Kotlin Retrofit'te Kullanılan HTTP Metotları: GET, POST, PUT ve DELETE

Öncelikle Retrofit nedir? Sorusundan başlayalım.Retrofit, Android'de REST API'lerle etkileşim kurmayı kolaylaştıran popüler bir HTTP istemci kütüphanesidir. Bu kütüphane, sunucuya istek göndermek ve verileri almak için çeşitli HTTP metotları kullanır. En yaygın kullanılan metotlar şunlardır:

GET:

- **Tanımı:** Bu metod, belirtilen URI'deki kaynağı almak için kullanılır. Sunucudan veri istemek için kullanılır. Genellikle veri almak için kullanılır ve sunucu üzerinde herhangi bir değişiklik yapmaz.
- **Kullanım Alanları:**
  - Bir kullanıcının profilini veya bir makalenin içeriğini gibi statik verileri almak.
  - Bir listeleme işlemi gerçekleştirmek (örneğin, tüm kullanıcıları veya tüm makaleleri listelemek).

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

POST:

- **Tanımı:** Bu metod, belirtilen URI'ye yeni bir kaynak eklemek için kullanılır. Genellikle form verileri veya JSON verileri gibi gönderilen verileri sunucuya iletir.
- **Kullanım Alanları:**
  - Yeni bir kullanıcı kaydetmek veya bir makale oluşturmak gibi yeni bir veri oluşturmak.
  - Bir formdan veya dosyadan veri göndermek.

```
@POST("/users/new")
fun createUser(@Body newUser: User): Call<User>
```

PUT:

- **Tanımı:** Bu metod, belirtilen URI'deki mevcut kaynağı güncellemek için kullanılır. Genellikle bir kaynağı tamamen değiştirmek için kullanılır.
- **Kullanım Alanları:**
  - Bir kullanıcının profilini veya bir makalenin içeriğini güncellemek gibi mevcut bir veriyi değiştirmek.

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

DELETE:

- **Tanımı:** Bu metod, belirtilen URI'deki kaynağı silmek için kullanılır. Genellikle bir kaynağı kaldırmak için kullanılır.
- **Kullanım Alanları:**
  - Bir kullanıcıyı silmek veya bir makaleyi silmek gibi bir veriyi silmek.

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

Retrofit İçerisinde En Çok Kullanılan Annotations'lar:

Yukarıda GET,POST,PUT,DELETE kullanımından bahsederken onların annotationslarını da kullnmış olduk.Şimdi de sık kullanılan diğer annotationslar hakkında bilgiler verelim.

- **@Headers**: Bu anotasyon, HTTP isteğine özel başlıklar eklemek için kullanılır. Örneğin, belirli bir önbellekleme politikası tanımlamak için veya kimlik doğrulama bilgilerini geçmek için kullanılabilir.

```
@Headers("Cache-Control: max-age=3600")
@GET("/users/{id}")
fun getUser(@Path("id") userId: String): Call<User>
```

Yukarıdaki kodda, `@Headers` anotasyonu, GET isteği için geçerli olan özel bir HTTP başlığı olan "Cache-Control" başlığını belirtir. Bu, sunucudan alınan yanıtların önbelleğe alınacağını belirtir.

- **@Query**: Bu anotasyon, HTTP GET isteği sırasında query parametrelerini belirtmek için kullanılır. Query parametreleri, URL içinde "?key=value" şeklinde belirtilir.

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

Yukarıdaki kodda, `@Query` anotasyonu, "page" adında bir query parametresi ekler. Bu, `/users` endpoint'ine gönderilen GET isteğinde "page" parametresini belirler.

- **@Path**: Bu anotasyon, URL içindeki değişken parçalarını belirtmek için kullanılır. Dinamik URL oluşturmak için kullanılır ve genellikle RESTful servislerle çalışırken path parametrelerini temsil etmek için kullanılır.

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

Yukarıdaki kodda, `@Path` anotasyonu, belirli bir kullanıcının kimliğini içeren URL içindeki değişken parçayı belirtir. Bu sayede, `/users/{id}` endpoint'ine gönderilen GET isteğinde dinamik olarak kullanıcı kimliği belirtilebilir.

- **@Body**: Bu anotasyon, HTTP isteği gövdesini belirtmek için kullanılır. Genellikle POST, PUT gibi metodlarla kullanılır ve gönderilen verinin biçimini belirler.

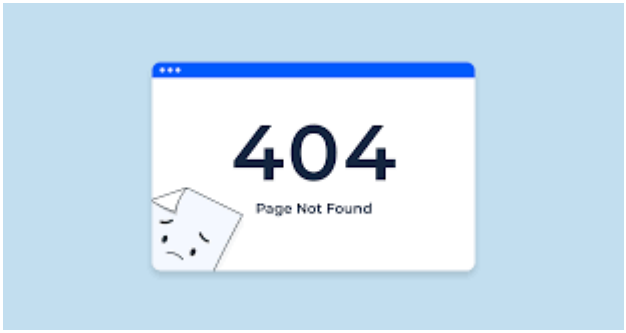
```
@POST("/users/new")
fun createUser(@Body newUser: User): Call<User>
```

Yukarıdaki kodda, `@Body` anotasyonu, POST isteği sırasında gönderilecek olan yeni bir kullanıcı nesnesini belirtir. Bu sayede, `/users/new` endpoint'ine gönderilen POST isteğinde gönderilecek olan veri belirlenir.

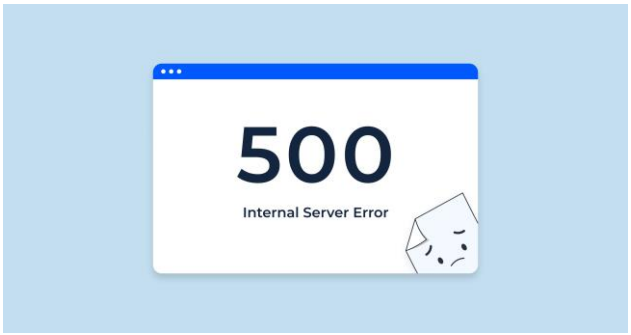
En Sık Kullanılan HTTP Durum Kodları Nelerdir?

HTTP durum kodları, bir istemcinin bir HTTP isteği gönderdiği sunucudan aldığı yanıtları belirtmek için kullanılır. İşte en yaygın olarak kullanılan HTTP durum kodlarının bazıları ve anlamları:

200 OK: İstek başarılı bir şekilde gerçekleştirildi ve istenen bilgiler sunucu tarafından başarıyla döndürüldü.



404 Not Found: İstek yapılan kaynak sunucuda bulunamadı. Genellikle URL hataları veya geçersiz bağlantılar için kullanılır.



500 Internal Server Error: Sunucu, isteği işlerken bir hata oluştuğunu belirtir. Bu genellikle sunucu tarafındaki bir uygulama hatasını gösterir.

401 Unauthorized: Kullanıcı kimlik doğrulaması gerektiren bir kaynağa erişmeye çalıştığında sunucu tarafından gönderilir. Kullanıcı kimlik doğrulaması başarısız olduğunda bu durum kodu alınır.

403 Forbidden: Sunucu, istemcinin istenen kaynağa erişimini reddettiğini belirtir. Genellikle yetki eksikliği durumunda alınır.

302 Found (Redirect): İstek yapılan kaynak başka bir konuma taşındı. Tarayıcı genellikle bu konuma otomatik olarak yönlendirilir.

304 Not Modified: Tarayıcı, sunucudan kaynağın son güncellenme tarihinden sonra herhangi bir değişiklik olmadığını belirten bu kodu alır. Bu durumda, tarayıcı önbellekten veriyi alabilir.

Bu durum kodları, HTTP protokolü aracılığıyla sunucu ve istemci arasındaki iletişimi standartlaştırarak, birçok web hizmeti ve uygulama tarafından kullanılır. Bu kodlar, istemcilerin ve sunucuların birbirleriyle iletişim kurmasını ve hataları anlamasını sağlar, böylece kullanıcılar daha iyi bir deneyim yaşayabilirler.