar)

○ Tüm testlerin bitmesini bekler

Hatayı yok sayar b.c derleme yapılandırmamızda buna sahibiz

**<testFailureIgnore>true</testFailureIgnore**>

Verify, testten sonra gelen bir Maven yaşam döngüsüdür

Mvn testi, bir şey başarısız olursa testi çalıştırmayı durduracaktır

Aldığımız orijinal html raporu o kadar da iyi değil, istatistiksel verilere ihtiyacımız var

Örn; "html:target/cucumber-report" → Kaç testin başarılı/başarısız yüzdesi olduğu gibi

Daha fazla istatistiksel veri raporlaması için Cucumber Sandwich kullanacağız (bu pom xml'de bir bağımlılık dosyasıdır)

● cukesrunner'da; "json:target/cucumber.json" ekleyin

JSON dosyasından alınan bir html raporudur → Bu raporun çalışma şekli, raporu oluşturmak için Json dosyasının kullanılmasıdır

○ Sürüm 3.15 (videodan)

Pom'a yeni bir build xml ekleyin (zaten pom dosyanızda, TestProject adlı dosyada)

Sadece bu json raporunu alacaksınız (grafikler ve istatistikler içeren cucumber raporu. Bu rapor yalnızca SİZİN görmeniz için yerel olmalıdır, Jenkins için değil) SADECE MVN Verify'ı çalıştırırsanız

■ ANCAK ÇALIŞTIRDIKTAN SONRA HER ZAMAN BİR JSON DOSYASI (cucumber raporundan farklı) ALACAKSINIZ

TEST, HATTA W.O DOĞRULAMAK

Bu JSON dosyası Jenkins için çok önemlidir- cucumber rapor eklentisi için

**● TestProject derlemesi:**

○ <id>yürütme</id>

○ <phase>verify</phase>- html (json) raporunun yalnızca verify kullanıldığında oluşturulmasının nedeni budur

○ <goals>

○ <goal>generate</goal> </goals>

Rapor size bir json dosyası da verecektir

Testleri verify kullanarak çalıştırmak için pom dosyasına sağ tıklayın ve maven build...

Ayrıca parametreler de ekleyebilirsiniz (xml dosyası olan runner değişkeni ve değeri gibi) - Hedefleri yazın: verify

Bunu komut satırında çalıştırmak için

Pom dosyasının konumuna gidin ve mvn verify yazın

○ Syntax mvn<lifecycle/goal>

mvn verify kullanarak Yürütme Sırası

1. Pom dosyasına karşı çalıştırın

2. Pom dosyası xml dosyasını çalıştırır

3. Xml cukesrunner dosyasını çalıştırır

4. Cukesrunner, cucumber özellik dosyasını/testini çalıştırır

Json cucumber raporu ekran görüntüsü gösteriyor mu?

1. **Limiti 100 olan parametreye ve çalışan kimliği = 100 olan yol parametresine ihtiyaç duyduğum bir yöntemi nasıl yazabilirim?**

● Şu şekilde yazardım;

**.and().params("limit",100)**

**.and().pathParams("employee\_id", 110)**

1. **Backend-API nedir?**

Uygulama mantık kodunun olduğu yerdir. Koşullarınız vb.

**Nasıl test edilir?**

1) Manuel olarak → Postman vb. araçları kullanarak. İstek göndererek ve yanıtları doğrulayarak.

2) Otomasyon → Java + RestAssured Kütüphanesi

<https://medium.com/@ozaneseriu/postman-ile-backend-api-otomasyonuna-i%CC%87lk-ad%C4%B1m-13573eacc0c2>

**Kaynak: Albert Document**

Zeliha Öznük