RETROFIT İLE HTTP METOTLARI ÇEŞİTLERİ :

GET, POST, PUT, DELETE

@GET

Sunucudan veri çekmek için kullanılır. Verileri model olarak tanımladıktan sonra istediğimiz verileri uygulama içerisinde kullanıma hazır ve tanımlanmış bir şekilde göstermemizi sağlar. Kullanım örneği aşağıdaki gibidir.

```
public interface GitHubService {
  @GET("users/{user}/repos")
  Call<List<Repo>> listRepos(@Path("user") String user);
}
```

@POST

Sunucudan verileri almak için kullanılır. Uygulama içerisinden gelen veya kullanıcının kaydetmek istediği bilgileri sunucu tarafına göndermek için kullanılan metottur. Kullanım örneği aşağıdaki gibidir.

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

@PUT

Sunucudaki verilerin değiştirilmesi istendiğinde kullanılan metottur.

Çoklu veri değiştirmek istendiğinde @Multipart annotation ‘ı kullanılmalıdır ve her parça veri @Part annotation’ı alarak belirtilmelidir.

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

@DELETE

Sunucudaki bir verinin silinmesi isteğini göndermek için kullanılan metottur. Aynı GET metodundaki gibi sunucudaki kaydını tespit etmek amacıyla @Path annotation’ı ile kullanılır.

```
@DELETE("/api/item/{id}")
Call<Response> deleteItem(@Path("id") int itemId);
```

RETROFIT ILE EN ÇOK KULLANILAN DİĞER ANNOTATİON'LAR

@Path

Bu annotation sunucuda verinin yerini belirtmek için kullanılır. Örnek olarak aşağıdaki gibi get metodunu kullanırken @Path("user") dediğimizde aslında sunucudaki user anahtar kelimeli verileri belirtmiş oluyoruz.

```
@GET("users/{user}/repos")
Call<List<Repo>> listRepos(@Path("user") String user);
```

@Query

Query annotation'ı Get veya Post ile kullanılabilir. Örnek olarak Get metodu ile kullanılmak istendiğinde sunucudan gelen veirlerin belirli bir aralıkta, sıralamada veya bir filtrelenmiş durumda olmasını sağlar.

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

@Body

Body uygulama tarafından Retrofit yoluyla sunucuya gidecek olarak verilerin tanımlanmasında kullanılır.

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

@Header

Yapılan isteğe bir başlık eklemek için kullanılır. Böylece sunucu ile uygulama arasında yapılan iletişim sırasında kimlik doğrulama görevini gerçekleştirir.

```
@GET("user")
Call<User> getUser(@Header("Authorization") String authorization)
```

@Headers

Birden fazla header kullanımında durumu kolaylaştırmak için @Headers annotation'ını da kullanabiliriz.

```
@Headers({
    "Accept: application/vnd.github.v3.full+json",
    "User-Agent: Retrofit-Sample-App"
})
@GET("users/{username}")
Call<User> getUser(@Path("username") String username);
```

@Field

Bu annotation kullanıcıdan alınan form doldurma gibi POST isteklerinde kullanılan @FormUrlEncoded annotation'ının içindeki verileri tanımlamak için kullanılır.

```
@FormUrlEncoded
@POST("user/edit")
Call<User> updateUser(@Field("first_name") String first, @Field("last_name") String last);
```

Burda user/edit endpointine gönderilecek olan first_name ve last_name parametrelerini belirtmek için kullanıyoruz.

HTTP STATUS CODES

HTTP Status Codes (Durum kodları) Sunucuya istek attığımızda sunucudan istemciye dönen yanıttır.

En çok kullanılan HTTP Status Code'lar :

200 (Success/OK)

Tarayıcı ve sunucu tarafında her şeyin yolunda olduğu anlamına gelen ideal durum kodudur. Bir geliştirici olarak görmeyi en sevdiğim durum kodu.

301 (Permanent Redirect)

Kalıcı yönlendirme olarak geçen bu durum kodu bir web sayfasının kalıcı olarak başka bir web sayfasına yönlendirildiği ve sayfayı ziyaret eden kullanıcının da otomatik olarak yönlenmesini sağlayan durum kodudur.

302 (Temporary Redirect)

Bu durum kodu bir web sayfasının geçici olarak bir başka web sayfasına yönlendirildiğini ifade eden durum kodudur.

304 (Not Modified)

Evet gerçekten ismi modifiye edilmemiş... İnanın bana kontrol ettim. Bu durum kodu uygulamanın çalıştığı cihazın önbelleğinde depolanan kaynakların değişmediğini belirten durum kodudur.

400 (Bad Request)

İstemci bir istek attığında sunucudan bu isteğe karşılık bir geri dönüş yapamadığında bu hata kodu görüntülenir. Yanıltıcı veya geçersiz bir istek gönderilmiş olabilir.

401 (Unauthorized Error)

Uygulama içerisinde bir yetkilendirme durumu sırasında sunucuya gönderilen geçersiz kimlik bilgileri veya yetersiz yetki durumların belirten bir hata kodudur.

403 (Forbidden)

İstemci tarafından gönderilen isteğin anlaşıldığı fakat yine de sunucu tarafından ret alındığı durumlarını belirtir. Yani bilinçli olarak kullanıcıya kullanım kısıtlaması geldiği bir durum gibi düşünülebilir.

404 (Not Found)

Sunucu , istemci tarafından yapılan kaynak isteğine karşı bir sonuç bulamadığında karşılaşılan durum kodudur.

500 (Internal Server Error)

Sunucuyla bağlantı sırasında hata oluştuğunda ve istenen sayfaya ulaşılamadığında bu mesaj görüntülenir.

501 (Not Implemented)

İstemci tarafından bir istek yapıldığında, sunucu isteğe karşılık bir kaynağı desteklemediğinde veya sunucu yapılan isteği tanımadığında karşılaşılan durum kodudur.