RETROFIT

Retrofit, Android için geliştirilmiş, servis işlemlerinde kullanılan ve servisten gelen verileri uygulamamıza çekmemizi sağlayan bir kütüphanedir. Bu kütüphaneyi projemize ekledikten sonra internet izni alıp gerekli Service ve Client sınıflarını kurduktan sonra artık veriyi çekip kullanabiliriz.

Fakat bu veriyi çekerken servis sınıfımızda uygulayabileceğimiz veri çekme metotları vardır. Bu metotları kullanarak ilgili veriyi nasıl çekmemiz gerektiğini belirtiriz. İlk metodumuzdan başlayalım.

GET

Get metodu ile http protokolünden diğer deyişle Api'den gelen verileri almak için kullanılır. Sunucu url'sine istek gönderir yani sunucuya herhangi bir veri gönderilmez. Url'e ise parametreler verebiliriz.Bu parametreyi verirken fonksiyonda @Path anotasyonunu kullanırız.Aşağıda parametreli ve parametresiz get isteği örnekleri bulunmaktadır.

Parametreli:

```
@GET("users/{userId}")
fun getUser(@Path("userId") userId: Int): Call<User>
```

Parametresiz:

```
@GET("products")
fun getProducts(): Call<List<Product>>
```

POST

Genellikle sunucuya veri gönderilirken kullanılır ve bu veri json veya form verisi olabilir.Bu isteği yaparken dikkat etmemiz gereken şey sunucunun bizden istediği şekilde veriyi göndermemizdir. Post işlemiyle sunucuda veri güncelleme, yeni veri ekleme işlemlerini uygulamamızdan kolaylıkla yapabiliriz. Get isteğine göre daha riskli ve güvensizdir.

Sunucuya post ile user nesnesi gönderimi

```
@POST("users")
fun createUser(@Body user: User): Call<User>
```

Sunucuya post ile dosya gönderimi

```
@POST("upload")
fun uploadFile(@Body file: RequestBody): Call<String>
```

Sunucuya post ile form verisi gönderimi

```
@FormUrlEncoded
@POST("submit")
fun submitForm(
    @Field("username") username: String,
    @Field("password") password: String
): Call<String>
```

PUT

Put isteği post isteğine benzer tek farkı ise put isteği sadece güncellemeyi sağlar. Yani sunucuda olan veriye gönderdiğimiz veriyi koyar ve güncelleştirir.Ayrıca yeni veri gönderimi de yapılabilir.Post işleminden farkı ise şudur: Put isteği genellikle belirli bir veriyi güncelleştirmek için kullanılır yani Post isteği her seferinde sunucuyu değiştirirken put ise değiştirmeyebilir.

Kayıtlı user verisini güncelleştirme isteği:

```
@PUT("users/{userId}")
fun updateUser(@Path("userId") userId: Int, @Body updatedUser: User): Call<User>
```

Yeni user nesnesi ekleme isteği:

```
@PUT("user")
fun replaceUser(@Body newUser: User): Call<User>
```

DELETE

Genellikle var olan kaydı silmek için kullanılır ve hangi kaydı silmek istediğimizi de belirtmemiz gerekir. Aşağıdaki örnekleri inceleyebiliriz.

Var olan user nesnesini gönderdiğimiz id'den bulup silen delete isteği:

```
@DELETE("users/{userId}")
fun deleteUser(@Path("userId") userId: Int): Call<Void>
```

Tüm user nesnelerini silen delete isteği:

```
@DELETE("users")
fun clearUsers(): Call<Void>
```

ANNOTATİONS(Anotasyonlar)

Sunucuya yapılan Http isteklerini anlattık şimdi bu http istekleriyle beraber kullanılan ve uygulamamızda bize yardımcı olan, Retrofit içerisinde bulunan anotasyonları anlamaya başlayalım.

1- **@Path**
Kullandığımız url'e dinamik veri gönderirken kullanırız.Url içinde {} işaretlerini kullanarak değeri yazarız.

```
@GET("users/{userId}")
fun getUserById(@Path("userId") userId: Int): Call<User>
```

2- **@Query**
Url içindeki sorgu parametrelerini yönetmemizi sağlar.Url'de sorgular ? işaretinden sonra gelir.

```
@GET("users")
fun getUsers(@Query("page") page: Int): Call<List<User>>
```

3- **@QueryMap**
@Query ile aynı işlemi yapar ama birden fazla veriyi mapleyerek göndermemizi sağlayan bir anotasyondur.

```
@GET("users")
fun getUsers(@QueryMap options: Map<String, String>): Call<List<User>>
```

4- **@Header**
http başlıklarını belirlememizi sağlar.yetkilendirme işlemlerini sağlar. Bu başlıklar Authorization,Accept,Content-Type,Cache-Control gibi başlıklar olabilir.

```
@GET("user")
fun getUser(@Header("Authorization") token: String): Call<User>
```

5- @QueryName

Url'de belirli bir sorgu parametresini belirtiyor.

```
@GET("user")
fun getUser(@QueryName("id") userId: Int): Call<User>
```

6- @Field

Form verilerini post ile gönderirken kullanılır.Form işlemlerinde sıklıkça kullanılır.

```
@POST("register")
fun registerUser(
    @Field("username") username: String,
    @Field("password") password: String)
: Call<User>
```

7- @FieldMap

Form verileri gönderirken mapleyerek daha çok veriyi daha dinamik göndermemizi sağlar.

```
@POST("register")
fun registerUser(@FieldMap userData: Map<String, String>): Call<User>
```

8- @Part

Özellikle dosya işleme yüklemlerinde kullanılır.

```
@POST("upload")
fun uploadFile(@Part file: MultipartBody.Part): Call<String>
```

9- @Headers

Herhangi bir isteğe daha özel ve çoklu başlık belirtmemizi sağlar.

```
@Headers("Cache-Control: max-age=640000")
@GET("users")
fun getUsers(): Call<List<User>>
```

10-@Streaming

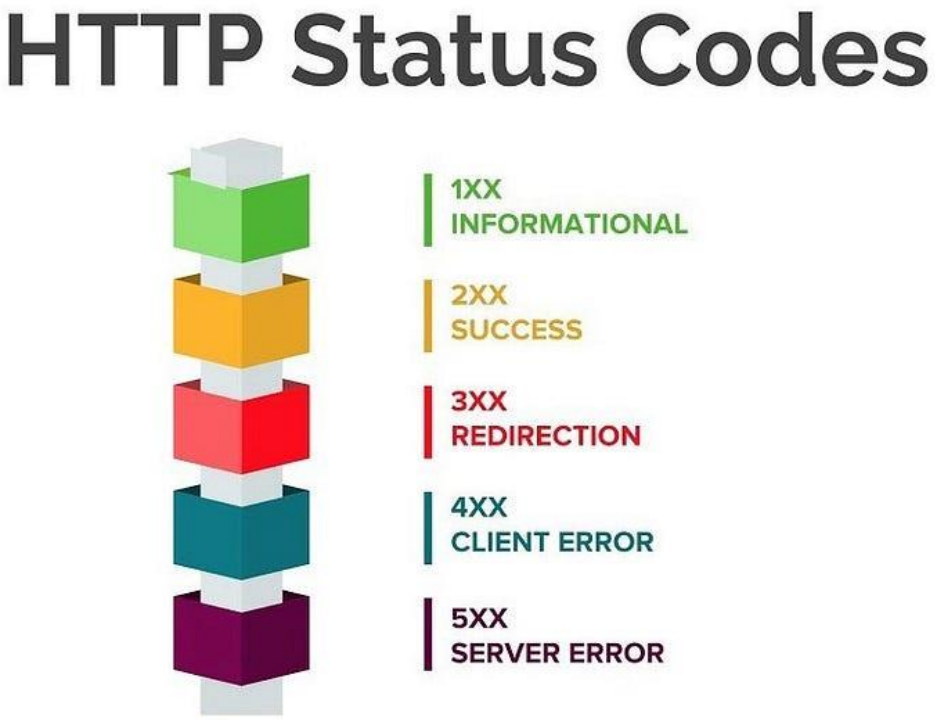
Büyük dosyalar büyük veri akışları için kullanılır.

```
@Streaming
@GET("files/largeFile.zip")
fun downloadLargeFile(): Call<ResponseBody>
```

11-@Body

http isteğinin gövdesine veri eklememizi sağlar.

```
@POST("users")
fun createUser(@Body user: User): Call<User>
```



İnternetin aktif şekilde kullanıldığı teknoloji devrinde , kullanıcıya internetle ilgili herhangi bir problem, uyarı veya bilgilendirme gereği duyduğumuzda http kodlarını kullanırız.Bu kodlar 3 haneli olarak belirlenmiştir ve her kodun kendi bir anlamı vardır. 100-500 arasında belirlenen bu kodlar problem büyüklüğüne göre 100'den 500'e doğru artar.Hadi kodların bize iletmek istedikleri durumlara bakalım.

INFORMATION CODES

100

Continue olarak isimlendirilir.Sunucunun istekleri başarılı şekilde devam ettirilir.

101

Protokol değişikliği talebi gelmiş ve başarılı şekilde kabul edilmiştir mesajını veriyor.

SUCCESSFULL CODES

200

İstek başarılı şekilde gerçekleşir.

201

Yeni kaynaklar oluşturulma işlemi tamamlanmıştır.

204

İstek başarılı oldu ama yanıt gelmedi.

DIRECTIONAL CODES

301

Kaynak adresi kalıcı olarak değişmiştir diye yönlendirir. Ziyaretçi, otomatik olarak bu adrese yönlendirilir.

302

Kaynak adresi geçici olarak değişkenlik göstermiştir. 301 kodundan farkı şudur; sayfa geçici olarak bakıma alınmış olabilir veya test aşamasında olabileceği durumlarda bu kodun kullanımı çok uygundur.

304

Kaynak, istemcinin ön belleğinde var ama güncel değil ve gönderilemiyor.

CLIENT ERROR CODES

400

Sunucu isteği anlayamadı ve işleyemedi hatası verir.

401

İstek için kimlik doğrulaması gerekiyor diye hata belirtir.

403

İstemcinin kaynağa erişim izni yok.

404

İstek yapılan kaynak bulunamadı. Sayfa veya kaynak silinmiş veya url'si değişmiş olabilir.

410

Kaynak yoktur hatası verir ama 404'ten farkı ise kaynağın kesin ve kalıcı olarak silindiğini belirtir.

SERVER ERROR CODES

500

Sunucu , genel bir hata nedeniyle isteği işleyemedi hatası verir.

502

Sunucu , isteği işlerken geçersiz bir yanıt alıyor ve herhangi başka bir sunucuya ihtiyaç duyuyor.

503

Sunucu şu anda işleği işleyemiyor.Genelde aşırı yük ve istek sonucunda bu hata ortaya çıkıyor.