ANKARA ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM4537 – iOS İLE MOBİL GELİŞTİRME I

“CHORD iOS” PROJE FİNAL RAPORU

Canlı Demo: https://chord.borak.dev

Github Repo: https://github.com/brckfrc/chord_ios

Video: https://youtu.be/59dusNjMBpM

Bora KOCABIYIK

21290270

15/01/2026

İÇİNDEKİLER

1. GİRİŞ

1.1. Proje Özeti

Chord iOS projesi, mevcut bir .NET backend altyapısını temel alarak Flutter ile geliştirilmiş, Discord benzeri modern ve gerçek zamanlı bir mobil sohbet uygulamasıdır. Projenin stratejik hedefi, ağ tabanlı ve gerçek zamanlı iletişim paradigmalarını (WebSocket/SignalR, WebRTC) mobil bir platform üzerinde başarılı bir şekilde uygulamaktır. Bu kapsamda, kullanıcı kimlik doğrulaması, sunucu ve kanal bazlı mesajlaşma, sesli sohbet odaları, arkadaşlık sistemi ve çevrimdışı destek gibi zengin bir özellik seti sunulmuştur. Bu rapor, projenin teknik derinliğini ve işlevsel yeteneklerini detaylandırarak, geliştirme sürecinin bir özetini sunmaktadır.

1.2. Raporun Kapsamı

Bu rapor, Chord iOS istemcisinin geliştirme sürecini ve nihai durumunu kapsamlı bir şekilde belgelemek amacıyla hazırlanmıştır. Raporun ilerleyen bölümlerinde, uygulamanın teknik mimarisi, kullanılan teknoloji yığını, backend servisleriyle olan entegrasyonu, geliştirilen temel özellikler, süreç boyunca karşılaşılan zorluklar ve bu zorluklara getirilen çözümler ele alınacaktır. Son olarak, projenin mevcut durumunun ötesine bakılarak geleceğe yönelik potansiyel geliştirme planları sunulacaktır. Bu yapı, projenin teknik ve işlevsel bütünlüğünü anlamak için bir yol haritası sunmayı hedefler ve bir sonraki bölüm olan mimari tasarıma zemin hazırlar.

2. PROJE MİMARİSİ VE TEKNOLOJİ YIĞINI

2.1. Genel Mimari

Bir mobil uygulamanın başarısı, temelini oluşturan mimari tasarımın sağlamlığına doğrudan bağlıdır. Chord iOS projesinin mimarisi, uygulamanın güvenilirliğini, ölçeklenebilirliğini ve bakım kolaylığını sağlamak üzere dikkatle planlanmıştır. Üst düzeyde mimari, Chord iOS mobil istemcisinin merkezi bir .NET backend API'si ile etkileşimini içerir. Bu etkileşim, dosya depolama işlemleri için **MinIO** ve WebRTC tabanlı sesli iletişim için **LiveKit** gibi üçüncü parti servislerle zenginleştirilmiştir. İstemci, bu servislere doğrudan değil, backend API'si üzerinden kontrollü ve güvenli bir şekilde erişir. Bu istemci-sunucu modeli, iş mantığının merkezileştirilmesini ve mobil uygulamanın daha hafif ve odaklı kalmasını sağlar.

2.2. Mobil Uygulama Mimarisi (Flutter)

Chord iOS projesinin Flutter tabanlı iç mimarisi, sürdürülebilir, test edilebilir ve ölçeklenebilir bir kod tabanı oluşturmak amacıyla katmanlı bir yaklaşımla tasarlanmıştır. Bu mimari, sorumlulukların net bir şekilde ayrılmasını sağlayarak geliştirme sürecini basitleştirir.

- **Data Katmanı:** Uygulamanın dış dünya ile iletişim kuran en alt katmanıdır. dio (HTTP istemcisi) ve signalr_core (WebSocket) gibi paketler aracılığıyla backend API'sine ve SignalR hub'larına yapılan tüm ağ isteklerini yönetir. Verinin ham olarak alınıp gönderildiği noktadır.
- **Repository Katmanı:** Data katmanından gelen ham verileri işleyerek uygulama için anlamlı hale getirir. Gerekli durumlarda verileri yerel önbelleğe (Hive veritabanı) yazar veya buradan okur. Bu katman, uygulamanın geri kalanına temiz ve tutarlı bir veri erişim arayüzü (API) sunarak, veri kaynağının detaylarını soyutlar.
- **State (Durum) Yönetimi Katmanı:** Uygulamanın genel durumunu yönetmek için Riverpod paketini kullanır. Kullanıcı kimlik bilgileri, sunucu ve kanal listeleri, anlık

mesajlar ve kullanıcı durumları gibi reaktif veriler bu katmanda tutulur. UI katmanının, veri değişikliklerine otomatik olarak tepki vermesini ve kendini güncellemesini sağlar.

- **UI (Arayüz) Katmanı:** Kullanıcının doğrudan etkileşimde bulunduğu görsel katmandır. Flutter widget'ları kullanılarak oluşturulmuştur. Sadece kullanıcı arayüzünü oluşturmak ve kullanıcı eylemlerini (buton tıklamaları, form gönderimleri vb.) alarak state yönetimi katmanına iletmekle sorumludur.

2.3. Kullanılan Teknolojiler

Chord iOS projesi, modern ve endüstri standartlarına uygun bir teknoloji yığını üzerine inşa edilmiştir. Aşağıdaki tablo, projede kullanılan temel teknolojileri ve araçları özetlemektedir.

Kategori	Teknoloji	Açıklama
Çekirdek	Flutter 3.38+, Dart 3.10+	Çapraz platform (iOS/Android) UI geliştirme çerçevesi ve programlama dili.
Durum Yönetimi	Riverpod 2.5	Modern, derleme zamanı güvenliğine sahip state management çözümü.
Navigasyon	GoRouter 13.2	Flutter ekibi tarafından önerilen, bildirimsel (declarative) yönlendirme paketi.
Ağ İletişimi	Dio 5.4	Güçlü özelliklere sahip HTTP istemcisi (REST API iletişimi için).
Gerçek Zamanlı	SignalR Core 1.0	.NET backend ile WebSocket üzerinden gerçek zamanlı iletişim kurmak için kullanılır.
Yerel Depolama	Hive 2.2	Hızlı ve hafif bir NoSQL veritabanı (çevrimdışı mod ve önbellekleme için).
Güvenli Depolama	flutter_secure_storage 9.0	JWT gibi hassas verileri iOS Keychain üzerinde güvenli bir şekilde saklamak için kullanılır.
Hata Takibi	Sentry Flutter 8.0	Uygulama çökmelerini ve hatalarını gerçek zamanlı olarak izlemek ve raporlamak için kullanılır.
Dosya Yükleme	image_picker 1.0, video_player 2.8	Galeri ve kameradan resim/video seçimi ve video oynatma için kullanılır.
Yerel Bildirimler	flutter_local_notifications 17.0	Uygulama açıkken (foreground) yerel bildirimler göstermek için kullanılır.
WebRTC	livekit_client 2.3	WebRTC tabanlı sesli iletişim için LiveKit SFU entegrasyonu.
Tercih Depolama	shared_preferences 2.2	Kullanıcı tercihlerini (bildirim ayarları vb.) saklamak için kullanılır.
Ağ İzleme	connectivity_plus 6.0	Ağ bağlantı durumunu izlemek ve çevrimdışı/çevrimiçi modlar arasında geçiş yapmak için kullanılır.

Seçilen bu teknoloji yığını, projenin performans, güvenlik ve sürdürülebilirlik hedeflerine ulaşmasında kritik bir rol oynamıştır. Bu sağlam temel, backend servisleriyle karmaşık entegrasyonların verimli bir şekilde yönetilmesine olanak tanımıştır.

3. BACKEND ENTEGRASYONU

3.1. API Mimarisi ve İletişim

Mobil uygulamanın başarısı, backend servisleriyle ne kadar verimli, güvenli ve stabil iletişim kurduğuna doğrudan bağlıdır. Chord iOS istemcisi, Chord Backend API'si ile iki ana kanal üzerinden haberleşmektedir: durum bilgisi gerektiren ve senkron işlemler için **RESTful API** ve anlık veri akışı gerektiren asenkron işlemler için **SignalR WebSocket** bağlantıları. Bu çift kanallı mimari, her işlem türü için en uygun iletişim yöntemini kullanarak hem ağ verimliliğini hem de kullanıcı deneyimini optimize eder.

3.2. REST API Entegrasyonu

Mobil uygulama; kullanıcı kaydı, girişi, sunucu ve kanal yönetimi, mesaj geçmişini çekme gibi durum bilgisi gerektiren ve anlık olmayan işlemler için backend'in REST API'sini kullanır. Bu entegrasyon, Dio paketi aracılığıyla yönetilmektedir. Dio'nun "interceptor" (araya girici) mekanizması, her API isteğine otomatik olarak JWT accessToken'ı ekleyerek ve token süresi dolduğunda refreshToken kullanarak oturumu yenileyerek kimlik doğrulama sürecini şeffaf ve merkezi bir şekilde yönetir.

Uygulamanın etkileşimde bulunduğu ana endpoint grupları şunlardır:

- **/api/Auth:** Kullanıcı kaydı, girişi, oturum yenileme ve mevcut kullanıcı bilgilerini alma işlemleri.
- **/api/Guilds, /api/Channels:** Sunucu ve kanal oluşturma, listeleme, güncelleme ve silme gibi yönetimsel işlemler. Kanal türleri arasında metin kanalları (text), duyuru kanalları (announcement) ve sesli kanallar (voice) bulunur.
- **/api/Friends, /api/DMs:** Arkadaşlık isteklerini (gönderme, kabul etme, engelleme) ve özel mesaj kanallarını (oluşturma, listeleme) yönetir.
- **/api/Mentions:** Kullanıcının okunmamış mention'larını listeleme, okunmamış mention sayısını alma ve mention'ları okundu olarak işaretleme işlemleri.
- **/api/Upload:** Resim, video ve diğer dosyaları MinIO depolama servisine yükleme.
- **/api/Voice/token:** Bir sesli kanala katılmak için gereken LiveKit erişim token'ını alma.
- **/api/Invites:** Sunucu davet linkleri oluşturma ve yönetme işlemleri.

3.3. Gerçek Zamanlı İletişim: SignalR

SignalR, Chord projesinin gerçek zamanlı iletişim omurgasını oluşturur. Anlık mesajlaşma, kullanıcı durum güncellemeleri (presence), "yazıyor…" bilgisi ve sesli kanal etkileşim bildirimleri gibi düşük gecikme gerektiren özellikler, signalr_core paketi kullanılarak WebSocket üzerinden implemente edilmiştir. Bu yaklaşım, sürekli HTTP yoklamasına (polling) göre çok daha verimli ve performanslıdır.

Uygulamanın bağlandığı iki ana SignalR Hub'ı ve işlevleri şunlardır:

- **ChatHub:** Metin ve özel mesajların gönderilip alınmasını (ReceiveMessage, DMReceiveMessage), mesaj düzenleme/silme bildirimlerini (MessageEdited, MessageDeleted), yazıyor göstergelerini (UserTyping), kullanıcıların sesli kanallara girip çıkması gibi durum güncellemelerini (UserJoinedVoiceChannel, UserLeftVoiceChannel), @mention bildirimlerini (UserMentioned) ve DM okundu bildirimlerini (DMMarkAsRead) yönetir.

- **PresenceHub:** Kullanıcıların çevrimiçi (Online), boşta (Idle), rahatsız etmeyin (DND), görünmez (Invisible) veya çevrimdışı gibi durumlarını (UserOnline, UserOffline, UserStatusChanged) uygulama genelindeki tüm istemcilere anlık olarak yayınlar. Ayrıca arkadaşlık isteği bildirimlerini (FriendRequestReceived, FriendRequestAccepted) yönetir. Bu sayede kullanıcı listeleri, profiller ve arkadaşlık durumları her zaman güncel kalır.

Backend ile kurulan bu sağlam REST ve SignalR iletişimi, uygulamanın sunduğu zengin ve reaktif özellik setinin temelini oluşturarak, bir sonraki bölümde detaylandırılacak olan veri yönetimi ve özellik implementasyonlarına olanak tanır.

4. GELİŞTİRİLEN TEMEL ÖZELLİKLER VE UYGULAMA DETAYLARI

4.1. Stratejik Özelliklerin Analizi

Bu bölümde, projenin kapsamını ve başarısını sergileyen en önemli özelliklerin teknik ve işlevsel detayları analiz edilecektir. Geliştirilen her bir özellik, modern bir sohbet uygulamasından beklenen temel işlevselliği sağlamanın yanı sıra, kullanıcı deneyimini zenginleştiren ve projenin genel akademik hedeflerine katkıda bulunan yenilikçi yaklaşımları da içermektedir.

4.2. Kimlik Doğrulama ve Güvenli Oturum Yönetimi

Uygulama, endüstri standardı olan JWT (JSON Web Token) tabanlı bir kimlik doğrulama akışı kullanır. Kullanıcı başarıyla giriş yaptığında, backend tarafından bir accessToken (kısa ömürlü) ve bir refreshToken (uzun ömürlü) üretilir. Bu token'lar, hassas verilerin güvenli bir şekilde saklanması için flutter_secure_storage paketi kullanılarak iOS cihazının Keychain'inde şifrelenmiş olarak depolanır. Dio HTTP istemcisine entegre edilen bir interceptor, her API isteğinden önce accessToken'ın geçerliliğini kontrol eder. Eğer token'ın süresi dolmuşsa, interceptor arka planda refreshToken'ı kullanarak otomatik olarak yeni bir accessToken talep eder ve isteği bu yeni token ile tekrarlar. Bu mekanizma, kullanıcının oturumunun kesintisiz ve güvenli bir şekilde devam etmesini sağlar.

4.3. Sunucu ve Kanal Yönetimi

Uygulama, Discord benzeri bir sunucu ve kanal yapısı sunar. Kullanıcılar birden fazla sunucuya (guild) üye olabilir ve her sunucu içinde farklı türde kanallar bulunur:

1. **Metin Kanalları (Text Channels):** Kullanıcıların mesajlaşabileceği standart kanallar. Her metin kanalı, sunucu içinde benzersiz bir isme sahiptir.
2. **Duyuru Kanalları (Announcement Channels):** Önemli duyuruların yapıldığı özel kanallar. Bu kanallar, metin kanallarından ayrı bir bölümde (ANNOUNCEMENTS) gösterilir ve özel bir ikon (kampanya ikonu) ile işaretlenir.
3. **Sesli Kanallar (Voice Channels):** WebRTC tabanlı sesli iletişim için kullanılan kanallar. Bu kanallar, VOICE CHANNELS bölümünde gösterilir.

ChannelSidebar widget'ı, kanalları türlerine göre gruplandırır ve her grup için ayrı bir bölüm (section) oluşturur. Kullanıcılar, her bölümün başlığına tıklayarak o bölümü daraltabilir veya genişletebilirler. Daraltılmış bir bölümde, sadece seçili kanal (varsa) gösterilir. Bu özellik, çok sayıda kanalı olan sunucularda navigasyonu kolaylaştırır. Her bölümün başlığında, yeni kanal oluşturmak için bir "+" butonu bulunur.

Sunucu davet sistemi, InviteModal widget'ı ile yönetilir. Kullanıcılar, sunucu ayarlarından veya kanal başlığından davet linki oluşturabilirler. Oluşturulan link, panoya kopyalanabilir ve diğer

kullanıcılarla paylaşılabilir. Davet linkleri, backend'deki /api/Invites endpoint'i üzerinden oluşturulur ve benzersiz bir kod içerir.

Kanal listesi boş olduğunda, kullanıcıya yeni kanal oluşturması için yönlendiren bir empty state mesajı gösterilir. Bu, kullanıcı deneyimini iyileştirir ve uygulamanın kullanımını kolaylaştırır.

ChannelSidebar, kullanıcı bir sunucuya geçtiğinde otomatik olarak o sunucudaki metin kanallarına (maksimum 10 kanal) SignalR üzerinden katılır (auto-join). Bu özellik, kullanıcının her kanalı manuel olarak açmasına gerek kalmadan mesajları gerçek zamanlı olarak almasını sağlar. Auto-join işlemi, SignalR bağlantısı kurulduğunda ve sunucu değiştiğinde otomatik olarak tetiklenir. Kullanıcı bir kanaldan ayrıldığında veya sunucu değiştirdiğinde, önceki kanallardan otomatik olarak ayrılır (auto-leave).

ChannelView'da, sağ tarafta MemberList widget'ı gösterilir. Bu widget, sunucudaki tüm üyeleri durumlarına göre gruplandırarak listeler (Online, Idle, DND, Offline). Her üye için avatar, kullanıcı adı, durum göstergesi ve rol bilgisi gösterilir. MemberList, PresenceHub'dan gelen gerçek zamanlı durum güncellemelerini dinleyerek anlık olarak güncellenir.

4.4. Gerçek Zamanlı Mesajlaşma ve Kullanıcı Etkileşimi

SignalR entegrasyonu, uygulamanın en dinamik özelliklerine güç vermektedir.

- **Anlık Mesajlaşma:** Kullanıcılar bir mesaj gönderdiğinde, bu mesaj ChatHub üzerinden ilgili kanaldaki tüm istemcilere anlık olarak iletilir. Sunucudan gelen ReceiveMessage olayı dinlenerek arayüz güncellenir. Mesaj listesi, infinite scroll (sonsuz kaydırma) özelliği ile sayfalandırılmıştır. Kullanıcı yukarı kaydırdıkça, daha eski mesajlar otomatik olarak yüklenir.

- **Mesaj Gruplandırma:** Aynı kullanıcının ardışık mesajları, Discord benzeri şekilde gruplandırılır. Bu sayede arayüz daha temiz görünür ve mesaj geçmişi daha okunabilir hale gelir. Gruplandırılmış mesajlarda, kullanıcı avatarı ve adı sadece ilk mesajda gösterilir.

- **Mesaj Düzenleme ve Silme:** Kullanıcılar kendi mesajlarını düzenleyebilir ve silebilirler. Bu işlemler SignalR üzerinden anlık olarak tüm istemcilere iletilir (MessageEdited, MessageDeleted olayları) ve arayüz otomatik olarak güncellenir.

- **Yazıyor Göstergesi (Typing Indicator):** Kullanıcılar mesaj yazarken, ilgili kanaldaki diğer kullanıcılara "X kullanıcısı yazıyor…" şeklinde bir gösterge görünür. Bu gösterge, kullanıcı yazmayı bıraktıktan birkaç saniye sonra otomatik olarak kaybolur. UserTyping SignalR olayı ile gerçek zamanlı olarak yönetilir.

- **İyimser Güncellemeler (Optimistic Updates) ve Ghost Messages:** Kullanıcı deneyimini iyileştirmek için, bir mesaj gönderildiğinde sunucu onayı beklenmez. Mesaj, arayüzde anında "ghost message" (hayalet mesaj) olarak, hafif soluk bir renkte ve gönderim durumu belirteciyle gösterilir. Ghost message'lar, MessageDto modelinde isPending flag'i ile işaretlenir ve geçici bir ID ile oluşturulur. Sunucudan onay mesajı (ReceiveMessage olayı) geldiğinde, MessageProvider otomatik olarak aynı içeriğe sahip ghost message'ı bulur ve gerçek mesajla değiştirir. Eğer mesaj gönderimi başarısız olursa, ghost message arayüzden kaldırılır ve kullanıcıya hata mesajı gösterilir. Çevrimdışı durumda gönderilen mesajlar da ghost message olarak gösterilir ve pending messages queue'ya eklenir. Online olduğunda, bu mesajlar otomatik olarak gönderilir ve ghost message'lar gerçek mesajlarla değiştirilir. Bu yaklaşım, ağ gecikmelerini kullanıcıya hissettirmez ve çevrimdışı durumda bile kullanıcıya anında geri bildirim sağlar.

- **Kullanıcı Durumu (Presence):** PresenceHub, kullanıcıların durumlarını (“Online”, “Idle”, “DND”, “Invisible”) uygulama genelinde senkronize eder. Bir kullanıcının durumu değiştiğinde, bu bilgi diğer tüm istemcilere anlık olarak iletilir ve kullanıcı listeleri, avatarlar yanındaki durum göstergeleriyle güncellenir.

- **@Mention Sistemi:** Kullanıcılar, mesajlarında @ karakterini kullanarak diğer kullanıcılardan bahsedebilirler. MessageComposer widget’ında @ karakteri yazıldığında, aktif sunucudaki üyelerin listesini gösteren bir autocomplete dropdown’ı açılır. Kullanıcılar kendilerini mention edemezler (self-mention prevention). Bahsedilen kullanıcının adı mesaj içinde mavi arka plan ile vurgulanır. ChatHub üzerinden ilgili kullanıcıya UserMentioned olayı gönderilir ve uygulama içinde yerel bir bildirim gösterilir. Kullanıcılar, okunmamış mention’larını MentionsPanel widget’ı üzerinden görüntüleyebilir ve bir mention’a tıklayarak ilgili mesaja otomatik olarak yönlendirilirler. Unread mention sayısı kullanıcı avatarının yanında bir badge olarak gösterilir.

4.5. Sesli Kanallar ve WebRTC Entegrasyonu (LiveKit)

Proje, WebRTC tabanlı yüksek kaliteli sesli iletişim altyapısı sunar. Ölçeklenebilirlik ve performans nedenleriyle, her istemcinin diğer tüm istemcilere doğrudan bağlandığı P2P (Peer-to-Peer) mimarisi yerine, medya akışlarını merkezi olarak yöneten LiveKit SFU (Selective Forwarding Unit) mimarisi tercih edilmiştir. Bu sayede çok daha fazla kullanıcı aynı odada stabil bir şekilde iletişim kurabilir.

Akış şu adımlarla gerçekleşir:

1. Kullanıcı bir sesli kanala katılmak istediğinde, mobil uygulama backend’deki /api/Voice/token endpoint’ine bir istek gönderir.

2. Backend, bu kullanıcı ve oda için özel, kısa ömürlü bir LiveKit erişim token’ı üretir ve bunu istemciye yanıt olarak döner.

3. Mobil uygulama, aldığı bu token ile LiveKit sunucusuna güvenli bir WebSocket bağlantısı kurarak sesli odaya katılır.

4. Odaya katıldıktan sonra, susturma (mute/deafen) gibi durum değişiklikleri ve konuşma göstergeleri (speaking indicators) hem LiveKit’in kendi olayları üzerinden hem de tüm istemcilerle senkronizasyon için SignalR ChatHub üzerinden yönetilir.

5. Discord benzeri bir kullanıcı deneyimi sunmak için, her sesli kanalın altında o kanaldaki aktif katılımcıların listesi gösterilir. Bu liste, kullanıcıların sesli kanallara katılıp ayrılmasıyla anlık olarak güncellenir. Aktif sesli kanalda bulunan kullanıcılar için kanal başlığı yeşil renk ile vurgulanır ve konuşma sırasında speaking indicators (yeşil border ve glow efekti) gösterilir.

6. VoiceBar widget’ı, aktif sesli kanal bilgilerini, bağlantı durumunu ve katılımcı sayısını gösterir. Tüm korumalı route’larda (ProtectedRoute) otomatik olarak görünür ve kullanıcıların sesli kanallara hızlı erişimini sağlar. VoiceBar’da bağlantı kalitesi göstergeleri (excellent, good, poor, disconnected) renkli nokta ile gösterilir. Kullanıcı deneyimini zenginleştirmek için haptic feedback özellikleri eklenmiştir: mute/unmute işlemlerinde hafif titreşim, deafen işlemlerinde orta seviye titreşim ve kanaldan ayrılma işlemlerinde güçlü titreşim kullanılır.

7. VoiceBar’a tıklandığında, VoiceRoomView adlı full-screen bir sayfa açılır. Bu sayfa, sesli kanaldaki tüm katılımcıları grid layout’ta gösterir. Her katılımcı için avatar, kullanıcı adı, mute/deafen durumu ve konuşma göstergesi (speaking indicator)

gösterilir. VoiceRoomView, kullanıcıların sesli kanal içindeki tüm katılımcıları bir arada görmesini ve sesli sohbet deneyimini zenginleştirmesini sağlar.

8. Sesli kanal bağlantı stabilitesi için gelişmiş bir retry mekanizması implemente edilmiştir. VoiceProvider, sesli kanala katılma işlemi başarısız olduğunda otomatik olarak yeniden deneme yapar.

9. Retry mekanizması, üssel geri çekilme (exponential backoff) algoritması kullanır: ilk denemede 2 saniye bekler, sonraki denemelerde bekleme süresi katlanarak artar (4, 8, 16, 32 saniye). Maksimum 5 deneme yapılır. Her denemede, ağ bağlantısı kontrol edilir ve bağlantı yoksa retry işlemi ertelenir. Bu yaklaşım, geçici ağ sorunlarında bile kullanıcının sesli kanala başarıyla katılmasını sağlar.

4.6. Arkadaşlık ve Özel Mesajlaşma Sistemi

Uygulama, kullanıcıların sunucular dışında birebir etkileşim kurabilmesi için kapsamlı bir arkadaşlık ve özel mesajlaşma (DM) sistemi içerir. Arkadaşlık isteği gönderme, kabul etme, reddetme ve kullanıcı engelleme gibi tüm sosyal etkileşimler, ilgili REST API endpoint'leri (/api/Friends) üzerinden yönetilir. İki kullanıcı arasında bir DM kanalı oluşturulduktan sonra, bu kanaldaki mesajlaşma trafiği de ChatHub üzerinden tamamen gerçek zamanlı olarak gerçekleşir, bu da özel sohbetlerin en az sunucu kanalları kadar anlık ve reaktif olmasını sağlar.

FriendsSidebar header'ının altında, online ve idle durumundaki arkadaşların horizontal listesi gösterilir. Bu listedeki bir arkadaşa tıklandığında, mevcut DM varsa açılır, yoksa yeni bir DM oluşturulur. Bu özellik, kullanıcıların aktif arkadaşlarıyla hızlı bir şekilde iletişim kurmasını sağlar.

4.7. Çevrimdışı Desteği ve Önbellekleme

Chord iOS, ağ bağlantısı kesildiğinde veya zayıf olduğunda bile kullanıcıya kesintisiz bir deneyim sunmayı hedefler. Bu amaçla, Hive yerel veritabanı kullanılarak güçlü bir önbellekleme ve çevrimdışı destek mekanizması geliştirilmiştir.

- **MessageCacheService:** Mesaj önbellekleme işlemleri, MessageCacheService adlı özel bir servis sınıfı tarafından yönetilir. Bu servis, Hive veritabanını kullanarak mesajları kanal bazında saklar. Her kanal için ayrı bir Hive box'ı kullanılır ve mesajlar JSON formatında serialize edilerek saklanır. MessageCacheService, mesaj ekleme, güncelleme, silme ve listeleme işlemlerini yönetir. Ayrıca, çevrimdışı durumda gönderilecek mesajları (pending messages) da saklar. Pending messages, kanal ID'si ile ilişkilendirilmiş özel bir Hive key'inde tutulur ve online olduğunda otomatik olarak gönderilir.

- **Mesaj Önbellekleme:** Sunuculardan, kanallardan ve özel mesajlaşmalardan daha önce çekilen mesajlar, MessageCacheService aracılığıyla Hive veritabanında yerel olarak önbelleğe alınır. Her mesaj, kanal ID'si ile ilişkilendirilerek saklanır ve tarih sırasına göre sıralanır. Bu sayede kullanıcı, çevrimdışıyken bile geçmiş konuşmalarına erişebilir. Mesajlar, SignalR'dan geldiğinde veya REST API'den çekildiğinde otomatik olarak önbelleğe yazılır. Mesaj düzenlendiğinde veya silindiğinde, önbellek de otomatik olarak güncellenir.

- **Pending Messages Queue:** Kullanıcı çevrimdışıyken bir mesaj göndermeye çalıştığında, bu mesaj reddedilmek yerine MessageCacheService'in pending messages queue'suna eklenir. Pending message, mesaj içeriği, ekler (attachments), reply bilgisi ve timestamp gibi tüm gerekli bilgileri içerir. Cihaz tekrar internete bağlandığında, MessageRepository otomatik olarak pending messages queue'sunu

kontrol eder ve sırayla tüm mesajları sunucuya gönderir. Her mesaj başarıyla gönderildikten sonra, pending messages queue'sundan kaldırılır. Eğer bir mesaj gönderimi başarısız olursa, mesaj queue'da kalır ve bir sonraki online durumunda tekrar gönderilmeye çalışılır.

- **Ağ Durumu İzleme ve Otomatik Senkronizasyon:** Ağ bağlantı durumu, connectivity_plus paketi kullanılarak sürekli izlenir. ConnectivityService, ağ durumu değişikliklerini Stream olarak sağlar ve uygulama bu değişikliklere anında tepki verir. Bağlantı kesildiğinde, uygulama çevrimdışı moda geçer ve yeni mesajlar pending queue'ya eklenir. Online olduğunda, MessageProvider otomatik olarak pending messages queue'sunu senkronize eder ve önbellekteki mesajları backend ile karşılaştırarak günceller. Bu sayede, kullanıcı çevrimdışıyken bile mesaj gönderebilir ve online olduğunda tüm mesajlar otomatik olarak gönderilir.

4.8. Dosya Yükleme ve Video Desteği

Uygulama, kullanıcıların mesajlara resim, video ve belge eklemesine olanak tanır. image_picker paketi kullanılarak, kullanıcılar galeriden veya kameradan dosya seçebilir. Yüklenen dosyalar, backend'deki /api/Upload endpoint'i üzerinden MinIO depolama servisine gönderilir ve 25MB boyut limiti ile sınırlandırılmıştır. UploadRepository, multipart/form-data formatında dosya yükleme işlemlerini yönetir ve Dio'nun onSendProgress callback'i kullanılarak yükleme ilerlemesini (progress) gerçek zamanlı olarak takip eder.

MessageComposer widget'ı, dosya yükleme sürecini kullanıcı dostu bir şekilde yönetir. Kullanıcı dosya seçtiğinde, dosya önizlemesi (thumbnail) hemen gösterilir. Yükleme sırasında, her dosya için ayrı bir progress bar gösterilir ve yükleme yüzdesi gerçek zamanlı olarak güncellenir. Yükleme tamamlandığında, progress bar kaybolur ve dosya hazır duruma gelir. Eğer yükleme başarısız olursa, dosya önizlemesinde hata ikonu gösterilir ve kullanıcı dosyayı kaldırabilir veya tekrar yüklemeyi deneyebilir. Kullanıcı, yükleme sırasında dosyayı iptal edebilir ve yeni dosyalar ekleyebilir. Birden fazla dosya seçildiğinde, dosyalar sırayla yüklenir ve her birinin progress'i ayrı ayrı gösterilir. Bu sayede, kullanıcı yükleme sürecini tam olarak takip edebilir ve gerektiğinde müdahale edebilir.

Video dosyaları için otomatik thumbnail oluşturma ve inline playback özellikleri sunulmuştur. video_player paketi kullanılarak, video mesajları doğrudan sohbet içinde oynatılabilir. Farklı aspect ratio'lara (4:3, 16:9, 1:1) sahip videolar doğru şekilde gösterilir. Resim dosyaları için photo_view paketi kullanılarak full screen görüntüleme ve zoom özellikleri sağlanmıştır. Belge dosyaları (PDF, Word, Excel, TXT, CSV, ZIP, RAR) için dosya türüne göre uygun icon gösterimi ve indirme özelliği mevcuttur.

4.9. Yerel Bildirimler

Uygulama, kullanıcıların uygulama açıkken (foreground) bildirim almasını sağlar. flutter_local_notifications paketi kullanılarak, @mention ve yeni DM mesajları için yerel bildirimler gösterilir. NotificationService, bildirim servisini başlatır ve iOS için gerekli izinleri (alert, badge, sound) yönetir.

Kullanıcılar, bildirim tercihlerini UserSettingsModal üzerinden kanal ve DM için ayrı ayrı yapılandırabilir. UserSettingsModal, kullanıcı profil butonuna (GuildSidebar'daki avatar) tıklandığında açılır. Bu modal, kullanıcı durumu (status) değiştirme ve bildirim tercihleri yapılandırma özelliklerini birleşik bir arayüzde sunar. Kullanıcılar, Online, Idle, DND ve Invisible durumlarını seçebilirler (Offline durumu seçilemez, çünkü Invisible zaten offline gibi görünür). Bu tercihler shared_preferences paketi kullanılarak yerel olarak saklanır. NotificationPreferencesProvider, bu tercihleri Riverpod state yönetimi ile yönetir. Bildirimlere tıklandığında, deep linking mekanizması ile ilgili sayfaya (kanal veya DM) otomatik

yönlendirme yapılır. Kullanıcı kendisini mention ettiğinde bildirim gösterilmez (self-mention ignore).

Geliştirilen bu temel özellikler, bir araya gelerek Chord iOS'u sadece işlevsel değil, aynı zamanda kullanıcı dostu, modern ve rekabetçi bir iletişim platformu haline getirmektedir.

5. GELİŞTİRME SÜRECİ VE KARŞILAŞILAN ZORLUKLAR

5.1. Fazlara Ayrılmış Geliştirme Yaklaşımı

Projenin başarısı, planlı, yapılandırılmış ve fazlara ayrılmış bir geliştirme sürecine dayanmaktadır. chord_ios_roadmap.md belgesinde detaylandırıldığı gibi, projenin tamamı küçük, yönetilebilir ve hedefe odaklı fazlara bölünmüştür. Bu modüler yaklaşım, geliştirme sürecindeki ilerlemeyi net bir şekilde izlemeyi kolaylaştırmış, potansiyel risklerin erken tespit edilmesine imkan tanımış ve her fazın sonunda somut çıktılar elde edilmesini sağlamıştır.

Projenin ana geliştirme fazları şu şekilde özetlenebilir:

- **Faz 1-2:** Proje iskeletinin oluşturulması, temel paketlerin entegrasyonu, navigasyon yapısının kurulması ve JWT tabanlı kimlik doğrulama (Auth) sisteminin tamamlanması.
- **Faz 3-5:** Sunucu, kanal, mesajlaşma ve kullanıcı durumu (presence) gibi temel arayüzlerin geliştirilmesi. Bu fazda, SignalR bağlantıları kurularak uygulamanın gerçek zamanlı yeteneklerinin temeli atılmıştır.
- **Faz 6-7:** WebRTC (LiveKit) entegrasyonu ile sesli kanalların hayata geçirilmesi. Bağlantı, ses aktarımı ve durum senkronizasyonu bu fazda stabil hale getirilmiştir.
- **Faz 7.5-9:** WebRTC bağlantı stabilitesi iyileştirmeleri, arkadaşlık sistemi, dosya yükleme ve yerel bildirimler gibi kritik düzeltmelerin ve ileri düzey özelliklerin eklenmesi.
- **Faz 10-12:** Kullanıcı deneyimi (UX) iyileştirmeleri, erişilebilirlik, kapsamlı test süreçleri (widget/integration) ve projenin nihai dokümantasyonunun hazırlanması.

5.2. Karşılaşılan Teknik Zorluklar ve Çözümler

Geliştirme süreci boyunca, projenin teknik derinliğini ve problem çözme yeteneğini ortaya koyan önemli zorluklarla karşılaşılmıştır.

- **Zorluk 1: WebRTC Bağlantı Stabilitesi**
    - **Problem:** LiveKit ile yapılan ilk entegrasyonlarda, sesli kanallara katılırken sık sık onConnectionChangeFAILED hataları alınıyor ve kullanıcılar arasında ses iletimi sağlanamıyordu. Bağlantı durumu kararsızdı ve kopmalar yaşanıyordu.
    - **Çözüm:** Sorunu aşmak için çok yönlü bir strateji izlendi. Öncelikle, bağlantı durumu daha agresif bir şekilde izlenmeye başlandı; periyodik kontroller ve LiveKit olay dinleyicileri ile bağlantının sağlığı sürekli denetlendi. Bağlantı koptuğunda devreye giren, üssel geri çekilme (exponential backoff) algoritması kullanan otomatik bir yeniden bağlanma mantığı geliştirildi. Ayrıca, kullanıcının kanaldan bilinçli olarak ayrıldığı durumları (_isLeavingChannel bayrağı) yöneterek, gereksiz yeniden bağlanma denemelerinin önüne geçildi ve state yönetimi daha sağlam hale getirildi.

- **Zorluk 2: Veritabanı Seçimi ve Değişimi**
    - **Problem:** Projenin başlangıcında, çevrimdışı önbellekleme için modern ve hızlı bir veritabanı olan Isar tercih edilmişti. Ancak, geliştirme sürecinde özellikle Android platformunda Gradle derleme hatalarına ve uyumluluk sorunlarına yol açtığı tespit edildi.
    - **Çözüm:** Projenin ilerlemesini riske atmamak adına stratejik bir karar alınarak veritabanı Hive olarak değiştirildi. Hive'ın saf Dart ile yazılmış olması, daha basit bir API sunması ve platforma özgü bağımlılıklarının daha az olması, onu daha stabil bir seçenek haline getirdi. Bu geçiş, projenin derleme süreçlerini sorunsuz hale getirmiş ve geliştirme hızını artırmıştır.
- **Zorluk 3: Gerçek Zamanlı Durum Senkronizasyonu**
    - **Problem:** Okunmamış özel mesaj (DM) sayısı veya bekleyen arkadaşlık istekleri gibi verilerin, kullanıcı arayüzünde anlık olarak güncellenmesinde eksiklikler yaşanıyordu. Örneğin, bir DM okunduğunda veya yeni bir arkadaşlık isteği geldiğinde, kullanıcının ilgili ekranı manuel olarak yenilemesi gerekiyordu.
    - **Çözüm:** Sorunun kaynağının, backend'den gelen bazı SignalR olaylarını dinleyen listener'ların mobil istemcide eksik olması olduğu anlaşıldı. DMMarkAsRead gibi eksik olay dinleyicileri ChatHub entegrasyonuna eklendi. Arkadaşlık isteği bildirimlerinin daha güvenilir olması için ilgili olay dinleyicileri ChatHub'dan PresenceHub'a taşındı. Bu değişikliğin temel nedeni, arkadaşlık durumu gibi kullanıcıya özgü ve global olan bildirimlerin, kanal bazlı iletişimden sorumlu ChatHub yerine, kullanıcının uygulama genelindeki varlığından sorumlu olan PresenceHub üzerinden yönetilmesinin mimari olarak daha tutarlı olmasıdır. Riverpod state yönetimi yapısı, bu yeni olayları dinleyerek ilgili durumu anında güncelleyecek ve arayüze yansıtacak şekilde revize edildi.

Karşılaşılan her zorluk, projenin kod tabanını daha dayanıklı, hatalara karşı daha toleranslı ve sonuç olarak daha olgun bir ürün haline getirmiştir.

6. SONUÇ

6.1. Projenin Değerlendirilmesi

Chord iOS projesi, başlangıçta belirlenen hedeflere başarıyla ulaşmıştır. Modern mobil geliştirme teknolojileri olan Flutter ve Riverpod'ı, .NET, SignalR ve LiveKit gibi güçlü bir backend altyapısıyla birleştirerek, kapsamlı, işlevsel ve performanslı bir gerçek zamanlı iletişim uygulaması ortaya konulmuştur. Geliştirilen kimlik doğrulama, anlık mesajlaşma, sesli sohbet, arkadaşlık sistemi ve çevrimdışı destek gibi özellikler, projenin "ağ tabanlı ve gerçek zamanlı iletişim paradigmalarını mobil bir platformda uygulama" şeklindeki temel akademik hedefini başarıyla karşıladığını göstermektedir. Proje, hem teknik bir başarı hem de modern mobil uygulama geliştirme prensiplerinin pratik bir uygulaması olarak değerlendirilmektedir.

6.2. Gelecek Geliştirmeler

Proje, sağlam bir temel üzerine inşa edilmiş olup, gelecekte eklenebilecek birçok potansiyel geliştirme ve iyileştirme alanı barındırmaktadır. chord_ios_roadmap.md belgesinde yer alan planlar doğrultusunda, öngörülen bazı gelecek adımlar şunlardır:

- **Push Bildirimleri:** Uygulama arka plandayken veya kapalıyken kullanıcılara anlık bildirimler (yeni mesajlar, @bahsetmeler, arkadaşlık istekleri) göndermek için APNs (Apple Push Notification service) ve FCM (Firebase Cloud Messaging) entegrasyonunun yapılması.
- **Performans Optimizasyonları:** Uygulama açılış süresini kısaltmak, batarya tüketimini ve ağ kullanımını azaltmak amacıyla kod bölme (code splitting), tembel yükleme (lazy loading) ve sık güncellenen widget'larda memoization gibi ileri düzey performans iyileştirme tekniklerinin uygulanması.
- **Kapsamlı Test Süreçleri:** Uygulamanın kararlılığını ve güvenilirliğini en üst düzeye çıkarmak için mevcut testlerin kapsamının genişletilmesi. Özellikle kritik kullanıcı akışlarını kapsayan widget ve entegrasyon testlerinin sayısının artırılması.
- **Kullanıcı Deneyimi (UX) İyileştirmeleri:** Veri yüklenirken kullanılan iskelet ekranlar (skeleton screens), listeleri yenilemek için "aşağı çek" (pull-to-refresh) hareketi ve görme engelli kullanıcılar için VoiceOver gibi gelişmiş erişilebilirlik (accessibility) özelliklerinin eklenmesiyle uygulamanın daha akıcı ve kapsayıcı hale getirilmesi.
- **Gelişmiş Etkileşimler:** Mesaj silme veya kanal susturma gibi işlemler için kaydırma hareketleri (swipe gestures) gibi modern mobil etkileşim desenlerinin entegrasyonu.

7. KAYNAKÇA

1. Flutter Documentation. https://docs.flutter.dev
2. Dart Documentation. https://dart.dev/guides
3. Riverpod (riverpod.dev) Documentation. https://riverpod.dev
4. go_router Package (pub.dev). https://pub.dev/packages/go_router
5. Dio Package (pub.dev). https://pub.dev/packages/dio
6. signalr_core Package (pub.dev). https://pub.dev/packages/signalr_core
7. Hive Package (pub.dev). https://pub.dev/packages/hive
8. flutter_secure_storage Package (pub.dev). https://pub.dev/packages/flutter_secure_storage
9. Sentry for Flutter Documentation. https://docs.sentry.io/platforms/flutter/
10. image_picker Package (pub.dev). https://pub.dev/packages/image_picker
11. video_player Package (pub.dev). https://pub.dev/packages/video_player
12. flutter_local_notifications Package (pub.dev). https://pub.dev/packages/flutter_local_notifications
13. LiveKit Documentation. https://docs.livekit.io
14. livekit_client Package (pub.dev). https://pub.dev/packages/livekit_client
15. connectivity_plus Package (pub.dev). https://pub.dev/packages/connectivity_plus
16. shared_preferences Package (pub.dev). https://pub.dev/packages/shared_preferences
17. ASP.NET Core Documentation. https://learn.microsoft.com/aspnet/core/
18. SignalR for ASP.NET Core Documentation. https://learn.microsoft.com/aspnet/core/signalr/introduction
19. Microsoft SQL Server Documentation. https://learn.microsoft.com/sql/sql-server/
20. MinIO Documentation. Erişim: https://min.io/docs/minio

EKLER

EK-1 - Chord Mobil Arayüzü

EK-1 - Chord Mobil Arayüzü