**SpotifyController Sınıfı**

Bu sınıf, Music Tracker projesinde Spotify API’si ile etkileşime girerek kullanıcıların dinleme geçmişine, profil bilgilerine erişmek ve kullanıcıya özel çalma listesi oluşturmak için geliştirilmiştir. ASP.NET Core mimarisinde ControllerBase sınıfından türetilmiştir ve IDatabaseService, IJwtService, ILastfmService ve ISpotifyService bağımlılıklarıyla çalışmaktadır. SpotifyController kullanıcıya çeşitli uç noktalar sunar: recent, user-info, generate-spotify-playlist, get-listening-history. Bunu yanı sıra bazı uç noktalara yardımcı olmak amacıyla HandleListeningHistory adlı bir yardımı metot içermektedir.

**Recent Uç Noktası**

Bu uç nokta, kullanıcıya ait Spotify hesabındaki en son dinlenmiş parçaları almak için kullanılır. Authorize etiketi ile korunan bu uç noktası, sadece oturum açmış ve geçerli erişim jetonuna (access\_token) sahip kullanıcılar tarafından çağırılabilir. Spotify API’si üzerinden alınan son dinlemeler, sistemdeki bilinen parçalarla eşleştirilir ve gerekirse veri tabanına eklenir. Ayrıca dinleme geçmişi ayrı olarak güncellenir.

Kodun Genel Akışı:

1. Kullanıcının JWT içinden kimliği (userId) alınır.
2. Veri tabanından, kullanıcıya ait Spotify erişim bilgileri getirilir.
3. Erişim jetonunun (access\_token) geçerliliği kontrol edilir; eğer süresi dolmuşsa yenileme jetonu (refresh\_token) ile yeni jeton alınır.
4. Spotify API’si üzerinden /me/player/recently-played uç noktasına istek yapılır.
5. API’den dönen veri ayrıştırılır, her bir parçanın PlayedAT zaman bilgisi ve temel özellikleri (Id, Title, Artist, AlbumName, Duration) alınır.
6. Parçalar veri tabanında mevcut değilse, Last.fm API’si üzerinden alınan bilgilerle zenginleştirilir ve veri tabanına eklenir.
7. Dinleme geçimi kayıt altına alınır ve istemciye TrackWithTimestampDto modeli biçiminde sunulur.

Kod Bloğu:

|  |
| --- |
| [Authorize]  [HttpGet("recent")]  public async Task<IActionResult> GetRecentlyPlayed([FromQuery] int limit = 10)  {  var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;  if (string.IsNullOrEmpty(userId))  return Unauthorized("User ID not found in token.");  var user = await \_databaseService.GetSpotifyUserAsync(userId);  if (user == null || string.IsNullOrEmpty(user.SpotifyToken.AccessToken))  {  return Unauthorized("Spotify access token not found for user.");  }  var accessToken = user.SpotifyToken.AccessToken;    if (user.SpotifyToken.IsExpired())  {  var newAccesToken = await \_spotifyService.RefreshSpotifyAccessToken(user.SpotifyToken.RefreshToken);  if (newAccesToken != null)  {  user.SpotifyToken.AccessToken = newAccesToken;  await \_databaseService.AddOrUpdateUserAsync(user);  accessToken = newAccesToken;  }  else  {  return Unauthorized("Unable to refresh access token");  }  }  var client = \_httpClientFactory.CreateClient();  client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);  var response = await client.GetAsync($"https://api.spotify.com/v1/me/player/recently-played?limit={limit}");  if (!response.IsSuccessStatusCode)  return StatusCode((int)response.StatusCode, "Failed to get recently played tracks");  var json = await response.Content.ReadAsStringAsync();  using var doc = JsonDocument.Parse(json);  var spotifyTracks = new List<SpotifyTrack>();  foreach (var item in doc.RootElement.GetProperty("items").EnumerateArray())  {  var track = item.GetProperty("track");  Console.WriteLine($"Track Played At: {item.GetProperty("played\_at").GetString()}");  var playedAt = item.GetProperty("played\_at").GetDateTime();  long timestamp = new DateTimeOffset(playedAt).ToUnixTimeMilliseconds();  spotifyTracks.Add(new SpotifyTrack  {  Id = track.GetProperty("id").GetString(),  Title = track.GetProperty("name").GetString(),  Artist = track.GetProperty("artists")[0].GetProperty("name").GetString(),  AlbumName = track.GetProperty("album").GetProperty("name").GetString(),  Duration = track.GetProperty("duration\_ms").GetInt32(),  PlayedAt = playedAt, });  }  var trackWithTimestampDtos = new List<TrackWithTimestampDto>();  var trackIdList = spotifyTracks.Select(t => t.Id).ToList();  var knownTracks = await \_databaseService.GetTracksAsync(trackIdList);  var tasks = spotifyTracks.Select(async track =>  {  if (!knownTracks.TryGetValue(track.Id, out var lastfmTrack))  {  lastfmTrack = await \_lastfmService.GetLastfmTrackAsync(track);  if (lastfmTrack != null)  {  await \_databaseService.AddTrackAsync(lastfmTrack);  }  }  if (lastfmTrack != null)  {  await HandleListeningHistory(userId, track.Id, track.PlayedAt);  return new TrackWithTimestampDto  {  Track = lastfmTrack,  Timestamp = new DateTimeOffset(track.PlayedAt).ToUnixTimeMilliseconds()  };  }  return null;  });  var results = await Task.WhenAll(tasks);  return Ok(results.Where(r => r != null));} |

**User-Info Uç Noktası**

Bu uç nokta, kullanıcıya ait güncel profil bilgilerini doğrudan Spotify API’si üzerinden almak amacıyla geliştirilmiştir. Spotify sunucusundan gerçek zamanlı veri çekilir. Kullanıcı e-posta adresi, kullanıcı adı, profil fotoğrafı ve Spotify ID gibi bilgilere erişilir. Authorize etiketi ile korunan bu uç noktası, sadece oturum açmış ve geçerli erişim jetonuna (access\_token) sahip kullanıcılar tarafından çağırılabilir.

Kodun Genel Akışı:

1. Kullanıcının JWT içinden kimliği (userId) alınır.
2. Veri tabanından, kullanıcıya ait Spotify erişim bilgileri getirilir.
3. Erişim jetonunun (access\_token) geçerliliği kontrol edilir; eğer süresi dolmuşsa yenileme jetonu (refresh\_token) ile yeni jeton alınır.
4. Spotify API’sine /me uç noktasına istek gönderilir.
5. Dönen JSON yanıtı ayrıştırılarak kullanıcı bilgileri çıkarılır.
   * id
   * email
   * display\_name
   * images
6. Elde edilen bilgiler bir SpotifyUser nesnesine yerleştirilerek istemciye döndürülür.

Kod Bloğu:

|  |
| --- |
| [Authorize]  [HttpGet("user-info")]  public async Task<IActionResult> GetUserInfo()  {  var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;  if (string.IsNullOrEmpty(userId))  return Unauthorized("User ID not found in token.");  var user = await \_databaseService.GetSpotifyUserAsync(userId);  if (user == null || string.IsNullOrEmpty(user.SpotifyToken.AccessToken))  {  return Unauthorized("Spotify access token not found for user.");  }  var accessToken = user.SpotifyToken.AccessToken;  if (user.SpotifyToken.IsExpired())  {  var newAccesToken = await \_spotifyService.RefreshSpotifyAccessToken(user.SpotifyToken.RefreshToken);  if (newAccesToken != null)  {  user.SpotifyToken.AccessToken = newAccesToken;  await \_databaseService.AddOrUpdateUserAsync(user);  accessToken = newAccesToken;  }  else  {  return Unauthorized("Unable to refresh access token");  }  }  var client = \_httpClientFactory.CreateClient();  client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);  var response = await client.GetAsync("https://api.spotify.com/v1/me");  if (!response.IsSuccessStatusCode)  {  return StatusCode((int)response.StatusCode, "Failed to get user information");  }  var json = await response.Content.ReadAsStringAsync();  using var doc = JsonDocument.Parse(json);  var userInfo = new SpotifyUser();  userInfo.Id = doc.RootElement.GetProperty("id").GetString();  if (doc.RootElement.TryGetProperty("email", out JsonElement emailElement))  {  userInfo.Email = emailElement.GetString();  }  else  {  userInfo.Email = null;  }  if (doc.RootElement.TryGetProperty("display\_name", out JsonElement displayNameElement))  {  userInfo.DisplayName = displayNameElement.GetString();  }  if (doc.RootElement.TryGetProperty("images", out JsonElement imagesElement) &&  imagesElement.ValueKind == JsonValueKind.Array &&  imagesElement.GetArrayLength() > 0)  {  var firstImage = imagesElement[0];  userInfo.ProfileImage = new ProfileImage  {  Url = firstImage.GetProperty("url").GetString(),  Height = firstImage.TryGetProperty("height", out var heightEl) ? heightEl.GetInt32() : 0,  Width = firstImage.TryGetProperty("width", out var widthEl) ? widthEl.GetInt32() : 0,  };  }  return Ok(userInfo);} |

**Generate-Spotify-Playlist Uç Noktası**

Bu uç nokta, kullanıcıya özel Spotify çalma listeleri oluşturmak için geliştirşimiştir. Authorize etiketi ile korunan bu uç noktası, sadece oturum açmış ve geçerli erişim jetonuna (access\_token) sahip kullanıcılar tarafından çağırılabilir. Kullanıcıdan alınan liste adı, açıklama, listeye eklenecek parça kimlikleri ve liste gizliliği gibi bilgiler kullanılarak Spotify API aracılığıyla yeni bir çalma listesi oluşturulur. Bu işlem iki aşamadan oluşur: önce liste oluşturulur, ardından şarkılar listeye eklenir.

Kodun Genel Akışı:

1. PlaylistInfo nesnesi üzerinden; liste adı (PlaylistName), açıklama (Description), gizlilik durumu (IsPublic) ve eklenecek şarkı kimlikleri (TrackIds) alınır.
2. Veri tabanından, kullanıcıya ait Spotify erişim bilgileri getirilir.
3. Erişim jetonunun (access\_token) geçerliliği kontrol edilir; eğer süresi dolmuşsa yenileme jetonu (refresh\_token) ile yeni jeton alınır.
4. Spotify API’ye POST isteği gönderilerek kullanıcı adına yeni bir çalma listesi oluşturulur (/users/{userId}/playlists).
5. Spotify’ın döndürdüğü yanıt üzerinden yeni listenin kimliği (playlistId) alınır.
6. İkinci aşamada ise bir diğer POST isteğiyle, şarkı kimlikleri belirtilen çalma listesine eklenir (/playlists/{playlistİd}/tracks).
7. Başarılı işlem sonucunda kullanıcıya çalma listesi bağlantısı ve kimliği ile birlikte olumlu yanıt döndürülür.

Kod Bloğu:

|  |
| --- |
| [Authorize]  [HttpPost("generate-spotify-playlist")]  public async Task<IActionResult> GenerateSpotifyPlaylist([FromBody] PlaylistInfo playlistInfo)  {  if (playlistInfo == null)  return BadRequest("No tracks provided.");  var playlistName = string.IsNullOrWhiteSpace(playlistInfo.PlaylistName) ? "Generated Playlist" : playlistInfo.PlaylistName;  var playlistDescription = playlistInfo.Description ?? "Created with Music Tracker";  var isPublic = playlistInfo?.IsPublic ?? false;  var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;  if (string.IsNullOrEmpty(userId))  return Unauthorized("User ID not found in token.");  var user = await \_databaseService.GetSpotifyUserAsync(userId);  if (user == null || string.IsNullOrEmpty(user.SpotifyToken.AccessToken))  {  return Unauthorized("Spotify access token not found for user.");  }  var accessToken = user.SpotifyToken.AccessToken;  if (user.SpotifyToken.IsExpired())  {  var newAccesToken = await \_spotifyService.RefreshSpotifyAccessToken(user.SpotifyToken.RefreshToken);  if (newAccesToken != null)  {  user.SpotifyToken.AccessToken = newAccesToken;  await \_databaseService.AddOrUpdateUserAsync(user);  accessToken = newAccesToken;  }  else  {  return Unauthorized("Unable to refresh access token");  }  }  var client = \_httpClientFactory.CreateClient();  client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);  var createPlaylistPayload = new  {  name = playlistName,  description = playlistDescription,  @public = isPublic  };  var playlistUrl = $"https://api.spotify.com/v1/users/{userId}/playlists";  var createPlaylistResponse = await client.PostAsJsonAsync(playlistUrl, createPlaylistPayload);  if (!createPlaylistResponse.IsSuccessStatusCode)  return StatusCode((int)createPlaylistResponse.StatusCode, "Failed to create Spotify playlist.");  var playlistContent = await createPlaylistResponse.Content.ReadAsStringAsync();  var playlistJson = JsonDocument.Parse(playlistContent);  var playlistId = playlistJson.RootElement.GetProperty("id").GetString();  if(playlistId == null)  {  return StatusCode((int)createPlaylistResponse.StatusCode, "Failed to get Spotify playlist ID.");  }  var addItemToPlaylistUrl = $"https://api.spotify.com/v1/playlists/{playlistId}/tracks";  var addItemsToPlaylistPayload = new  {  uris = playlistInfo.TrackIds.Select(id => $"spotify:track:{id}").ToList(),  position = 0  };  var addItemToPlaylistResponse = await client.PostAsJsonAsync(addItemToPlaylistUrl, addItemsToPlaylistPayload);  if (!addItemToPlaylistResponse.IsSuccessStatusCode)  {  return StatusCode((int)createPlaylistResponse.StatusCode, "Failed add tracks to Spotify playlist.");  }  return Ok(new  {  message = "Playlist created successfully!",  playlistId,  playlistUrl = playlistJson.RootElement.GetProperty("external\_urls").GetProperty("spotify").GetString()  });} |

**Get-Listening-History Uç Noktası**

Bu uç nokta, kullanıcının dinleme geçmişini sistem veri tabanından okuyarak sunar, doğrudan Spotify API’sine bir istek göndermez. Bu sayede veri tabanında kaydedilmiş geçmiş veriler hızlıca çekilir ve istemciye iletilir. Bu uç noktası sayfalama destekli kayıtlar döndürür.

Kodun Genel Akışı:

1. JWT çerezi içinden kullanıcı kimliği (userId) elde edilir.
2. IDatabaseService üzerinden GetListeningHistoryTracksAsync metodu çağırılarak, veri tabanından kullanıcının dinleme geçmişi kayıtları (LastfmTrack nesnesi şeklinde) çekilir.
3. İsteğe bağlı olarak limit ve startAfter parametrelerini kullanılır.
4. Başarılı sorgu sonucunda 200 OK ile geçmiş listesi döndürülür.
5. İşlem sırasında bir hata oluşursa, 500 Internal Server Error yanıtı ve genel bir hata mesajı iletilir.

Kod Bloğu:

|  |
| --- |
| [Authorize]  [HttpGet("get-listening-history")]  public async Task<IActionResult> GetListeningHistory([FromQuery] int limit = 10, [FromQuery] long? startAfter = null)  {  var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;  if (string.IsNullOrEmpty(userId))  return Unauthorized("User ID not found in token.");  try  {  var history = await \_databaseService.GetListeningHistoryTracksAsync(userId, limit, startAfter);  Console.WriteLine(history);  return Ok(history);  }catch(Exception ex)  {  Console.WriteLine($"Error getting listening history: {ex.Message}");  return StatusCode(500, "Something went wrong");  }  } |

**HandleListeningHistory Yardımcı Metotu**

Bu yardımcı metot, kullanıcıların Spotify’da dinledikleri parçaları sistemin dinleme geçmişi veri tabanına kaydetmek amacıyla geliştirilmiştir. Bu işlem GetRecentlyPlayed (recent uç noktası) gibi Spotify API’sinden alınan verileri işleyen uç noktalarda çağırılır. Metot aynı dinleme olayının birden fazla kez kaydedilmesini engelleyecek şekilde tasarlanmıştır.

Kodun Genel Akışı:

1. Spotfiy API’den alınan dinlenme zaman bilgisi (playedAt) Unix Timestamp (milisaniye) biçimine dönüştürülür.
2. IDatabaseService üzerinden CheckIfHistoryExists metodu aracılığıyla aynı kullanıcı, parça ve zaman kombinasyonunun daha önce kaydedilip kaydedilmediği kontrol edilir.
3. Eğer sistemde aynı kayıt bulunmuyorsa, IDatabaseService üzerinden AddListeningHistoryAsync metodu ile dinlenme geçmişine yeni bir kayıt eklenir.
4. Bu işlemler bir uç noktasıyla kullanıcıya sunulmaz arka planda gerçekleştirilir.

Kod Bloğu:

|  |
| --- |
| private async Task HandleListeningHistory(string userId, string trackId, DateTime playedAt)  {  long timestamp = new DateTimeOffset(playedAt).ToUnixTimeMilliseconds();  bool alreadyLogged = await \_databaseService.CheckIfHistoryEntryExists(userId, trackId, timestamp);    if (!alreadyLogged)  {  await \_databaseService.AddListeningHistoryAsync(userId, new ListeningHistoryEntry  {  TrackId = trackId,  Timestamp = timestamp,  });  }  return;} |