Retrofit İçinde Kullanılan HTTP Metodları

Retrofit, Rest API lar ile etkileşim kurmamızı sağlayan bir kütüphanedir. Bunun için tasarlanmıştır ve GET, POST, PUT, DELETE gibi HTTP metodlarını kullanarak API'a istekler gönderebiliriz.

@GET

Get yöntemini sunucudan veri çekmek için kullanılır. En sık kullanılan metodlardan biridir. Bu yöntemde sunucuya bir veri gönderilmez. Sunucuda herhangi bir değişiklik yapmaz. @GET notasyonu yazıldıktan sonra istek atacağımız URL'in base kısmından sonrası yazılır. Servisin döndürdüğü cevap için mutlaka bir data class olmalıdır. JSON formatında alınan veriler, data class nesnesine dönüştürülerek kullanılır.

@GET("news/today")

fun getNews() : Call<News>

Kullanım Alanı: Örnek vermek gerekirse, herhangi bir web sitesini görüntülediğimizde GET yöntemini kullanırız. Herhangi bir platformda bir kullanıcının profilini görüntülerken kullanılır. Bir haber sitesinin veya uygulamasının haber akışını görüntülerken kullanılır.

@POST

Post yöntemi de en çok kullanılan bir diğer HTTP metodudur. Bu metodda sunucuya veri gönderilir. Örneğin bir kullanıcı girişi yapılmak istendiğinde kullanıcı adı ve şifre bilgileri servise gönderilir ve servis bu bilgilere göre bir doğrulama mesajı döndürür. Burada sunucuya gönderilen veriler, servisin yapısına göre string* veya nesne** olarak gönderilebilir.

@POST("auth/login"

*fun userLogin("username", "password"): Call<User>

**fun userLogin(@Body userLogin: UserLogin): Call<User>

Kullanım Alanı: Kullanıcı girişi ve kaydı işlemlerinde kullanılır. Bir gönderiye yorum ekleme işlemlerinde kullanılır. E-ticaret uygulamalarında yeni bir ürün ekleme işleminde sunucuya ürün bilgileri gönderilir. Bu gibi sunucuya veri gönderdiğimi uygulamlarda genellikle POST yöntemi kullanılır

@PUT

Put yöntemiyle ise URI üzerindeki kaynağımızı güncellemek istediğimzde veya yeni bir kaynak oluşturmak istediğimizde kullanabileceğimiz bir yöntemdir. Örneğin bir kullanıcının verilerini bu yöntemle güncelleyebiliriz.

@PUT(“update_user_name/{userId}”)

fun updateUserInfo(@Path(“userId”) userId: String, “newName”) : Call<Response>

Kullanım Alanı: Kullanıcı bilgileri güncelleme işlemlerinde, örneğin kullanıcı ismi, e posta adresi vs gibi. Kaynağımızdaki bir dosyayı güncellerken veya yeni bir dosya eklerken PUT notasyonu kullanılabilir.

@DELETE

Delete ile yaptığımız isteklerde ise silme işlemi gerçekleştiririz. Bu yöntemi kullanırken, veri kaybı yaşamamak için dikkatli olunmalıdır.

@DELETE("users/{userId}")

fun deleteUser(@Path("userId") userId: Int): Call<Response>

Kullanım Alanı: Kullanıcı silme, yorum silme, sepetten ürün silme gibi işlemlerde bu notasyon kullanılabilir

Anotations

@Body notasyonu, servisin bizden göndermemizi istediği verileri tek tek eklemek yerine bir nesne oluşturarak göndermemizi sağlar. Örnek vermek gerekirse bizden giriş işlemi için username ve password parametrelerini isteyen bir servise,

UserInfo(val username:String, val password:String) sınıfından bir nesne oluşturup Body notasyonu ile bunu servise gönderebiliriz.

```
@POST("auth/login")
  fun userLogin( @Body userInfo: UserInfo) : Call<User>
```

@Path notasyonu istek göndereceğimiz zaman değişken URL ihtiyaçları için kullanılır. Örneğin bir sosyal medya uygulamasında kullanıcıların profillerini görmek istiyoruz. Ancak hangi kullanıcıyı görmek istediğimiz bilgisi bizim elimizde. Dolayısıyla bu bilgiyi servise gönderebilmek adına Path notasyonu kullanılır. Aşağıdaki örnekte showProfile fonksiyonu bir userId parametresi alarak çalışır ve bu parametre @Path notasyonu sayesinde URL'i tamamlar.

```
@GET(“users/{userId}”)
  fun showProfile(@Path(“userId”) userId): Call<User>
```

@Header notasyonu ile isteğe özel HTTP başlığı eklenebilir. Özellikle kimlik doğrulama bilgilerini HTTP başlıklarında taşımak güvenli bir yöntemdir. URL veya gövdede açık bir şekilde taşınmaz ve gönderilen isteği izleyen kişilerin bu bilgilere erişimi zorlaşır.

```
@GET("user/profile")
  fun getUserProfile(@Header("Authorization") token: String): Call<UserProfile>
```

@Query notasyonu ile istek atarken URL'e sorgu parametreleri eklenir. Örneğin, bir şehrin hava durumunu sorgulamak istiyoruz. Query yöntemi ile hangi şehrin hava durumunu sorgulayacağımızı URL'e gönderebiliriz.

```
@GET(“weather”)
 fun getWeather(@Query(“city”) “istanbul”) : Call<Weather>
```

Bu örnekte URL > “weather?city=istanbul” şeklinde olacaktır.

HTTP Kodları

200 OK: Sunucuya atılan istek başarılı bir şekilde gönderildi ve yanıtlandığını gösterir

201 Created: İsteğin başarılı bir şekilde gittiğini ve sunucuda yeni bir kaynak oluştuğunu bildirir.

400 Bad Request: Bu kodda isteğin sunucu tarafından yorumlanamadığını ve kullanıcının gönderdiği verinin hatalı olduğunu anlarız

401 Unauthorized: Kimlik doğrulamasının gerekli olduğu bir istek atıldığını ancak kimliğin doğrulanamadığını gösterir

403 Forbidden: İstek, sunucu tarafından engellendiğinii kullanıcının erişim iznine sahip olmadığını gösterir.

404 Not Found: İstek atılan URL'in sunucu tarafından bulunamadığını belirtir.

500 Internal Server Error: Sunucu bir iç hata ile karşılaştı ve isteği yerine getiremedi. Genellikle sunucu tarafında bir hata olduğunu gösterir.