ChatGPT

Çok Kullanıcılı Anı Defteri Uygulaması – Özellik Analizi ve Uygulama Planı

Genel Bakış

Bu uygulama, **Notion benzeri bir not tutma veya yapılacaklar listesi aracı değil**, birden fazla kullanıcının bir araya gelerek metin, fotoğraf, video, ses gibi çeşitli medya türleriyle **anılarını kaydedebileceği bir dijital günlük** olarak tasarlanıyor. Hedef platformlar Android ve iOS (mobil ve tablet), dolayısıyla uygulama deneyiminin mobil cihazlara uygun, akıcı ve kullanıcı dostu olması kritik önemde. Mevcut prototip uygulamada bazı temel özellikler bulunuyor, ancak **yayınlanabilir düzeyde rekabetçi bir ürün için eklenmesi gereken birçok özellik ve servis eksik**. Aşağıda, benzer uygulamalarda bulunan ancak mevcut uygulamada eksik olan tüm önemli özellikler listelenmiş ve her biri için ayrıntılı bir implementasyon planı sunulmuştur.

Eksik Temel Özellikler ve Servisler

1. Profil ve Ayarlar Ekranının Birleştirilmesi

Neden gerekli: Kullanıcıların profil bilgilerini görüp düzenleyebilecekleri ve uygulamanın temasından bildirim ayarlarına kadar tüm tercihlerinin yönetilebileceği merkezi bir **"Profil & Ayarlar"** ekranı, modern uygulamalarda beklenen bir kolaylıktır. Mevcut prototipte profil yönetimi ve uygulama ayarları ayrı sayfalar olarak düşünülmüş durumda. Bunları tek bir yerde toplamak, kullanıcı deneyimini iyileştirir ve uygulamayı yayınlarken daha profesyonel bir izlenim verir [1] [2].

Neler içermeli:
- **Profil Bilgileri:** Kullanıcının avatarı, görünen adı, e-posta adresi ve uygulamaya özel kimlik numarası (örn. *J-XXXX* formatında) tek bir kart üzerinde gösterilmelidir [3] [4]. Avatar'a dokunarak profil fotoğrafı değiştirmek (örn. görüntü seçici açılması) olanağı sunulabilir.
- **Ayarlar:** Tema modu (Sistem/Açık/Koyu) seçimi, renk teması seçimi, veri yedekleme ve geri yükleme, önbellek temizleme, veritabanı boyutu gösterimi, hesap çıkışı, uygulama hakkında (sürüm, lisans) gibi alt bölümler halinde listelenmelidir [5] [6]. Özellikle tema ve renk seçimi sayesinde kullanıcı arayüzü kişiselleştirilebilir hale gelecektir.
- **Teknik altyapı:** Bu ekran bir *ConsumerStatefulWidget* olarak Riverpod kullanarak tasarlanabilir. Mevcut `UserService` akışı ile kullanıcının profil verileri (isim, fotoğraf URL, vb.) gerçek zamanlı alınacak, `ThemeProvider` aracılığıyla tema tercihi kaydedilip uygulanacak, `AuthService` ile çıkış yapılacaktır [7] [8]. Gerekli durumlarda `SharedPreferences` ile kalıcı ayarlar (örn. seçilen tema rengi) saklanabilir.

Uygulama Adımları:
1. **Yeni Ekran:** `profile_settings_screen.dart` adıyla yeni bir ekran oluşturulacak. Bu ekranda üst kısım profil, alt kısım ayarlar olarak bölünecek [1] [9].
2. **Profil Bölümü:** Kullanıcı avatarı ve bilgileri üstte büyükçe gösterilecek. Avatar için mevcutsa Firebase

Storage'daki fotoğraf kullanılacak, yoksa varsayılan bir ikon gösterilecek. İsim, e-posta ve kopyalanabilir kullanıcı ID'si alt alta sıralanacak [3].
3. **Ayarlar Bölümü:** Alt kısımda "Görünüm", "Veri Yönetimi", "Hesap" ve "Hakkında" şeklinde bölümler olacak. Görünüm altında tema modu seçiciler ve renk paleti sunulacak; Veri yönetimi altında **Yedekleme** (varsa Google Drive veya cihaz depolamasına dışa aktarma), **Önbellek Temizleme** ve toplam veri kullanımı gösterimi bulunacak [5]. Hesap bölümünde çıkış yapma; Hakkında'da uygulama sürümü ve lisans bilgileri sunulacak [10] [6].
4. **Navigasyon:** Mevcut yandan açılan menüdeki (Drawer) "Ayarlar" girişi, bu yeni Profil&Ayarlar ekranına yönlendirecek şekilde güncellenecek. Eski ayrı Ayarlar ekranı uygulamadan kaldırılacak veya devre dışı bırakılacak [11].

Bu birleşik ekran sayesinde kullanıcılar uygulama ve hesap ayarlarını tek bir yerden yönetebilecekler. Planlanan implementasyon, mevcut kod altyapısına kolaylıkla entegre edilebilir nitelikte olup (ör. `UserService.myProfileStream` gibi mevcut API'lar kullanılıyor [12]), özellik eklendiğinde test senaryoları olarak profil bilgilerinin doğru görüntülenmesi, tema değiştirince uygulama görünümünün güncellenmesi, yedekleme ve önbellek temizleme fonksiyonlarının çalışması gibi durumlar kontrol edilecektir.

2. Tema Desteği ve Renk Özelleştirme

Neden gerekli: Karanlık/aydınlık mod desteği ve farklı vurgu renk temaları sunmak, günümüzde kullanıcıların uygulamalardan beklediği temel özelleştirme seçenekleridir. Mevcut uygulamada yalnızca tek bir varsayılan tema olabilir; yayınlanacak sürüm için kullanıcılara sistem temasına uyum, açık veya koyu mod seçimi ve belki birkaç ön tanımlı renk paletinden tercih yapma olanağı verilmelidir [13] [14]. Bu, uygulamayı kişiselleştirilebilir ve geniş kitlelere hitap edebilir hale getirir.

Neler içermeli:
- **Tema Modu:** Uygulama, cihazın varsayılan ayarına (sistem modu) uyacak şekilde otomatik olabilir ya da kullanıcı zorla açık/koyu seçebilir. Bir RadioListTile grubu ile seçim UI'da sunulabilir [14].
- **Renk Temaları:** Uygulamanın vurgu rengi (butonlar, başlıklar vs.) için birkaç seçenek (ör. Mavi, Yeşil, Mor, vb.) tanımlanmalı. Profil-Ayarlar ekranında bir renk paleti seçici (ör. küçük renk daireleri) ile kullanıcı dilediğini seçebilmelidir [15].
- **Uygulama:** Seçilen tema ve renk, anında uygulama genelinde uygulanmalı. Flutter'da `ThemeMode` ve tema verileri (`ThemeData`) Riverpod ile yönetilebilir; örneğin `ThemeNotifier` yardımıyla kullanıcı tercihi kaydedilip `main.dart` içinde MaterialApp'in temasına yansıtılacak [16].

Uygulama Adımları:
1. **Tema Seçici UI:** Profil&Ayarlar ekranındaki "Görünüm" bölümüne, sistem/açık/koyu mod için seçenekler ve renk paleti için bir özel widget eklenecek [14]. Bu widget, ön tanımlı `AppColorTheme` değerlerini listeleyip seçimi sağlayacak.
2. **Durum Yönetimi:** `themeProvider` ile seçilen değerler saklanacak. Zaten uygulamada Riverpod kullanıldığı için, örneğin `setThemeMode(ThemeMode mode)` ve `setColorTheme(AppColorTheme theme)` metodları halihazırda tanımlı ya da tanımlanacak ve paylaşılan durum üzerinden tüm UI'nın yeniden çizilmesini sağlayacak [16].
3. **Ayarların Kalıcılığı:** Seçimler `SharedPreferences` veya benzeri bir mekanizmayla cihazda tutulacak ki uygulama yeniden açıldığında kullanıcı teması korunabilsin [17].
4. **Main.dart Entegrasyonu:** MaterialApp'in theme ve darkTheme parametreleri, kullanıcı tercihlerine göre

ayarlanacak. Renk temaları ise ThemeData içinde accentColor/primarySwatch gibi değerler üzerinden uygulanabilir. Bu entegrasyonun düzgün çalıştığı test edilmeli (ör. bir toggle ile tema değiştirip anında görünüm değişiyor mu) [18].

Tema desteği, uygulamayı hem **kullanıcı deneyimi açısından çekici** kılar, hem de farklı ortamlarda (gece/gündüz) rahat kullanım sağlar. Bu özellik, profil/ayarlar ekranı ile entegre şekilde yukarıdaki adımlarla kolayca hayata geçirilebilir.

3. Zengin Medya İçeriği Desteğinin Tamamlanması

Neden gerekli: Bir anı defteri uygulamasının öne çıkan yönü, salt metin değil **fotoğraf, video, ses kaydı, çizim** gibi zengin medya öğelerini desteklemesidir. Mevcut uygulama prototipinde metin, görüntü, video ve temel seviyede ses kaydı blokları bulunuyor [19] [20]. Ancak, bu alanlarda bazı eksikler mevcut (örneğin ses kaydı süresi gösterimi yok, çizim desteği tam olmayabilir). Yayınlanacak versiyonda tüm bu medya türlerinin sorunsuz eklenip görüntülenebilmesi, kullanıcıların beklentisini karşılamak için şarttır. Örneğin popüler Journey günlüğü uygulamasında kullanıcılar **fotoğraf, video, GIF ve ses dosyalarını günlüklerine kolayca ekleyebiliyorlar** [21] – bu artık standart bir özellik haline gelmiş durumda.

Neler içermeli:
- **Gelişmiş Ses Kaydı:** Halihazırda bir ses kayıt servisi var ancak kayıt süresi gösterimi veya dalga formu görselleştirmesi gibi detaylar eksik [22]. Ses kayıt bloğu iyileştirilerek kayıt süresi sayaçları, çalma/duraklatma kontrolleri ve belki kayıt kalitesi ayarları eklenmeli.
- **Çizim (El Yazısı) Desteği:** Kullanıcıların parmak veya kalemle çizimler yapıp ekleyebileceği bir çizim özelliği planlanmalı. Arka planda, sayfalarda doodle/ink verileri için bir alan zaten düşünülmüş (inkData şeklinde) [23], bunu son kullanıcıya açacak bir arayüz (tuval ekranı) ve çizimi kaydedip tekrar görüntüleme işlevi eklenmeli. Çizimler daha sonra çıkartmaya dönüştürülebilecek şekilde de kullanılabilir (aşağıda Sticker özelliğine bakınız).
- **Medya Entegrasyonu:** Fotoğraf ve videolar için halihazırda **görsel/video blok widget'ları** mevcut [19]. Bunlara ek olarak, çoklu fotoğraf galerisi (birden fazla fotoğrafı kolaj şeklinde ekleme) veya videolar için tam ekran izleme modları gibi iyileştirmeler değerlendirilebilir. Journey uygulaması, hatıraları zenginleştirmek için resim ve videoların yanı sıra konum ve hava durumu bilgileri de ekleme olanağı sunuyor [24] [25]; biz de en azından konum etiketleme gibi bir özelliği ileride düşünebiliriz.

Uygulama Adımları:
1. **Ses Bloğunun İyileştirilmesi:** `AudioRecorderService` ve ilgili widget'larda süre takibi eklenmesi (kayıt sırasında geçen süreyi gösteren bir sayaç) ve kaydedilen sesi oynatmak için bir oynatıcı arayüzü hazırlanması. Gerekirse dalga formu çizmek için bir paket (ör. **flutter_sound** veya **waveform visualization** paketleri) entegre edilebilir.
2. **Çizim Bloğunun Eklenmesi:** Yeni bir "çizim" blok türü tanımlanacak. `flutter_drawing_board` gibi bir paket yardımıyla kullanıcının çizim yapabileceği bir tam ekran modal veya ayrı bir ekran sunulabilir [26] [27]. Çizim tamamlandığında bu bir image (PNG) olarak kaydedilip sayfaya eklenecek veya vektörel olarak saklanacak (ilk sürüm için PNG yeterli). Bu sayede kullanıcı el yazısıyla notlar veya küçük eskizler de ekleyebilir.
3. **Medya Depolama:** Bütün medya içerikleri (foto, video, ses, çizim) offline-first mimariye uygun şekilde **lokal veritabanına referanslarıyla kaydedilip** Firebase Storage'a yüklenecek. Mevcut uygulamada Firebase Storage entegrasyonu medya dosyaları için zaten düşünülmüş [28]. Bu yapının tüm yeni medya türlerini kapsadığından emin olmak gerekiyor (örn. çizimler de storage'a yüklenecek).

4. **Testler:** Her medya türü için ekle/görüntüle işlemleri cihazdan cihaza test edilmeli (farklı boyut ve formatlardaki resimler, videolar, vb. düzgün işleniyor mu; internet yokken ekleme yapılıp sonradan senkronize oluyor mu; vb.). Ayrıca medya ekleme arayüzlerinin (galeriden seçim, kamera açma, mikrofon izni vb.) kullanıcı akışı sorunsuz olmalı.

Bu iyileştirmelerle birlikte, uygulama "anı yakalama" konusunda tam teşekküllü hale gelecek. Kullanıcılar günlüklerine zengin içerik eklerken herhangi bir eksiğin olmadığını görmeli, fotoğraflarla dolu anılar veya sesli günlükler tutabilmelidir. Zaten popüler günlük uygulamaları da benzer zengin medya desteğine sahip durumda [21], dolayısıyla rekabetçi olmak için bu özelliklerin tamamlanması şarttır.

4. Arama ve Kategorize Etme Özellikleri

Neden gerekli: Zamanla kullanıcılar birçok günlük girişi biriktirebilir. Bunların içinden belirli bir anıyı veya konuyu bulabilmek için uygulamada **arama yapabilme** ve içerikleri etiketleme/kategorize etme özelliği önemlidir. Mevcut uygulama tasarımında kullanıcıların birden fazla ayrı "journal" (günlük defteri) oluşturabildiği anlaşılıyor (ör. her biri bir kategori gibi düşünülebilir), fakat tek bir günlük içinde arama yapma veya etiket bazlı filtreleme desteği henüz yok. Yayınlamadan önce, özellikle metin içeriklerde kelime araması yapabilmek ve belki **etiketler (tags)** atayarak girişleri sınıflandırabilmek uygulamaya değer katacaktır.

Neler içermeli:
- **Metin Arama:** Kullanıcı bir anahtar kelime girdiğinde, o kelimenin geçtiği günlük sayfalarını listeleyebilmeliyiz. Arama motoru basit bir full-text search olabilir. Flutter tarafında, tüm günlük içeriklerini (metin bloklarını) indeksleyip aramak için SQLite'ın FTS özelliği veya Firestore'un arama kabiliyeti (sınırlı) kullanılabilir. Gerekirse basitçe tüm sayfaları çekip client tarafında aramak da ilk sürüm için uygulanabilir.
- **Etiketleme:** Her bir sayfaya kullanıcı bir veya birden fazla etiket ekleyebilmeli (örn. #tatil, #iş, #aile gibi). Bu etiketler sayesinde benzer içerikler gruplanabilir. Uygulamada halihazırda birden fazla defter oluşturma imkanı olduğundan, etiket şart olmayabilir ancak tek bir defter içinde konulara göre filtreleme için faydalı olacaktır.
- **Tarihsel/Takvim Görünümü:** (İsteğe bağlı) Günlük girdilerinin tarihine göre bir takvim üzerinden navigasyon veya kronolojik bir zaman tüneli sunmak da kullanıcıların anılarını zaman bazlı bulmalarını kolaylaştırır. Örneğin Journey uygulaması "Atlas" ve "Calendar" görünümü ile geçmişe dönük gezinmeyi keyifli hale getiriyor [29]. Bizim uygulamada da, sayfaların oluşturulma tarihine göre sıralanabildiği veya takvimden seçilebildiği bir arayüz eklenebilir.

Uygulama Adımları:
1. **Arama Servisi:** `SearchService` adıyla bir servis oluşturulacak. Bu servis, parametre olarak verilen sorgu kelimesine göre lokal veritabanında sayfa içeriklerini arayacak. Örneğin, SQLite kullanıldığı için bir FTS5 sanal tablo kurulabilir veya mevcut sayfa tablolarına bir *content* sütunu eklenip LIKE aramaları yapılabilir. Performans için uygun görülürse Firestore tarafında basit bir indeksleme de yapılabilir ancak istemci tarafı arama öncelikli olmalı (offline desteği için).
2. **Arama UI:** Uygulamanın menüsüne veya ana ekranına bir arama butonu/ikon eklenmeli. Hatta yandaki navigasyon çekmecesinde (drawer) kullanıcı avatarının yanında arama ikonu konulması planlanmıştır [30] [31]. Bu ikona tıklandığında bir arama sayfası veya dialog açılarak kullanıcı kelime girebilecek ve sonuçlar listelenecektir. Sonuç listesinde, ilgili sayfanın başlığı veya ilk birkaç kelimesi gösterilip tıklandığında o sayfaya gidilmesi sağlanır.
3. **Etiketleme:** Sayfa düzenleyicisine (editor) bir "etiket ekle" işlevi konulabilir. Basit bir metin girişiyle

kullanıcı etiket(ler) yazıp kaydedebilir. Bu etiketler, sayfanın veritabanı kaydında ayrı bir alanda saklanır ve istenirse Firestore'a da eklenir. Daha sonra etiket listesine göre filtreleme yapacak bir UI hazırlanır (örn. *etiket bulutu* veya etiket listesi). Bu kısım opsiyonel bir iyileştirme olup, öncelik arama özelliğindedir.

Arama/filtreleme fonksiyonlarının eklenmesiyle, kullanıcılar uygulamada çok sayıda içerik biriktiğinde bile istediklerine kolayca ulaşabilecekler. Bu özellikler yayınlanan uygulamanın **kullanışlılığını ve profesyonelliğini** arttıracaktır, zira kullanıcılar genellikle eski bir anıyı veya belirli bir konuyu hızlıca bulabilmek isterler.

5. Çoklu Kullanıcı Ekip (Ortak Journal) Sistemi

Neden gerekli: Uygulamamızın alametifarikası, **birden fazla kişinin ortaklaşa bir günlük tutabilmesi**. Mevcut prototipte arkadaş ekleme gibi temel sosyal özellikler olsa da, tam anlamıyla ortak bir journal yapısı henüz yok. "Ekip Sistemi" adı altında, kullanıcıların bir grup oluşturup o grup üyeleriyle belirli bir günlük defterini paylaşabilmesi sağlanmalı [32] [33]. Bu sayede örneğin bir aile albümü, arkadaş grubu anı defteri veya çiftlerin ortak günlüğü mümkün olur. Day One uygulamasında yeni tanıtılan "Shared Journals" özelliği tam da bunu sağlıyor ve kullanıcıların **davetle katılabildiği, ortak anı ekleyebildiği güvenli günlükler** oluşturmasına imkan veriyor [34]. Bizim uygulamamızda da benzer şekilde, ekip kurma ve ekipçe yazma desteği olmazsa, birden fazla kullanıcılı deneyim eksik kalır.

Neler içermeli:
- **Ekip & Roller:** Kullanıcılar bir "ekip" (group) oluşturabilmeli, bu ekibe bir isim (ve belki simge/avatar) verebilmeli ve diğer üyeleri davet edebilmeli. Ekip üyeleri arasında *sahip*, *düzenleyici* ve *izleyici* gibi roller olmalı ki içerik yönetimi kontrollü olsun [35] [36]. Örneğin sadece sahipler yeni üye davet edebilir, düzenleyiciler içerik ekleyip düzenleyebilir, izleyiciler sadece okuyabilir.
- **Ortak Günlükler:** Her journal, artık *kişisel* veya *ekip* olarak işaretlenmeli. Ekip günlükleri belirli bir ekip ID'sine bağlı olacak ve o ekip üyelerinin erişimine açık olacak [37]. Kişisel journal'lar ise sadece sahibine görünür. Day One'da kişisel ve paylaşılan günlükler ayrıdır ve paylaşılan günlüklerdeki içeriklerin sadece davetlilere göründüğü garanti edilir [38] [39]. Bizim sistemde de günlük kayıtlarına bir `type` ve `teamId` alanı ekleyerek bu ayrımı yapacağız [40].
- **Ekip Yönetim Arayüzü:** Kullanıcı, ekiplerini listeleyebildiği bir ekran görecek (örn. "Ekiplerim" listesi) [41]. Her ekibin içinde ekip bilgilerini düzenleyebileceği (isim, açıklama, avatar) ve üyeleri yönetebileceği (üyeleri listeleme, rollerini değiştirme, ekipten çıkarma) bir yönetim ekranı olacak [42] [43]. Bu ekranlarda sadece ekip sahibi gerekli yönetim yetkilerine sahip olmalı.
- **Senkronizasyon:** Ekip bilgileri ve ekip günlükleri de Firestore ve lokal veritabanı arasında senkronize olacak. Yani offline çalışan biri kendi ekibine eklediği girdiyi çevrimiçi olunca ekipteki herkesin cihazına aktarabilecek. Mevcut senkronizasyon alt yapısı (Hybrid Logical Clock tabanlı oplog) bu yapıya uygun şekilde genişletilmeli [44] [45].

Uygulama Adımları:
1. **Model ve Veritabanı:** `Team` ve `TeamMember` modelleri tanımlanacak [46] [47]. SQLite tarafında **teams** ve **team_members** tabloları oluşturulacak, bir de mevcut **journals** tablosuna `teamId` ve `type` sütunları eklenecek (günlük bir ekibe bağlıysa personal/team ayrımı için) [37]. Gerekli DAO ve migrasyonlar yazılacak. Firestore'da da benzer koleksiyon yapıları (teams, team_members) oluşturulacak.
2. **Servisler:** `TeamService` adında bir servis katmanı eklenecek. Bu servis, ekip oluşturma (`createTeam`), ekip üyelerini okuma (`watchTeamMembers`), üye ekleme/çıkarma, rol değiştirme gibi metodlar içerecek [48] [49]. Ayrıca mevcut `FirestoreService` ve `SyncService` ekipleri destekleyecek

şekilde güncellenecek (ör. yeni bir ekip oluşturulduğunda Firestore'a yazma, vs.) [50] [51] .
3. **UI Ekranları:**
- **Ekip Listesi:** Menüde "Ekipler" girişi olabilir veya Journallar listesinde ayrı kategorize edilebilir. Ekip listesi ekranında kullanıcının üyesi olduğu tüm ekipler kartlar halinde gösterilecek (ekip adı, kaç üye, son aktivite gibi bilgiler) [41] . Yeni ekip eklemek için bir buton (FAB) bulunacak.
- **Ekip Detayı & Üye Yönetimi:** Bir ekip seçildiğinde, ekip detay ekranı açılacak. Burada ekip adı/ açıklaması düzenlenebilir, ekip avatarı değiştirilebilir, alt kısımda üye listesi görülebilir [52] [43] . Üye listesinde her üyenin adı, rolü belirtilir; eğer yönetici (owner) isek rollerini değiştirmek veya ekipten çıkarmak için seçenekler sunulur [53] [54] . Ayrıca yeni üye ekleme butonu da burada yer alır.
- **Ortak Journal İşaretleme:** Yeni bir journal oluştururken bunun kişisel mi ekip mi olduğu seçilecek (ör. bir "Ekip Seç" alanı veya toggle ile). Ekip journalları, ilgili ekip üyelerine "paylaşılan" olarak görünür olacak. Uygulamanın anasayfasında belki paylaşılan journallar için bir ikon veya ayrı bölüm konabilir, böylece kullanıcı hangi defterlerin ortak olduğunu bilir (Day One, paylaşılan günlükleri ayrı tutuyor ve gösteriyor [55]).
4. **Erişim Kontrolü:** Uygulamanın her günlük sayfası açılışında, eğer sayfa bir ekip günlüğüne aitse ve mevcut kullanıcı o ekip üyesi değilse erişim engellenmeli. Bu tür güvenlik kontrolleri hem lokalde (UI'da gösterme) hem bulutta (Firestore rules) uygulanacak [44] . Örneğin Firestore tarafında bir journal'ın teamId'si varsa sadece o ekibin üyeleri okuyabilsin kuralı yazılacak.

Ekip sistemi uygulandıktan sonra, uygulamamız gerçekten **"paylaşılan anı defteri"** konseptini karşılamış olacak. Kullanıcılar sevdikleriyle ortak günlük tutabilecek, birlikte fotoğraflar ekleyip yazılar yazabilecekler. Bu, ürünün öne çıkan özelliği olarak doğru şekilde çalışmalı ve kolay kullanılabilir olmalı. Day One'ın paylaşılan günlükler özelliği dahi Premium abonelik gerektirirken, bizim uygulamada bu özelliğin mevcut olması önemli bir rekabet avantajı sağlayabilir [34] [56] .

6. Davet Sistemi (Kullanıcı Davet Etme)

Neden gerekli: Ekip veya ortak journal özelliğinin doğal bir uzantısı olarak, **başka kullanıcıları günlük paylaşımına davet etmek** şarttır. Kullanıcılar uygulama içinde arkadaşlarını bulup ekleyebilmeli veya uygulama dışından bir bağlantı ile davetiye göndererek yeni kullanıcılar çekebilmelidir. Mevcut uygulamada arkadaş ekleme sistemi olduğu belirtilmiş (displayId üzerinden ekleme) [57] , ancak spesifik bir davet/katılım mekanizması yok. Bu özellik, uygulamanın sosyal yönünü güçlendirir ve kullanıcı tabanını büyütmeyi kolaylaştırır (örneğin, bir kullanıcı aile üyelerini ortak günlüğe katılmaya davet edebilir).

Neler içermeli:
- **Davet Linki Oluşturma:** Kullanıcı, bir ekip veya journal için davet linki üretebilmelidir. Bu link, başka birine gönderildiğinde (örn. SMS, WhatsApp ile) tıklandığında uygulamayı açarak ilgili daveti işler. Linkler güvenli olmalı ve süreli/sınırlı kullanılabilir olabilir (ör. 7 gün içinde kullanılmazsa geçersiz).
- **Uygulama İçi Davet:** Uygulama içindeki arkadaş listesinden veya e-posta ile arama yaparak bir kullanıcıyı doğrudan davet edebilme imkanı da olmalı. Örneğin, "Üye Ekle" dediğimizde çıkan ekranda bir kullanıcı arama alanı (kullanıcı adını veya email ile) ve bulduğun kişiyi davet et seçeneği olacak [58] .
- **Davet Yönetimi:** Gönderdiğim davetiyeler ve aldığım davetiyeler bir listede takip edilebilmeli. Kullanıcı, kendisine gelen davetleri bir **bildirim** olarak görmeli ve kabul/ret işlemi yapabilmeli. Aynı şekilde gönderdiği davetlerin kabul edilip edilmediğini de izleyebilmeli.
- **Derin Link & Onboarding:** Davet linkine tıklayarak gelen yeni kullanıcı, uygulama yüklü değilse önce mağazaya yönlendirilmeli, yükleyip açtığında davetin bilgisi (hangi ekibe/journal'a davet edildiği) uygulama

içinde anlaşılmalı ve ona göre kayıt sonrası otomatik katılım veya onay süreci işletilmeli. Bu uçtan uca akış, kullanıcı deneyimi açısından mümkün olduğunca pürüzsüz olmalı.

Uygulama Adımları:
1. **Model ve Veritabanı:** `Invite` adında bir model oluşturulacak. Davetiyenin bir ID'si, ne tip davet olduğu (journal mı takım mı), hedef ID'si (ilgili journal veya takım), daveti gönderen ve alan kullanıcı ID'leri, durum (beklemede/kabul/ret), rol (özellikle takım davetiyse rolü de belirtebilir) ve opsiyonel bir mesaj içerecek [59] [60]. SQLite'da **invites** tablosu kurulacak ve Firestore'da da invites koleksiyonu tutulacak.
2. **Invite Servisi:** `InviteService` adında bir servis yazılacak. Bu servis: davet oluşturma (`createInvite`), davet linki oluşturma (bir davetiye nesnesine karşılık gelen URL üretme), daveti kabul etme, reddetme, davetleri listeleme gibi metodlar içerecek [61] [62]. Davet linki oluştururken belki **UUID** tabanlı bir kod kullanılabilir. Mevcut `uuid` paketi işimize yarayacak [63].
3. **UI ve Bildirimler:**
- Davet gönderirken: Bir *Invite Dialog* hazırlanacak [64]. Örneğin bir journal içinde "Paylaş" dediğimizde karşımıza "Davet Et" seçeneği gelecek ve burada kullanıcı arayıp seçebileceğiz veya link oluşturabileceğiz. Kullanıcı arama kısmı, arkadaş listesinden veya global kullanıcı veritabanından arama yapabilir [58]. Kişi seçildikten sonra davet yolla işlemi `InviteService.createInvite` çağırır.
- Gelen davetler: Uygulamada bir **Bildirimler** ekranı veya sekmesi olacak. Drawer menüsünde "Bildirimler" seçeneği ve yanında bekleyen davet sayısı rozeti planlandı [65] [66]. Bu ekranda aldığınız davetler listelenir, her birinin kabul/ret butonları olur. Kabul edilince arka planda ilgili journal/team verileri kullanıcıya eklenir (örn. takım üyesi tablosuna kayıt eklenir) [67] [68]. Reddedilirse davet güncellenir. Gönderen kişi de belki kendi gönderdiği davetin kabul edildiğini bu listeden veya bir bildirimden görebilir.
- Derin Linkler: Davet linki tıklanınca uygulamanın belirli bir URL scheme ile açılması gerekir. Hem Android'de hem iOS'te deep link entegrasyonu yapılacak [69]. Uygulama kapalıysa açıp ilgili daveti gösterme, açıksa doğrudan navigasyon yapma gibi senaryolar test edilmeli [69]. Bu kısım teknik ama kullanıcı için görünmez; doğru çalışması kritik.

Davet sistemi, uygulamanın sosyal bağlarını güçlendirecek ve yeni kullanıcı kazanımını kolaylaştıracaktır. Örneğin bir kullanıcı anne-babasını aile anı defterine davet edebilecek; onlar da linke tıklayıp kolayca katılabilecek. Bu sayede uygulama **organik olarak yayılabilir**. Ayrıca davet mekanizması, ileride e-posta veya SMS ile de entegre edilebilir. Day One, paylaşılan günlüklerde kullanıcıları uygulama içi davetle ekliyor ve güvenlik için herkesten Day One hesabı istiyor [70]; biz de benzer şekilde davetlilerin uygulamada hesap oluşturmasını gerektireceğiz, ancak davet linkleriyle bu süreç oldukça akıcı hale getirilecek.

7. Paylaşım Sistemi (Harici veya Kısıtlı Paylaşma)

Neden gerekli: Uygulama içi davet, kullanıcılar arasında tam erişimli paylaşımı sağlıyor. Bunun yanı sıra, kullanıcıların günlük içeriklerini uygulama **dışında seçili kişilerle veya platformlarda paylaşma** ihtiyacı da olabilir. Örneğin bir sayfayı PDF olarak dışa aktarıp e-posta ile göndermek veya belirli bir anıyı web üzerinden görüntülenebilir bir bağlantı olarak paylaşmak istenebilir. Journey uygulaması, bulut senkronizasyonu üzerinden **web linkleri üretip bu linkleri şifre korumalı şekilde paylaşma** özelliği sunuyor [71] [72]. Bu sayede kullanıcılar günlüklerinden seçtikleri bazı kayıtları uygulamayı kullanmayanlarla da güvenli biçimde paylaşabiliyor. Benzer bir özellik, bizim uygulamayı daha esnek kılar.

Neler içermeli:
- **Tekil Girdi Paylaşma:** Kullanıcı, günlük içerisindeki tek bir sayfayı (girdiyi) uygulama dışına aktarabilmeli. Bu, **PDF veya resim** olarak dışa aktarmak şeklinde olabilir ya da özel bir web sayfasında görüntülenmesi için

link üretilebilir. Basit yol olarak, sayfayı PDF'ye dönüştürüp paylaşma özelliği değerlendirilebilir (Waffle uygulaması bile PDF olarak tüm journal'ı dışa aktarmaya izin veriyor [73]).
- **Journal Paylaşma:** Bir bütün günlük defterini (veya en azından okunabilir versiyonunu) paylaşmak belki büyük ölçekli olabilir ancak, özellikle ortak journallarda üyelerin dışında birine tüm içeriği göstermek istenirse kolay yol gerekir. Bu durumda da gene PDF/HTML dışa aktarımı veya web yayını opsiyonu olabilir.
- **Erişim Kontrolü & Güvenlik:** Eğer web link ile paylaşım yapılacaksa, bu linkler **yalnızca bağlantıyı bilenlerin erişebileceği** şekilde olmalı, tercihen şifre koruması da konulabilmeli (Journey bu imkanı tanıyor [74]). Paylaşılan içerikler sunucuda herkese açık barındırılmayacak, gizli ID'lerle veya kısıtlı süreyle yayınlanacak.
- **Diğer Platformlara Entegrasyon:** Kullanıcı belki anısını sosyal medyada paylaşmak isteyebilir. Bunun için sayfayı görüntüye çevirip Instagram/Twitter'da paylaşma kolaylığı sağlanabilir. Bu, uygulamanın kendisi dışında reklamını yapma açısından da faydalı olur.

Uygulama Adımları:
1. **Teknik Altyapı:** Paylaşım için iki yaklaşım düşünülebilir:
a) **PDF/Resim Oluşturma:** Flutter'ın PDF kütüphaneleri (ör. **flutter_pdf** gibi) kullanılarak bir sayfanın içerikleri (metin, foto, vs.) PDF'e dökülebilir. Alternatif olarak sayfanın ekran görüntüsünü alıp resim olarak paylaşmak basit ama her zaman uygun değil (içerik uzun olursa ekran görüntüsü yetmez). PDF yaklaşımı daha bütüncül.
b) **Web Link (Backend):** Eğer bir backend servisi kullanmak opsiyonumuz varsa, paylaşılacak sayfa içeriğini sunucuya (ör. Firestore'a) "shared_entries" gibi bir yapıya atıp bir ID'si üzerinden bulut işleviyle web sayfası üretilebilir. Ancak bu oldukça uğraştırıcı olabilir, ilk sürüm için PDF yeterli.
2. **UI Desteği:** Her sayfanın içinde "Paylaş" butonu eklenecek. Buna basıldığında kullanıcıya "PDF olarak paylaş" veya "Link oluştur" gibi seçenekler sunulabilir. PDF seçilirse arka planda PDF oluşturulup paylaşım intent'i açılır (kullanıcı cihazındaki WhatsApp, e-posta vb. ile gönderebilir). Link oluşturulursa birkaç saniye içinde bir link üretilip kopyalanır ve kullanıcıya "Link kopyalandı, paylaşabilirsiniz" mesajı gösterilir. Eğer şifre koruma istenirse link oluştururken bir pin belirleme seçeneği verilebilir, ama MVP için gereksiz karmaşıklık olabilir.
3. **Invite vs Share Ayrımı:** Zaten uygulama içi davet yukarıda yapıldı. Bu paylaşım özelliği ondan ayrı tutulacak. Örneğin bir sayfa içinde **"Sayfayı Dışa Aktar"** veya **"Web'de Paylaş"** şeklinde bir menü, sadece kendi içeriklerimiz için aktif olur. Day One uygulaması, tekil girdi paylaşımını doğrudan dışa açık yapmıyor (sadece davetlilerle sınırlı tutuyor) [70], fakat Journey dışa link veriyor. Stratejimizi kullanım senaryosuna göre belirlemeliyiz; belki MVP'de paylaşım linki yerine sadece PDF export yeterli olabilir.

Bu özellikle, kullanıcılar uygulamayı daha **esnek biçimde kullanabilecek**. Örneğin bir seyahat günlüğündeki sayfayı PDF yapıp arkadaşına e-posta atabilecek, ya da bebeklerinin ilk adım attığı anın olduğu sayfayı aile WhatsApp grubunda paylaşabilecek. Bu tür paylaşımlar, uygulamanın reklamını da yapacağından, büyüme açısından olumlu etki sağlayabilir.

8. Sticker ve Kişiselleştirilmiş Süslemeler

Neden gerekli: Dijital bir günlük uygulamasını eğlenceli ve kişisel kılan unsurlardan biri de **çıkartmalar, süslemeler, özel şekiller** kullanabilmektir. Mevcut uygulamada hali hazırda bazı **resim çerçeveleri** özelliği eklenmiş (polaroid, bant, gölge gibi 13 farklı çerçeve stili var [75]), bu iyi bir başlangıç. Bunu bir adım ileri taşıyarak kullanıcının kendi sticker'larını oluşturup kullanması, sayfaları daha da **özgün** hale getirmesini sağlar. Örneğin kullanıcı sevdiği bir emojiyi sticker'a dönüştürüp sayfaya yapıştırabilir, veya kendi çizdiği bir

şekli sticker olarak tekrar tekrar kullanabilir. Bu özellik, özellikle mobil kullanıcı kitlesinin hoşuna gidecek, uygulamayı sadece bir not defteri değil adeta bir dijital **scrapbook** gibi hissettirecektir.

Neler içermeli:
- **Sticker Oluşturma:** Kullanıcı üç yoldan sticker oluşturabilir: (1) Kendi galerinden bir resim kesip sticker yapmak, (2) Uygulamadaki emoji kütüphanesinden seçim yapmak, (3) Kendi çizimini sticker'a dönüştürmek [76] [77]. Oluşturulan sticker'lar bir isim ve kategori ile kaydedilir. Örneğin "Kalp emoji" ismiyle bir sticker oluşturup *Emoji* kategorisine atabilir.
- **Sticker Yönetimi:** Kullanıcı, oluşturduğu tüm sticker'ları görebileceği bir "Stickerlerim" ekranına sahip olmalı [78]. Buradan sticker silebilir veya kategorilerini düzenleyebilir. Çok sayıda sticker olursa kategori filtresiyle aradığı tipte sticker'ı bulabilmeli.
- **Sticker Kullanımı:** Editör içinde, resim ekleme benzeri bir süreçle sticker eklemek mümkün olmalı. Hali hazırda belki bazı hazır sticker veya emoji desteği düşünülmüş olabilir; biz kullanıcı sticker'larını da bu listeye eklemeliyiz [79]. Sticker eklerken kategorilere göre tarama yapıp seçmek kullanışlı olacaktır. Eklendikten sonra sticker da diğer resimler gibi hareket ettirilebilir/yeniden boyutlandırılabilir bloklar olarak sayfaya gelecek.
- **Depolama:** Sticker'lar da sonuçta küçük resim dosyalarıdır. Bunları kullanıcıya özel Firebase Storage alanında tutacağız (ör. `users/{uid}/stickers/...`), ayrıca lokal veritabanında da bir sticker tablosu ile bilgiler (isim, tip, URL vs) saklanacak [80] [81].

Uygulama Adımları:
1. **Model ve Veritabanı:** `UserSticker` adında bir model tanımlanacak. İçinde bir ID, kullanıcının ID'si, ad, kategori, tür (resim/emoji/çizim), sticker görüntüsünün URL'i, lokal cache yolu (opsiyonel), boyut bilgileri gibi alanlar olacak [82] [83]. SQLite'da **user_stickers** tablosu oluşturulacak, Firestore'da da kullanıcının alt koleksiyonu olarak tutulabilir.
2. **Sticker Servisi:** `StickerService` oluşturulacak. Bu servis, yeni bir sticker yaratma (verilen görüntü dosyasını alıp storage'a yükleyip kaydetme), sticker'ları stream ile okuma (kullanıcının tüm sticker'larını veya kategoriye göre sticker'larını getirme), silme/güncelleme fonksiyonlarını içerir [84] [85]. Mümkün olduğunca mevcut `StorageService` yeniden kullanılır, belki sadece yeni metodlar eklenir.
3. **UI Akışı:**
- **Sticker Oluşturucu:** Kullanıcı, sticker eklemek istediğinde bir "Sticker Oluştur" seçeneğine yönlendirilebilir. Bu ekran sekmeler halinde resim, emoji, çizim seçeneklerini sunacak [77]. Resim seçilirse cihaz galerisini açar, kullanıcı bir foto seçer ve belki kırpma aracı ile istediği bölümü sticker yapar. Emoji seçilirse bir emoji picker açıp seçtirir. Çizim seçilirse mini bir çizim tahtası açıp çizer. Sonraki adımda isim ve kategori sorulur ve kaydet dediğinde `StickerService.createSticker` çağrılır [76] [79].
- **Sticker Picker:** Editörde sticker ekleme butonuna tıklandığında, var olan sticker'ları gösteren bir pencere açılır. Burada kategori filtreleme olabilir (üstte kategori sekmeleri veya dropdown), altta o kategoriye ait stickerların küçük önizlemeleri listelenir [79]. Kullanıcı birine tıklayınca sticker sayfaya eklenir. Eğer hiç sticker yoksa bu ekranda "Sticker oluştur" kısayolu da gösterilebilir.
- **Sticker Yönetimi Ekranı:** Profil veya Ayarlar menüsü altına "Stickerlarım" diye bir seçenek konulabilir. Bu ekranda kullanıcının tüm sticker'ları listelenir. Her biri için sil veya düzenle (isim/category değiştir) opsiyonu sunulur [78]. Çok fazla sticker varsa arama veya kategoriyle filtreleme de burada işe yarar.
4. **Teknik Detaylar:** Sticker'ı storage'a yüklerken uygun boyutta tutmak önemli (çok büyük resimleri belki sıkıştırabiliriz). Ayrıca sticker ekleme işlemleri sırasında uygulama performansı düşmemeli; bu nedenle resim işleme işleri mümkünse arka planda yapılmalı.

Sticker özelliği sayesinde, uygulamanın **eğlence ve yaratıcılık katsayısı artacak**. Kullanıcılar kendilerine özgü çıkartmalarla günlük sayfalarını süslerken uygulamaya daha fazla bağlanacaklar. Özellikle genç kullanıcı kitlesi için bu tip özelleştirmeler önemli bir çekim noktasıdır. Bu özellik düzgün planlanıp eklendiğinde, uygulama sıradan bir dijital not defterinden çok, kullanıcıların **kişisel anı albümü** hissi verecek bir platform haline gelecektir.

9. Journal Kapak Özelleştirmeleri ve Önizlemeleri

Neden gerekli: Birden fazla günlük defteri oluşturmaya izin veren uygulamalarda, her defterin ayrı bir **kapak tasarımı veya görseli** kullanılması yaygındır. Bu, kullanıcıya her defteri görsel olarak ayırt etme ve kişiselleştirme imkanı tanır. Mevcut uygulamada journal kapakları için bazı stil seçenekleri olduğu belirtilmiş, örneğin polaroid, bantlı stil vs. Ancak kapak önizleme boyutu küçük veya özelleştirme sınırlı olabilir. Yayınlanacak uygulamada **daha fazla kapak stili**, belki kullanıcı tarafından özel kapak resmi yükleme, ve kapak seçim ekranında **canlı önizleme** gibi UX iyileştirmeleri yapılmalı [86] [87].

Neler içermeli:
- **Daha Fazla Tasarım:** Halihazırda 13 stil varsa bunu daha da zenginleştirebiliriz. Örneğin farklı renkler, dokular, temalar ekleyerek (gezi teması, okul teması vb. kapaklar). Hatta animasyonlu kapaklar bile düşünülebilir ama öncelik statik görseller.
- **Özel Kapak Yükleme:** Kullanıcı isterse galeriden kendi fotoğrafını kapak resmi yapabilmeli. Bu çok talep edilen bir özellik olabilir (örneğin aile albümü defterine ailenin bir grup fotoğrafını kapak yapmak gibi).
- **Kapak Seçiminde Önizleme:** Kullanıcı kapak stilini değiştirirken, sonucu anında büyük bir önizleme alanında görebilmeli [88]. Örneğin bir dialog açıldığında sol tarafta mevcut seçili kapak büyükçe gösterilir, sağ tarafta farklı kapak seçenekleri küçük grid halinde listelenir; birine dokununca sol önizleme anında değişir.
- **Etkileşimli Kapaklar:** Kütüphaneden seçilen statik resimler dışında, belki gradient renk seçimi veya desen oluşturma gibi interaktif kapak oluşturma araçları sunulabilir (ilk sürüm için şart değil, ancak renk seçiciler eklenebilir [87]).

Uygulama Adımları:
1. **UI İyileştirme:** Journal kartlarının listelendiği ekranda (muhtemelen anasayfa veya "Journallarım" listesi), her bir journal'ın kapağını daha büyük ve detaylı göstereceğiz [89]. Şu an küçük ikon gibi görünüyorsa bunu karta yaymak, üzerine journal adı ve belki son eklenen tarihini yazmak iyi olur. Ayrıca kapaklara tıklayınca hafif ölçeklendirme/ışıklanma efektleriyle seçili olduğu belirtilebilir [86].
2. **Kapak Düzenleme Akışı:** Bir journal'ın ayarlarından "Kapağı Düzenle" seçeneği konulacak. Bu açıldığında kullanıcıya üç seçenek sunulur: Mevcut temalardan seç, renk/gradyan oluştur, veya fotoğraf yükle. Mevcut temaları seçince alt alta kategoriler/temalar listelenir (örn. "Klasik", "Modern", "Eğlenceli" gibi) ve her birinde birkaç stil olabilir [90]. Renk/gradyan seçmek isterse bir renk paleti ve desen seçici sunarız (opsiyonel). Fotoğraf yüklemek isterse cihazdan seçip kırparak kullanırız.
3. **Canlı Önizleme:** Kapak düzenleme dialogu, yukarıda bahsedilen şekilde, gerçek zamanlı önizleme yapacak. Flutter'da seçilen stilin Journal kartına uygulanması kolay; `JournalTheme.getCoverDecoration(style)` gibi bir fonksiyon mevcut olabilir [91]. Bu fonksiyon yeni stillerle güncellenecek [92]. Özel resim seçildiyse, imageProvider o an önizlemede kullanılacak. Kullanıcı onaylarsa kapak stil bilgisi ilgili journal objesine kaydedilecek.
4. **Veri Yapısı:** Journal modelinde `coverStyle` veya `coverImage` gibi alanlar zaten olabilir. Mevcut kodda `coverStyle` varsayılan "default" olarak tutuluyor [93]. Bunu genişleterek eğer kullanıcı özel resim yüklediyse `coverStyle = 'image'` ve ayrı bir `coverImageUrl` alanında Firebase'deki URL'sini

tutabiliriz [94].
5. **Ekstra:** Eğer kapaklara kategori eklersek (mesela çizgisiz defter, çizgili defter gibi?), belki ileri sürümlerde. Şu an için öncelik görsel çeşitlilik.

Bu geliştirmeyle, kullanıcılar **her bir günlük defterini kişiselleştirebilecek**. Waffle uygulaması örneğin estetik journal kapaklarını öne çıkan bir özellik olarak sunuyor [95] ve kullanıcıların ruh halini yansıtan kapak seçmelerine olanak tanıyor. Bizim uygulamada da kapak özelleştirme, kullanıcı bağlılığını arttıracak hoş bir dokunuş olacak. Özellikle özel fotoğraf koyabilmek, uygulamayı bir anlamda dijital albüm haline getiriyor.

10. Navigasyon ve Menü (Drawer) İyileştirmeleri

Neden gerekli: Uygulamanın genel gezinti yapısı kullanıcıya açık ve kullanışlı olmalıdır. Mevcut prototipte bir yan menü (Drawer) olduğu belirtilmiş, ancak bu menünün kapsamı ve tasarımı yayınlanacak bir ürün için geliştirilebilir. Özellikle profil bilgilerini menüye koymak, bildirimler için görsel uyarı (badge) göstermek, yardımcı bağlantıları eklemek gibi UX detayları önemlidir. Yeni özellikler eklendikçe, menüde bunların erişimini de düşünmek gerekir (ör. "Ekipler", "Stickerlarım" gibi bölümler eklenebilir). Bu yüzden **drawer tasarımını güncellemek ve zenginleştirmek** gerekecek [96] [65].

Neler içermeli:
- **Menü Başlığı (Header):** Drawer'ın üst kısmında kullanıcı profil fotoğrafı, adı ve e-postası görüntülenmeli [97]. Bu, uygulamaya kişisel bir dokunuş katar. Ayrıca header içine hızlı aksiyon butonları konabilir – örneğin **"Yeni Journal Oluştur"** ve **"Ara"** ikonları header'da yer alabilir [30].
- **Menü Öğeleri:** Anasayfa, Journallar, Arkadaşlar/Ekipler, Bildirimler, Profil&Ayarlar gibi ana bölümler listelenmeli [65]. Her birinin yanında anlamlı ikonlar olmalı. Aktif olunan sayfa menüde işaretlenmeli (örneğin farklı renk veya tik işareti) [98]. Ayrıca alt tarafa "Yardım/SSS" ve "Hakkında" gibi ikincil öğeler eklenebilir.
- **Bildirim Rozetleri:** Sosyal bir uygulamada, yeni bir olay olduğunda (örn. yeni davet, yeni arkadaş isteği, vs.) kullanıcının bunu görmesi için ilgili menü öğesinde bir **kırmızı badge** ile sayı göstermek faydalıdır. Tasarımda Bildirimler ve Arkadaşlar menü öğelerinin yanında örnek sayılar (3, 5 gibi) konulmuş [66]. Bu altyapı kurulmalı: Uygulama, açıkken veya açıldığında, o anki bekleyen bildirim/davet sayısını hesaplayıp menüde gösterecek.
- **Animasyon ve Güzelleştirme:** Drawer açılışına hafif bir kayma animasyonu, menü öğelerine tıklandığında ripple efekti gibi görsel iyileştirmeler profesyonel bir his verir [99] [100]. Flutter'ın animasyon yetenekleriyle (SlideTransition, ScaleTransition vs.) bunlar eklenebilir.

Uygulama Adımları:
1. **Drawer Tasarımını Güncelle:** `app_drawer.dart` dosyasını düzenleyerek yeni tasarımı uygula. Üst kısma bir `DrawerHeader` ekleyip içine kullanıcı bilgisini yerleştir [97]. `DrawerHeader` içinde Column -> CircleAvatar (profil foto), Text (isim), Text (email) şeklinde yapı kurulacak. Yanına/altına da ikon butonlar (yeni journal, arama) eklenecek [30].
2. **Menü Listesi:** Drawer'ın gövdesinde bir ListView ile menü öğeleri sıralanır [65]. Mevcut menü öğelerini yeni ihtiyaçlara göre düzenle: "Journallar" ve "Ekipler" ayrı ayrı listelenebilir veya "Journallar" altında alt menü olarak ekip günlükleri vs. gösterilebilir. Basit olması için belki "Ekipler" ana menüde yer alır. "Bildirimler" menüde olmalı ki kullanıcı gelen davet/istekleri oradan takip edebilsin.
3. **Badge Uygulaması:** Menü item widget'ına badge eklemek için, leading icon'u bir Stack içinde sarmalayarak, köşeye Positioned küçük bir daire ve içinde sayı metni konabilir [98]. Bu badge değerlerini hesaplamak için `NotificationService` benzeri bir yapı kurulur; örneğin bekleyen davetlerin sayısını

`InviteService.watchIncomingInvites()` stream'inden alıp .length ile bulabiliriz. Arkadaş istekleri için de `friendsRequests.length` vs. Bu değerleri menüye sağlayan bir provider yazılabilir [101] [102].
4. **Gezinti ve Durum:** Menünün her tıklanan öğesi ilgili sayfaya götürmeli. Bunu halihazırda GoRouter veya benzeri ile yapıyoruz. Eklenen yeni sayfalar (Ekip listesi, Bildirimler, Stickerlar vb.) için de rotalar tanımlanıp menüye bağlanacak. Aktif sayfanın menüde gösterimi için ya route bazlı bir check koyabilir veya seçili item'ı highlight eden bir state tutabiliriz. Kod örneğinde aktif olana trailing bir check ikonu konmuş [103] – bunu uygulayabiliriz.
5. **Son Rötuşlar:** Drawer'ın açılma animasyonu ve menü item'ların hover/tıklama animasyonları eklenecek (SlideTransition, ScaleTransition) [99]. Ayrıca çıkış yap butonu alta konulacak (kırmızı renkle) ve versiyon bilgisi de altına küçük yazıyla eklenebilir [104].

Bu iyileştirilmiş navigasyon menüsü sayesinde uygulama **daha profesyonel ve kullanıcı dostu** bir görünüme kavuşacak. Kullanıcılar kendi adlarını ve avatarlarını görerek kişisel bir bağ kuracak, bildirim rozetleri sayesinde uygulama içi etkinliklerden haberdar olacaklar. Özellikle sosyal uygulamalarda bu tip göstergeler kullanıcıyı uygulamayı açmaya teşvik eder (örneğin uygulama ikonu üzerinde de push bildirimi ile sayı gösterilebilir, ileride düşünebiliriz). Sonuç olarak, menü revizyonu genel UX'i güçlendiren bir dokunuş olacak.

11. Bildirimler ve Kullanıcı Etkileşimleri

Neden gerekli: Uygulama sosyal bir yapıya büründükçe, **bildirim** sistemi önemli hale gelir. Sadece uygulama içindeki "Bildirimler" sayfasında davetleri göstermek yeterli olmayabilir; kritik olaylarda kullanıcının cihazına anlık bildirim göndermek uygulamayı canlı tutar. Örneğin biri sizi bir journal'a davet ettiğinde, telefonunuza bir push bildirimi düşmesi veya ortak günlükte yeni bir içerik eklendiğinde grup üyelerinin haberdar edilmesi istenebilir. Ayrıca uygulama içindeki küçük etkileşimler (bir sayfayı beğenme, yorum yapma gibi) ileride eklenirse, bunların da bildirim mekanizmasıyla desteklenmesi gerekir.

Neler içermeli:
- **Push Notification Entegrasyonu:** Firebase Cloud Messaging (FCM) kullanarak temel push bildirim altyapısı kurulmalı. Kullanıcılar uygulamaya ilk giriş yaptığında izin istenip FCM token alınır. Önemli olaylar olduğunda (davet, yeni giriş ekleme, yorum vs.) ilgili kullanıcılara sunucudan push atılır.
- **Uygulama İçi Bildirim Merkezi:** Zaten menüde "Bildirimler" ekranı planlandı. Bu ekran hem davetleri hem de gelecekte diğer bildirim türlerini listeleyecek şekilde genelleştirilebilir. Örneğin "X kullanıcısı ortak günlüğe yeni bir fotoğraf ekledi" gibi bir satır da buraya düşebilir. İlk sürümde sadece davetleri ele alsa da yapı genişlemeye açık olmalı.
- **Etkinlik Akışı (Opsiyonel):** Kullanıcıların arkadaş listesinde veya ekiplerinde olanların önemli aktivitelerini görebileceği bir "feed" de düşünülebilir. Bu, anı defteri uygulaması için şart değil ama etkileşimi arttırabilir. (Örn. "Arkadaşın Y yeni bir journal başlattı" gibi). Bu çok gerekli görülmezse es geçilebilir.

Uygulama Adımları:
1. **FCM Kurulumu:** Firebase projesinde Cloud Messaging etkinleştirilecek. Flutter app tarafında `firebase_messaging` paketi entegre edilip, uygulama başlarken izin istenir ve token alınır. Bu token Firestore'daki user profile'ına kaydedilebilir ki sunucusuz mimaride bile belki diğer kullanıcı cihazlarına P2P bildirim göndermenin yolu olsun (gerçi ideal olan bir Cloud Function ile mesaj göndermek). İlk planda belki tüm push bildirimlerini sunucu olmadan çözmek zor; alternatif olarak davet gönderirken *invitee* kullanıcının FCM token'ını Firestore'dan okuyup, gönderen cihazdan direkt bir bildirim gönderimi denenebilir (bu güvenlik ve teslimat açısından sınırlı bir yöntem, bu nedenle belki Firestore Trigger + FCM Cloud Function

gerekir).
2. **Bildirim İçerikleri:** Hangi durumlarda bildirim gideceğini tanımlayalım: Yeni davet alındığında (hedef kullanıcıya "X sizi Y adlı journal'a davet etti"), davet kabul edildiğinde (davet gönderen kullanıcıya "Y davetinizi kabul etti"), ortak journal'a içerik eklendiğinde (tüm diğer üyelere "X yeni bir giriş ekledi: [giriş başlığı]"), size arkadaşlık isteği geldiğinde vb. Bu metinler uluslararasılaştırma düşünülerek hazırlanmalı.
3. **Uygulama İçi Gösterim:** Kullanıcı uygulamayı açtığında da bu olayları kaçırmamalı. Bu yüzden "Bildirimler" ekranı bu olayları listelemeli. Zaten davetleri gösteriyor olacağız; aynı tablo/koleksiyon yapısını genişleterek genel bildirimleri orada tutabiliriz (veya ayrı bir notifications koleksiyonu da olabilir). Badge sistemi ile menüde sayısını göstereceğiz. Bildirimler ekranında da okunmamışları vurgulayabilir, "tümünü okundu yap" seçeneği ekleyebiliriz.
4. **Test ve İzinler:** Push bildirim için cihazlarda izin yönetimi önemli. iOS'ta kullanıcı izin vermezse rahatsız etmemek lazım. Android'de otomatik izin var ama "Doze Mode" vs. engeller olabiliyor. Bu altyapıyı iyi test edip, mümkünse Firebase Analytics ile bildirim tıklanma/teslim istatistiklerini de toplayabiliriz.

Bu sayede uygulamanın kullanıcıları **her zaman bağlantıda kalır**. Ortak tutulan bir günlükte birinin eklediği anıyı diğerleri anında görürse etkileşim artar, kullanıcılar uygulamayı düzenli kullanmaya motive olur. Örneğin Waffle uygulaması, eşler veya arkadaşlar arasında her gün bildirimlerle birbirlerine yazmalarını teşvik ediyor ve kullanıcılar "Bildirim geldiğinde yüzüm gülüyor" şeklinde geri bildirim veriyorlar [105]. Bizim uygulamada da doğru yerde kullanılacak bildirimler, kullanıcı bağlılığını ve memnuniyetini artıracaktır.

12. Genel UX İyileştirmeleri ve Son Rötuşlar

Yukarıdaki büyük özelliklerin yanı sıra, uygulama yayınlanmadan önce kapsamlı bir **kullanıcı deneyimi (UX) değerlendirmesi** yapılarak ufak tefek iyileştirmeler de gerçekleştirilmelidir. Bu kapsamda:

- **Onboarding (İlk Kullanım) Akışı:** Yeni kullanıcıların uygulamayı ilk açtıklarında ne yapacaklarını anlamaları için kısa bir tanıtım veya ipucu turu eklenebilir. Örneğin 3 adımlı bir slayt gösterisiyle "Fotoğraf ve yazılarla anılarınızı saklayın", "Sevdiklerinizi davet edip birlikte günlük tutun" gibi mesajlar verilebilir. Hemen ardından ilk journal oluşturma veya ilk sayfayı ekleme yönlendirmesi yapılabilir. Bu, kullanıcıların uygulamayı boş bulup kapatmasını engeller.
- **Performans ve Hız:** Uygulamanın sayfa geçişleri, liste scroll performansı, editör içi hareketler olabildiğince akıcı olmalı. Özellikle medya ağırlıklı bir uygulama olduğundan, resimlerin uygun boyutlarda yüklendiği ve cache mekanizmalarının kullanıldığı kontrol edilmeli. Gereksiz yeniden çizim (rebuild) yapılan widgetlar optimize edilmeli. Örneğin liste ve gridlerde `const` anahtar kelimesi kullanımı, `ListView.builder` ile tembel yükleme [106], görüntülerin önbelleğe alınması gibi teknikler uygulanacak.
- **Hata Durumları ve Geri Bildirim:** Kullanıcı bir işlem yaptığında (mesela davet gönder, fotoğraf yükle vs.), arka planda işlem oluyorsa bir yükleniyor göstergesi verilmeli, başarılı tamamlanınca onay mesajı, hata olursa anlaşılır bir uyarı gösterilmeli. Uygulama hiçbir durumda sessizce başarısız olmamalı. Özellikle ağ/senkronizasyon hatalarında kullanıcıya bağlantı sorunu olduğu veya işlemin tekrar denenebileceği bilgisi verilmeli.
- **Erişilebilirlik:** Uygulamayı farklı yeteneklere sahip kişiler de kullanabileceği için erişilebilirlik gözden geçirilmeli. Yazı boyutlarının cihazın erişilebilirlik ayarlarına göre ölçeklenebilmesi, renk kontrastının yeterli olması, ekran okuyucu (TalkBack/VoiceOver) için anlamlı label'ların tüm butonlarda bulunması gerekiyor [107] [108]. Örneğin görsel içerikli bir butonda sadece simge varsa, o simge için `semanticLabel` atamak gerekli.

- **Analytics ve Crash Reporting (Opsiyonel):** Uygulamayı yayınladıktan sonra kullanıcı davranışlarını ve olası hataları izleyebilmek için **Firebase Analytics** ve **Crashlytics** entegre etmek faydalı olacaktır [109] [110]. Bu sayede hangi özellikler daha çok kullanılıyor (örneğin sticker oluşturma popüler mi değil mi), nerede takılınıyor veya uygulama hangi cihazlarda çöküyor gibi bilgiler toplanıp gelecekteki güncellemelerde rehber alınabilir. Bu servisler opsiyonel ancak profesyonel bir uygulama için değerlidir.

Tüm bu ince ayarlar, uygulamanın genel kalitesini yükseltecektir. Özellikle ilk izlenim (onboarding), hız ve pürüzsüzlük konuları, kullanıcı tutma oranını etkiler. Son bir genel test/denetim aşamasında bu konular bir kontrol listesi ile değerlendirilmelidir [111] [112]. Gerekirse kapalı beta testlerle bir grup kullanıcıdan geri bildirim alınabilir. Elde edilen bulgular ışığında ufak ayarlar yaptıktan sonra artık uygulamayı güvenle yayınlamak mümkün olacaktır.

Uygulama Planı ve Fazlandırma

Yukarıda belirtilen eksik özelliklerin uygulanması için bir **aşamalı (faz) plan** öngörüyoruz. Bu sayede geliştirme süreci önceliklendirilmiş olacak ve adım adım ilerlenebilecektir. Önerilen plan ve sıralama şöyledir:

- **Faz 1: Kritik Temel İyileştirmeler (1-2 hafta)**
  İlk etapta, mevcut uygulamanın zayıf kaldığı temel alanlar güçlendirilecek. Bu kapsamda Profil ve Ayarlar ekranının birleştirilmesi, tema ve renk özelleştirme desteğinin eklenmesi önceliklidir. Bu sayede uygulama tekil kullanıcı için bile tutarlı ve kişiselleştirilebilir hale gelecek. Ayrıca bu aşamada medya desteğinin (özellikle ses kaydı ve genel editor kararlılığının) gözden geçirilip küçük düzeltmeleri yapılabilir. Faz 1 sonunda uygulama tek kullanıcı senaryosunda tatmin edici bir günlük deneyimi sunuyor olmalıdır. (Not: Bu aşamadaki bazı görevler yukarıda ile işaretlenmiş olarak halihazırda gerçekleştirilmiştir [113].)

- **Faz 2: İşbirliği ve Paylaşım Özellikleri (2-3 hafta)**
  İkinci aşamada uygulamanın çok kullanıcılı özellikleri hayata geçirilecek. Ekip (ortak journal) sistemi tam olarak implementasyon edilecek [114], kullanıcı davet etme akışı eklenecek ve temel paylaşım fonksiyonları (davet linki, belki sayfa dışa aktarma) kurulacak. Bu, teknik olarak en zahmetli kısım olabilir zira veritabanı şemasından senkronizasyona birçok katmanda değişiklik gerektiriyor [35] [59]. Yaklaşık 2-3 haftalık bir geliştirme sürecinde, öncelikle *Team* + *Invite* sistemleri, ardından temel *Share* özelliği entegre edilir. Bu faz tamamlandığında uygulama kullanıcıları, birbirlerini ekleyip ortak anılar tutabilecek ve ilk defa sosyal etkileşim kısmı devreye girmiş olacak.

- **Faz 3: Zengin İçerik ve UX İyileştirmeleri (1-2 hafta)**
  Üçüncü aşamada odak, uygulamayı eğlenceli ve pürüzsüz hale getirecek ek özelliklerdir. Sticker oluşturma ve yönetme özelliği bu fazın en büyük parçasıdır [115]. Ardından journal kapak özelleştirme ve drawer/menü iyileştirmeleri tamamlanır. Bu noktada uygulamanın görsel bütünlüğü ve kullanışlılığı artacak; kullanıcı arayüzü son halini alacaktır. Ayrıca bildirim rozetleri, küçük animasyonlar gibi detaylar da bu aşamada eklenir. Faz 3 sonunda, listelenen tüm büyük özellikler implementasyonu tamamlanmış olacaktır [116].

- **Faz 4: Son Kontroller ve Yayına Hazırlık (1 hafta)**
  Dördüncü ve son aşamada, genel bir UX kontrolü, performans optimizasyonu ve hata ayıklama

süreci yapılır [112] [117]. Erişilebilirlik incelemesi gerçekleştirilir; gerekli düzeltmeler yapılır. Onboarding ekranı eğer eklenmediyse burada eklenebilir. Firebase Analytics/Crashlytics gibi araçlar bağlanıp son testler gerçekleştirilir. Uygulamanın hem çevrimdışı hem çevrimiçi senaryolarda tutarlı çalıştığı, farklı cihaz ekran boyutlarında arayüzün bozulmadığı doğrulanır. Bu aşamanın sonunda uygulama mağaza yönergelerine uygun hale getirilmiş (ikonlar, ekran görüntüleri, gizlilik politikası metni vs. hazırlanmış) ve yayınlanmaya hazır durumdadır.

Her faz sonunda ekip içi bir değerlendirme ve test yapılarak, bir sonraki faza geçmeden hatalar düzeltilmelidir. Bu plan sayesinde, kademeli olarak özellikler eklenecek ve her adımda uygulama gittikçe daha rekabetçi bir ürün haline gelecektir. Sonuç olarak, tüm kritik özellikleri tamamlanmış, kullanıcı deneyimi zengin, **mobil platformlara uygun bir dijital anı defteri** uygulamasını başarıyla hayata geçirebiliriz.

Kaynaklar:

- Uygulamanın mevcut kod tabanından çıkarılan gereksinimler [1] [2] [32] ve planlamalar.
- Journey ve Day One gibi popüler günlük uygulamalarının özellik listeleri (çoklu ortam desteği, paylaşımlı günlükler, güvenlik önlemleri vb.) [21] [39].
- Waffle ortak günlük uygulamasının kullanıcı etkileşimini artıran estetik kapaklar ve günlük bildirimleri gibi özellikleri [95] [105].
- Erişilebilirlik ve performans için Flutter geliştirme tavsiyeleri [107] [117].

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [22] [26] [27] [30] [31] [32] [33] [35] [36] [37] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [57] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69] [75] [76] [77] [78] [79] [80] [81] [82] [83] [84] [85] [86] [87] [88] [89] [90] [91] [92] [93] [94] [96] [97] [98] [99] [100] [101] [102] [103] [104] [106] [107] [108] [109] [110] [111] [112] [113] [114] [115] [116] [117] new.md
https://github.com/BGirginn/journal/blob/ad0ed53a434541431ab67bd1d6eda4bb4d7a3e74/new.md

[21] [24] [25] [29] [71] [72] [74] Free Online Journal & Diary App | Journey.Cloud
https://journey.cloud/

[23] [28] CLAUDE.md
https://github.com/BGirginn/journal/blob/ad0ed53a434541431ab67bd1d6eda4bb4d7a3e74/CLAUDE.md

[34] [38] [39] [55] [56] [70] Shared Journals
https://dayoneapp.com/shared-journals/

[73] [105] Waffle: Shared Journal App - App Store
https://apps.apple.com/us/app/waffle-shared-journal/id1519726353

[95] Waffle: Shared Journal - Apps on Google Play
https://play.google.com/store/apps/details?id=ai.poppy.waffle&hl=en_US