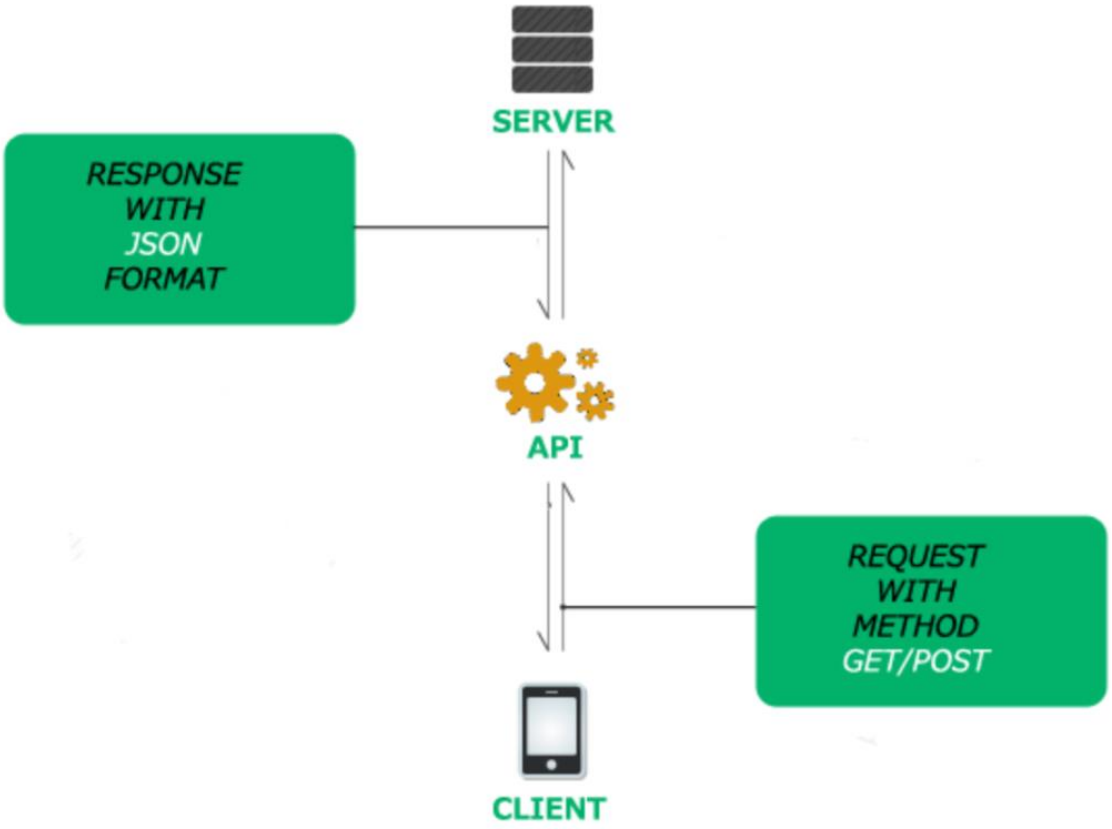
RETROFİT

Retrofit kütüphanesi, Square tarafından geliştirilen Android, Java ve Kotlin için tip güvenli bir REST istemcisidir. Retrofit kütüphanesi sayesinde, API'lerle kimlik doğrulama yapma ve etkileşimde bulunma, Ok Http ile ağ istekleri gönderme konusunda güçlü bir çerçeveye erişim sağlarız. Bu kütüphane yardımıyla, bir web API'den JSON veya XML verilerini indirmek kolaylaşır. Retrofit kütüphanesinde, veriler indirildikten sonra, cevaptaki her "kaynak" için tanımlanması gereken Düz Eski Java Nesnesi (POJO) içine ayrıştırılır. Retrofit, REST tabanlı bir web servisi üzerinden veri almayı ve yüklemeyi hızlı ve kolay bir şekilde yapmamızı sağlayan bir kütüphanedir.

Retrofit, HTTP isteklerini ve yanıtlarını alma, gönderme ve oluşturma sürecini yönetir. Uygulamanın hata verip çökmesini önlemek için sorunları çözer. Bağlantı havuzları kullanarak gecikmeyi azaltır. Yanıtları önbelleğe almak için kullanılır, böylece aynı isteklerin tekrar gönderilmesi önlenir.



REST Nedir?

REST(Representational State Transfer) adından da anlaşılacağı gibi uzak sistemdeki mantıksal kaynakları HTTP protokolü ile GET, POST, PUT, PATCH, DELETE methodlarını çağırarak kullanmaktır.

KURULUM

Öncelikle build.gradle dosyasına alttaki gibi dependency ekliyoruz.

```
implementation(libs.retrofit)
```

JSON çıktılarını POJO classlara dönüştürmek için GSON kullanacağız. Retrofit bize GSON için ayrı bir converter kütüphanesi sağlıyor. Bunu da aynı şekilde dependecylere ekliyoruz.

```
implementation(libs.converter.gson)
```

Network istekleri yapacağımız için projemizde INTERNET iznini de almamız gerekiyor. Onun için Manifest dosyasına alttaki izni ekliyoruz.

```
<uses-permission android:name="android.permission.INTERNET" />
```

GET

Veri almak için kullanılır. En çok bilgi çekme işlemlerinde kullanılır. Örneğin, bir kullanıcı listesini, haber kaynaklarını ya da herhangi bir veri kaydını getirmek için GET metodu kullanılır. GET istekleri, URL üzerinden sorgu parametreleri ile veri gönderilebilir.

```
@GET("group/{id}/users")
Call<List<User>> groupList(@Path("id") int groupId);
```

POST

Yeni bir kaynak oluşturmak için kullanılır. POST metodunda gönderilen veriler genellikle HTTP isteğinin gövdesinde yer alır. Örneğin, yeni bir kullanıcı oluşturmak veya bir form göndermek için POST kullanılır.

```
@POST("users/new")
Call<User> createUser(@Body User user);
```

PUT

Mevcut bir kaynağı güncellemek için kullanılır. PUT, genellikle belirli bir kaydı tamamen güncellemek için kullanılır. Gönderilen veri, hedef kaynağın yeni versiyonunu temsil eder ve genellikle HTTP gövdesinde taşınır.

```
@PUT("user/photo")
Call<User> updateUser(@Part("photo") RequestBody photo, @Part("description") RequestBody description);
```

DELETE

Mevcut bir kaynağı silmek için kullanılır. Silinecek öğenin genellikle URL üzerinden belirtilir. Örneğin, belirli bir kullanıcıyı sistemden kaldırmak için DELETE metodu kullanılabilir.

```
@DELETE("/typicode/demo/posts/1")
suspend fun deleteEmployee(): Response<ResponseBody>
```

RETROFİT İLE EN ÇOK KULLANILAN DİĞER ANNOTATION'LAR

@Path

Retrofit kütüphanesinde @Path annotation'ı, dinamik URL yapılarını desteklemek için kullanılır. Bu annotation, bir API isteğinin URL'sinde değişken olarak yer alan bölümleri dinamik bir şekilde değiştirmenizi sağlar. @Path kullanımı, API'nin aynı temel URL yapısını kullanarak farklı kaynaklara ulaşmasına olanak tanır, bu da uygulamanın esnekliğini ve yeniden kullanılabilirliğini artırır.

```
@GET("/image/{id}")
Call<ResponseBody> example(@Path("id") int id);
```

@Query

HTTP isteklerine URL sorgu parametreleri eklemek için kullanılır. Bu annotation, genellikle GET metotlarında kullanılır ve belirli kaynaklara daha spesifik erişim sağlamak veya belirli filtreleme işlemleri yapmak için çok işlevseldir.

```
@GET("/friends")
Call<ResponseBody> friends(@Query("group") String group);
```

@Body

HTTP isteğinin gövdesine bir nesne eklemek için kullanılır. Genellikle POST, PUT ve PATCH gibi metotlarla kullanılan bu annotation, gönderilecek veriyi doğrudan HTTP isteğinin gövdesine yerleştirir. @Body annotation'ı ile işaretlenen parametre, Retrofit tarafından serileştirilerek JSON (veya başka bir format) olarak sunucuya gönderilir. Bu serileştirme işlemi genellikle Gson, Jackson veya Moshi gibi bir JSON dönüştürücü kullanılarak yapılır.

```
@POST("/api/v1/create")
suspend fun createEmployee(@Body requestBody: RequestBody): Response<ResponseBody>
```

@Field

Form-encoded gönderimler için kullanılır ve genellikle POST ya da PUT gibi HTTP metotlarıyla birlikte kullanılır. Bu annotation, bir HTTP isteğinin gövdesine bir veya birden fazla veri parçası eklemek için kullanılır, ancak bu veriler URL'de değil, HTTP isteğinin gövdesinde form-data olarak gönderilir.

```
@FormUrlEncoded
@POST("/list")
Call<ResponseBody> example(@Field("name") String... names);
```

@FormUrlEncoded

Form verilerinin HTTP istekleri üzerinden nasıl gönderileceğini belirlemek için kullanılır. Bu annotation, gönderilen verilerin application/x-www-form-urlencoded MIME tipinde olduğunu belirtir, yani form verileri anahtar-değer çiftleri olarak URL-encoded formatında gövdeye eklenir. Genellikle POST veya PUT HTTP metotlarıyla kullanılır.

Bu annotation eklenmiş bir metotta **@Field** veya **@FieldMap** gibi annotation'lar kullanılarak anahtar-değer çiftleri belirlenir.

```
@FormUrlEncoded
@POST("/post")
suspended fun createEmployee(@FieldMap params: HashMap<String?, String?>): Response
```

@Part ve @PartMap

@Part ve @PartMap annotation'ları, Retrofit kütüphanesinde çok-partili (multipart) HTTP istekleri göndermek için kullanılır. Bu annotation'lar, genellikle dosya yüklemeleri gibi işlemler için kullanılır ve HTTP gövdesine bir veya birden fazla dosya veya veri parçası eklemek amacıyla tasarlanmıştır. Bu tür istekler, multipart/form-data MIME tipinde kodlanır.

@Part annotation'ı, bir **@Multipart** annotation'ı ile işaretlenmiş metotta kullanılır ve gönderilecek her bir parçanın (part) API'ye nasıl gönderileceğini belirtir. **@Part** kullanımı genellikle dosya yükleme senaryolarında görülür, ancak metin gibi diğer veri türlerini de içerebilir.

```
@Multipart
@POST("/")
Call<ResponseBody> example(
    @Part("description") String description,
    @Part(value = "image", encoding = "8-bit") RequestBody image);
```

@PartMap ise, birden fazla parçanın kolaylıkla gönderilmesi için kullanılır. Map yapısı kullanılarak anahtar-değer çiftleri halinde verileri saklar ve gönderir. @PartMap kullanılırken, Map'in anahtarları parçaların isimlerini, değerleri ise gönderilecek RequestBody nesnelerini veya MultipartBody.Part nesnelerini temsil eder.

```
@Multipart
@POST("/post")
suspend fun uploadEmployeeData(@PartMap map: HashMap<String?, RequestBody?>): Respons
```

@Multipart

Retrofit kütüphanesinde çok parçalı (multipart) HTTP isteklerini tanımlamak için kullanılır. Bu tür istekler, genellikle dosyalar ve büyük veri miktarlarını gönderirken kullanılan multipart/form-data MIME tipi ile kodlanır. @Multipart ile işaretlenmiş metotlar, genellikle @Part veya @PartMap annotation'ları ile birlikte kullanılır, böylece istek gövdesinde birbirinden farklı veri türleri ve içerikler gönderilebilir.

@Header

Retrofit kütüphanesinde HTTP isteklerine dinamik header (başlık) eklemek için kullanılır. Bu, her istekte farklı değerlere sahip olabilecek başlıklar eklemeyi sağlar, örneğin kullanıcı belirteçleri (tokens), oturum bilgileri veya diğer özelleştirilebilir veriler gibi.

```
@GET("/")
Call<ResponseBody> foo(@Header("Accept-Language") String lang);
```

HTTP KODLARI

Http Durum Kodları nedir?

İki taraflı bir etkileşim, kullanıcılar bir web sitesini ziyaret etmek istediğinde gerçekleşir. Bu etkileşimin bir tarafında tarayıcı, diğer tarafında ise sunucu bulunur. Bu etkileşim, kullanıcının bir web sayfasına erişmek istemesiyle başlar. Bir kullanıcı web sayfasına eriştiğinde, tarayıcı aslında sayfayı görüntülemek için sunucuya bir istek gönderir. Sunucu, bu isteğe üç haneli bir durum kodu ile yanıt verir. Sunucunun tarayıcıya verdiği üç haneli yanıtlar HTTP durum kodları olarak adlandırılır. Bazen bu durum kodları bir hatayı gösterebilirken, diğer zamanlarda sayfanın sorunsuz yüklendiğini gösterebilir. Dolayısıyla, HTTP durum kodlarının her zaman hatalar olduğunu varsaymak doğru değildir.

HTTP Durum Kodları Neden Önemlidir?

HTTP durum kodları hem kullanıcılar hem de web siteleri için çok önemli bir unsurdur. Kullanıcılar erişmek istedikleri web sayfalarında bir sorun olduğunda, sorunun kökenini bilmek isterler. Bu yüzden HTTP durum kodları kullanıcılar için çok önemlidir. Aynı şekilde, belirli sorunlar nedeniyle sayfaları yüklenmeyen web siteleri hem kullanıcı deneyimi hem de SEO performansı açısından olumsuz etkilenebilir. Bu nedenle, sorunun kökenini anlamak ve çözmek için sunucunun kullanıcılara hangi yanıt kodlarını gönderdiğini bilmek çok önemlidir.

HTTP Durum Kodları Sınıfları

1xx: Sunucunun tarayıcının isteğini aldığını ve sürecin başladığını belirten bilgilendirici durum kodları.

2xx: Sunucunun tarayıcının isteğini aldığını ve anladığını ve isteğin başarılı olduğunu belirten durum kodları.

3xx: İstenen kaynağın başka bir konuma taşındığını ve yönlendirileceğini belirten durum kodları.

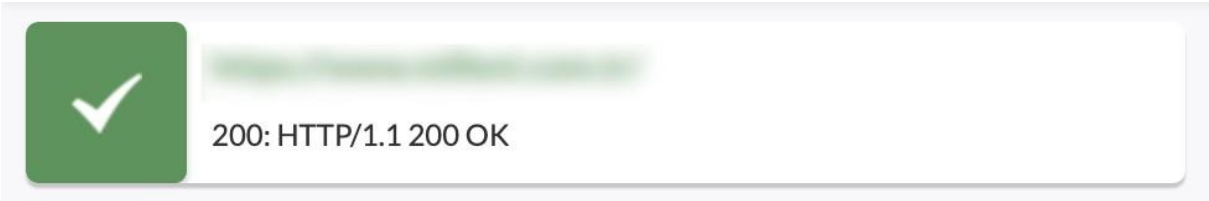
4xx: İsteğin yerine getirilemediğini ve istenen web sayfası veya web sitesinin kullanılamadığını belirten durum kodları.

5xx: Sunucunun tarayıcının isteğini başarıyla aldığını, ancak sunucu tarafındaki sorunlar nedeniyle isteğin yerine getirilemediğini belirten durum kodları.

Yaygın Durum Kodları

200 Durum Kodu (OK)

Bu ideal bir durum kodudur. Bir web sayfası sorunsuz yüklendiğinde, sunucu tarayıcıya 200 durum kodu ile yanıt verir. Sunucu tarayıcıya 200 durum kodu ile yanıt verdiğinde, ziyaretçi ve web sitesi için her şeyin olması gerektiği gibi olduğunu söyleyebiliriz.

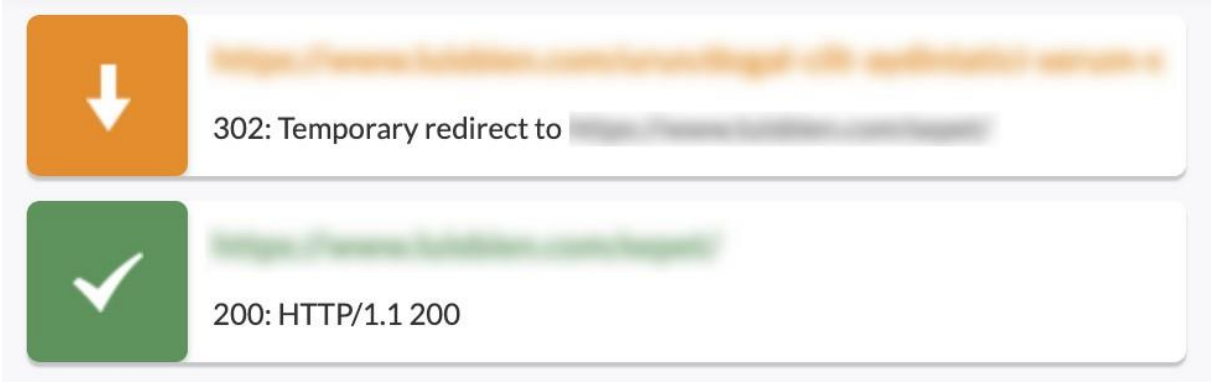


301 Durum Kodu (Moved Permanently)

Bir web sayfasının başka bir web sayfasına kalıcı olarak taşındığı durumlarda ziyaretçiyi otomatik olarak yönlendiren bir durum kodudur. 301 durum kodunu kullanarak web sayfalarına yönlendirme yapılırsa ve bu sayfalar içerik açısından birbirine oldukça benzerse güç kaybı en aza indirgenebilir. Bu nedenle, web sitesi taşıma gibi işlemler için tavsiye edilen en önemli durum kodlarından biridir.

302 Durum Kodu (Found)

Bu durum kodu, bir web sayfasının geçici olarak başka bir web sayfasına taşındığını gösterir. 301 durum kodundan farklı olarak, istenen sayfanın yeniden aktif hale geleceği belirli durumlar için kullanılır. Bu durumlar, web sayfasının test ediliyor olması, bakımda olması veya bir e-ticaret sitesi ise belirli bir ürünün geçici olarak tükendiği durumlar olabilir. Ancak, kullanıcılar 301 yönlendirmesi ile 302 yönlendirmesi arasındaki farkı anlayamaz. Web sayfasına erişen kullanıcılar otomatik olarak diğer sayfaya yönlendirilir.



403 Durum Kodu (Forbidden)

Bu durum kodu, kullanıcının sunucuya yaptığı isteğe yanıt olarak, kullanıcının istenen web sayfasına erişimine izin verilmediğini veya web sayfasının yasaklandığını gösterir.

404 Durum Kodu (Not Found)

Bu durum kodu, istenen web sayfasının sunucuda bulunamadığını gösterir. Web sayfası silinmiş veya URL'si değiştirilmiş olabilir. Ancak, 404 durum kodu bu durumun geçici mi yoksa kalıcı mı olduğunu açıklamaz. Kullanıcılar istedikleri bir web sayfasında 404 durum kodu ile karşılaştıklarında, genellikle web sitesinden ayrılıp farklı sitelere yönelirler. Bu, özellikle çok trafik alan veya URL'si kullanıcılar tarafından iyi bilinen sitelerde web sitesine olumsuz etki yapar. Bu nedenle, mümkünse 404 durum koduna sahip web sayfalarının eşdeğer bir sayfaya yönlendirilmesi önerilir. Ancak, web sayfası bir süre sonra tekrar aktif olacaksa veya bir e-ticaret sitesinde ürün tekrar stokta olacaksa, sayfanın 404 durum kodu ile kalması daha uygun olur. Sayfa 404 kodu ile kalmışsa, onları diğer sayfalara yönlendirerek izleyici kitlesini korumaya çalışmalıyız. Özelleştirilmiş 404 sayfaları ile, kullanıcıların web sitesinden ayrılmak yerine farklı sayfalara yönlendirilmesine yardımcı olabiliriz.