Retrofit İçerisinde Kullanılan HTTP Method Çeşitleri

Retrofit, Android uygulamalarında RESTful API'larla iletişim kurmayı kolaylaştıran bir kütüphanedir. Bu kütüphane, HTTP isteklerini kolayca yapabilmek için çeşitli HTTP yöntemlerini destekler. Retrofit içinde kullanılan bazı HTTP yöntemleri ve kullanım alanları aşağıda belirtilmiştir.

1. **GET**
    - **Nedir?**

    GET, sunucuya istek göndermek için kullanılan HTTP metodlarından biridir. Temel olarak, sunucudan veri almak için kullanılır ve güvenli(safe), sisteme etkisiz(idempotent) ve önbelleğe alınabilir(cachable) olarak tasarlanmıştır. GET metodu belirtilen kaynağın temsilini talep eder. Basitçe ifade etmek gerekirse, GET, sunucuya "Bu URL'deki bilgiyi bana gönderebilir misiniz?" diye sormak gibidir.

    - **Nerede kullanılır?**

    1. Bir haber sitesinden en son başlıkları almak.
    2. Bir e-ticaret uygulamasında ürünlerin listesini almak.
    3. Bir hava durumu servisinden belirli bir konumun hava durumu bilgilerini almak.
    4. Bir blog sitesinden en son yazıları veya kategorilere göre yazıları almak.

2. **POST**
    - **Nedir?**

    POST, bir kaynağı oluşturmak veya güncellemek için sunucuya veri göndermek için kritik bir HTTP yöntemidir. GET'ten farklı olarak, POST, belirli bir kaynağa işlenmek üzere veri göndermek için kullanılır. Genellikle bir dosya yüklenirken veya tamamlanmış bir web formu gönderilirken kullanılır.

    - **Nerede kullanılır?**

    1. Bir sosyal medya uygulamasında yeni bir gönderi oluşturmak.
    2. Bir kullanıcının oturum açma bilgilerini sunucuya göndererek giriş yapmak.
    3. Bir e-ticaret uygulamasında yeni bir sipariş oluşturmak.
    4. Bir forum sitesinde yeni bir konu başlatmak veya yeni bir yorum göndermek.

3. **PUT**
    - **Nedir?**

    PUT, genellikle RESTful API'lerde kullanılan bir HTTP yöntemidir. Geleneksel olarak, PUT mevcut bir kaynağı yeni verilerle güncellemek veya eğer mevcut değilse bir kaynak oluşturmak için kullanılır. Eğer URI bir kaynağa işaret ediyorsa, o kaynak değiştirilir; eğer URL mevcut bir kaynağa işaret etmiyorsa, sunucu o URL ile yeni bir kaynak oluşturabilir.

- **Nerede kullanılır?**

1. Bir kullanıcının profil bilgilerini güncellemek.
2. Bir e-ticaret uygulamasında bir siparişin durumunu güncellemek (örneğin, "Hazırlanıyor"dan "Kargolandı"ya).
3. Bir dosya paylaşım uygulamasında dosyanın adını veya içeriğini güncellemek.

4. **DELETE**
   - **Nedir?**

DELETE, sunucudaki kaynakların silinmesini kolaylaştıran HTTP yöntemlerinden biridir. World Wide Web (WWW) tarafından desteklenen bir istek yöntemidir. Tasarımı gereği, DELETE yöntemi idempotenttir. Yani, aynı DELETE isteğini birden fazla kez gönderirseniz, ilk istek kaynağı siler ve sonraki istekler kaynağın zaten silindiğini belirten 404 Not Found hatası döndürür. Bu, bir dosyayı çöp kutusuna atmaya benzer; bir kere gitti mi, gitmiştir.

- **Nerede kullanılır?**

1. Bir kullanıcının hesabını silmek.
2. Bir e-ticaret uygulamasında bir siparişi iptal etmek.
3. Bir dosya paylaşım uygulamasında bir dosyayı silmek.
4. Bir blog sitesinde bir yazıyı veya yorumu silmek.

Retrofit İçerisinde En Çok Kullanılan Annotations'lar

1. **@GET**
   - @GET annotation, GET HTTP yöntemini belirtir. Bu, belirtilen URL'ye bir GET isteği gönderileceğini belirtir.
   - Bu örnekte, getUser fonksiyonu belirli bir kullanıcıyı almak için kullanılır. @Path annotation ile belirtilen id, belirli bir kullanıcının kimliğini temsil eder.

```
import retrofit2.http.GET
import retrofit2.http.Path

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

2. **@POST**
    - @POST annotation, POST HTTP yöntemini belirtir. Bu, belirtilen URL'ye bir POST isteği gönderileceğini belirtir.
    - Bu örnekte, createUser fonksiyonu yeni bir kullanıcı oluşturmak için kullanılır. Kullanıcının bilgileri, @Body annotation ile belirtilen user nesnesinde bulunur.

```
import retrofit2.Call
import retrofit2.http.Body
import retrofit2.http.POST

interface ApiService {
    @POST("/users")
    fun createUser(@Body user: User): Call<User>
}
```

3. **@PUT**
    - @PUT annotation, PUT HTTP yöntemini belirtir. Bu, belirtilen URL'ye bir PUT isteği gönderileceğini belirtir.
    - Bu örnekte, updateUser fonksiyonu belirli bir kullanıcıyı güncellemek için kullanılır. @Path annotation ile belirtilen id, güncellenecek kullanıcının kimliğini belirtir. Kullanıcının güncellenmiş bilgileri, @Body annotation ile belirtilen user nesnesinde bulunur.

```
import retrofit2.Call
import retrofit2.http.Body
import retrofit2.http.PUT
import retrofit2.http.Path

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

4. **@DELETE**
    - @DELETE annotation, DELETE HTTP yöntemini belirtir. Bu, belirtilen URL'ye bir DELETE isteği gönderileceğini belirtir.
    - Bu örnekte, deleteUser fonksiyonu belirli bir kullanıcıyı silmek için kullanılır. @Path annotation ile belirtilen id, silinecek kullanıcının kimliğini belirtir.

```
import retrofit2.Call
import retrofit2.http.DELETE
import retrofit2.http.Path

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

5. **@Path**

- @Path annotation, URL'nin değişken bölümlerini belirtmek için kullanılır. Bu, belirli bir değeri URL'de değişken olarak belirtmek için kullanılır.
- Örneğin, /users/{id} gibi bir URL'de {id} kısmı değişken bir parametreyi temsil eder.

```
import retrofit2.http.GET
import retrofit2.http.Path

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

6. **@Query**

- @Query annotation, URL sorgu parametrelerini belirtmek için kullanılır. Bu, isteğe ekstra bilgi eklemek için kullanılır.
- Örneğin, /users?id=123 gibi bir istek göndermek için kullanılabilir.

```
import retrofit2.http.GET
import retrofit2.http.Query

interface UserService {
    @GET("/users")
    fun getUsers(@Query("page") page: Int, @Query("limit") limit: Int): Call<List<User>
}
```

7. **@Body**

- @Body annotation, isteğin gövdesini belirtmek için kullanılır. Bu, istek gönderirken veri taşımak için kullanılır. Genellikle JSON veya XML formatında verileri göndermek için kullanılır.

```
import retrofit2.http.POST
import retrofit2.http.Body

interface UserService {
    @POST("/users")
    fun createUser(@Body user: User): Call<User>
}
```

8. **@Header**

- @Header annotation, HTTP başlıklarını belirtmek için kullanılır. Bu, isteğin başlıklarına özel bilgi eklemek için kullanılır.
- Örneğin, yetkilendirme için bir JWT tokenini başlık olarak eklemek isteyebilirsiniz.

```
import retrofit2.http.GET
import retrofit2.http.Header

interface UserService {
    @GET("/user")
    fun getUser(@Header("Authorization") authToken: String): Call<User>
}
```

En Çok Kullanılan HTTP Status Kodlar

- **200 Status Code – OK →** HTTP Durum Kodu 200, istemci tarafından yapılan isteğin başarılı olduğunu ve sunucunun beklenen yanıtı verebildiğini belirtir.
- **201 Status Code – Created →** HTTP Durum Kodu 201, istemcinin isteğine yanıt olarak başarıyla yeni bir kaynağın oluşturulduğunu belirtir.
- **204 Status Code - No Content →** HTTP Durum Kodu 204, başarılı bir isteği işaret eder, ancak 200 OK durumunun aksine, yanıt gövdesinde gönderilecek ek bilgi olmadığını belirtir.
- **301 Status Code – Moved Permanently →** HTTP Durum Kodu 301, bir yeniden yönlendirme yanıtı durum kodudur. İstemcinin erişmeye çalıştığı kaynağın kalıcı olarak farklı bir URL'ye taşındığını ve istemcinin gelecekteki istekler için bu yeni URl'yi kullanması gerektiğini belirtir.
- **302 Status Code – Found →** HTTP Durum Kodu 302, bir yeniden yönlendirme yanıtı durum kodudur. Bu, istemcinin ulaşmaya çalıştığı kaynağın geçici olarak farklı bir URL'de bulunduğunu gösterir. Bu durum kodu istemcilerin gelecekteki istekler için orijinal URL'yi kullanmaya devam etmesi gerektiğini ima eder.
- **304 Status Code – Not Modified→** HTTP Durum Kodu 304, özel bir yeniden yönlendirme yanıtı durum kodudur. İstemciye kaynağı nerede bulacağını söyleyen diğer yönlendirme kodlarından farklı olarak, istemciye o anda önbellekte bulunan kaynak sürümünün hala en son sürüm olduğunu ve yeniden getirilmesine gerek olmadığını bildirir. Bu özellikle gereksiz veri aktarımını azaltarak web performansını optimize etmek için kullanışlıdır.
- **307 Status Code - Temporary Redirect →** HTTP Durum Kodu 307, bir yeniden yönlendirme yanıtı durum kodudur. İstemcinin isteğini farklı bir konuma göndermesi gerektiğini ancak gelecekteki isteklerinde yine de orijinal URL'yi kullanması gerektiğini belirtir.
- **400 Status Code – Bad Request →** HTTP Durum Kodu 400, bir istemci hatası yanıt kodudur. İstemci tarafı geçersiz girişi veya hatalı biçimlendirilmiş istek sözdizimi nedeniyle sunucunun isteği anlayamadığını veya işleyemediğini gösterir. İstemci bir hata yapmıştır ve sunucu, sorun çözülene kadar isteği işleme koyamaz veya işleyemez.
- **401 Status Code – Unauthorized →** HTTP Durum Kodu 401, bir istemci hata yanıt kodudur. İstemcinin istenen yanıtı alabilmesi için kimliğini doğrulaması gerektiğini belirtir. 401 durum kodu, hedef kaynak için geçerli kimlik doğrulama bilgilerinin bulunmaması nedeniyle isteğin uygulanmadığını belirtir.
- **403 Status Code – Forbidden →** HTTP Durum Kodu 403, bir istemci hata yanıt kodudur. İstemcinin isteğinin sunucu tarafından anlaşıldığını ancak sunucunun buna izin vermeyi reddettiğini gösterir. Kimlik doğrulamayla ilgili 401 Unauthorized durumunun aksine, 403 Forbidden durumu, istemcinin kimliğinin doğrulandığını ancak istenen kaynağa erişim izinlerinin bulunmadığını gösterir.
- **404 Status Code – Not Found →** HTTP Durum Kodu 404, sunucunun istenen kaynağı bulamadığını belirten bir istemci hata yanıt kodudur. İstemcinin sunucuyla iletişim kurabildiğini ancak sunucunun istenen şeyi bulamadığını belirtir. Bu hata kodu, web'de sık sık görülmesi nedeniyle en tanınabilir kodlardan biridir.
- **500 Status Code - Internal Server Error →** HTTP Durum Kodu 500, genel bir hata yanıt kodudur. Sunucunun nasıl başa çıkacağını bilmediği bir durumla karşılaştığını belirtir. Genellikle bu kod, sorunun istemcinin isteğinden değil, sunucudan kaynaklandığını gösterir.

- **502 Status Code – Bad Gateway →** HTTP Durum Kodu 502, bir sunucu hatası yanıt kodudur. İnternetteki bir sunucunun başka bir sunucudan geçersiz yanıt aldığını gösterir. Genellikle proxy sunucularda kullanılan bu hata, sorunun istemcinin isteğinde değil, erişmeye çalıştığı upstream sunucusunda olduğunu gösterir.
- **503 Status Code - Service Unavailable →** HTTP Durum Kodu 503, bir sunucu hatası yanıt kodudur. Sunucunun geçici aşırı yüklenmesi veya sunucunun bakımı nedeniyle şu anda isteği yerine getiremediğini gösterir. Bunun anlamı, bunun bir süre sonra düzelecek geçici bir durum olduğudur.
- **504 Status Code – Gateway Timeout →** HTTP Durum Kodu 504, bir sunucu hatası yanıt kodudur. Bu, ağ geçidi veya proxy görevi gören bir sunucunun, upstream başka bir sunucudan veya isteği tamamlamak için erişmesi gereken başka bir yardımcı hizmetten zamanında yanıt almadığının sinyalini verir. Aslında bu, bir sunucunun diğerinden hızlı yanıt almadığına dair bir bildirimdir.

TAHA GÖKPINAR

KAYNAKÇA

[1] https://www.akto.io/academy/what-is-api

[2] Chat-GPT