**Database Servisi**

IDatabaseService arayüzü, Music Tracker uygulamasında veri tabanı işlemlerini soyutlamak ve merkezi bir yapı altında toplamak amacıyla tasarlanmıştır. Bu servis, kullanıcı bilgilerinin, dinlenilen şarkı kayıtlarının ve dinlenme geçmişinin sistemsel olarak işlenmesinden sorumludur. Arayüz, veri tabanına doğrudan erişimi engelleyerek veri işlemlerini tek bir kanal üzerinden yönetir ve katmanlar arası bağımlılığı azaltır.

Servis içerisinde kullanıcı ekleme veya güncelleme, Spotify şarkılarının Last.fm detayları ile birlikte saklanması, dinlenme geçmişinin kayıt altına alınması ve geçmiş verilerin zaman damgasına göre sıralı biçimde çekilmesi gibi işlemleri yürüten çeşitli metotlar yer almaktadır.

**Servis Metotları:**

**AddOrUpdateUserAsync Metodu**

AddOrUpdateUserAsync metodu, sisteme giriş yapan kullanıcının veri tabanında kayıtlı olup olmadığını kontrol eder ve uygun şekilde kullanıcıyı ekler veya günceller. Metot, AddOrUpdateSpotifyUserAsync isimli alt metoda bir sarmalayıcı (wrapper) olarak çalışmakta ve asenkron biçimde kullanıcı işlemini gerçekleştirmektedir.

Kodun Genel Akışı:

1. Kullanıcı Dokümanına Erişim: Firestore veri tabanındaki “users” koleksiyonu üzerinden, kullanıcının Spotify ID’sine karşılık gelen doküman alınır.
2. Mevcut Kullanıcı Kontrolü: DocumentSnapshot üzerinden, kullanıcının daha önce eklenip eklenmediği kontrol edilir.
   * Eğer kullanıcı mevcutsa:
     + Mevcut doküman SpotifyUser nesnesine dönüştürülür.
     + Erişim (access\_token) ve yenileme (refresh\_token) jetonları güncellenir.
     + SetOptions.MergeAll kullanılarak sadece güncellenmesi gereken alanlar veri tabanına iletilir.
   * Eğer kullanıcı mevcut değilse:
     + SpotifyUser nesnesi yeni bir doküman olarak “users” koleksiyonuna eklenir.
3. Sonuç Döndürme: İşlem sonucuna bağlı olarak kullanıcı başarıyla eklendiği ya da güncellendiği bilgisini içeren bir mesaj döndürülür. Herhangi bir hata durumunda, hata mesajı loglanır ve kullanıcıya bir açıklama iletilir.

Kod Bloğu:

|  |
| --- |
| // Wrapper for method AddOrUpdateSpotifyUserAsync  public async Task<string> AddOrUpdateUserAsync(SpotifyUser spotifyUser)  {  return await AddOrUpdateSpotifyUserAsync(spotifyUser);  }  public async Task<string> AddOrUpdateSpotifyUserAsync(SpotifyUser spotifyUser)  {  try  {  var userCollection = \_firestore.Collection("users");  var userDoc = userCollection.Document(spotifyUser.Id);  var userSnapshot = await userDoc.GetSnapshotAsync();  if (userSnapshot.Exists)  {  var existingUser = userSnapshot.ConvertTo<SpotifyUser>();  existingUser.SpotifyToken.AccessToken = spotifyUser.SpotifyToken.AccessToken;  existingUser.SpotifyToken.RefreshToken = spotifyUser.SpotifyToken.RefreshToken;  await userDoc.SetAsync(existingUser, SetOptions.MergeAll);  return "SpotifyUser updated successfully.";  }  else  {  await userDoc.SetAsync(spotifyUser);  return "SpotifyUser added successfully.";  }  }  catch(Exception ex)  {  Console.Error.WriteLine($"Error in AddOrUpdateSpotifyUserAsync: {ex.Message}");  return $"An error occurred: {ex.Message}";  }} |

**GetSpotifyUserAsync Metodu**

GetSpotifyUserAsync metodu, verilen kullanıcı kimliğine (userId) karşılık gelen Spotify kullanıcı bilgilerini Firestore veri tabanından asenkron olarak sorgulayan bir veri erişim fonksiyonudur. Kullanıcının sistemde tanımlı olup olmadığını kontrol etmek, yetkilendirme ve kullanıcıya özel veri işlemlerinde temel teşkil etmektedir.

Kodun Genel Akışı:

1. Kullanıcı Dokümanına Erişim: Firestore veri tabanındaki “users” koleksiyonu üzerinden, kullanıcının Spotify ID’sine karşılık gelen doküman alınır.
2. Mevcutluk Kontrolü: userDoc.Exists kontrolü yapılarak kullanıcının veri tabanında kayıtlı olup olmadığı belirlenir.
   * Kayıt varsa:
     + Doküman, SpotifyUser nesnesine dönüştürülerek döndürülür.
   * Kayıt yoksa:
     + Metot null döndürür.
3. Hata Durumu: Firestore işlemlerinde bir hata oluşursa, hata mesajı stderr çıktısına yazılır ve metodun değeri null gönderilir.

Kod Bloğu:

|  |
| --- |
| public async Task<SpotifyUser?> GetSpotifyUserAsync(string userId)  {  try  {  var userDoc = await \_firestore.Collection("users").Document(userId).GetSnapshotAsync();  return userDoc.Exists ? userDoc.ConvertTo<SpotifyUser>() : null;  }  catch (Exception ex)  {  Console.Error.WriteLine($"Error in GetSpotifyUserAsync: {ex.Message}");  return null;  }  } |

**AddTrackAsync Metodu**

AddTrackAsync metodu, bir şarkının Last.fm verilerine dayanarak sistemin Firestore veri tabanına eklenmesini sağlayan bir veri kayıt fonksiyonudur. Bu metot, dinleme geçmişinden veya Spotify’dan alınan parçaların detaylı Last.fm bilgileriyle birlikte veri tabanına eklenmesinde kullanılır.

Kodun Genel Akışı:

1. Parçalar Dokümanına Erişim: “tracks” adlı koleksiyon üzerinden, parçanın SpotifyId bilgisi doküman kimliği olarak kullanılarak yeni bir doküman referansı (DocumentReference) oluşturulur.
2. Veri Yazma: Parametre olarak LastfmTrack nesnesi, ilgili dokümana SetAsync yöntemiyle yazılır. Bu işlem parçayı veri tabanına yeni olarak ekler veya aynı ID’ye sahip bir kayıt varsa onu günceller.
3. Hata Yönetimi: İşlem sırasında bir hata oluşması durumunda; hata mesajı konsola loglanır, hata üst koda fırlatılır (throw).

Kod Bloğu:

|  |
| --- |
| public async Task AddTrackAsync(LastfmTrack track)  {  try  {  var trackRef = \_firestore.Collection("tracks").Document(track.SpotifyId);  await trackRef.SetAsync(track);  }  catch (Exception ex)  { Console.WriteLine($"Error adding track: {ex.Message}");  throw new Exception("Failed to add track to Firestore.", ex);  } |

**GetTrackAsync Metodu**

GetTrackAsync metodu, verilen bir Spotify parça kimliğine (SpotifyId) karşılık gelen şarkıyı sistemin Firestore veri tabanından sorgulamak amacıyla geliştirilmiştir. Bu işlem, parçaların daha önce sisteme kaydedilip kaydedilmediğini kontrol etmek ve varsa bilgileri direkt geri döndürmek için kullanılmaktadır.

Kodun Genel Akışı:

1. Parçalar Dokümanına Erişim: “tracks” adlı koleksiyon üzerinden, parçanın SpotifyId bilgisi doküman kimliği olarak kullanılarak yeni bir doküman referansı (DocumentReference) oluşturulur.
2. Doküman Sorgulama: GetSnaphsotAsync metodu ile doküman alınır.
   * Doküman mevcutsa: LastfmTrack tipine dönüştürülerek true değeriyle birlikte döndürülür.
   * Doküman mevcut değilse: Metot, (false, null) çiftini döndürür.
3. Hata Yönetimi: İşlem sırasında bir hata oluşması durumunda; hata mesajı konsola loglanır, hata üst koda fırlatılır (throw).

Kod Bloğu:

|  |
| --- |
| public async Task<(bool, LastfmTrack)> GetTrackAsync(string spotifyId)  {  try  {  var trackRef = \_firestore.Collection("tracks").Document(spotifyId);  var snapshot = await trackRef.GetSnapshotAsync();  if (snapshot.Exists)  {  return (true, snapshot.ConvertTo<LastfmTrack>());  }  return (false, null);  }  catch (Exception ex)  {  Console.WriteLine($"Error getting track: {ex.Message}");  throw new Exception("Failed to get track from Firestore.", ex);  }  } |

**UpdateTrackAsync Metodu**

UpdateTrackAsync metodu, veri tabanında daha önceden kayıtlı olan bir şarkının bilgilerini güncellemek amacıyla geliştirilmiştir. Metot, Spotify parça kimliği (SpotifyId) üzerinden tanımlanan şarkı dokümanını bulur ve gelen güncel bilgilerle çakışan alanları merge yöntemiyle günceller. Bu sayede veri bütünlüğü korunur ve mevcut dokümanın tamamı silinip yeniden yazılmadan yalnızca gerekli alanlar değiştirilmiş olur.

Kodun Genel Akışı:

1. Parametre olarak LastfmTrack nesnesinden SpotifyId çıkarılır ve “tracks” koleksiyonundaki ilgili doküman referansı alınır.
2. Belge Güncelleme: SetAsync metodu, SetOptions.MergeAll seçeneği ile çağırılarak yalnızca değişiklik yapılan alanları mevcut dokümanın üzerine yazar.
3. Hata Yönetimi: İşlem sırasında bir hata oluşması durumunda; hata mesajı konsola loglanır, hata üst koda fırlatılır (throw).

Kod Bloğu:

|  |
| --- |
| public async Task UpdateTrackAsync(LastfmTrack updatedTrack)  {  try  {  string spotifyId = updatedTrack.SpotifyId;  var trackRef = \_firestore.Collection("tracks").Document(spotifyId);  await trackRef.SetAsync(updatedTrack, SetOptions.MergeAll); }  catch (Exception ex)  {  Console.WriteLine($"Error updating track: {ex.Message}");  throw new Exception("Failed to update track in Firestore.", ex);  }  } |

**AddListeningHistoryAsync Metodu**

AddListeningHistoryAsync metodu, bir kullanıcının belirli bir zaman aralığında dinlediği parçayı dinleme geçmişi koleksiyonuna eklemek amacıyla kullanılır. Her dinleme kaydı, kullanıcıya özgü “listening\_history” alt koleksiyonuna eklenir.

Kodun Genel Akışı:

1. Firestore veri tabanında “users” koleksiyonu altında userId ile temsil edilen kullanıcıya ait belgeye erişilir. Bu belgenin altında yer alan “listening\_history” alt koleksiyonundan referans alınır.
2. Dinleme geçmişi belgesinin ID’si olarak “{TrackId}\_{Timestamp}” formatı kullanılır. Bu yaklaşım, aynı parçanın aynı zamanda birden fazla kez kaydedilmesini önler.
3. ListeningHistoryEntry nesnesi, oluşturulan belge kimliği ile belirtilen konuma yazılır.
4. Hata Yönetimi: İşlem sırasında bir hata oluşması durumunda; hata mesajı konsola loglanır, hata üst koda fırlatılır (throw).

Kod Bloğu:

|  |
| --- |
| public async Task AddListeningHistoryAsync(string userId, ListeningHistoryEntry entry)  {  try  {  var historyRef = \_firestore  .Collection("users")  .Document(userId)  .Collection("listening\_history");  var docId = $"{entry.TrackId}\_{entry.Timestamp}";  await historyRef.Document(docId).SetAsync(entry);  Console.WriteLine("Added To Listening History");  }  catch (Exception ex)  {  Console.WriteLine($"Error at AddListeningHistory: {ex.Message}");  throw new Exception("Failed to add track to the listening history.", ex);  }  } |

**CheckIfHistoryEntryExists Metodu**

CheckIfHistoryEntryExists metodu, belirli bir kullanıcıya ait dinleme geçmişi koleksiyonunda, belirtilen şarkı kimliği (trackId) ve zaman damgası (timestamp) ile oluşturulmuş bir kaydın daha önce sisteme eklenip eklenmediğini kontrol eder. Bu kontrol tekrarlayan dinleme geçmişi kayıtlarının oluşmasını engellemek için tasarlanmıştır.

Kodun Genel Akışı:

1. Dinleme geçmişi belgesi, “TrackId\_Timestamp” formatında bir kimliğe (document ID) sahiptir. Verilen trackId ve timestamp parametreleri kullanılarak bu kimlik (docId) oluşturulur.
2. “users” koleksiyonu altındaki kullanıcıya ait belgeye ulaşılır. Ardından “listening\_history” alt koleksiyonundaki belirtilen docId’ye karşılık gelen belge referansı alınır.
3. Firestore’dan GetSnapshotAsync metodu ile doküman çekilir. snapshot.Exists ifadesi ile belgenin var olup olmadığı kontrol edilir.
4. Sonuç Döndürme:
   * Kayıt varsa: true döner.
   * Kayıt yoksa: false döner.
5. Hata Yönetimi: İşlem sırasında bir hata oluşması durumunda; hata mesajı konsola loglanır, hata üst koda fırlatılır (throw).

Kod Bloğu:

|  |
| --- |
| public async Task<bool> CheckIfHistoryEntryExists(string userId, string trackId, long timestamp)  {  try  {  var docId = $"{trackId}\_{timestamp}";  var docRef = \_firestore  .Collection("users")  .Document(userId)  .Collection("listening\_history")  .Document(trackId);  var snapshot = await docRef.GetSnapshotAsync();  return snapshot.Exists;  }catch (Exception ex)  {  Console.WriteLine($"Error at CheckIfHistoryEntryExists: {ex.Message}");  throw new Exception("Failed to check if listening history entry exists.", ex);  }  } |

**GetListeningHistoryEntries ve GetlisteningHistoryTracksAsync Metodları**

Bu iki metot, Music Tracker uygulamasında kullanıcıların dinleme geçmişi verilerini sorgulamak amacıyla kullanılır. Her iki metot, kullanıcı ara yüzünde geçmişe ait parçaların sıralı olarak listelenmesi için veri tabanından veri çeker. İkinci metot, birinci metodun sonuçlarını genişleterek her kayıtla ilişkili şarkı bilgisini de içeren daha zengin bir veri yapısı sunar.

GetListeningHistoryEntries Metodu: Kullanıcının “listening\_history” alt koleksiyonundaki dinleme kayıtlarını zaman damgasına göre azalan biçimde sıralı olarak getiri. Parameter olarak limit ve isteğe bağlı startAfter (paginasyon için kullanılır) alır. ListeningHistoryEntry tipinde bir liste döndürür.

GetListeningHistoryTracksAsync Metodu: GetListeningHistoryEntries metodunu çağırarak elde edilen her kayda ait parça bilgilerini GetTrackAsync ile sorgular. Sonuçta her kaydın yanında şarkı detaylarını da içeren TrackWithTimesstampDto listesini döndürür.

Kod Parçaları:

|  |
| --- |
| public async Task<List<ListeningHistoryEntry>> GetListeningHistoryEntries(string userId, int limit, long? startAfter)  {  try  {  var historyRef = \_firestore  .Collection("users")  .Document(userId)  .Collection("listening\_history");  Query query = historyRef  .OrderByDescending("Timestamp")  .Limit(limit);  if (startAfter.HasValue) query = query.StartAfter(startAfter.Value);  var snapshot = await query.GetSnapshotAsync();  return snapshot.Documents  .Select(doc => doc.ConvertTo<ListeningHistoryEntry>())  .ToList();  }catch(Exception ex)  {  Console.WriteLine($"Error at GetListeningHistoryEntries: {ex.Message}");  throw new Exception("Failed to get listening history entries.", ex);  }    } |

|  |
| --- |
| public async Task<List<TrackWithTimestampDto>> GetListeningHistoryTracksAsync(string userId, int limit, long? startAfter)  {  try  {  var entries = await GetListeningHistoryEntries(userId, limit, startAfter);  var result = new List<TrackWithTimestampDto>();  foreach (var entry in entries)  {  var (found, track) = await GetTrackAsync(entry.TrackId);  if (found && track != null)  {  result.Add(new TrackWithTimestampDto  {  Track = track,  Timestamp = entry.Timestamp  });  }  else  {  Console.WriteLine($"Track with ID {entry.TrackId} not found in database.");  }  }  return result;  }  catch (Exception ex)  {  Console.WriteLine($"Error at GetListeningHistoryTracksAsync: {ex.Message}");  throw new Exception("Failed to get full listening history tracks.", ex);  }  } |

**GetTracksAsync Metodu**

GetTracksAsync metodu, birden fazla Spotify parça kimliği (List<SpotifyId>) kullanarak “tracks” koleksiyonundan eşleşen belgeleri alır ve bu belgeleri LastfmTrack modeline dönüştürerek bir sözlük yapısı (dictionary) olarak döndürür.

Kodun Genel Akışı:

1. Parametre olarak gelen spotifyTrackIds listesi, Firestore sorgusunda kullanılmak üzere hazırlanır.
2. “tracks” koleksiyonuna erişilir ve WhereIn filtresi uygulanarak spotifyId alanı, bu listedeki değerlerle eşleşen belgeler sorgulanır.
3. Elde edilen belgeler Firestore’un GetSnapshotAsync metodu ile alınır.
4. Firestore’un Documents.ToDictionary metodu kullanılarak:
   * Anahtar olarak her belgedeki spotifyId alanı alınır.
   * Değer olarak belge LastfmTrack nesnesine dönüştürülür.
5. Oluşturulan Dictionary<string, LastfmTrack> yapısı çağıran metoda döndürülür.

Kod Bloğu:

|  |
| --- |
| public async Task<Dictionary<string, LastfmTrack>> GetTracksAsync(List<string> spotifyTrackIds)  {  var snapshot = await \_firestore.Collection("tracks")  .WhereIn("spotifyId", spotifyTrackIds)  .GetSnapshotAsync();  return snapshot.Documents.ToDictionary(  doc => doc.GetValue<string>("spotifyId"),  doc => doc.ConvertTo<LastfmTrack>());  } |