**Spotify Servisi**

SpotifyService sınıfı, Music Tracker uygulamsında Spotify API ile olan doğrudan iletişimi sağlayan servis katmanıdır. Bu servis kullanıcıların Spotify erişim bilgilerini güncellemek ve belirli bir şarkının Spotify üzerindeki verilerine ulaşmak amacıyla kullanılır. ISpotifyService ara yüzü üzerinden tanımlanan servis, uygulamadaki diğer bileşenlerin Spotify ile veri alışverişi yapabilmesini sağlar.

Servis iki temel işlevi yerine getirir: Kullanıcının daha önce alınmış ancak süresi dolmuş olan erişim jetonunu Refresh Token kullanarak yenilemek ve Last.fm üzerinden elde edilen sanatçı ve şarkı adı bilgilerine göre Spotify’da şarkı aramak.

**Servis Metotları:**

**RefreshSpotifyAccessToken Metodu**

RefreshSpotifyAccessToken metodu, süresi dolmuş bir Spotify erişim token’ini (AccessToken) yenilemek amacıyla kullanılır. Bu işlem, kullanıcıların uygulamaya tekrar giriş yapmadan kullanabilmesi için gereklidir. Metot, Spotify’ın “<https://accounts.spotify.com/api/token>” uç noktasına bir POST isteği göndererek yeni bir erişim jetonu alır.

Kodun Genel Akışı:

1. Yeni bir HTTP istemcisi oluşturulur.
2. Spotify API’ye gönderilecek POST verileri (postData) hazırlanır.
   * grant\_type: “refresh\_token”.
   * refresh\_token: Parametre olarak alınan refresh token değeri.
   * client\_id: Uygulamanın Spotify API istemci kimliği.
   * client\_secret: Uygulamanın Spotify API istemci gizli anahtarı.
3. Bu veriler FormUrlEncodedContent biçiminde kodlanır ve Spotify token uç noktasına POST isteği gönderilir.
4. Dönen yanıt (response) metin olarak okunur.
5. JSON yanıtı SpotifyTokenResponse nesnesine dönüştürülür.
6. Aksi halde işlem başarısız kabul edilerek null döndürülür.

Kod Bloğu:

|  |
| --- |
| public async Task<String?> RefreshSpotifyAccessToken(string refreshToken)  {  var client = \_httpClientFactory.CreateClient();  var postData = new Dictionary<string, string>  {  {"grant\_type", "refresh\_token" },  {"refresh\_token", refreshToken },  {"client\_id", ClientId },  {"client\_secret", ClientSecret }  };  var response = await client.PostAsync("https://accounts.spotify.com/api/token", new FormUrlEncodedContent(postData));  var content = await response.Content.ReadAsStringAsync();  var refreshedToken = JsonSerializer.Deserialize<SpotifyTokenResponse>(content);  if (refreshedToken != null && !string.IsNullOrEmpty(refreshedToken.AccessToken))  {  return refreshedToken.AccessToken;  }  else  {  return null;  }  } |

**SearchSpotifyForTrack Metodu**

SearchSpotifyForTrack metodu, verilen şarkı adı (track) ve sanatçı adı (artist) bilgilerine göre Spotify API üzerinden aram yaparak eşleşen ilk parçayı getirir.

Kodun Genel Akışı:

1. Parametre olarak verilen userId üzerinden bilgileri \_databaseService.GetSpotifyUserAsync metodu ile sorgulanır.
2. Kullanıcının erişim jetonu (AccessToken) boşsa veya kullanıcı bulunamazsa null döndürülür.
3. Jetonun süresi dolmuşsa:
   * RefreshSpotifyAcessToken metodu çağırılarak yeni bir jeton alınmaya çalışılır.
     + Yeni bir jeton alınabilirse kullanıcı verisi güncellenir.
     + Alınamazsa null döndürülür.
4. Yeni bir HttpClient nesnesi oluşturulur ve Authorization başlığına erişim jetonu eklenir.
5. Spotify API’nin /v1/search uç noktasına arama sorgusu hazırlanır.
   * Sorgu parametresi olarak şarkı ve sanatçı bilgileri eklenir.
6. API’ye GET isteği gönderilir.
   * Yanıt başarısızsa hata içeriği loglanır ve null döndürülür.
7. Başarılı yanıt JsonDocument olarak parse edilir.
8. Yanıttaki “tracks” -> “items” alanı kontrol edilir.
   * Eğer sonu listesi boşsa null döndürülür.
9. İlk sonuç alınarak bir SpotifyTrack nesnesine dönüştürülür.
   * id, name, artist, album name, duration alanları doldurulur.
10. Oluşturulan SpotifyTrack nesnesi döndürülür.

Kod Bloğu:

|  |
| --- |
| public async Task<SpotifyTrack> SearchSpotifyForTrack(string artist, string track, string userId)  {  var user = await \_databaseService.GetSpotifyUserAsync(userId);  if (user == null || string.IsNullOrEmpty(user.SpotifyToken.AccessToken))  {  return null;  }  var accessToken = user.SpotifyToken.AccessToken;  if (user.SpotifyToken.IsExpired())  {  var newAccesToken = await RefreshSpotifyAccessToken(user.SpotifyToken.RefreshToken);  if (newAccesToken != null)  {  user.SpotifyToken.AccessToken = newAccesToken;  await \_databaseService.AddOrUpdateUserAsync(user);  accessToken = newAccesToken;  }  else  {  return null;  }  }  var client = \_httpClientFactory.CreateClient();  client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);  var baseUrl = "https://api.spotify.com/v1/search";  var queryUrl = $"?q=track:{Uri.EscapeDataString(track)}%20artist:{Uri.EscapeDataString(artist)}";  var url = baseUrl +  queryUrl +  "&type=track&limit=1";  Console.WriteLine(url);  var response = await client.GetAsync(url);  Console.WriteLine($"Response Status Code: {response.StatusCode}");  if (!response.IsSuccessStatusCode)  {  var errorContent = await response.Content.ReadAsStringAsync();  Console.WriteLine($"Search Error: {errorContent}");  return null;  }  var json = await response.Content.ReadAsStringAsync();  using var doc = JsonDocument.Parse(json);  var items = doc.RootElement.GetProperty("tracks").GetProperty("items");  if (items.GetArrayLength() == 0)  {  return null;  }  var item = items[0];  var trackResult = new SpotifyTrack  {  Id = item.GetProperty("id").GetString(),  Title = item.GetProperty("name").GetString(),  Artist = item.GetProperty("artists")[0].GetProperty("name").GetString(),  AlbumName = item.GetProperty("album").GetProperty("name").GetString(),  Duration = item.GetProperty("duration\_ms").GetInt32()  };  return trackResult;  } |