HTTP Methodları

→ *@GET*

GET metodu, HTTP protokolünde bir istek göndermek için kullanılan bir yöntemdir. Temel olarak, sunucudan belirli bir kaynağın ya da verinin alınmasını sağlar. GET isteği, sunucuya bir kaynak için bir URI (Uniform Resource Identifier) iletilir ve sunucu bu URI'ye karşılık gelen kaynağı yanıt olarak gönderir.

```
@GET("endpoint") // Endpoint'in URL'sini belirtiyoruz
suspend fun getData(): DataResponse  // GET isteği için fonksiyon
tanımlıyoruz, ResponseModel yanıtı temsil eder
```

→ *@POST*

POST metodu, HTTP protokolünde bir istek göndermek için kullanılan bir yöntemdir. GET metodundan farklı olarak, POST isteği sunucuya yeni veri göndermek için kullanılır. POST isteği genellikle sunucuya veri eklemek, güncellemek veya silmek gibi değişiklikler yapmak için kullanılır.

```
@POST("endpoint") // Endpoint'in URL'sini belirtiyoruz
suspend fun postData(@Body requestModel: RequestModel): DataResponse  //
POST isteği için fonksiyon tanımlıyoruz, RequestModel isteğin gövdesini
temsil eder
```

→ *@PUT*

PUT metodu, HTTP protokolünde bir istek göndermek için kullanılan bir yöntemdir ve genellikle bir kaynağın güncellenmesi için kullanılır. PUT isteği, belirli bir URI'ye sahip kaynağın tamamını ya da bir kısmını güncellemek için kullanılır.

```
@PUT("endpoint/{id}") // Endpoint'in URL'sini ve güncellenecek kaydın
ID'sini belirtiyoruz
suspend fun putData(@Path("id") id: String, @Body requestModel:
RequestModel):  DataResponse // PUT isteği için fonksiyon tanımlıyoruz,
RequestModel isteğin gövdesini temsil eder
```

→ *@DELETE*

DELETE metodu, HTTP protokolünde bir istek göndermek için kullanılan bir yöntemdir. Bu metod, belirtilen URI'deki kaynağı silmek için kullanılır. DELETE isteği, bir kaynağın kalıcı olarak kaldırılmasını sağlar.

```
 @DELETE("endpoint/{id}") // Endpoint'in URL'sini ve silinecek kaydın
ID'sini belirtiyoruz
 suspend fun deleteData(@Path("id") id: String): BaseResponse // DELETE
isteği için fonksiyon tanımlıyoruz
```

Annotations

→ @Body

Body annotation, HTTP isteğinin gövdesini belirtmek için kullanılır. Genellikle POST veya PUT isteklerinde veri göndermek için kullanılır. Bu annotasyon sayesinde, Retrofit otomatik olarak isteği uygun formata dönüştürür ve isteği sunucuya iletir.

```
interface ApiService {
    @POST("users/new")
    suspend fun createUser(@Body user: User): UserResponse
}
```

Burada, createUse rfonksiyonu, **`@POST`** annotasyonuyla işaretlenmiş ve isteğin gövdesinde **`User`** sınıfının bir örneğini alır. Bu örnek, isteğin gövdesinde JSON formatında bir kullanıcı nesnesi göndermek için kullanılabilir.

→ @Path

Path annotation, URL'deki değişken parçaları için değer sağlamak için kullanılır. Bu, dinamik olarak oluşturulmuş URL'lerle çalışırken oldukça yararlıdır. Örneğin, kullanıcı kimlik numarasını içeren bir URL'yi dinamik olarak oluşturabilirsiniz.

```
interface ApiService {
    @GET("users/{id}")
    suspend fun getUserById(@Path("id") userId: Int): UserResponse
}
```

Burada, getUserById fonksiyonu, **`@GET`** annotasyonuyla işaretlenmiş ve **`id`** isimli bir yol parametresini alır. Bu değer, **`userId`** parametresi aracılığıyla sağlanır ve URL'deki **`{id}`** yer tutucusuna yerleştirilir.

→ @Header

Header annotation, HTTP isteğine özel başlık eklemek için kullanılır. Örneğin, kimlik doğrulama token'ı gibi özel bir başlık gerektiren bir isteği göndermek için kullanılabilir.

```
interface ApiService {
    @GET("user/profile")
    suspend fun getUserProfile(@Header("Authorization") token: String):
ProfileResponse
}
```

Burada, getUserProfile fonksiyonu, **`@GET`** annotasyonuyla işaretlenmiş ve **`Authorization`** başlığını alır. Bu başlık, isteği gönderirken kullanılan kimlik doğrulama token'ını içerir.

→ **@Query**

Query annotation, Retrofit'te GET isteklerinde query parametrelerini belirtmek için kullanılır. Query parametreleri, isteğin URL'sine eklenen ve sunucuya istek yaparken belirli bilgileri iletmek için kullanılan parametrelerdir. Örneğin, bir arama sorgusu yaparken veya filtreleme kriterlerini belirlerken query parametreleri kullanılabilir.

```
interface ApiService {
    // GET isteği için bir method tanımlayın ve query parametrelerini alın
    @GET("endpoint")
    suspend fun getData(
        @Query("param1") param1: String,
        @Query("param2") param2: Int
    ): DataResponse // API'nin yanıtını belirten sınıfı dönün
}
```

Örnekte, **`@GET`** annotasyonu ile belirtilen **`getData`** fonksiyonu, **`param1`** ve **`param2`** adında iki query parametresini alır:

HTTP Kodları

1. **200 OK**: İstek başarılı bir şekilde gerçekleştirildi ve sunucu, istenen kaynağı başarılı bir şekilde döndürdü.
2. **400 Bad Request**: Sunucu, isteği anlamadı veya işleyemedi. Genellikle yanlış biçimlendirilmiş veya eksik parametrelerle yapılan isteklerde ortaya çıkar.
3. **401 Unauthorized**: İstemci, kimlik doğrulama gerektiren bir istekte bulundu, ancak yetkilendirme başarısız veya hiç gerçekleşmedi. Bu durum, istemcinin erişim izni olmadığı anlamına gelir.
4. **403 Forbidden**: İstek, sunucu tarafından reddedildi. Bu genellikle istemcinin erişim izni olmadığı veya isteği gerçekleştirmek için gerekli izinlere sahip olmadığı durumlarda oluşur.
5. **404 Not Found**: İstek yapılan kaynak bulunamadı. Sunucu, istemcinin istediği kaynağı bulamadığını belirtmek için bu durum kodunu döndürür.
6. **500 Internal Server Error**: Sunucuda bir hata oluştu ve istek işlenemedi. Genellikle sunucu tarafında bir uygulama hatası olduğunu belirtir.